Trazabilidad de Requisitos en Almacenes de Datos basada en MDA

Alejandro Maté, Juan Trujillo

Lucentia Research Group
Department of Software and Computing Systems
University of Alicante
{amate,jtrujillo}@dlsi.ua.es

Abstract. La complejidad del proceso de desarrollo de los Almacenes de Datos (AD) requiere una aproximación sistemática para tener éxito. Una aproximación ampliamente aceptada es la híbrida, en la que requisitos y fuentes de datos han de ser acomodados en un modelo híbrido. El inconveniente de esta aproximación es que se pierden las relaciones entre requisitos, elementos conceptuales y fuentes de datos, al no incluir trazabilidad explícita en el proceso. Debido a ello, la validación de requisitos se ve mermada y aumenta la complejidad de los procesos de Extracción, Transformación y Carga. En este artículo, proponemos un metamodelo de trazas para ADs y detallamos las relaciones entre los requisitos y los elementos conceptuales. Además, mostramos la implementación de reglas Query/View/Transformation para automatizar la generación de trazas, evitando así su elaboración manual.

Key words: Trazabilidad, Almacenes de Datos, Requisitos, Model Driven Architecture.

1 Introducción

Los Almacenes de Datos (AD) integran datos de fuentes heterogéneas en estructuras multidimensionales (hechos y dimensiones), para dar soporte al proceso de toma de decisiones [10, 13]. Por esta razón, el desarrollo del AD es un proceso complejo que ha de ser planificado cuidadosamente a fin de satisfacer las necesidades de los usuarios.

Una aproximación, ampliamente aceptada para el desarrollo de ADs, es la aproximación híbrida. Esta aproximación hace uso tanto de las fuentes de datos como de los requisitos de usuario [17], identificando aquellos requisitos que no pueden ser cubiertos en etapas tempranas del desarrollo. Primero, se recoge la información de ambos mundos y, a continuación, se resuelven las incompatibilidades acomodando requisitos y fuentes de datos en un único modelo. Sin embargo, esta aproximación presenta un inconveniente. Al contrario que en un desarrollo software habitual, un elemento del AD puede presentar correspondencias tanto con requisitos como con las fuentes de datos de manera simultánea. Dado que los elementos han de acomodarse, típicamente han de ser modificados, con lo que la trazabilidad por correspondencia de nombre se pierde con, al

menos, uno de los orígenes que dieron lugar al elemento. Según nuestra experiencia, estos cambios se realizan en prácticamente todos los proyectos, ya que es habitual que los requisitos no concuerden con las fuentes de datos. Por ello, se requiere un esfuerzo adicional para identificar qué partes del AD corresponden, no sólo con cada requisito, sino también con cada parte de las fuentes.

En el proceso de acomodación, las relaciones entre los distintos elementos no son almacenadas, y por tanto estas relaciones se pierden, al no incluir trazabilidad de manera explícita en el proceso de desarrollo. Esta pérdida complica la validación de requisitos [23, 25, 28], resultando difícil conocer con exactitud el estado en el desarrollo de cada requisito y tomar decisiones sobre implementaciones alternativas, si, por ejemplo, un requisito no se puede cumplir. A pesar de que la trazabilidad ha sido estudiada a fondo [1, 2, 4, 8, 9, 11, 21, 22, 25], ha sido omitida casi por completo en el desarrollo de ADs. La única mención sobre trazabilidad en este campo es la que se realiza en [16], donde sólo se menciona trazabilidad implícita por correspondencia de nombre.

En trabajos previos [15–18], hemos definido una aproximación híbrida para el desarrollo de ADs en el contexto del marco Model Driven Architecture (MDA) [19]. En nuestra aproximación, los requisitos se especifican en un Modelo Independiente de la Computación (CIM) mediante un perfil de UML [16] basado en el marco de i* [29]. A continuación, los requisitos son derivados de manera automática, reconciliados con las fuentes de datos en un modelo híbrido, y refinados a lo largo de una serie de capas (Modelo Independiente de la Plataforma (PIM) y Modelo Específico de la Plataforma (PSM)), hasta que se alcanza la implementación final, como puede verse en la figura 1.

La derivación automática se realiza utilizando transformaciones modelo a modelo, especificadas mediante reglas Query/View/Transformation (QVT) [20]. QVT es un lenguaje definido por el Object Management Group (OMG) y propuesto como estándar de transformaciones entre modelos. Sin embargo, dada nuestra experiencia en proyectos reales, a pesar de disponer de este tipo de transformaciones, la carencia de un soporte para trazabilidad imposibilita validar adecuadamente los requisitos, introduciendo un sobrecoste cuando estos cambian.

En este artículo complementamos nuestros trabajos previos con la inclusión del primer metamodelo de trazas adaptado para ADs, así como describiendo la manera de derivar de forma automática los modelos de trazas correspondientes. De esta forma, al incluir la trazabilidad, aumentamos la reutilización, el mantenimiento y la comprensión de los modelos de requisitos [23, 28], y somos capaces de analizar de manera sencilla qué requisitos han sido cubiertos y qué elementos se verán afectados por un cambio en un requisito dado.

