Universidad Rey Juan Carlos

Escuela Técnica Superior de Ingeniería Informática

Departamento de Lenguajes y Sistemas Informáticos II

Incorporando la Gestión de la Trazabilidad en un entorno de Desarrollo de Transformaciones de Modelos Dirigido por Modelos

Autor: Álvaro Jiménez Rielo

Director de Tesis: Juan Manuel Vara Mesa

Co-Directora de Tesis: Verónica Andrea Bollati

Móstoles, Marzo 2012

Universidad Rey Juan Carlos

El Dr. D. Juan Manuel Vara Mesa, Profesor de Universidad del Departamento de Lenguajes y Sistemas Informáticos II de la Universidad Rey Juan Carlos de Madrid y la Dra. Dª Verónica Andrea Bollati, Profesora de Universidad del Departamento de Lenguajes y Sistemas Informáticos II de la Universidad Rey Juan Carlos de Madrid, director y codirectora, respectivamente, de la Tesis Doctoral: "Incorporando la Gestión de la Trazabilidad en un entorno de Desarrollo de Transformaciones de Modelos Dirigido por Modelos" realizada por el doctorando D. Álvaro Jiménez Rielo,

HACEN CONSTAR QUE:

Esta tesis doctoral reúne los requisitos para su defensa y aprobación

En Madrid, a 5 de Marzo de 2012

Fdo.: Juan Manuel Vara Mesa Fdo: Verónica Andrea Bollati

Dadme un punto de apoyo y moveré el mundo.

Arquímedes

Si todo lo que tiene es un martillo,
todo lo que vea le parecerá un clavo.

Observación de Baruch (Leyes de Murphy)

Resumen

La IEEE define la trazabilidad como: "El grado de relación que puede establecerse entre dos o más productos de un proceso de desarrollo, especialmente productos que tienen relaciones predecesor-sucesor o maestro-subordinado con otro producto."

Una correcta gestión de la trazabilidad permitiría conocer cómo evolucionan los elementos del sistema a lo largo del proceso de desarrollo y cómo se relacionan los diferentes elementos entre sí. Este tipo de información puede utilizarse en diferentes actividades, como el análisis del impacto, la toma de decisiones de diseño y en general, cualquier actividad relacionada con el mantenimiento del sistema. Así, sería deseable que cualquier metodología de desarrollo software proporcionara un soporte adecuado para la gestión de la trazabilidad. Sin embargo, aunque esta necesidad ha sido reconocida históricamente, lo cierto es que hasta la fecha, son pocas las propuestas metodológicas y tecnológicas que la soportan.

Por otro lado, la Ingeniería Dirigida por Modelos (MDE, Model-Driven Engineering) es el último paso en la tendencia a elevar el nivel de abstracción al que se diseña y construye el software: del ensamblador pasamos a los lenguajes estructurados, que dieron lugar a la orientación a objetos, etc. Los principios fundamentales de la MDE son potenciar el rol de los modelos a lo largo de todo el ciclo de vida y potenciar el nivel de automatización en el proceso de desarrollo. La pieza que une estos dos principios son las transformaciones de modelos. En general, las transformaciones se utilizan para ir refinando la especificación del sistema, recogida en un conjunto de modelos de alto nivel, hasta obtener un nivel de detalle suficiente para abordar la generación automática del código que implementa el sistema. No obstante, también pueden utilizarse para cualquier otra tarea relacionada con el procesamiento de modelos, como la validación, o el merging.

Así, el papel clave que juegan las transformaciones a lo largo del proceso de desarrollo proporciona un nuevo escenario en el que abordar la gestión de la trazabilidad de una forma mucho más eficiente y completa. La idea de fondo es simple e intuitiva: el conjunto de reglas que componen una transformación representan las relaciones que deben darse entre los elementos de los modelos origen y los elementos de los modelos destino implicados en dicha transformación, por tanto, se pueden utilizar esas relaciones para identificar y crear las trazas entre los diferentes elementos del sistema, que habrán sido

producidos como resultado de ejecutar diferentes transformaciones a lo largo del proceso de desarrollo.

Por otro lado, dado que las transformaciones son otro artefacto software, parece lógico aplicar los principios de la MDE a su desarrollo, es decir, concebirlas como un producto que podemos obtener a partir de refinamientos sucesivos desde una especificación de alto nivel. En este caso, dicha especificación no será más que el conjunto de relaciones que deberían darse entre los elementos de los modelos de origen y destino.

Combinando estas ideas, la presente tesis doctoral aborda la construcción de un entorno de desarrollo de transformaciones de modelos dirigido por modelos que incorpora la generación de trazas. Dicho entorno parte de una especificación de alto nivel de las relaciones entre los elementos de los modelos de origen y de los modelos de destino, que es refinada automáticamente hasta llegar a un modelo de transformación que puede ser serializado en el código que implementa la transformación. La ejecución de la transformación generada producirá, además del modelo/s destino, el modelo de trazas que recoge las relaciones entre los artefactos origen y los artefactos destino.

Agradecimientos

En primer lugar quiero agradecer a mis directores de tesis, Juancho y Vero, toda la confianza que habéis depositado en mí desde los primeros momentos, mucho antes del comienzo de esta tesis. Desde que comencé a trabajar con vosotros, siempre habéis apostado por mí y me he sentido muy valorado tanto personal como profesionalmente. Si no hubiera sido por eso, creo que en algún momento hubiera tirado la toalla. Este agradecimiento también quiero extenderlo a Esperanza, que de la misma forma, siempre me ha mostrado su confianza en mí. Y gracias a los tres por la presión y apretarme las tuercas en ciertos momentos, sino no sé cuándo hubiera acabado esto.

También quiero dar las gracias al resto de mis compañeros del Grupo Kybele. De todos ellos, quisiera mencionar de forma especial a varias personas: Valeria, tú fuiste quién me contrató por primera vez como becario cuando todavía estaba en 3º y sin aquel "adelante", dudo que hoy estuviera pensando en acabar esta tesis. También quiero agradecerte que estuvieras presente en mis tribunales de PFC y TFM y los sabios comentarios que me diste en ambas ocasiones. A Jenifer, por toda la ayuda prestada a lo largo de toda la realización de esta tesis y sus maravillosos diseños para los iconos de la herramienta. A Iván, Feliu y David, todos futuros doctores, animo chicos! Cuando os queráis dar cuenta, estaréis como yo: escribiendo el final de vuestras tesis e intentando no olvidaros de nadie en los agradecimientos. Iván, gracias por todas esas charlas sobre trazabilidad que tanto me han ayudado a levantar la cabeza del papel y ver la realidad.

A mis amigos de toda la vida, esos chicos del "Amigos del Grano de Café": Raúl, Diego, Samu, Juez, Tena, Rubén, Héctor, Alex, Adrián (para mí siempre serás Adri) y los jeques: Jesús y Fran (para ti tendría que escribir una tesis entera, hermano! Que desde los 6 años, fíjate si hay cosas que agradecer). La mayoría siempre hemos estado en momentos importantes de la vida del resto: desde los primeros pelos de la barba y las primeras noches de fiesta hasta prepararnos selectividad y entrar en la universidad o conducir por primera vez; y ahora algunos hasta con casa propia…; así que por supuesto, no podíais faltarme en este momento importante en mi vida.

A mis padres…por todo. Por enseñarme todas esas cosas de la vida que no se enseñan en los colegios o en las universidades: gracias por enseñarme a valorar lo gratificante que es conseguir las cosas por uno mismo o que mis éxitos y fracasos en la vida, los disfrutaré o sufriré yo mismo. Toda la educación que he recibido de vosotros ha hecho de mí la persona que soy hoy. De forma individual,

a mi padre, Antonio, quiero agradecerle todo el esfuerzo que ha hecho a lo largo de estos años para intentar darme TODO lo que he necesitado e incluso, a veces cosas que ni siquiera necesitaba pero venían muy bien (aunque no lo creas, lo valoro mucho más de lo que piensas) así como su papel de "poli malo" que no me dejaba salirme del camino correcto. A mi madre, Carmen, por ser mi Pepito Grillo, la voz de mi conciencia. Siempre has sabido estar en tu sitio, darme los consejos adecuados y convencerme como nadie de lo equivocado que podía llegar a estar en algunos momentos. Definitivamente, solo puedo decir cosas buenas de tí.

A mi hermana, Mª Carmen, por ser en muchos momentos mi segunda madre y en otros muchos, mi mejor consejera y mi mejor amiga (¿recuerdas los sábados por la mañana haciendo el idiota por casa? Que grandes momentos!). De forma especial quiero dedicarle esta tesis a mi sobrina Andrea, que aunque quizás todavía no sepa valorar el esfuerzo que esto supone, si ha sabido entender que su tío tenía que trabajar y que no le podía dedicar todo el tiempo que hubiera querido. A mi cuñado, Cristobal, quiero agradecerle que siempre esté ahí para lo que haga falta y por ser un tío tan auténtico.

Y tirando de tópicos… se dice que detrás de cada hombre, hay una gran mujer. Yo además quiero añadir que: detrás de cada doctorando, hay una gran sufridora. GRACIAS, Helena. A pesar (te lo prometí) de que nunca te hizo ilusión que me embarcara en esta aventura y siendo la persona que más ha sufrido mis agobios o faltas de tiempo, desde el primer momento me has apoyado en mis decisiones y has sabido darme ánimos para conseguirlo, sobre todo en estos últimos momentos. Además, tengo que reconocer, como ya te he dicho en muchas ocasiones, que si no te hubiera conocido no sé qué hubiera sido de mí ;-) . Quizás, a estas alturas en lugar de pensar en terminar la tesis, estaría pensando en acabar la carrera. En los últimos años has sido mi apoyo, mi referencia y una de mis mayores motivaciones para conseguir todos los retos a los que me he enfrentado. No quiero entrar en terreno personal, pero espero poder tener, a partir de ahora, más tiempo para ti.

Por último, quiero agradecerte a ti, lector, que estés empleando tu tiempo en leer estas líneas.

Gracias a todos[1],

Álvaro.

[1] La foto de la portada se encuentra publicada en http://juanluisvera.blogspot.com/2011/11/el-camino-tambien-deja-huella.html bajo licencia CC (http://creativecommons.org/licenses/by-nc/3.0/).

Índice

Lista de Figuras

Lista de Tablas

Introducción

En la presente tesis, se propone la definición de un marco metodológico para la generación (semi-)automática de enlaces de traza en un entorno de desarrollo de transformaciones de modelos dirigido por modelos así como la construcción de un entorno tecnológico que dé soporte a dicho marco metodológico.

En la primera sección de este capítulo (sección 1.1) se presenta el contexto y el problema a abordar este trabajo así como las principales contribuciones del mismo. A continuación, en la sección 1.2 se establece la hipótesis principal y los objetivos que derivan directamente de la misma. En la sección 1.3 se describe el contexto en que se desarrolla este trabajo, haciendo referencia a los proyectos de investigación en los que se ha tomado parte. Y finalmente, en la sección 1.4 se proporciona una visión general del resto de este documento.

1.1 Planteamiento del Problema

A lo largo de la historia de la Ingeniería del Software, los desarrolladores e investigadores han realizado grandes esfuerzos con el objetivo de simplificar el proceso de desarrollo de software a partir del aumento gradual del nivel de abstracción al que se diseña y se desarrolla. Así, por ejemplo, la programación en lenguaje ensamblador dio paso al empleo de lenguajes de programación como C o Fortran, que elevaban el nivel de abstracción al que se programaba y diseñaba. Esta tendencia ha continuado hasta nuestros días y así han surgido otros paradigmas como la orientación a objetos o la orientación a aspectos [176, 214].

El último paso natural en esta tendencia de aumentar el nivel de abstracción al que se construye el software es la **Ingeniería Dirigida por Modelos** (*MDE, Model-Driven Engineering*) [22].

Con la llegada de MDE, el escenario en el que se construye el software ha cambiado drásticamente: los desarrolladores han pasado de centrarse en la codificación de la solución a centrarse en el modelado del problema. Sin embargo, conviene mencionar que la idea de usar modelos a lo largo del proceso de desarrollo no es en absoluto nueva. Con anterioridad a la llegada de MDE, los modelos ya estaban presentes en tareas como la documentación o el diseño del sistema e incluso en algunos casos, a partir de estos modelos era posible generar la estructura o esqueleto del sistema. Un ejemplo de este escenario podría ser *Rational Rose* [88], que a partir de modelos UML es capaz de generar la

especificación de las clases recogidas en dichos modelos, lo que podría utilizarse como punto de partida para implementar el resto del sistema. Sin embargo una vez que cumplían su función, tradicionalmente limitada a las fases de análisis y diseño (de alto nivel), estos modelos eran descartados en fases sucesivas del proceso de desarrollo y en la mayoría de los casos no eran actualizados para reflejar cambios posteriores.

Los principales cambios o novedades que aporta MDE son: el rol principal que juegan los modelos en el proceso de desarrollo y el aumento del nivel de automatización del proceso en sí mismo. De hecho, una de las mejores formas de aprovechar al máximo las ventajas que ofrece MDE, en términos de desarrollos más rápidos y baratos, pasa por automatizar al máximo posible el proceso de desarrollo [14, 66].

El impacto de MDE puede verse reflejado en nuevas metodologías de **Desarrollo de Software Dirigido por Modelos** (*MDSD, Model-Driven Software Development*) [186, 213]. Estas metodologías se centran en potenciar el nivel de automatización en la construcción de software, proponiendo el modelado del sistema a diferentes niveles de abstracción, de forma que se construyan modelos cada vez más detallados hasta que estos puedan ser utilizados como planos de menor nivel para la generación de código o en algunos casos, constituyan el sistema ejecutable en sí mismo [1, 47, 60, 129, 222].

En este contexto, uno de los puntos clave para aumentar el nivel de automatización es el empleo de **transformaciones** de modelo a modelo (M2M, *model-to-model*) y de transformaciones de modelo a código (M2T, *model-to-text*) que actúan como eslabones, uniendo cada paso del proceso [22, 73, 179, 192]. Así, las primeras se utilizan para convertir uno (o varios) modelos del sistema en otro modelo (u otros), mientras que las segundas se utilizan para la serialización de los modelos en el código que implementa el sistema.

Una de las interpretaciones más adoptadas de los principios de MDE es la realizada por el consorcio internacional *Object Management Group*[2] (*OMG*): **Arquitectura Dirigida por Modelos** (*MDA, Model-Driven Architecture*) [66, 69, 84, 106, 139, 153, 180, 219]. MDA define una arquitectura basada en tres niveles conceptuales o niveles de abstracción: CIM (*Computation Independent Model*), PIM (*Platform Independent Model*) y PSM (*Platform Specific Model*).

Los detalles del dominio de la aplicación se modelan mediante Modelos Independientes de Computación (nivel CIM) que sirven de conexión entre los

[2] http://www.omg.org

expertos del negocio y los desarrolladores del sistema. La funcionalidad y estructura del sistema, abstrayéndose de los detalles tecnológicos de la plataforma que soportará al sistema se modelan mediante Modelos Independientes de Plataforma (nivel PIM). Estos modelos podrán ser refinados tantas veces como sea necesario, con el fin de obtener una descripción del sistema con el nivel de claridad y abstracción óptimo. Para combinar estas especificaciones independientes de plataforma con los detalles específicos de la plataforma en la que se implementará el sistema final, se utilizan Modelos Específicos de Plataforma (nivel PSM). Partiendo de diferentes modelos PSM se puede obtener distintas implementaciones de un mismo sistema. Este nivel PSM, a su vez puede contener diferentes modelos que representan distintos grados de abstracción, pudiéndose agrupar los mismos en modelos que representan elementos generales a todas las plataformas así como modelos que representan elementos dependientes de una plataforma específica. Esta distinción da lugar a la consideración de otro nivel de abstracción denominado PDM (*Platform Dependent Models*) [74, 153, 167]. Adicionalmente, el código que implementa el sistema puede ser considerado otro modelo de más bajo nivel de abstracción [153].

Por otro lado, una de las tareas más importantes de la Ingeniería del Software [12, 169] es la identificación, creación y mantenimiento de las relaciones de **trazabilidad.** La IEEE [89] (pp. 78) define la trazabilidad como: "*el grado de relación que puede establecerse entre dos o más productos de un proceso de desarrollo, especialmente productos que tienen un relación predecesor-sucesor o maestro-subordinado con otro producto*".

Una correcta identificación de las trazas permite conocer cómo evolucionan los elementos de un sistema y cómo se relacionan entre sí, a lo largo de todo su ciclo de vida [11]. Además, la adecuada gestión de la información de trazabilidad facilita el desempeño de otras actividades relacionadas con el desarrollo software tales como el análisis del impacto del cambio, pruebas de regresión, validación de requisitos, análisis de coberturas, toma de decisiones, gestión de la configuración y en general, todas las tareas de mantenimiento del software [5, 9, 37, 38, 42, 107, 142, 160, 177, 182, 193, 218].

Por todo ello, sería deseable que cualquier propuesta para el desarrollo de software diera soporte para la gestión de la trazabilidad [185]. Desafortunadamente, hasta el momento, la identificación y la gestión de las relaciones de traza ha sido casi exclusivamente una tarea manual, lo que la ha convertido en una tarea difícil y costosa en tiempo y esfuerzo, y en consecuencia, propensa a generar errores [57, 61, 86, 132, 148, 221].

Dado que las tecnologías que dan soporte a MDE han alcanzado cierto grado de madurez [136, 214], es el momento de afrontar nuevos retos en el desarrollo de software dirigido por modelos. En particular, esta tesis se centra en aplicar los principios de MDE a la identificación, creación y mantenimiento de las relaciones de **trazabilidad**. Estas tareas son, si cabe, aún más importante en desarrollos dirigidos por modelos, donde la construcción de los sistemas tiende a automatizarse y por tanto, se requiere un mayor control acerca de cómo han sido creados los artefactos. Esta tendencia hacia la automatización surge como una oportunidad para mejorar el grado de **automatización** en la creación y gestión de las relaciones de trazabilidad que tienen lugar en cualquier desarrollo dirigido por modelos [5, 77].

De acuerdo a los principios de MDE [102, 176, 179], asumimos que los modelos empleados para la construcción del sistema se encuentran conectados entre sí mediante transformaciones, que es lo esperado en un desarrollo dirigido por modelos [73]. Por tanto, sería posible emplear las relaciones de transformación para generar de forma automática los enlaces de traza entre los distintos elementos de los modelos del sistema.

Este escenario es representado en la Figura 1-1: dos modelos (Ma y Mb) se conectan mediante una transformación (MMa2MMb), definida a nivel de metamodelo. En MMa2MMb se ha definido una relación de transformación entre las clases *p* y *p'* y una relación de transformación entre las clases *q* y *q'*. A partir de estas relaciones de transformación, es posible identificar las relaciones de trazabilidad existentes entre dichas clases (a nivel de metamodelo). Así, como muestra la Figura 1-1, es posible transformar las instancias (a nivel de modelo) de la clase *p* (*p1* y *p2*) en instancias de la clase *p'* (*p'1* y *p'2*) y además, generar automáticamente las trazas existentes entre los elementos transformados (MTrace).

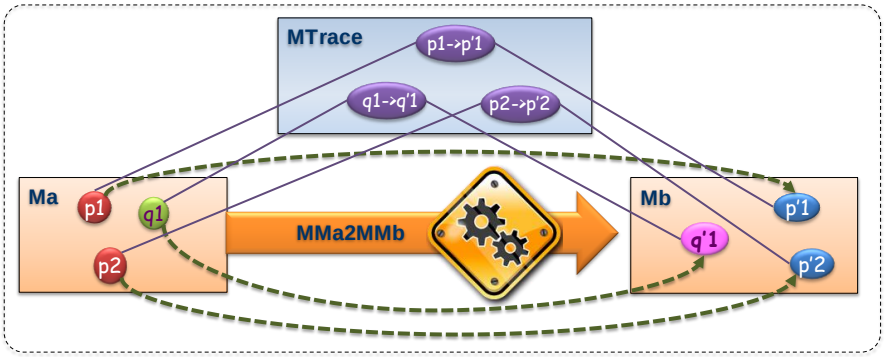

Figura 1-1. Ejemplo de trazas obtenidas en un escenario MDE

Ahora bien, mediante esta idea estaríamos trasladando el problema de la creación y gestión manual de las relaciones de trazabilidad por el de codificar o desarrollar una transformación. Este paso se considera un avance ya que, como se han mencionado anteriormente, de acuerdo a los principios de MDE los sistemas son construidos a partir de modelos y transformaciones entre dichos modelos, por tanto, la tarea de desarrollar transformaciones se lleva a cabo en cualquier caso.

No obstante, dado que las transformaciones de modelos son cada vez más complejas, debido al mayor número de artefactos implicados en las mismas, resulta recomendable aplicar los principios de MDE para su desarrollo [23, 24, 54, 82, 97, 122, 149]. En este contexto, sería aún mejor si pudiésemos reemplazar la codificación de las transformaciones por su especificación a alto nivel, de forma que dicha especificación sería también una definición implícita de las relaciones de trazabilidad a nivel de metamodelo. En otras palabras, resultaría conveniente poder modelar las transformaciones a alto nivel y refinar automáticamente dichas especificaciones en especificaciones (o modelos) de más bajo nivel. Además, el hecho de especificar las transformaciones (e implícitamente las relaciones de trazabilidad) en forma de modelo resulta en una serie de ventajas adicionales: en tanto en cuanto las transformaciones sean un modelo más, podremos generarlas, transformarlas, validarlas, compararlas, refactorizarlas, etc. utilizando técnicas propias de MDE [21].

Por todo ello, en esta tesis doctoral se plantea incorporar mecanismos de generación semi-automática de trazabilidad en un entorno de desarrollo de transformaciones de modelos dirigido por modelos como el que se presenta en [30]. Así, la propuesta que se presenta en esta tesis, *MeTAGeM-Trace*, permitirá generar de forma semi-automática modelos de trazas a partir del modelado de transformaciones de modelos a diferentes niveles de abstracción, independientemente del lenguaje en el que finalmente se ejecute la transformación.

Las principales contribuciones que incluirá el **entorno para el desarrollo de transformaciones de modelos dirigido por modelos con soporte para la generación de modelos de trazas**, denominado MeTAGeM-Trace serán:

- Gestión de la trazabilidad en el desarrollo dirigido por modelos de transformaciones de modelos:
  - Definición de un metamodelo de trazabilidad lo suficientemente genérico para cubrir las necesidades de cualquier escenario que se pueda dar en cualquier entorno dirigido por modelos.

  - Soporte para el modelado y generación de modelos de transformación a diferentes niveles de abstracción que integren información de trazabilidad para la generación de modelos de trazas conformes al metamodelo anterior.
  - Mecanismos de visualización *ad-hoc* para modelos de trazas conformes al metamodelo anterior. Estos mecanismos de visualización, tendrán en cuenta la naturaleza de los modelos de traza para resultar usables e intuitivos en ese contexto.
  - Incorporación de construcciones para el modelado de reglas de transformación para la generación de trazas en los DSLs definidos en [30] para el modelado de transformaciones a distintos niveles de abstracción.
  - Incorporación de mecanismos de visualización *ad-hoc* para las relaciones entre los elementos de los metamodelos implicados en la transformación que se desea modelar.
- Generación de enlaces de traza en el desarrollo de transformaciones de modelos. Esta implementación seguirá la propuesta presentada en [199] para el desarrollo dirigido por modelos de sistemas de información.

1.2 Hipótesis y Objetivos

La **hipótesis** planteada en esta Tesis Doctoral es que *es factible proporcionar un entorno de desarrollo de transformaciones de modelos dirigido por modelos que incorpore generación de información de trazas y permita, por tanto, generar trazas a partir de especificaciones de alto nivel de las relaciones entre los elementos de los modelos de origen y destino*.

El **objetivo principal** de este trabajo de investigación, derivado directamente de la hipótesis, es: *la definición y construcción de un entorno de desarrollo de transformaciones de modelos dirigido por modelos que incorpore generación de información de trazas y permita generar trazas entre los elementos de los modelos de origen y destino de una transformación.*

Para la consecución de este objetivo se han planteado los siguientes objetivos parciales:

O1. Estudio de trabajos previos:

O1.1. Análisis y evaluación de trabajos relacionados con la generación de información de trazabilidad en el desarrollo de transformaciones, incluyendo propuestas metodológicas y herramientas.

- **O1.2.** Análisis y evaluación de propuestas para el desarrollo dirigido por modelos de transformaciones de modelos.

O2. Especificación de un metamodelo de trazabilidad genérico.

O3. Estudio y mejora (si procede) de la propuesta MeTAGeM para el desarrollo transformaciones de modelos [30]:

- **O3.1.** Estudio y mejora (si procede) del metamodelo para la definición de modelos de transformaciones a nivel PIM.
- **O3.2.** Estudio y mejora (si procede) de los metamodelos para la definición de modelos de transformación a nivel PSM. Este nivel se corresponde con la aproximación seleccionada por el desarrollador, esto es, imperativa, declarativa, híbrida, grafos, etc. En esta tesis se selecciona la aproximación híbrida (combinación de las aproximaciones imperativa y declarativa).
- **O3.3.** Estudio y mejora (si procede) de los metamodelos para la definición de modelos de transformación a nivel PDM. MeTAGeM propone ATL y RubyTL como lenguajes híbridos a nivel PDM. En esta tesis nos centraremos en ATL.

O4. Especificación de un metamodelo de transformación a nivel PDM de acuerdo a las características del lenguaje híbrido ETL.

O5. Identificación de construcciones que, incorporadas a los metamodelos de transformación definidos a nivel PIM, PSM y PDM, permitirán generar modelos de traza conformes al metamodelo definido en O2.

- **O5.1.** Incorporación de dichas construcciones a los metamodelos de transformación definidos a nivel PIM, PSM y PDM.

O6. Especificación de transformaciones de modelos:

- **O6.1.** Especificación de las transformaciones de modelos de transformaciones de nivel PIM a modelos de nivel PSM.
- **O6.2.** Especificación de las transformaciones de modelos de transformación de nivel PSM a modelos de nivel PDM, es decir, modelos dependientes de un lenguaje de transformación determinado. En concreto en esta tesis emplearemos ATL y ETL a nivel PDM.

O7. Construcción de la herramienta de soporte:

- **O7.1.** Especificación de la arquitectura de la herramienta de soporte.
- **O7.2.** Desarrollo del DSL para el modelado de enlaces de traza.

O7.3. Implementación de los DSLs para el modelado de transformaciones a nivel PIM, PSM y PDM.

O7.4. Implementación de transformaciones de modelos:

- Transformación de modelos definidos a nivel PIM a modelos definidos a nivel PSM.
- Transformación de modelos definidos a nivel PSM a modelos definidos a nivel PDM.
- Transformación de modelo a texto, que permitirán obtener el código para un lenguaje concreto (ATL y ETL).

O7.5. Integración de la funcionalidad proporcionada como resultado de las tareas anteriores. Para ello, se desarrollarán módulos que proporcionen una interfaz visual que facilite la ejecución de las transformaciones.

O8. Validación del entorno de desarrollo propuesto.

O8.1. Para ello se desarrollará un meta-caso de estudio que permitirá validar, tanto la propuesta metodológica, como el correcto funcionamiento de la herramienta.

O8.2. Además, se desarrollará un caso de estudio que consistirá en probar las transformaciones de modelos generadas en el meta-caso de estudio.

1.3 Marco de Investigación

En los últimos años, el grupo de investigación Kybele, del cual forma parte el doctorando, ha venido trabajando en la definición de MIDAS [49, 50, 131, 204, 209], una metodología centrada en la arquitectura para el desarrollo dirigido por modelos de Sistemas de Información (SI). En el marco de MIDAS, se han definido distintas propuestas destinadas a dar solución a problemas concretos en el contexto del desarrollo de SIs: SOD-M [48] una aproximación metodológica basada en MDA para el desarrollo orientado a servicios de SIs; PISA [4] una arquitectura de integración de portales Web basados en servicios web semánticos; MIDAS MDA Tool (M2DAT) [199] la herramienta MDA que soporta cada uno de los métodos propuestos en MIDAS; y MeTAGeM [30], una meta-herramienta para semi-automatizar el desarrollo de transformaciones de modelos para M2DAT.

En la Figura 1-2 se presenta la arquitectura de MIDAS. Esta arquitectura está basada en los principios de MDA [153] y se define sobre una base multidimensional que se propaga a través de varios niveles de abstracción y diferentes aspectos del desarrollo del sistema. En cada una de las dimensiones o aspectos se definen una serie de modelos y reglas de transformación entre dichos modelos, además se establecen las relaciones existentes entre los mismos.

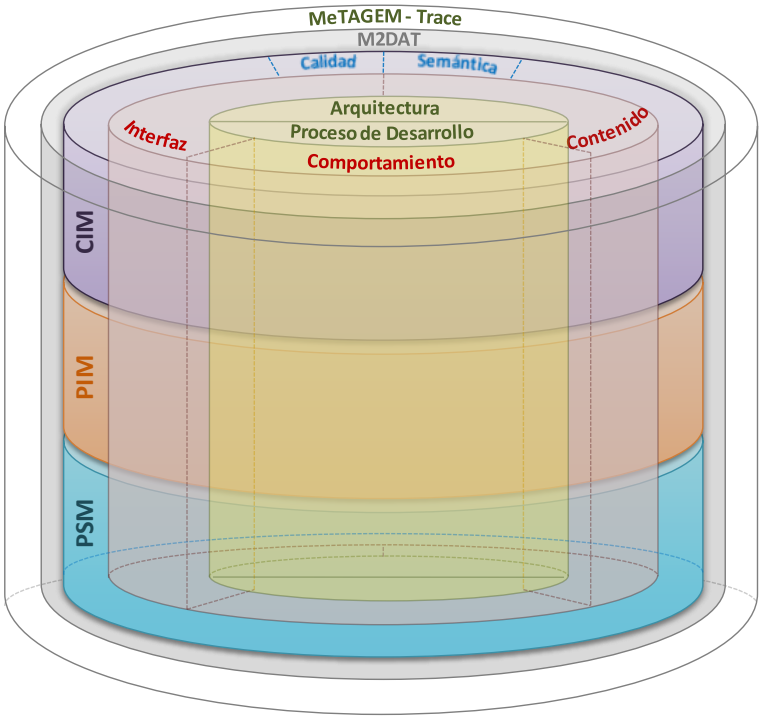

Figura 1-2. Arquitectura de MIDAS

- **Dimensión vertical.** Esta dimensión tiene una relación directa con los tres niveles de abstracción definidos por MDA: CIM, PIM y PSM. De acuerdo a esta clasificación: los conceptos relacionados con el dominio del problema son especificados en los modelos CIM; la estructura y funcionalidad del sistema, omitiendo los detalles tecnológicos de la plataforma sobre la que se desarrolla, son definidas en modelos a PIM; y la representación del sistema, teniendo en cuenta las características propias de la plataforma sobre la que se desarrolla, se especifica en modelos a nivel PSM. Para establecer relaciones entre dichos modelos, se definen transformaciones de modelos.

- **Núcleo de la arquitectura de modelos.** En MIDAS la arquitectura guía el proceso de desarrollo [134], por lo que sus modelos forman parte del núcleo de MIDAS. De hecho, la arquitectura especifica las características que afectan a todos los aspectos del sistema.
- **Capa interior concéntrica.** En esta capa, los modelos se organizan de acuerdo a los tres principales aspectos involucrados en el desarrollo de un SI: contenido, hipertexto y comportamiento.
- **Capa externas.** En el desarrollo software existen otros aspectos importantes, tales como la semántica, la seguridad o la calidad, que también son definidos por MIDAS. Estos aspectos son ortogonales a los anteriormente mencionados (contenido, hipertexto y comportamiento). Por esta razón, son incluidos en otra capa concéntrica del cilindro.
- **Herramientas de soporte (M2DAT y MeTAGeM-Trace).** Finalmente, las dos capas más externas hacen referencia a la herramienta que soporta MIDAS (M2DAT [199]) y a la herramienta MeTAGeM-Trace (objetivo de esta tesis doctoral), respectivamente. MeTAGeM-Trace se utiliza para desarrollar las transformaciones embebidas en los diferentes módulos de M2DAT. Para ello, MeTAGeM-Trace soporta el desarrollo semi-automático de transformaciones incluyendo, además, la generación de enlaces de traza.

1.3.1 Proyectos de Investigación

La investigación realizada en esta tesis doctoral se ha llevado a cabo en el Grupo de Investigación Kybele de la Universidad Rey Juan Carlos (URJC). Este trabajo, como muestra la Figura 1-3, se enmarca fundamentalmente en el contexto del proyecto de investigación MODEL-CAOS y su continuación, MASAI y ha sido co-financiado por el programa I+D de Personal Técnico de Apoyo.

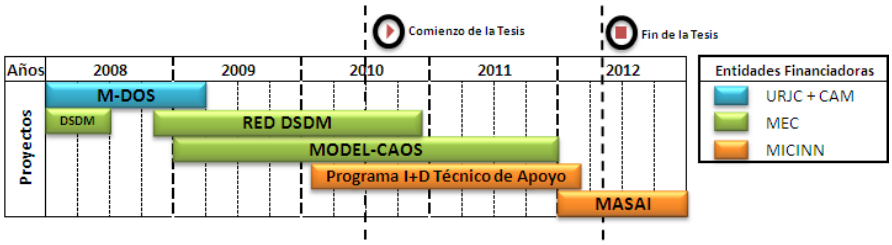

Figura 1-3. Proyectos de Investigación de la Tesis Doctoral

El proyecto **MODEL-CAOS** [TIN2008-03582], financiado por el Ministerio de Educación y Ciencia, comenzó en el año 2009 y ha tenido una

duración de tres años (hasta 2011). El objetivo principal de MODEL-CAOS ha sido la especificación de un marco para el desarrollo (semi-)automático de SI, centrándose en la utilización del paradigma de Orientación a Servicios. En el marco de dicho proyecto se propone el desarrollo de una meta-herramienta que permita automatizar las tareas del DSDM. Este proyecto toma como base los trabajos realizados en proyectos anteriores, actualizándolos mediante la inclusión de las últimas tendencias en el desarrollo de SI: desarrollo dirigido por modelos, orientación a servicios, etc., poniendo especial énfasis en el aspecto arquitectónico como elemento central que guía el proceso metodológico. En el contexto del proyecto MODEL-CAOS es donde se ha desarrollado la mayor parte de esta Tesis Doctoral. En particular, el doctorando ha trabajado en el desarrollo de una propuesta metodológica y tecnológica para el desarrollo (semi-)automático de transformaciones de modelos que incluyese generación de enlaces de traza.

La continuación del proyecto MODEL-CAOS es el proyecto **MASAI** [TIN TIN-2011-22617] cuyo objetivo principal es aplicar técnicas de MDE para proporcionar soluciones a algunos de los problemas que encontramos en la Ingeniería de Servicios (IS), aprovechando los conocimientos y experiencia adquiridos en el marco de los proyectos anteriores desarrollador por el grupo Kybele. En particular, el doctorando continúa refinando la propuesta presentada en esta Tesis Doctoral con el objetivo de producir una meta-herramienta completa que se utilizará para la construcción del entorno de desarrollo que soporta las propuestas metodológicas de MASAI para la evolución de servicios y el *Gap Analysis*.

Por otro lado, el **Subprograma de Personal Técnico de Apoyo** [MICINN-PTA – 2009], pertenece al Programa Nacional De Contratación e Incorporación de RRHH, cofinanciado por el Ministerio de Ciencia e Innovación. El principal objetivo de este subprograma es la cofinanciación de la contratación laboral de personal técnico en los centros de I+D para, de este modo, incrementar y mejorar las prestaciones y rendimiento de estos centros. La duración de la ayuda, en el caso de la contratación del doctorando, ha tenido una duración de dos años (febrero de 2010 a febrero de 2012).

Igualmente conviene mencionar que durante el desarrollo de esta Tesis Doctoral, el doctorando ha formado parte como investigador de las dos ediciones de la **Red de Desarrollo de Software Dirigido por Modelos** (Red DSDM) [TIN2005-25886-E y TIN2008-00889-E] (2006-2010). El objetivo de esta red ha sido facilitar el intercambio y la transferencia de conocimientos y experiencias entre los distintos grupos de la comunidad de DSDM, así como fomentar su cooperación, tratando de aunar y articular los esfuerzos nacionales para consolidar

la posición española y mejorar el reconocimiento internacional en el área de DSDM.

Además de los proyectos anteriores, el doctorando participó en el proyecto **M-DOS** [URJC-CM-2007-CET-1607], cofinanciado por la Universidad Rey Juan Carlos y la Comunidad de Madrid. En este proyecto se implementó un conjunto de metamodelos, correspondientes al aspecto de comportamiento de la arquitectura de MIDAS.

1.4 Estructura de la Tesis

El resto de los capítulos de esta tesis, se organizan de la siguiente manera:

- El **Capítulo 2** presenta el método de trabajo seguido en el desarrollo de esta tesis doctoral. En particular, la sección 2.1 detalla el método de investigación, la sección 2.2 el método seguido para la identificación del cuerpo de conocimiento y la sección 2.3, el método de construcción o desarrollo. Por último, la sección 2.4 presenta el método seguido para llevar a cabo la validación del entorno presentado en este trabajo.
- El **Capítulo 3** proporciona una visión completa sobre el estado del arte. Previamente la sección 3.1 proporciona una vista general de algunos conceptos previos relacionados con este trabajo. A continuación se realizan dos revisiones de la literatura para identificar y analizar respectivamente: propuestas para el desarrollo de transformaciones de modelos aplicando los principios del desarrollo dirigido por modelos (sección 3.2) y propuestas para la generación de trazabilidad en el contexto del desarrollo de transformaciones de modelos (sección 3.3).
- El **Capítulo 4** presenta la solución metodológica propuesta en esta tesis: MeTAGeM-Trace, un entorno para el desarrollo de transformaciones de modelos dirigido por modelos que incluye un meta-generador de trazas. En primer lugar, la sección 4.1 presenta un proceso de desarrollo dirigido por modelos para la construcción de transformaciones de modelos que considera cuatro niveles de abstracción (incluyendo el código final). Para concretar este proceso, la sección 4.2 presenta los metamodelos que se utilizan para el modelado de transformaciones a diferentes niveles de abstracción; la sección 4.3 presenta la especificación de las reglas de transformación que permiten conectar dichos modelos y finalmente la sección 4.4 recoge las reglas de

validación o restricciones que deben cumplir los modelos terminales conformes a dichos metamodelos.

- El **Capítulo 5** presenta la solución tecnológica que da soporte a la propuesta metodológíca presentada en el capítulo 4. Para ello, en la sección 5.1 se presenta la arquitectura conceptual de la herramienta, que se plasma en el diseño técnico presentado en la sección 5.2, siguiendo el proceso de desarrollo presentado en la sección 5.3.
- El **Capítulo 6** presenta la validación del entorno MeTAGeM-Tracede acuerdo al método de validación presentado en el Capítulo 2. En la sección 6.1 se define el protocolo de validación, en el que se establece el desarrollo de un meta-caso de estudio y de un caso de estudio, mientas que el diseño y la ejecución de cada uno se detallan en las secciones 6.2 y 6.3, respectivamente. Finalmente, la sección 6.4 resume las principales conclusiones extraídas de la validación del entorno.
- Finalmente, el **Capítulo 7** concluye con un resumen de las principales contribuciones de esta tesis. Para ello, se analizan los resultados obtenidos y las publicaciones en las que se han materializado estos resultados. Además, se plantean una serie de líneas de investigación futuras que se plantean a raíz de este trabajo de tesis doctoral.
- Adicionalmente, el **Apéndice A** recoge los detalles del proceso de revisión sistemática de la literatura que se ha llevado a cabo para la identificación y análisis de las propuestas centradas en la generación de trazabilidad a partir del desarrollo de transformaciones de modelos, mientras que el **Apéndice B** presenta el manual de usuario de la herramienta MeTAGeM-Trace.

Método de Trabajo

En este segundo capítulo se presentan los diferentes métodos de trabajo que se han seguido para llevar a cabo esta tesis doctoral. En primer lugar, en la sección 2.1 se presenta el método de investigación que define todo el proceso. Dicho método de investigación, a su vez, engloba otros métodos que son detallados en este capítulo: el método de identificación del cuerpo de conocimiento (sección 2.2), el método de desarrollo o construcción (sección 2.3) y el método de validación (sección 2.4).

2.1 Método de Investigación

Las diferencias que radican en la naturaleza de las Ingenierías, incluida la Ingeniería del Software, respecto de otras disciplinas empíricas y formales, dificultan la aplicación directa de los métodos de investigación tradicionales.

Teniendo en cuenta esta problemática, el método de investigación que se ha seguido en esta tesis se basa en el método propuesto en [135] para la investigación en Ingeniería del Software. Dicho método se basa en el hipotético-deductivo propuesto por Bunge [41] y se compone de un conjunto de pasos genéricos, mostrados en la Figura 2-1, de forma que pueda aplicarse a la investigación en cualquier disciplina.

De acuerdo con este método, el primer paso consiste en identificar un **cuerpo del conocimiento** y una serie de problemas o puntos de mejora en el estado actual de dicho cuerpo del conocimiento. Esta información es identificada mediante el seguimiento de un método de identificación del cuerpo del conocimiento que es detallado en la sección 2.2. Como resultado se obtiene un estado del arte (capítulo 3) que servirá de base teórica sobre la que se desarrolla la investigación, o en este caso concreto, la tesis doctoral. A partir del análisis exhaustivo del cuerpo del conocimiento y de sus problemas sin resolver o puntos de mejora, se determina el **problema** al cual se quiere dar respuesta. Una vez que se ha definido el problema, se formula la **hipótesis** y se define un conjunto de objetivos que debemos cumplir para alcanzar la solución que verifique la hipótesis planteada en la Sección 1.2.

El siguiente paso, como se puede ver en la Figura 2-1, consiste en la definición del método de investigación (método de trabajo) para la resolución y validación del problema. Los autores Marcos y Marcos [135], a diferencia de lo propuesto por Bunge [41], recomiendan definirlo de esta manera, ya que la

naturaleza de cada investigación tiene sus propias características y por lo tanto, no sería conveniente aplicar un único método universal. En el proceso de investigación de esta tesis se ha empleado una adaptación del método de trabajo propuesto en [30] y [199]. Este método se compone de dos etapas principales: **resolución** y **validación,** que son descritas en detalle en la siguiente sub-sección.

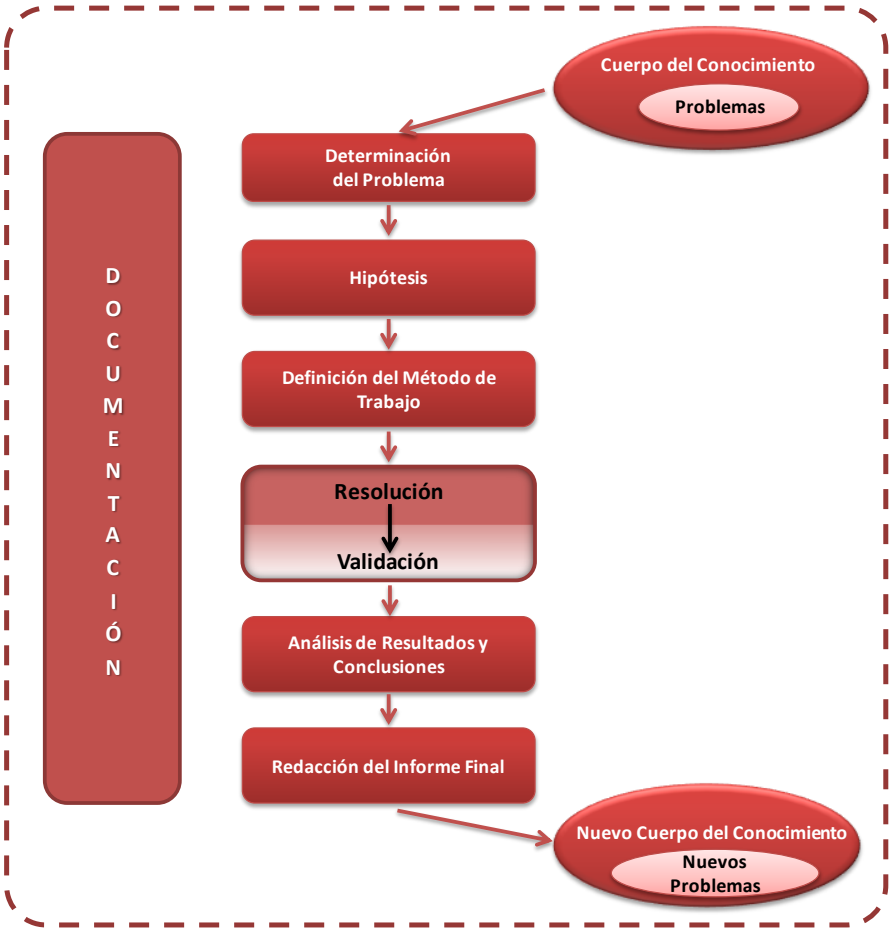

Figura 2-1. Método de Investigación propuesto por Marcos y Marcos [135]

Una vez que se han realizado las etapas de resolución y validación, la hipótesis y los objetivos planteados deben ser contrastados con los resultados obtenidos, especificando la forma y el grado en que han sido cumplidos. Del mismo modo, es recomendable especificar aquellos aspectos que no se han podido resolver y los nuevos problemas que han podido surgir a partir de la investigación realizada, ya que esta información puede servir como punto de partida para futuras

investigaciones [135]. En el capítulo 7 de esta tesis doctoral se presenta el **análisis de los resultados**, la **identificación de nuevos problemas** y las **conclusiones** obtenidas del proceso de investigación.

