

Hacia un Marco Práctico de Evaluación de Tecnologías de Transformación de Modelos en la Plataforma Eclipse

Emilio A. Sánchez

Integranova S.A.
Ciudad Politécnica de la Innovación.
Univ. Politécnica Valencia. Ed. 9B
46022 Valencia
esanchez@integranova.com

Gabriel Merin

Integranova S.A.
Ciudad Politécnica de la Innovación.
Univ. Politécnica Valencia. Ed. 9B
46022 Valencia
gmerin@integranova.com

Javier Muñoz

Integranova S.A.
Ciudad Politécnica de la Innovación.
Univ. Politécnica Valencia. Ed. 9B
46022 Valencia
jmunoz@integranova.com

Resumen

El amplio ecosistema de proyectos creado alrededor de la plataforma Eclipse está propiciando que se convierta en una tecnología ampliamente usada para implementar las ideas propuestas en el enfoque del Desarrollo de Software Dirigido por Modelos (DSDM). Son varias las tecnologías disponibles en Eclipse para realizar transformaciones entre modelos. En este trabajo se comparan tres de estas tecnologías (ATL, SmartQVT y la implementación de QVT Operational de Borland®) atendiendo a (1) el tratamiento de los modelos de entrada y salida y (2) el tratamiento de las relaciones entre modelos. Con ello no sólo se evalúan las tecnologías consideradas, sino que se establece un marco de evaluación aplicable a otras tecnologías de transformación de modelos.

1. Introducción

La plataforma Eclipse proporciona entorno abierto y multiplataforma sobre el que construir herramientas para el desarrollo de software y, en general, aplicaciones de todo tipo. Su naturaleza libre ha propiciado la creación de un rico ecosistema de proyectos (muchos de ellos distribuidos utilizando licencias libres) que abarcan múltiples dominios tecnológicos entre los que se encuentran el Desarrollo de Software Dirigido por Modelos (DSDM). Tanto la industria como la academia encuentran en esta plataforma una base sobre la que construir sus productos y/o poner en práctica sus ideas innovadoras, a la vez que les garantiza cierta interoperabilidad con otras herramientas debido al uso de tecnologías comunes.

La creación de una amplia oferta de tecnologías de libre disposición resulta interesante desde el punto de vista del desarrollador que se enfrenta a un nuevo proyecto, ya que aumenta la posibilidad de encontrar una tecnología que se ajuste a sus necesidades. Por otra parte, si la oferta es muy amplia puede resultar complicado encontrar aquella más idónea. En este contexto resultan necesarios marcos de evaluación que permitan comparar tecnologías que resuelven problemas similares (o incluso el mismo).

Las transformaciones entre modelos son una pieza clave de las herramientas DSDM. En la plataforma Eclipse existen diversas tecnologías que permiten realizar esta tarea [1][2][4]. Cuando una empresa o cualquier otra entidad se enfrenta al reto de desarrollar una herramienta DSDM sobre la plataforma Eclipse surge la necesidad de realizar una selección de la tecnología de transformación de modelos más adecuada a las necesidades del proyecto. Para ello es necesario contar con estudios precisos que caractericen al detalle las distintas tecnologías disponibles teniendo en cuenta especialmente aquellas características que son una fuente potencial de problemas a la hora de abordar proyectos en un contexto de producción.

La contribución de este trabajo es la definición de un marco de comparación que incluye ciertas características para la evaluación de tecnologías de transformación de modelos en la plataforma Eclipse y su aplicación práctica a tres tecnologías en concreto (ATL, SmartQVT y la implementación de QVT Operational de Borland®).

La estructura del artículo es la siguiente: en la sección 2 se presenta el marco de comparación. A continuación, la sección 3 describe brevemente las tecnologías evaluadas. En la sección 4 se muestra la experimentación realizada para alcanzar los

resultados, que se presentan en la sección 5. Finalmente, la sección 6 presenta las conclusiones del trabajo.

2. Marco de comparación

El marco de comparación para tecnologías de transformación de modelos presentado en este trabajo está centrado en dos tipos de características:

- El **tratamiento de los modelos de entrada y salida** (número de modelos permitidos y su comportamiento ante modelos conformes a varios metamodelos)
- El **tratamiento de las relaciones entre modelos** almacenados en distintos ficheros, tantos si estos han sido explícitamente especificados como modelos participantes en la transformación (modelos de entrada o salida) como si son referenciados por los modelos de entrada.