El resto del artículo se encuentra estructurado de la siguiente manera. La sección 2 presenta trabajos relacionados sobre temas de trazabilidad. La sección 3 introduce nuestro metamodelo para la trazabilidad en ADs y describe cómo se realiza la inclusión de los modelos de trazas necesarios en nuestra aproximación. La sección 4 presenta las reglas QVT para la derivación automática de trazas, así como detalles de la implementación en la herramienta Lucentia BI, basada

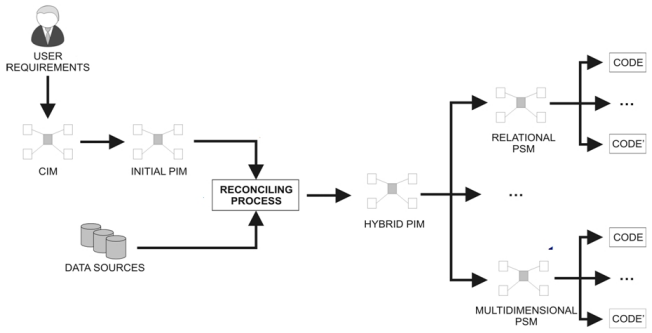

Fig. 1. Nuestra aproximación para el desarrollo de ADs

en Eclipse. Finalmente, la sección 5 describe las conclusiones y el trabajo futuro a realizar en esta área.

2 Trabajos Relacionados

En esta sección discutiremos sobre las propuestas existentes en cuanto a trazabilidad en otros campos, sus beneficios e inconvenientes, y también describiremos el estado actual dentro del campo de los ADs. En la actualidad, la trazabilidad se estudia tanto en el campo de la Ingeniería de Requisitos (IR), como en el de Desarrollo Dirigido por Modelos (MDD).

La mayor parte del trabajo realizado hasta ahora ha sido en el campo de IR [1–3, 8, 9, 22, 25, 30]. Algunos autores [8, 28] consideran más compleja la trazabilidad previa a la especificación de requisitos (pre-RS), dado que tiene que lidiar con artefactos escritos en lenguaje natural y distintos puntos de vista. Por contra, la trazabilidad posterior a la especificación de requisitos (post-RS) es considerada más simple, ya que los requisitos se encuentran previamente modelados.

Los principales beneficios de la trazabilidad [2, 22, 23] son: (i) mejora el análisis del impacto de cambios, (ii) mejora la comprensión de los distintos modelos (permitiendo identificar qué partes de la implementación satisfacen cada requisito [2]), y (iii) aumento de la reusabilidad y del mantenimiento, dado que el conjunto de elementos relacionados con un requisito es conocido gracias las trazas. De esta forma, dichos beneficios ayudan a reducir los costes asociados con el proyecto [22, 23].

Los principales inconvenientes mencionados sobre la trazabilidad son (i) la no existencia de un metamodelo o definición estándar, (ii) la necesidad de crear y almacenar manualmente las trazas, y (iii) la visión de la trazabilidad en sí misma como una sobrecarga, hasta que resulta necesaria, a pesar de que sus beneficios han sido comprobados [23]. Por ello, se crea una situación que complica la aplicación exitosa de la trazabilidad en los proyectos.

De cara a solucionar el primer inconveniente, en [25] se propuso una clasificación de ocho categorías de trazas. Por otra parte, los últimos dos inconve-

nientes pueden solventarse automatizando la generación y almacenamiento de trazas. Sin embargo, dado que en el campo de IR la trazabilidad está centrada en pre-RS, se requiere identificar trazas en documentos escritos típicamente en lenguaje natural. Como resultado, se generan modelos de trazas que han de ser supervisados manualmente, presentando un gran número de trazas que dificultan la comprensión y visualización [27].

Por otro lado, en el campo MDD se suele utilizar el marco MDA [1, 4, 11, 21, 27]. El proceso de derivación automática comienza en la capa CIM, donde se especifican los requisitos mediante modelos, típicamente intencionales. [6, 14, 16, 7, 24, 31]. En este sentido, la investigación en trazabilidad en el campo MDD se centra principalmente en post-RS, permitiendo que la automatización de la trazabilidad sea más sencilla y menos propensa a errores, ya que todos los elementos se encuentran modelados. No obstante, a pesar de ser más restrictivo, este campo tampoco dispone de un estándar para los conceptos de trazabilidad, existiendo principalmente dos definiciones distintas. La que utilizarmeos en este artículo es la presentada en [21], dónde se define la trazabilidad como "[...] the ability to chronologically interrelate uniquely identifiable entities in a way that matters. [...] [It] refers to the capability for tracing artifacts along a set of chained [manual or automated] operations."

En el campo de los ADs, como indicamos previamente, no se menciona la inclusión de trazabilidad en el proceso de desarrollo, a pesar de que las aproximaciones existentes podrían beneficiarse de ello. Dichas aproximaciones están basadas en transformaciones entre modelos a través de distintas capas, bien siguiendo el marco MDA [17] o un conjunto similar de capas [26]. Actualmente, cada vez que se realiza algún cambio, la trazabilidad, tal y como se define en [21], se pierde, dado que los elementos se encuentran asociados por correspondencia de nombre. A su vez, perdemos todos los beneficios proporcionados por la trazabilidad, que podrían obtenerse en el desarrollo de ADs con un coste reducido, dado que las trazas son susceptibles de generarse de manera automática. Además, la inclusión de la trazabilidad podría proporcionar soporte para automatizar procesos de análisis sobre los modelos, como las métricas de calidad presentadas en [26], permitiendo su cálculo automático. De esta forma, se incrementaría la calidad de la implementación final a la vez que se disminuyen los costes.