2.1.1 Etapa de Resolución y Validación

Como se ha indicado anteriormente, en la etapa de resolución y validación se ha seguido una adaptación del proceso propuesto y validado en [30] y [199]. Se trata de una adaptación de dos procesos ampliamente conocidos y empleados en el campo de la Ingeniería del Software: el desarrollo en cascada [28, 172] y el Proceso Unificado de *Rational* (RUP) [92, 117], tomando como base la definición de etapas consecutivas del primero y el proceso iterativo del segundo.

La elección de un proceso basado en los dos anteriores para abordar esta tesis doctoral se fundamenta en la similitud que existe entre la naturaleza del problema a resolver y los problemas que surgen en el desarrollo de software. En efecto, existen ciertos problemas de investigación en Ingeniería del Software (como el que se presenta en esta tesis) que tienen en sí mismo una naturaleza ingenieril, ya que se trata de la construcción de nuevos artefactos [133]; en el caso de este trabajo, se trata de construir un entorno para el desarrollo de transformaciones dirigido por modelos, con capacidad para generar modelos de trazas. Dicho entorno estará compuesto por una **propuesta metodológica** y una **herramienta que proporcione soporte tecnológico** a dicha propuesta.

La Figura 2-2 muestra, de forma simplificada, el proceso de resolución y validación empleado en esta tesis doctoral. Este proceso consta de **tres iteraciones**:

1ª Iteración: Desarrollo del módulo para el modelado de enlaces de traza.

2ª Iteración: Desarrollo de los módulos para el modelado de transformaciones de modelos con soporte para la generación de relaciones de traza.

3ª Iteración: Desarrollo de un meta-caso de estudio.

En primer lugar se lleva a cabo la especificación del proceso a partir del estudio del estado del arte o cuerpo del conocimiento (capítulo 3), la hipótesis planteada (sección 1.2) y la experiencia previa del grupo de investigación en el que se ha realizado esta tesis doctoral [3, 30, 31, 51, 52, 53, 94, 199, 202, 203].

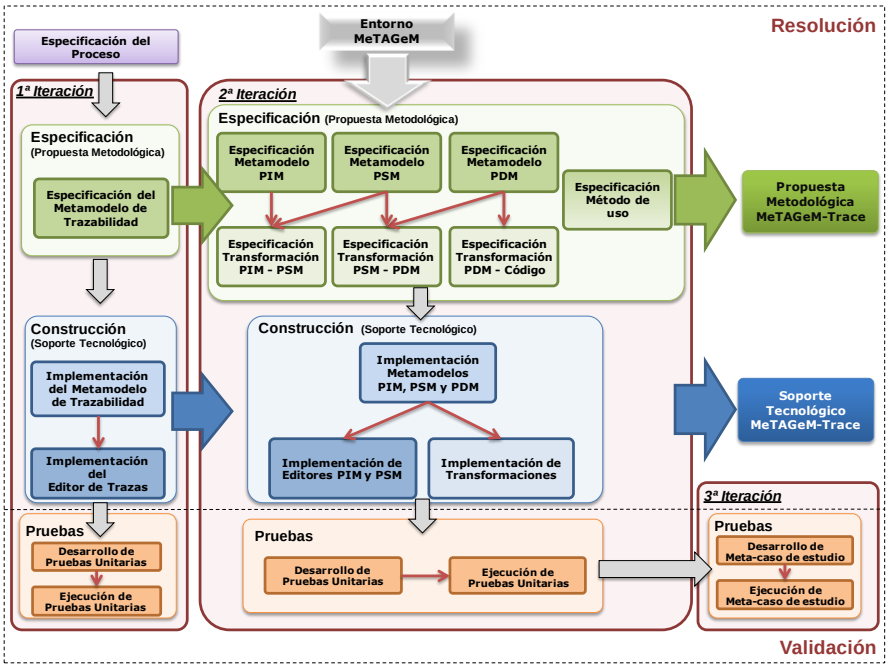

Figura 2-2. Etapa de Resolución y Validación del Método de Investigación

Una vez realizada esta especificación, se da comienzo a la **primera iteración** (desarrollo del módulo de trazabilidad).). Esta iteración se compone de una fase de **especificación**, una fase de **construcción** (resolución) y una fase de **pruebas** (validación). La fase de construcción y la fase de validación se realizan de acuerdo a los métodos de desarrollo y validación que se detallan en las secciones 2.3 y 2.4, respectivamente.

Los resultados obtenidos en la fase de especificación y construcción de la primera iteración, junto con el entorno MeTAGeM [30], sirven como entrada a la **segunda iteración** del proceso (desarrollo de los módulos para el modelado de transformaciones de modelos con soporte para la generación de enlaces de traza). Esta también se compone de las fases de especificación, construcción y pruebas. Como resultado, se obtiene la propuesta metodológica y la herramienta de soporte que se presentan en esta tesis doctoral y que se detallan en los capítulos 4 y 5, respectivamente.

Por último, la **tercera iteración** se compone de una fase de pruebas en la que se desarrolla y ejecuta un meta-caso de estudio que permite llevar a cabo la validación del entorno MeTAGeM-Trace (Capítulo 6).

Es necesario destacar que, como se ha mencionado anteriormente, la Figura 2-2 muestra una simplificación del proceso completo. Para mejorar la visualización de la misma se han omitido los flujos de retroalimentación existentes entre los distintos pasos del proceso. A modo de ejemplo, se puede mencionar que el desarrollo de las pruebas unitarias permite refinar la fase de especificación de la iteración correspondiente; o que la ejecución de dichas pruebas unitarias facilita datos para la mejora de la fase de construcción.

2.2 Método de Identificación del Cuerpo de Conocimiento

Como se ha indicado en la sección anterior, el punto de partida del método de investigación consiste en la identificación de un cuerpo de conocimiento o estado del arte.

Para identificar del cuerpo del conocimiento que sirve de base para la investigación de esta tesis doctoral, se ha empleado un proceso de **revisión sistemática de la literatura** (también conocido como revisión sistemática o RS), basado en las guías propuestas por Kitchenham [104, 105] y Biolchini *et al.* [27]. Una revisión sistemática es un proceso riguroso desarrollado para identificar, evaluar e interpretar toda la información relativa a un tema de investigación [27, 39, 105].

Una de las principales motivaciones para llevar a cabo un proceso de revisión sistemática surge como respuesta a la gran cantidad de información a la que tenemos acceso en nuestros días, gracias a la evolución de las tecnologías de la información en los últimos años y especialmente Internet. Disponer de tanta información facilita la recopilación de datos, sin embargo, no todo son ventajas. Ante tal cantidad de datos es muy fácil pasar por alto aquella información que *a priori*, parece poco relevante para el objetivo de la investigación. Además, es habitual que esta información se gestione de forma dinámica [13, 29] de forma que, por ejemplo, el número de resultados arrojados por una búsqueda puede aumentar considerablemente después un cierto periodo de tiempo.

A diferencia de una revisión literaria tradicional, una RS sigue una secuencia estricta y bien definida de pasos metodológicos que garantiza el alto valor científico de los resultados obtenidos [27]. Además, cada uno de los pasos del proceso, las estrategias para recuperar información y el enfoque con el que se aborda el tema de investigación se definen explícitamente, de manera que otros puedan reproducir el protocolo [27]. El proceso de revisión sistemática seguido en esta tesis doctoral es el mostrado en la Figura 2-3.

Este proceso se compone de tres fases consecutivas: **planificación** (*Planning*), **ejecución** (*Execution*), **análisis de resultados** (*Result Analisys*); y una cuarta fase de **empaquetado** (*Packaging*) que se ejecuta a lo largo de todo el proceso. Además, se definen dos puntos de control en los cuales se evalúa que el proceso de revisión sistemática se esté realizando correctamente [27].

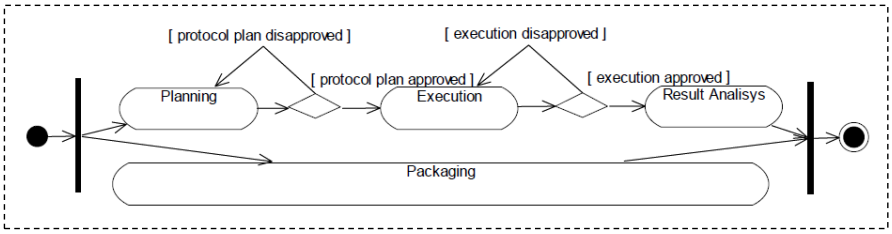

Figura 2-3. Proceso de Revisión Sistemática propuesto por Biolchini *et al.* [27]

La fase de planificación consiste en la definición del objetivo de la investigación y del protocolo para llevar a cabo la revisión [27]. En dicho protocolo se debe especificar información como: preguntas de investigación a las que se intentará responder con la información obtenida en la revisión, estrategias de búsqueda, definiendo las fuentes de información y los términos clave para realizar las búsquedas, criterios de inclusión y exclusión de los estudios que contribuirán a la revisión y los datos que se extraerán de cada estudio [105]. Además de la información anterior, se debe definir el proceso a seguir durante la fase de ejecución.

En la fase de ejecución, se ejecuta el protocolo definido previamente, así pues se lleva a cabo la identificación, análisis y evaluación de los estudios primarios de la revisión sistemática.

Finalmente, en la fase de análisis de resultados se sintetizan los datos obtenidos en la fase de ejecución para dar respuesta a las cuestiones de investigación definidas y al objetivo principal de la revisión sistemática.

De forma paralela a las fases anteriores se lleva a cabo la fase de empaquetado que se encarga de almacenar los resultados de cada una de las fases anteriores.

2.2.1 Proceso para guiar la Revisión Sistemática

Como se ha dicho previamente, en la etapa de planificación se define el proceso que guiará la ejecución de la revisión sistemática. Para definir dicho proceso, en esta tesis doctoral se ha seguido una adaptación del proceso presentado por Pino *et al.* en [166]. Dicho proceso está pensado principalmente

para realizar búsquedas en fuentes digitales como es el caso de esta tesis doctoral. La adaptación realizada del proceso consiste básicamente en las tres etapas mostradas en la Figura 2-4: proceso de búsqueda (Figura 2-4a), selección de estudios primarios (Figura 2-4b) y extracción de datos (Figura 2-4c). A continuación se explica cada una de estas etapas:

2.2.1.1 Proceso de búsqueda

El primer paso del proceso de búsqueda consiste en enumerar las fuentes digitales de las cuales se extraerán los datos y las cadenas de búsqueda que se usarán en cada fuente. Una vez que se ha identificado esta información, se procede a analizar de forma independiente cada motor de búsqueda (DSi) con el objetivo de adaptar las cadenas de búsqueda a las características de la sintaxis del motor. A continuación, se comienza a buscar estudios a partir de la ejecución de cada una de las cadenas adaptadas (QSi) en la fuente digital correspondiente (DSi).

Una vez que se ha ejecutado la búsqueda correspondiente, se enumeran y almacenan los estudios obtenidos (SDi). En este paso, para facilitar su posterior identificación se comienza a extraer algunos datos de los estudios como el título y los autores. A continuación, los estudios identificados, como resultado de ejecutar QSi en DSi, pasan a la etapa de selección de estudios primarios.

2.2.1.2 Selección de Estudios Primarios

En esta segunda etapa, se analiza cada uno de los estudios identificados en la etapa anterior con el objetivo de saber si cumple con los criterios de inclusión definidos anteriormente. Aquellos estudios que satisfacen los criterios de inclusión se convierten en **estudios relevantes** de la revisión sistemática. Este proceso se repite para todas los estudios identificados tras ejecutar la búsqueda de todas las cadenas adaptadas a todas las fuentes de datos seleccionadas.

Una vez que se han identificado todos los estudios relevantes, comienza la segunda parte de la etapa de selección de estudios primarios (parte derecha de la Figura 2-4b). Esta fase consiste básicamente en: enumerar los estudios relevantes ($RS_1 \ldots RS_N$); eliminar la duplicidad en dichos estudios, esto es, mantener solo una copia de cada estudio aunque se haya encontrado en varias fuentes; y por último analizar cada uno de los estudios relevantes, de acuerdo a los criterios de exclusión definidos. Esta última tarea permitirá conocer qué trabajos deben ser excluidos del conjunto de **estudios primarios** de la revisión sistemática.

Tras analizar todos los estudios relevantes y descartar aquellos que cumplen con los criterios de exclusión, los estudios primarios pasan a la última etapa.

2.2.1.3 Extracción de datos

En la última etapa se enumeran los estudios primarios que contribuyen a la revisión sistemática ($PS_1 \ldots PS_N$). A continuación, se analiza cada uno de ellos en detalle para extraer la información definida anteriormente y que nos permitirá dar respuesta al objetivo de la revisión sistemática.

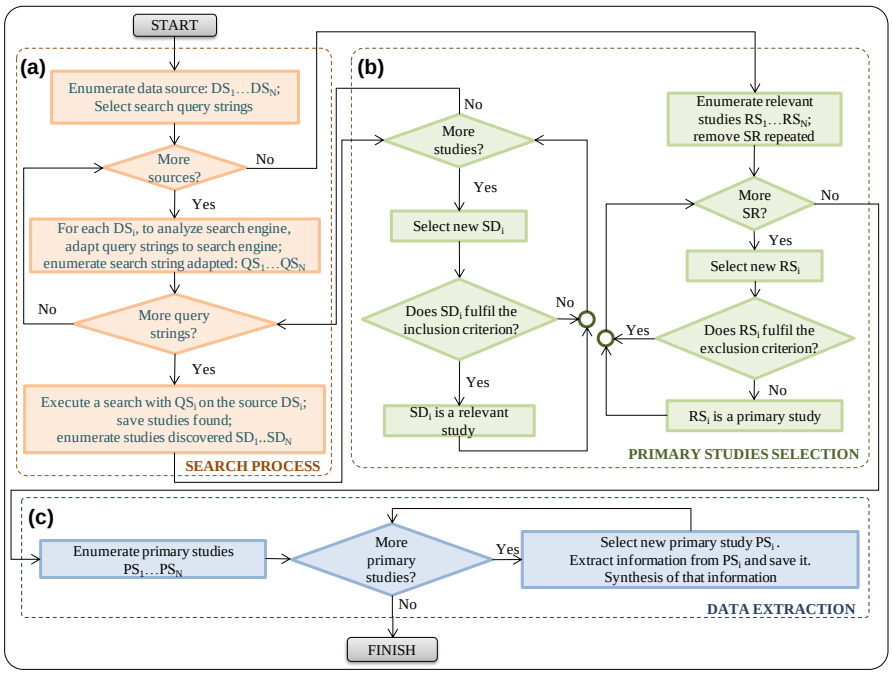

Figura 2-4. Proceso para guiar la RS: (a) proceso de búsqueda, (b) selección de estudios primarios y (c) extracción de datos.

El resultado de este proceso se presenta en el Capítulo 3, donde se presenta el estado del arte que sirve como base teórica a la investigación realizada en esta tesis doctoral.

2.3 Método de Desarrollo

Como se ha indicado en la sección 2.1.1, la etapa de resolución de esta tesis doctoral implica la especificación y construcción de varios lenguajes de modelado, que constituyen el núcleo de la propuesta. Por lo tanto, una de las primeras decisiones que debe tomarse es cómo abordar la especificación e implementación (o construcción) de dichos lenguajes de modelado. En otras palabras, definir el **método de desarrollo** de dichos lenguajes de modelado.

Un paso previo a la definición de dicho método es seleccionar la aproximación a seguir: utilizar perfiles UML o DSLs [219]. En esta tesis doctoral, se ha optado por el empleo de DSLs [68, 140], siguiendo la aproximación adoptada en MIDAS, el marco metodológico en el que se lleva a cabo esta tesis doctoral (sección 1.3). MIDAS, al igual que otras propuestas como UWE [108] o los trabajos de Mazón y Trujillo [137], abandonó el uso de perfiles UML en favor del empleo de DSLs [199]. El uso de DSLs ofrece una serie de ventajas frente al empleo de perfiles UML [199], algunas de las cuales se hacen aún más evidentes a la hora de desarrollar transformaciones de modelos.

Por todo ello, en esta tesis doctoral se ha seguido una adaptación del método de desarrollo de DSLs propuesto por Vara como resultado de un análisis exhaustivo del estado del arte sobre soluciones tecnológicas para soportar las distintas tareas relacionadas con el desarrollo dirigido por modelos [199].

La adaptación del método de desarrollo que se ha seguido en esta tesis doctoral es mostrado en la Figura 2-5.

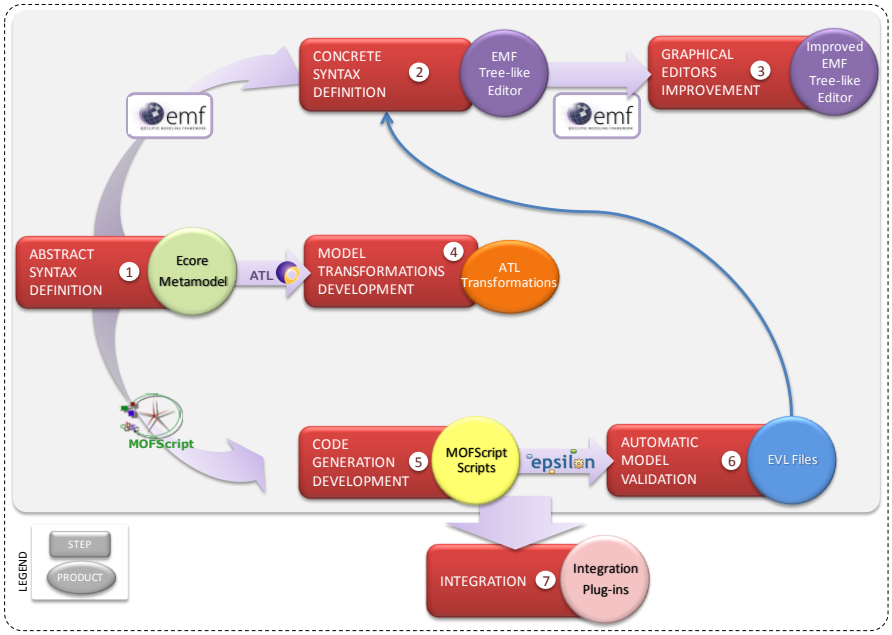

Figura 2-5. Método de Desarrollo

Como muestra la Figura 2-5, el método de desarrollo se compone de siete pasos relacionados entre sí. A continuación, se describe cada uno de ellos.

1. El primer paso consiste en la **definición de la sintaxis abstracta del DSL**. Para llevar a cabo esta tarea, el metamodelo que define el DSL es

construido en términos del lenguaje de metamodelo *Ecore*, mediante el empleo de *Eclipse Modeling Framework* (EMF) [40, 80, 188].

2. El segundo paso consiste en la **definición de la sintaxis concreta del DSL**. Para ello, el *framework* EMF proporciona los medios adecuados para generar editores en forma de árbol que permiten interactuar con modelos terminales conformes al metamodelo definido.
3. Dado que el editor proporcionado por EMF puede resultar demasiado genérico, el tercer paso del proceso consiste en **mejorar dicho editor** para adaptarlo a las necesidades de la naturaleza del DSL.
4. Una vez que se ha definido el DSL y los medios para crear modelos e interactuar con ellos (editores), es momento de definir las relaciones con otros DSLs ya existentes. Así, este paso consiste en el **desarrollo de las transformaciones de modelos** que permitirán conectar el nuevo DSL con los ya existentes. El desarrollo de estas transformaciones se lleva a cabo mediante los siguientes pasos:
    - Definición en lenguaje natural de un conjunto de reglas estructuradas.
    - Convertirlas en reglas ejecutables mediante su implementación con alguno de los lenguajes de transformaciones de modelos existentes [46]. En particular, se propone el uso de ATL (*ATLAS Transformation Language*) [98, 99, 101], considerado a día de hoy el estándar *de-facto* para transformaciones de modelos
5. Si el DSL se define a nivel PSM o PDM, es posible que se requiera generar código a partir de los modelos elaborados con dicho DSL. En tal caso, el sexto paso consistiría en el **desarrollo de transformaciones de modelo a texto para la generación de código**. Para llevar a cabo esta tarea, se propone el uso del lenguaje MOFScript [144, 147].
6. Una vez que se han desarrollado todas las transformaciones (M2M y M2T) en las que se encuentra implicado el DSL, hay que recoger todas aquellas restricciones que no han podido contemplarse en la definición del metamodelo del DSL o que no han podido ser controladas mediante las transformaciones. Para implementar estas **reglas de validación**, se propone utilizar EVL (*Epsilon Validation Language*, [116]).
7. Finalmente, dado que los pasos anteriores proporcionan como resultado un conjunto de *plug-ins*, el último paso consiste en realizar la **integración** de estos *plug-ins* entre sí y con otros módulos ya existentes, si fuera necesario.

La puesta en práctica de este proceso se presenta en el Capítulo 5, donde se explica en detalle los pasos seguidos para la especificación y construcción de la propuesta metodológica MeTAGeM-Trace y la herramienta que da soporte a dicha propuesta.

2.4 Método de Validación

La naturaleza científica de la Ingeniería del Software, al igual que otras disciplinas como la física, la medicina u otras ingenierías, hace necesaria una componente experimental que permita comprobar y validar las teorías propuestas. Así mismo, resulta recomendable experimentar con las técnicas software para conocer en qué escenarios funcionan correctamente, conocer sus limitaciones y descubrir sus posibles puntos de mejora [18, 183]. Para tal fin, Kish [103] definió tres tipos de investigaciones empíricas: encuestas, experimentos e investigaciones simples (o casos de estudio).

Las encuestas son investigaciones empíricas en las que los sujetos del estudio son una muestra representativa de la población a la que pertenecen [164]; los experimentos son las investigaciones en las que las posibles variables perturbadoras han sido aleatorizadas pero pueden controlarse [19]; y los casos de estudio son aquellos en los que no hay aleatoriedad de las variables perturbadoras, que no pueden controlarse, ni representatividad de los sujetos que componen la muestra del estudio [223].

Dado que los casos de estudios nos permiten analizar el comportamiento de una propuesta dentro de su contexto real [223], en esta tesis doctoral empleamos **casos de estudio de laboratorio** para la validación de la propuesta planteada.

De acuerdo con las ideas expuestas en [20, 62, 223], la Figura 2-6 muestra el proceso que se ha seguido en esta tesis doctoral para llevar a cabo la validación de la propuesta presentada mediante el empleo de un caso de estudio. Como cabe esperar, el proceso parte de una propuesta de la que se desea analizar diferentes aspectos de su comportamiento en un contexto real.

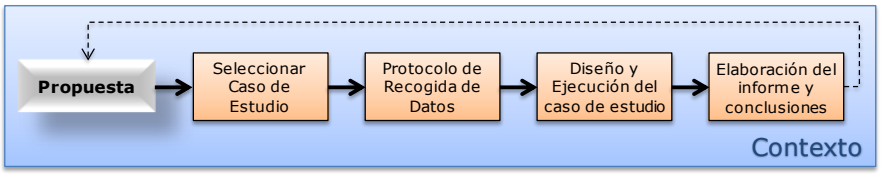

Figura 2-6. Proceso de Validación mediante Casos de Estudio

El primer paso del proceso consiste en la definición de uno o varios casos de estudio. Para ello es necesario conocer en profundidad la propuesta a validar, el contexto de la misma y los aspectos que se desean analizar [20]. Dado que, en la mayoría de los casos, mediante uno o varios casos de estudio no es posible generalizar el comportamiento de la propuesta bajo todas las circunstancias posibles, resulta recomendable seleccionar aquellos casos de estudio que sean suficientemente representativos, completos y que resulten en el mayor aprendizaje posible de su desarrollo y ejecución [187].

A continuación, se debe definir el protocolo para la recogida de datos. Esta fase consiste en la elaboración de un protocolo que proporcione una visión global del proyecto de caso de estudio, procedimientos de campo (credenciales, lugares en los que se ejecutará el caso de estudio, etc.), cuestiones del caso de estudio y una guía para documentar el caso de estudio [223]. Así mismo, en cuanto a la recolección de datos en sí misma, los datos correspondientes al desarrollo de un caso de estudio pueden encontrarse, según Yin [223], en varios tipos de fuentes distintas, aunque no solo limitado a ellas [20, 62]: documentación, registro de archivos, entrevistas, observaciones directas, observaciones participantes y artefactos físicos. Para maximizar el beneficio que proporcionan estas fuentes de datos, Yin [223] propone seguir tres principios en la recolección de datos:

1. Uso de múltiples fuentes de información para corroborar la misma evidencia o fenómeno.
2. Crear una base de datos del caso de estudio.
3. Mantener una cadena de evidencias de forma que un observador externo pueda ser capaz de seguir el camino entre las evidencias identificadas en el caso de estudio.

El tercer paso consiste en el diseño y ejecución del caso de estudio seleccionado. En cuanto al diseño del caso de estudio, Yin [223] propone definir secuencialmente:

1. Las preguntas a las que se quiere responder.
2. Las proposiciones que se analizarán.
3. Las unidades de análisis que se derivan de las preguntas y las proposiciones.
4. La relación existente entre las preguntas y las proposiciones definidas.
5. Los criterios para interpretar los resultados.

Además de proponer los pasos anteriores, Yin [223] establece que para evaluar la calidad de diseño de cualquier caso de estudio se pueden realizar test

que garanticen la validez de construcción, la validez interna, la validez externa y la fiabilidad.

En cuanto a la ejecución del caso de estudio, como su nombre indica consiste en poner en práctica el caso de estudio diseñado y recopilar aquellos datos definidos en la fase anterior, y necesarios para llevar a cabo la siguiente.

Por último, se elabora el informe correspondiente al desarrollo y ejecución del caso de estudio de acuerdo a la guía establecida en la fase de definición del protocolo de recogida de datos. Para llevar a cabo el informe del caso de estudio se deben tener en cuenta aspectos tales como la audiencia a la se dirige, si se trata de un informe que forma parte de un estudio más amplio y el tipo de caso de estudio realizado [20, 62, 187, 223]. Yin propone seis tipos de estructuras diferentes para llevar a cabo el informe del caso de estudio: analítica-lineal, comparativa, cronológica, construcción de teoría, "suspenso" y no secuencial [223]. A partir del informe, se analizan los resultados obtenidos en el desarrollo y ejecución del caso de estudio y se extrae un conjunto de conclusiones que nos permiten analizar la validez de la propuesta inicial. En caso necesario, con la información obtenida podremos ser capaces de corregir y mejorar las debilidades detectadas en la propuesta [223].

La puesta en práctica de este proceso se presenta en el Capítulo 6, donde se detalla el desarrollo de la transformación 'UML2XMLSchema', correspondiente a uno de los módulos de la herramienta M2DAT. Esta transformación ya ha sido desarrollada y probada anteriormente [199, 201, 208], por lo que se dispone de información suficiente para medir y analizar los resultados obtenidos en la ejecución del caso de estudio.

Estado del Arte

Con la llegada de MDE, los modelos y las transformaciones entre dichos modelos se han convertido en artefactos clave en cualquier proceso de desarrollo [14, 23, 102, 176]. Concretamente en el desarrollo de software dirigido por modelos, el software se describe o define mediante diferentes modelos a distintos niveles de abstracción, de forma que los modelos de alto nivel se conviertan (semi-)automáticamente, mediante la ejecución de transformaciones de modelos, en modelos de menor nivel de abstracción [22, 102, 179].

Este escenario, en el que se construye software (semi-)automáticamente a partir de la ejecución de transformaciones de modelos, proporciona las bases necesarias para automatizar otras actividades software. Sin embargo, llevar a cabo esta automatización resulta en aumentar la complejidad de la codificación de dichas transformaciones. Una posible solución para llevar a cabo la automatización de actividades software sin incrementar la complejidad en la codificación de las transformaciones que intervienen en el desarrollo de software dirigido por modelos para por aplicar los principios de MDE al desarrollo de dichas transformaciones [23, 24, 54, 122].

Así, esta tesis doctoral se centra en aumentar el nivel de automatización en la gestión de la trazabilidad, aprovechando las ventajas que ofrece la combinación de estos escenarios.

Por todo ello, en este capítulo se ofrece una visión global del estado del arte en estas cuestiones. Concretamente, este estado del arte se divide en varias secciones: definición detallada de conceptos básicos para el entendimiento del contexto del problema que se aborda en esta tesis doctoral (sección 3.1), revisión de la literatura para analizar propuestas existentes para el desarrollo dirigido por modelos de transformaciones de modelos (sección 3.2) y revisión de la literatura para analizar propuestas centradas en la gestión de la trazabilidad a partir del desarrollo de transformaciones de modelos (sección 3.3).

3.1 Conceptos Previos

A lo largo de la historia de la Ingeniería del Software, los investigadores han tenido problemas a la hora de ponerse de acuerdo en emplear una terminología común [2, 35]. Este problema se hace mayor cuando se trata una disciplina como la Ingeniería Dirigida por Modelos que se encuentra en proceso de expansión y madurez [181, 194, 214]. Por ello, en las siguientes sub-secciones se exponen

algunas definiciones y conceptos previos que es necesario que sean entendidos por el lector antes de estudiar y analizar el cuerpo del conocimiento en el que se basa la propuesta que se presenta en esta tesis doctoral.

3.1.1 Modelado

Como ya se ha podido comprobar en capítulos anteriores, los **modelos** son uno de los conceptos clave en el contexto de la Ingeniería Dirigida por Modelos. Sin embargo, este concepto es compartido por un gran número de disciplinas científicas, por lo que existen numerosas definiciones acerca de **qué es un modelo** [120]. Por ello, en esta sección se especifica qué se entiende por modelo en el contexto de esta tesis doctoral y por extensión, qué se entiende por **metamodelo**.

Así en el contexto de esta tesis doctoral, un modelo es una simplificación de un sistema, construido con objetivo pensado. Además, los modelos deben ser capaces de responder a las preguntas sobre el dominio del sistema del mismo modo que responde dicho sistema [26].

La definición anterior está muy relacionada con la forma en la que las ingenierías tradicionales hacen uso de los modelos. En dichas disciplinas los modelos son definidos como paso previo a la construcción de los sistemas, actuando como planos o mapas y proporcionan una especificación que permite describir el comportamiento y la estructura del sistema. Del mismo modo, la Ingeniería del Software ha adoptado el **modelado** como una de sus prácticas más comunes con el objetivo de definir el sistema software a construir y especificar su estructura y comportamiento [168, 184]. En este sentido, se adopta la definición de **modelo software** propuesta por Kleppe *et al.* en [106]: "*un modelo es una descripción (o parte) de un sistema en un lenguaje bien-formado*". La Figura 3-1 (a) ilustra la definición anterior.

Un lenguaje de modelado bien-formado es aquel que define los elementos que pueden formar parte de un modelo así como las relaciones que pueden darse entre estos elementos. Generalmente, esta información es recogida por el **metamodelo** del lenguaje. Así pues, un metamodelo no es más que un modelo que permite establecer las condiciones en las cuales pueden expresarse los modelos válidos en un determinado lenguaje [14, 43, 178]. En algunos casos, un metamodelo es creado conforme al lenguaje definido por sí mismo, en tal caso se denomina **metametamodelo**. En la Figura 3-1 (b) se representa la jerarquía de modelado, que muestra la relación existente entre los metametamodelos, los metamodelos y los modelos terminales [26]. De acuerdo a esta jerarquía, un modelo terminal se define de acuerdo a las reglas definidas en otro modelo

(metamodelo), que a su vez es definido conforme a otro modelo (metametamodelo), el cual se define de acuerdo a sí mismo. En la Figura 3-1 (c) se representa que un metamodelo es básicamente un modelo que se describe de acuerdo a otro metamodelo [106].

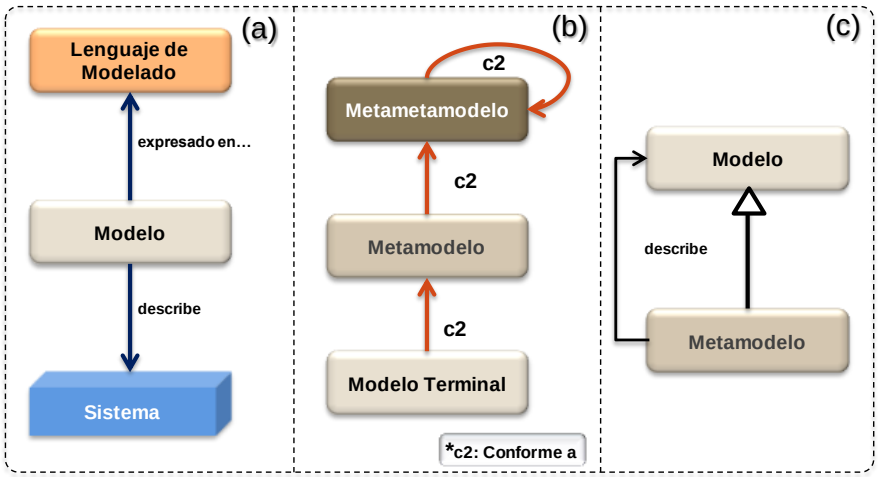

Figura 3-1. a) Definición de Modelo (Kleppe *et al.*), b) Jerarquía de Modelado y c) Relación entre los conceptos 'Modelo' y 'Metamodelo'

3.1.2 Transformación de Modelos

En la sección anterior hemos visto qué es un modelo y para qué sirve en el contexto de esta tesis doctoral. En esta sección veremos cómo se relacionan los modelos entre sí.

Trabajar con modelos interrelacionados implica invertir grandes esfuerzos a la hora de realizar ciertas tareas relacionadas con la gestión de dichos modelos como por ejemplo el mantenimiento, el refinamiento o la refactorización. Es habitual que dichas tareas se lleven a cabo mediante procesos automatizados que toman uno o varios modelos de origen o entrada y, a partir de la ejecución de un conjunto de reglas, producen uno o varios modelos de salida o destino. Estos procesos automatizados son denominados **transformaciones de modelos** [182].

En el contexto de la Ingeniería Dirigida por Modelos, al igual que Sendall y Kozaczynski [182], un gran número de autores han propuesto diversas definiciones para describir qué es una transformación de modelos [199]. A modo de ejemplo se puede mencionar la definición que propone la guía de MDA [153] (pág. 17): "*una transformación de modelos es el proceso de convertir un modelo en otro modelo del mismo sistema*".

En el contexto de esta tesis doctoral, se adopta la definición basada en metamodelos propuesta por Bézivin [22] y esbozada anteriormente por Lemesle [127]. La Figura 3-2 ilustra esta idea.

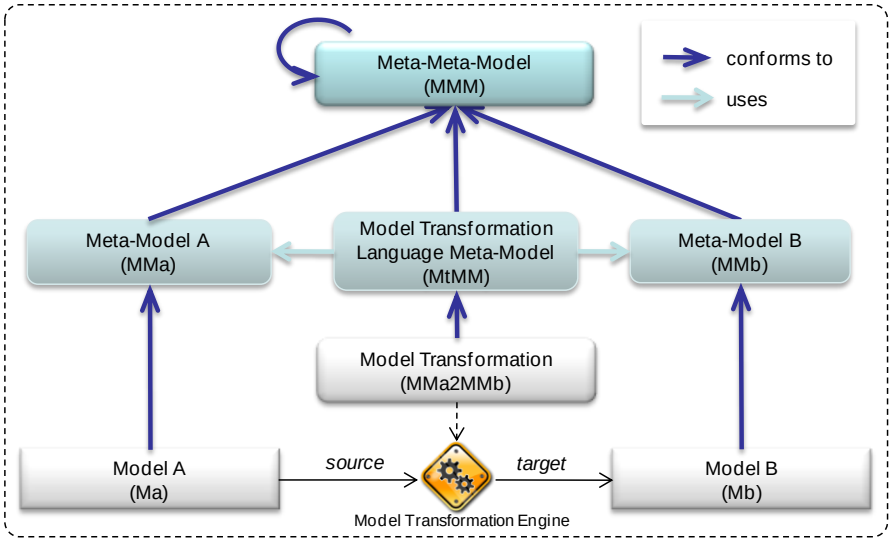

Figura 3-2. Proceso de Transformación de Modelos

En este proceso de transformación de modelos, el principal elemento es el metametamodelo (MMM), que proporciona un conjunto de abstracciones que permiten la definición de nuevos metamodelos. Así, en la Figura 3-2 se puede ver que se han definido los metamodelos origen (MMa) y destino (MMb) de la transformación como instancias de dicho metametamodelo, es decir, MMa y MMb son conformes a MMM.

Para realizar la transformación entre cualquier modelo origen conforme a MMa (Ma) y cualquier modelo destino conforme a MMb (Mb) es necesario definir un conjunto de reglas que relacionen los elementos de dichos metamodelos. Estas reglas son recogidas en una transformación de modelos (MMa2MMb). Dicha transformación será ejecutada por el motor de transformación, permitiendo generar modelos destino a partir de cualquier modelo conforme al metamodelo origen.

Además, si el conjunto de reglas y restricciones que guían la construcción de la transformación es recogida en un metamodelo (MtMM), cualquier transformación de modelos puede ser tratada como un modelo, es decir podremos generarla, transformarla o validarla como cualquier otro modelo del sistema [21, 24]. Esto ofrece una serie de ventajas, por ejemplo emplear transformaciones HOT

(*Higher Order Transformation*). Estas transformaciones de modelos son aquellas en las que su entrada y/o salida son transformaciones en sí mismas [190].

3.1.3 Ingeniería Dirigida por Modelos

Los conceptos descritos en las secciones anteriores son la base o los principios en los que se asienta la **Ingeniería Dirigida por Modelos**. Este paradigma propone centrarse en el **espacio del problema** en lugar de centrarse en la codificación (espacio de la solución). Así, las principales características de este paradigma es el papel central que juegan los **modelos** en el proceso de desarrollo y las **transformaciones** entre dichos modelos [22, 64, 102, 176, 179, 214].

La Ingeniería Dirigida por Modelos o MDE surge como un paso natural en la tendencia histórica de elevar el nivel de abstracción al que se diseña y construye el software. En este sentido, se recuerda cómo surgieron los primeros lenguajes de programación (por ejemplo el lenguaje ensamblador), que alejaban al programador de las complejidades asociadas al desarrollo en código máquina o cómo estos lenguajes dejaron paso a otros como C o Fortran [176, 214].

En el contexto de MDE, se identifican diferentes disciplinas que son clasificadas dependiendo de su objetivo. La Figura 3-3 muestra esta clasificación.

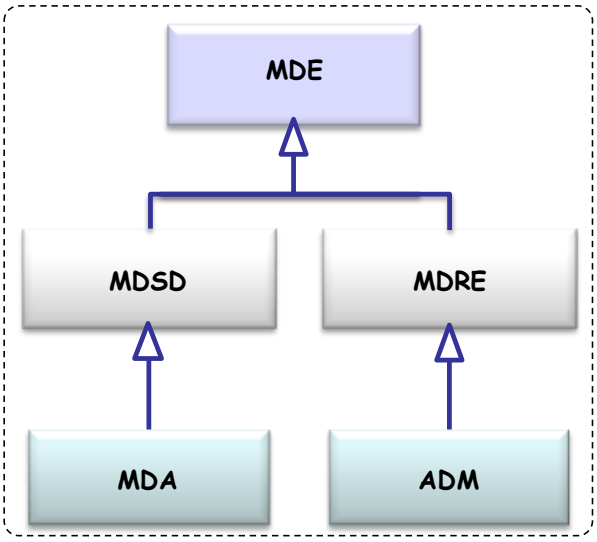

Figura 3-3. Disciplinas MDE

En la rama izquierda de la figura se encuentran aquellas disciplinas que aplican los principios de MDE a la Ingeniería del Software Directa: Desarrollo de

Software Dirigido por Modelos (*MDSD*) [186] y Arquitectura Dirigida por Modelos (*MDA*) [153]. En la rama de la derecha se encuentran las disciplinas que aplican los principios de MDE a la Ingeniería del Software Inversa: Ingeniería Inversa Dirigida por Modelos (*MDRE*) [173] y Modernización Dirigida por la Arquitectura (*ADM*) [150]. Dado que esta tesis doctoral se enmarca en el contexto de las dos primeras (MDSD y MDA), estas serán descritas en detalle en las siguientes secciones.

3.1.4 Desarrollo de Software Dirigido por Modelos

El Desarrollo de Software Dirigido por Modelos (DSDM) es la aplicación de los principios de MDE al desarrollo de software [85, 179, 186], por tanto puede entenderse como la rama de MDE centrada en el desarrollo software.

El DSDM no solo tiene por objetivo incrementar el nivel de abstracción, sino que además propone aumentar el nivel de **automatización** en el desarrollo de software. Para ello, propone especificar los sistemas mediante modelos a diferentes niveles de abstracción, de forma que los modelos de más alto nivel se convierten en otros de menor nivel hasta alcanzar modelos ejecutables mediante la generación de código [67] o la interpretación de modelos [1, 47, 60, 129, 222]. En este escenario, la clave para aumentar el nivel de automatización es el empleo de transformaciones de modelos en el paso de un modelo a otro modelo de menor nivel de abstracción.

Como muestra la Figura 3-4, cada paso de un proceso de DSDM consiste en aplicar un conjunto de reglas de transformación sobre uno o varios modelos de origen para producir uno o varios modelos de destino con menor nivel de abstracción.

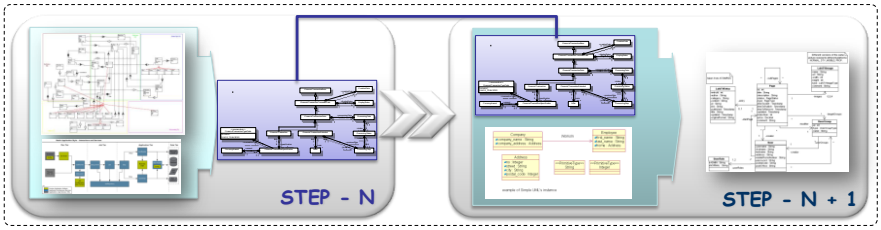

Figura 3-4. Simplificación de un proceso de DSDM

3.1.5 Arquitectura Dirigida por Modelos

Una de las interpretaciones más fieles de los principios de MDE es la **Arquitectura Dirigida por Modelos** (MDA, *Model-Driven Architecture)* [66, 69,

84, 106, 139, 153, 180, 219], de hecho Favre en [64] considera que MDA es la encarnación de MDE.

MDA, que fue presentado en el año 2001 por el consorcio OMG, es un marco de desarrollo software alineado con los principios de MDE (rol de los modelos y automatización). El principal objetivo de MDA es separar la especificación de un sistema concreto de los detalles relacionados con el uso de la plataforma por parte de dicho sistema. Para alcanzar este objetivo se centra en la definición de modelos formales como elementos de primera clase para el diseño e implementación de sistemas y la definición de transformaciones (semi-)automáticas entre los modelos [153].

MDA define tres grandes grupos de modelos, de acuerdo a su nivel de abstracción. Los requisitos del sistema se modelan mediante modelos **CIM** (*Computation Independent Model*). Los modelos **PIM** (*Platform Independent Model*) sirven para representar la funcionalidad y estructura del sistema, omitiendo los detalles tecnológicos asociados a las características específicas de la plataforma que soportará al sistema. Finalmente, las especificaciones descritas a nivel PIM son adaptadas a los detalles de las plataformas concretas por medio de los modelos **PSM** (*Platform Specific Model*), a partir de los cuales se genera el código del sistema. En el nivel PSM es posible definir modelos que representan distintos grados de abstracción, pudiéndose agrupar en modelos que describen elementos comunes y en modelos que contienen elementos específicos de una plataforma concreta. En este caso, se consideran modelos **PDM** (*Platform Dependent Models*) [74, 153, 167].

Un proceso de desarrollo basado en MDA comienza con la definición de los modelos correspondientes a la capa de abstracción CIM (en la práctica, en algunas ocasiones se omite). Mientras que estos modelos definen el dominio del negocio, los niveles inferiores especifican el dominio del sistema. Por ello, no es posible establecer una transformación directa entre los requisitos del sistema, especificados en la capa CIM, y los artefactos representados en las capas inferiores. En cambio, MDA sí define transformaciones concretas PIM-PSM y PSM-PDM entre los artefactos de dichas capas.

