Análisis basado en variables para trazabilidad en transformación de modelos

Omar Martínez Grassi, Claudia Pons

LIFIA, Facultad de Informática, UNLP, CAETI - Universidad Abierta Interamericana (UAI)

omartinez@rionegro.com.ar, cpons@lifia.info.unlp.edu.ar

Resumen. El desarrollo de software conducido por modelos (Model-Driven Software Development o MDSD) es un nuevo paradigma, cuyo concepto central son los modelos y sus transformaciones. El objetivo fundamental es separar la especificación de la funcionalidad del sistema de la especificación de la implementación de dicha funcionalidad sobre una plataforma específica. En este contexto, la trazabilidad ha ganado un rol principal dentro del desarrollo de software dado que la misma permite, entre otras cosas, la posibilidad de realizar trazas hacia atrás (*backward*) ante cambios en los requerimientos elicitados en etapas tempranas, y la posibilidad de evaluar el impacto de las modificaciones a realizarse en fases avanzadas. Este trabajo propone una técnica que permite la obtención de información de rastreabilidad a partir de la definición de una transformación de modelos escrita en lenguaje QVT Relations. Dicho proceso es totalmente automático y no depende de la ejecución de la transformación.

1. Introducción

La Arquitectura Conducida por Modelos o *Model Driven Architecture* (MDA) es un *framework* para desarrollo de software definido por el Object Management Group (OMG) cuyo concepto central son los modelos y sus respectivas transformaciones [9]. En efecto, MDA propone un ciclo de desarrollo basado en la transformación de un modelo de alto nivel en otro, con menor nivel de abstracción, que finalmente será convertido en código fuente. Dichos modelos son el PIM (Platform Independent Model) o modelo independiente de la plataforma, el PSM (Platform Specific Model) o modelo específico de la plataforma, y el código fuente. Una transformación es un proceso; cada proceso es descrito por una definición, la cual a su vez se compone de reglas, que son ejecutadas por herramientas de transformación. En particular, a los efectos del trabajo propuesto, nos interesa el estudio de la propiedad de trazabilidad en transformación de modelos.

Desde hace algunos años el OMG adoptó como estándar de transformación de modelos a QVT (Query/View/Transformation), una especificación de naturaleza híbrida declarativa/imperativa [10], que integra el estándar OCL 2.0 y lo extiende a su versión imperativa, definiendo tres lenguajes de dominio específico (DSL)

llamados *Relations*, *Core* (ambos declarativos) y *Operational Mappings* (imperativo). En este contexto, el proyecto Eclipse Modeling Framework (EMF), provee un entorno de modelado y generación de código para el desarrollo de aplicaciones basado en modelos que puede ser especificado mediante un subconjunto del lenguaje Java (conocido como Java Anotado), documentos XML o herramientas de modelado como Rational Rose™ [3]. El proyecto incluye la definición de Ecore, una implementación de MOF [10], herramienta fundamental para la representación de modelos. Al margen de Eclipse, no muchas herramientas implementan los lenguajes definidos por el estándar QVT. Entre ellas podemos mencionar a mediniQVT [1] (QVT Relations), a SmartQVT [12] (QVT Operational Mappings), y a OptimalJ (QVT Core).

Este trabajo presenta una propuesta de soporte de trazabilidad en transformación de modelos mediante el análisis del código QVT, la cual permite la inferencia de trazas entre los modelos origen y destino a partir de la especificación de la transformación, de forma sistemática, sin necesidad de código adicional ni intervención por parte del desarrollador.

2. Trazabilidad en transformación de modelos

2.1. El concepto de trazabilidad

El Glosario Estándar de Ingeniería de Software del IEEE [11] define trazabilidad como el grado en que una relación puede ser establecida entre dos o más productos del proceso de desarrollo, especialmente en aquellos que presentan relaciones predecesor-sucesor o maestro-subordinado entre sí, o el grado en que cada elemento de un producto de desarrollo de software establece su razón de existir. Es posible, sin embargo, encontrar definiciones más amplias, y más útiles a los efectos del desarrollo conducido por modelos. En [2] se define la trazabilidad como "cualquier relación que exista entre artefactos involucrados en el ciclo de vida de la ingeniería de software". Esta definición incluye relaciones explícitas o mapeos que son generados como resultado de transformaciones tanto hacia adelante (*forward*, como generación de código), como hacia atrás (*backward*, como ingeniería inversa), relaciones que son computadas en base a información existente (por ejemplo análisis de dependencia de código) o relaciones estadísticamente inferidas. De esta manera, la trazabilidad es lograda definiendo y manteniendo las relaciones entre los artefactos involucrados durante el ciclo de vida de la ingeniería de software, durante el desarrollo del sistema.

2.2. Trabajos relacionados

La generación automática de información de trazabilidad ha sido tema de investigación de varios trabajos. Uno de los primeros estudios del tema puede encontrarse en [8]. El mismo está basado en la obtención de trazas mediante un proceso escasamente acoplado que permite generar este tipo de información sin alterar la definición de las transformaciones de modelos, en el contexto del lenguaje ATL (ATLAS Transformation Language), un lenguaje de transformación

de modelos [7] presentado como candidato en el RFP (*Request For Proposal*) de QVT lanzado por el Object Management Group (OMG) [6].