Dada la idiosincrasia de los ADs, necesitaremos diferenciar entre (i) trazas provenientes de los requisitos (utilizadas para validación de requisitos y análisis de impacto), (ii) trazas provenientes de las fuentes de datos (para realizar consultas automáticas y derivar versiones iniciales de procesos de Extracción, Transformación y Carga) y (iii) trazas de enlace entre elementos multidimensionales presentes en los modelos conceptuales, de acuerdo a sus relaciones particulares[15].

3 Una Aproximación de Trazabilidad y un Metamodelo de Trazas para Almacenes de Datos

Como indicamos previamente, si queremos realizar operaciones automáticas con las trazas, debemos ser capaces de identificar el significado de cada una de el-

las. Por ello, necesitamos elaborar un conjunto de tipos de trazas que definan la semántica de las relaciones entre los distintos elementos. En esta sección, introduciremos los metamodelos de trazas propuestos en el campo MDD junto a nuestro metamodelo propuesto para ADs.

3.1 Metamodelos de Trazabilidad basados en el marco de la Arquitectura Dirigida por Modelos

Nuestra aproximación de trazabilidad está basada en el marco de trazas descrito por el OMG, incluído en el marco de MDA [19].

El metamodelo, presentado en la figura 2, está compuesto por un *Transformation record*, que representa la transformación que produce las trazas. Dicho *Transformation record* contiene el conjunto de trazas generadas, así como metadatos asociados, como pueden ser los parámetros utilizados por la transformación cuando se ejecutó. Para cada traza creada, existen de 0 a N elementos de los modelos enlazados por ella. Como en el caso anterior, la traza puede tener asociados metadatos, tales como qué regla de la transformación la creó.

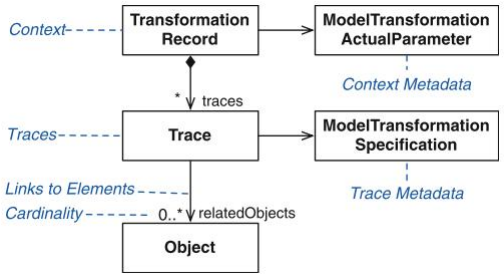

Fig. 2. Metamodelo para la trazabilidad en MDA

De acuerdo a esta propuesta, se creó el núcleo del ATLAS Model Weaver (AMW) [5], utilizado para enlazar elementos entre distintos modelos. Este núcleo constituye la base para la creación de nuestro metamodelo extendido para trazabilidad.

3.2 Metamodelo Propuesto

Nuestro metamodelo propuesto para trazabilidad extiende el metamodelo de AMW incluyendo los tipos necesarios para capturar la semántica utilizada en las relaciones presentes en ADs. El resultado de esta extensión se puede observar en la figura 3, resaltado con un recuadro rojo. Para más información acerca de las clases base de *weaving* (clases que comienzan con W), consultar [5].

En este metamodelo, un *TraceModel* tiene un conjunto de modelos enlazados (*wovenModels*) por el modelo de trazas. Cada uno de estos modelos enlazados

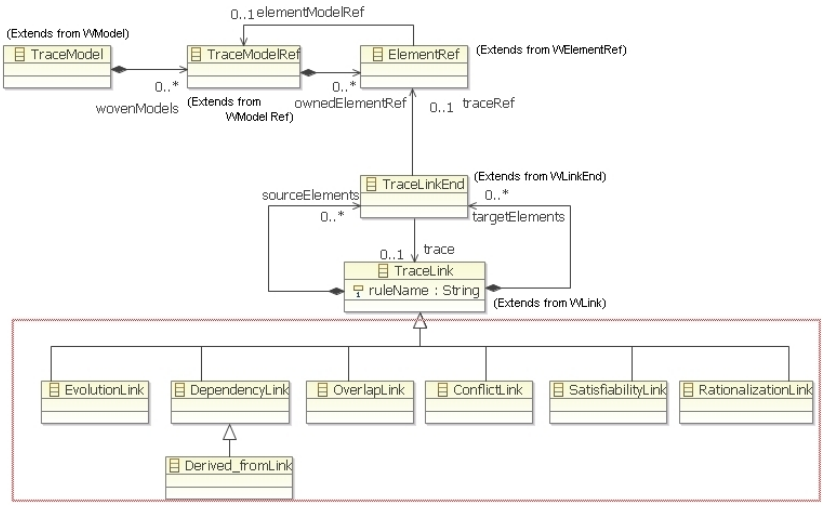

Fig. 3. Metamodelo de AMW extendido con enlaces semanticos para ADs

tiene una lista de referencias (*ElementRef*), que identifican qué elementos son enlazados por las trazas. A su vez, el modelo de trazas tiene un conjunto de *TraceLinks*, que definen las relaciones entre los elementos presentes en los modelos enlazados. Cada traza tiene un conjunto de elementos origen *sourceElements*, que actuaron como fuente de la derivación automática, así como un conjunto de elementos destino *targetElements*, que se crearon como resultado de la derivación. Además, una traza puede tener un padre, así como un conjunto de trazas hijas (propiedad heredada del modelo de *weaving* de AMW). Este aspecto resulta importante, ya que permite agrupar las trazas formando jerarquías, y creando distintos niveles de detalle en los modelos de trazas. De esta forma, el modelo de trazas puede contener y visualizar cientos de trazas de manera escalable. Los elementos enlazados por las trazas son representados por *TraceLinkEnds*, que incluyen la referencia al identificador del elemento correspondiente.