La Figura 3-5 ilustra una simplificación de las relaciones entre las capas de abstracción definidas por MDA así como las interacciones existentes entre modelos de los diferentes niveles.

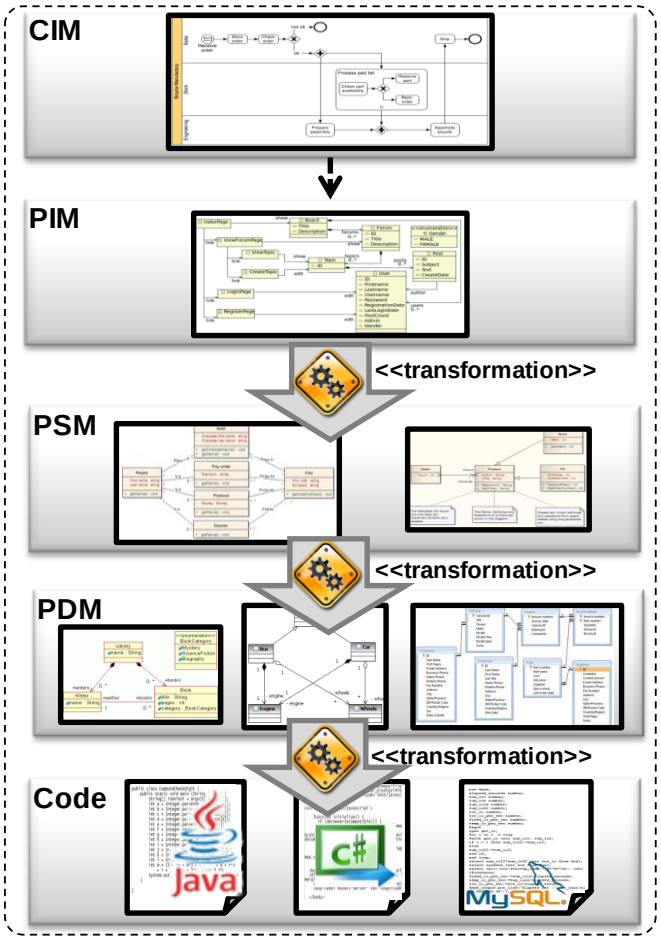

Figura 3-5. Niveles de abstracción MDA

Además de los niveles de abstracción mencionados, la OMG propuso emplear un conjunto de estándares para trabajar con MDA. Entre estos estándares se encuentran: UML [156], MOF [154], OCL [155], XMI [157] y QVT [151].

3.1.6 Trazabilidad

Tradicionalmente, en el contexto de la Ingeniería del Software el término **trazabilidad** ha ido asociado a las tareas de gestión de requisitos [5, 169]. De hecho, autores como Gotel y Finkelstein [76] definen la trazabilidad como: *la capacidad para describir y seguir la vida de un requisito, tanto hacia delante como hacia atrás*. En este contexto, el principal objetivo de la gestión de la

trazabilidad es asegurar que el software desarrollado cumple con las expectativas del usuario (o cliente), de forma permita comprobar si cada requisito ha sido satisfecho o si cada componente del sistema satisface al menos un requisito [11, 169].

Sin embargo, este concepto ha evolucionado a lo largo del tiempo y actualmente la trazabilidad es entendida como las relaciones existentes entre artefactos implicados en un ciclo de vida software [5]. Así, una de las definiciones más aceptadas en este sentido es la propuesta por la IEEE [89], que la define como: *el grado de relación que puede establecerse entre dos o más productos de un proceso de desarrollo, especialmente productos que tienen relaciones predecesor-sucesor o maestro-subordinado con otro producto.*

La información obtenida a partir de la gestión de la trazabilidad, facilita el desempeño de otras actividades que forman parte del ciclo de vida software. Así, puede utilizarse para la toma de decisiones, el análisis del impacto del cambio, el mantenimiento del sistema, la validación y verificación de los requisitos [5, 37, 90, 107, 138, 142, 169, 221]. Además, algunos estándares centrados en la mejora de los procesos software como el ISO12207 consideran a la gestión de la trazabilidad como una medida de calidad [90].

En cualquier desarrollo de software dirigido por modelos, la gestión de la trazabilidad cobra aún más importancia debido a la naturaleza de estos desarrollos. Como se ha mencionado anteriormente, el sistema se desarrolla a partir de la especificación de un modelo de alto nivel que se transforma (semi-)automáticamente en otros modelos de menor nivel de abstracción a partir de la definición de un conjunto de reglas de transformación. Dado que los desarrolladores de los modelos iniciales y de las transformaciones no tienen por qué ser los mismos e incluso pueden ser diferentes de los que ejecutan dichas transformaciones, la trazabilidad es un concepto crucial en este tipo de desarrollos ya que proporciona información acerca de cómo y porqué son creados los artefactos del sistema [112, 149].

Aprovechando las características mencionadas del desarrollo de software dirigido por modelos, esta tesis doctoral se centra en la identificación de relaciones de trazabilidad a partir de las relaciones existentes en las transformaciones que guían el proceso de desarrollo y en la generación de enlaces de traza entre las instancias de elementos implicados en dichas transformaciones.

En este punto, es necesario establecer la diferencia que existe, en el contexto de esta tesis doctoral, entre **relación de trazabilidad** y **enlace de traza**: mientras que las relaciones de trazabilidad son aquellas que se definen entre tipos

de elementos (por ejemplo, clases o elementos de un metamodelo), los enlaces de traza (o simplemente trazas) son las instancias de dichas relaciones (por ejemplo, entre objetos o elementos de un modelo). La Figura 3-6 ilustra estos conceptos.

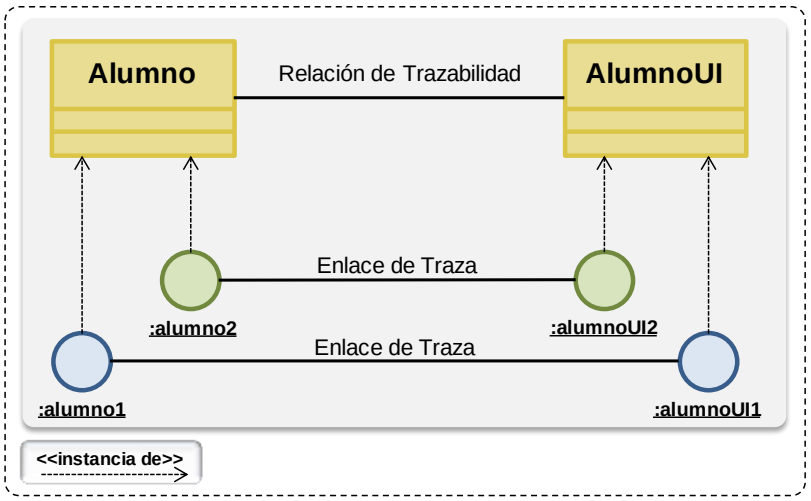

Figura 3-6. Conceptos 'relación de trazabilidad' y 'enlace de traza'

3.2 Propuestas para el Desarrollo Dirigido por Modelos de Transformaciones de Modelos

Una vez que las tecnologías que dan soporte a las tareas de propias de MDE han alcanzado ciertas cotas de madurez [214], los investigadores han comenzado a emprender nuevos proyectos cada vez más extensos y complejos. Como consecuencia, las transformaciones de modelos que se desarrollan en el contexto de estos proyectos son también más extensas y complejas, de forma que sería recomendable seguir un proceso de desarrollo propio que permita asegurar y validar su construcción [63, 75]. Dado que estas transformaciones son construidas en el contexto de un proceso dirigido por modelos, parece lógico y deseable aplicar los principios del DSDM para su desarrollo [23, 24, 54, 82, 97, 122, 149]. Así, las transformaciones de modelos deberían especificarse como modelos alto nivel y ser refinados automáticamente en modelos más detallados (menor nivel de abstracción) hasta obtener el código de la transformación o modelos ejecutables.

Por todo ello, en esta sección se realiza un estudio de la literatura para identificar y analizar aquellas propuestas que ponen en práctica este planteamiento. Es necesario apuntar que el estudio de la literatura que se hace en

esta sección no corresponde a un proceso de revisión sistemática completo, sino que se trata de una evolución y mejora del estado del arte presentado en [30].

En las siguientes sub-secciones se presentan los objetivos o cuestiones de investigación a las que se quiere dar respuesta con este estudio de la literatura (sección 3.2.1), los criterios que se emplean para evaluar y comparar cada una de las propuestas identificadas en la literatura (sección 3.2.2), los datos que se extraen de cada propuesta (sección 3.2.3), el análisis detallado de cada una de ellas (sección 3.2.4) y por último, se presentan las conclusiones obtenidas del estudio de la literatura (sección 3.2.5).

3.2.1 Cuestiones de Investigación

El principal objetivo consiste en identificar y analizar propuestas centradas en aplicar los principios del desarrollo dirigido por modelos a la construcción de transformaciones de modelos. Para dar respuesta a este objetivo principal, una vez que se hayan identificado y analizado las propuestas correspondientes, en la sección 3.2.5 se dará respuesta a cada una de las siguientes **cuestiones de investigación (CI)** que se proponen:

- **CI-1.** ¿A qué niveles de abstracción se propone especificar las transformaciones de modelos?

  Las propuestas a analizar están centradas en el DDM de transformaciones de modelos y uno de los pilares del desarrollo dirigido por modelos es especificar el sistema (en este caso, una transformación) mediante modelos a diferentes niveles de abstracción. Por tanto, parece interesante conocer a qué niveles de abstracción es factible modelar las transformaciones.

- **CI-2.** ¿Qué ideas se proponen para automatizar el proceso de desarrollo de las transformaciones?

  Otro de los principios básicos del DDM es la automatización entre los diferentes niveles de abstracción empleados para especificar el sistema, por ejemplo a través de transformaciones de modelos. Así, en esta segunda pregunta se analiza cómo se automatiza el paso entre los modelos de los distintos niveles de abstracción y la generación del código que implementa la transformación (si aplica).

- **CI-3.** ¿Existen propuestas metodológicas centradas en el DDM de transformaciones de modelos que propongan o sugieran ideas para la gestión de las trazas a partir del modelado de dichas transformaciones?

El principal objetivo de esta tesis doctoral es dar soporte para la generación de enlaces de traza a partir del modelado de transformaciones, por ello, en esta cuestión de investigación se analiza si las propuestas para el DDM de transformaciones de modelos ofrecen ideas o métodos para la gestión de la información de trazabilidad a partir del modelado de dichas transformaciones.

- **CI-4.** ¿Existen herramientas que den soporte tecnológico al DDM de transformaciones de modelos?

  En esta cuarta pregunta se analiza si las propuestas metodológicas han sido puestas en práctica, es decir, si existen herramientas para dar soporte al DDM de transformaciones de modelos. En caso afirmativo, se analiza cómo se ofrece dicho soporte.

- **CI-5.** ¿Cuáles son las limitaciones del estado del arte en el DDM de transformaciones de modelos?

  Finalmente, en la última cuestión se combinarán las respuestas a las preguntas anteriores (CI-1, CI-2, CI-3 y CI-4) para identificar problemas o limitaciones en el estado del arte actual del tema objetivo.

3.2.2 Criterios de Evaluación

Una vez que se han seleccionado los estudios que nos ayudarán a dar respuesta a las preguntas anteriores, sería deseable disponer de mecanismos que nos permitan evaluar y comparar dichos estudios. Por tanto, de acuerdo a la guía para realizar una revisión sistemática propuesta por Kitchenham y Charters [105], para evaluar la calidad de cada una de las propuestas identificadas y proporcionar una comparativa cuantitativa en ellas, en esta sección se define un conjunto de **criterios de evaluación (CE)** y un conjunto de **valores** que pueden adoptar dichos criterios.

Para facilitar la comparativa entre las propuestas, los valores que se les pueden asignar para cada uno de los criterios de evaluación son: Sí (S) = 1; Parcial = 0.5; y No (N) = 0. Los criterios de evaluación para las propuestas centradas en el DDM de transformaciones de modelos son:

- **CE-1**. ¿La propuesta metodológica propone especificar la transformación en los niveles PIM y PSM definidos por MDA [153] antes de ser serializada en código?

  Valores: Sí (S), la propuesta define explícitamente que una transformación de modelos debe ser especificada al menos en estos dos niveles o

equivalentes; Parcial (P), la propuesta contempla que la transformación sea definida, al menos, a nivel PIM o PSM o equivalentes; No (N), la propuesta no contempla el modelado de la transformación a ninguno de estos niveles.

- **CE-2.** ¿La propuesta metodológica sugiere cómo automatizar el paso entre los diferentes niveles de abstracción a los que se especifica la transformación?

  Valores: S, proporciona métodos para automatizar o semi-automatizar la conversión de modelos a distintos niveles de abstracción, por ejemplo, mediante transformaciones entre dichos modelos; P, no indica cómo automatizar el proceso, sin embargo, considera que es una característica deseable; N, el paso entre los niveles es totalmente manual.

- **CE-3.** ¿La propuesta metodológica especifica cómo generar el código que implementa la transformación a desarrollar?

  Valores: S, indica cómo llevar a cabo la serialización de los modelos de bajo nivel que describen la transformación; P, no especifica cómo ponerlo en práctica, pero identifica a la generación de código como una característica deseable; N, no considera la generación del código de la transformación.

- **CE-4.** ¿Propone algún mecanismo para visualizar los modelos de transformación?

  Valores: S, propone el desarrollo de un editor o visualizador *ad-hoc*, considerando la naturaleza de los modelos de transformación; P, la representación de los modelos puede no ser la más óptima, ya que considera la visualización a través de métodos genéricos para cualquier tipo de modelo, como por ejemplo el editor en forma de árbol que ofrece EMF [188]; N, la propuesta no considera la visualización de los modelos o no sugiere ninguna forma de representación.

- **CE-5.** ¿La propuesta metodológica proporciona o sugiere mecanismos de validación de los modelos de transformación, de forma que los errores no se propaguen a los modelos de menor nivel de abstracción?

  Valores: S, indica cómo deberían validarse los modelos de transformación para asegurar que son correctos; P, no indica cómo deben validarse los modelos, pero considera que es una característica deseable; N, no considera en absoluto la validación de modelos.

- **CE-6.** ¿La especificación a alto nivel de la transformación es independiente de cualquier lenguaje de transformación concreto?

Valores: S, la definición de la transformación a alto nivel es independiente de cualquier lenguaje de transformación; P, la especificación de la transformación no es completamente independiente, o es dependiente de un lenguaje concreto pero consideran deseable que fuera independiente; N, la especificación de la transformación es totalmente dependiente de un lenguaje concreto.

- **CE-7.** ¿La propuesta metodológica considera la gestión de la trazabilidad a partir del DDM de transformaciones de modelos?

  Valores: S, propone cómo gestionar la trazabilidad a partir del desarrollo de transformaciones; P, no indica cómo, pero considera que sería posible gestionar trazabilidad a partir de las transformaciones; N, no considera en absoluto la gestión de la trazabilidad.

- **CE-8.** ¿Proporciona alguna implementación, herramienta, o soporte tecnológico?

  Valores: S, proporciona una herramienta o marco de trabajo completo para dar soporte a la propuesta metodológica; P, proporciona una implementación parcial de la propuesta; N, se trata de una propuesta puramente teórica que no ofrece implementación para ponerla en práctica.

3.2.3 Extracción de Datos

Para facilitar el análisis y evaluación de las propuestas que forman el estado del arte y dar respuesta a las preguntas anteriores es necesario recopilar cierta información de dichas propuestas. Así, para las propuestas centradas en el DDM de transformaciones de modelos se ha decidido extraer la siguiente información:

- Título, autores, resumen y año de publicación de cada estudio identificado (perteneciente a la propuesta).
- Niveles de abstracción que considera para el modelado de la transformación.
- Si automatiza el DDM de transformaciones de modelos.
- Si proporciona o sugiere cómo sería posible generar el código que serializa los modelos de la transformación.
- Si sugiere cómo deberían visualizarse los modelos que especifican la transformación.

- Si considera o recomienda la validación de los modelos que especifican la transformación.
- Si considera la gestión de la trazabilidad a partir del DDM de transformaciones de modelos.
- Si ofrece soporte tecnológico y cómo lo hace (si aplica).

Estos datos se extraen en la siguiente sub-sección, donde se analiza en detalle cada una de las propuestas por separado.

3.2.4 Análisis de las propuestas

Una vez que se ha definido la información que se va a extraer y evaluar de cada propuesta, en esta sub-sección se procede a, por un lado, analizar de forma individual las propuestas identificadas en el estado del arte y por otro, comparar dichas propuestas de acuerdo a los criterios de evaluación definidos en la sección 3.2.2.

3.2.4.1 J. Bézivin *et al.*

Según los autores de la propuesta presentada en [25], en los desarrollos MDA la mayor parte de las transformaciones eran codificadas en un lenguaje propietario e inestable. Sin embargo, consideran que deberían ser tratadas como cualquier artefacto software y por tanto, deberían analizarse, diseñarse, implementarse, probarse y mantenerse de forma independiente a cualquier plataforma.

Por todo ello, en [25] se centran en aplicar los principios de MDA al desarrollo de las transformaciones de modelos. Proponen especificar dichas transformaciones mediante el empleo de dos niveles de abstracción que se corresponden con los niveles PIM y PSM definidos por MDA: p*latform-independent transformations* (PIT) y *platform-specific transformations* (PST). En este contexto se entiende 'plataforma' como la herramienta que permite realizar la especificación, diseño y ejecución de las transformaciones.

A partir de la definición de estos niveles de abstracción, proponen un proceso de desarrollo de transformaciones que consiste en realizar los siguientes pasos: definir la transformación de forma independiente a la plataforma y trasladar dicha definición independiente a una dependiente de plataforma. Este proceso de desarrollo de transformaciones puede llevarse a la práctica de forma ortogonal a cualquier proceso MDA, es decir, en una dimensión se puede disponer de los niveles PIM y PSM y en otra los niveles PIT y PST definidos.

En esta línea de investigación también se encuentra [24]. En este trabajo, los autores Bézivin, Büttner, Gogolla, Jouault, Kurtev y Lindow realizan una comparativa entre el uso de codificar las transformaciones de modelos y modelarlas conforme a metamodelos de transformación genéricos, es decir, a nivel PIT. A partir de estos modelos de transformación se podrían obtener diferentes transformaciones de modelos, dependiendo de la plataforma de implementación utilizada. Además, en este trabajo los autores presentan algunas de las ventajas derivadas del modelado de las transformaciones de modelos como pueden ser: la definición de las transformaciones a alto nivel, transformar los modelos que especifican las transformaciones, permitir la validación de las transformaciones, etc.

A continuación, se presentan algunos de los datos extraídos del análisis de la propuesta que permitirán dar valor a los criterios de evaluación definidos en la sección 3.2.2:

- **Niveles de abstracción a los que se especifica la transformación.** Los autores definen dos niveles de abstracción (PIT y PST) que se corresponden con los niveles PIM y PSM definidos por MDA. Se trata de definir la transformación de forma independiente y dependiente a la plataforma, que en este caso es entendida como la herramienta que permite realizar la especificación, diseño y ejecución de las transformaciones. Para implementar los modelos de transformación a nivel PIT los autores proponen el uso de UML, ya que debido a sus mecanismos de estructuración estática (paquetes, clases y métodos), los autores consideran que UML tiene el poder de modelar transformaciones independientemente de la complejidad de las mismas. Para el modelado de las transformaciones a nivel PST, proponen el uso de la sintaxis definida por las plataformas en las que se desee llevar a cabo la implementación.
- **Nivel de automatización.** El proceso de desarrollo propuesto consiste en el modelado de la transformación a nivel PIT y la conversión de este modelo a otro a nivel PST. Los autores asumen que esta conversión se llevará a cabo mediante una transformación de modelos.
- **Generación de código.** La propuesta contempla la necesidad de obtener el código que implementa dicha transformación en el lenguaje seleccionado por el desarrollador, aunque no se especifica cómo se obtendrá este código ni cómo se debería realizar la implementación de la misma.
- **Visualización de modelos.** La propuesta no aborda cómo se deberían visualizar los modelos que especifican las transformaciones.

- **Validación de modelos.** Los autores indican que los modelos, como cualquier otro modelo, pueden ser validados. Para ello, proponen el empleo de UML y herramientas de validación de OCL. Sin embargo, no se detalla cómo llevar a cabo dicha validación.
- **Gestión de la trazabilidad.** La propuesta no considera la gestión de la trazabilidad a partir del modelado de las transformaciones.
- **Soporte tecnológico.** En las conclusiones de [25], los autores indican que, en el marco del programa de investigación CARROLL, están trabajando en el desarrollo de un conjunto de herramientas de código abierto para dar soporte a la propuesta metodológica presentada. Sin embargo, no se ha podido acceder a dichas herramientas y el programa CARROLL parece haber terminado, ya que la última actualización de su sitio web (http://www.carroll-research.org) fue en el año 2005.

3.2.4.2 M. Didonet del Fabro

En su tesis doctoral [54], Didonet del Fabro propone una solución genérica para la gestión de las relaciones existentes entre los elementos de diferentes modelos. Esta propuesta o solución está basada en los principios de MDE y se denomina *model weaving*. El alcance de la tesis presentada por Didonet del Fabro es el estudio de los aspectos conceptuales, prácticos y de aplicación en relación con modelos de *weaving*.

Una de las principales aportaciones de este trabajo al estado del arte es la definición semi-automática de transformaciones de modelos a partir de modelos de *weaving*. En dichos modelos se establecen las relaciones existentes entre los elementos del modelo de entrada y los elementos del modelo de salida. A partir de dichas relaciones y mediante la ejecución de una transformación de modelos tipo HOT denominada *TransfGen*, se obtiene el modelo de la transformación dependiente de las características de un lenguaje concreto (equivalente al nivel PDM) y posteriormente el código que implementa dicha transformación.

Para validar la aproximación propuesta, el autor presenta una herramienta genérica y adaptable denominada *ATLAS Model Weaver* (AMW). AMW es una herramienta que, implementada sobre el entorno de trabajo Eclipse, proporciona un editor gráfico de modelos de *weaving* para permite definir y visualizar las relaciones existentes entre los elementos de uno o varios modelos de entrada y uno o varios modelos de salida. Además, incorpora la implementación de la transformación *TransfGen* que permite generar, a partir de los modelos de *weaving*, modelos conformes al metamodelo del lenguaje de transformación ATL

[98]. Para serializar los modelos de transformación conformes al metamodelo de ATL en código ATL, la herramienta dispone de un extractor de código desarrollado con la herramienta TCS (*Textual Concrete Syntax,* [100]).

En cuanto a los datos extraídos del análisis de la propuesta:

- **Niveles de abstracción a los que se especifica la transformación.** El autor no menciona que el nivel de abstracción al que se especifica la transformación se corresponda con ninguno de los niveles definidos por MDA. Sin embargo, propone definir las relaciones que darán lugar a la transformación un modelo independiente del lenguaje que implementará la transformación. Por tanto, se puede considerar equivalente al nivel PIM definido por MDA. Dicho modelo, es convertido en un modelo conforme al metamodelo del lenguaje que implementa la transformación, este modelo describe cómo será la transformación en términos de la sintaxis concreta de un lenguaje, por ello no sería equivalente al nivel PSM, sino al nivel PDM.
- **Nivel de automatización.** El autor propone el empleo de una transformación de tipo HOT para convertir un modelo de *weaving* en un modelo conforme al metamodelo de un lenguaje de transformación concreto. De esta forma, las relaciones definidas en el modelo de *weaving* entre los elementos de los metamodelos de entrada y salida serán transformadas de forma automática a reglas de transformación. En cuanto a la implementación de dicha transformación HOT, es necesario destacar que la herramienta presentada solo da soporte para la generación de modelos de transformación conformes al lenguaje ATL.
- **Generación de código.** La generación de código en lenguaje ATL es definida y soportada mediante un extractor implementado con la herramienta TCS. Mediante esta funcionalidad, es posible obtener código ATL a partir de un modelo conforme al metamodelo definido por ATL.
- **Visualización de modelos.** AMW, la herramienta que da soporte a la propuesta presentada, permite la definición y visualización gráfica de las relaciones existentes a alto nivel. El editor propuesto, compuesto por varios paneles de visualización, permite identificar las relaciones existentes entre los elementos de los metamodelos cualesquiera de entrada y salida.
- **Validación de modelos.** No aborda la validación de los modelos de transformación.
- **Gestión de la trazabilidad.** El autor presenta cómo los modelos de *weaving* pueden ser usado para visualizar y almacenar trazas que se generan a partir de

la ejecución de una transformación de modelos. Esta aportación tiene algunos puntos en común con la propuesta que se presenta en esta tesis doctoral.

- **Soporte tecnológico.** Esta propuesta ofrece soporte tecnológico mediante la herramienta AMW, desarrollada sobre el entorno Eclipse. Dicha herramienta permite la definición de modelos de *weaving* para establecer las relaciones a alto nivel. A partir de dichos modelos, mediante la ejecución de una transformación HOT, se genera un modelo de la transformación conforme al metamodelo del lenguaje de transformación ATL que posteriormente, usando un extractor implementado con TCS, permite la generación del código de la transformación en términos de la sintaxis del lenguaje ATL.

3.2.4.3 E. Guerra *et al.*

En los trabajos [82] y [83], los autores Guerra, De Lara, Kolovos, Paige y Dos Santos presentan una familia de lenguajes de modelado, denominado *transML*, que abarca el ciclo de vida completo del desarrollo de transformaciones: desde la definición de los requisitos hasta las pruebas. Esta propuesta sigue un enfoque basado en MDE, por tanto, según sus autores permite el desarrollo parcialmente automático de los modelos que describen la transformación y del código que serializa dichos modelos. Como se defiende en esta tesis doctoral, la aplicación de los principios de MDE al desarrollo de transformaciones permite crear transformaciones para cualquier lenguaje de implementación de transformaciones.

Esta propuesta surge para proporcionar un método de desarrollo de transformaciones basado en los principios de la ingeniería, de forma que las transformaciones de modelos puedan ser empleadas en entornos industriales o comerciales. Así, definen que el proceso de desarrollo de las transformaciones debería incluir las siguientes fases (siempre que la complejidad de la transformación lo requiera): requisitos, análisis, diseño de la arquitectura, diseño a alto nivel, diseño detallado, codificación y pruebas. Según los autores, la notación empleada en cada una de estas fases tiene que tener en cuenta las especificaciones del desarrollo.

Para llevar a cabo la fase de requisitos, los autores indican que es posible emplear cualquier técnica propia de la Ingeniería de Requisitos, sin embargo, *transML* propone una representación de los requisitos en forma de diagramas, similares a los diagramas SysML (http://www.omg.org/spec/SysML/1.1/). Para ello, definen un metamodelo propio de captura de requisitos.

Para la fase de análisis, proponen una adaptación de las técnicas de la Ingeniería del Software, de forma que para cada requisito aprobado se definan ejemplos para la transformación, denominados *casos de transformación* (similares a los casos de uso en UML).

Para la definición de la arquitectura de la transformación se incluye un lenguaje de modelado que permite la descomposición modular de las transformaciones en unidades funcionales.

En cuanto al diseño, la propuesta define dos niveles: diseño a alto nivel y diseño a bajo nivel. El diseño a alto nivel se realiza por medio de la definición de un diagrama de mapeo (*mapping diagram*), en el que se especifican las relaciones existentes entre los elementos de los diferentes modelos que participan en las transformaciones, omitiendo los detalles acerca de cómo llevar a cabo la transformación. El diseño a bajo nivel consiste en definir cómo debe implementarse la transformación, haciendo una separación entre su estructura y su comportamiento. Para modelar ambos niveles de diseño, la propuesta proporciona dos metamodelos que permiten representar las características de cada uno de los niveles de diseño.

Para la fase de implementación, *transML* no incluye ningún lenguaje propio de implementación de las transformaciones, pero brinda mecanismos para utilizar los lenguajes de transformación existentes (QVT, ATL, ETL, etc.). Siguiendo la filosofía MDE, se puede generar el código para diferentes plataformas a partir de los diagramas especificados en los niveles anteriores. En cambio, para llevar a cabo la fase de pruebas sí que proporciona un lenguaje propio que permite la definición de casos de prueba.

Para soportar la propuesta metodológica, los autores han implementado los metamodelos definidos en cada una de las etapas por medio de EMF, además han definido una serie de transformaciones de modelos y varios generadores de código que permiten la transformación entre los diagramas de las diferentes etapas. Por medio de estas transformaciones, se automatiza parcialmente el proceso presentado. Estas transformaciones se han implementado con el lenguaje ETL y para la generación de código se utiliza EGL.

En cuanto a los datos extraídos del análisis de la propuesta:

- **Niveles de abstracción a los que se especifica la transformación.** En la etapa de diseño, la propuesta identifica dos niveles a los que se debería describir el diseño de la transformación: diseño a alto nivel y diseño a bajo nivel. En el primero se deben definir las relaciones entre los elementos implicados en la transformación, sin tener en cuenta los detalles de implementación, del mismo

modo que se hace en un modelo a nivel PIM. En el diseño a bajo nivel se redefinen y completan los diagramas de diseño a alto nivel, incluyendo detalles de implementación, de forma similar a los modelos PSM.

- **Nivel de automatización.** Para automatizar el proceso descrito, la propuesta define, aunque no detalla, un conjunto de transformaciones entre los modelos correspondientes a cada una de las fases definidas.
- **Generación de código.** Los autores proponen el empleo del lenguaje EGL para desarrollar generadores de código. De hecho, la propuesta implementa un generador de código para el lenguaje de transformaciones ETL.
- **Visualización de modelos.** Los autores indican que los modelos de las diferentes etapas deben ser visualizados mediante diagramas, pero se trata de diagramas genéricos (por ejemplo, diagramas SysML) que no consideran la naturaleza específica de los modelos de transformación.
- **Validación de modelos.** Para llevar a cabo la validación de los modelos, proponen el empleo del lenguaje de validación EVL que permite definir restricciones de forma similar al lenguaje OCL.
- **Gestión de la trazabilidad.** Los autores de la propuesta consideran de gran importancia la trazabilidad, pero en el ámbito interno del desarrollo de la transformación y entre los modelos de las distintas etapas de dicho desarrollo. En cambio, no ofrecen propuestas para la gestión de la trazabilidad derivada de la transformación que se desarrolla. Tan solo indican que *transML* puede adaptarse tanto a lenguajes de transformación que ofrezcan soporte implícito de trazabilidad como a los que lo ofrezcan de forma explícita.
- **Soporte tecnológico.** Los autores han implementado los metamodelos definidos para cada una de las fases, un conjunto de transformaciones de modelos y varios generadores de código. Sin embargo, no ofrecen una herramienta completa y empaquetada que de soporte a toda la propuesta presentada.

3.2.4.4 A. Kusel

Kusel en [122] presenta un *framework* llamado TROPIC (*Transformations on Petri Nets in Color*) para el desarrollo de transformaciones de modelos. TROPIC permite definir dichas transformaciones en diferentes niveles de abstracción, proporcionando una visión abstracta del mapeo de los elementos y una visión concreta de la transformación. Además, facilita la reutilización de transformaciones, proporcionando una biblioteca de componentes reutilizables que

favorece la productividad en el desarrollo de transformaciones y mejora la calidad de las mismas.

Para llevar a cabo el modelado de las transformaciones, TROPIC ofrece dos puntos de vista: una vista de mapeo y una vista de transformación. En la vista de mapeo se establecen, mediante UML2, las relaciones existentes entre los elementos de entrada y los elementos de salida que intervienen en la transformación. A partir de la información definida en la vista de mapeo, se obtiene la vista de transformación ejecutable que se representa mediante Redes de Petri coloreadas, denominadas *redes de transformación*. Dichas redes son ejecutables y de hecho, Kusel junto a otros autores presentan, en [220], un depurador para mejorar su desarrollo y ejecución.

Para dar soporte a parte de esta propuesta, han desarrollado una primera versión del prototipo TROPIC que permite editar, ejecutar y depurar las redes de transformación. La implementación de TROPIC no es completa e independiente por sí misma, sino que requiere de otras herramientas como EMF y GMF. El prototipo TROPIC, actualmente incluye dos editores: uno para la especificación de la transformación y otro que muestra la representación gráfica de dicha especificación en términos de redes de transformación. Además, proporciona funcionalidades de depuración, como la depuración "paso a paso" y otras funcionalidades de edición. A partir de TROPIC, se ha desarrollado una serie de patrones de transformación reutilizables llamados operadores de mapeo (*Mapping Operators*, MOP). Como primer prototipo utilizan la herramienta AMW para probar el funcionamiento de los MOPs.

En cuanto a los datos extraídos del análisis de la propuesta:

- **Niveles de abstracción a los que se especifica la transformación.** La propuesta ofrece dos puntos de vista o niveles de abstracción para el modelado de las transformaciones. En la vista de mapeo se establecen las relaciones entre los elementos y la semántica de la transformación y en la vista de transformación se obtienen redes de transformación que contienen la lógica de ejecución de la transformación. Así, al igual que en propuestas anteriores, dado que propone una especificación independiente de la ejecución de la transformación y una dependiente, se considera que los niveles definidos en la propuesta son equivalentes a los definidos por MDA.
- **Nivel de automatización.** La propuesta indica que la vista de transformación debería ser generada de forma directa a partir de la vista de mapeo, sin embargo no establece cómo llevar a cabo dicha generación.

- **Generación de código.** La vista de transformación se representa mediante redes de transformación ejecutables, por tanto, no surge la necesidad de generar código. Sin embargo, en el prototipo que da soporte a la propuesta sí se contempla la generación de la transformación en código ATL a partir de los operadores de mapeo.
- **Visualización de modelos.** Para definir y representar la vista de mapeo se propone el empleo de UML y para la vista de transformación se emplean Redes de Petri Coloreadas. Por tanto, se considera que son visualizaciones genéricas que no han sido descritas específicamente para visualizar la especificación de transformaciones de modelos.
- **Validación de modelos.** La propuesta no considera la validación de las especificaciones de la transformación.
- **Gestión de la trazabilidad.** En la propuesta presentada, la autora no contempla la gestión de la trazabilidad a partir del desarrollo de transformaciones de modelos.
- **Soporte tecnológico.** Parte de la propuesta es soportada por el prototipo TROPIC para crear y manipular las redes de transformación mediante dos editores. El prototipo TROPIC ha sido desarrollado mediante el uso de herramientas disponibles en el proyecto EMP de Eclipse (EMF, GMF y AMW).

3.2.4.5 J. M. Küster *et al.*

Los autores Küster, Ryndina y Hauser presentan en [125] un método sistemático e iterativo para el desarrollo de transformaciones de modelos centrado en el diseño de las mismas con el objetivo de asegurar su calidad. Los autores identifican tres fases diferentes: el diseño a alto nivel, el diseño a bajo nivel y la validación del diseño.

El método consiste en definir semi-formalmente, a partir de un análisis de los requisitos que debe cumplir la transformación, las reglas de la transformación, abstrayéndose de sus detalles más específicos. Estas reglas de transformación de alto nivel son refinadas y completadas, siguiendo unas guías de diseño, hasta obtener el diseño de las reglas de transformación a bajo nivel, de forma que sirva como base para la implementación o para la generación del código de la transformación. La propuesta presentada se ilustra mediante un caso de estudio basado en modelos de procesos de negocio.

Sin embargo, los autores no presentan ninguna herramienta para dar soporte tecnológico a la propuesta planteada, por tanto, se trata de una propuesta

puramente metodológica que se centra en mostrar los pasos a seguir para el desarrollo de transformaciones de modelos. Esta idea también se ve reflejada en otros trabajos de Küster juntos a otros autores, como [123] y [124], donde utilizan dicho método para el diseño e implementación de transformación de modelos.

En cuanto a los datos extraídos del análisis de la propuesta:

- **Niveles de abstracción a los que se especifica la transformación.** Los autores proponen definir el diseño de la transformación en dos niveles de abstracción. En primer lugar, se diseñan parcialmente las reglas de la transformación a alto nivel, omitiendo algunos de sus detalles. Por tanto, se considera que este nivel es similar al nivel PIM, propuesto por MDA. Posteriormente, las reglas definidas a alto nivel son refinadas y completadas en un diseño de bajo nivel, de manera similar a como se hace en los modelos PSM de MDA.
- **Nivel de automatización.** La propuesta establece un proceso iterativo compuesto por cinco pasos para refinar sistemáticamente las reglas de la transformación definidas a alto nivel en las reglas a bajo nivel. Sin embargo, los autores no identifican cómo llevar a cabo este proceso de forma automática, tan solo se limitan a describir los pasos.
- **Generación de código.** Los autores indican que las reglas de transformación validadas pueden ser implementadas manualmente o convertidas a código de forma automática. Sin embargo, no proporcionan cómo debería llevarse a cabo dicha generación de código.
- **Visualización de modelos.** Los autores reconocen la necesidad del modelado gráfico de las transformaciones. A nivel PIM proponen el uso de la sintaxis utilizada en el contexto de las transformaciones y a nivel PSM proponen el uso de los diagramas de objeto de UML.
- **Validación de modelos.** Para los autores, la validación del diseño es uno de los aspectos más importantes en el desarrollo de transformaciones de modelos. Por ello, proponen realizar la validación de los modelos de forma sintáctica y semántica antes de su implementación. Sin embargo, no especifican cómo llevar a cabo dicha validación.
- **Gestión de la trazabilidad.** En la propuesta presentada, los autores no contemplan la gestión de la trazabilidad a partir del desarrollo de transformaciones de modelos.
- **Soporte tecnológico.** Los autores no presentan ninguna herramienta para dar soporte a la propuesta metodológica presentada, aunque en las conclusiones de

[125] indican que uno de los trabajos futuros es la construcción de una herramienta para este propósito.

3.2.4.6 K. Lano y S. Kolahdouz-Rahimi

La propuesta presentada en [126] consiste en la definición de un proceso de desarrollo sistemático de transformaciones de modelos basado en semánticas precisas.

Con el objetivo de mitigar los problemas de migración y reusabilidad derivados del gran número de lenguajes de transformación, los autores proponen una aproximación genérica de desarrollo dirigido por modelos de transformaciones de modelos, basada en notaciones UML y MOF.

El proceso propuesto consta de las siguientes cuatro etapas: requisitos, especificación abstracta, diseño y especificación explícita e implementación.

En la primera etapa se detallan los requisitos de la transformación en términos de los metamodelos de origen y destino, incluyendo aquellas restricciones que sean necesarias. En esta primera etapa es posible emplear diagramas de casos de uso para describir la funcionalidad del sistema y los requisitos no funcionales.

En la etapa de especificación abstracta se formalizan los requisitos como restricciones en un lenguaje tipo OCL. Dichas restricciones se emplean para definir de forma global las relaciones existentes entre los modelos para cada caso de uso.

Según los autores, las transformaciones se pueden desarrollar en diferentes fases. Por ello, por cada una de las fases en que se divide la transformación, en la etapa de diseño y especificación explícita se definen las reglas de la transformación en términos de operaciones especificadas por pre y post condiciones y se detalla el orden de ejecución de dichas reglas. En este punto se puede verificar que las reglas satisfacen la especificación y son deterministas y bien definidas. Según los autores, dado que estas reglas y fases se definen de forma independiente, existe la posibilidad de ser reutilizadas en otras transformaciones. Por último, se codifica la transformación un lenguaje de transformación existente.

En cuanto a los datos extraídos del análisis de la propuesta:

- **Niveles de abstracción a los que se especifica la transformación.** En este trabajo, los autores proponen un proceso de desarrollo de transformaciones de modelos dirigido por modelos que se compone en varias etapas en las que se describen las transformaciones a diferentes niveles de abstracción. Dichos

niveles se corresponden con los pasos requeridos para desarrollar una transformación: definición de los requisitos, especificación abstracta como restricciones que definen las relaciones entre los modelos implicados en la transformación, diseño y especificación explícita de las reglas de la transformación y el orden de las mismas, y por último, la codificación de las reglas de transformación.

Los autores no establecen ninguna relación de estos niveles con los definidos por MDA, sin embargo la especificación abstracta y la especificación explícita de la transformación pueden considerarse equivalentes a los niveles PIM y PSM.

- **Nivel de automatización.** El proceso propuesto para el desarrollo de transformaciones de modelos dirigido por modelos no índica cómo llevar a cabo el paso entre los artefactos que se definen en las distintas etapas del proceso, por tanto se asume que no considera la automatización de esta tarea.
- **Generación de código.** A pesar de definir una fase de implementación, los autores no especifican si el código de la transformación se puede obtener mediante un proceso de generación de código o si debe ser creado manualmente.
- **Visualización de modelos.** Los autores proponen que los modelos que describen las diferentes etapas del proceso de desarrollo de las transformaciones sean definidos en términos de los lenguajes de modelado UML y MOF.
- **Validación de modelos.** La propuesta no define explícitamente cómo llevar a cabo la validación de las especificaciones de la transformación. Sin embargo, en la etapa de diseño y especificación explícita sí se considera que es posible verificar que las reglas de la transformación satisfacen la especificación abstracta de la transformación y que se puede comprobar si las reglas se encuentran bien definidas y son deterministas.
- **Gestión de la trazabilidad.** En la propuesta presentada, los autores no contemplan la gestión de la trazabilidad a partir del desarrollo transformaciones de modelos.
- **Soporte tecnológico.** Los autores no presentan ninguna implementación de la propuesta metodológica presentada, por tanto, se asume que se trata de una propuesta puramente teórica.

3.2.4.7 D. Varró y A. Pataricza

Según los autores Varró y Pataricza, las aproximaciones existentes para el desarrollo de transformaciones de modelos se centran en asegurar que el funcionamiento de la ejecución de las transformaciones sea correcto y no consideran importante otras características propias de la Ingeniería del Software tradicional como la reusabilidad, el mantenimiento o el rendimiento. Por ello, en [207] presentan una propuesta que, siguiendo los principios definidos por MDA [153], se centra en definir las transformaciones de modelos como modelos. En concreto, los autores proponen el uso de transformaciones genéricas y meta-transformaciones.

Las transformaciones genéricas definen reglas de transformación independientes de plataforma donde los tipos de algunos de sus componentes son variables y se resuelven en tiempo de ejecución de la transformación. Estas características resultan en transformaciones más compactas y reutilizables, pero sufren un grave problema de rendimiento respecto de las transformaciones tradicionales. Para dotar de fundamentos teóricos a esta idea, los autores proponen VPM. Se trata de un marco de trabajo teórico, presentado por los mismos autores en [206], donde las transformaciones de modelos se definen y representan mediante un formalismo basado en transformaciones de grafos.

Por otro lado, definen a las meta-transformaciones como aquellas transformaciones que utilizan otras transformaciones como salida o entrada. Un requisito indispensable de estas transformaciones es almacenar las reglas de la transformación como modelos terminales. Según los autores, los principales beneficios derivados del uso de meta-transformaciones son el incremento de rendimiento respecto de las transformaciones genéricas y la mejora en la mantenibilidad de las transformaciones.

Para poner en práctica las ideas propuestas y dar soporte a VPM., los autores presentan una actualización de la herramienta VIATRA (http://www.eclipse.org/gmt/VIATRA2/). Concretamente, esta actualización permite llevar a cabo el modelado de transformaciones independientes de plataforma (PIT) y de transformaciones dependientes de plataforma (PST).

En cuanto a los datos extraídos del análisis de la propuesta:

- **Niveles de abstracción a los que se especifica la transformación.** La propuesta presentada hace uso de VPM, un marco de trabajo multinivel. En la propuesta metodológica no indican los niveles a los que se deberían definir las transformaciones. Sin embargo, en la herramienta que da soporte a la propuesta se permite el modelado de las transformaciones de forma

independiente de plataforma (modelos PIT) y dependiente de plataforma (modelos PST). Estos niveles coinciden con los propuestos por Bézivin en [25], descritos en la sección 3.2.4.1. Al igual que ese caso, se considera que los niveles propuestos se corresponden con los niveles PIM y PSM descritos por MDA.