Un planteo más complejo puede encontrarse en [5]. Este estudio propone un framework de trazabilidad genérico para aumentar arbitrariamente diversos esquemas de transformación de modelos con mecanismos de rastreabilidad, sostenido por un lenguaje de dominio específico (*domain-specific language*) para trazabilidad, llamado *Trace-DSL*. El trabajo abarca un amplio espectro de aspectos de trazabilidad del paradigma de desarrollo conducido por modelos, y propone una solución genérica para distintos lenguajes de transformación de modelos.

Otro de los variados trabajos en la materia está basado en la generación de información de rastreabilidad mediante la utilización de un lenguaje de metamodelado [4]. Éste, denominado TML (*Traceability Metamodelling Language*) permite definir un metamodelo TM_M que capture toda la información de trazabilidad posible entre un metamodelo origen MM_A y uno destino MM_B para un dominio particular.

Más adelante se analizarán en detalle estos trabajos, y se compararán con el esquema propio presentado en el presente documento.

3. Análisis basado en variables

El presente estudio aborda la problemática de la obtención de información de rastreabilidad de manera automática, es decir, sin tener que depender de alguien que especifique de qué manera son generados los elementos de un modelo destino a partir de un modelo origen ni de la ejecución de una transformación. A diferencia de otras propuestas similares, el estudio plantea que dadas las características sintáctico-gramaticales del lenguaje de especificación de transformaciones estándar QVT, es posible inferir cierto tipo de trazas mediante el análisis del código fuente. Este análisis consiste en el reconocimiento de ciertas estructuras dentro de la especificación de dicha transformación en el lenguaje QVT Relations que pueden ser traducidas como trazas, y permiten explicar luego el origen de algunos elementos del modelo destino. Se han determinado cuatro tipos de trazas que pueden ser reconocidos a partir del estudio del código mencionado. La presente sección describe las características que posibilitan la inferencia de trazas.

3.1. Inferencia de trazas: el análisis

A continuación estudiaremos los patrones identificados que permiten la derivación de trazas, ejemplificando cada caso con un fragmente de código fuente QVT donde se presentan:

Caso 1: Inferencia de trazas mediante una variable auxiliar Cuando una regla obligatoria (*top-level*) asigne un valor a algún elemento del modelo destino definido en el ámbito de un dominio de tipo *enforce*, mediante el uso de una variable

previamente utilizada sobre un elemento del modelo origen de igual manera definido en el ámbito de un dominio *checkonly*, podremos decir que el elemento del modelo origen mapeará directamente en el elemento del modelo destino. Si tomamos como ejemplo el fragmento de código QVT de la Figura 1, vemos que la variable *pn* nos permite inferir una traza entre los atributos *umlName* y *rdbmsName* de las entidades *UmlPackage* y *RdbmsSchema*. De ésta manera, con el análisis propuesto, la traza determinada sería *umlName::UmlPackage* → *rdbmsName::RdbmsSchema*, es decir, el atributo *rdbmsName* de toda entidad de tipo *RdbmsSchema* será igual al atributo *umlName* de la entidad *UmlPackage* del modelo SimpleUML (origen) a partir del cual fue derivado el modelo SimpleRDBMS (destino).

```
top relation PackageToSchema {
 pn : String;
 checkonly domain uml p:SimpleUML::UmlPackage
 { umlName = pn };
 enforce domain rdbms s:SimpleRDBMS::RdbmsSchema
 { rdbmsName = pn };
}
```

Fig. 1. Uso de variable auxiliar

```
relation ClassToPkey {
 cn : String;
 checkonly domain uml c:SimpleUML::UmlClass
  { umlName = cn };
 enforce domain rdbms k:SimpleRDBMS::RdbmsKey
  { rdbmsName = cn + '_pk' };
}
```

Fig. 2. Función de variable auxiliar

Caso 2: Inferencia de trazas mediante una expresión en función de una variable auxiliar Este caso es una generalización del descripto anteriormente, cuya diferencia se basa en que el elemento del modelo destino, definido en el ámbito de un dominio de tipo *enforce,* será una función de la variable utilizada con igual fin sobre el elemento del modelo origen (Figura 2). Como vemos, la expresión que describe el valor que el atributo *rdbmsName* adoptará luego de la transformación está dado por la función *F* de la variable *cn*, definida como *F(cn) = cn + '_pk'*, donde el operador '+' representa la concatenación de cadenas de caracteres. En dicho caso, podremos inferir que todo atributo *rdbmsName* de la entidad *RdbmsKey* en un modelo SimpleRDBMS será igual al atributo *umlName* de la entidad *UmlClass*, del modelo SimpleUML, concatenado al sufijo '_pk' o, en forma equivalente, que *rdbmsName = umlName + '_pk'*.