De cara a incluir significado semántico a las trazas en el metamodelo, extendemos el elemento *TraceLink*, alineando los tipos posibles con la clasificación realizada en [25]. Pese a que podríamos haber utilizado un número más reducido de enlaces, dado que en nuestro caso *Overlap* y *Conflict* resultan próximos, hemos decidido, de cara a facilitar la interoperabilidad y estandarización, incluir los elementos respetando su diferenciación. No obstante, en nuestro caso, cada traza incluirá únicamente un único sentido semántico (dado que no introducimos roles ya que se tratan a nivel CIM). Por ello, la definición de cada tipo de traza utilizado es la siguiente:

– *Satisfiability* y *Dependency* se utilizan para dar soporte a trazabilidad vertical (entre capas distintas). En el primer caso, las trazas de este tipo recogerán las relaciones que provienen de los requisitos (CIM) a los elementos concep-

tuales (PIM). En el segundo caso, utilizaremos una especialización del tipo *Dependency*, *Derived_from*, a fin de identificar las trazas provenientes de las fuentes de datos hacia los elementos multidimensionales a nivel PIM.

- *Evolution* se incluyen para dar soporte a la trazabilidad horizontal, la cual gestiona cambios realizados sobre elementos en la misma capa (e.g. de PIM a PIM).
- *Overlap* y *Conflict* se utilizan para resolver conflictos derivados de la aproximación híbrida, cuando un elemento proviene tanto de los requisitos como de las fuentes de datos presentando formas distintas. En este caso, el diseñador decidirá qué elemento será derivado como solución correcta al conflicto.
- *Rationalization* se incluyen como mecanismo para permitir al usuario realizar sus propias anotaciones o cambios en el modelo, así como describir las decisiones tomadas.

Una vez que hemos presentado nuestro metamodelo de trazas, necesitamos describir qué modelos se crearán y qué información almacenarán. Con el fin de incluir soporte para trazabilidad en nuestra aproximación, [16, 17], incluiremos los modelos trazas que se muestran en la figura figura 4.

El primero de los modelos, *CIM2PIMTrace*, está dedicado a dar soporte a tareas relacionadas con los requisitos (como validación de requisitos y de transformaciones o cálculo de medidas basadas en las trazas), y hace explíticas las relaciones[16, 17] existentes entre los elementos del nivel CIM y aquellos del nivel PIM. Este es un caso de trazabilidad vertical (el modelo origen y el destino se encuentran en capas distintas), en el que todas las relaciones son perfectamente conocidas, dado que se generan de manera automática. Por ello, sólo necesitamos crear los elementos correspondientes al modelo de trazas de manera simultánea a la creación del modelo destino, mediante una transformación modelo a modelo. Este modelo almacenará principalmente trazas de tipo *Satisfiability*.

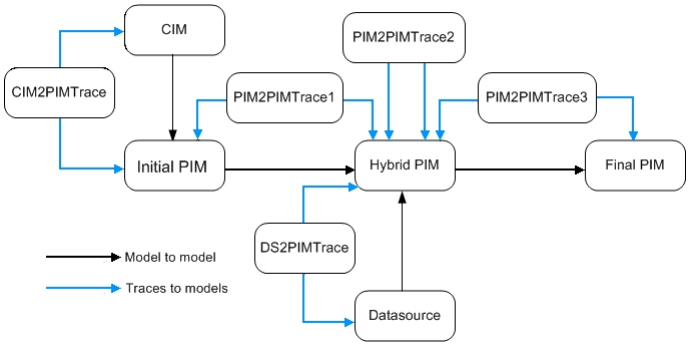

Fig. 4. Inclusión de modelos de trazas en el desarrollo de ADs

El segundo modelo, *DS2PIMTrace*, con una filosofía similar al primero, da soporte a operaciones relacionadas con las fuentes de datos. Este modelo almacena las trazas correspondientes a las relaciones entre las fuentes de datos, representadas en la capa PSM, y el PIM híbrido. El PIM híbrido es el resultado de la transformación que utiliza como entrada el primer modelo PIM ("PIM inicial") y las fuentes de datos. Este PIM híbrido es trazado tanto al PIM inicial, de cara a permitir trazar los requisitos a lo largo de todo el proceso, como a las fuentes de datos, de cara a mantener las tablas y atributos fuente que participan en la creación del AD. Al igual que en el caso anterior, la trazabilidad de fuentes a PIM híbrido es trazabilidad de tipo vertical, almacenándose en este caso trazas de tipo *Derived_from* en el modelo *DS2PIMTrace*.