- **Nivel de automatización.** Los autores de este trabajo no sugieren técnicas para llevar a cabo la conversión (semi-)automática de un modelo de un nivel de abstracción en un modelo de otro nivel distinto.
- **Generación de código.** Aunque la definición de propuesta metodológica no contemple cómo se debería llevar a cabo la generación del código que implementa la transformación, los autores al presentar la herramienta que da soporte a la propuesta sí describen la generación de código como una de las características de dicha herramienta. Sin embargo, no describen claramente cómo se lleva a cabo.
- **Visualización de modelos.** Los autores proponen que la descripción de las definiciones de la transformación se lleve a cabo mediante un formalismo de patrones y reglas basado en las características de las transformaciones de grafos.
- **Validación de modelos.** Los autores de esta propuesta no consideran la validación de los modelos que describen la transformación que se está desarrollando.
- **Gestión de la trazabilidad.** En la propuesta presentada, los autores no contemplan la gestión de la trazabilidad.
- **Soporte tecnológico.** Para dar soporte a la propuesta presentada, los autores presentan una actualización de VIATRA, su herramienta para el desarrollo de transformaciones de modelos. Esta nueva versión de la herramienta, construida sobre el entorno Eclipse, ofrece soporte tecnológico para los niveles de modelado propuestos por VPM en [206] y el desarrollo de transformaciones genéricas y meta-transformaciones presentadas en [207]. La principal característica de esta herramienta es el modelado de las transformaciones independientes de plataforma (PIT) y de las transformaciones dependientes de plataforma (PST).

3.2.4.8 A. Vignaga

La propuesta presentada por Vignaga en [211] consiste en la definición de una metodología genérica para aplicar técnicas MDE al desarrollo y evolución de las transformaciones de modelos. Dicha metodología, en palabras de su autor, se

centra en las actividades de diseño e implementación, aunque debería contemplar todo el ciclo de vida. En este trabajo, además el autor identifica dos problemas asociados al cuerpo del conocimiento: la ausencia de un paradigma de programación dominante o consensuado para la implementación de transformaciones de modelos; y la falta de una clasificación o categorización unificada de las transformaciones de modelos.

Según el autor, la metodología está definida como un proceso completo expresada mediante un modelo SPEM y propone un ciclo de vida basado en un modelo iterativo e incremental, estructurado en etapas, una para la construcción y una para la evolución. Sin embargo, en [211] no se detalla dichos procesos.

Continuando esta línea de investigación, Vignaga junto a Perovich y Bastarrica proponen en [212] realizar la especificación de las transformaciones de modelos siguiendo una estructura de cuatro niveles de abstracción. El primer nivel, el más bajo, debería contener el código que implementa la transformación, en términos de constructores específicos de plataforma. El segundo nivel debería contener una abstracción de los constructores especificados en el nivel anterior, suprimiendo los constructores específicos de plataforma. En el tercer nivel, se debería especificar las relaciones entre los diferentes elementos. Por último, en el cuarto nivel se debería incluir mecanismos de modularización, especificando lo qué debería hacer la transformación, pero con menor detalle que en las etapas anteriores.

En [212], los autores identifican como tarea futura la construcción de una herramienta para dar soporte a la propuesta presentada. Sin embargo, en la actualidad esta línea de investigación parece que ha sido abandonada ya que no se ha encontrado dicha implementación ni nuevos trabajos relacionados.

En cuanto a los datos extraídos del análisis de la propuesta:

- **Niveles de abstracción a los que se especifica la transformación.** La propuesta define cuatro niveles de abstracción que van desde código que implementa la transformación (nivel más bajo) hasta el nivel en el que se definen los mecanismos de modularización (nivel más alto). Los dos niveles superiores son independientes de plataforma, por tanto existe una equivalencia con los modelos PIM; y los dos niveles inferiores contienen información relativa a la plataforma, por lo que se corresponden con los modelos PSM de MDA.
- **Nivel de automatización.** Aunque los autores reconocen la necesidad de establecer mecanismos de automatización en el paso de unos niveles de abstracción a otros, no especifican cuáles ni cómo deberían ser.

- **Generación de código.** Una de las capas de abstracción establecidas es el código de la transformación en el lenguaje que corresponda, sin embargo no establecen cómo crear o generar dicho código.
- **Visualización de modelos.** Los autores consideran la necesidad de disponer mecanismos de visualización, de hecho cuando identifican como trabajo futuro el desarrollo de su herramienta de soporte, mencionan la herramienta AMW como ejemplo para el mapeado de información.
- **Validación de modelos.** Los autores no especifican cómo se debería llevar a cabo la validación, sin embargo la identifican como una tarea deseable.
- **Gestión de la trazabilidad.** En la propuesta presentada, los autores no contemplan la gestión de la trazabilidad.
- **Soporte tecnológico.** En [212], los autores identifican la construcción de una herramienta de soporte como trabajo futuro, sin embargo no se han encontrado pruebas de que se haya desarrollado dicha aplicación.

3.2.5 Discusión

Tras analizar una por una todas las propuestas identificadas en la literatura para el desarrollo dirigido por modelos de transformaciones de modelos, en esta sección se le asigna a cada propuesta los valores correspondientes a los criterios de evaluación propuestos en la sección 3.2.2 y posteriormente, se analizan los valores obtenidos con el objetivo de dar respuesta a las cuestiones de investigación propuestas en la sección 3.2.1.

En la Tabla 3-1 se presentan los valores asignados a cada propuesta por cada uno de los criterios de evaluación. Se recuerda que la puntuación de cada uno de los posibles valores es: S=1; P=0.5; y N=0. Por tanto, dado que se han definido ocho criterios de evaluación (CE), cada una de las propuestas aspira a obtener, como máximo, ocho puntos. La penúltima columna de la tabla ("Total") muestra la puntuación obtenida por cada una de las propuestas y la última columna ("% máximo posible") indica el porcentaje de puntos que ha obtenido cada propuesta respecto de 8, es decir del total de puntos posibles.

Como muestra la Tabla 3-1, todas las propuestas han obtenido una puntuación que se encuentra en el rango entre 3 y 6.5 puntos, o lo que es lo mismo entre el 37.50% y el 81.25% de la puntuación total. Cabe destacar que el 50% de las propuestas analizadas (4 de 8) han obtenido una puntuación igual al 50% de la puntuación máxima (4 puntos sobre 8 posibles), mientras que de las otras cuatro

propuestas restantes: dos de ellas se sitúan por encima del 50% de los puntos posibles y las otras dos por debajo de esa cifra.

Tabla 3-1. Resumen de los Criterios de Evaluación en cada propuesta

	Niveles PIM y PSM o equivalentes	Paso entre niveles (Automático)	Generación de código	Visualización	Validación	Independencia de lenguaje de transformación	Gestión de la Trazabilidad	Herramienta de Soporte		
Propuesta	**CE-1**	**CE-2**	**CE-3**	**CE-4**	**CE-5**	**CE-6**	**CE-7**	**CE-8**	**Total**	**% máximo posible**
Bézivin *et al.*	S	S	P	N	P	S	N	N	**4,00**	**50,00%**
Didonet del Fabro	P	S	S	P	N	S	P	S	**5,50**	**68,75%**
Guerra *et al.*	S	S	S	P	S	S	P	P	**6,50**	**81,25%**
Kusel	S	P	P	P	N	S	N	P	**4,00**	**50,00%**
Küster *et al.*	S	N	P	P	S	S	N	N	**4,00**	**50,00%**
Lano y Kolahdouz-Rahimi	S	N	N	P	P	S	N	N	**3,00**	**37,50%**
Varró y Pataricza	S	N	P	P	N	S	N	S	**4,00**	**50,00%**
Vignaga	S	P	P	N	P	S	N	N	**3,50**	**43,75%**

Total CE	**7,50**	**4,00**	**4,50**	**3,00**	**3,50**	**8,00**	**1,00**	**3,00**	**34,50**
% máximo posible CE	**93,75%**	**50,00%**	**56,25%**	**37,50%**	**43,75%**	**100,00%**	**12,50%**	**37,50%**	**53,91%**

En las últimas dos filas de la tabla (“Total CE” y “% máximo posible CE”) se presenta la puntuación alcanzada para cada uno de los criterios de evaluación, sumando los puntos asignados a cada propuesta para dicho criterio; y el porcentaje de puntos que supone respecto del total de puntos posibles. Se recuerda que, en esta revisión de la literatura, se han analizado 8 propuestas y que el máximo valor que puede asignar por cada CE es 1, por tanto la puntuación máxima a la que aspira cada CE es de 8 puntos. Esta puntuación máxima tan solo ha sido alcanzada por uno de los criterios de evaluación (CE-6), aunque CE-1 con 7.5 puntos se acerca bastante dicha puntuación máxima. Otro dato a tener en cuenta es que la mitad de los criterios de evaluación (CE-4, CE-5, CE-7 y CE-8) han obtenido una puntuación por debajo del 50% del máximo de puntos posibles. De estos criterios, llama la atención que la gestión de la trazabilidad solo haya obtenido un punto.

A partir de estos datos se puede afirmar que, dado que todas las propuestas analizadas han obtenido una puntuación entre el 37.50% y el 81.25% de los puntos posibles, todas ellas se encuentran relacionadas en mayor o menor medida con el

objetivo de investigación planteado. Además, se puede afirmar que existe un consenso en la literatura en cuanto a ciertas características que debe cumplir toda propuesta centrada en el desarrollo dirigido por modelos de transformaciones de modelos, como son la independencia del lenguaje que finalmente codifique la transformación o la especificación de la transformación a diferentes niveles de abstracción. Sin embargo, existen otras características como puede ser la validación o la visualización de las especificaciones de la transformación que no son afrontadas por todas las propuestas o no lo hacen por completo. Dado que la mitad de los CE ha obtenido una puntuación por debajo del 50% de los puntos posibles, parece que las características evaluadas por dichos CE son posibles puntos de mejora en el estado del arte en el desarrollo de transformaciones de modelos dirigidas por modelos.

Una vez que se dispone de los valores asignados a cada propuesta, se procede a analizarlos para dar respuesta a cada una de las cuestiones de investigación (CI) planteadas en la sección 3.2.1:

- **CI-1.** ¿A qué niveles de abstracción se propone especificar las transformaciones de modelos?

Una de las principales características del desarrollo de software dirigido por modelos es la definición del sistema en diferentes niveles de abstracción [85, 186, 213]. El caso de la construcción de transformaciones de modelos no puede ser diferente y por tanto, para considerar que se sigue una aproximación de DDM es necesario definir una jerarquía de niveles de abstracción en los que se describe y especifica la transformación. Una de las aproximaciones más adoptadas en este sentido es MDA, que define tres niveles de abstracción principales: CIM, PIM y PSM [106, 153, 180]. Para evaluar cómo se cumple este requisito en la literatura analizada se ha empleado el criterio de evaluación CE-1 que evalúa si las propuestas proponen el uso de los niveles PIM y PSM definidos por MDA o niveles análogos. Este criterio de evaluación ha obtenido un 93.75% de los puntos posibles, lo que significa que la especificación de las transformaciones a diferentes niveles de abstracción es una cuestión bastante afrontada por las propuestas analizadas.

Aunque todas las propuestas analizadas consideren el uso de niveles de abstracción para la construcción de transformaciones, ninguna de ellas propone usar los niveles de abstracción según los define MDA. Sin embargo, todas las propuestas analizadas, a excepción de la presentada por Didonet del Fabro, proponen el uso de al menos dos niveles de abstracción: uno independiente de

plataforma y uno dependiente de plataforma, es decir niveles de abstracción que se pueden considerar equivalentes a los niveles PIM y PSM definidos por MDA.

Acerca de esta CI se puede concluir que, en vista a los datos obtenidos, una tarea que es necesario tener en cuenta a la hora de desarrollar una propuesta para el DDM de transformaciones de modelos es la definición de los niveles de abstracción a los que se especifican las transformaciones. En este sentido, es habitual la definición de, al menos, un nivel independiente de plataforma que omita los detalles específicos del lenguaje o motor de transformación que se utilice para implementar y ejecutar la transformación y un nivel menos abstracto que sí considere los detalles o características propias del lenguaje que implementará la transformación.

- **CI-2.** ¿Qué ideas se proponen para automatizar el proceso de desarrollo de las transformaciones?

En la cuestión anterior se analiza una de los principios del DSDM, en esta se estudia el comportamiento de las propuestas analizadas respecto a otro de los principios fundamentales del DSDM: la automatización del proceso de desarrollo [85, 179, 182, 186, 213].

Para dar respuesta a esta cuestión, se hace uso de los valores asignados a los criterios de evaluación CE-2 y CE-3, encargados de evaluar si las propuestas definen técnicas o métodos para automatizar el paso entre los modelos de los diferentes niveles de abstracción y para la generación de código, respectivamente. Ambos CE han obtenido una puntuación en torno al 50% de la puntuación máxima, 4 puntos en el caso de CE-2 y 4.5 puntos en el de CE-3. A pesar de haber obtenido una puntuación relativamente baja, todas las propuestas, excepto la presentada por Lano y Kolahdouz-Rahimi, afrontan al menos una de estas dos cuestiones.

En cuanto a automatización del paso entre modelos de diferentes niveles de abstracción, solo tres propuestas (Bézivin *et al.*, Didonet del Fabro y Guerra *et al.*) proponen alguna técnicas para llevar a cabo esta tarea. Todas ellas coinciden en la conveniencia del uso de transformaciones de modelos. Además, los autores Kusel y Vignaga también consideran que automatizar el paso entre los niveles de abstracción es una tarea deseable aunque no indican cómo se debería hacer.

En cuanto a la generación del código que implementa la transformación, todas las propuestas, excepto la presentada por Lano y Kolahdouz-Rahimi, consideran que es una característica deseable en un entorno de desarrollo de transformaciones de modelos, sin embargo, solo las propuestas de Didonet del Fabro y Guerra *et al.* proponen cómo afrontarla. En el caso de Didonet del Fabro

se propone el uso de la herramienta TCS para crear un extractor de código que serialice los modelos definidos en código ATL. La propuesta presentada por Guerra *et al.* recomienda usar el lenguaje EGL para desarrollar generadores de código.

La ausencia de métodos para la generación de código puede deberse a que, como se desprende de los valores asignados a CE-6, todas las propuestas presentan métodos orientados al desarrollo de transformaciones de forma independiente a los lenguajes de transformación existentes.

- **CI-3.** ¿Existen propuestas metodológicas centradas en el DDM de transformaciones de modelos que propongan o sugieran ideas para la gestión de las trazas a partir del modelado de dichas transformaciones?

Uno de los objetivos principales de esta tesis doctoral es combinar las técnicas de generación y gestión de la información de trazabilidad con las técnicas de DDM de transformaciones de modelos. Por ello, esta cuestión de investigación tiene por objetivo analizar si las propuestas para el DDM de transformaciones de modelos ofrecen métodos o ideas para obtener información de trazabilidad a partir de las transformaciones de modelos generadas.

Para dar respuesta a esta pregunta se analiza el criterio de evaluación CE-7, que se centra directamente en esta cuestión. Dicho criterio ha obtenido la puntuación más baja (1 punto) de todos los CE definidos, por tanto es un tema que se encuentra muy inmaduro en el estado del arte. De hecho, ninguna propuesta de las que han sido analizadas ofrece técnicas para la gestión de la trazabilidad y tan solo 2 de ellas (Didonet del Fabro y Guerra *et al.*) establecen cierta relación entre la gestión de la trazabilidad y el DDM de transformaciones de modelos. En concreto, Didonet del Fabro presenta cómo los modelos de *weaving* pueden ser empleados para visualizar y almacenar información de trazabilidad; y Guerra *et al.* consideran la gestión de la trazabilidad como una tarea importante pero en el ámbito interno del desarrollo de las transformaciones, es decir, entre los modelos que definen la transformación y no contemplan la gestión de la trazabilidad a partir de las transformaciones desarrolladas.

En vista de los datos obtenidos se pone de manifiesto que existe espacio para la mejora en el estado del arte actual en DDM de transformaciones de modelos.

- **CI-4.** ¿Existen herramientas que den soporte tecnológico al DDM de transformaciones de modelos?

En el campo de la investigación en Ingeniería del Software es ampliamente conocida la distancia que existe entre la teoría y la práctica. Es habitual encontrar

grandes propuestas teóricas que, por unos motivos u otros (aproximaciones poco realistas, tecnologías insuficientes, falta de financiación, etc.), resultan complicadas de poner en práctica [130, 143, 163]. Por este motivo, esta cuestión de investigación tiene por objetivo comprobar si en el estado del arte actual, además de la existencia de propuestas metodológicas para el DDM de transformaciones de modelos, es posible encontrar implementaciones o herramientas completas que ofrezcan soporte tecnológico para dichas propuestas.

Los valores asignados al criterio de evaluación CE-8 para cada una de las propuestas así como los datos extraídos de dichas propuestas proporcionan la información necesaria para dar respuesta a esta cuestión. El criterio CE-8 es uno de los que ha obtenido peores resultados (3 puntos) y el 50% de las propuestas no consideran poner en práctica sus ideas teóricas. Así pues, parece que la distancia entre teoría y práctica se vuelve a hacer patente.

De las cuatro propuestas que sí emplean esfuerzos en implementar o poner en práctica sus ideas metodológicas, dos de ellas (Guerra *et al.* y Kusel) no proporcionan herramientas completas sino que presentan implementaciones parciales y dependientes de otras herramientas disponibles en el proyecto EMP (*Eclipse Modeling Project*, http://eclipse.org/modeling). EMP es un conjunto de herramientas e implementaciones de referencia que facilitan la construcción de herramientas de soporte tecnológico para propuestas basadas en los principios de MDE. Las dos propuestas que sí proporcionan una implementación completa para dar soporte a su propuesta metodológica son las presentadas por Didonet del Fabro y Varró y Pataricza.

El autor Didonet del Fabro presenta AMW, una herramienta construida sobre el entorno Eclipse, que permite la definición de modelos de *weaving* para la establecer relaciones a alto nivel entre elementos de diferentes modelos. A partir de la definición de relaciones de transformación en estos modelos y mediante la ejecución de una transformación tipo HOT, es posible obtener un modelo de la transformación conforme a las características del lenguaje de transformación ATL. Además, dichos modelos se pueden serializar en el código ATL que implementa la transformación definida.

En el caso de los autores Varró y Pataricza, presentan una actualización de su herramienta para el desarrollo de transformaciones de modelos (VIATRA). La nueva versión de VIATRA, a diferencia de la anterior, ha sido construida sobre el entorno Eclipse. Su principal característica es permitir el modelado de las transformaciones a nivel PIT y PST.

Tras analizar el estado del arte en cuanto al soporte tecnológico, se puede afirmar que esta cuestión se encuentra aún inmadura en el estado del arte actual. Teniendo en cuenta que las cuatro propuestas que realizan alguna aproximación en este sentido coinciden en llevar a cabo las implementaciones sobre el entorno Eclipse, se puede asumir que, en este caso, el problema no está relacionado con la falta de medios tecnológicos.

- **CI-5.** ¿Cuáles son las limitaciones del estado del arte en el DDM de transformaciones de modelos?

Por último, analizando las respuestas a las preguntas anteriores y los datos obtenidos del análisis de las propuestas, se da respuesta a esta cuestión que tiene por objetivo resumir las limitaciones o puntos de mejora que se han identificado en el estado del arte actual en el DDM de transformaciones de modelos.

En el apartado de discusión de CI-1 se han analizado los niveles de abstracción que, según las propuestas, deberían emplearse para el desarrollo de transformaciones de modelos dirigido por modelos. En este punto, se ha identificado que todas las propuestas, a excepción de una, coinciden en la necesidad de definir, al menos, un modelo independiente de plataforma y un modelo dependiente de plataforma. En este sentido, el único punto de mejora identificado es la falta de consenso acerca del tipo de modelos empleados dentro de dichos niveles. Una posible solución a esta diferencia de criterios es el uso de un estándar como MDA, que define explícitamente los niveles de abstracción que se deberían emplear en la construcción de artefactos software.

En la discusión de CI-2, se analiza la automatización del proceso de desarrollo de las transformaciones de modelos. Para dar respuesta a dicha cuestión se ha estudiado cómo se automatiza el paso entre los modelos de diferentes niveles de abstracción y cómo se automatiza la generación del código que implementa la transformación. Como es de esperar en un contexto dirigido por modelos, los autores consideran que la automatización del proceso es un aspecto importante que se debe tener en cuenta. Sin embargo, solo tres propuestas de las ocho propuestas analizadas sugieren ideas o técnicas para automatizar el paso entre los modelos de los distintos niveles de abstracción, en concreto, proponen el empleo de transformaciones de modelos. Y tan solo, dos de ellas proporcionan ideas o técnicas para automatizar ambas tareas. A la vista de estos datos, se pone de manifiesto que una de las debilidades del estado del arte en el DDM de transformaciones de modelos es la automatización del proceso de desarrollo. Una posible solución es dirigir las investigaciones futuras hacia las ideas identificadas

en la literatura: uso de transformaciones de modelos y generadores o extractores de código.

En la CI-3 se analiza si existen propuestas que consideren la gestión de información de trazabilidad a partir de las transformaciones de modelos. En este sentido, se ha identificado un gran posible punto de mejora en el estado del arte, ya que ninguna de las propuestas analizadas afronta esta cuestión. Como se defiende en esta tesis doctoral, a partir de la ejecución de las transformaciones de modelos es posible realizar (semi-)automáticamente otras actividades software. En el contexto de un desarrollo dirigido por modelos, el software se construye a partir de la ejecución de transformaciones que contienen las relaciones entre los elementos de los modelos que describen a dicho software, por tanto parece factible aprovechar estas ventajas para obtener las trazas del sistema a partir de las transformaciones. Una posible forma de adaptar esta idea al DDM de transformaciones de modelos sea definir constructores específicos de trazabilidad en los modelos que describen la transformación.

En la discusión acerca de CI-4, acerca de las herramientas que proporcionan soporte tecnológico a las propuestas metodológicas, se ha identificado que, aunque existen entornos de trabajo como Eclipse que facilitan el desarrollo de herramientas para dar soporte a propuestas MDE, la mitad de las propuestas analizadas no ofrecen implementaciones para poner en práctica sus ideas teóricas. En este sentido, ya que la tecnología actual proporciona los medios necesarios para construir herramientas de soporte, sería deseable que las propuestas para el DDM de transformaciones de modelos proporcionaran herramientas completas para convertir en realidad este tipo de desarrollos.

Finalmente, otras limitaciones que se han identificado en el análisis del estado del arte es la ausencia de técnicas de visualización de modelos de transformaciones que proporcionen una representación ajustada a las características de estos modelos. Como se puede comprobar en la Tabla 3-1, todas las propuestas que sugieren cómo visualizar los modelos que representan las transformaciones, proponen el uso de editores de modelos genéricos. Tan solo el editor propuesto por Didonet del Fabro, aunque genérico, se adapta a las características de un modelo de transformación, ya que permite visualizar relaciones entre elementos. Otro tema que se encuentra inmaduro en el estado del arte actual es empleo de técnicas, como la validación, que mejoren las transformaciones desarrolladas. Dado que gestionar las transformaciones como modelos, sería deseable aprovechar las ventajas que ofrece la gestión de los modelos en este sentido [21].

3.3 Propuestas para la Generación de Trazabilidad a partir del Desarrollo de Transformaciones de Modelos

Como se ha comprobado en la sección anterior, las propuestas para el desarrollo dirigido por modelos de transformaciones de modelos no consideran la gestión de la trazabilidad. Por ello y dado que en esta tesis doctoral uno de los principales objetivos es la generación y gestión de la información de trazabilidad, para complementar el cuerpo de conocimiento, en esta sección se realiza un nuevo estudio de la literatura cuyo principal objetivo es identificar y analizar diferentes propuestas centradas en la generación y gestión de información de trazabilidad a partir del desarrollo de transformaciones de modelos.

Mientras que en la sección anterior el foco de atención principal era la aplicación de los principios de MDE (uso de modelos y automatización del proceso) al desarrollo de transformaciones de modelos, en esta el foco principal es la generación y gestión de la información de trazabilidad que se deriva de la ejecución de las transformaciones de modelos, sin tener en cuenta si aplican los principios de MDE.

Para llevar a cabo esta revisión de la literatura se ha seguido un proceso completo de revisión sistemática basado en las guías propuestas por Kitchenham y Charters [105] y Biolchini [27], como indica el método presentado en la sección 2.2 para la identificación del cuerpo de conocimiento.

Los detalles de este proceso de revisión sistemática son descritos en el Apéndice A de este documento, mientras que en las siguientes sub-secciones se presentan los objetivos o cuestiones de investigación a las que se quiere dar respuesta con este estudio de la literatura (sección 3.3.1), los criterios que se emplean para evaluar y comparar cada una de las propuestas identificadas en la literatura (sección 3.3.2), los datos que se extraen de cada propuesta (sección 3.3.3), el análisis detallado de cada una de ellas (sección 3.3.4) y por último, se presentan las conclusiones obtenidas del estudio de la literatura (sección 3.3.5).

3.3.1 Cuestiones de Investigación

El principal objetivo de esta revisión de la literatura consiste en identificar y analizar propuestas centradas en generar y gestionar información de trazabilidad a partir del desarrollo de transformaciones de modelos. Para dar respuesta a este objetivo principal, se propone el siguiente conjunto de **cuestiones de investigación (CI)**, a las que se darán respuesta en la sección 3.3.5:

- **CI-1.** ¿Qué nivel de automatización ofrecen o sugieren las propuestas metodológicas para generar información de trazabilidad?

  Aunque en este estudio de la literatura el objetivo principal no es analizar si las propuestas aplican los principios de MDE al desarrollo de las transformaciones, sí que resulta interesante conocer qué tareas es posible automatizar y hasta qué nivel. Así será posible conocer si, en el estado del arte actual, la generación de información de trazabilidad se considera una tarea manual o si por el contrario, se proporcionan métodos que la automaticen (completa o parcialmente).

- **CI-2.** Según las propuestas metodológicas identificadas, ¿cómo se deberían almacenar, visualizar y analizar las trazas generadas?

  Como se ha indicado en la introducción de este documento, la información de trazabilidad puede ser muy útil para llevar a cabo otras actividades del ciclo de vida software, como por ejemplo, el análisis del impacto del cambio, la toma de decisiones o el mantenimiento general del sistema [5, 37, 38, 107, 142]. Por tanto, resulta interesante conocer cómo, según las propuestas metodológicas, deberían llevarse a cabo ciertas tareas relacionadas con la gestión de las trazas como el almacenamiento (en modelos de trazas, en repositorios de trazas, embebidas en los modelos terminales, etc.), la visualización (de forma gráfica o textual) y las técnicas para facilitar su análisis (informes técnicos, estudios estadísticos, clasificaciones, etc.).

- **CI-3.** ¿Existen herramientas o marcos de trabajo que den soporte tecnológico para la gestión de la trazabilidad en el contexto del desarrollo de transformaciones de modelos?

  Una de las principales metas de este estudio de la literatura es identificar si las propuestas para la generación y gestión de la trazabilidad a partir del desarrollo de transformaciones de modelos han sido puestas en práctica por medio de herramientas MDE. Por tanto, el objetivo de esta cuestión de investigación es analizar si las propuestas ofrecen a los usuarios algún tipo de soporte tecnológico.

- **CE-4**. ¿Cuáles son las limitaciones encontradas en las propuestas para la generación de trazabilidad a partir del desarrollo de transformaciones de modelos?

  Finalmente, en la última cuestión se combinarán las respuestas a las preguntas anteriores (CI-1, CI-2 y CI-3) para identificar problemas o limitaciones en el estado del arte actual del tema objetivo.

3.3.2 Criterios de Evaluación

Al igual que en la revisión de la literatura anterior y de acuerdo a la guía para realizar una revisión sistemática propuesta por Kitchenham y Charters [105], es recomendable disponer de mecanismos que permitan evaluar y comparar las propuestas que ayudarán a dar respuesta a las preguntas anteriores. Por ello, en esta sección se define un conjunto de **criterios de evaluación (CE)** y un conjunto de **valores** que pueden adoptar dichos criterios.

Para facilitar la comparativa entre las propuestas, los valores que se les pueden asignar para cada uno de los criterios de evaluación son: Sí (S) = 1; Parcial = 0.5; y No (N) = 0. Los criterios de evaluación para las propuestas centradas en la generación de trazabilidad a partir de transformaciones de modelos son:

- **CE-1**. ¿La propuesta metodológica sugiere o indica cómo se debería automatizar la identificación de las relaciones de trazabilidad? (Se recuerda que: las relaciones de trazabilidad son aquellas que se definen entre tipos de elementos, por ejemplo clases o elementos de un metamodelo)

  Valores: Sí (S), proporciona o específica algún método para mejorar el nivel de automatización de esta tarea; Parcial (P), no indica cómo debería automatizarse, pero lo considera deseable; No (N), la propuesta no considera la automatización de la identificación de las relaciones de trazabilidad, por tanto la asume como una tarea manual.

- **CE-2.** ¿La propuesta metodológica sugiere o indica cómo se debería automatizar la generación de las trazas? (Se recuerda que: los enlaces de trazas, o simplemente trazas, son las instancias de las relaciones de trazabilidad, por ejemplo, entre objetos o elementos de un modelo).

  Valores: S, la propuesta indica cómo sería posible automatizar la generación de las trazas; P, la propuesta no especifica cómo automatizar esta tarea, aunque considera que sería deseable automatizarla; N, no considera la automatización de la generación de trazas, es decir, se considera manual.

- **CE-3.** ¿La propuesta para la generación y gestión de la información de trazabilidad es independiente de cualquier lenguaje de transformación concreto?

  Valores: S, la propuesta es independiente de cualquier lenguaje de transformación; P, no es completamente independiente, o es dependiente de un lenguaje concreto pero consideran deseable que fuera independiente; N, la propuesta es totalmente dependiente de un lenguaje concreto.

- **CE-4.** ¿Propone alguna técnica para el almacenamiento de las trazas?

  Valores: S, la propuesta sugiere cómo almacenar las trazas generadas, en general, existen dos aproximaciones: *almacenamiento interno* (en los modelos terminales) y *almacenamiento externo* (en otras estructuras de datos, fuera de los modelos terminales); P, considera que el almacenamiento de las trazas es una característica a tener en cuenta, pero no sugiere ningún método de almacenamiento; N, no considera el almacenamiento de las trazas.

- **CE-5.** ¿Propone alguna técnica para la visualización o representación de las trazas?

  Valores**:** S, sugiere el desarrollo de una herramienta *ad-hoc* para la visualización de las trazas o sugiere el empleo de alguna representación ya creada para este propósito; P, sugiere el empleo de visualizaciones no específicas para la representación de trazas, es decir son representaciones genéricas y por tanto la visualización resultante no es óptima. Por ejemplo, si las trazas son almacenadas en modelos y emplea un editor de modelos genérico como el que ofrece EMF; N, la propuesta no sugiere ningún tipo de visualización de trazas.

- **CE-6.** ¿La propuesta metodológica sugiere o propone técnicas para llevar a cabo el análisis de la información de trazabilidad generada?

  Valores: S, la propuesta centra parte de sus objetivos en proporcionar técnicas de análisis de la información de trazabilidad; P, la propuesta considera interesante clasificar o analizar la información obtenida a partir de las trazas, pero no específica cómo hacerlo; N, la propuesta no considera el análisis de las trazas.

- **CE-7.** ¿Proporciona alguna implementación, herramienta, o soporte tecnológico?

  Valores: S, proporciona una herramienta o marco de trabajo completo para dar soporte a la propuesta metodológica; P, proporciona una implementación parcial de la propuesta; N, se trata de una propuesta puramente teórica que no ofrece implementación para ponerla en práctica.

3.3.3 Extracción de Datos

Para facilitar el análisis y evaluación de las propuestas que forman el estado del arte y dar respuesta a las preguntas anteriores es necesario recopilar

cierta información de dichas propuestas. En esta revisión de la literatura se ha decidido extraer la siguiente información de cada una de las propuestas:

- Título, autores, resumen y año de publicación de cada estudio identificado (perteneciente a la propuesta).
- Si las relaciones de trazabilidad se identifican de forma (semi-)automática o deben especificarse manualmente por el usuario.
- Nivel de abstracción de la transformación en el que se identifican las relaciones de trazabilidad (en el código de la transformación o en algún modelo que especifique la transformación a alto nivel).
- Si la información de trazabilidad puede ser generada a partir de una transformación definida de acuerdo a cualquier lenguaje de transformación o si por el contrario depende de un lenguaje o motor de transformación específico.
- Si las trazas se generan (semi-)automáticamente o si por el contrario, debe crearlas el usuario.
- La forma de almacenar los enlaces de traza (en modelos de trazas, repositorios, modelos terminales, etc.).
- Tipo de metamodelo de trazabilidad propuesto (específico o genérico), si aplica. Esto es, una propuesta puede proporcionar un metamodelo de trazabilidad genérico que sea para todos los casos o puede permitir la definición de un metamodelo específico para cada escenario de trazabilidad.
- La forma que se visualizan/representan las trazas.
- Las técnicas que proporciona para el análisis de las trazas.
- Si ofrece soporte tecnológico y cómo lo hace (si aplica).

Estos datos se extraen en la siguiente sub-sección, donde se analiza en detalle cada una de las propuestas por separado.

3.3.4 Análisis de las propuestas

Una vez que se ha definido la información que se va a extraer y evaluar de cada propuesta, en esta sub-sección se procede a, por un lado, analizar de forma individual las propuestas identificadas y por otro, comparar dichas propuestas de acuerdo a los criterios de evaluación definidos en la sección 3.3.2.

3.3.4.1 L. Bondé *et al.*

El trabajo presentado en [32] está centrado en el ámbito del desarrollo de sistemas embebidos, donde los diseñadores deben manejar multitud de modelos a distintos niveles de abstracción.

Los autores defienden que aplicar MDA puede ser muy beneficioso para este tipo de desarrollos. Sin embargo, se plantean una cuestión: cuando un modelo que describe el sistema es transformado a otros modelos de menor nivel de abstracción, deberíamos poder asegurar la interoperabilidad entre estos modelos de menor nivel. La interoperabilidad entre los modelos es entendida por los autores como la posibilidad de intercambiar información entre los modelos. Esto es, tenemos un modelo de alto nivel compuesto por un elemento *a* y un elemento *b* conectados entre sí. Cuando a partir de este, se generen varios modelos a bajo nivel, si existe interoperabilidad entre estos modelos de bajo nivel, podremos conectar el elemento *a* de uno de los modelos con el elemento *b* de otro.

Para dar respuesta a este problema, proponen una aproximación basada en el empleo de modelos de trazas. Estos modelos son generados automáticamente a partir de las relaciones definidas en las transformaciones de los modelos. Para llevar a cabo la transformación, los autores proponen el uso del motor de transformación *ModTransf* (http://www.lifl.fr/~dumoulin/mdaTransf/), que está basado en reglas XML. La transformación además de proporcionar el modelo de salida, proporciona el modelo de trazas generado. Posteriormente, dicho modelo de trazas es analizado para conocer qué ajustes son necesarios para permitir el intercambio de información entre los modelos. A partir de esta información se genera un puente (*bridge*) que solventa los problemas encontrados y permite la interconexión entre los modelos.

Los modelos de trazas generados son conformes a un metamodelo de trazabilidad de propósito general que se compone de los elementos básicos de una transformación de modelos: elementos de los modelos y relaciones entre los elementos. Además, permite definir qué operaciones (creación, enlace, copia, conversión y transformación) se establecen entre los elementos.

A continuación, se presentan algunos de los datos extraídos del análisis de la propuesta que permitirán dar valor a los criterios de evaluación definidos en la sección 3.3.2:

- **Identificación de las relaciones de trazabilidad.** En la propuesta presentada, las relaciones de trazabilidad son identifican automáticamente a partir de las relaciones implícitas en las reglas de la transformación.
- **Nivel de abstracción en que se identifican las relaciones de trazabilidad.** Como se indica en el punto anterior, las relaciones se identifican en las propias

reglas de la transformación que se ejecuta, por tanto, se considera que son identificadas a bajo nivel.

- **Generación de enlaces de traza.** Los autores proponen que, haciendo uso de las relaciones de trazabilidad identificadas, al ejecutar una transformación de modelos se obtenga de forma automática por un lado, la salida de la transformación y por otro, un modelo que contenga las trazas entre los elementos implicados en dicha transformación. Las transformaciones son ejecutadas en el motor de transformación *ModTransf* que está basado en reglas de tipo XML.
- **Almacenamiento.** La propuesta sugiere que las trazas obtenidas sean almacenadas en un modelo de trazas creado para ese propósito.
- **Metamodelo de trazabilidad.** La propuesta presenta un metamodelo de trazabilidad genérico. Dado que las relaciones de trazabilidad se identifican a partir de las relaciones existentes en las reglas de transformación, dicho metamodelos se compone de los elementos básicos que definen una transformación de modelos: los elementos de los modelos y las relaciones entre dichos elementos. Además de esta información, permite especificar qué operaciones (creación, enlace, copia, conversión y transformación) se establecen entre los elementos.
- **Visualización.** Los autores no indican ni sugieren cómo visualizar las trazas generadas por la propuesta. Sin embargo, en [32] muestran un modelo de trazas resultante en términos de XML. Por tanto se puede interpretar que, aunque no afronten cómo se deberían visualizar las trazas, consideran que es una tarea a tener en cuenta en el contexto de la propuesta.
- **Análisis de las trazas.** La propuesta no considera el análisis de la información de trazabilidad generada.
- **Soporte tecnológico.** Los autores no proporcionan ninguna herramienta o implementación para dar soporte a la propuesta metodológica presentada. Por tanto, se considera que es una propuesta puramente teórica.

3.3.4.2 A. Boronat *et al.*

Los autores Boronat, Carsí y Ramos proponen en [34], un marco de trabajo para la gestión de modelos genéricos que permite la definición de operadores algebraicos para resolver problemas específicos.

Esta propuesta se apoya en la idea de que en todo proceso MDA, los modelos son refinados continuamente y para ello deben existir operadores que permitan especificar la forma de realizar dichos refinamientos. Esta propuesta no

está orientada a la gestión de la trazabilidad en el desarrollo de transformaciones de modelos. Sin embargo, ha sido incluida en este estado del arte porque su principal contribución es proporcionar una metodología de generación de trazas independiente del lenguaje de transformación empleado.

El método propuesto es el siguiente: para generar el modelo de trazas, el usuario puede emplear el metamodelo de trazabilidad genérico que proporciona MOMENT o bien, extender dicho metamodelo para construir uno específico. A continuación, a partir de los operadores algebraicos definidos sobre los modelos y de forma independiente al metamodelo de trazabilidad, se extrae el modelo de trazas de forma automática. Además, la propuesta también permite que el usuario defina las trazas de forma manual en un modelo de trazas conforme al metamodelo seleccionado.

Para dar soporte tecnológico a esta propuesta, los autores emplean la herramienta MOMENT (MOdel manageMENT, http://moment.dsic.upv.es/) que ha sido implementada como extensión del entorno de trabajo Eclipse. Concretamente ha sido construida sobre Maude para las especificaciones algebraicas y sobre EMF para llevar a cabo el modelado.

En cuanto a los datos extraídos del análisis de la propuesta:

- **Identificación de las relaciones de trazabilidad.** Los autores proponen el empleo de especificaciones algebraicas para determinar operaciones entre modelos. A partir de dichas especificaciones se identifican automáticamente las relaciones de trazabilidad existentes entre los elementos de los modelos.
- **Nivel de abstracción en que se identifican las relaciones de trazabilidad.** Esta propuesta no está orientada a la generación de trazabilidad a partir del desarrollo de transformaciones de modelos. Por este motivo, este criterio no aplica en esta propuesta.
- **Generación de enlaces de traza.** Los autores proponen que las trazas se generen automáticamente cada vez que se ejecuta un operador algebraico que realiza una operación entre dos modelos relacionados.
- **Almacenamiento.** La propuesta defiende que las trazas generadas automáticamente se almacenen en modelos de trazas conformes al metamodelo de trazabilidad definido por el usuario o al metamodelo de trazabilidad que ofrece por defecto.
- **Metamodelo de trazabilidad.** Los autores presentan un metamodelo de trazabilidad genérico descrito en términos de UML. Dicho metamodelo

permite establecer las relaciones entre distintos elementos, independientemente de cómo sean sus correspondientes metamodelos.

- **Visualización.** En las conclusiones de este trabajo, los autores indican que están analizando el editor propuesto por la herramienta AMW para editar y visualizar los modelos.
- **Análisis de las trazas.** La propuesta presentada no considera el análisis de la información de trazabilidad generada.
- **Soporte tecnológico.** MOMENT es la herramienta que da soporte a esta propuesta metodológica. Se trata de una herramienta desarrollada sobre el entorno Eclipse que proporciona un conjunto de operadores genéricos que permiten interactuar con modelos EMF. Además, permite importar artefactos software desde fuentes de datos heterogéneas como modelos UML, esquemas de bases de datos relacionales (definidas con la herramienta Rational Rose) o modelos XML.

3.3.4.3 N. Drivalos *et al.*

Los autores Kolovos, Paige y Polack presentan en [112] un trabajo centrado en analizar las diferentes formas de almacenar la información de trazabilidad de modelos. Esta propuesta no está centrada en la gestión de la trazabilidad en el desarrollo de transformaciones de modelos sino en la gestión de los metamodelos de trazabilidad. Sin embargo, sus aportaciones en cuanto a la especificación de los metamodelos resultan de gran interés para el objetivo de esta revisión de la literatura.

Los autores analizan dos aproximaciones: almacenar las trazas en los modelos terminales que describen el sistema y crear modelos de trazas. La primera aproximación facilita la visualización y el análisis por parte de observadores humanos. Sin embargo, provoca contaminación en el modelo terminal, ya que incluye información que no es propia de la especificación del sistema. La segunda aproximación defiende el empleo de un modelo de trazas encargado de almacenar toda esta información, de forma que los modelos terminales que especifican el sistema se mantienen limpios de contaminación. Los modelos de trazas deben ser conformes a un metamodelo de trazabilidad, que puede ser genérico para todos los escenarios de trazabilidad o específico para cada escenario concreto. La principal desventaja de esta aproximación se encuentra a la hora de analizar los modelos y las relaciones de trazabilidad, ya que la información se encuentra dispersa en varios modelos.

Ambas aproximaciones tienen ventajas e inconvenientes, por ello, los autores proponen combinar ambas ideas. Así, proponen generar modelos externos que contengan la trazabilidad y posteriormente, fusionarlos (*merging*) con los modelos terminales del sistema, de forma que se consiguen modelos "a la carta". Para llevar a cabo esta fusión entre los modelos, los autores proponen el empleo de EML (*Epsilon Merging Language* [110]), un lenguaje basado en reglas que permite interactuar directamente con los modelos.

Continuando el estudio de las diferentes alternativas para el almacenamiento de la trazabilidad, los autores Drivalos, Paige, Fernandes y Kolovos presentan, en [58], un estudio acerca de cómo almacenar de las trazas en modelos de trazabilidad. Como se indica en [112], es posible emplear un metamodelo de trazabilidad genérico o emplear metamodelos específicos para cada uno de los escenarios de trazabilidad.

Los metamodelos genéricos permiten modelar todos los escenarios de trazabilidad y favorecen la interoperabilidad entre los modelos de trazas, entendida como la característica que permite que un modelo generado mediante una herramienta de gestión de la trazabilidad pueda ser manejado por otra. Este tipo de metamodelos facilitan la interoperabilidad porque, mediante una transformación definida entre los metamodelos de trazabilidad de las herramientas, es posible obtener modelos de trazas conformes a todos ellos. Por otra parte, su principal inconveniente es que no siempre se ajusta fielmente a todas las posibles relaciones de trazabilidad.

En cambio, el uso de un metamodelo específico para cada escenario sí permite ajustarse a las características del contexto, ya que el metamodelo es diseñado en torno a dichas características. Sin embargo plantea dos inconvenientes: mayor esfuerzo de construcción y reduce la interoperabilidad. La construcción de un metamodelo para cada escenario de trazabilidad implica un mayor esfuerzo para los desarrolladores. En cuanto a la interoperabilidad, al contrario que en el caso anterior, no es posible definir una transformación general que proporcione interoperabilidad entre los escenarios, ya que cada escenario dispone de un metamodelo específico.

En [58], los autores defienden el empleo de metamodelos de trazabilidad específicos por ser más rigurosos a la hora de definir las relaciones de trazabilidad. Además indican que, gracias a las tecnologías de gestión de modelos, sus desventajas se verán reducidas. Continuando con esta línea de investigación, en [56], presentan TML (*Traceability Metamodelling Language*), un lenguaje para reducir el esfuerzo que supone la construcción de metamodelos de trazabilidad.

Para solucionar el problema de la interoperabilidad entre los modelos, proponen el desarrollo de transformaciones HOT entre los metamodelos conformes al metamodelo de TML. En [57] se presenta una mejora de la propuesta que permite mantener y evolucionar automáticamente la trazabilidad modelada con TML, mediante el empleo de *scripts*.

Las ideas anteriores pueden verse aplicadas en [159], donde se plantean de forma unificada las conclusiones obtenidas en los estudios anteriores para por un lado, demostrar cómo identificar los tipos de enlaces de traza y por otro, describir una aproximación para definir la semántica de las trazas.