Caso 3: Inferencia de trazas mediante el uso de una constante Se define para aquellos casos en los cuales un elemento del modelo destino definido en el ámbito de un dominio *enforce* es inicializado con un valor constante. Tomando como ejemplo el código presentado en la Figura 3, vemos que el atributo *rdbmsType* de toda entidad *rdbmsColumn* será igual a la constante 'NUMBER' independientemente de los valores de los elementos del modelo origen asociado.

Caso 4: Inferencia de trazas mediante una variable auxiliar, definida como función en la cláusula Where Este caso es análogo al primer tipo de traza, descripto en el Caso 1. La diferencia radica en que la variable auxiliar es definida como una función de otras variables en alguna sentencia de la cláusula *where* de la

```
top relation ClassToTable {
  . . .
  enforce domain rdbms t : SimpleRDBMS::RdbmsTable {
    rdbmsSchema = s : SimpleRDBMS::RdbmsSchema {},
    rdbmsName = cn,
    rdbmsColumn = cl : SimpleRDBMS::RdbmsColumn {
      rdbmsName =cn + '_tid',
      rdbmsType = 'NUMBER'
    },
    rdbmsKey = k : SimpleRDBMS::RdbmsKey {
      rdbmsColumn = cl : SimpleRDBMS::RdbmsColumn{}
    }
  };
  . . .
}
```

Fig. 3. Ejemplo de uso de una constante

relación (Ver Figura 4). En este caso se inferirá, por un lado, que el nombre (atributo *rdbmsName*) de una columna (entidad *rdbmsColumn*) dentro de una clave foránea (entidad *rdbmsForeignKey*) será la concatenación del nombre (atributo *umlName*) de la clase fuente (entidad *umlSource*) concatenada mediante el símbolo '_' al nombre (atributo *umlName*) de la asociación (entidad *umlAssociation*), concatenado a su vez con mediante el símbolo '_' al nombre (atributo *umlName*) de la clase destino (entidad *umlDestination*) + '_tid'. Por otro lado, se determinará que el nombre (atributo *rdbmsName*) de la clave foránea (entidad *rdbmsForeignKey)* será análogo al primero, sin el sufijo '_tid'.

3.2. Ventajas y desventajas del análisis basado en variables

En primer lugar, la técnica presentada es completamente automática, es decir, en ninguna etapa del proceso de análisis se requiere la intervención de un ser humano para obtener sus resultados. Por tal, colabora con la productividad de los ingenieros al no requerir esfuerzos, y está libre de errores que éstos pudieran cometer. En segundo lugar, a diferencia de algunas implementaciones, como por ejemplo mediniQVT [1], la información de trazabilidad es generada a nivel de modelo, no de instancias, de manera que permite determinar no sólo el mapeo de un elemento en otro, sino la expresión o forma del mismo, sea cual fuere la instancia del modelo origen que se transforme o la correspondiente instancia del modelo destino. Esto, a su vez, permite la posibilidad de verificar y eventualmente forzar la consistencia e integridad de la relación entre ambos modelos, lo cual puede ser de gran ayuda en particular cuando el modelo destino es modificado unilateralmente, y no como resultado de cambios en el modelo origen luego procesados por la transformación, lo que sería el flujo natural del proceso de modificación. Otra de las ventajas que se observan es que la obtención de trazas no depende del proceso de transformación, sino sólo de su definición. En consecuencia, esto puede ayudar al desarrollador como herramienta de *debugging* en la depuración de la especificación de la misma, brindándole indicios de los resultados que se obtendrán tras la aplicación de la misma. Esta independencia, a su vez, brinda flexibilidad y facilita el mantenimiento de la información de ras-

```
top relation AssocToFKey {
  an : String;
  scn : String;
  dcn : String;
  fkn : String;
  fcn : String;
  checkonly domain uml a : SimpleUML::UmlAssociation {
    umlNamespace = p : SimpleUML::UmlPackage {},
    umlName = an,
    umlSource = sc : SimpleUML::UmlClass {
      umlKind = 'Persistent',
      umlName = scn
    },
    umlDestination = dc : SimpleUML::UmlClass {
      umlKind = 'Persistent',
      umlName = dcn
    }
  };
  enforce domain rdbms fk : SimpleRDBMS::RdbmsForeignKey {
    rdbmsName = fkn,
    rdbmsOwner = srcTbl : SimpleRDBMS::RdbmsTable {
      rdbmsSchema = s : SimpleRDBMS::RdbmsSchema {}
    },
    rdbmsColumn = fc : SimpleRDBMS::RdbmsColumn {
      rdbmsName = fcn,
      rdbmsType = 'NUMBER',
      rdbmsOwner = srcTbl
    },
    rdbmsRefersTo = pKey : SimpleRDBMS::RdbmsKey {
      rdbmsOwner = destTbl : SimpleRDBMS::RdbmsTable {}
    }
  };
  when {
    ClassToPkey(dc, pKey);
    PackageToSchema(p, s);
    ClassToTable(sc, srcTbl);
    ClassToTable(dc, destTbl);
  }
  where {
    fkn = scn + '_' + an + '_' + dcn;
    fcn = fkn + '_tid';
  }
}
```

Fig. 4. Variable auxiliar y expresión en cláusula *Where*

treabilidad ya que la misma puede ser almacenada en un repositorio, o generarse *ad hoc*, sin contaminar los modelos ni la especificación de la transformación.