Los últimos tres modelos de trazas, *PIM2PIMTrace*, *PIM2PIMTrace2* y *PIM2PIMTrace3*, se encargan de enlazar horizontalmente los elementos, a lo largo de los sucesivos refinamientos que sufren a nivel PIM. El primero de los modelos contiene únicamente trazas de tipo *Evolution*, y su función es la de conectar los elementos que provienen de los requisitos con el PIM híbrido. Por otra parte, el modelo *PIM2PIMTrace2* representa las relaciones intra-modelo entre elementos provenientes de los requisitos y aquellos provenientes de las fuentes de datos. Dado que estas relaciones no son propias del metamodelo de ADs, son almacenadas en forma de trazas, mediante los tipos *Overlap* y *Conflict*. *Overlap* define una situación en la que ambos elementos (provenientes de requisitos y fuentes de datos) definen el mismo concepto de manera compatible, estando en desacuerdo típicamente en el nombre. Por otro lado, *Conflict* define una situación en la que ambos elementos definen conceptos distintos. De esta forma, el modelo *PIM2PIMTrace2* es utilizado como acomodador de los elementos, y sus trazas han de ser definidas de manera manual, ya que típicamente no existe información que guíe el proceso de correspondencia. Finalmente, el modelo *PIM2PIMTrace3*, recoge los elementos del PIM híbrido que son utilizados para derivar el modelo PIM final. Este modelo se elabora con información tanto del modelo híbrido como de las relaciones intra-modelo almacenadas previamente, y contiene principalmente trazas de tipo *Evolution*.

Una vez que hemos definido el metamodelo de trazas, y los modelos correspondientes, hemos de definir de manera formal la derivación automática de las trazas mediante reglas QVT [20].

4 Implementación de la Trazabilidad CIM a PIM en la Herramienta Lucentia BI

En esta sección mostraremos la definición de las transformaciones necesarias para la generación trazas de CIM a PIM, de manera simultánea a la generación de elementos del AD, de forma que puedan ser consultadas y actualizadas a lo largo del tiempo. Además, también mostraremos cómo se ha realizado la implementación en la herramienta Lucentia BI, basada en Eclipse.

De acuerdo a nuestra propuesta para el desarrollo de ADs [16, 17], utilizamos una aproximación híbrida para derivar elementos en un PIM inicial a partir de

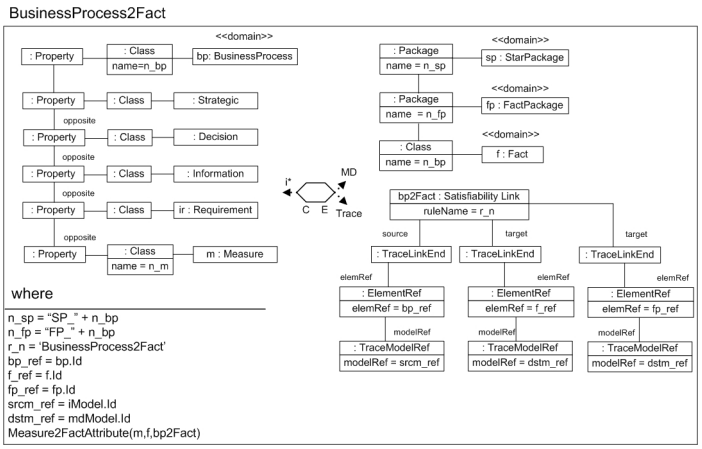

Fig. 5. QVT que deriva un hecho y sus trazas a partir de un proceso de negocio

los requisitos, utilizando reglas QVT. Las reglas QVT especifican una transformación comprobando si se cumple un patrón definido en el modelo origen. Si se cumple dicho patrón, la regla QVT genera los elementos correspondientes al patrón especificado en el metamodelo destino. Dichas reglas QVT deben obtener a nivel PIM cuatro elementos básicos: hechos, atributos del hecho o medidas, dimensiones y niveles de jerarquía. La primera QVT, que mostramos en la figura 5, se encarga de la generación del hecho y de sus correspondientes estructuras a nivel PIM, a partir de un *BusinessProcess* en el modelo CIM.

En la parte izquierda de la transformación se encuentra el metamodelo origen, nuestro perfil de i* para modelado requisitos en ADs. El proceso de negocio se encuentra modelado junto a sus objetivos relacionados, representados mediante objetivos estratégicos, decisionales y de información. Estos objetivos modelan la lógica del negocio, a partir de la cual obtenemos los requisitos de información, que, a su vez, son descompuestos en medidas (indicadores de desempeño) y contextos de análisis. En la parte derecha de la transformación se encuentran los metamodelos destino. Por un lado, tenemos nuestro perfil multidimensional, compuesto por el hecho (centro del análisis y relacionado con el proceso de negocio) y sus estructuras de empaquetamiento correspondientes (*FactPackage* y *StarPackage*). Por otro lado, tenemos el metamodelo de trazas propuesto, formado en este caso por la traza que hace explícita la relación entre el proceso de negocio y el hecho. Dicha traza tiene un elemento origen, el proceso de negocio, y tres elementos destino, uno de los cuales, referente al *StarPackage*, se ha omitido.