A continuación se describen más detalladamente cada una de estos tipos de características.

2.1. Modelos de entrada y salida

En cuanto al tratamiento de los modelos con los que permite trabajar la tecnología se contemplan dos características a estudiar:

- La cantidad de modelos implicados en la transformación (de entrada o de salida), que puede ser 1, un número definido, o un número indefinido. Ya en [6] se identifica esta propiedad como una de las principales características para clasificar lenguajes y técnicas de transformación de modelos.
- En caso de ser más de 1 metamodelo, también es importante diferenciar si todos son conformes al mismo metamodelo o hay distintos metamodelos implicados. Como veremos, esto puede ser fuente de complicaciones a la hora de realizar la transformación.

En definitiva, combinando las anteriores características, se identifican una serie de escenarios. Para cada uno de estos escenarios se debe estudiar si la tecnología es capaz de tratarlos

adecuadamente o no. A continuación se describen estos escenarios

1. **Un sólo modelo:** Es la transformación más sencilla. La tecnología debe funcionar correctamente con un sólo modelo de entrada o salida.
2. **Un número definido de modelos del mismo metamodelo:** La transformación recibe como modelos de entrada (o genera como modelos de salida) un número conocido de modelos conformes al mismo metamodelo. Por defecto modelos conformes al mismo metamodelo son tratados como un único modelo y no se diferencian sus elementos (Figura 1); es decir, la reglas se aplican del mismo modo a los elementos de ambos modelo.

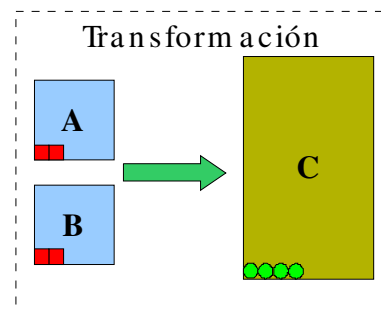


Figura 1. Modelos conformes al mismo metamodelo sufren la misma transformación

Sin embargo pueden aparecer complicaciones si se desea realizar operaciones diferentes sobre elementos de los distintos modelos (Figura 2).

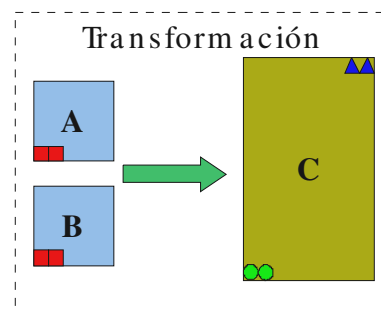


Figura 2. Modelos conformes al mismo metamodelo sufren distinta transformación

3. **Un número definido de modelos de distintos metamodelos:** Los modelos (de entrada o de salida) son conformes cada uno a un metamodelo distinto. Habitualmente este escenario implica menor dificultad ya que al tratarse de diferentes metamodelos la especificación de las reglas identifica unívocamente a que modelo se aplica.

4. **Un número indefinido de modelos del mismo metamodelo:** En este caso no se conoce a priori el número de modelos de entrada, pero sí se sabe que todo serán conformes al mismo metamodelo. Las herramientas no suelen trabajar bien con un número indefinido de modelos debido a restricciones del lenguaje que implementan. Sin embargo es posible que en un proyecto se deba realizar una transformación en la que el número de modelos no es conocido o se sabe que será un número variable dependiendo de una selección del usuario.

5. **Un número indefinido de modelos de distintos metamodelos:** En esta situación se añade la complejidad de no conocer a priori los metamodelos a los que serán conformes los modelos que van a participar en la transformación.

2.2. Relaciones entre modelos

Para estudiar el tratamiento de las relaciones entre modelos utilizaremos los conceptos “modelos principales en la transformación” y “modelo referenciado en la transformación”. A continuación se definen estos conceptos:

- Un **modelo es principal en la transformación** cuando ha sido definido explícitamente como modelo de entrada o salida de la misma.
- Un **modelo es referenciado en la transformación** cuando contiene elementos que son referenciados por algún modelo principal u otro modelo referenciado en la transformación.

Respecto a los enlaces entre modelos es necesario estudiar el comportamiento de las distintas tecnologías cuando éstos se producen entre modelos principales o hacia modelos referenciados.