La principal desventaja de la propuesta consiste en que el análisis de la especificación de una transformación y de los modelos origen y destino, debe ser realizada dos veces: para el análisis basado en variables (por la herramienta que lo implementa), y al momento de la ejecución de la transformación por parte del motor que implemente QVT. Por otro lado, las trazas generadas no necesariamente son las únicas existentes, y es factible que evetualmente existan trazas que el método no pueda determinar. El presente trabajo no pretende demostrar que las trazas que pueden ser inferidas son la totalidad de las existentes.

3.3. Soporte tecnológico: QVTrace

La propuesta descripta ha sido implementada en una herramienta llamada QVTrace. La misma se trata de un plugin de Eclipse, que a pesar de ser un prototipo nos ha permitido llevar a la práctica la presente técnica y comprobar su aplicabilidad en el marco del MDSD (Model-Driven Software Development). QVTrace a sido desarrollada con la visión de ser una herramienta complementaria a otras disponibles en el marco del desarrollo conducido por modelos. El hecho de ser un plugin de Eclipse, le brinda versatilidad y acentúa su interoperabilidad con el resto de los programas afines. Sus entradas son la definición de la transformación (código QVT) y los modelos origen y destino en formato Ecore, estándar de representación de modelos en el EMF [3], que es el marco de trabajo de desarrollo conducido por modelos de Eclipse, y la salida generada es una colección de trazas definidas en un metamodelo *ad-hoc*. Dicha implementación contempla los cuatro casos de inferencia de trazas descriptos en este trabajo.

Metamodelo de trazabilidad Para el modelado de trazas se desarrolló un metamodelo basado en la simplicidad, ajustado a las necesidades del problema abordado (Figura 5). El mismo consta de una clase *Trace* que mantiene la información asociada a la traza, en este caso los elementos origen y destino (atributos *source* y *target*), la relación (atributo *relation*) que la generó, y la expresión (atributo *expression*) de la misma. Los elementos relacionados por la traza son de tipo *TraceElement*, los cuales son a su vez subtipos de *TraceableModelElement*, una clase abstracta que determina qué tipo de elemento del modelo puede ser incluidos en una traza. Este diseño está relacionado con la representación de modelos utilizada, en el caso puntual de QVTrace se adoptó la representación Ecore, estándar del EMF, y se limitó a objetos de tipo *EStructuralFeature*, que son los componentes que conforman una *EClass*, la cual es un tipo de componente de modelo posible que brinda dicha especificación, y se compone de objetos de tipo *EReference* y *EAttribute*.

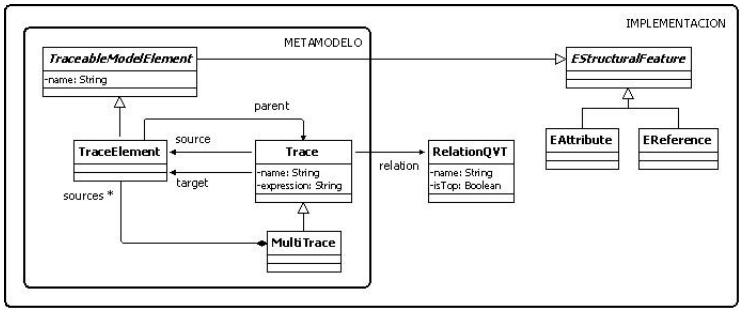

Fig. 5. Metamodelo de trazabilidad implementado por QVTrace

A diferencia de los metamodelos propuestos en trabajos de similar índole, el desarrollo presentado aquí modela las trazas como una relación unívoca entre un

elemento del modelo origen y uno del modelo destino, mientras que en la mayoría de los casos esta relación se generaliza como *muchos-a-muchos*. La propuesta sí contempla una posible traza entre n elementos del modelo origen y uno del modelo destino, la cual es tipificada en la clase *MultiTrace*, subtipo de clase *Trace*. Este enfoque, que podría ser considerado un limitación desde el diseño, responde en realidad a una virtud. El algoritmo de inferencia de trazas trabaja a nivel del mínimo elemento rastreable en el contexto de la representación de modelos elegida, Ecore, y por las características del análisis basado en variables, si uno o más elementos del modelo origen generan múltiples elementos del modelo destino, entonces se generaran múltiples trazas *Trace* o *MultiTrace*, según el caso.

La segunda diferencia con la mayoría de los metamodelos propuestos tiene que ver con la semántica de la traza. Uno de los atributos de la clase *Trace*, denominado *expression*, almacena la expresión que revela el significado de la transformación, es decir, de qué manera un elemento del modelo origen de transforma en un determinado elemento del modelo destino. Por ejemplo, asumiendo una transformación $A2B$ donde un elemento x del modelo A se convierte en un elemento y del modelo B, tendremos una traza $x \rightarrow y$ donde el atributo *expression* contendrá $y = x$, agregando significado a la relación.