La "C" en el centro de la figura significa que el modelo origen es comprobado únicamente, mientras que la "E" indica que los modelos destino son forzados a cumplir con la especificación. Esto significa que, cada vez que se encuentra el patrón descrito en el modelo origen, los patrones objetivo son generados en los modelos objetivo. Finalmente, la claúsula *Where* modela las operaciones adi-

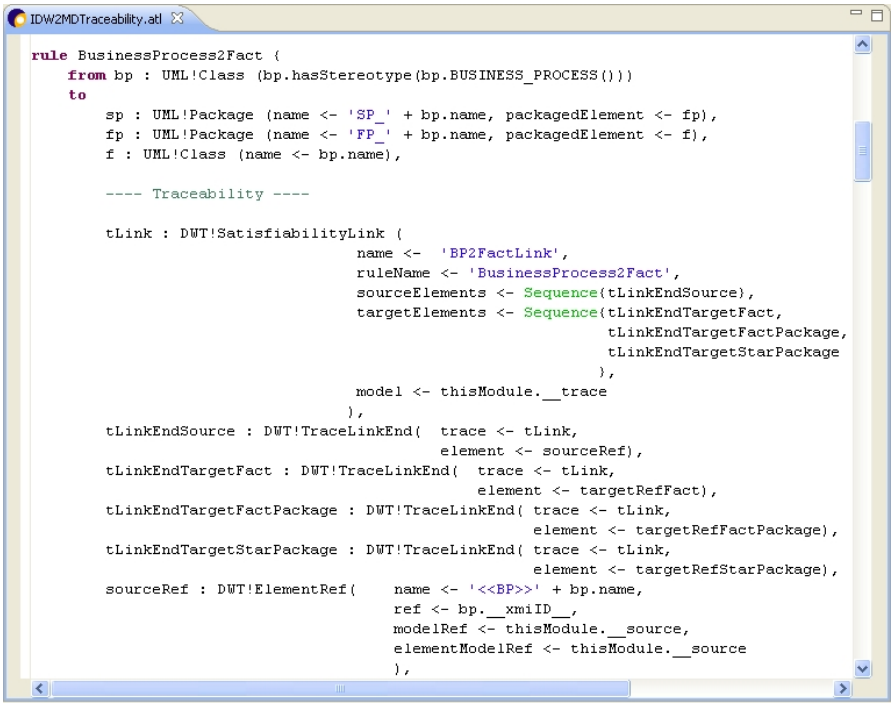

Fig. 6. Implementación de la derivación del hecho y su traza mediante código ATL

cionales que se realizan sobre los elementos de la QVT. En este caso, se especifican las asignaciones de valores pendientes y una llamada a otra regla para transformar la medida del proceso de negocio en un *FactAttribute* del hecho.

Esta regla QVT está adaptada para resultar próxima a la implementación. En la figura 6, se puede ver la implementación de esta regla en lenguaje ATL [12]. De manera análoga a la regla QVT, la claúsula *from* de la regla ATL especifica los elementos origen relevantes, en este caso el proceso de negocio, y cláusula *to* los elementos destino que son generados: el hecho, sus paquetes y la traza asociada. Finalmente, la claúsula *do* (omitida) realiza las operaciones adicionales necesarias, como la llamada a la regla *Measure2FactAttribute*.

La transformación *Measure2FactAttribute*, mostrada en la figura 7, es similar a la anterior. En este caso, una medida "m" del proceso de negocio se transforma en un atributo del hecho "fa". Uno de los aspectos más relevantes de esta transformación es que, en este caso, se incluye el nuevo elemento dentro del hecho previamente generado. Además, aprovechando esta lógica, la traza de dicho atributo también puede incluirse dentro de la traza correspondiente, obteniendo un agrupamiento próximo al modelo que se genera, y simplificando la visualización posterior del modelo de trazas.

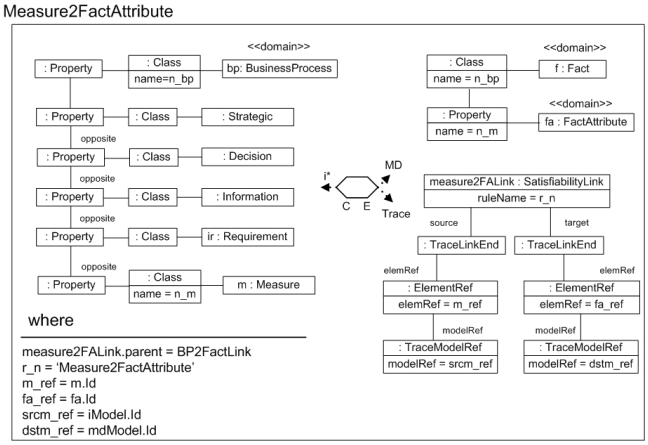

Fig. 7. QVT para transformar una medida en un atributo del hecho correspondiente y generar las traza asociada

Las dos transformaciones (una omitida) restantes se encargan de la generación de dimensiones y niveles de jerarquía, a partir de contextos presentes a nivel CIM. La generación de una dimensión "d", su paquete, y su nivel base "b", se realiza al encontrar un contexto "c" que constituye la raíz de un grupo de contextos. En la figura 8 se muestra la transformación correspondiente a este caso.