1. **Enlaces con modelos referenciados en la transformación:** En este caso algún elemento de uno de los modelos implicados en la transformación se encuentra relacionado con un elemento de un modelo referenciado

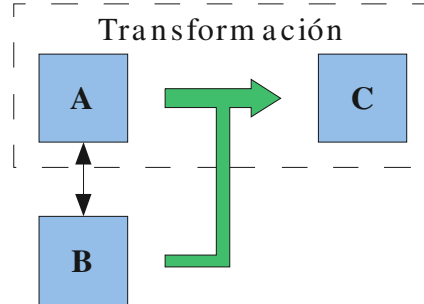


Figura 3. Transformación accediendo a un modelo referenciado para producir la salida

Si se trata de un elemento perteneciente a un modelo de entrada es deseable poder seguir esa relación accediendo a la información (el valor de sus propiedades y sus relaciones) del elemento enlazado (Figura 3). En el caso de modelos de salida, es deseable poder enlazar una entidad que ha sido generada con una entidad perteneciente a un modelo referenciado (Figura 4).

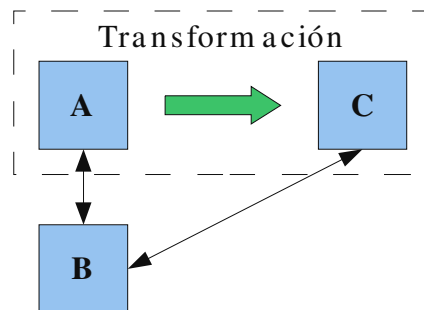


Figura 4. Transformación referenciando en la salida a un modelo referenciado

2. **Enlaces con otros modelos principales de la transformación:** En este escenario los modelos enlazados forman ambos parte de la transformación. El comportamiento deseado es que la herramienta sea capaz de interpretar que

el elemento relacionado se encuentra en uno de los modelos principales de la transformación y no lo trate como un elemento de un modelo referenciado (Figura 5).

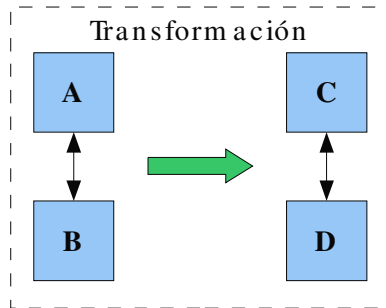


Figura 5. Transformación manteniendo las referencias entre modelos principales

3. Tecnologías evaluadas

Existe una amplia oferta de tecnologías disponibles para realizar transformaciones de modelos en el contexto de la plataforma Eclipse. Todas estas tecnologías tienen en común que manipulan modelos en formato Ecore, la tecnología de metamodelado utilizada en el Eclipse Modeling Framework (EMF)¹. Esta tecnología puede considerarse como una implementación de un subconjunto de MOF, el estándar de OMG para metamodelado.

En este trabajo se han seleccionado tres tecnologías de transformación de modelos sobre las que se ha aplicado el marco de comparación descrito en la Sección 2.

- **ATL:** Atlas Transformation Language (ATL) es la herramienta propuesta por el grupo de investigación ATLAS INRIA & LINA. Es un lenguaje de transformación de modelos especificado tanto como un metamodelo como una sintaxis textual concreta. El lenguaje es híbrido: declarativo e imperativo. Sin embargo el estilo más utilizado es el declarativo pudiendo expresar mappings fácilmente. Actualmente forma parte del

Tabla 1. _____

¹ <http://www.eclipse.org/modeling/emf/>

subproyecto "M2M"² dentro del "Eclipse Modeling Project". En este trabajo se ha utilizado la versión de la *feature* 1.0.9, correspondiente al lanzamiento realizado en febrero de 2007.

- **QVT Operacional de Borland®:** La implementación del estándar QVT [7] realizada por Borland® para su herramienta Together®. No soporta relaciones declarativas, tan solo transformaciones imperativas. Actualmente su liberación ha sido aprobada por Borland® y se encuentra en fase de integración en el subproyecto "M2M" del "Eclipse Modeling Project". En este trabajo se ha utilizado la versión de la *feature* 8.1.1.
- **SmartQVT:** Herramienta desarrollada por France Telecom R&D y financiada parcialmente por el proyecto europeo IST Modelware. Implementa el lenguaje MOF 2.0 QVT-Operational, un lenguaje híbrido con una estructura de reglas declarativas y un flujo de ejecución imperativo. Requiere de un intérprete externo de Python, ya que el analizador de QVT ha sido desarrollado utilizando este lenguaje. En este trabajo se ha utilizado la versión de la *feature* 0.1.4.