Soporte para la inferencia de trazas La inferencia de trazas es soportada a través del esquema propuesto en la Figura 6. El mismo consta de un componente llamado *TraceAnalizer*, el cual utilizando los modelos origen y destino (objetos *Model*), la definición de la transformación (objeto *TransformationQVT*), y una estrategia de rastreabilidad (componentes que implementan la interfase *ITraceStrategy*) genera las trazas correspondientes. La estrategia de trazabilidad es en

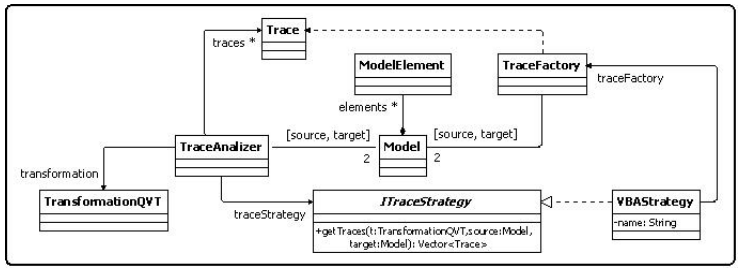

Fig. 6. Soporte de trazas en QVTrace

esencia el mecanismo mediante el cual se obtienen las trazas correspondientes. El diseño está pensado para este componente pueda ser fácilmente extendido o reemplazado por otro que implemente el método que define la interfase mencionada, el cual como puede verse en la signatura recibe una transformación QVT, un par de modelos origen y destino y devuelve como resultado una colección de trazas de tipo *Trace*. La responsabilidad de la creación de trazas está a cargo del objeto *TraceFactory*, quien es el encargado de la generación de objetos *Trace*

(ver flecha con línea punteada en el diagrama). Toda estrategia de trazabilidad implementada debe recurrir a este objeto factoría para la creación de las mismas.

Diseño de la solución En términos generales, el flujo de trabajo propuesto por QVTrace comienza con el procesamiento de los datos de entrada, y finaliza con la obtención de las trazas. Este proceso consta de tres fases:

1. Lectura de los modelos origen y destino, en formato Ecore, y generación de los objetos correspondientes.
2. Lectura y parsing de la definición de la transformación, en lenguaje QVT, modelado y transformación en objetos.
3. Análisis de la definición de la transformación y obtención de trazas mediante la aplicación de la estrategia de trazabilidad definida. Creación de objetos *Trace*.

La Figura 7 muestra esquemáticamente los componentes de QVTrace. Las entradas del proceso son los archivos de modelos (.Ecore) origen y destino, y el código QVT de la transformación, los cuales son utilizados para la inferencia de trazas . La obtención de trazas es realizada por un componente llamado *TraceAnalizer*, en colaboración con dos objetos fundamentales en el proceso: uno de tipo *TraceStrategy*, el cual implementa la estrategia utilizada para la inferencia de las trazas, y otro de tipo *TraceFactory*, el cual se encarga de la creación de trazas. De esta forma, desacoplamos la creación de la traza, y por tanto el conocimiento del metamodelo, del proceso de inferencia de trazas.

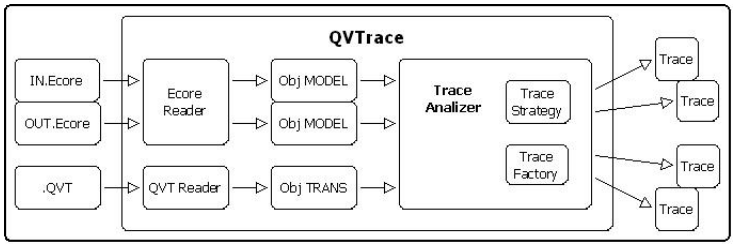

Fig. 7. Esquema general de QVTrace

4. Comparación frente a otros enfoques

A efectos de poder valorar el contenido de la propuesta presentada, a continuación se desarrollará un estudio comparativo con los trabajos relacionados que fueron abordados en la Sección 2.2. En primer lugar se analizará la propuesta de

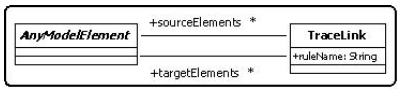

Fig. 8. Metamodelo de trazabilidad de Jouault

trazabilidad escasamente acoplada para ATL de Jouault [8], y luego continuaremos con el esquema de Grammel [5] de extración de datos de rastreabilidad basado en *facets*. El objetivo de la comparación es mostrar de qué manera ha abordado cada enfoque el problema de obtención de información de rastreabilidad, y contrastar diferencias y semejanzas de los trabajos analizados con la propuesta propia. La comparación se basa en dos puntos críticos de todo enfoque de esta índole: por un lado, el metamodelo de trazabilidad planteado, es decir, la manera que cada propuesta representa las trazas, y por otro, en el mecanismo de obtención de información de trazabilidad implementado por cada uno.

4.1. Esquema de trazabilidad escasamente acoplada

La propuesta de Jouault fue uno de los primeros trabajos de generación de información de trazabilidad automática en el contexto del desarrollo conducido por modelos. Es un trabajo de referencia, que muestra una solución absolutamente enmarcada en el paradigma MDSD (*Model-Driven Software Development*) y por tal ha sido elegido como parámetro de comparación frente al estudio presentado en este trabajo.