En cuanto a los datos extraídos del análisis de la propuesta:

- **Identificación de las relaciones de trazabilidad.** La propuesta no proporciona información acerca de cómo identificar las relaciones de trazabilidad.
- **Nivel de abstracción en que se identifican las relaciones de trazabilidad.** Al igual que en el punto anterior: no indica cómo identificar las relaciones de trazabilidad.
- **Generación de enlaces de traza.** La propuesta no tiene por objetivo generar enlaces de traza. Sin embargo, ya que proporciona TML para construir metamodelos de trazabilidad, sería posible crear manualmente modelos de trazas conformes a dichos metamodelos.
- **Almacenamiento.** Esta propuesta realiza un profundo estudio acerca de cómo se deberían almacenar las trazas. Analiza las ventajas y los inconvenientes de dos aproximaciones: almacenar las trazas en los modelos terminales que describen el sistema y crear modelos de trazas para almacenamiento de esta información. En [112], proponen combinar ambas aproximaciones para obtener 'modelos a la carta'. Sin embargo, a partir de [58], los autores comienzan a defender y analizar el almacenamiento en modelos de trazas
- **Metamodelo de trazabilidad.** Para almacenar las trazas en modelos de trazas, estudian la conveniencia de emplear metamodelos de trazabilidad genéricos o crear metamodelos específicos para cada escenario de trazabilidad. Tras evaluar las dos alternativas determinan que ambas presentan ventajas e inconvenientes pero prefieren emplear metamodelos de trazabilidad específicos por ser más rigurosos a la hora de definir las trazas. Para facilitar la construcción de estos metamodelos presentan el lenguaje TML.
- **Visualización.** Los autores de la propuesta no consideran la visualización de la información de trazabilidad.

- **Análisis de las trazas.** En [159] realizan un estudio centrado en mostrar cómo identificar los tipos de enlace de traza implicados en un proceso MDE. Se trata de un proceso denominado TEAP (*Traceability Elicitation and Analysis Process*) para guiar la construcción de clasificaciones de trazas.
- **Soporte tecnológico.** No dispone de herramienta que soporte la propuesta. Sin embargo, en [159] indican que están trabajando en una herramienta de soporte.

3.3.4.4 B. Grammel y S. Kastenholz

En [77], Grammel y Kastenholz realizan un estudio acerca de los retos, problemas y limitaciones a los que se enfrenta el desarrollo de software dirigido por modelos en cuanto a la gestión de la trazabilidad. Entre estas cuestiones destacan la gran cantidad de metamodelos de trazabilidad existentes y las diferentes aproximaciones empleadas por los motores de transformación para la gestión de la trazabilidad (gestión implícita y gestión explícita).

La gestión implícita de la trazabilidad consiste en obtener automáticamente las trazas a partir de las relaciones de transformación definidas. Esta aproximación implica algunas desventajas derivadas de su automatización, como por ejemplo que el desarrollador no participa en la definición y creación de las trazas, por tanto no puede adaptarlas a sus necesidades. Además, en estos casos cada motor de transformación define su propio metamodelo de trazabilidad, por lo que surge un problema de interoperabilidad a la hora de manejar trazas generadas a partir de distintos motores de transformación. En cambio, la gestión explícita sí necesita del trabajo del desarrollador para generar las trazas. Esta aproximación implica un mayor esfuerzo para los desarrolladores ya que es necesario que incluyan, explícitamente, las relaciones de trazabilidad en las reglas de transformación.

Dado que ambas aproximaciones tienen ventajas e inconvenientes, los autores presentan una propuesta que permite unificar los metamodelos de trazabilidad para favorecer la interacción entre los modelos de trazas, adecuar el metamodelo a las necesidades concretas de los escenarios de trazabilidad y minimizar los esfuerzos de los desarrolladores para generar trazabilidad. El objetivo principal es el desarrollo de un *framework* de trazabilidad genérico que extienda las funcionalidades proporcionadas por los actuales motores de transformaciones de modelos. De forma que se aprovechen las ventajas que ofrecen estos motores, independientemente de si gestionan la trazabilidad implícita o explícitamente.

La propuesta presentada consiste en una interfaz genérica llamada GTI (*Generic Traceability Interface*) que sirve para establecer un punto de conexión

con los diferentes mecanismos de transformaciones de modelos. GTI permite obtener, de forma automática, las relaciones de trazabilidad que son generadas en la transformación, ya sea de forma implícita o explícita y almacenarlas en un repositorio. Para el tratamiento de la información de la trazabilidad, de forma genérica e independiente de los conectores empleados, presentan un lenguaje de dominio específico para trazabilidad: Trace DSL. El lenguaje Trace-DSL proporciona un metamodelo de trazabilidad de propósito general que permite definir relaciones de creación, actualización, consulta y borrado entre diferentes artefactos. Además, dicho metamodelo puede ser extendido mediante el empleo de facetas para cada artefacto y relación.

En cuanto a los datos extraídos del análisis de la propuesta:

- **Identificación de las relaciones de trazabilidad.** El trabajo presentado propone que las relaciones de trazabilidad se identifican automáticamente o manualmente, dependiendo del motor de transformación al que se extienda. De forma que si dicho motor de transformación ofrece soporte para la gestión implícita de la trazabilidad, las relaciones se identificarán automáticamente. En caso contrario, las relaciones de trazabilidad se deben definir manualmente por el usuario.
- **Nivel de abstracción en que se identifican las relaciones de trazabilidad.** La propuesta no establece ningún nivel de abstracción para especificar la transformación, solo propone el uso de una interfaz genérica para extender motores de transformación ya existentes. Por tanto, se asume que el nivel de abstracción en el que se identifican las relaciones de trazabilidad es el más bajo, es decir la transformación ejecutable.
- **Generación de enlaces de traza.** La propuesta presentada consiste en una interfaz genérica que extienda la funcionalidad de los motores de transformación existentes, independientemente de si gestionan la trazabilidad de forma implícita o explícita. En cualquier caso, las trazas se generan automáticamente a partir de la ejecución de la transformación, sea cual sea el motor de transformación empleado.
- **Almacenamiento.** Los autores proponen que las trazas generadas sean almacenadas en un repositorio creado para cumplir con este objetivo. Dichas trazas deberán ser almacenadas en un formato conforme al metamodelo definido por Trace-DSL.
- **Metamodelo de trazabilidad.** En esta propuesta se presenta Trace-DSL, un lenguaje específico de trazabilidad. Dicho lenguaje proporciona un

metamodelo de trazabilidad genérico que puede extenderse a través de facetas para cada artefacto y relación.

- **Visualización.** Los autores no especifican cómo se debería visualizar o representar la información de trazabilidad obtenida.
- **Análisis de las trazas.** La propuesta no considera el análisis de la información de trazabilidad generada.
- **Soporte tecnológico.** Los autores no especifican que han implementado la propuesta y dado que no se ha encontrado ninguna herramienta que ofrezca soporte, se asume que no se dispone de soporte tecnológico.

3.3.4.5 E. Guerra *et al.*

En [81], los autores Guerra, De Lara, Kolovos y Paige presentan una aproximación basada en el empleo de patrones declarativos triples para definir especificaciones inter-modelado. Esta propuesta no tiene por objetivo dar soporte para la gestión de la trazabilidad en el desarrollo de transformaciones. Sin embargo, ha sido incluida en este estado del arte porque el empleo de especificaciones declarativas que se presenta puede trasladarse al desarrollo de transformaciones de modelos, mediante el empleo de diferentes lenguajes de transformación declarativos o híbridos.

En este contexto, el concepto 'inter-modelado' se entiende como la actividad de construir modelos que describen cómo deben relacionarse los lenguajes de modelado. Esta definición incluye diferentes tareas, propias de la Ingeniería Dirigida por Modelos, como por ejemplo las transformaciones de modelos o la gestión de la trazabilidad. Los autores critican que, en muchas ocasiones, se emplea una especificación para cada una de las tareas, a pesar de estar interactuando con los mismos lenguajes de modelado. Así, los autores defienden que a partir de una especificación inter-modelado que defina las relaciones entre dos lenguajes de modelado sería posible llevar a cabo diferentes tareas como transformaciones o generación de trazas entre dichos modelos.

Para validar la propuesta, los autores presentan un caso de estudio centrado en dos escenarios de inter-modelado concretos: comparativa de modelos bajo condiciones específicas (*model matching*) y generación de trazas.

Además, para dar soporte tecnológico a la propuesta presentan PAMOMO (http://astreo.ii.uam.es/~eguerra/tools/pamomo/main.htm), una herramienta que permite llevar a cabo la definición de especificaciones inter-modelado y a partir de ellas, realizar operaciones como la construcción de modelos de trazas o verificar si un modelo de trazas es correcto. Para traducir y ejecutar las especificaciones, esta

herramienta emplea el lenguaje EOL (*Epsilon Object Language* [109]), un lenguaje que permite interactuar directamente con modelos EMF. Para llevar a cabo la ejecución de las especificaciones proporciona dos modos de ejecución: *offline* y *online*. En el primero de ellos, la especificación se compila una vez y puede usarse para cualquier modelo de entrada. En el modo *online*, el desarrollador elige los modelos de entrada y salida y la especificación es aplicada directamente sobre ellos. Como resultado de la ejecución se obtiene un modelo de trazas que puede ser visualizado en *ModeLink* (http://www.eclipse.org/epsilon/doc/modelink/), un editor desarrollado por Epsilon que permite visualizar al mismo tiempo el modelo de origen, el modelo destino y el modelo que contiene las relaciones entre los elementos de los modelos de origen y destino.

En cuanto a los datos extraídos del análisis de la propuesta:

- **Identificación de las relaciones de trazabilidad.** Las relaciones de trazabilidad se identifican manualmente a partir de la definición de los patrones declarativos que permiten realizar operaciones entre dos modelos.
- **Nivel de abstracción en que se identifican las relaciones de trazabilidad.** La propuesta no está orientada a la generación de trazas a partir del desarrollo y ejecución de transformaciones de modelos, por ello, las relaciones de trazabilidad no se relacionan con la definición de transformaciones. Por tanto, este dato no aplica en esta propuesta.
- **Generación de enlaces de traza.** Los autores presentan en este trabajo una propuesta para la definición de patrones declarativos que permitan llevar a cabo operaciones inter-modelado. En este trabajo presentan un caso de estudio centrado en dos escenarios, uno de ellos consiste en la creación automática de modelos de trazas a partir de la definición de especificaciones inter-modelado.
- **Almacenamiento.** La propuesta metodológica no especifica cómo se deberían almacenar las trazas generadas. Sin embargo, en la sección donde se presenta la herramienta que da soporte a dicha propuesta, los autores especifican claramente que las trazas son almacenadas en un modelo EMF.
- **Metamodelo de trazabilidad.** Los autores no presentan ningún metamodelo de trazabilidad. En cambio, sí presentan un metamodelo genérico para el modelado de patrones declarativos que relacionan dos lenguajes de modelado.
- **Visualización.** Los autores no especifican en la propuesta cómo deberían visualizarse los modelos de trazas. Sin embargo, en la herramienta que da

soporte tecnológico a la propuesta (PAMOMO) emplean el editor *ModeLink* para visualizar los modelos de trazas obtenidos.

- **Análisis de las trazas.** La propuesta presentada no considera el análisis de la información de trazabilidad generada.
- **Soporte tecnológico.** La propuesta metodológica presentada en este trabajo es soportada por la herramienta PAMOMO. Se trata de una herramienta construida sobre el entorno Eclipse y permite llevar a cabo la definición de especificaciones inter-modelado y a partir de ellas, realizar operaciones sobre modelos EMF como la construcción de modelos de trazas. El modelo de trazas resultante puede ser visualizado a través de *ModeLink*, un editor desarrollado por Epsilon que permite visualizar al mismo tiempo un modelo de origen, uno de destino y otro que contenga las relaciones existentes entre los elementos de los anteriores.

3.3.4.6 F. Jouault *et al.*

En [97], Jouault presenta una aproximación para la gestión de trazabilidad en el desarrollo de transformaciones de modelos, implementadas en ATL. Jouault propone definir explícitamente las relaciones de trazabilidad en las reglas que implementan la transformación.

Dado que uno de los principios de MDE es el empleo de modelos a lo largo de todo el ciclo de vida, el autor propone almacenar las trazas generadas en un modelo más del proceso de desarrollo. Para ello, propone un metamodelo de trazabilidad genérico que permite definir elementos de los modelos y las relaciones entre ellos.

Para aplicar esta idea a transformaciones ya existentes y facilitar la construcción de nuevas transformaciones se presenta *TraceAdder*, una herramienta compuesta por una transformación ATL. Dado que la herramienta que da soporte al lenguaje ATL permite convertir el código de la transformación en un modelo, el funcionamiento de *TraceAdder* consiste en ejecutar una transformación sobre un modelo de transformación ATL para obtener un nuevo modelo de dicha transformación que contenga la información necesaria para generar el modelo de trazas.

Las principales contribuciones de este trabajo son: tratar a las transformaciones de modelos como un modelo más del proceso de desarrollo y la incorporación explicita de las relaciones de trazabilidad en las reglas de la transformación. La principal ventaja de esta propuesta es el débil acoplamiento que existe entre la generación de la trazabilidad y el motor de la transformación,

ya que el usuario tiene la posibilidad de definir explícitamente qué relaciones existen y cómo modelarlas. Así, la generación de las trazas depende de las decisiones del desarrollador y no de cómo el motor de transformación gestiona la trazabilidad.

Las ideas propuestas por Jouault en [97] han servido como guía para un gran número de trabajos posteriores:

Así, en [121] se presenta un estudio acerca de cómo modularizar el desarrollo de transformaciones implementadas en lenguajes basados en reglas. Los autores presentan varios escenarios implementados en ATL, uno de ellos es la gestión de trazabilidad en el desarrollo de transformaciones de modelos y para ello, incluyen la información de la trazabilidad en las reglas de transformación, tal como propone Jouault.

En [17], Barbero, Didonet del Fabro y Bézivin presentan una clasificación que divide la gestión de la información de trazabilidad en dos niveles: nivel microscópico y nivel macroscópico. El nivel microscópico se corresponde con la relación de trazabilidad entre elementos de diferentes modelos y el nivel macroscópico establece relaciones de trazabilidad entre los modelos. Para llevar a cabo la gestión de la trazabilidad a nivel microscópico, emplean la propuesta que presenta Jouault.

Freddy Allilaire presenta en [6] un pequeño estudio centrado en el empleo de ATL en el contexto de la herramienta *Obeo Traceability*. Esta herramienta proporciona un mecanismo para la gestión de la trazabilidad interna. Para proporcionar esta funcionalidad emplea la gestión interna de la trazabilidad que realiza el motor de transformación de ATL. También permite la gestión explícita de la trazabilidad basándose en la idea presentada por Jouault.

En [63], los autores Falleri, Huchard y Nebut presentan la implementación de un *framework* para la gestión de la trazabilidad implementado en Kermeta (http://www.kermeta.org), un lenguaje orientado a modelos. Este *framework* presenta un metamodelo de trazabilidad genérico que, como indican los propios autores, se encuentra inspirado en el trabajo de Jouault. Para generar los modelos de trazabilidad, emplean también la idea de Jouault, ya que la generación de las trazas se encuentra implícita en la transformación de los modelos.

En cuanto a los datos extraídos del análisis de la propuesta:

- **Identificación de las relaciones de trazabilidad.** La idea inicial que propone el autor consiste en la identificación manual de las relaciones de trazabilidad por parte del desarrollador de la transformación. Sin embargo, proporciona la herramienta *TraceAdder* que permite identificar las relaciones de trazabilidad

de forma automática a partir de las relaciones existentes en las reglas de la transformación.

- **Nivel de abstracción en que se identifican las relaciones de trazabilidad.** La propuesta sugiere que las relaciones de trazabilidad sean identificadas a nivel del código que implementa la transformación. Sin embargo, la herramienta que da soporte a la propuesta metodológica identifica las relaciones de trazabilidad a nivel de un modelo conforme al metamodelo que describe el lenguaje ATL, es decir, identifica las relaciones de trazabilidad a nivel PDM.
- **Generación de enlaces de traza.** El autor propone aprovechar la ejecución de las transformaciones de modelos ATL para generar al mismo tiempo y de forma automática las trazas entre los elementos implicados en la transformación.
- **Almacenamiento.** El autor se basa en uno de los principios de MDE (uso de los modelos a lo largo de todo el ciclo de vida) para defender que las trazas deberían almacenarse y tratarse como un modelo más del sistema. Por ello, propone que las trazas generadas a partir de las transformaciones de modelos sean almacenadas en un modelo de trazas.
- **Metamodelo de trazabilidad.** Para almacenar y modelar las trazas del sistema, el autor propone un metamodelo de trazabilidad genérico básico que permite describir elementos de los modelos y las relaciones entre dichos elementos.
- **Visualización.** En [97], el autor no especifica cómo se deberían visualizar o representar los modelos de trazas. Sin embargo en trabajos posteriores, se presentan propuestas para visualizar este tipo de modelos mediante los editores proporcionados por las herramientas AMW y EMF. El editor propuesto por AMW permite representar de forma genérica las relaciones entre varios modelos, mientras que el editor proporcionado por EMF permite visualizar cualquier tipo de modelo conforme al metamodelo *Ecore*, descrito por EMF.
- **Análisis de las trazas.** Esta propuesta no considera realizar ningún tipo de análisis sobre las trazas generadas.
- **Soporte tecnológico.** En [97] se presenta *TraceAdder*, una herramienta compuesta por una transformación tipo HOT, implementada en el lenguaje de transformación ATL. *TraceAdder* recibe como entrada un modelo que describe una transformación ATL (nivel PDM) y genera automáticamente otro modelo de la transformación. El modelo ATL resultante será el mismo pero añadiendo,

a cada una de las reglas de la transformación, la información necesaria para que al ejecutar el código que implementa la transformación se obtenga, además del modelo de salida, un modelo con las trazas existentes entre los elementos implicados en la transformación.

3.3.4.7 T. Levendovszky *et al.*

Los autores Levendovszky, Balasubramanian, Smith, Shi y Karsai presentan, en [128], una propuesta para la gestión de la trazabilidad en el desarrollo dirigido por modelos de sistemas aéreos. En este contexto, la gestión de la trazabilidad se considera un aspecto obligatorio durante el desarrollo.

Según los principios del desarrollo dirigido por modelos, todas las modificaciones o evoluciones del sistema deben ser consecuencia de la ejecución de una transformación. Por este motivo, los autores presentan una aproximación para la gestión de la trazabilidad a partir del desarrollo y ejecución las transformaciones, ya sean de tipo M2M o M2T. El método propuesto se basa en la idea propuesta por Jouault en [97]: definir las relaciones de trazabilidad en la especificación de la transformación y a partir de dichas relaciones, obtener automáticamente las trazas como resultado de la ejecución de la transformación.

Para llevar a cabo las transformaciones entre los modelos, proponen el uso de *GReAT*, una aproximación basada en la reescritura de grafos [16]. Por otro lado, para las transformaciones M2T, o generación de código, proponen que sean implementadas mediante lenguajes de programación imperativos. Como resultado de ambos tipos de transformaciones se obtiene por un lado, el modelo de salida o el código de salida y por otro lado, las trazas. Los autores proponen que las trazas generadas sean representadas de formas distintas dependiendo del tipo de transformación ejecutada. Así, si es una transformación, las trazas deberán ser representadas en un formato orientado a grafos. En cambio si se trata de una generación de código, las trazas serán representadas de forma textual. Para gestionar y establecer la estructura de las trazas generadas entre los modelos presentan un metamodelo de trazabilidad genérico. Este metamodelo permite representar los elementos que forman parte de las reglas de la transformación y las trazas que existen entre ellos.

En las conclusiones de [128], los autores indican que han implementado una herramienta que muestra las trazas entre los elementos de los modelos y el código del sistema. Sin embargo, no proporcionan más información acerca de dicha herramienta, por tanto no ha sido posible encontrar los medios para acceder a ella. Como trabajo futuro plantean, como principal dirección, examinar la

posibilidad de incluir las relaciones de trazabilidad de forma automática en la transformación.

En cuanto a los datos extraídos del análisis de la propuesta:

- **Identificación de las relaciones de trazabilidad.** La propuesta sigue una aproximación similar a la presentada por Jouault en [97] para la creación de la información de trazabilidad: las relaciones de trazabilidad son incluidas directamente en la transformación. En esta propuesta no se proporcionan medios para automatizar la identificación de estas relaciones, por tanto, es una tarea manual que debe realizar el desarrollador de la transformación. Sin embargo, en las conclusiones de [128] plantean como principal línea de investigación futura analizar la posibilidad de automatizar esta tarea.
- **Nivel de abstracción en que se identifican las relaciones de trazabilidad.** Esta propuesta no establece ningún nivel de abstracción para el desarrollo y especificación de las transformaciones. Por ello, el nivel al que se identifican las relaciones de trazabilidad es el nivel de menor abstracción posible, es decir la propia transformación ejecutable.
- **Generación de enlaces de traza.** Como ya se ha dicho anteriormente, esta propuesta sigue una aproximación similar a la presentada por Jouault. Por tanto, al igual que Jouault, los autores proponen que los enlaces de traza sean generados automáticamente a partir de la ejecución de las transformaciones de modelos que contienen las relaciones de trazabilidad definidas entre los elementos que forman parte de las reglas de la transformación.
- **Almacenamiento.** En cuanto al almacenamiento de las trazas generadas, los autores proponen el empleo de dos estructuras: una para almacenar las trazas generadas a partir de transformaciones M2M y otra para las trazas generadas a partir de transformaciones M2T. La propuesta no detalla cómo son dichas estructuras, tan solo indica que en la primera las trazas son representadas en forma de grafos y en la segunda, de forma textual.
- **Metamodelo de trazabilidad.** Los autores presentan un metamodelo de trazabilidad genérico para establecer la estructura de las trazas generadas a partir de la ejecución de las transformaciones de modelos. Este metamodelo permite describir los elementos que forman parte de las reglas de la transformación y las trazas existentes entre ellos.
- **Visualización.** Los autores proponen que las trazas generadas se representen de forma diferente, dependiendo del tipo de transformación ejecutada. Así, en el caso de transformaciones entre modelos, las trazas son obtenidas en un

formato orientado a grafos y en el caso de transformaciones de modelo a texto, de forma textual.

- **Análisis de las trazas.** La propuesta no considera el análisis de las trazas generadas.
- **Soporte tecnológico.** En las conclusiones del trabajo, los autores indican que han implementado una herramienta para representar las trazas generadas entre los elementos de los modelos y el código del sistema, sin embargo no especifican las características de dicha herramienta ni los medios para acceder a ella.

3.3.4.8 Oldevik *et al.*

En los trabajos [145] y [149], los autores presentan una aproximación para la gestión de la trazabilidad en el desarrollo de transformaciones M2T. Dado que el código que implementa el sistema es generado mediante estas transformaciones, los autores defienden que, a través de ellas, es posible generar las trazas que relacionan los elementos de los modelos con los bloques de código que los implementan.

En estos trabajos, los autores analizan dos lenguajes de generación de código para la gestión de la trazabilidad en el desarrollo de transformaciones de modelo a texto: *MOF Model to Text Transformation* [158] y *MOFScript* [144]. La principal diferencia que existe entre ambas aproximaciones es que *MOF Model to Text Transformation* gestiona la trazabilidad de forma explícita y *MOFScript* lo hace de forma implícita. A pesar de esta diferencia, según los autores, ambas aproximaciones pueden ser igualmente válidas. Sin embargo, los autores finalmente deciden presentar su propuesta usando *MOFScript* y de hecho, ya en [149] se centran totalmente en el desarrollo de la propuesta en torno a este lenguaje. El lenguaje *MOFScript* dispone de su propio metamodelo de trazabilidad que permite definir referencias a los elementos de los modelos, bloques de código de los ficheros fuente y las trazas existentes entre ellos. A partir de este metamodelo, cuando se ejecuta una generación de código, además de generar el código fuente, es posible crear un modelo de trazas.

Para implementar esta propuesta, los autores emplean la herramienta que da soporte al lenguaje *MOFScript* para definir y ejecutar las transformaciones M2T. Además, en [149] presentan un prototipo denominado *Traceability Model Analysis* para visualizar las trazas generadas, conocer qué partes del sistema no tienen trazas asociadas, identificar trazas obsoletas y realizar un análisis de impacto del cambio. En [149] también plantean como trabajo futuro el desarrollo de una

aproximación intermedia que permita generar trazabilidad de forma automática y de forma manual, así como definir un metamodelo más genérico que no solo sea válido en el contexto de las transformaciones M2T.

Continuando en esta línea de investigación, en [138], Svein J. Melby presenta su tesis de máster que es dirigida por Olsen y Berre. Esta tesis proporciona una visión global de la trazabilidad en el contexto de MDE y sus principales aportaciones son: un metamodelo de trazabilidad genérico y un editor gráfico para el modelado de la trazabilidad, implementado usando el *framework* Eclipse GMF (http://www.eclipse.org/modeling/gmf/). El metamodelo de trazabilidad propuesto no está orientado sólo a la gestión de la trazabilidad en la generación de código, sino que permite modelar trazabilidad a lo largo de todo el ciclo de desarrollo del sistema.

En cuanto a los datos extraídos del análisis de la propuesta:

- **Identificación de las relaciones de trazabilidad.** Esta propuesta se basa en aprovechar las ventajas que ofrece el motor de generación de código *MOFScript*. Dicho motor proporciona un mecanismo de gestión implícita de la trazabilidad que es empleado para la identificación automática de las relaciones de trazabilidad.
- **Nivel de abstracción en que se identifican las relaciones de trazabilidad.** Al igual que otras propuestas anteriores, esta tampoco define niveles de abstracción para especificar las transformaciones. Además, en este caso las relaciones de trazabilidad son gestionadas de forma implícita por el motor que ejecuta la transformación. Por todo ello, se asume que las relaciones de transformación son identificadas a nivel de la transformación ejecutable.
- **Generación de enlaces de traza.** A pesar de que en [138], Melby proporcione un editor gráfico que permita crear manualmente modelos de trazas, el núcleo de la propuesta aboga por crear los enlaces de traza de forma automática a partir de la ejecución de transformaciones de modelo a texto.
- **Almacenamiento.** Los autores no describen explícitamente cómo se deberían almacenar las trazas generadas, sin embargo en todo momento asumen que las trazas serán recogidas y almacenadas en un modelo de trazas creado específicamente para dicho objetivo.
- **Metamodelo de trazabilidad.** Debido a la evolución constante de la propuesta, en cada uno de los trabajos analizados se propone un nuevo metamodelo de trazabilidad genérico. Los metamodelos presentados en [145] y [149] permiten definir trazas entre elementos de modelos y segmentos o bloques de código susceptibles de ser trazados. Dado que el metamodelo

presentado en [138] está diseñado no solo para modelar trazas a partir de transformaciones M2T, permite definir trazas entre dos artefactos cualesquiera del sistema.

- **Visualización.** Al igual que en el punto anterior, debido a la evolución de la propuesta se pueden encontrar diferentes tipos de representaciones propuestas. Así, mientras en [145] los modelos de trazas se representan mediante modelos genéricos en formato UML, en [149] se presenta un prototipo denominado TAP que, aunque se encuentra orientado a ofrecer análisis de trazabilidad, proporciona un visor que muestra las trazas de forma jerárquica. Finalmente, en [138] se presenta un editor gráfico desarrollado con GMF que permite editar y visualizar gráficamente las trazas del sistema.
- **Análisis de las trazas.** En [149] presentan TAP (*Traceability Analysis Prototype*), un prototipo que permite realizar varios análisis sobre las trazas generadas. Las funcionalidades de este prototipo permiten llevar a cabo actividades como análisis de huérfanos, análisis de cobertura y análisis del impacto del cambio.
- **Soporte tecnológico.** Una vez más, se puede ver como la evolución de la propuesta deriva en diversas aproximaciones, pero en este caso en cuanto al soporte tecnológico de la propuesta presentada. En [145] no presentan ninguna herramienta, simplemente la propuesta se pone en práctica haciendo uso de la herramienta *MOFScript Tool*; en [149] presentan un prototipo, llamado *Traceability Model Analysis*, que permite visualizar y analizar las trazas generadas; y en [138] presentan un editor gráfico GMF para el modelado de trazas. Todas estas implementaciones han sido desarrolladas sobre el entorno Eclipse.

3.3.4.9 P. Sánchez *et al.*

Los autores Sánchez, Alonso, Rosique, Álvarez y Pastor presentan, en [175], una propuesta para la gestión de la trazabilidad de los requisitos de seguridad en el desarrollo dirigido por modelos de aplicaciones robóticas. La experiencia previa de los autores está centrada en el desarrollo de aplicaciones para robots de servicio para misiones de alto riesgo, como por ejemplo tareas nucleares. En este contexto la seguridad es un aspecto clave en el desarrollo del sistema y por tanto, es necesario asegurar el cumplimiento de los requisitos de seguridad. Esta propuesta se enmarca en un contexto concreto, sin embargo, puede ser extendida a otros tipos de sistemas.

Los autores defienden que la aplicación de técnicas de trazabilidad en el desarrollo dirigido por modelos, puede resultar muy beneficiosa para el desarrollo de sistemas que requieran un alto grado de seguridad, ya que la información obtenida a partir de las trazas permite analizar el desarrollo del sistema. Los autores se centran una de las ideas que sustentan esta tesis doctoral: en los desarrollos dirigidos por modelos, el sistema evoluciona a partir de transformaciones entre modelos. Así, dado que estas transformaciones establecen relaciones entre los elementos de los modelos, sería posible obtener las trazas de la del sistema a partir de ellas.

Para poner en práctica esta idea, los autores presentan un marco de trabajo para gestionar la trazabilidad de los requisitos de seguridad. El método propuesto consiste en: el usuario debe modelar los requisitos de seguridad y desarrollar una arquitectura de la solución de acuerdo a los requisitos. A continuación, el usuario debe definir un primer modelo de trazas conforme a un metamodelo de trazabilidad genérico, basado en el propuesto por Kolovos *et al*. en [112], que permite el modelado de elementos del sistema y las relaciones existentes entre ellos. A partir de esta información, en las sucesivas transformaciones que son implementadas mediante ATL (M2M) o JET (M2T), el *framework* se encarga de generar de forma automática los diferentes modelos de trazas.

Una de las principales aportaciones que realiza esta propuesta es la posibilidad de obtener un informe detallado con la información de las trazas, una vez que se han generado los modelos de trazas. Para proporcionar esta funcionalidad, los autores presentan la herramienta *Safety-Requirements-Trace-Tool* (SRTT). Esta herramienta genera, mediante una transformación de modelo a texto implementada en el lenguaje JET, un informe de trazabilidad en formato HTML. Según los autores, mediante estos informes, los usuarios tienen la posibilidad de conocer qué requisitos se han tenido en cuenta durante el desarrollo, qué elementos del sistema los soportan y qué relación existe entre ellos, así como realizar análisis de impacto del cambio.

En cuanto a los datos extraídos del análisis de la propuesta:

- **Identificación de las relaciones de trazabilidad.** Las relaciones de trazabilidad no son especificadas manualmente por el usuario en un primer modelo de trazas que servirá para construir los sucesivos modelos de trazas.
- **Nivel de abstracción en que se identifican las relaciones de trazabilidad.** Las relaciones de trazabilidad se identifican en un modelo que no se encuentra

relacionado con transformaciones que generan el sistema, por tanto, este dato no es aplicable a esta propuesta.

- **Generación de enlaces de traza.** Los autores proponen que el usuario defina manualmente el primer modelo de trazas y a partir de este modelo, se generan automáticamente otros dos modelos de trazas a lo largo del desarrollo del sistema. El primero de ellos es generado a partir de la ejecución de una transformación ATL (M2M) y el segundo es creado, en el momento de generar el código que implementa el sistema, como resultado de ejecutar una transformación en términos del lenguaje JET (M2T).
- **Almacenamiento.** La propuesta, que sigue los principios de MDE, asume en todo momento que las trazas generadas durante el desarrollo de los sistemas deben ser almacenadas en modelos de trazas. Estos modelos de trazas son conformes al metamodelo de trazabilidad definido.
- **Metamodelo de trazabilidad.** Los autores presentan un metamodelo de trazabilidad genérico inspirado en el propuesto por Kolovos *et al.* en [112]. El metamodelo que se propone permite definir elementos de los modelos y enlaces entre dichos elementos. Este metamodelo es tan genérico que podría emplearse para definir cualquier tipo de relación entre elementos de modelos, no solo trazas.
- **Visualización.** Esta propuesta no especifica cómo mostrar o representar los modelos de trazas obtenidos.
- **Análisis de las trazas.** En este trabajo, los autores presentan SRTT (*Safety-Requirements-Trace-Tool*), una herramienta implementada sobre JET que permite generar informes en formato HTML a partir de las trazas generadas. Según los autores, este tipo de informes permiten conocer qué requisitos se han tenido en cuenta durante el desarrollo del sistema, qué elementos del sistema los soportan y qué relación existe entre ellos, y llevar a cabo análisis de impacto del cambio.
- **Soporte tecnológico.** La propuesta metodológica ha sido puesta en práctica mediante el uso de algunas herramientas disponibles sobre el entorno de desarrollo Eclipse (ATL, JET, UML, EMF). Además en este trabajo, los autores presentan SRTT, una herramienta implementada *ad-hoc* para generar informes a partir de los modelos de trazas obtenidos.

3.3.4.10 P. Valderas y V. Pelechano

La propuesta presentada en [193] se enmarca en el contexto de la gestión de la trazabilidad de los requisitos en el desarrollo dirigido por modelos de aplicaciones web.

Dado que en un desarrollo dirigido por modelos, los requisitos se convierten en artefactos del sistema mediante la ejecución de transformaciones, según los autores, las transformaciones que guían el desarrollo, además de proporcionar nuevos artefactos, deberían facilitar información acerca de cómo se han ejecutado. Gracias a esta información será posible conocer si la transformación se ha ejecutado correctamente y si los artefactos resultantes se corresponden con los requisitos del sistema.

Valderas y Pelechano proponen un método basado en transformaciones de grafos en términos de la herramienta AGG (*Attributed Graph Grammars tool*, http://tfs.cs.tu-berlin.de/agg/). Dicho método parte de un modelo de requisitos basados en tareas para aplicaciones web, posteriormente es convertido, mediante una transformación XSL, a un formato XML conocido por AGG. A continuación, el usuario define la transformación en formato AGG, detallando las relaciones de trazabilidad existentes entre los modelos. A partir del modelo de requisitos se ejecuta la transformación y se obtiene un nuevo grafo que describe el modelo de navegación del sistema web, incluyendo las trazas generadas a partir de la transformación.

En este trabajo, además del método anterior, los autores presentan *TaskTracer*, una herramienta que permite analizar modelos de navegación que contienen información de trazabilidad en formato AGG. Mediante este análisis, la herramienta detecta y almacena cada una de las trazas existentes en un repositorio de trazas, de acuerdo a un metamodelo genérico inspirado en los trabajos de Jouault [97] y Kolovos *et al*. [112]. Este metamodelo permite mapear las relaciones existentes entre los elementos del modelo de navegación y los elementos del modelo que describe los requisitos.

Una vez que se han identificado y almacenado las trazas existentes, la herramienta genera un informe de trazabilidad en formato HTML. Este informe de trazabilidad permite validar si la transformación contempla todos los requisitos definidos y si lo hace de forma correcta. Además, determina el impacto de los cambios de los requisitos.

En cuanto a los datos extraídos del análisis de la propuesta:

- **Identificación de las relaciones de trazabilidad.** La propuesta presenta un método basado en transformaciones de grafos en el que los usuarios deben definir una transformación en formato AGG. El método requiere que los

usuarios incluyan manualmente las relaciones de trazabilidad en dicha transformación.

- **Nivel de abstracción en que se identifican las relaciones de trazabilidad.** Como se ha indicado en el punto anterior, los desarrolladores deben incluir manualmente las relaciones de trazabilidad en la transformación que se va a ejecutar, por tanto, se considera que la identificación de las relaciones se lleva a cabo en el nivel de abstracción más bajo.
- **Generación de enlaces de traza.** A partir de la información proporcionada por el usuario (modelo de requisitos y transformación AGG), se ejecuta la transformación y se obtiene automáticamente un nuevo grafo que describe el modelo de navegación del sistema web, incluyendo las trazas generadas.
- **Almacenamiento.** Los autores proponen que, en primer lugar, las trazas sean almacenadas en los modelos de navegación. Posteriormente, mediante el uso de la herramienta *TaskTracer*, cada una de las trazas encontradas en el modelo navegacional son almacenada en un repositorio de trazas que es definido de acuerdo a un metamodelo de trazabilidad.
- **Metamodelo de trazabilidad.** En este trabajo, los autores presentan un metamodelo de trazabilidad genérico basado en los metamodelos propuestos por Jouault [97] y Kolovos *et al*. [112]. Dicho metamodelo, que es empleado para describir al repositorio de trazas, permite establecer relaciones entre los elementos del modelo de navegación y los elementos del modelo de requisitos.
- **Visualización.** En este trabajo, los autores no especifican ninguna forma particular de visualizar la información de trazabilidad obtenida. En este sentido, solo consideran que el modelo de navegación que incluye las trazas se representa en forma de grafo y que a partir de dichas trazas se pueden generar informes en formato HTML.
- **Análisis de las trazas.** *TaskTracer*, el prototipo que se presenta en este trabajo es construido con el objetivo de generar informes de trazabilidad para facilitar el estudio acerca de cómo los requisitos del sistema son soportados por los modelos de navegación. Según los autores, este tipo de informes permiten: validar la completitud y la corrección de las transformaciones, es decir si se han tenido en cuenta todos los requisitos y se ha hecho de forma correcta; y establecer el impacto de los cambios en los requisitos.
- **Soporte tecnológico.** Las transformaciones que proponen en esta propuesta son implementadas sobre la herramienta AGG, por lo que se basa en una herramienta ya existente. Por otro lado, los autores presentan *TaskTracer*, un

prototipo que permite analizar modelos de navegación en formato AGG para identificar las trazas, almacenarlas en un repositorio y posteriormente, generar informes de trazabilidad en formato HTML. Los autores no especifican cómo se ha desarrollado este prototipo ni las herramientas o tecnologías empleadas.

3.3.5 Discusión

Una vez que se han analizado una por una todas las propuestas identificadas en la literatura, en esta sección se procede a: en primer lugar asignar a cada propuesta los valores correspondientes a los criterios de evaluación propuestos en la sección 3.3.2; y a continuación, analizar los valores obtenidos, para dar respuesta a las cuestiones de investigación propuestas en la sección 3.3.1.

En la Tabla 3-2 se presentan los valores asignados a cada propuesta por cada uno de los criterios de evaluación. Se recuerda que la puntuación de cada uno de los posibles valores es: S=1; P=0.5; y N=0. Por tanto, dado que se han definido siete criterios de evaluación (CE), cada una de las propuestas aspira a obtener, como máximo, siete puntos. La penúltima columna de la tabla ("Total") muestra la puntuación obtenida por cada una de las propuestas y la última columna ("% máximo posible") indica el porcentaje de puntos que ha obtenido cada propuesta respecto de 7, es decir del total de puntos posibles.

Como se puede comprobar en la Tabla 3-2 y al igual que la revisión de la literatura presentada anteriormente, todas las propuestas analizadas han obtenido una puntuación superior a cero puntos. De hecho el 80% de las propuestas han obtenido una puntuación igual o superior a 3.5 puntos, o lo que es lo mismo, el 50% de la puntuación máxima. Sin embargo, también cabe destacar que ninguna ha alcanzado más de 5.5 (80% de la puntuación máxima).

En las últimas dos filas de la tabla ("Total CE" y "% máximo posible CE") se presenta la puntuación alcanzada para cada uno de los criterios de evaluación, sumando los puntos asignados a cada propuesta para dicho criterio; y el porcentaje de puntos que supone respecto del total de puntos posibles. Se recuerda que se han analizado 10 propuestas y que el máximo valor que puede asignar por cada CE es 1, por tanto la puntuación máxima a la que aspira cada CE es de 10 puntos. En este caso se puede observar que mientras dos CE se acercan al máximo posible (CE-2 y CE-4), cuatro CE no alcanzan el 50% de los puntos posibles (CE-3, CE-5, CE-6, y CE-7).

Por todo lo anterior, se puede afirmar que las propuestas analizadas cumplen en parte con el objetivo buscado en esta revisión de la literatura, pero que ninguna cumple al 100% con los criterios establecidos. Así mismo se observa, que

los criterios de evaluación definidos tampoco son cubiertos por completo. Por todo ello, se deduce que es muy posible encontrar puntos de mejora en el estado del arte actual en gestión de la trazabilidad a partir del desarrollo de transformaciones de modelos.

Tabla 3-2. Resumen de los Criterios de Evaluación en cada propuesta

	Relaciones de Trazabilidad (Automáticas)	Generación de Trazas (Automáticas)	Independiente del lenguaje de transformación	Almacenamiento de las Trazas	Visualización de las Trazas	Análisis de Trazabilidad	Herramienta de Soporte		
Propuesta	**CE-1**	**CE-2**	**CE-3**	**CE-4**	**CE-5**	**CE-6**	**CE-7**	**Total**	**% máximo posible**
Bondé *et al.*	S	S	N	S	P	N	N	**3,50**	**50,00%**
Boronat *et al.*	S	S	S	S	P	N	S	**5,50**	**78,57%**
Drivalos *et al.*	N	N	S	S	N	S	N	**3,00**	**42,86%**
Grammel y Kastenholz	S	S	S	S	N	N	N	**4,00**	**57,14%**
Guerra *et al.*	N	S	S	P	P	N	P	**3,50**	**50,00%**
Jouault *et al.*	S	S	N	S	P	N	S	**4,50**	**64,29%**
Levendovszky *et al.*	P	S	N	P	P	N	P	**3,00**	**42,86%**
Oldevik *et al.*	S	S	N	S	S	S	P	**5,50**	**78,57%**
Sánchez *et al.*	N	S	N	S	N	S	P	**3,50**	**50,00%**
Valderas y Pelechano	N	S	N	S	N	S	P	**3,50**	**50,00%**

Total CE	**5,50**	**9,00**	**4,00**	**9,00**	**3,50**	**4,00**	**4,50**	**39,50**
% maximo posible CE	**55,00%**	**90,00%**	**40,00%**	**90,00%**	**35,00%**	**40,00%**	**45,00%**	**56,43%**

Una vez que se dispone de los valores asignados a cada propuesta, se procede a emplearlos para dar respuesta a cada una de las cuestiones de investigación (CI) planteadas en la sección 3.3.1:

- **CI-1.** ¿Qué nivel de automatización ofrecen o sugieren las propuestas metodológicas para generar información de trazabilidad?

Como se ha indicado en secciones y capítulos anteriores, uno de los principios que defiende MDE es incrementar el nivel de automatización de los procesos [14]. Así, dado que esta tesis doctoral se enmarca en el contexto de este paradigma, resulta interesante conocer cómo se pueden aprovechar las ventajas que ofrece el escenario propuesto por MDE para la gestión de la trazabilidad. Concretamente el uso de las técnicas de automatización empleadas en MDE, como

por ejemplo las transformaciones de modelos, simplifican y mejoran el proceso de generación automática de trazas entre artefactos software.

Siguiendo esta idea, es ampliamente reconocido que una de las mejores formas de automatizar la generación de los enlaces de trazas consiste en ser capaces de obtenerlas automáticamente a partir de las relaciones de trazabilidad especificadas a nivel de metamodelo. Además idealmente, sería deseable poder identificar dichas relaciones a nivel de metamodelo, por medio de alguna técnica de automatización.

Así, para conocer qué formas de automatización ofrecen las propuestas para la generación de trazas a partir del desarrollo de transformaciones de modelos, se analizan los resultados obtenidos en los criterios de evaluación CE-1 (automatización de la identificación de relaciones de trazabilidad) y CE-2 (automatización de la generación de trazas). Estos criterios han obtenido 5.5 (55% de la máxima puntuación) y 9 puntos (90%), respectivamente. Estos datos sugieren que mientras la automatización de la generación de trazas es una tarea ampliamente afrontada, la identificación automática de las relaciones de trazabilidad es una tarea que aún se encuentra inmadura.

De todas las propuestas estudiadas, se ha encontrado que tan solo una de ellas (Drivalos *et al.*) no considera ninguno de los dos tipos de automatización, aunque es necesario destacar que el principal objetivo de dicha propuesta se centra en otros aspectos como el almacenamiento de las trazas. Las otras nueve propuestas restantes sí proponen técnicas para automatizar al menos una de las dos tareas descritas. Concretamente, las nueve ofrecen mecanismos para la generación automática de las trazas, pero solo cinco de ellas (Bondé *et al.*, Boronat *et al.*, Grammel y Kastenholz, Jouault *et al.* y Oldevik *et al.*), además proponen técnicas para la identificación automática de las relaciones de trazabilidad. En el caso de la propuesta de Levendovzsky *et al.* no se indica cómo llevar a cabo automáticamente la identificación de las relaciones de trazabilidad, pero sí considera que es una tarea a tener en cuenta. En vista de este escenario, *a priori* parece más sencillo automatizar la generación de trazas que la identificación de las relaciones de trazabilidad.