4. Evaluación y experimentación

Con el objetivo de aplicar el marco de comparación sobre las tecnologías de transformación seleccionadas se llevó a cabo un proceso de experimentación y evaluación. Para facilitar la implementación de algunas de las pruebas se definió un caso de estudio de referencia compuesto por dos modelos, utilizando Ecore como lenguaje de metamodelado. El caso de estudio se muestra esquemáticamente en la Figura 6. En estos modelos se incluyen elementos que permitieron la evaluación de las características del marco de comparación (referencias a elementos del propio modelo y de un modelo almacenado en otro archivo).

Tabla 1. _____

² <http://www.eclipse.org/m2m/>

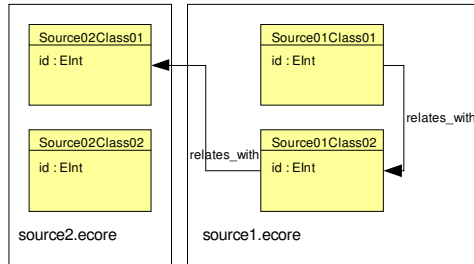


Figura 6. Caso de estudio de referencia

4.1. Evaluación tratamiento modelos de entrada y salida

Para la evaluación del tratamiento de los modelos de entrada y salida, en primer lugar es necesario estudiar las capacidades del lenguaje que implementa la herramienta. Cada lenguaje proporciona una expresividad que impone las primeras limitaciones a los escenarios que son soportados y, por tanto, es necesario conocer la expresividad del lenguaje o la implementación particular que se ha realizado de un lenguaje.

El siguiente paso fue intentar implementar transformaciones haciendo uso de las distintas tecnologías que pusieran en práctica cada uno de los escenarios descritos en la Sección 2.1. Para ello se definieron una serie de casos de prueba a partir de las características o capacidades que debían ser estudiadas y utilizando el caso de estudio de referencia. A continuación se describen los casos de prueba definidos:

- **Caso de Prueba MDL01. (Ejercita escenario 1; 1 de entrada y 1 de salida).** A partir de source2.ecore generar output_md101_sc01.ecore añadiendo la cadena "new_" al inicio del nombre de los elementos EClass.
- **Caso de Prueba MDL02. (Ejercita escenario 1; 1 de entrada y 1 de salida).** A partir de source2.ecore generar output_md102_sc01.uml conforme a la implementación de UML2 del proyecto Eclipse. Se convertirá cada elemento Ecore::EPackage en un UML2::Package y cada Ecore::EClass en UML2::Class copiándose en ambos casos sus atributos "name".

- **Caso de Prueba MDL03. (Ejercita escenario 2; 1 de entrada y 2 de salida).** A partir de source2.ecore generar output_md103_sc02a.ecore añadiendo la cadena "A_" al inicio del nombre de los elementos EClass, y output_md103_sc02b.ecore añadiendo la cadena "B_" al inicio del nombre de los elementos EClass.
- **Caso de Prueba MDL04. (Ejercita escenario 2; 2 de entrada y 1 de salida).** A partir de source1.ecore y source2.ecore generar output_md104_sc02.ecore que contengan los elementos de ambos añadiendo la cadena "A_" al inicio del nombre de los elementos EClass.
- **Caso de Prueba MDL05. (Ejercita escenario 2; 2 de entrada y 1 de salida).** A partir de source1.ecore y source2.ecore generar output_md105_sc02.ecore que contengan los elementos de ambos añadiendo la cadena "1_" al inicio del nombre de los elementos EClass de source1.ecore y la cadena "2_" al inicio del nombre de los elementos EClass de source2.ecore.
- **Caso de Prueba MDL06. (Ejercita escenario 2; 2 de entrada y 2 de salida).** A partir de source1.ecore y source2.ecore generar output_md106_sc02a.ecore que contengan los elementos de ambos añadiendo la cadena "1a_" al inicio del nombre de los elementos EClass de source1.ecore y la cadena "2a_" al inicio del nombre de los elementos EClass de source2.ecore; y output_md106_sc02b.ecore que también contengan los elementos de ambos pero añadiendo la cadena "1b_" al inicio del nombre de los elementos EClass de source1.ecore y la cadena "2b_" al inicio del nombre de los elementos EClass de source2.ecore
- **Caso de Prueba MDL07. (Ejercita escenario 3; 1 de entrada y 2 de salida).** A partir de source2.ecore generar output_md107_sc03.ecore añadiendo la cadena "new_" al inicio del nombre de los elementos EClass; y output_md107_sc03.uml conforme a la implementación de UML2 del proyecto Eclipse, realizando el mismo mapping que en casos anteriores.