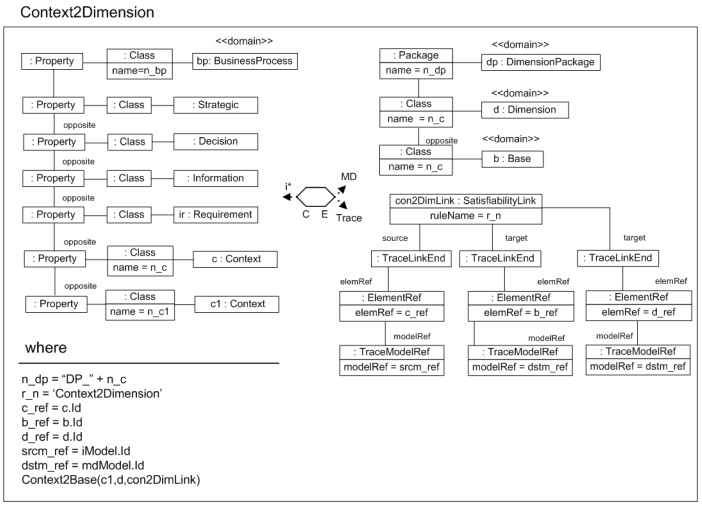

Fig. 8. QVT para transformar un contexto en una dimensión

En ella, de manera análoga al caso del hecho, se genera una traza que enlaza como origen el contexto "c" con los distintos elementos del nivel PIM generados. El caso alternativo, omitido por razones de espacio, se da cuando un contexto "c1" no es la raíz del grupo de contextos. En dicho caso el contexto se transforma en una base "b1" y, una vez generado, se enlaza mediante una asociación con la base previa en la jerarquía. Al igual que en el caso de la medida, de cara a mejorar la visualización, las trazas correspondientes a niveles superiores de una jerarquía son incluídas dentro de la traza que enlaza el contexto raíz con la dimensión en sí. A modo de ejemplo, al ejecutar las transformaciones sobre un modelo CIM referente a una universidad, obtenemos como resultado el modelo de trazas que se muestra en la figura 9.

En este modelo de trazas podemos observar las distintas trazas de tipo *Satisfiability Link* que relacionan los elementos del nivel CIM con aquellos del nivel PIM. Además, también podemos observar la existencia de trazas anidadas, correspondientes a elementos incluidos, bien dentro del hecho (atributos del hecho), o bien dentro de una dimensión (bases), las cuales pueden desplegarse en caso de ser necesario ver el detalle. Los elementos referenciados en el modelo de trazas aparecen nombrados incluyendo su tipo, no obstante esto es sólo una cuestión de nomenclatura para facilitar la identificación por parte del usuario, y no influyen de manera alguna en el propio modelo de trazas. Finalmente, podemos observar las referencias a los modelos enlazados, incluyendo el listado de elementos correspondientes a cada uno. A pesar de que este modelo presenta el resultado de la generación de trazas a partir de un CIM basado en objetivos, las trazas podrían

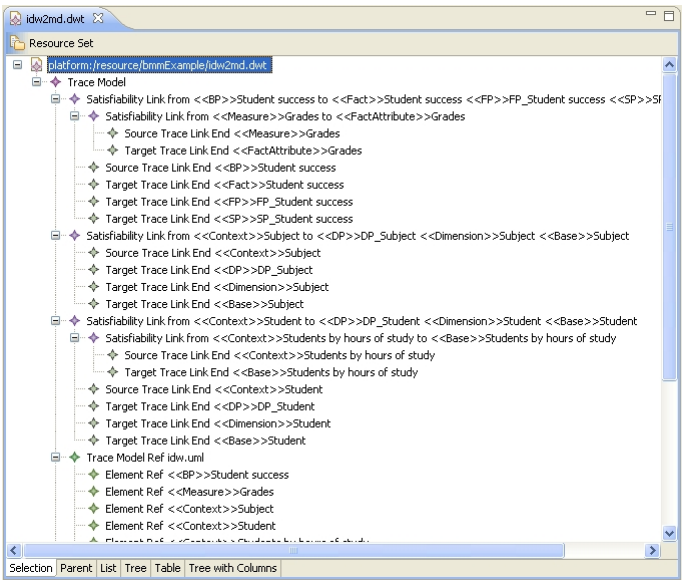

Fig. 9. Modelo de trazas entre CIM y PIM generado automáticamente

ser utilizadas para enlazar cualquier tipo de elemento en un CIM distinto de otra propuesta, siempre que se pudiesen identificar de manera única.

5 Conclusiones y Trabajo Futuro

En este artículo hemos presentado el primer metamodelo de trazas adaptado para el desarrollo de ADs, e integrado en el marco MDA. Hemos mostrado los distintos tipos semánticos utilizados así como los modelos de trazas necesarios para dar soporte de trazabilidad en nuestro proceso de desarrollo. Además, nos hemos focalizado en las relaciones entre CIM y PIM, describiendo las transformaciones QVT para derivar automáticamente los modelos de trazas correspondientes. El gran beneficio de nuestra propuesta es la mejora en validación de requisitos y la facilidad para identificar el alcance de cada elemento. De esta forma, resulta sencillo realizar análisis de impacto y regenerar las partes afectadas.

Nuestros planes para el futuro inmediato son desarrollar un nuevo conjunto de reglas QVT para explorar las relaciones entre elementos en el nivel PIM y aquellos en el nivel PSM, y analizar el potencial de la información almacenada en las trazas para dar soporte a análisis automático. Además, tenemos pensado desarrollar un marco reactivo para trazabilidad de cara a permitir que el mantenimiento sea lo más automático posible.

Agradecimientos. Este trabajo ha sido parcialmente financiado por los proyectos MESOLAP (TIN2010-14860) y SERENIDAD (PEII-11-0327-7035) del Ministerio de Educación de España y la Junta de Comunidades de Castilla La Mancha. Alejandro Maté está financiado por la Generalitat Valenciana mediante una beca Vali+D (ACIF/2010/298).