Para llevar a cabo la generación automática de las trazas, la mayoría de las propuestas que dirigen sus esfuerzos a esta tarea (7 de 9) lo hacen a partir de la ejecución de transformaciones, solo las propuestas de Boronat *et al.* y Guerra *et al.* emplean otros métodos particulares. Sin embargo, observando los datos obtenidos para el criterio de evaluación CE-3 (independencia de un lenguaje de transformación), se comprueba que tan solo 4 de las 10 propuestas analizadas

(40%) presentan sus métodos de forma independiente al lenguaje de transformación que se utilice. Además, profundizando en estas cuatro propuestas se observa que: una de ellas (Drivalos *et al.*), como ya se ha mencionado, no sugiere métodos para automatizar esta tarea; otras dos propuestas independientes del lenguaje de transformación son las de Boronat *et al.* y Guerra *et al.* que son justamente aquellas que no proponen utilizar transformaciones para generar las trazas. Por tanto, la única propuesta que presenta un método para generar trazas a partir de la ejecución de transformaciones y que pueda emplearse para distintos lenguajes de transformación es la presentada por Grammel y Kastenholz.

El resto de propuestas que proponen la generación automática de trazas a partir de la ejecución de transformaciones lo hacen de forma dependiente a un lenguaje o motor de transformación concreto. Por tanto, trasladar estas propuestas a otros escenarios donde se usen otros lenguajes o motores de transformación puede resultar una tarea compleja y tediosa, e incluso en algunos casos, imposible. Así pues, sería deseable disponer de métodos o técnicas que alivien este problema.

En cuanto a cómo se automatiza la identificación de las relaciones de trazabilidad a nivel de metamodelo, solo un 50% de las propuestas analizadas dirigen sus esfuerzos en esta dirección, el resto propone que el usuario identifique las relaciones de forma manual. Las formas propuestas para automatizar esta tarea son diversas. Así, los trabajos de los autores Bondé *et al.* y los de Oldevik *et al.* proponen aprovechar la gestión implícita de la trazabilidad que proporcionan algunos motores de transformación. Boronat *et al.* proponen definir operadores algebraicos para realizar operaciones entre modelos y obtener las relaciones de trazabilidad a partir de esa información. Los autores Grammel y Kastenholz proponen una interfaz que extiende a los motores de transformación y que obtiene de él las relaciones de trazabilidad independientemente de cómo gestione la trazabilidad (implícita o explícita). Finalmente, Jouault presenta una propuesta híbrida: defiende que los desarrolladores incluyan las relaciones de trazabilidad en las reglas de la transformación, pero al mismo tiempo presenta una transformación que automatiza esta tarea.

En conclusión, para automatizar la generación de información de trazabilidad sería deseable automatizar por un lado, la identificación de las relaciones de trazabilidad a nivel de metamodelo y por otro, automatizar la generación de las trazas a partir de las relaciones anteriores. En este sentido, la generación automática de las trazas es una tarea más madura y la mayoría de las propuestas se ponen de acuerdo en el uso de transformaciones para llevar a cabo esta automatización. Sin embargo, la mayoría de estas aproximaciones son dependientes de un lenguaje o motor de transformación concreto. En cuanto a la

identificación automática de las relaciones de trazabilidad, resulta una tarea mucho más inmadura y los autores no han encontrado un consenso acerca de cómo se debería hacer.

- **CI-2.** Según las propuestas metodológicas identificadas, ¿cómo se deberían almacenar, visualizar y analizar las trazas generadas?

En la pregunta anterior se ha dado respuesta a cómo se genera la información de trazabilidad. Ahora se da a conocer cómo se trata dicha información una vez que se ha obtenido. Para dar respuesta a esta cuestión de investigación, los criterios CE-4, CE-5 y CE-6 evalúan cómo se comportan las propuestas de acuerdo al almacenamiento de las trazas, su visualización y al análisis de las mismas.

En cuanto al almacenamiento de las trazas se puede afirmar que es un aspecto bastante maduro del estado del arte ya que ha obtenido un 90% de la puntuación total (9 puntos de 10 posibles). En este sentido parece que existe un consenso entre los autores, puesto que la mayoría aboga por crear modelos de trazas conformes a metamodelos de trazabilidad genéricos para almacenar esta información. Este es el escenario esperado en el contexto de MDE, ya que uno de los principios de este paradigma es el uso de modelos a lo largo de todo el ciclo de vida del sistema y considerarlos como artefactos de máxima importancia. Sin embargo la coexistencia de multitud de metamodelos de trazabilidad resulta en un problema de interoperabilidad entre los modelos de trazas generados con distintas herramientas. Aunque es cierto que la mayoría de estos metamodelos tienen elementos comunes o similares (elementos de los modelos y relaciones entre ellos), por lo que sería factible definir transformaciones entre dichos metamodelos para mejorar este problema de interoperabilidad.

La visualización de las trazas ha obtenido la peor puntuación de los criterios evaluados (3.5 sobre 10) y no porque sea una cuestión menor. De hecho, el 60% de las propuestas analizadas realiza alguna aproximación en este sentido, pero la mayoría de ellas propone emplean visualizaciones genéricas para representar las trazas obtenidas (calificadas con valor P en la Tabla 3-2), tan solo la propuesta Oldevik *et al.* sugiere una técnica específicas para representar este tipo de información, aunque quizás esta aproximación no sea la más acertada para la representación de trazas entre elementos ya que se trata de un editor gráfico de tipo diagramador. Entre las propuestas que sugieren representaciones genéricas cabe destacar que varias coinciden a la hora de proponer representaciones basadas en editores que permiten visualizar al mismo tiempo varios modelos terminales y uno modelo que muestra las relaciones entre los elementos de los modelos

anteriores, ejemplos de estas representaciones se encuentran en las herramientas AMW o ModeLink.

Por último, otro de los criterios que han obtenido peor puntuación (4 puntos sobre 10 posibles), el análisis de la información de trazabilidad. De las diez propuestas analizadas solo cuatro de ellas presentan técnicas para llevar a cabo el análisis de las trazas generadas, el resto no considera esta tarea. Por tanto, se deduce que es un tema que actualmente se encuentra bastante inmaduro. En cuanto a las propuestas a las que se les ha asignado un "Sí" en este criterio, los trabajos de Sánchez *et al.* y los trabajos de Valderas y Pelechano coinciden en proponer la generación de informes de trazabilidad en formato HTML. En cuanto a las otras dos propuestas: la de Oldevik *et al.* presenta el desarrollo de un prototipo que de soporte un conjunto de análisis bien identificados como el análisis del impacto del cambio, el análisis de cobertura y el análisis de huérfanos; y la propuesta presentada por Drivalos *et al.* propone una guía para realizar una clasificación de los tipos de enlaces de traza que se generan a lo largo de un proceso MDE.

A la vista de los datos obtenidos, se puede afirmar que el almacenamiento de las trazas es un tema ampliamente afrontado en el cual la mayor parte de los autores coinciden en el uso de modelos de trazas. En cambio, la visualización y análisis de dichas trazas resultan tareas actualmente poco afrontadas y por tanto, inmaduras en el contexto de este estado del arte.

- **CI-3.** ¿Existen herramientas o marcos de trabajo que den soporte tecnológico para la gestión de la trazabilidad en el contexto del desarrollo de transformaciones de modelos?

Como ya se ha comentado en varias ocasiones a lo largo de esta tesis doctoral, desde un punto de vista teórico, MDE ofrece un nuevo escenario que fomenta llevar a cabo nuevas mejoras en la gestión de la trazabilidad en el desarrollo de software. Dando respuesta a esta pregunta se comprueba si el tradicional *gap* que existe entre las ideas y el desarrollo de las mismas [130, 143, 163] se pone de manifiesto una vez más. *A priori*, con los datos que proporciona la Tabla 3-2 parece cumplirse de nuevo ya que el CE-7, que evalúa si las propuestas metodológicas son llevadas a la práctica por medio de una herramienta de soporte, ha obtenido un 45% de la puntuación máxima que podría obtener.

Sin embargo, esta baja puntuación no se debe a que sea un tema poco afrontado por los autores sino porque en la mayoría de los casos la implementación que se proporciona es parcial y no constituye una herramienta completa. En realidad, 7 de las 10 propuestas analizadas realiza alguna aproximación en cuanto a la implementación de la propuesta, pero solo 2 de ellas

(Boronat *et al.* y Jouault *et al.*) proporcionan una herramienta que da soporte completo a la propuesta metodológica. Esto puede deberse a que algunos autores se centran en implementar aquellas partes de su propuesta que resultan más novedosas y para poner en práctica el resto de la propuesta emplean herramientas ya existentes. Por tanto, se puede afirmar que actualmente existen conjuntos de herramientas que combinadas adecuadamente ofrecen soporte para la gestión de la trazabilidad en el ámbito del desarrollo de transformaciones de modelos, pero que existen pocas implementaciones que, por si solas, den soporte a una propuesta completa.

En cuanto a cómo se han desarrollado estas implementaciones, cabe destacar que la mayoría de ellas han sido construidas sobre el entorno Eclipse y en particular, mediante el uso de las tecnologías proporcionadas en el contexto del proyecto EMP.

- **CI-4**. ¿Cuáles son las limitaciones encontradas en las propuestas para la generación de trazabilidad a partir del desarrollo de transformaciones de modelos?

Dando respuesta a las cuestiones anteriores, se han identificado algunas limitaciones o puntos de mejora en el estado del arte actual en gestión de la trazabilidad a partir del desarrollo de transformaciones de modelos.

En el apartado de discusión de CI-1, se han identificado dos limitaciones o puntos de mejora: automatizar la identificación de las relaciones de trazabilidad a nivel de metamodelo es una tarea aún inmadura y en general, la generación de las trazas a partir de la ejecución de transformaciones de modelos es muy dependiente del motor que ejecute dicha transformación. En el contexto de MDE, las transformaciones de modelos resultan una pieza clave, por ello en los últimos años ha sido un campo de investigación emergente y en continua evolución del que han surgido multitud de herramientas y lenguajes para desarrollar transformaciones [46, 182]. Por este motivo sería deseable que estas transformaciones propuestas para la generación de información de trazabilidad fueran independientes de cualquier motor de transformación existente o incluso, de futuros motores o lenguajes. De esta forma la evolución de la tecnología no sería una limitación a la hora de adaptar las propuestas a nuevos entornos o necesidades. Además, dado que las transformaciones son un artefacto más del sistema y que cada vez resultan más complejas (más todavía si se emplean para generar además las trazas del sistema), sería deseable aplicar las técnicas de MDE a su construcción, es decir seguir un proceso dirigido por modelos para el desarrollo de transformaciones de modelos. Esta aproximación permitirá obtener especificaciones de

transformaciones a alto nivel, es decir, independiente del lenguaje que finalmente la implemente y al mismo tiempo, si el nivel de abstracción es lo suficientemente alto será posible identificar las relaciones de trazabilidad entre los elementos implicados en la transformación.

En la CI-2 se discute acerca de cómo se gestiona la información de trazabilidad una vez que ha sido generada. En este sentido, se ha analizado cómo se lleva a cabo el almacenamiento, visualización y análisis de las trazas. En cuanto al almacenamiento de las trazas se ha detectado que la mayor parte de los autores coinciden en señalar el empleo de modelos de trazas como la forma de almacenamiento más adecuada a este contexto. Como consecuencia, existe un gran número de metamodelos de trazabilidad, lo que implica un problema de interoperabilidad entre los modelos conformes a dichos metamodelos. Aunque como se ha comentado anteriormente, estos metamodelos contienen bastantes similitudes por lo que sería posible definir transformaciones que minimicen este inconveniente. El uso de técnicas de modelado también se encuentra en parte relacionado con la realidad de que la mayoría de las propuestas no proporcionen métodos específicos para visualizar o representar los modelos de trazas. Actualmente, algunas de las herramientas de modelado más empleadas en este contexto, como por ejemplo EMF, proporcionan editores y visualizadores genéricos que permiten visualizar cualquier modelo conforme a los metamodelos definidos. Desafortunadamente, este tipo de visualizadores genéricos no se ajustan fielmente a la naturaleza de los modelos de trazas, por lo que no ofrecen la mejor visualización posible, por ejemplo este tipo de representaciones se limitan a mostrar los objetos del modelo de trazas y no permiten navegar hasta los elementos trazados en su correspondiente modelo. En este sentido algunas de las propuestas coinciden en la necesidad de disponer de editores que permitan visualizar al mismo tiempo el modelo de trazas y los modelos trazados y permitir la interacción entre ellos, como los que proporcionan las herramientas AMW y ModeLink.

En la discusión de la CI-2 también se identificado una limitación acerca del análisis de las trazas generadas. En el contexto de la gestión de la trazabilidad existen evidencias acerca de la importancia de analizar la información de trazabilidad para llevar a cabo otras actividades software [9, 142, 148, 177, 193]. Sin embargo, entre las propuestas analizadas solo un 40% de ellas presentan técnicas o métodos para facilitar el análisis de la información de trazabilidad.

Por último respondiendo a la CI-3, correspondiente a la existencia de implementaciones que den soporte a las propuestas metodológicas y cómo se construyen, se ha detectado que a pesar de existir entornos de trabajo como

Eclipse y el proyecto EMP que facilitan el desarrollo de herramientas para dar soporte a propuestas MDE, un gran número de autores han implementado solo parte de su propuesta. Por ello, actualmente existen pocas implementaciones que den soporte completo a una propuesta para la generación de información de trazabilidad. Esta limitación puede parecer *a priori* de poca importancia, sin embargo si para poner en práctica la generación de trazas es necesario recurrir a un conjunto de implementaciones parciales, es muy probable que en algún momento aparezcan incompatibilidades entre dichas implementaciones o que alguna de ellas sea abandonada por sus desarrolladores. Por ello, sería recomendable disponer de herramientas completas que den soporte a toda la propuesta metodológica.

Propuesta Metodológica: MeTAGeM-Trace

En capítulo anterior se ha realizado un proceso de identificación del cuerpo del conocimiento y una de las principales conclusiones extraídas es la ausencia de propuestas que combinen el desarrollo dirigido por modelos de transformaciones de modelos y la generación de las trazas del sistema a partir de dichas transformaciones. Así, para afrontar esta limitación del estado del arte y tal como se describe en el Capítulo 1, esta tesis doctoral tiene por objetivo proporcionar un entorno para el desarrollo de transformaciones de modelos dirigido por modelos y la generación (semi-)automática de modelos de trazas. Para alcanzar este objetivo, se propone la metodología **MeTAGeM-Trace** que se compone de:

- Un proceso de desarrollo de transformaciones de modelos dirigido por modelos.
- Un conjunto de metamodelos que permitan describir transformaciones y relaciones de trazabilidad a distintos niveles de abstracción.
- Un conjunto de reglas de transformación que permitan obtener (semi-)automáticamente, a partir de un modelo que describe la transformación a un nivel de abstracción concreto, un modelo que la describa a un nivel de abstracción menor.
- Un conjunto de reglas de transformación que permitan obtener (semi-)automáticamente la serialización de los modelos que describen la transformación a bajo nivel.
- Un conjunto de reglas de validación para evitar que los errores en los modelos se propaguen a los niveles de abstracción inferiores.
- Un metamodelo de trazabilidad de carácter general que permita definir las trazas que se derivan de la ejecución de una transformación de modelos.

Como ya se ha mencionado en varias ocasiones en esta tesis, el nuevo escenario de desarrollo que surge con la llegada de MDE ofrece una serie de características que facilitan la mejora en la automatización de las actividades implicadas en un desarrollo de software. De acuerdo a los principios del desarrollo de software dirigido por modelos, el sistema es construido a partir de su definición en un modelo de alto nivel y la ejecución de transformaciones hasta que se obtiene un modelo ejecutable o el código que implementa el sistema [14, 73, 179, 182]. Generalmente, las reglas que componen dichas transformaciones contienen información del tipo “a partir del elemento *a*, generamos el elemento *b*”.

Por otro lado, la generación y mantenimiento de la información de la trazabilidad ha sido tradicionalmente considerada una tarea manual, costosa y propensa a errores [57, 61, 86, 132, 148, 221]. Por todo ello, sería posible aprovechar la información contenida en las transformaciones para obtener (semi-)automáticamente las trazas que relacionan los elementos del sistema. De esta forma se estaría reduciendo el esfuerzo que supone generar y mantener este tipo de información ya que estaría embebido en la construcción de las transformaciones, que es una tarea que se ha de realizar en cualquier caso. Sin embargo, de este modo se aumentaría la complejidad de las transformaciones a construir.

Como consecuencia del aumento de la complejidad en las transformaciones implicadas en el desarrollo del software, resulta recomendable aplicar a su construcción los mismos principios que se aplican al desarrollo del resto de artefactos implicados en el proceso de desarrollo, es decir los principios de MDE.

En el contexto del Grupo de Investigación Kybele, en el que se ha realizado esta tesis, en trabajos anteriores se presentó MeTAGeM [30, 93], un entorno de DDM de transformaciones de modelos. A partir de esta experiencia y la identificación del cuerpo del conocimiento presentado en el capítulo anterior, en esta tesis doctoral se propone incorporar un **meta-generador de trazabilidad embebido** en un entorno de desarrollo de transformaciones de modelos dirigido por modelos. Concretamente MeTAGeM-Trace, el entorno que se define en esta tesis doctoral, se centra en mejorar, ampliar y evolucionar MeTAGeM, de acuerdo al planteamiento presentado.

Así, uno de los primeros pasos metodológicos consiste en el análisis del entorno al que se desea dotar del meta-generador de trazabilidad. En este caso, el entorno MeTAGeM propone la construcción de transformaciones de modelos mediante el uso de modelos a diferentes niveles de abstracción, de acuerdo a la clasificación de niveles que propone MDA [153]. En el entorno MeTAGeM-Trace, dichos modelos describen, al nivel de abstracción correspondiente, las reglas de la transformación y además, las relaciones de trazabilidad entre los elementos implicados en dichas reglas.

Por todo ello, la propuesta metodológica MeTAGeM-Trace permite modelar a nivel PIM, es decir sin tener en cuenta la plataforma final en que se implementará la transformación, las relaciones de transformación entre los elementos que forman parte de uno o varios modelos de entrada y los elementos de uno o varios modelos de salida. A partir de este modelo independiente de plataforma se obtiene otro dependiente de plataforma (PSM) que describe las reglas de la transformación de acuerdo al paradigma de transformación

seleccionado (imperativo, declarativo, híbrido, basado en grafos, etc.) y contiene implícitamente las relaciones de trazabilidad obtenidas a partir de las reglas de la transformación. De forma que cuando se obtenga la transformación ejecutable y se lleve a cabo su ejecución se generen, además los elementos de salida, las trazas derivadas de las relaciones de trazabilidad. El siguiente paso consiste en la generación de un modelo PDM que describa la transformación conforme al metamodelo del lenguaje que implementará la transformación y que contenga las construcciones necesarias para la generación de las trazas. Este escenario es mostrado por la Figura 4-1.

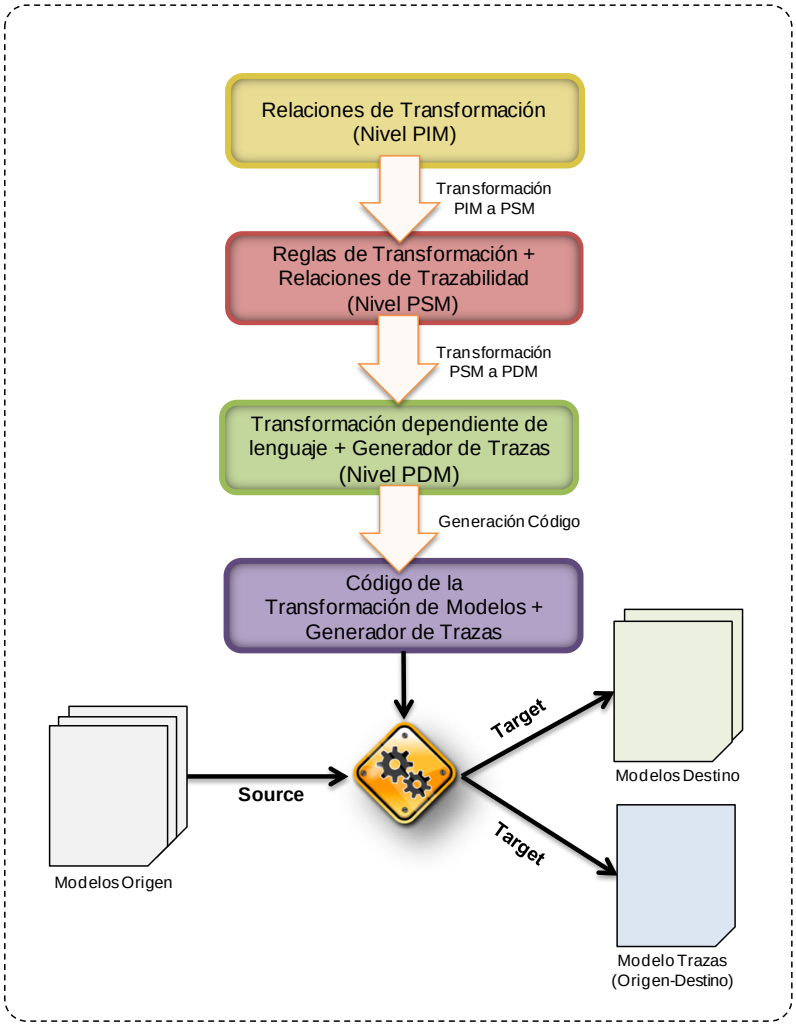

Figura 4-1. Escenario de Generación de Modelos de Trazas a partir del DDM de Transformaciones de Modelos

Para poder gestionar los modelos anteriores, MeTAGeM-Trace, como herencia directa de la propuesta MeTAGeM, define un conjunto de metamodelos que permiten modelar las transformaciones y las relaciones de trazabilidad a diferentes niveles de abstracción. Dichos metamodelos se han definido conforme a MOF [154], con el objetivo de proporcionar interoperabilidad con otras herramientas MDE basadas en MOF. Así, cualquiera de los modelos elaborados o generados con MeTAGeM-Trace podrá ser utilizado y/o gestionado desde cualquier otra herramienta (basada en el uso de modelos), tan solo desarrollando una transformación que permita conectar los metamodelos de ambas herramientas. Además, el hecho de especificar las transformaciones (e implícitamente las relaciones de trazabilidad) en términos de modelo proporciona en una serie de ventajas adicionales: en cuanto las transformaciones y las relaciones de trazabilidad sean un modelo más, podremos generarlas, transformarlas, validarlas, compararlas, refactorizarlas, etc. utilizando técnicas propias de la Ingeniería Dirigida por Modelos [21].

Esta aproximación disminuye los esfuerzos a realizar por los desarrolladores de las transformaciones y los encargados de crear la información de trazabilidad del sistema. Por un lado, se automatiza la generación y mantenimiento de las trazas del sistema al incluirse en la ejecución de las transformaciones y por otro lado, dado que las transformaciones se definen a alto nivel, omitiendo los detalles tecnológicos, no es necesario que los desarrolladores conozcan a la perfección los lenguajes y motores de transformación.

En las siguientes secciones se explicarán cada uno de los componentes de la metodología que se presenta en esta tesis.

4.1 Proceso de Desarrollo de Transformaciones que incorporan Generación de Trazas

Como se ha comentado arriba, el proceso se realiza de acuerdo a cuatro niveles de abstracción: **PIM, PSM, PDM** y **código**. En la Figura 4-2 se muestra el proceso mediante un diagrama de actividad SPEM [152]. Este proceso es una adaptación del presentado en [30].

El primer paso del proceso consiste en modelar la transformaciones de modelos a alto nivel (modelos a nivel PIM), omitiendo las cuestiones técnicas de la implementación final. Para ello, se indican las relaciones existentes entre los diferentes elementos de los metamodelos que intervienen en la transformación.

A partir del modelo definido a nivel PIM se genera, por medio de una transformación de modelos, el modelo a nivel PSM que contiene la definición de

las reglas de la transformación y de las relaciones de trazabilidad existentes entre los elementos implicados en dichas reglas, teniendo en cuenta detalles tecnológicos específicos de una aproximación de transformaciones de modelos concreta (declarativa, imperativa, híbrida, basado en grafos, etc.). Este modelo específico se podrá refinar manualmente, pudiéndose agregar, modificar o eliminar los elementos necesarios para adaptarse a las necesidades del desarrollo.

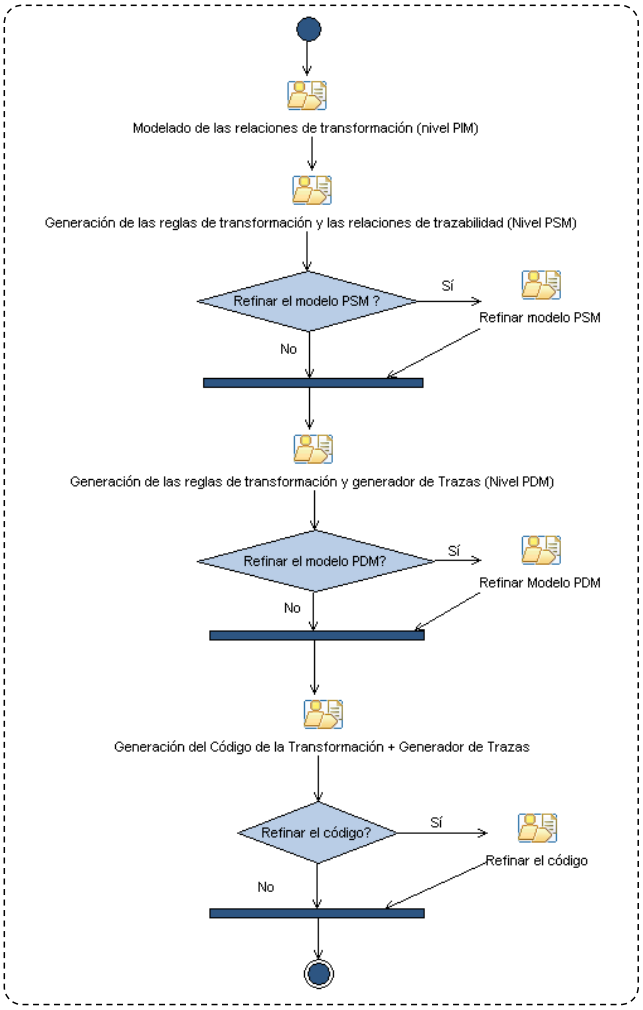

Figura 4-2. Diagrama SPEM del Proceso de MeTAGeM-Trace

Siguiendo la misma idea del paso anterior, a partir del modelo obtenido a nivel PSM y aplicando nuevamente transformaciones, se obtiene el modelo que contiene las reglas de transformación de forma dependiente de plataforma y las construcciones necesarias para generar las trazas (nivel PDM). Este modelo,

conforme al metamodelo que describe el lenguaje de transformación que implementará la transformación, también podrá ser refinado.

Finalmente, la última actividad consiste en la generación del código de la transformación que serializa el modelo generado a nivel PDM. Esta generación se hará de forma (semi-)automática por medio de transformaciones de modelos a texto. El código generado también puede ser refinado y ejecutado usando el IDE proporcionado por el propio lenguaje de transformación de modelos.

El resultado final del proceso es la obtención de un código de la transformación en el lenguaje que se haya seleccionado. Dicho código, al ser ejecutado generará, al mismo tiempo, los modelos de salida correspondientes a la transformación definida y un modelo que contendrá las trazas existentes entre los elementos de la entrada y la salida de la transformación.

4.2 Especificación de Metamodelos

Para poder realizar el modelado de las transformaciones de modelos en diferentes niveles de abstracción, MeTAGeM propone la definición de un conjunto de metamodelos que se corresponden con cada uno de los niveles de abstracción propuestos [30]. En esta sección se presenta la jerarquía de modelado así como cada uno de dichos metamodelos, adaptados a las necesidades del nuevo entorno MeTAGeM-Trace, que se propone en esta tesis doctoral. La Figura 4-3 muestra la jerarquía de modelado que propone MeTAGeM-Trace, como resultado de esta adaptación.

En primer lugar, a **nivel PIM**, se propone el empleo de MOF como lenguaje de metametamodelado para la definición del metamodelo MeTAGeM que permite modelar las relaciones de transformación a alto nivel, de forma independiente a la plataforma que ejecutará la transformación. Concretamente, los modelos de MeTAGeM contendrán las relaciones existentes entre los elementos de uno o varios metamodelos origen y uno o varios metamodelos destino.

A **nivel PSM**, se define el conjunto de metamodelos que se corresponden con cada una de las diferentes aproximaciones de transformación de modelos. Dichos metamodelos deberán contener los constructores necesarios para permitir el modelado de las relaciones de trazabilidad. En concreto, en esta tesis doctoral se define el metamodelo correspondiente a la aproximación híbrida.

En el **nivel PDM** se encuentran los modelos que describen la transformación de forma dependiente de la plataforma que la implementará. Por tanto, en este nivel es necesario incluir o definir (en caso de que no existan) los metamodelos de los lenguajes de transformación en los que se desee obtener el

código de la transformación. Dado que los metamodelos incluidos en este nivel se corresponden con la definición de los lenguajes de transformación y por ello, no es posible definir explícitamente constructores que generen las trazas, esta información deberá ser definida en términos de reglas de transformación o elementos de dichas reglas (según corresponda), conforme a las estructuras definidas por el lenguaje de transformación que se quiera emplear. En el contexto de esta tesis doctoral, dado que a nivel PSM se ha propuesto seguir la aproximación híbrida, a nivel PDM se especifican los metamodelos de dos de los lenguajes de transformación híbridos más empleados en la actualidad: ATL [101] y ETL [113].

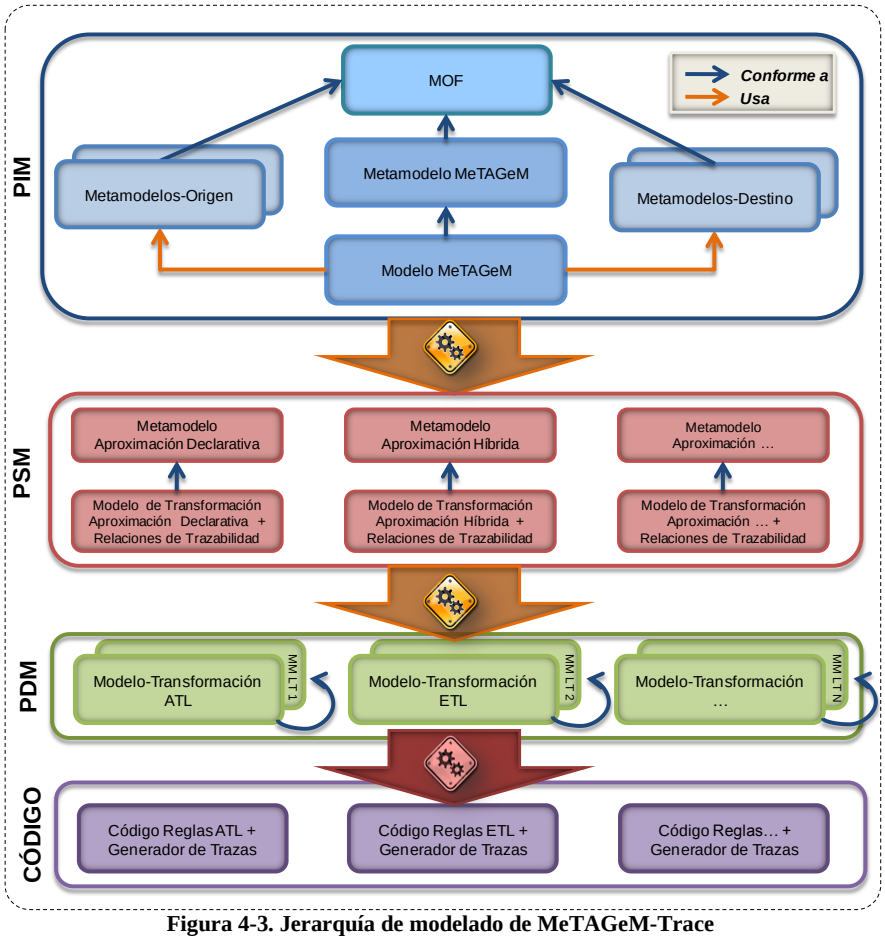

Figura 4-3. Jerarquía de modelado de MeTAGeM-Trace

Finalmente, en el nivel más bajo de la jerarquía, se encuentra el **código** de la transformación que serializa los modelos PDM. Dicho código tendrá las estructuras necesarias para generar uno o varios modelos de salida conformes a los

metamodelos destino y las estructuras necesarias para generar un modelo que contenga las trazas derivadas de la ejecución de la transformación. Dado que a nivel PDM, MeTAGeM-Trace incluye los metamodelos de los lenguajes ATL y ETL, el código de la transformación será creado en términos de estos lenguajes.

Una vez mostrada la jerarquía de modelado y las relaciones existentes entre los metamodelos y modelos que intervienen en el proceso presentado, en las siguientes sub-secciones se procede a describir cada uno de los metamodelos mencionados. En primer lugar se presenta el metamodelo de trazabilidad que define cómo deben ser los modelos de trazas creados por el generador de trazas embebido en las transformaciones.

4.2.1 Metamodelo de Trazabilidad

Dado que el objetivo del meta-generador de trazabilidad tiene por objetivo incluir en las reglas de la transformación la información necesaria para construir modelos de trazas, es necesario definir un **metamodelo de trazabilidad** que defina la sintaxis abstracta de dichos modelos.

La mayor parte de las propuestas analizadas para la gestión de la trazabilidad coinciden en la necesidad del uso de **metamodelos genéricos** que permitan modelar las trazas en cualquier dominio de aplicación. Esta necesidad también se pone de manifiesto en MeTAGeM-Trace, ya que no se conoce cómo serán los metamodelos de origen y destino de la transformación a desarrollar y en consecuencia, es imposible conocer cómo serán las trazas derivadas de la ejecución de dicha transformación. Por esta razón, el metamodelo de trazabilidad que se presenta en esta tesis doctoral es genérico para cualquier escenario de trazabilidad. Para especificar dicho metamodelo, durante el proceso de estudio del estado del arte presentado en el Capítulo 3, se han analizado diversos metamodelos de trazabilidad [34, 97, 112, 128, 138, 175, 217]. La mayor parte de ellos coinciden en ciertos **elementos comunes**: elementos de los modelos y las relaciones entre ellos. Así, a partir de esta información de base y mediante un proceso iterativo, se ha definido el metamodelo de trazabilidad que se muestra la Figura 4-4.

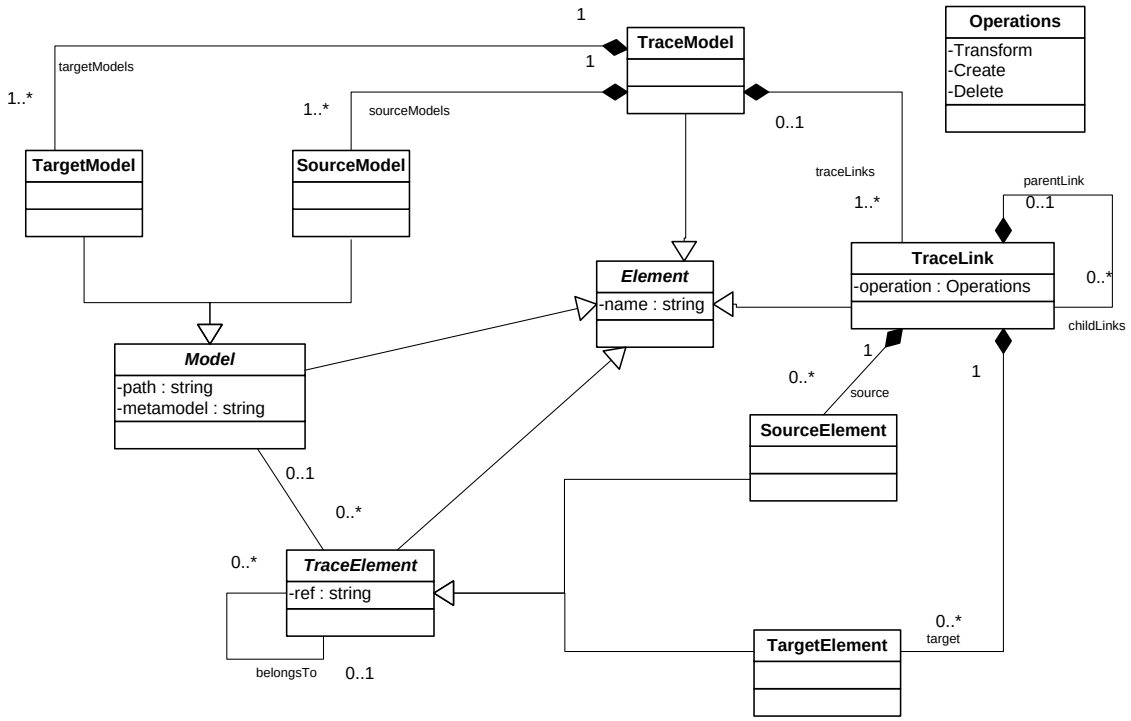

Figura 4-4. Metamodelo de Trazabilidad (MeTAGeM-Trace)

El elemento principal del metamodelo de trazabilidad es la metaclase *TraceModel* que representa al modelo de trazas y debe contener uno o varios modelos origen (*SourceModel*), uno o varios modelos destino (*TargetModel*) y un conjunto de trazas (*TraceLink*). Dichas trazas pueden contener elementos origen, elementos destino y otras trazas de menor entidad (para representar las trazas entre los atributos de los elementos). Además por cada una de las trazas se almacena si se trata de una operación de transformación, creación o borrado (*Operations*). Del mismo modo que se registra esta información, podrían almacenarse otros datos como la fecha de creación, el usuario que ha ejecutado la transformación, la ubicación, etc.

4.2.2 Metamodelo MeTAGeM (nivel PIM)

Este metamodelo ya fue propuesto en [30] con el objetivo de disponer de un **metamodelo independiente del lenguaje de transformación** que implemente la transformación y suficientemente genérico como para poder emplearse en cualquier tipo de transformación de modelos y en cualquier dominio de aplicación.

Para alcanzar esta genericidad e independencia, este metamodelo permite la definición de un conjunto de elementos origen, pertenecientes a uno o más modelos origen; un conjunto de elementos destino de uno o más modelos destino; y el tipo de relaciones existentes entre los elementos origen y destino. Dichos tipos de relaciones se establecen de acuerdo a varias características: el rol de la relación en la transformación (principal, secundaria o abstracta), su cardinalidad (uno-a-uno, uno-a-cero, uno-a-muchos, cero-a-uno, muchos-a-uno o muchos-a-muchos) y su comportamiento (copia, concatenación u otros).

Tras analizar en profundidad el metamodelo independiente de plataforma propuesto en [30], se ha determinado que, debido a la genericidad en la semántica de las relaciones que pueden ser definidas y a su alto nivel de abstracción: la incorporación del meta-generador de trazabilidad en el entorno de DDM de transformaciones no requiere llevar a cabo grandes cambios en dicho metamodelo.

Así, el metamodelo a nivel PIM de MeTAGeM-Trace solo incluye cambios de nomenclatura (por ejemplo, la metaclase *NtoN* ha pasado a denominarse *ManyToMany*), redefinición de los tipos enumerados y la incorporación nuevas metaclases abstractas (*TransformationElement* y *RelationElement*) con el objetivo de mejorar la calidad de dicho metamodelo.

En la Figura 4-5 se muestra el metamodelo MeTAGeM tras haber realizado dichos cambios.

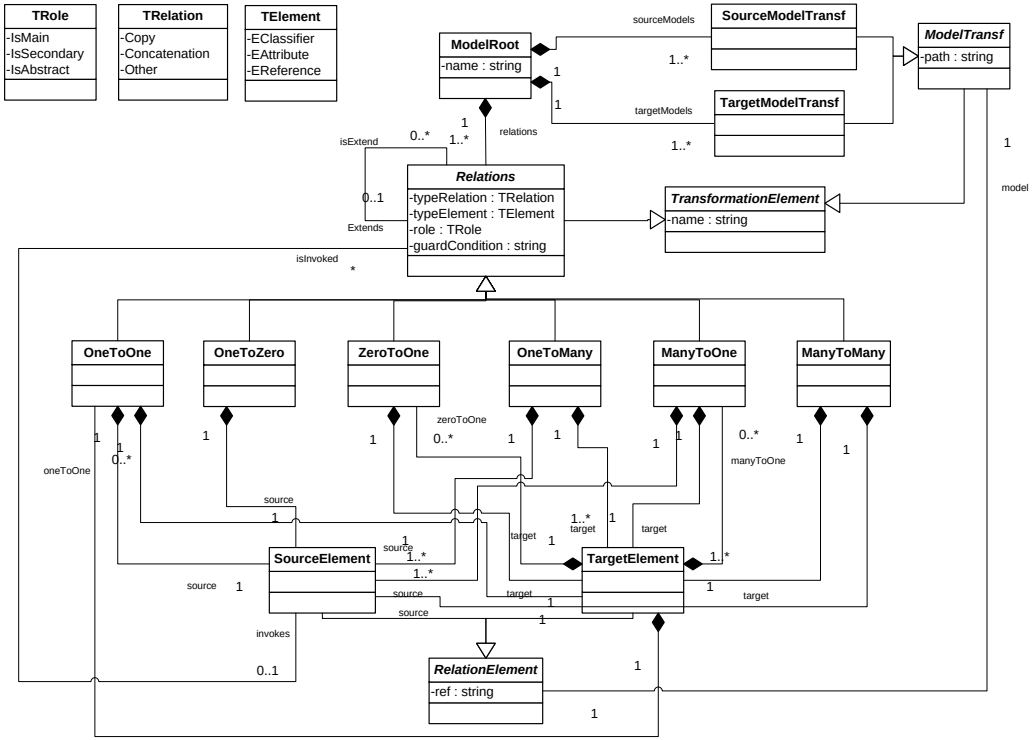

Figura 4-5. Metamodelo MeTAGeM (nivel PIM)

4.2.3 Metamodelo Específico de Plataforma (nivel PSM)

Como ya se ha comentado en varias ocasiones, el principal objetivo de esta tesis doctoral es incluir, en un entorno de DDM de transformaciones de modelos, los mecanismos necesarios para la generación y gestión de las trazas derivadas de la ejecución de la transformación que se desarrolla. El entorno de DDM de transformaciones de modelos que se ha empleado como punto de partida para llevar a cabo esta idea es el entorno MeTAGeM. En tanto en cuanto MeTAGeM propone a nivel PSM el modelado de transformaciones que siguen una **aproximación híbrida**, la propuesta MeTAGeM-Trace también sigue dicho paradigma de transformaciones a nivel PSM. Pero de igual forma puede aplicarse a otras aproximaciones.

El metamodelo que se propone a este nivel en [30], fue definido a partir un análisis de los lenguajes de transformación de modelos más representativos que siguen la aproximación híbrida, con el objetivo de determinar los elementos comunes a dichos lenguajes. Así, los principales elementos de este metamodelo son la definición de los metamodelos de origen y destino de la transformación, las reglas de la transformación y las operaciones (o funciones) que se definen en el contexto de la transformación.

Dado que en este caso los elementos que relacionan los elementos de los metamodelos origen y los metamodelos destino (*rule* y *binding*) tienen una semántica propia del mundo de las transformaciones, es recomendable e incluso necesario definir nuevos elementos que describan las relaciones de trazabilidad entre los elementos de los metamodelos. Por ello, se propone un nuevo metamodelo para definir las transformaciones que siguen la aproximación híbrida, incluyendo los constructores apropiados para el modelado de las relaciones de trazabilidad. Este nuevo metamodelo se muestra en la Figura 4-6.

El metamodelo para la descripción de las transformaciones de aproximación hibrida permite el modelado de los aspectos más comunes de los lenguajes que siguen esta aproximación: modulo (elemento raíz), reglas, elementos origen y destino de las reglas, asignaciones tipo *binding*, operaciones y metamodelos origen y destino.