Metamodelo de trazabilidad El trabajo propone un metamodelo de trazabilidad simple, compuesto por una clase *TraceLink*, la cual contiene un atributo para almacenar el nombre de la regla que genera dicha traza, y mantiene dos colecciones de objetos de tipo *AnyModelElement*, denominadas *sourceElements* y *targetElements* (Figura 8), donde dichas colecciones almacenan los elementos origen y destino que se encuentran relacionados por la aplicacion de la regla. La clase *AnyModelElement* es abstracta, y obviamente dependerá de entorno de aplicación del metamodelo.

Obtención de la información de trazabilidad La propuesta sugiere la obtención de trazas mediante el agregado de código extra ATL a las reglas que conforman la transformación y agregando un modelo de salida de trazabilidad, que permite que la información de rastreabilidad sea capturada al momento de producirse la transformación de modelos. Esta modificación no altera la lógica del programa, si bien agrega contenido adicional a la definición de la transformación. Adicionalmente los autores presentan un esquema, también soportado por el mismo lenguaje de transformación, que permite que el código adicional pueda ser agregado automáticamente a las reglas de la definición, sin necesidad de hacerlo manualmente. Como ejemplo, se presenta la definición de una transformación sencilla, de un modelo origen A, con un único atributo llamado *name*, de tipo *String*[1], en un modelo destino B, con un único atributo llamado *name* también,

[1] Cadena de caracteres, en términos genéricos

de tipo *String*. La Figura 9 muestra el código original, el cual para obtener el soporte de trazabilidad es reemplazado luego por la versión de la Figura 10. El ejemplo, si bien es elemental, nos permite realizar las siguientes observaciones:

```
1. module Src2Dst;
2.  create OUT : Dst from IN : Src;
3.
4.  rule A2B {
5.    from
6.      s : Src!A
7.    to
8.      t : Dst!B (
9.          name <- s.name
10.     )
11. }
```

Fig. 9. Transf. Src2Dst en ATL

```
1. module Src2DstPlusTrace;
2.  create OUT : Dst, trace : Trace from IN : Src;
3.
4.  rule A2BPlusTrace {
5.   from
6.     s : Src!A
7.   to
8.     t : Dst!B (
9.         name <- s.name
10.    ),
11.    traceLink : Trace!TraceLink (
12.      ruleName <- 'A2BPlusTrace',
13.      targetElements <- Sequence {t}
14.    )
15.    do {
16.      traceLink.refSetValue('sourceElements',
17.        Sequence {s});
18.    }
19. }
```

Fig. 10. Src2Dst modificada

- La obtención de información de trazabilidad es implementada a nivel de la transformación. Requiere de su ejecución para la generación de las trazas.
- No afecta la lógica de la definición, pero agrega información adicional que hace difusa la legibilidad de la transformación original.
- Es absolutamente independiente de los modelos involucrados en la transformación.
- Utiliza un metamodelo de trazabilidad genérico, adaptable a otros posibles esquemas.
- El proceso de acondicionamiento automático de reglas para el soporte de trazabilidad puede ser realizado antes de la compilación de la transformación.

4.2. Extracción de datos de trazabilidad basado en *Facets*

El trabajo de Grammel propone un framework de trazabilidad genérico que permite aumentar arbitrariamente un modelo de transformación con información de rastreabilidad. Es una propuesta genérica diseñada para dar soporte a cualquier aproximación de transformación de modelos. El framework está basado en una interfase de trazabilidad genérica (GTI, *Generic Traceability Interfase*) que provee el punto de conexión para lenguajes de transformación arbitrarios y brinda un API para conectarlos con el motor de trazabilidad. El framework, además, define un lenguaje de dominio específico llamado Trace-DSL que en esencia determina qué tipo de información de trazabilidad es intercambiable entre la interfaz genérica y los conectores de los motores de rastreabilidad.

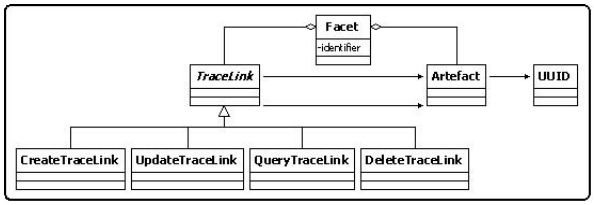

Fig. 11. Fragmento de lenguaje Trace-DSL

Metamodelo de trazabilidad El esquema propuesto por Grammel es muy interesante dado que adopta un enfoque totalmente distinto al de la mayoría de los autores. A continuación analizaremos un fragmento del lenguage de dominio específico Trace-DSL desarrollado para dar soporte de trazabilidad (Figura 11). En este modelo, las trazas son representadas por el componente *TraceLink*, el cual es considerado una abstracción de la transición de un artefacto hacia otro. El objeto *Artefact*, individualizado con un único identificador universal (*UUID*) representa cualquier producto rastreable generado en el proceso de desarrollo, como un requerimiento o clase, o un componente de otro artefacto (por ejemplo un método de una clase). La transición representada siempre es dirigida, por lo que una traza entre artefactos origen y destino siempre da lugar a una relación *desde-hasta* entre dichos artefactos. Considerando a la trazabilidad como el seguimiento de todos los cambios posibles aplicados a elementos de modelos durante una transformación, Grammel propone dividir dicha transformación en un conjunto de operaciones elementales eo_s y definir un tipo de traza para cada operación, con los correspondientes elementos de los modelos origen y destino involucrados. De acuerdo a estas consideraciones, el autor propone los siguientes tipos:

- *CreateTraceLink*: Para generación de nuevos elementos del modelo destino.
- *UpdateTraceLink*: Para modificaciones sobre elementos existentes.
- *DeleteTraceLink*: Para operaciones de eliminación de elementos.
- *QueryTraceLink*: Para operaciones de consulta sobre modelos.

Para asignar tipos a los artefactos y trazas, la propuesta utiliza el concepto de faceta (*facet*), donde el lenguaje Trace-DSL asigna un conjunto de facetas a cada artefacto y traza, simplificando la jerarquía de tipos y buscando dotar de extensibilidad al metamodelo propuesto.

Obtención de la información de trazabilidad Como hemos visto, el uso de conectores permite al enfoque la interacción eventualmente con cualquier lenguaje de transformación de modelos arbitrario. En particular, y considerando que nuestra propuesta, QVTrace, ha sido definida en el contexto de QVT, nos concentraremos en determinar cómo se lleva a cabo la obtención de trazas este mecanismo para transformaciones escritas para dicho lenguaje. El conector QVT desarrollado está escrito en el lenguaje de mapeos operacionales que provee QVT (Operational Mappings). Dicho lenguaje permite definir transformaciones utilizando

una aproximación imperativa o bien complementar transformaciones relacionales (escritas en lenguaje QVT Relations) con operaciones imperativas (enfoque híbrido), cuando es difícil proveer una especificación completamente declarativa de una relación. Cada relación define una clase que será instanciada para la traza entre elementos de modelo que están siendo transformados, y esta tiene un mapeo unívoco con una operación que el mapeo operacional implementa.

Bajo este esquema, la extracción de información de trazabilidad ocurre en dos pasos: en primer lugar, la transformación de modelos es ejecutada permitiendo la conversión de cualquier instancia del metamodelo de trazabilidad interno de QVT en una instancia de Trace-DSL mediante la definición de un mapeo operacional entre ambos metamodelos. El segundo paso consiste en la importación de la instancia Trace-DSL generada en el correspondiente repositorio. Aunque exigua, esta descripción nos permite observar lo siguiente:

- Al igual que con la propuesta de Jouault, esta opera a nivel de transformación. Es decir, requiere la ejecución de la transformación para la generación de las trazas.
- Permite la interacción con arbitrarios lenguajes de transformación de modelos, aunque depende fuertemente del conector que posibilita el enlace con el motor de transformación.
- En el contexto de QVT, utiliza el lenguaje Operational Mappings.
- Define un metamodelo de trazabilidad muy interesante, tipificando los distintos escenarios de traza (generación de nuevos elementos, actualización, eliminación y consulta).
- Es independiente de los modelos involucrados.
- No agrega información adicional que pudiera alterar los modelos o la lógica de la transformación.

4.3. Resumen de las principales diferencias

Luego de repasar los detalles de los dos trabajos de referencia seleccionados y analizadas las características del enfoque propuesto, contrastaremos las soluciones según los criterios definidos anteriormente.

Metamodelo de trazabilidad A lo largo de la Sección hemos detallado las principales características de los metamodelos puestos en contraste. Por un lado hemos visto un esquema de Jouault muy genérico, simple, y flexible, cuyo énfasis está puesto en el mantenimiento de la información de trazabilidad desde el punto de vista de la relación entre elementos de los modelos origen y destino. En segundo lugar, repasamos el metamodelo de Grammel, con un enfoque por escenarios que caracteriza distintos tipos de traza en base al impacto en el modelo destino. Finalmente, como una propuesta intermedia el esquema de QVTrace, el cual es más parecido al de Jouault, aunque más restrictivo respecto de los participantes de la relación de traza y cuyo eje es el significado de la relación, no solo los integrantes de la misma, esto es la manera en que un elemento del modelo origen se transforma en uno del modelo destino.

Si bien es posible seguir enumerando diferencias, cada metamodelo tiene sus fortalezas y debilidades, y las mismas surjen de la aplicación y el contexto en que han sido definidas. Claramente no existe "el" metamodelo de trazabilidad, sino distinto enfoques que enfatizan aspectos puntuales en función de la utilización y los resultados que busquen obtenerse.