- **Caso de Prueba MDL08. (Ejercita escenario 3; 2 de entrada y 1 de salida).** A partir de source2.ecore y un archivo .uml generar output_md108_sc03.ecore que contenga un paquete raíz, los elementos ECore::EClass del archivo .ecore y los elementos UML2::Class que estén contenido en el primer paquete del archivo .uml mapeados como elementos ECore::EClass.
- **Caso de Prueba MDL09. (Ejercita escenario 4; n de entrada y 1 de salida).** A partir de un número indeterminado de archivos .ecore de entrada, generar output_md109_sc04.ecore que contenga los elementos de ambos añadiendo la cadena "new_" al inicio del nombre de los elementos EClass.
- **Caso de Prueba MDL10. (Ejercita escenario 4; 1 de entrada y n de salida).** A partir de source2.ecore generar tantos archivos .ecore como elementos EClass contiene. Cada uno de estos archivos contendrá uno de elementos de los elementos EClass añadiendo la cadena "new_" al inicio de su nombre. La regla de transformación debe ser genérica; es decir, no debe depender de source2.ecore.
- **Caso de Prueba MDL11 (Ejercita escenario 5; n de entrada y 1 de salida).** A partir de un número indeterminado de modelos de entrada conformes a metamodelos sin determinar, generar un archivo output_md110_sc05.ecore que contenga un elemento EClass por cada elemento de los modelos de entrada manteniendo las relaciones entre ellos y copiando, si existe, el nombre.

Como se puede observar, esta batería de casos de prueba cubre todos los escenarios y posibles situaciones conflictivas descritas en la descripción del marco de evaluación.

4.2. Evaluación tratamiento relaciones entre modelos

Para la evaluación del tratamiento de relaciones entre modelos se siguió un enfoque similar a la evaluación del tratamiento de los modelos de entrada y salida. A partir de las características o capacidades que debían ser estudiadas y utilizando el caso de estudio de referencia, se definieron una serie de casos de prueba para los que las distintas

tecnologías debían proporcionar los mecanismos necesarios para su resolución.

A continuación se presentan los casos de prueba definidos para estudiar la funcionalidad en esta área:

- **Caso de Prueba LINK01. (Evalúa la característica 1).** A partir de source1.ecore generar un archivo output_link01.ecore que produzca un elemento EClass en cuyo atributo nombre tendrá la concatenación del nombre del elemento EClass origen y el elemento EClass con el que estaba enlazado a través de la relación "relates_with". Es decir; se deberán generar dos clases de nombre "Source01Class01Source01Class02" y "Source01Class02Source02Class01".
- **Caso de Prueba LINK02. (Evalúa la característica 1).** A partir de source1.ecore generar un archivo output_link02.ecore que copie los elementos EClass añadiendo la cadena "new_" al inicio del nombre. La copia debe incluir las referencias originales "relates_with". Como resultado, la referencia "relates_with" de la EClass "new_Source01Class02" deberá apuntar a Source02Class01" de source2.ecore.
- **Caso de Prueba LINK03. (Evalúa la característica 2).** A partir de source1.ecore y source2.ecore generar un archivo output_link03.ecore que produzca un elemento EClass en cuyo atributo nombre tendrá la concatenación del nombre del elemento EClass origen y el elemento EClass con el que estaba enlazado a través de la relación "relates_with". Es decir; se deberán generar dos clases de nombre "Source01Class01Source01Class02" y "Source01Class02Source02Class01".
- **Caso de Prueba LINK04. (Evalúa la característica 2).** A partir de source1.ecore y source2.ecore generar un archivo output_link04.ecore que copie los elementos EClass añadiendo la cadena "new_" al inicio del nombre. La copia debe incluir las referencias originales "relates_with". Como resultado, la referencia "relates_with" de la EClass "new_Source01Class02" deberá apuntar a Source02Class01" de source2.ecore.
- **Caso de Prueba LINK05. (Evalúa la característica 2).** A partir de source1.ecore y

source2.ecore generar dos archivos output_link05a.ecore y output_link05b.ecore en los que se repliquen los elementos EClass añadiendo la cadena “new_” al inicio del nombre. En este caso las referencias “relates_with” serán a los nuevos elementos EClass. Como resultado, la referencia “relates_with” de la EClass “new_Source01Class02” deberá apuntar a “new_Source02Class01” del archivo output_link05b.ecore.