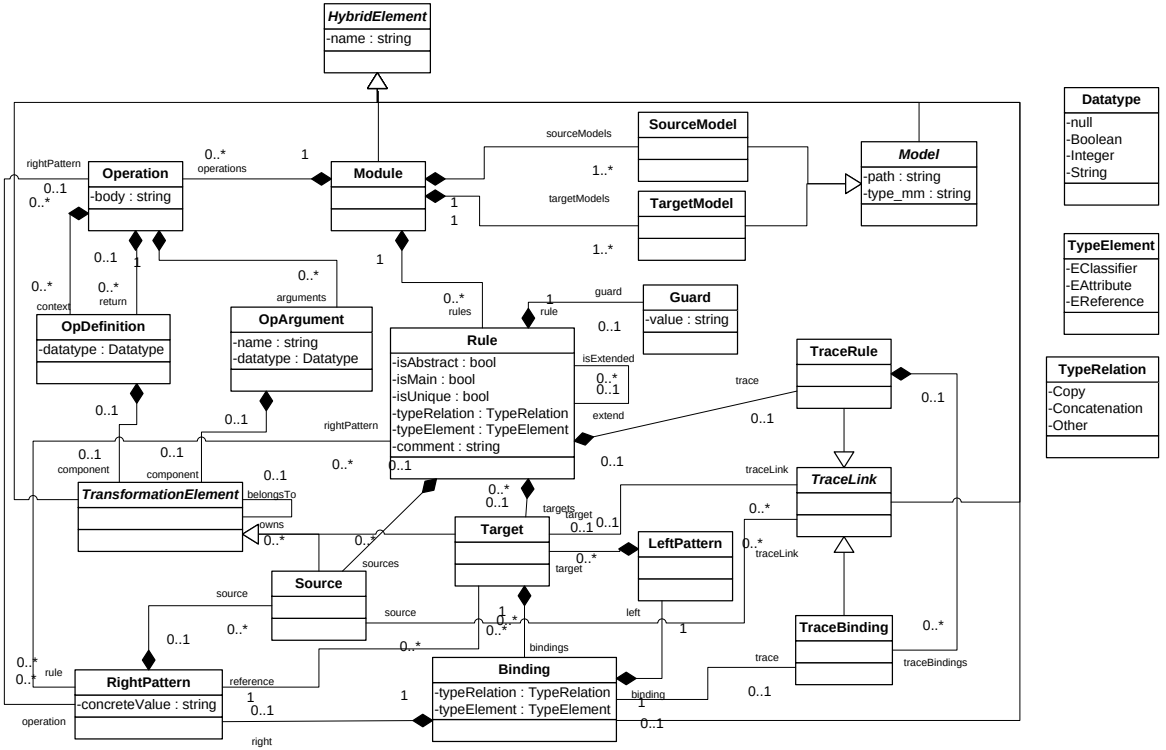

Figura 4-6. Metamodelo de Transformaciones de aproximación Híbrida (nivel PSM)

Además para el modelado de las relaciones de trazabilidad, se incluyen las metaclases ***TraceLink*** (abstracta), ***TraceRule*** y ***TraceBinding***. La metaclase *TraceRule* permite el modelado de las relaciones de trazabilidad que se derivan directamente de las reglas de la transformación, estableciendo automáticamente el vínculo entre los elementos de la regla. Por otro lado, con el objetivo de registrar no solo las relaciones de trazabilidad entre los elementos principales de la regla, sino también entre los elementos implicados en los *bindings* (generalmente, atributos y referencias), se define la metaclase *TraceBinding.*

Cada una de estas relaciones de trazabilidad (*TraceRule* y *TraceBinding*), a través de las asociaciones de la metaclase abstracta *TraceLink*, puede tener entre 0 y N elementos origen y de 0 a N elementos destino. Estas cardinalidades se derivan de las relaciones que se pueden establecer a nivel PIM (uno-a-uno, uno-a-cero, uno-a-muchos, cero-a-uno, muchos-a-uno o muchos-a-muchos).

4.2.4 Metamodelos Dependientes de Plataforma (nivel PDM)

En el nivel anterior, se propone el uso de un metamodelo correspondiente a la aproximación híbrida para realizar el modelado de las transformaciones, por lo que a nivel PDM se deben seleccionar lenguajes de transformación que sigan dicha aproximación.

En MeTAGeM, el entorno de desarrollo de transformaciones empleado como punto de partida de MeTAGeM-Trace, propone el uso de los lenguajes de transformación ATL [101] y RubyTL [174]. Dichas propuestas se basan, principalmente, en las siguientes afirmaciones: ATL es uno de los lenguajes de transformación más maduros y es considerado el estándar *de-facto* en el desarrollo de transformaciones de modelos. Además, para el desarrollo de ATL se han aplicado los principios de MDE y por ello, proporciona la implementación de su propio metamodelo; RubyTL es un lenguaje de transformaciones embebido en el lenguaje de programación Ruby y en el momento de llevar a cabo la identificación del estado del arte que se presenta en [30], se identifica como el segundo lenguaje de transformación con mayor documentación.

En esta tesis doctoral, en el momento que se aborda la elección de los lenguajes de transformación que se emplean a nivel PDM (y por extensión, a nivel de código):

- ATL sigue siendo considerado el estándar *de-facto* para el desarrollo de este tipo de artefactos y quizás el lenguaje de transformaciones más empleado entre la comunidad de desarrolladores. Otros factores importantes a la hora de la elección de ATL son: la gran cantidad de documentación disponible, proporciona una implementación de su propio metamodelo y existen

implementaciones para extraer el código ATL a partir de los modelos ATL [8, 15, 98, 99, 101].

- RubyTL ha perdido importancia en el campo de los lenguajes de transformaciones de modelos, a favor de otros lenguajes como ETL (híbrido), mediniQVT (declarativo), QVTO (imperativo) o VIATRA (basado en grafos).

  Así mismo, la experiencia previa en el trabajo con RubyTL ha mostrado las dificultades asociadas al uso de un lenguaje de transformación embebido en un lenguaje no orientado a trabajar con modelos (Ruby) [30, 93].

- ETL es un lenguaje de transformación de modelos basado en EOL (*Epsilon Object Language*, [109]), un lenguaje imperativo orientado a modelos. ETL pertenece a la familia de lenguajes de Epsilon (*Extensible Platform for Specification of Integrated Languages for mOdel Management*,[114, 115]), que forma parte del proyecto Eclipse GMT (*Generative Modeling Technologies*). Como parte de dicho proyecto, Epsilon dispone de su propio sitio web alojado en el sitio de Eclipse donde se proporciona una amplia documentación y ejemplos de uso y se ofrece un foro de discusión, ampliamente usado por la comunidad de desarrolladores. Además, en [113] se presenta una especificación del metamodelo de ETL.

Por todo lo anterior, en el nivel PDM de MeTAGeM-Trace se encuentran los lenguajes de transformación **ATL** y **ETL**. A continuación se muestran los metamodelos de dichos lenguajes:

4.2.4.1 ATL

En la Figura 4-7, se presenta el metamodelo simplificado de ATL. De hecho, la implementación de este metamodelo requiere componentes del metamodelo de OCL. Como se puede comprobar en la Figura 4-7, se trata de un metamodelo bastante complejo, sin embargo, es posible identificar claramente los aspectos que se han definido, en el nivel PSM, como comunes a los lenguajes que siguen una aproximación híbrida.

El elemento raíz es la metaclase *Module* que permite contener *Helpers* y *Rules*. Un *Helper* es una estructura que define ATL y se corresponde con las operaciones que define el desarrollador en el ámbito del módulo de la transformación. En cuanto a las reglas, ATL define tres tipos de reglas que se corresponden con los diferentes modos de programación soportados: *CalledRule* (regla imperativa), *MatchedRule* (regla declarativa) y *LazyMatchedRule*, que es una especialización de la regla de tipo *MatchedRule*.

Las funcionalidades de cada una de las reglas que define ATL son las siguientes:

- *Called Rule*: permite introducir constructores imperativos en las reglas ATL. Este tipo de reglas deben ser invocadas de forma explícita para ser ejecutadas y pueden recibir parámetros de entrada. Deben ser invocadas desde la sección de código imperativo de una regla tipo *MatchedRule* u otra regla tipo *CalledRule*. Además, tiene dos propiedades, excluyentes entre sí, de tipo booleanas que permiten identificar el momento de ejecución de la regla: al inicio de la transformación (*isEntrypoint*) o al finalizar la transformación (*isEndpoint*).
- *Matched Rule*: son las reglas principales de una transformación ATL, ya que permite especificar: 1) qué elemento o elementos del modelo destino se debe generar a partir de uno o varios elementos del modelo origen, y 2) la manera en que se debe inicializar la generación de los elementos del modelo destino. En el momento de la ejecución, el motor de ATL evalúa cada uno de los elementos del modelo origen y verifica si en el conjunto de reglas existe alguna cuyo patrón de origen se corresponda con los elementos de modelo origen; si hubiese alguna coincidencia se generan los elementos correspondientes en el modelo destino. Este tipo de reglas pueden ser abstractas (atributo *isAbstract*) y en consecuencia, extendidas por otras reglas.

LazyMatched Rule: es una especialización del tipo de regla *MatchedRule*. Sirve para definir una regla que se debe ejecutar más de una vez sobre un mismo elemento de origen. Este tipo de regla debe ser invocada de forma explícita por otra regla para ser ejecutada. La propiedad *isUnique* sirve para restringir su ejecución; si tiene el valor verdadero, se ejecutará una sola vez y en las posteriores llamadas devolverá la referencia al elemento generado en la primera ejecución.

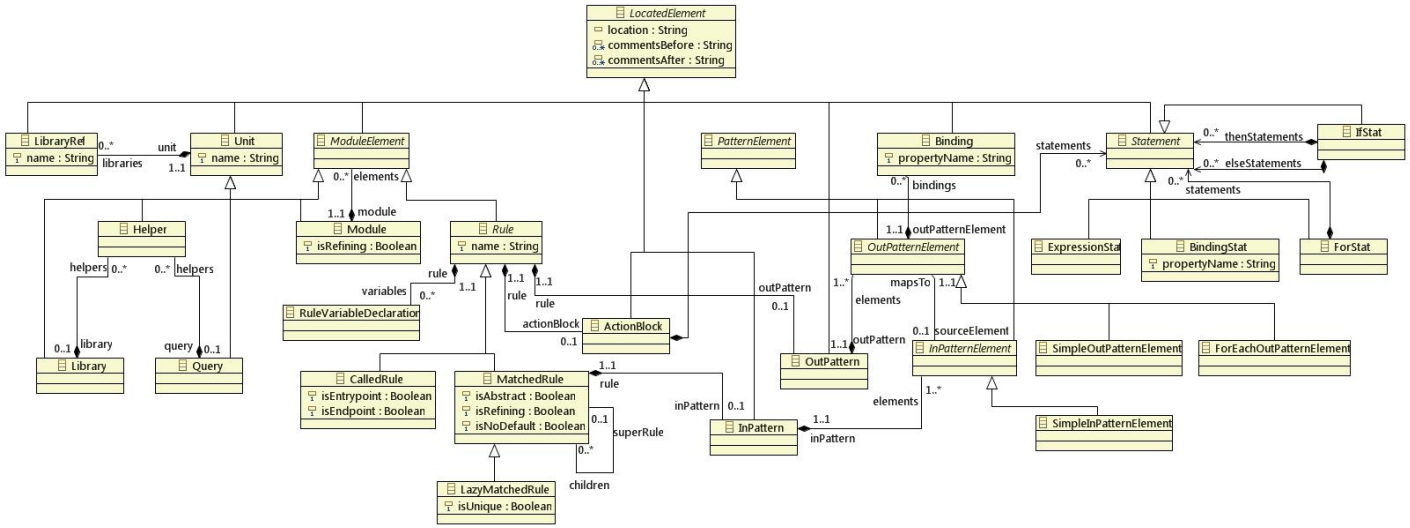

Figura 4-7. Metamodelo de ATL

4.2.4.2 ETL

Como se ha comentado anteriormente, ETL (*Epsilon Transformation Language*, [113]) es un lenguaje híbrido de definición de transformaciones que pertenece a la familia de lenguajes Epsilon, donde el lenguaje principal es EOL. La sintaxis abstracta de ETL define componentes propios de un lenguaje de transformaciones de modelos como el módulo o las reglas de transformación, sin embargo para ofrecer otras funcionalidades, como las operaciones o la declaración de los elementos, requiere del uso del lenguaje EOL.

A diferencia de ATL, ETL no proporciona una especificación completa de su metamodelo. Tan solo, en [113, 114] se puede encontrar una especificación de la sintaxis abstracta del lenguaje. Por este motivo, para realizar el modelado de las transformaciones en términos del lenguaje ETL, ha sido necesario realizar la especificación e implementación del mismo. Para llevar a cabo esta tarea se ha analizado en profundidad la sintaxis abstracta del lenguaje así como ciertos aspectos del lenguaje EOL y los ejemplos disponibles en el sitio web de la herramienta Epsilon. El resultado de este proceso es el metamodelo de ETL que muestra la Figura 4-8.

Al igual que en los metamodelos anteriores, la metaclase raíz es el módulo de la transformación (*EtlModule*). Este módulo puede contener bloques de código EOL que definan las *pre-condiciones* y *post-condiciones* de la transformación (*EolBlock*), las reglas de la transformación (*TransformationRule*) y operaciones que se pueden invocar a lo largo de la ejecución de la transformación (*Operation*).

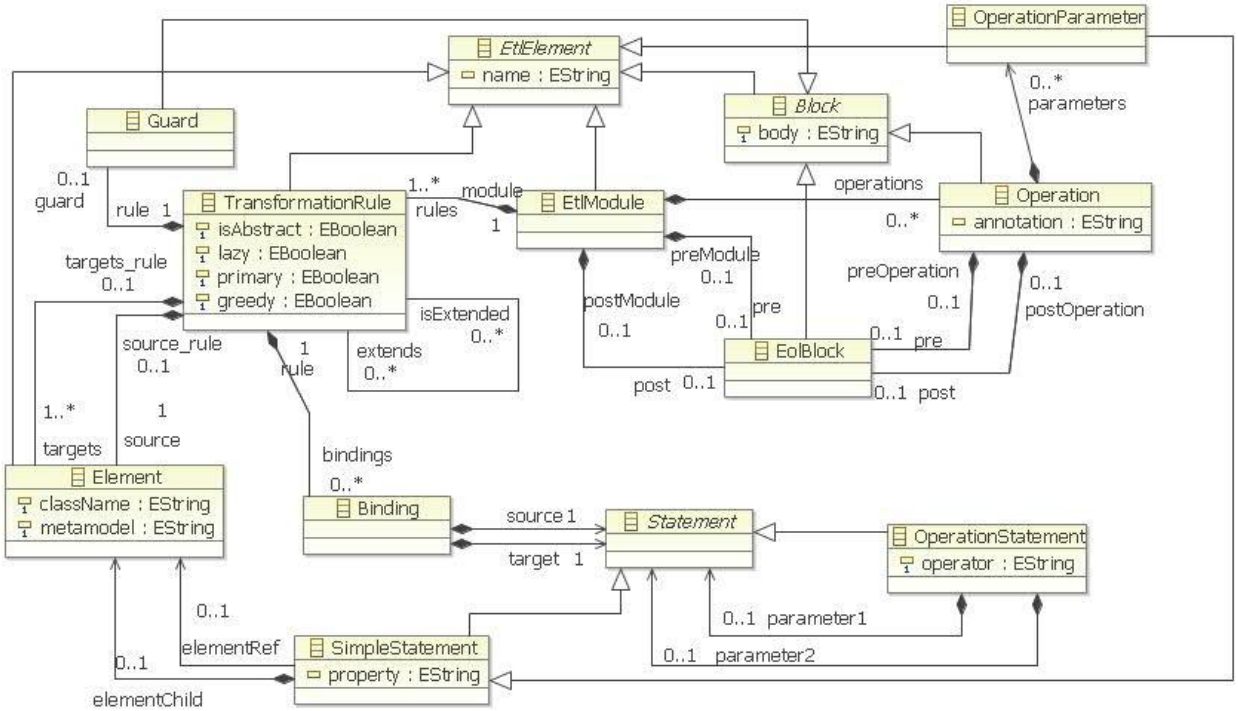

Figura 4-8. Metamodelo de ETL

ETL permite la definición de distintos tipos de reglas: reglas *matched*, reglas *lazy*, reglas abstractas, reglas *greedy* y reglas primarias. En el metamodelo de ETL presentado, la información que describe el tipo de regla creada se define en un conjunto de atributos booleanos en la metaclase *TransformationRule*: *isAbstract*, *lazy*, *primary* y *greedy*. En caso de que el atributo *lazy* sea falso, se asume que se trata de una regla tipo *matched*.

Las funcionalidades de cada una de las reglas que define ETL son las siguientes:

- *Matched Rule*: tienen la misma funcionalidad que en ATL, es decir son las reglas principales de la transformación. Permiten definir qué elemento o elementos del modelo destino se deben generar a partir de uno o varios elementos del modelo origen.
- *Lazy Rule*: en ETL, este concepto es empleado del mismo modo que en ATL, por tanto este tipo de reglas son aquellas que deben ser invocadas explícitamente por otras reglas de la transformación.
- *Abstract Rule*: se trata de reglas abstractas que pueden ser extendidas por otras reglas de la transformación.
- *Greedy Rule*: es un tipo de reglas en las que los elementos que las invocan no tienen porqué ser exactamente del mismo tipo definido, sino que también pueden ser de un subtipo del mismo.
- *Primary Rule*: se trata de reglas que se ejecutan prioritariamente respecto de otras reglas de igual equivalencia.

4.3 Especificación de las Transformaciones

Para automatizar el paso entre los modelos de los distintos niveles de abstracción presentados en MeTAGeM-Trace (ver Figura 4-3) se propone el uso de dos transformaciones de modelos: PIM-PSM y PSM-PDM; y una transformación de modelo a texto: PDM-Código.

En cuanto a la forma de definir estas transformaciones, como se indica en el método de construcción detallado en la sección 2.3, primero se definen en lenguaje natural y posteriormente se implementan mediante un lenguaje de transformaciones bien definido. El hecho de definir las reglas de transformación en lenguaje natural antes de su implementación puede considerarse como las fases de análisis y diseño en el desarrollo de las transformaciones. Dado que el proceso de desarrollo propuesto por MeTAGeM-Trace es guiado por el conjunto de

transformaciones que se describen en esta sección, toda aproximación que ayude a mejorar la calidad de dichas transformaciones, como su definición en lenguaje natural, resulta en modelos mejor construidos y por tanto, favorece la mejora del proceso en general.

En las siguientes sub-secciones se presentan las transformaciones mencionadas en términos de lenguaje natural.

4.3.1 Transformación de modelos PIM a modelos PSM

Para realizar la especificación de las reglas de transformación es necesario analizar los metamodelos que intervienen en la transformación, en este caso los metamodelos definidos a nivel PIM y PSM (metamodelo MeTAGeM y metamodelo de transformaciones de aproximación híbrida, respectivamente), para determinar las relaciones existentes entre los elementos de cada uno de ellos. Una vez que se ha realizado dicho análisis, la Tabla 4-1 muestra las reglas de transformación definidas, a nivel de metamodelo, para crear modelos PSM a partir de modelos PIM. Las reglas abstractas aparecen como sombreadas en la tabla.

En dicha tabla se pueden observar cinco tipos de relaciones principales:

1. La relación de transformación de los elementos raíz de los metamodelos (*ModelRoot -> Module*).
2. La transformación de los elementos que describen a los metamodelos de origen y destino de la transformación.
3. La transformación de las relaciones a nivel PIM en las reglas de la transformación y las relaciones de trazabilidad, a nivel PSM. (Los elementos que describen las relaciones de trazabilidad aparecen en negrita).
4. La transformación de las relaciones a nivel PIM en las asignaciones tipo *binding* y las relaciones de trazabilidad, a nivel PSM. (Los elementos que describen las relaciones de trazabilidad aparecen en negrita).
5. La transformación de los elementos origen y destino implicados en la transformación.

Tabla 4-1. Transformación PIM a PSM

Metamodelo MeTAGeM (PIM)	**Metamodelo Híbrido (PSM)**
ModelRoot	*Module*
SourceModelTransf	*SourceModel*
TargetModelTransf	*TargetModel*

Metamodelo MeTAGeM (PIM)	**Metamodelo Híbrido (PSM)**
Relations (no incluida en otra *Relations*)	*Rule* ***TraceRule***
OneToOne (no incluida en otra *Relations*)	*Rule* - *Source: 1 elemento* - *Target: 1 elemento* ***TraceRule*** - ***Source: 1 elemento*** - ***Target: 1 elemento***
OneToZero (no incluida en otra *Relations*)	*Rule* - *Source: 1 elemento* - *Target: Sin elementos* ***TraceRule*** - ***Source: 1 elemento*** - ***Target: Sin elementos***
ZeroToOne (no incluida en otra *Relations*)	*Rule* - *Source: Sin elementos* - *Target: 1 elemento* ***TraceRule*** - ***Source: Sin elementos*** - ***Target: 1 elemento***
OneToMany (no incluida en otra *Relations*)	*Rule* - *Source: 1 elemento* - *Target: N elementos* ***TraceRule*** - ***Source: 1 elemento*** - ***Target: N elementos***
ManyToOne (no incluida en otra *Relations*)	*Rule* - *Source: N elementos* - *Target: 1 elemento* ***TraceRule*** - ***Source: N elementos*** - ***Target: 1 elemento***
ManyToMany (no incluida en otra *Relations*)	*Rule* - *Source: N elementos* - *Target: N elementos*

Metamodelo MeTAGeM (PIM)	Metamodelo Híbrido (PSM)
	TraceRule - ***Source: N elementos*** - ***Target: N elementos***
Relations (Incluida en otra *Relations*)	*Binding* ***TraceBinding***
OneToOne (Incluida en otra *Relations*)	*Binding* *RightPattern* - *Source: 1 elemento* *LeftPattern* - *Target: 1 elemento* ***TraceBinding*** - ***Source: 1 elemento*** - ***Target: 1 elemento***
ManyToOne (Incluida en otra *Relations*)	*Binding* *RightPattern* - *Source: N elementos* *LeftPattern* - *Target: 1 elemento* ***TraceBinding*** - ***Source: N elementos*** - ***Target: 1 elemento***
ZeroToOne (Incluida en otra *Relations*)	*Binding* *RightPattern* - *Source: Sin elementos* *LeftPattern* - *Target: 1 elemento* ***TraceBinding*** - ***Source: Sin elementos*** - ***Target: 1 elemento***
SourceElement	*Source*
TargetElement	*Target*

4.3.2 Transformación de modelos PSM a modelos PDM

De manera similar que en la sección anterior, se definen a continuación las reglas de transformación entre el metamodelo PSM y los metamodelos especificados a nivel PDM (metamodelo de ATL y ETL).

Es necesario destacar que estas transformaciones son las más complejas de todo el proceso propuesto por MeTAGeM-Trace ya que, además de transformar las reglas de la transformación, es necesario crear las estructuras necesarias que componen el generador de trazas embebido en la transformación. Dichas estructuras se generan a partir de las relaciones de trazabilidad definidas a nivel PSM y de acuerdo a las características propias del lenguaje al que se modela la transformación a nivel PDM.

4.3.2.1 Transformación PSM a ATL (PDM)

En la Tabla 4-2 se muestra una simplificación de las reglas de transformación entre los elementos del metamodelo de nivel PSM (aproximación híbrida) y los elementos del metamodelo de ATL de nivel PDM, usando lenguaje natural. Se trata de una simplificación porque solo se muestran los principales elementos generados, con el objetivo de facilitar la comprensión al lector.

A modo de ejemplo para ilustrar la complejidad de la definición completa de la transformación, en la regla definida entre los módulos de la transformación (a nivel PSM y PDM) para generar correctamente el elemento de salida *CalledRule ('CreateTraceModelRoot')* es necesario generar, además de dicho elemento, otros diez: *OutPattern*, *SimpleOutPattern*, *OclModelElement*, *ActionBlock*, *BindingStat*, dos *VariableExp*, *NavigationOrAttributeCallExp* y dos *VariableDeclaration*.

Al igual que en la tabla anterior, los elementos generados con el objetivo de obtener el generador de trazas embebido en la transformación son mostrados en negrita.

Tabla 4-2. Transformación PSM a ATL (PDM)

Metamodelo Híbrido (PSM)	Metamodelo ATL (PDM)
Module	*Module* ***OclModel (TraceModel)*** ***CalledRule ('CreateTraceModelRoot')*** ***Helper ('getTraceModelRoot')*** ***Helper ('getName')***

Metamodelo Híbrido (PSM)	**Metamodelo ATL (PDM)**
SourceModel	*OclModel (SourceModel)* ***CalledRule('createSourceModel_TraceModel')*** ***Helper ('getSourceModel_TraceModel')***
TargetModel	*OclModel (TargetModel)* ***CalledRule('createTargetModel_TraceModel')*** ***Helper ('getTargetModel_TraceModel')***
Rule (isMain==true and sources>0)	*MatchedRule* *InPattern* *OutPattern*
Rule (isMain==false and sources>0 and isUnique==false)	*LazyMatchedRule* *InPattern* *OutPattern*
Rule (isMain==false and sources>0 and isUnique==true)	*UniqueLazyMatchedRule* *InPattern* *OutPattern*
Rule (sources==0)	*CalledRule* *OutPattern*
*Source (*entrada de una regla *and traceLink==0)*	*SimpleInPatternElement*
*Source (*entrada de una regla *and traceLink>0)*	*SimpleInPatternElement* ***SimpleOutPatternElement (TraceElement:Source)***
*Source (*entrada de una *RightPattern and traceLink>0)*	***SimpleOutPatternElement (TraceElement:Source)***
*Target (*entrada de una regla *and traceLink==0)*	*SimpleOutPatternElement*
*Target (*entrada de una regla *and traceLink>0)*	*SimpleOutPatternElement* ***SimpleOutPatternElement (TraceElement:Target)***
*Target (*entrada de una *RightPattern and traceLink>0)*	***SimpleOutPatternElement (TraceElement:Target)***
Binding	*Binding*
Operation	*Helper*
Argument	*Parameter*

Metamodelo Híbrido (PSM)	Metamodelo ATL (PDM)
TraceRule	***SimpleOutPatternElement (TraceLink)*** - ***source: rule.sources*** - ***target: rule.targets*** - ***childLinks: traceRule.traceBindings***
TraceBinding	***SimpleOutPatternElement (TraceLink)*** - ***source: binding.sources*** - ***target: binding.targets***
Datatype Boolean	*BooleanType*
Datatype Integer	*IntegerType*
Datatype String	*StringType*
TransformationElement	*OclModelElement*

4.3.2.2 Transformación PSM a ETL (PDM)

Al igual que para el lenguaje ATL, en esta ocasión la Tabla 4-3 muestra una simplificación de las relaciones de transformación definidas entre los elementos del metamodelo PSM y los elementos del metamodelo de ETL (nivel PDM).

Tabla 4-3. Transformación PSM a ETL (PDM)

Metamodelo Híbrido (PSM)	Metamodelo ETL (PDM)
Module	*EtlModule*
SourceModel	***Añade información al bloque*** *Pre*
TargetModel	Añade información al bloque *Pre*
Rule (sources==1 and targets>0)	*TransformationRule* - *source: rule.sources* - *target: rule.targets +**r.trace+ r.trace.traceBindings***
Binding	*Binding*
LeftPattern	*SimpleStatement*
TransformationElement (incluido en un *OpDefinition or* en un *OpArgument*)	*Element*
Source (incluido en una *Rule and traceLink==0*)	*Element*
Source (incluido en una *Rule and traceLink>0*)	*Element* ***Element (TraceElement:Source)***

Metamodelo Híbrido (PSM)	Metamodelo ETL (PDM)
Source (incluido en un *RightPattern and traceLink==0*)	*Element*
Source (incluido en un *RightPattern and traceLink>0*)	*Element* ***Element (TraceElement:Source)*** - ***belongsTo: source.rule***
Target(incluido en una *Rule and traceLink==0*)	*Element*
Target (incluido en una *Rule and traceLink>0*)	*Element* ***Element (TraceElement:Target)***
Target (incluido en un *LeftPattern and traceLink==0*)	*Element*
Target (incluido en un *LeftPattern and traceLink>0*)	*Element* ***Element (TraceElement:Target)*** - ***belongsTo: target.rule***
TraceRule	***Element (TraceLink)*** - ***source: traceRule.sources*** - ***target: traceRule.targets***
TraceBinding	***Element (TraceLink)*** - ***source: traceBinding.sources*** - ***target: traceBinding.targets***
Guard	*Guard*
Operation	*Operation*
OpDefinition (return==Boolean or return==Integer or return==String)	*SimpleStatement*
OpArgument (return==Boolean or return==Integer or return==String)	*OperationParameter*

Además de las reglas de transformación anteriores, es necesario definir tres operaciones imperativas para crear los bloques pre y post (*EolBlock*) y para crear una operación *getName* que proporcione en tiempo de ejecución los nombres de los elementos que participan en la transformación.

4.3.3 Transformación de modelo PDM a código

Dentro del proceso descrito por MeTAGeM-Trace, la última transformación se refiere a la serialización de los modelos PDM al código que implementa la transformación desarrollada.

En el caso de lenguaje ATL no ha sido necesario definir esta transformación de modelo a texto, ya que como se ha indicado anteriormente el *plug-in* de ATL para Eclipse proporciona un extractor de código ATL a partir de modelos conformes al metamodelo del lenguaje. En el caso de ETL, al no disponer de estos extractores o generadores de código, sí es necesario definir cómo se genera el código de la transformación a partir del metamodelo de ETL definido.

Para llevar a cabo esta tarea, se ha seguido el mismo procedimiento que en las transformaciones entre modelos. Así, en la Tabla 4-4 se presenta la transformación entre los modelos ETL y el código ETL en lenguaje natural.

Tabla 4-4. Transformación ETL (PDM) a ETL (Código)

Metamodelo Híbrido (PSM)	**Metamodelo ETL (PDM)**

EtlModule	“pre ” *pre.name* “ {”
	  pre.body
	“}”
	module.transformationRules
	module.operations
	“post ” *pre.name* “ {”
	  post.body
	“}

TransformationRule	[(“*@greedy*”, “*@abstract*”, “*@lazy*”, “*@primary*”)]
	“rule” *name*
	“transform” *source*
	“to” *targets*
	[“extends” *extends*]
	“{“
	  [“guard: ” *guard.body*]
	  bindings
	“}”

Binding	*target* “ := ” *source* “ ; ”
SimpleStatement (*Binding*)	[*ref* “.”] *property*
SimpleStatement (*Operation*)	[*child.metamodel* “!” *className*] [*ref* “.”] *property*

Metamodelo Híbrido (PSM)	Metamodelo ETL (PDM)
OperationStatement	[*parameter1*] *operator* [*parameter2*]
Element	*name* " : " *metamodel* "! " *class*
Operation	["@" *annotation*] ["@" *pre*] ["@" *post*] "operation" *context* ["(" *parameters* ")"] ": return" *return* "{" *body* "}"
OperationParameter	*name* " : " [*ref.metamodel* "!" *ref.className*] ["." *property*]

4.4 Especificación de la validación de los modelos

Una de las principales ventajas de aplicar los principios de MDE al desarrollo de las transformaciones es que estas pueden ser tratadas como cualquier otro modelo del proceso de desarrollo, de forma que se puedan realizar tareas propias de MDE sobre ellas [21].

Una de las tareas más comunes y más soportadas por las herramientas MDE es la validación de modelos. Desde que los sistemas se construyen a partir de la definición de un conjunto de modelos previos a la implementación del mismo, cualquier error en un modelo puede ser transmitido a los modelos posteriores y generalmente, hasta la implementación final del sistema. Por este motivo, el uso de mecanismos de validación de los modelos resulta clave para la detección de errores e inconsistencias en los modelos previos a la implementación y de esta forma, mejorar la calidad de dichos modelos y en consecuencia, del sistema final [141].

Dado que las transformaciones son otro de los elementos clave en el desarrollo dirigido por modelos, sería recomendable aplicar mecanismos de validación a los modelos que definen las transformaciones a alto nivel y de esta forma mejorar la calidad de las transformaciones resultante. Así, en esta sección se presentan las reglas de validación definidas para los metamodelos de los niveles de abstracción PIM y PSM que se definen en MeTAGeM-Trace. Al igual que en el caso de las transformaciones, en primer lugar, las reglas de validación se definirán en lenguaje natural.

4.4.1 Validación del metamodelo PIM

En la Tabla 4-5 se muestran, en lenguaje natural, las reglas de validación que debe cumplir todo modelo conforme al metamodelo definido a nivel PIM. Estas reglas se basan, principalmente, en asegurar que el nombre de los elementos esté bien formado y que se cumpla la cardinalidad de las relaciones entre los elementos.

Tabla 4-5. Validación del metamodelo PIM

Metaclase	Reglas de Validación
ModelRoot	- El atributo *name* debe estar definido. - El atributo *name* debe ser válido (debe comenzar por una letra y solo puede contener letras, números y caracteres '-' o '_') - Debe contener como mínimo un modelo origen - Debe contener como mínimo un modelo destino - Debe contener como mínimo una relación
ModelTransf	- El atributo *name* debe estar definido. - El atributo *name* debe ser válido.
SourceModelTransf	- Las mismas que *ModelTransf* (herencia).
TargetModelTransf	- Las mismas que *ModelTransf* (herencia).
Relations	- El atributo *role* debe estar definido. - El atributo *typeElement* debe estar definido. - El atributo *typeRelation* debe estar definido. - Solo puede extender a relaciones abstractas.
OneToOne	- Las mismas que *Relations* (herencia). - Debe tener un elemento origen. - Debe tener un elemento destino.
OneToZero	- Las mismas que *Relations* (herencia). - Debe tener un elemento origen.
ZeroToOne	- Las mismas que *Relations* (herencia). - Debe tener un elemento destino.
OneToMany	- Las mismas que *Relations* (herencia). - Debe tener un elemento origen. - Debe tener uno o más elementos destino.
ManyToMany	- Las mismas que *Relations* (herencia). - Debe tener uno o más elementos origen. - Debe tener uno o más elementos destino.

4.4.2 Validación del metamodelo PSM

Al igual que en el caso anterior, la Tabla 4-6 muestra las reglas de validación que deben cumplir los modelos conformes al metamodelo definido a nivel PSM (metamodelo de aproximación híbrida).

Tabla 4-6. Validación del metamodelo PSM

Metaclase	Reglas de Validación
Module	- El atributo *name* debe estar definido. - El atributo *name* debe ser válido (debe comenzar por una letra y solo puede contener letras, números y caracteres '-' o '_') - Debe contener como mínimo un modelo origen - Debe contener como mínimo un modelo destino - Debe contener como mínimo una relación
Model	- El atributo *name* debe estar definido. - El atributo *name* debe ser válido. - El atributo *type_mm* debe estar definido. - El atributo *path* debe estar definido
SourceModel	- Las mismas que *Model* (herencia).
TargetModel	- Las mismas que *Model* (herencia).
Rule	- El atributo *name* debe estar definido. - El atributo *isAbstract* debe estar definido. - El atributo *isMain* debe estar definido. - El atributo *typeElement* debe estar definido. - El atributo *typeRelation* debe estar definido. - Solo puede extender a reglas abstractas.
Operation	- El atributo *name* debe estar definido. - El atributo *body* debe estar definido.
OpDefinition	- Debe devolver (*return*) un tipo de dato (*datatype*) o un elemento (*element*), tan solo uno de los dos.
OpArgument	- El atributo *name* debe estar definido. - Debe devolver (*return*) un tipo de dato (*datatype*) o un elemento (*element*), tan solo uno de los dos.
Guard	- El atributo *value* debe estar definido.
TransformationElement	- El atributo *name* debe estar definido.
Source	- Las mismas que *TransformationElement* (herencia).
Target	- Las mismas que *TransformationElement* (herencia).

Metaclase	Reglas de Validación
Binding	- El atributo *typeElement* debe estar definido. - El atributo *typeRelation* debe estar definido. - El atributo *left* debe estar definido. - El atributo *right* debe estar definido.
RightPattern	- Debe definir por lo menos uno de los siguientes atributos: *concreteValue, source, operation, reference* o *rule.* - Si define un valor concreto (*concreteValue*) no puede definir los atributos *source, operation, reference y rule.* - No puede definir al mismo tiempo los atributos *rule* y *operation.* - No puede definir al mismo tiempo los atributos *rule* y *reference.* - No puede definir al mismo tiempo los atributos *reference* y *operation.* - No puede definir al mismo tiempo los atributos *reference* y *source.*
LeftPattern	- El atributo *target* debe estar definido.

Las reglas de validación descritas en la tabla anterior son generales para cualquier modelo conforme al metamodelo PSM definido. Sin embargo, dado que los modelos PSM serán transformados en modelos conformes a los metamodelos de ATL o ETL (nivel PDM), resulta recomendable establecer reglas de validación adicionales para los modelos que sean transformados a uno u otro lenguaje. Así, en las próximas sub-secciones se presentan dichas reglas de validación que se suman a las ya presentadas:

4.4.2.1 ATL

En la Tabla 4-7, se muestran las reglas de validación que debe cumplir todo modelo PSM para su correcta transformación a un modelo ATL. Estas reglas de validación se establecen de acuerdo a las características particulares del lenguaje de transformación ATL. Por ejemplo, la regla de validación adicional que se comprueba en la metaclase *Operation* deriva de la obligatoriedad de definir, en ATL, el tipo de información que devuelve un *helper*.

Tabla 4-7. Validación del metamodelo PSM para transformaciones a ATL

Metaclase	Reglas de Validación
Rule	- Una regla con más de un elemento origen debe establecer el atributo *typeRelation* a valor *'Concatenation'.*
Operation	- El atributo *return* debe estar definido.
Binding	- Un *binding* con más de un elemento origen debe establecer el atributo *typeRelation* a valor *'Concatenation'.*
RightPattern	- Si pertenece a una regla que no tiene elementos origen, no puede definir los siguientes atributos: *source, operation y rule.*

4.4.2.2 ETL

Del mismo modo que en la sección anterior, la Tabla 4-8 presenta las reglas de validación adicionales que debe cumplir todo modelo PSM para ser transformado en un modelo conforme a las características del lenguaje de transformación ETL.

Tabla 4-8. Validación del metamodelo PSM para transformaciones a ETL

Metaclase	Reglas de Validación
Rule	- Solo puede tener un elemento origen.
Binding	- Un *binding* con más de un elemento origen debe establecer el atributo *typeRelation* a valor *'Concatenation'.*

Solución Tecnológica: MeTAGeM-Trace

Para dar soporte a la metodología definida en el capítulo anterior, en este capítulo se presenta el desarrollo de la **herramienta MeTAGeM-Trace,** que se compone de:

- Una arquitectura de capas o niveles que describe la relación entre los componentes que forman parte de la herramienta.
- La implementación de los metamodelos que describen la transformación y las relaciones de trazabilidad a distintos niveles de abstracción.
- La implementación del metamodelo de trazabilidad que describe las trazas generadas en la ejecución de una transformación de modelos.
- Un conjunto de editores que permitan el modelado y visualización de los modelos que describen la transformación y las trazas generadas, de acuerdo a los metamodelos implementados.
- Un conjunto de restricciones para llevar a cabo la validación de los modelos que describen la transformación y las relaciones de trazabilidad.
- La implementación de los conjuntos de reglas de transformación que permiten realizar transformaciones entre los modelos definidos en los diferentes niveles de abstracción y la posterior generación de código.

Para definir de la estructura de MeTAGeM-Trace se ha seguido el proceso propuesto para la definición de las herramientas MeTAGeM [30] y M2DAT [199]. Dichos proceso, se basa en la especificación de la herramienta teniendo en cuenta dos niveles: la arquitectura conceptual y el diseño técnico.

En cuanto a la construcción o implementación de la herramienta se propone un proceso iterativo e incremental basado en módulos interconectados, de forma que cada uno de estos módulos se construya de acuerdo al método de desarrollo que se detalla en la sección 2.3.

Así, en las próximas secciones se presenta la arquitectura conceptual de MeTAGeM-Trace, omitiendo los detalles técnicos (sección 5.1); el diseño técnico que establece las tecnologías a usar para hacer realidad la arquitectura conceptual (sección 5.2); y finalmente, la descripción de la implementación de la herramienta (sección 5.3).

5.1 Arquitectura Conceptual de MeTAGeM-Trace

La arquitectura de MeTAGeM-Trace, que es mostrada por la Figura 5-1, sigue la arquitectura **basada en capas** propuesta por MeTAGeM [30]. Esta arquitectura mantiene un alto grado de **modularización**, de forma que MeTAGeM-Trace puede verse como un conjunto de módulos independientes que interactúan entre sí. Estos módulos se consideran independientes porque cada uno de ellos define un DSL completo para dar soporte a uno de los metamodelos presentados en la propuesta metodológica, esto es, proporciona el soporte necesario (editores de modelos, mecanismos de validación, transformaciones, etc.) para interactuar con los modelos terminales conformes al metamodelo del DSL. Concretamente, como muestra la Figura 5-1, la arquitectura de MeTAGeM-Trace se compone de **cinco módulos** diferentes: uno para el modelado de las relaciones de transformación a nivel PIM (DSL-MeTAGeM), uno para el modelado de las reglas de transformación y relaciones de trazabilidad a nivel PSM (DSL-Aproximación Híbrida), uno para el modelado de transformaciones basadas en el lenguaje ATL (DSL-ATL), uno para el modelado de transformaciones basadas en el lenguaje ETL (DSL-ETL) y uno para el modelado de las trazas generadas a partir de la ejecución de las transformaciones (DSL-Trace). Una de las principales ventajas de esta modularización es la posibilidad de incluir, en el futuro, nuevos DSLs como por ejemplo para dar soporte a otros lenguajes de transformación de modelos o nuevos paradigmas de transformación, sin afectar a los ya existentes. Del mismo, es posible eliminar de forma sencilla cualquiera de los módulos de la herramienta.

Además, cada uno de los módulos que componen la arquitectura conceptual de MeTAGeM-Trace sigue una aproximación basada en capas [118, 119, 161], donde cada una de las capas representa una vista en particular de la arquitectura: capa de presentación, capa lógica y capa de persistencia.

La **capa de presentación** contiene los editores que le permiten al usuario de MeTAGeM-Trace manipular y visualizar los modelos terminales conformes a los metamodelos de los DSLs que componen la herramienta. Además, contiene un conjunto de asistentes de creación para generar desde cero dichos modelos.

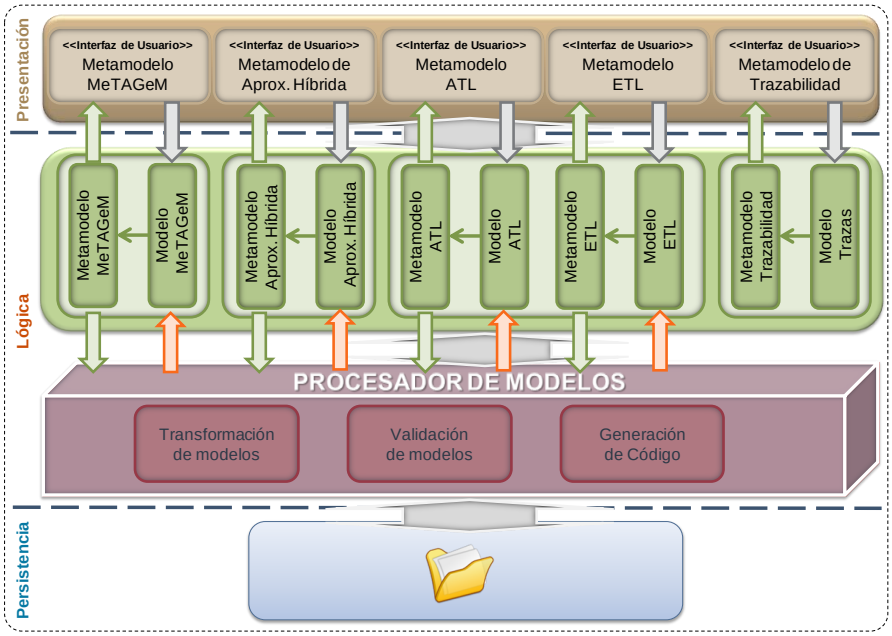

Figura 5-1. Arquitectura Conceptual de MeTAGeM-Trace

La **capa lógica** se puede comprobar la existencia de dos subcapas o subniveles. El primero de ellos se corresponde con el manejador lógico de los modelos, es decir, proporciona a la capa de presentación las acciones (creación, modificación, borrado, etc.) que el usuario puede realizar sobre el contenido de los modelos a través de la interfaz de la herramienta. El segundo nivel, denominado **procesador de modelos** [215], consiste en un módulo donde se alojan las tareas adicionales, propias de la gestión de los modelos [21, 23, 54, 186], que permiten la integración de los DSLs mediante transformaciones entre modelos, la generación de código a partir de los modelos o la validación de los modelos para mejorar la calidad de los mismos. En concreto, el procesador de modelos incluido en la herramienta MeTAGeM-Trace consta de:

- Un conjunto de reglas de transformación entre los modelos MeTAGeM y los modelos de transformación que siguen una aproximación híbrida (transformación PIM-PSM).
- Un conjunto de reglas de transformación entre los modelos de transformación que siguen una aproximación híbrida y los modelos que definen la transformación a nivel PDM de acuerdo a las características del lenguaje ATL (transformación PSM-PDM).
- Un conjunto de reglas de transformación entre los modelos de transformación que siguen una aproximación híbrida y los modelos que

definen la transformación a nivel PDM de acuerdo a las características del lenguaje ETL (transformación PSM-PDM).