Obtención de la información de trazabilidad Si el metamodelo de trazabilidad define qué información de rastreabilidad obtener, el segundo criterio de comparación considerado ha sido el "cómo" obtenerla. Como hemos visto, tenemos por un lado un mecanismo de Jouault netamente conducido por modelos, definido con las mismas herramientas del lenguaje de transformación, automatizado, implementado a nivel de transformación, que agrega información extra a la definición de dicha transformación, alterando no la lógica pero sí la legibilidad de la misma. En contraste repasamos el trabajo de Grammel, que está basado en un DSL desarrollado *ad hoc*, genérico, adaptado a un número arbitrario de lenguajes de transformación de modelos, aunque dependiente de las posibilidades y herramientas que éste pueda proveerle para la construcción de un componente de conexión fundamental, implementado a nivel de transformación. Finalmente, la propuesta de estos autores que sugieren que desde el conocimiento de la definición de la transformación y los modelos involucrados es posible identificar ciertos patrones o construcciones del lenguaje que permiten reconocer trazas de manera independiente a la transformación, caracterizando no sólo los participantes de la misma, sino también el significado o forma, que no altera de ninguna manera los modelos ni la lógica de la transformación, automatizada, implementada por fuera de las posibilidades del lenguaje de transformación de modelos, pero integrada con uno de los principales entornos de desarrollo utilizados en el marco del paradigma conducido por modelos.

5. Conclusiones y trabajos futuros

A lo largo del presente trabajo se repasaron los aspectos mas destacados del paradigma de desarrollo conducido por modelos o MDSD y se abordó el concepto de trazabilidad como característica deseable en toda transformación de modelos. En este contexto, se presentó una técnica de análisis de trazabilidad basado en variables que permite identificar en la definición de una transformación escrita en lenguaje QVT ciertos patrones que posibilitan la inferencia automática de trazas. Al margen de ser una propuesta teórica, esta idea ha sido implementada en un prototipo llamado QVTrace, el cual está diseñado como plugin de Eclipse, y pensado para que interactúe con otras herramientas de desarrollo conducido por modelos diseñadas para dicho framework.

A continuación, se detallaron las principales características de QVTrace y se realizó un análisis comparativo de éste junto a dos esquemas de similar índole: la propuesta de trazabilidad escasamente acoplada de Jouault y el framework de trazabilidad genérico de Grammel para extracción de trazas basada en *facets*. Tomando como eje de comparación el metamodelo de trazabilidad definido y

el mecanismo de obtención de trazas, hemos podido comprobar que lejos de existir "el" metamodelo, la representación de las trazas está fuertemente ligada a la implementación, tanto del mecanismo de obtensión de trazas, como a la representación de los modelos, y a la información que se desee mantener. Por el lado de los mecanismos de generación, a diferencia de los trabajos de los otros autores, la propuesta propia se independiza completamente del proceso de transformación de modelos y trabaja sobre la definición de la misma con la convicción que ésta encierra información de trazabilidad de manera implícita.

Por último, es importante mencionar posibles pasos a seguir en esta línea propuesta que comienza con el presente trabajo. En particular, sería de mucho interés poder determinar nuevas construcciones o casos de traza que permitan inferir nueva información de rastreabilidad no identificada hasta el momento, o en su defecto, en caso de encontrar el límite de los posibles tipos de trazas que la técnica permite generar, poder demostrarlo formalmente.

Referencias

1. mediniQVT. http://projects.ikv.de/qvt.
2. N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni. Model traceability. *IBM Syst. J.*, 45(3):515–526, 2006.
3. Frank Budinsky, Stephen A. Brodsky, and Ed Merks. *Eclipse Modeling Framework*. Pearson Education, 2003.
4. Nikolaos Drivalos, Dimitrios Kolovos, Richard Paige, and Kiran Fernandes. Engineering a DSL for software traceability. In Dragan Gasevic, Ralf Lammel, and Eric Van Wyk, editors, *Software Language Engineering*, volume 5452 of *Lecture Notes in Computer Science*, pages 151–167. Springer Berlin / Heidelberg, 2009.
5. Birgit Grammel and Stefan Kastenholz. A generic traceability framework for facet-based traceability data extraction in model-driven software development. In *Proceedings of the 6th ECMFA Traceability Workshop*, ECMFA-TW '10, pages 7–14, New York, NY, USA, 2010. ACM.
6. Object Management Group. *MOF 2.0 Query/Views/Transformations RFP*, omg document edition, October 2002.
7. F. Jouault and I. Kurtev. Transforming models with ATL. In *Satellite Events at the MoDELS 2005 Conference*, volume 3844 of *Lecture Notes in Computer Science*, pages 128–138, Berlin, 2005. Springer Verlag.
8. Frédéric Jouault. Loosely coupled traceability for ATL. In *Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability*, Nuremberg, Germany, 2005.
9. Anneke G. Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
10. OMG. Meta object facility (MOF) 2.0 Query/View/transformation specification version 1.0. http://www.omg.org/spec/QVT/1.0/PDF/, April 2008.
11. The Institute of Electrical and Electronics Engineers. *IEEE Standard Glossary of Software Engineering Terminology*. New York, USA, September 1990.
12. Bert Vanhooff, Stefan Van Baelen, Wouter Joosen, and Yolande Berbers. Traceability as input for model transformation. In *Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability*, Nuremberg, Germany, 2007.