En este caso la batería de casos de prueba también cubre todos los escenarios y posibles situaciones conflictivas descritas en la descripción el marco de evaluación.

5. Resultados de la evaluación

Como resultado del proceso de evaluación descrito en la Sección 4, se obtuvieron una serie de resultados y conclusiones para cada una de las tecnologías consideradas en el trabajo. A continuación se presentan brevemente.

5.1. ATL

En cuanto al **tratamiento de los modelos de entrada y salida**, ATL se comporta *correctamente trabajando con un sólo modelo de entrada y salida* (escenario 1) o *cuando al manejar diferentes modelos cada uno de ellos son conformes a distintos metamodelos* (escenario 3).

Sin embargo presenta algunas *dificultades cuando diferentes modelos son conformes a un mismo metamodelo* (escenario 2). En este caso los elementos de ambos modelos son tratados por igual; es decir, les afectan las mismas reglas ya que pertenecen al mismo metamodelo. Si el hecho de que un elemento pertenezca a un modelo o a otro tiene implicaciones semánticas y se desea realizar un tratamiento diferente, la redacción de las reglas se complica para diferenciar los elementos según el modelo al que pertenecen. En este caso es necesario incluir guardas que explícitamente seleccionen los elementos sobre los que se debe aplicar una regla.

Por ejemplo, en el código que se muestra a continuación se seleccionan explícitamente los elementos EPackage del modelo de entrada “IN”.

```
from
p1:.ecore!EPackage
(
.ecore!EPackage.allInstancesFrom('IN')->includes(p1)
// Resto del código de la transformación
)
```

También se presentan *complicaciones en el caso de tratarse de un número indefinido de modelos* (escenarios 4 y 5) ya que el lenguaje no soporta de manera natural esta posibilidad. Sin embargo es posible solventar el problema utilizando artificios externos.

Respecto al tratamiento de los enlaces, el entorno de desarrollo de ATL permite indicar explícitamente que la transformación contiene referencias entre modelos. Para ello es necesario activar la propiedad “Allow inter-model refer” (Figura 7). En este caso **los enlaces a modelos referenciados** (caso 1) se tratan correctamente.

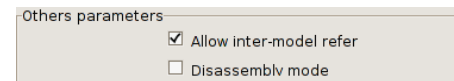


Figura 7. Propiedad de ATL para tratar referencias entre modelos

Por contra el tratamiento de **enlaces entre modelos principales en la transformación** (caso 2) no es el deseado, ya que en todo momento se tratan como si fueran enlaces a modelos referenciados. Esto puede ser origen de complicaciones dependiendo de qué transformación se desee realizar. Por lo tanto, se debe tener en cuenta que, como comportamiento general, el elemento que se obtiene siguiendo el enlace se considera un elemento distinto al presente en el modelo de la transformación.

5.2. SmartQVT

Respecto al **tratamiento de los modelos de entrada y salida**, SmartQVT tiene un comportamiento similar a ATL. También es *correcto el tratamiento de un único modelo* (escenario 1) o *modelos conformes a distintos metamodelos* (escenario 3).

Sin embargo, una diferencia importante con ATL es el tratamiento de modelos pertenecientes a un

mismo metamodelo (escenario 2). En una transformación de SmartQVT los metamodelos reciben un nombre identificativo, siendo posible que diferentes nombres para los metamodelos referencien en realidad al mismo. Para ello, en el archivo de configuración se utilizarán dos identificadores distintos referenciando el mismo identificador de metamodelo, tal y como se muestra a continuación:

```
Ecore.nsURI=http://www.eclipse.org/emf/2002/Ecore
Ecore2.nsURI=http://www.eclipse.org/emf/2002/Ecore
UML.nsURI=http://www.eclipse.org/uml2/2.0.0/UML
UML2.nsURI=http://www.eclipse.org/uml2/2.0.0/UML
```

Mientras que en la cabecera de la transformación se utilizaría de la siguiente manera:

```
transformation ECore2UML(in.ecoremodel:Ecore, in.ecoremodel2:Ecore2, out.umlmodel:UML2, out.umlmodel2:UML22);
```

Esta técnica se puede utilizar en ATL, pero en este caso alias distintos apuntando a los mismos metamodelos no se consideran metamodelos diferentes.