References

1. Aizenbud-Reshef, N., Nolan, B., Rubin, J., Shaham-Gafni, Y.: Model traceability. IBM Systems Journal 45(3), 515–526 (2006)
2. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.: Recovering traceability links between code and documentation. IEEE Transactions on Software Engineering 28(10), 970–983 (2002)
3. Arkley, P., Mason, P., Riddle, S.: Position paper: Enabling traceability. In: Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering. pp. 61–65. Edinburgh, Scotland (2002)
4. Barbero, M., Del Fabro, M., Bézivin, J.: Traceability and provenance issues in global model management. In: ECMDA-TW. pp. 47–56 (2007)
5. Del Fabro, M., Bézivin, J., Valduriez, P.: Weaving Models with the Eclipse AMW plugin. In: Eclipse Modeling Symposium, Eclipse Summit Europe 2006. Esslingen, Germany (2006)
6. Franch, X.: Incorporating Modules into the i* Framework. In: CAiSE, LNCS, vol. 6051, pp. 439–454. Springer Berlin (2010)
7. Giorgini, P., Rizzi, S., Garzetti, M.: GRAnD: A goal-oriented approach to requirement analysis in data warehouses. DSS 45(1), 4 – 21 (2008)

8. Gotel, O., Finkelstein, A.: An analysis of the requirements traceability problem. In: ICRE. pp. 94–101. IEEE (1994)
9. Gotel, O., Morris, S.: Macro-level Traceability Via Media Transformations. In: Requirements Engineering: Foundation for Software Quality, LNCS, vol. 5025, pp. 129–134. Springer Berlin (2008)
10. Inmon, W.: Building the data warehouse. Wiley-India (2009)
11. Jouault, F.: Loosely coupled traceability for atl. In: ECMDA-TW. pp. 29–37. Nuremberg, Germany (2005)
12. Jouault, F., Kurtev, I.: Transforming models with ATL. In: Satellite Events at the MoDELS 2005 Conference. pp. 128–138. Springer (2006)
13. Kimball, R.: The data warehouse toolkit. Wiley-India (2009)
14. Kolp, M., Giorgini, P., Mylopoulos, J.: Organizational Patterns for Early Requirements Analysis. In: CAiSE, LNCS, vol. 2681, pp. 617–633. Springer Berlin (2003)
15. Luján-Mora, S., Trujillo, J., Song, I.Y.: A UML profile for multidimensional modeling in data warehouses. DKE 59(3), 725–769 (2006)
16. Mazón, J.N., Pardillo, J., Trujillo, J.: A model-driven goal-oriented requirement engineering approach for data warehouses. In: Proceedings of the 2007 conference on Advances in conceptual modeling: foundations and applications. pp. 255–264. ER'07, Springer-Verlag (2007)
17. Mazón, J.N., Trujillo, J.: An MDA approach for the development of data warehouses. DSS 45(1), 41–58 (2008)
18. Mazón, J.N., Trujillo, J., Lechtenbörger, J.: Reconciling requirement-driven data warehouses with data sources via multidimensional normal forms. DKE 63(3), 725–751 (2007)
19. OMG: A Proposal for an MDA Foundation Model (2005)
20. OMG: The Meta-Object Facility 2.0 Query/View/Transformation. Final Adopted Specification (2005)
21. Paige, R., Olsen, G., Kolovos, D., Zschaler, S., Power, C.: Building model-driven engineering traceability classifications. ECMDA-TW pp. 49–58 (2008)
22. Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. IEEE Transactions on Software Engineering 27(1), 58–93 (2001)
23. Ramesh, B., Stubbs, C., Powers, T., Edwards, M.: Requirements traceability: Theory and practice. Annals of software engineering 3(1), 397–415 (1997)
24. Samia Kaabi, R., Souveyet, C., Rolland, C.: Eliciting service composition in a goal driven manner. In: ICSOC. pp. 308–315 (2004)
25. Spanoudakis, G., Zisman, A.: Software traceability: a roadmap. Handbook of Software Engineering and Knowledge Engineering (2005)
26. Vassiliadis, P.: Data Warehouse Modeling and Quality Issues. Ph.D. thesis, Athens (2000)
27. Walderhaug, S., Stav, E., Johansen, U., Olsen, G.: Traceability in Model-Driven Software Development. Designing Software-Intensive Systems: Methods and Principle pp. 133–159 (2008)
28. Winkler, S., von Pilgrim, J.: A survey of traceability in requirements engineering and model-driven development. Software and Systems Modeling 9, 529–565 (2010)
29. Yu, E.S.K.: Modelling strategic relationships for process reengineering. Ph.D. thesis, Toronto, Ont., Canada, Canada (1995)
30. Yu, Y., Jurjens, J., Mylopoulos, J.: Traceability for the maintenance of secure software. In: ICSM 2008. pp. 297–306. IEEE (2008)
31. Yu, Y., Niu, N., Gonzalez-Baixauli, B., Candillon, W., Mylopoulos, J., Easterbrook, S., do Leite, J., Vanwormhoudt, G.: Tracing and validating goal aspects. In: RE'07. pp. 53–56. IEEE (2007)