Una vez que los nombres utilizados para los metamodelos son distintos, la herramienta los tratará como si de metamodelos distintos se tratase. Esto implica una ventaja ya que es posible identificar unívocamente cada uno de los modelos implicados en la transformación; pero también supone una desventaja porque elementos de distintos modelos son incompatibles entre sí al pertenecer a metamodelos diferentes.

SmartQVT *tampoco proporciona un soporte explícito para el tratamiento de transformaciones en las que el número de modelos no es conocido* (escenarios 4 y 5). Como con ATL, se debe recurrir a manipulaciones externas mediante composición de transformaciones.

En cuanto al **tratamiento de enlaces entre modelos** las dolencias son las mismas que con ATL. Los enlaces a modelos referenciados (caso 1) se interpretan correctamente (en este caso sin necesidad de indicar en la transformación la presencia de los mismos). En cambio, los enlaces

con modelos principales en la transformación (caso 2) también se consideran como externos.

5.3. QVT Operacional de Borland®

Respecto al tratamiento de los modelos manipulados, es importante destacar que la implementación del lenguaje QVT Operacional realizada por Borland® *únicamente soporta transformaciones que manipulen un modelo de entrada y uno de salida* (escenario 1). Por lo tanto, *no es posible realizar todos los escenarios que manipulen más de un modelo* (escenario 2, 3, 4 y 5).

Respecto al **tratamiento de las relaciones entre modelos**, maneja adecuadamente el acceso a modelos referenciados en la transformación (escenario 1). Debido a que no soporta múltiples modelos, no procede estudiar el soporte de los enlaces entre modelos principales en la transformación.

5.4. Resumen de la evaluación

A continuación se muestra una tabla en la que se resume el resultado obtenido con cada una de las soluciones propuestas. Para rellenar esta tabla se ha utilizado una serie de símbolos que se describen a continuación:

√: El escenario se implementa con éxito.

√₊: El escenario se implementa, superando algunas dificultades mediante mecanismos ofrecidos por la tecnología y con mayor flexibilidad que las otras tecnologías incluidas en la comparativa.

√₋: El escenario se implementa, superando algunas dificultades mediante mecanismos ofrecidos por la tecnología pero con menor flexibilidad que las otras tecnologías incluidas en la comparativa.

×₊: La tecnología no permite solucionar el problema planteado, pero es posible implementarlo utilizando mecanismos externos.

×: La tecnología no permite solucionar el problema planteado.

	ATL	SmartQVT	Borland QVT Operational
Número de modelos			
Modelos de entrada			
1.- Un sólo modelo	√	√	√
2.- Un número definido de modelos del mismo metamodelo	√.	√+	×
3.- Un número definido de modelos de distintos metamodelos	√	√	×
4.- Un número indefinido de modelos del mismo metamodelo	√+	√+	×
5.- Un número indefinido de modelos de distintos metamodelos	×+	×+	×
Modelos de salida			
1.- Un sólo modelo	√	√	√
2.- Un número definido de modelos del mismo metamodelo	√.	√+	×
3.- Un número definido de modelos de distintos metamodelos	√	√	×
4.- Un número indefinido de modelos del mismo metamodelo	×+	×+	×
5.- Un número indefinido de modelos de distintos metamodelos	×+	×+	×
Enlaces entre modelos			
Modelos de entrada			
1.- Enlaces con modelos referenciados en la transformación	√	√	√
2.- Enlaces con otros modelos principales de entrada la transformación	×+	×+	×
Modelos de salida			
1.- Enlaces con modelos referenciados a la transformación	√	√	√
2.- Enlaces con otros modelos principales de salida la transformación	×+	×+	×

6. Conclusión

En este trabajo se ha definido un marco de comparación para la evaluación de tecnologías de transformación de modelos en la plataforma Eclipse y se ha aplicado a tres tecnologías: ATL,

SmartQVT y la implementación de QVT Operational de Borland®. Como resultado de la aplicación del marco de comparación se puede concluir que tanto ATL como SmartQVT tratan de manera similar las características estudiadas en el marco. Mientras que la implementación de QVT Operational de Borland® presenta importantes carencias debido a la imposibilidad de tratar con

más de un modelo. Por lo tanto, la decisión de optar por ATL o SmartQVT depende más de propiedades no contempladas en este marco de comparación, como la dependencia de tecnologías externas (SmartQVT necesita de un intérprete externo de Python) o la base de usuarios y documentación existente (ATL tiene una larga trayectoria, es utilizado actualmente en numerosos proyectos y se pueden encontrar fácilmente ejemplos de uso; mientras que SmartQVT es más reciente, pero implementa un estándar).

Existen otros trabajos que presentan propiedades para realizar clasificaciones de lenguajes o tecnologías de transformación. Czarnecky, en [1], propone una serie de características para la clasificación de lenguajes de transformación de modelos. Estas características incluyen la direccionalidad de las reglas, el tipo de sintaxis, la ejecutabilidad, la estrategia de aplicación de las reglas, etc. Por otra parte, Mens [6] propone una taxonomía de transformaciones de modelos basado en las discusiones de un grupo de trabajo de los seminarios de Dagstuhl (en los que participó el propio Czarnecky). En esta taxonomía se clasifican las características en cuatro áreas que responden a cuatro preguntas:

- ¿Qué necesita ser transformado en qué?
- ¿Cuáles son las características importantes de una transformación de modelos?
- ¿Cuál es el criterio de éxito de un lenguaje o herramienta de transformación?
- ¿Qué mecanismos pueden utilizarse para transformarse modelos?

A diferencia de estos trabajos, más generales, el trabajo presentando en este artículo se centra en características muy específicas en un espacio tecnológico muy concreto, como es la plataforma Eclipse. Además, el trabajo se acompaña de una serie de casos de prueba que facilitan la realización de la fase de evaluación. De este modo, el trabajo tiene una aplicabilidad más inmediata y esperamos que un alto valor para aquellas empresas y entidades que deseen o necesiten seleccionar una tecnología de transformación de modelos en un contexto similar.

Actualmente nuestro trabajo en esta línea de investigación se centra en enriquecer el marco de comparación con nuevas características que se identifiquen como potenciales fuentes de problemas a la hora de implementar transformaciones entre modelos. Por otra parte,

también estamos interesados en ampliar el abanico de tecnologías incluidas en la comparativa, de manera que se pueda disponer de mayor información sobre las distintas opciones disponibles a la hora de seleccionar una herramienta de transformación de modelos.

Referencias

- [1] F. Allilaire and T. Idrissi. ADT: Eclipse Development Tools for ATL. In Proceedings of the Second European Workshop on Model Driven Architecture (EWMDA-2), Canterbury, UK, September 2004.
- [2] E. Biermann, K. Ehrig, C. Köhler, G. Kuhns, G. Taentzer and E. Weiss. Graphical Definition of In-Place Transformations in the Eclipse Modeling Framework. In *Model Driven Engineering Languages and Systems*. volume 4199/2006 pages 425-439 of Lecture Notes in Computer Science, Springer-Verlag.
- [3] Czarnecki, K., Helsen, S. Classification of model transformation approaches. OOPSLA2003 Workshop on Generative Techniques in the Context of MDA, Anaheim, CA, USA, 2003
- [4] J. Sánchez, J. García, and M. Menarguez. RubyTL: A Practical, Extensible Transformation Language. In *2nd European Conference on Model Driven Architecture*, volume 4066, pages 158--172. Lecture Notes in Computer Science, June 2006.
- [5] Jouault, F., Kurtev, I.: Transforming models with ATL. In: Satellite Events at the MoDELS 2005 Conference. Volume 3844 of Lecture Notes in Computer Science., Springer-Verlag (2006) 128–138
- [6] T. Mens and P. Van Gorp, "A Taxonomy of Model Transformation," *Proc. Int'l Workshop Graph and Model Transformation*, 2005
- [7] OMG: MOF QVT Final Adopted Specification, OMG Document ptc/2005-11-01, <http://www.omg.org/docs/ptc/05-11-01.pdf>. (2005)