- Un conjunto de reglas de validación para verificar la validez de los modelos MeTAGeM.
- Un conjunto de reglas de validación para verificar la validez de los modelos de transformación que siguen una aproximación híbrida.
- Un conjunto de reglas de transformación de modelo a código para serializar los modelos conformes al metamodelo de ATL en el código ATL que implementa la transformación (generación de código ATL).
- Un conjunto de reglas de transformación de modelo a código para serializar los modelos conformes al metamodelo de ETL en el código ETL que implementa la transformación (generación de código ETL).

Por último, se encuentra la **capa de persistencia** que se define como un sistema de archivos que incorpora las políticas tradicionales de control de versiones.

5.2 Diseño técnico de MeTAGeM-Trace

Después de definir la arquitectura conceptual de MeTAGeM-Trace, el siguiente paso consiste en la selección de las tecnologías que permitirán poner en práctica dicha arquitectura. Como se ha indicado en varias ocasiones a lo largo de esta tesis doctoral y detallado en la sección 2.3, para llevar a cabo la construcción o implementación de la herramienta se ha seguido una adaptación del método de desarrollo presentado por Vara en [199]. De acuerdo a dicho método, la plataforma seleccionada para la construcción de MeTAGeM-Trace es el entorno de desarrollo Eclipse y en particular EMF (*Eclipse Modeling Framework*, [40, 188]), un *framework* de Eclipse que brinda facilidades para la generación de editores a partir de metamodelos definidos por el usuario. Además, todas las herramientas empleadas para la construcción de la herramienta (ATL, EVL y MOFScript) son extensiones o *plug-ins* de Eclipse y la herramienta MeTAGeM-Trace en sí misma, también es construida como un *plug-in* de dicha plataforma.

A continuación, en la Figura 5-2, se muestra la relación entre la arquitectura conceptual de MeTAGeM-Trace y la tecnología empleada para su construcción.

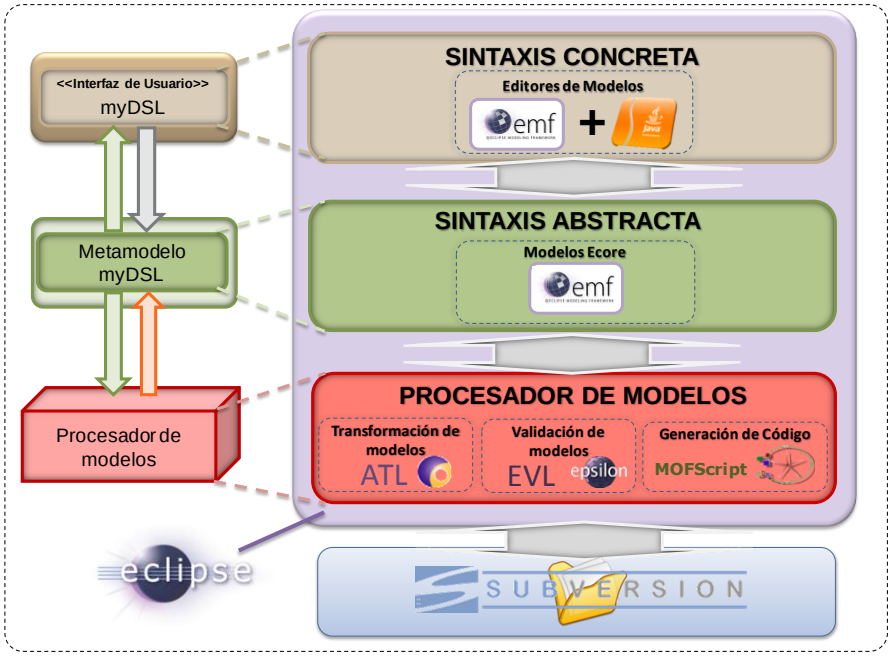

Figura 5-2. Diseño técnico de MeTAGeM-Trace

Siguiendo la estructura mostrada por la figura, para la **sintaxis concreta** de los DSL que se corresponde con la capa de persistencia, se ha decidido implementar un **editor de modelos** para cada uno de los metamodelos de los DSLs que componen la herramienta. Para llevar a cabo esta tarea, EMF proporciona de forma semi-automática un editor en forma de árbol, construido sobre el lenguaje de programación Java. Sin embargo, estos editores por defecto ofrecen una visualización genérica que no se ajusta a la naturaleza de los modelos gestionados por MeTAGeM-Trace. Por este motivo, como se detalla en la sección 5.3.2, se ha empleado el propio lenguaje de programación Java para crear editores de modelos *ad-hoc*.

La **sintaxis abstracta**, que representa la especificación de cada DSL, es manejada mediante las facilidades proporcionadas por el *framework* EMF para la manipulación de modelos. Esta herramienta permite, entre otras cosas, almacenar y recuperar los modelos en el formato XMI [157]. Esta sintaxis abstracta es usada como entrada o salida de las tareas de modelado que se incluyen en el **procesador de modelos**.

Como se ha indicado en la sección anterior, MeTAGeM-Trace ofrece soporte para la ejecución de transformaciones entre los modelos de los diferentes niveles de abstracción, para la validación de los modelos a nivel PIM y PSM y para la generación del código de la transformación en los lenguajes ATL y ETL.

De acuerdo a las tecnologías propuestas en [199] para dar soporte a estas actividades de modelado en el contexto de la plataforma Eclipse, para el desarrollo de las transformaciones de modelos se ha empleado el lenguaje ATL; para la validación de los modelos se ha usado el lenguaje EVL (*Epsilon Validation Language*, [116]), perteneciente a la familia de lenguajes de Epsilon; y para la generación de código se ha usado la herramienta MOFScript [144].

Finalmente, para la **persistencia** de los modelos con MeTAGeM-Trace se usa un sistema de control de versiones tradicional. En concreto, se utiliza un *plug-in* de Eclipse denominado *Subclipse.* Este *plug-in* es una implementación para Eclipse del sistema de control de versiones *Subversion* [165]

5.3 Implementación

En las secciones anteriores se han presentado los componentes que forman la estructura de la herramienta MeTAGeM-Trace y los medios tecnológicos a emplear para llevar a cabo su implementación. En esta, se detalla cómo se ha realizado la construcción de cada uno de ellos sobre la plataforma **Eclipse**.

Así, siguiendo el orden de construcción definido en el método de desarrollo (sección 2.3), en primer lugar se muestra la implementación de los metamodelos. Posteriormente, se detalla la construcción de los editores que permiten interactuar con los modelos terminales conformes a los metamodelos construidos y la construcción de los asistentes para la creación de dichos modelos. A continuación se muestra la implementación de las tareas incluidas en el procesador de modelos: transformaciones de modelos, generadores de código y validación de modelos. Y finalmente, se describe cómo se ha llevado a cabo la integración de todos los módulos que componen la herramienta.

5.3.1 Implementación de Metamodelos

En la sección 4.2 se ha detallado la especificación de los metamodelos que forman parte de MeTAGeM-Trace. Con el objetivo de incluir dichas especificaciones en la herramienta de soporte, esta sección presenta la implementación de dichos metamodelos que, como se ha mencionado en la sección anterior, se ha llevado a cabo mediante las funcionalidades ofrecidas por el ***framework* EMF**.

5.3.1.1 Metamodelo MeTAGeM (nivel PIM)

Para permitir el modelado de las relaciones de transformación entre varios metamodelos de origen y destino a alto nivel y de forma independiente a los detalles tecnológicos, en la sección 4.2.2 se presentó la especificación del metamodelo MeTAGeM, sobre la que se han realizado unos pocos cambios respecto de la especificación del metamodelo presentado para el mismo propósito en [30]. En cambio, en cuanto a la implementación existe una evolución fundamental que ha requerido la implementación completa del metamodelo.

El entorno de DDM de transformaciones de modelos MeTAGeM [30], antecesor del entorno MeTAGeM-Trace que se presenta en esta tesis doctoral, emplea la funcionalidad ofrecida por la herramienta AMW (*Atlas Model Weaver*, [54]) para la implementación de DSL a nivel PIM. La herramienta AMW permite establecer las relaciones entre modelos mediante modelos de *weaving.* Sin embargo, en los últimos tiempos dicha herramienta no está siendo actualizada al mismo ritmo que la plataforma Eclipse y el resto de tecnologías necesarias para la construcción de la herramienta MeTAGeM-Trace, dando lugar a algunos problemas de compatibilidad. De hecho para la construcción del entorno MeTAGeM fue necesario establecer una colaboración con el desarrollador de AMW con el objetivo de obtener una versión actualizada de la herramienta [30]. Por este motivo, y a pesar de ofrecer mecanismos propios para la visualización y creación de los modelos, se ha decidido prescindir de AMW para el desarrollo de MeTAGeM-Trace. Para reemplazar la funcionalidad ofrecida por AMW, se ha optado por emplear EMF, el *framework* sobre el que trabaja AMW, e implementar dicha funcionalidad de acuerdo a las necesidades del DSL MeTAGeM.

En cuanto a la implementación del metamodelo que describe la especificación presentada en la sección 4.2.2, se ha definido un modelo *Ecore*, que se muestra en la Figura 5-3.

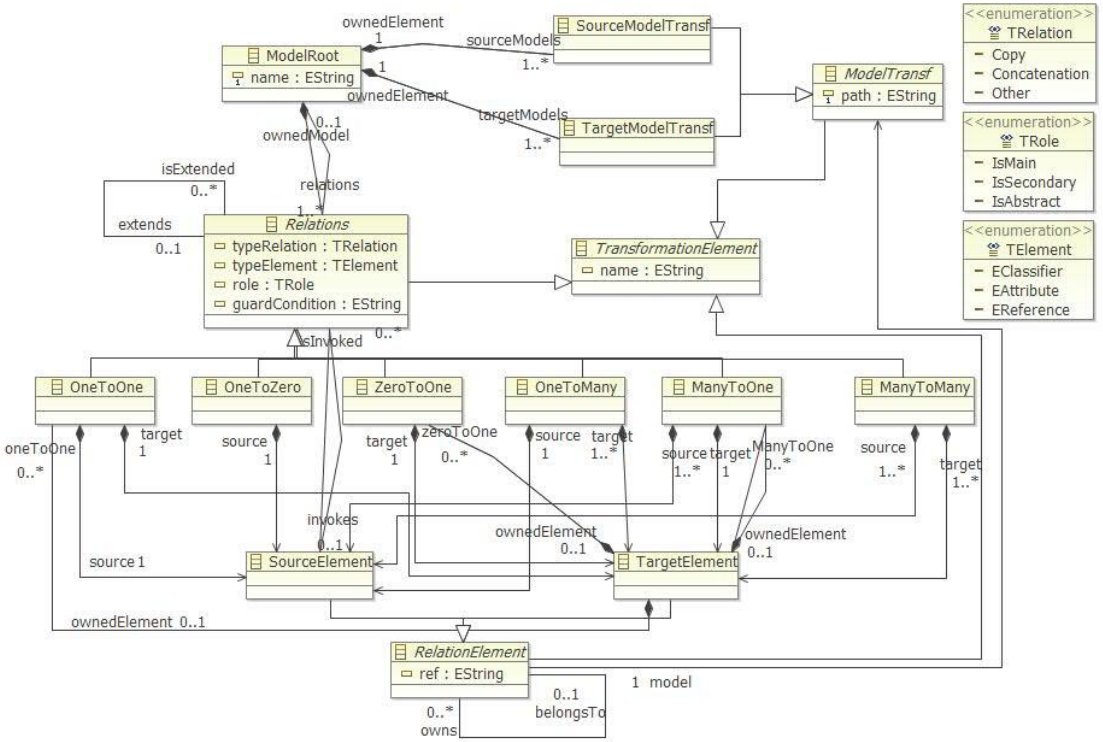

Figura 5-3. Metamodelo MeTAGeM (Modelo *Ecore*)

Ecore, que es el metametalenguaje de EMF, es una implementación simplificada de EMOF (*Essential MOF*, [154]) que genera un archivo con extensión *.ecore* donde se implementan cada una de las metaclases especificadas en el metamodelo. Es importante mencionar, que debido a la naturaleza XML subyacente de *Ecore*, cualquier metamodelo definido a partir de *Ecore* debe incluir un elemento raíz. En el caso de este metamodelo, la metaclase raíz es *ModelRoot.*

Para sustituir la funcionalidad mencionada, proporcionada por AMW, ha sido necesario incluir cierta información que anteriormente era heredada de los metamodelos base de AMW. Así, por ejemplo para conocer la ubicación física de los metamodelos que intervienen en la transformación, se ha definido un atributo *path* en la metaclase abstracta *ModelTransf.* Para identificar los elementos implicados en la transformación con los objetos reales de los metamodelos es posible emplear dos opciones: definir una referencia a un tipo *EObject* (el elemento del cual heredan todos los elementos de los modelos Ecore) y de esta forma, enlazar los elementos o registrar el identificador XMI que poseen los elementos de los modelos Ecore, para posteriormente buscar la coincidencia de este dato. En este caso, se ha optado por la segunda opción, dado que la primera implica cargar los metamodelos que participan en la transformación, en el editor de los modelos MeTAGeM.

5.3.1.2 Metamodelo de Aproximación Híbrida (nivel PSM)

De la misma forma que en caso anterior, para la implementación del metamodelo de transformación específico de plataforma, que describe las transformaciones que siguen una aproximación híbrida (sección 4.2.3), se ha creado un modelo *Ecore*, que se muestra en la Figura 5-4.

En este caso, el elemento raíz del modelo *Ecore* es la metaclase *Module* que contiene la representación de los metamodelos participantes en la transformación, las reglas de dicha transformación y las operaciones definidas por el usuario en el ámbito de la transformación.

Como se detallará más adelante, también se desea que a este nivel se visualice al mismo tiempo los metamodelos de origen y destino de la transformación y las relaciones existentes entre los elementos de dichos metamodelos (reglas de transformación y relaciones de trazabilidad). Para ello, ha sido necesario implementar en el metamodelo a nivel PSM un conjunto de conceptos que ya han aparecido en el metamodelo a nivel PIM (atributo *path* en las metaclases que describen los metamodelos y uso de las referencias XMI para identificar el elemento que forma parte de la regla de la transformación).

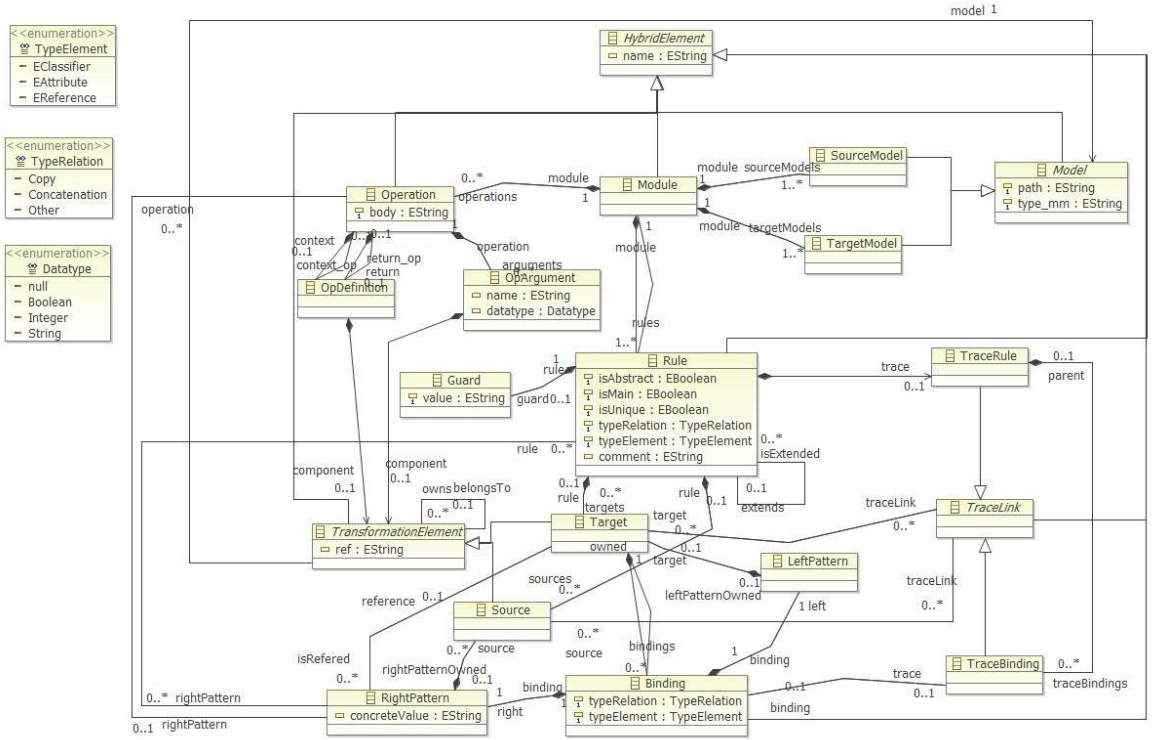

Figura 5-4. Metamodelo de Aproximación Híbrida (Modelo *Ecore*)

Acerca de la implementación de las referencias entre las metaclases del metamodelo de aproximación híbrida, cabe destacar que se han implementado de forma bidireccional con el objetivo de facilitar el desarrollo de las transformaciones en las que este metamodelo interviene como origen o destino.

5.3.1.3 Metamodelos Dependientes de Plataforma (nivel PDM)

Como ya se adelantó en la sección 4.2.4, para el modelado de las transformaciones a nivel PDM se propone el uso de los lenguajes de transformación de aproximación híbrida ATL y ETL. Para el modelado del lenguaje ATL se utiliza el *plug-in* existente de ATL para Eclipse, por lo que no es necesario implementar el DSL correspondiente.

En el caso del modelado de la transformación en el lenguaje ETL, dado que no se dispone de ninguna implementación para este propósito, sí es necesario implementar un DSL para este lenguaje de transformación.

Al igual que en la implementación de los metamodelos anteriores, la construcción del metamodelo de ETL se ha realizado en términos de un modelo *Ecore*, que se muestra en la Figura 5-5. El elemento principal o raíz de este metamodelo es la metaclase *EtlModule* que representa la transformación en sí misma. Dicho módulo se compone de operaciones, reglas de transformación y bloques de código EOL. Estos bloques de código EOL no pertenecen explícitamente a la sintaxis abstracta del lenguaje, sin embargo para ofrecer algunas funcionalidades, ETL utiliza algunos conceptos del lenguaje EOL. Por este motivo, se ha definido la metaclase abstracta *Block* que contiene un atributo de tipo *string* que permite incluir código en la transformación.

En cuanto a las reglas de transformación de ETL, como ya se presentó en la sección 4.2.4.2, es posible definir reglas tipo *matched*, reglas abstractas, reglas *lazy*, reglas *greedy* y reglas *primary*. Para determinar de qué tipo de regla se trata, en la metaclase *TransformationRule* se ha definido un conjunto de atributos booleanos (*isAbstract, lazy, primary* y *greedy*). Cada regla se compone de un elemento de entrada, uno o más elementos de salida y un conjunto de asignaciones (*bindings)*. Además es posible establecer una pre-condición que debe cumplirse antes de llevar a cabo la ejecución de la regla (metaclase *guard*).

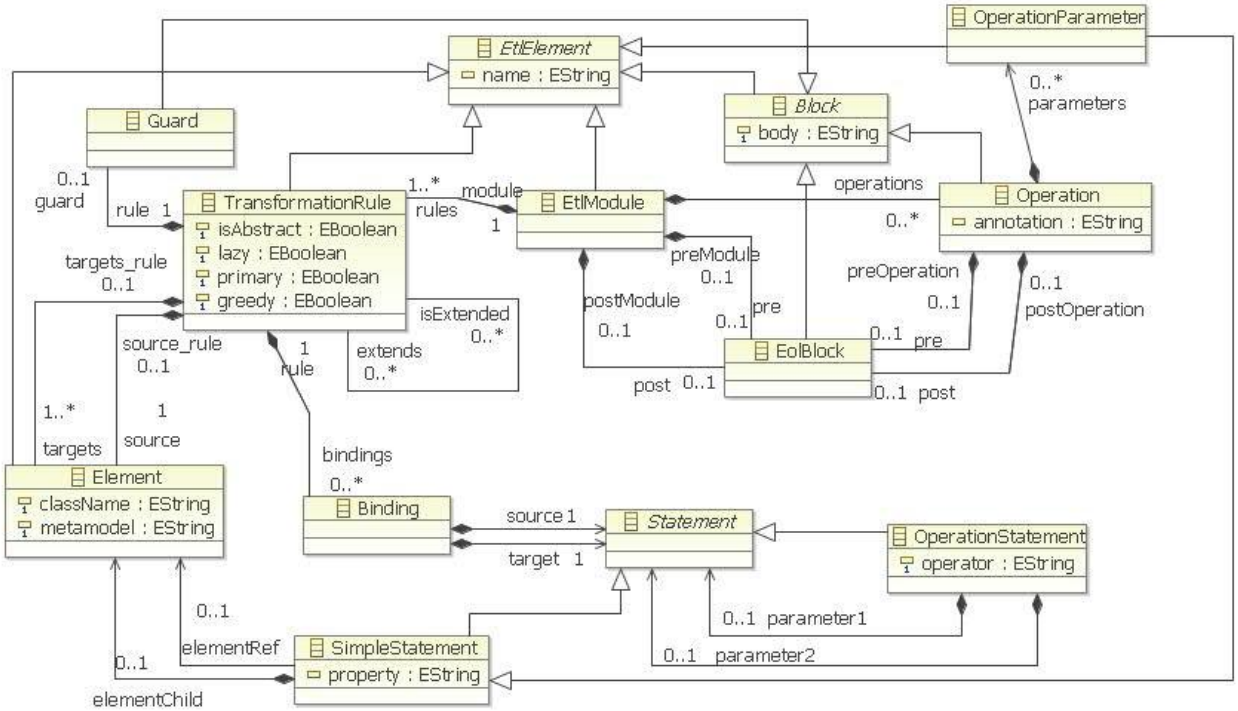

Figura 5-5. Metamodelo de ETL (Modelo *Ecore*)

5.3.1.4 Metamodelo de Trazabilidad

En las secciones anteriores se ha presentado la implementación de los metamodelos de los distintos niveles de abstracción a los que se propone el modelado de la transformación que incluye el generador de trazas. En esta, se presenta la implementación del metamodelo de trazabilidad que define la sintaxis abstracta de los modelos de trazas que serán creados por el generador de trazas embebido en las transformaciones generadas con la herramienta MeTAGeM-Trace. La Figura 5-6 dicha implementación, en términos de un modelo *Ecore*.

Como toda implementación de un metamodelo con EMF, es necesario definir un elemento raíz, que en este caso es la metaclase *TraceModel*. Al igual que en la implementación de los metamodelos PIM y PSM, con el objetivo de conocer la ubicación física de los modelos de entrada y salida, se define un atributo *path* que almacena esta información (metaclase M*odel*). Además, como en casos anteriores, para identificar los elementos reales que se representan en el modelo de trazas, se almacena su identificador en un atributo *ref* (*TraceElement*).

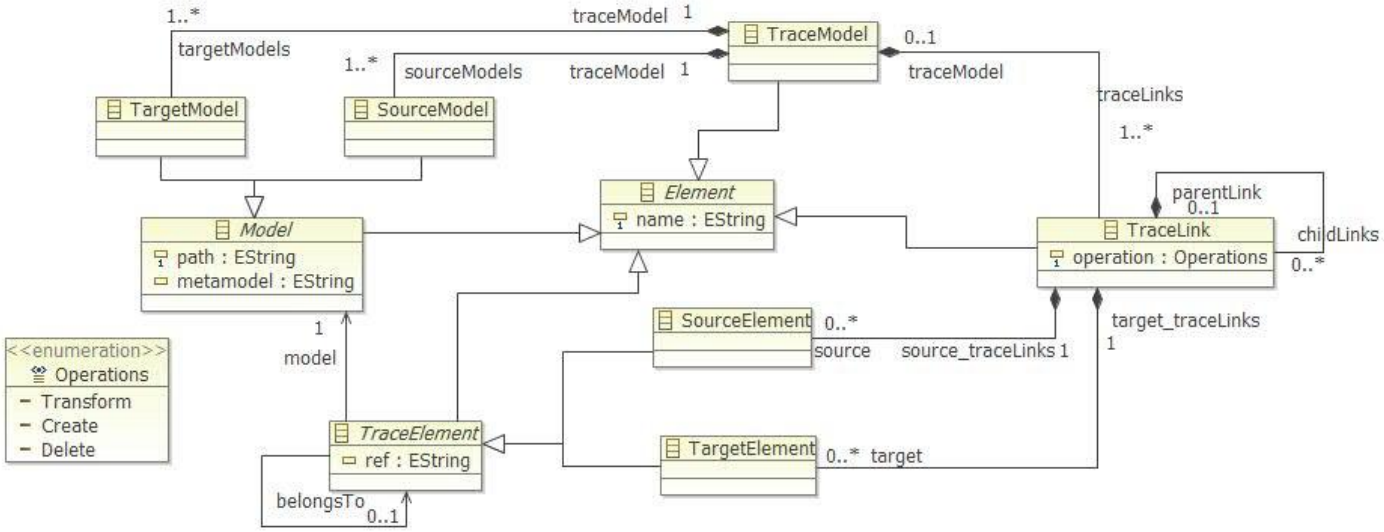

Figura 5-6. Metamodelo de Trazabilidad (Modelo *Ecore*)

5.3.2 *Implementación de Editores y Visualizadores de modelos*

El marco de trabajo EMF, a partir del modelo *Ecore* que representa el metamodelo permite generar por un lado, el código Java que implementa la lógica del DSL y por otro, el código Java que implementa un **editor en forma de árbol** (*tree-like*) para modelos conformes a dicho metamodelo. En otras palabras, a partir de la sintaxis abstracta del DSL, EMF permite generar la sintaxis concreta.

Para obtener este editor en forma de árbol, el primer paso consiste en la creación del modelo generador (*GenModel*) a partir del modelo base (*Ecore*). La mayor parte de la información necesaria para generar el editor *tree-like* se encuentra almacenada en el modelo *Ecore* (clases, atributos, referencias, cardinalidades, etc.). Sin embargo, algunos datos necesarios para alimentar a la plantillas JETs (*Java Emitter Templates*) que generan el código fuente del editor, como por ejemplo la ubicación del código o la forma de generarlo, es necesario definirlos de forma externa al modelo *Ecore*. Así, el modelo generador (*GenModel*) contiene referencias cruzadas con los elementos del modelo *Ecore* para conocer la sintaxis abstracta del DSL (como muestra la Figura 5-7) y toda la información requerida por las JETs para generar el editor en forma de árbol [40, 80, 188].

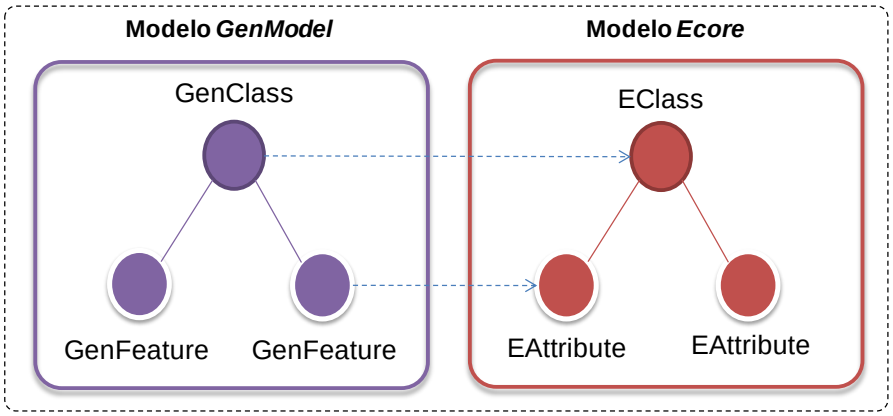

Figura 5-7. Relación entre los modelos *GenModel* y *Ecore*

La principal ventaja de la separación entre los modelos *Ecore* y *GenModel* es que el modelo *Ecore* permanece puro e independiente de cualquier información que sólo es relevante para la generación de código. En cambio, la desventaja principal es que el modelo generador puede desincronizarse si las referencias al modelo cambian. Para evitar este problema, las clases del modelo generador

incluyen métodos que propagan los cambios desde el modelo base al modelo generador. El uso de estos métodos asegura que los dos archivos se mantengan sincronizados automáticamente.

A partir del modelo *Genmodel*, EMF genera código Java estructurado en tres proyectos: el código del modelo (*model*), el código de edición (*edit*) y el código del editor (*editor*). Además, del mismo modo se puede generar código para los casos de pruebas (*test*).

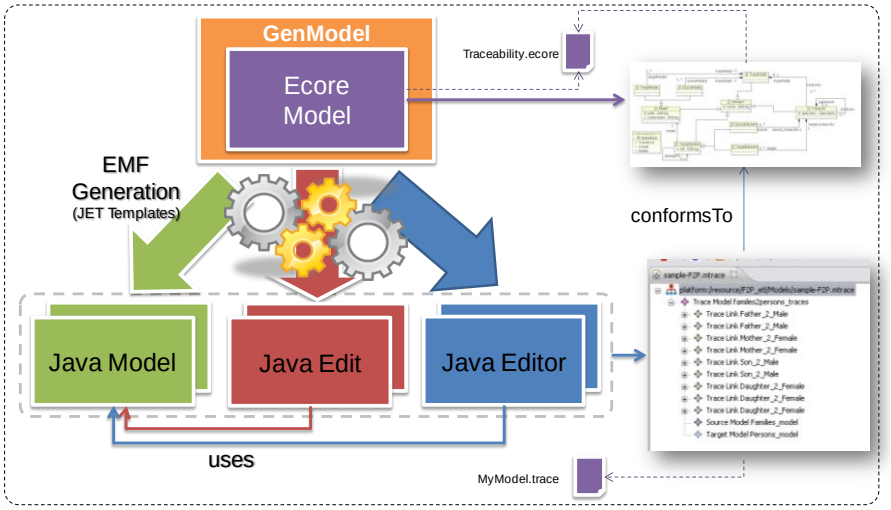

Figura 5-8. Vista general de la Generación de Editores EMF

Estos proyectos de código Java se encuentran relacionados entre sí, como muestra la Figura 5-8. Concretamente, el paquete de código *model* (contenido en el proyecto que contiene el metamodelo del DSL) que permite el acceso al metamodelo, crear modelos conformes a dicho metamodelo y la serialización de los mismos, es usado por los proyectos *edit* y *editor* que contienen las funcionalidades relacionadas con la interfaz de usuario. En otras palabras, el paquete *model* proporciona la lógica del DSL y los proyectos *edit* y *editor* contienen el código Java que implementa el editor en forma de árbol que proporciona EMF.

En la Figura 5-9 se muestra, a modo de ejemplo, el editor en forma de árbol generado con EMF a partir del modelo *Ecore* de trazabilidad (sección 5.3.1.4). Se trata de un editor completo, sencillo y funcional, sin embargo como se puede observar, no ofrece una visualización óptima para los modelos de trazas ya que no ofrece una representación adaptada a la naturaleza relacional de estos modelos.

Este inconveniente se produce no solo en el DSL de trazabilidad sino en todos los DSL que componen MeTAGeM-Trace.

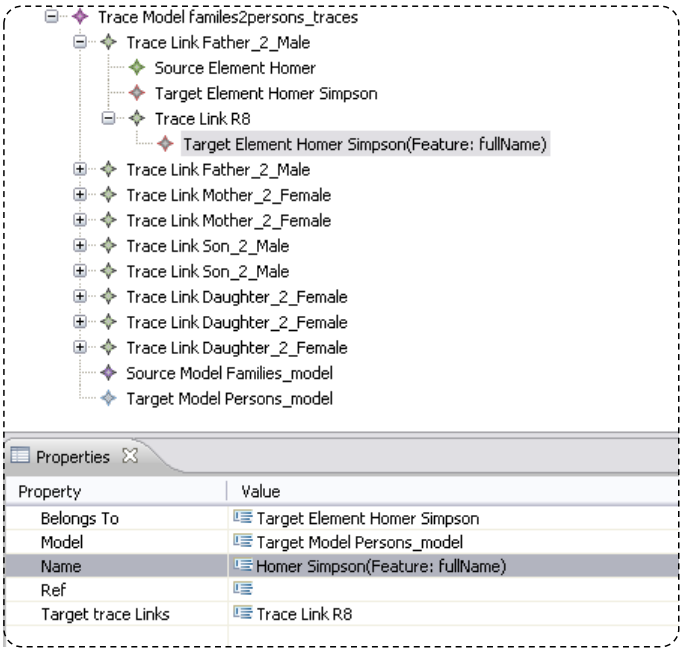

Figura 5-9. Ejemplo editor en forma del árbol (EMF)

Entre las limitaciones encontradas en el estado del arte presentado en el Capítulo 3, se identificó la necesidad de disponer de visualizadores específicos para representar los modelos de transformación y los modelos de trazas, teniendo en cuenta las características propias de estos tipos de modelos.

Entre las propuestas analizadas en el estado del arte, en diversos estudios ([17, 34, 54, 81, 122, 212]) se sugería o proponía el empleo de **editores multipanel** como los proporcionados por las herramientas AMW y ModeLink. Este tipo de editores permite mostrar al mismo tiempo varios modelos y un modelo que contiene las relaciones entre los anteriores. Sin embargo, ambos editores también presentan algunas limitaciones. El editor ModeLink que pertenece al proyecto Epsilon, tan solo permite la definición de dos o tres paneles, es decir solo es posible visualizar el modelo de relaciones y como máximo dos modelos relacionados. En cuanto al editor proporcionado por la herramienta AMW, destacan dos inconvenientes para su uso en el contexto de MeTAGeM-Trace. El primero de ellos ya se ha mencionado anteriormente y es, que por motivos de compatibilidad, se ha decidido no emplear AMW en la construcción de la herramienta MeTAGeM-Trace y el segundo se corresponde con

el editor en sí mismo. Dado que se trata de un editor de relaciones genérico, no establece la diferencia entre modelos origen y destino y por ello, los modelos relacionados no se organizan de forma adecuada en el editor. La organización de los modelos en el editor AMW es la siguiente: primer modelo relacionado, modelo de relaciones, resto de modelos relacionados. Así, es posible que si se tienen tres modelos origen y dos destino, la apariencia visual haga pensar al usuario que se trata de un modelo origen y cuatro destino (organización de los modelos en el editor: origen1, relaciones, origen2, origen3, destino1 y destino2).

Por ello, una de las principales contribuciones de esta tesis doctoral es el **desarrollo de editores/visualizadores específicos** para los modelos de transformación y modelos de trazas. A partir del estudio y observación de los editores multipanel proporcionados por las herramientas anteriores, se propone el uso de un editores/visualizadores que tengan una estructura similar a la mostrada en la Figura 5-10:

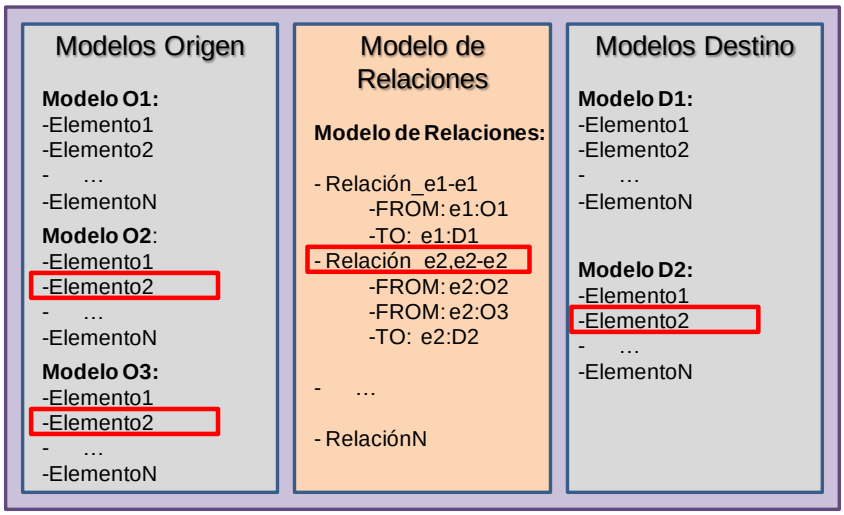

Figura 5-10. Esbozo de la estructura de los editores/visualizadores multipanel

Así, los editores/visualizadores que proporciona la herramienta MeTAGeM-Trace tienen las siguientes características:

- Contienen tres paneles para mostrar los modelos: modelos origen, modelo de relaciones y modelos destino. Si coexisten varios modelos origen o varios modelos destino, estos se organizarán verticalmente en el panel correspondiente.
- El usuario tiene la posibilidad de arrastrar objetos desde los modelos origen y destino al modelo de relaciones para facilitar la construcción del mismo.

- El usuario tiene la posibilidad de seleccionar una relación y automáticamente se identifican, en sus modelos correspondientes, los elementos implicados en dicha relación.
- El usuario tiene la posibilidad de seleccionar un elemento de los modelos origen o destino y el editor automáticamente identifica las relaciones en las que participa el elemento seleccionado.

A continuación, en la siguiente sub-sección, se detalla el proceso de implementación de los editores incluidos en la herramienta MeTAGeM-Trace. Este tipo de editores se emplean, de forma similar, para la visualización de los modelos a nivel PIM y PSM y para la visualización de los modelos de trazas generados a partir de la ejecución de la transformación desarrollada. Por ello, para facilitar la comprensión del lector, se detalla tan solo la implementación de uno de ellos. En concreto, se describen los pasos seguidos para la implementación del editor multipanel para la representación de los modelos de trazas.

5.3.2.1 Implementación del editor multipanel para el DSL de Trazabilidad

El primer paso en la construcción de un editor multipanel es llevar a cabo el desarrollo del editor en forma de árbol que proporciona EMF y que sirve como base para la implementación de este nuevo editor.

Para la construcción del editor en forma de árbol es necesario generar el modelo *GenModel* a partir del modelo *Ecore* y posteriormente, mediante el menú contextual del elemento raíz del modelo generador, seleccionar *Generate All* (esta opción genera al mismo tiempo los paquetes de código *model*, *edit*, *editor* y *test*).

Dado que se ha generado el modelo en forma de árbol, se ha decidido no prescindir de él, sino ofrecerlo como otra alternativa al editor multipanel. En otras palabras, MeTAGeM-Trace ofrece **dos editores** para el manejo y visualización de los modelos terminales conformes a los DSLs construidos: **un editor en forma de árbol y un editor multipanel**. Para ofrecer esta doble funcionalidad, en el fichero *plugin.xml* del proyecto *editor* se crea un nuevo editor en la pestaña *extensions*, como muestra la Figura 5-11. El editor *EMF Model Editor* es el editor por defecto que proporciona EMF al generar el editor en forma de árbol. El atributo *class* define la clase que implementa el editor. En este caso, se trata de la clase *TraceabilityEditor*. Para crear el nuevo editor de trazas se duplica dicha clase, que se encuentra en el proyecto de código *editor*, y se renombra a *TraceabilityEditorTrace*.

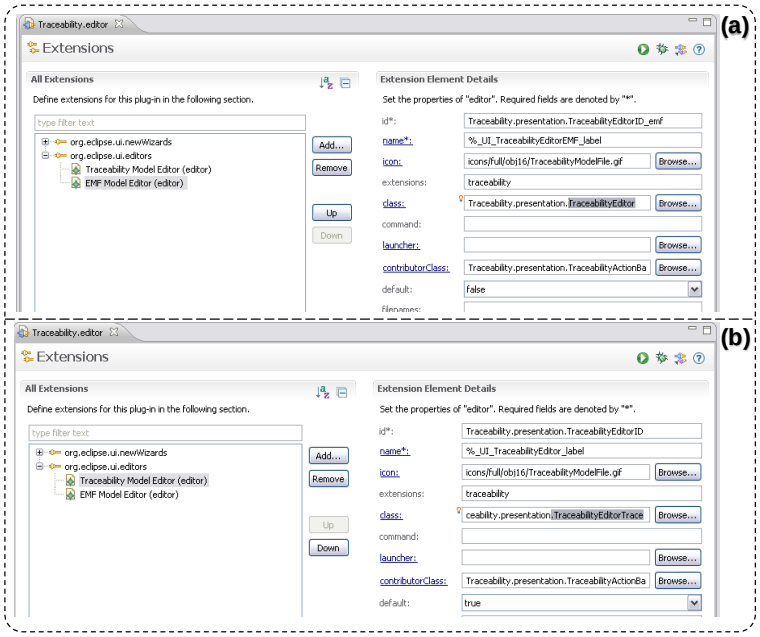

Figura 5-11. Creando extensiones tipo *editor*

En este punto, es posible definir el editor que se desea establecer por defecto. En este caso, se ha decidido que el **editor por defecto** de los modelos de trazas sea el editor multipanel. Para ello en el editor *EMF Model Editor*, el atributo *default* tomará valor *false* y en el editor *Traceability Model Editor* tomará valor *true*.

En este momento de la implementación se dispone de dos editores con distinto nombre pero exactamente la misma funcionalidad. Para dotar al editor multipanel de la funcionalidad deseada, se emplea el lenguaje de programación Java que es usado para implementar los editores EMF.

5.3.2.1.1 Modificando la información a mostrar en el editor

De acuerdo a las características descritas del editor multipanel, dado que se mostrarán los modelos origen y destino originales, no es necesario que en el modelo de relaciones se muestren las instancias de las metaclases que representan a los modelos origen y destino (*SourceModel* y *TargetModel*) ni que sea posible definir nuevos modelos. Para ello, en el proyecto *edit* se duplica la clase que representa al nodo raíz del metamodelo, *TraceModelItemProvider*, y se procede a realizar su renombrado (*TraceModelItemProviderTrace*). Estas clases se corresponden con los editores en forma de árbol y multipanel, respectivamente. En

la clase correspondiente al editor multipanel, se modifica el método *getChildrenFeatures* con el objetivo de eliminar o comentar las líneas de código que permiten la visualización de los elementos de tipo *SourceModel* y *TargetModel* (Figura 5-12). Es necesario destacar que todos los métodos modificados manualmente deben incluir la etiqueta "@NOT generated" para evitar que sean reescritos en posteriores generaciones de código a partir del modelo *GenModel*.

```
/**
* This specifies how to implement {@link #getChildren} and is used to deduce an
appropriate feature for an
* {@link org.eclipse.emf.edit.command.AddCommand}, {@link
org.eclipse.emf.edit.command.RemoveCommand} or
* {@link org.eclipse.emf.edit.command.MoveCommand} in {@link #createCommand}.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @NOT generated
*/
@Override
public Collection<? extends EStructuralFeature> getChildrenFeatures(Object object)
{
if (childrenFeatures == null) {
      super.getChildrenFeatures(object);
      childrenFeatures.add(TraceabilityPackage.Literals.TRACE_MODEL__TRACE_LINKS);
// To avoid showing models in traceability editor
//childrenFeatures.add(TraceabilityPackage.Literals.TRACE_MODEL__SOURCE_MODELS);
//childrenFeatures.add(TraceabilityPackage.Literals.TRACE_MODEL__TARGET_MODELS);
            }
            return childrenFeatures;
      }
```

Figura 5-12. Método *getChildrenFeatures* modificado

Dado que en el modelo de trazas no se representarán los modelos origen y destino, no parece tener sentido permitir la creación de estos elementos en el contexto de esta representación. Por ello, para eliminar esta funcionalidad se modifica el método *collectNewChildDescriptors*. Al igual que en el caso anterior, como muestra la Figura 5-13, se procede a eliminar o comentar las líneas de código correspondientes a la creación de los modelos origen y destino.

Una vez que se han modificado estos métodos, dado que anteriormente se ha duplicado la clase en la que se implementan, es necesario duplicar la clase que invoca a dichos métodos. Por ello, en el mismo proyecto *edit* se procede a duplicar la clase *TraceabilityItemProviderAdapterFactory* y esta nueva clase se renombra a *TraceabilityItemProviderAdapterFactoryTrace*. En esta nueva clase se modifican las llamadas a *TraceModelItemProvider* por *TraceModelItemProviderTrace.*

A continuación, en el proyecto *editor*, en la clase *TraceabilityEditorTrace*, creada anteriormente, se modifica el método *initializeEditingDomain*, sustituyendo "`new TraceabilityItemProviderAdapterFactory()`" por "`new TraceabilityItemProviderAdapterFactoryTrace()`".

```
protected void collectNewChildDescriptors(Collection<Object>
newChildDescriptors, Object object) {
super.collectNewChildDescriptors(newChildDescriptors, object);

newChildDescriptors.add(createChildParameter
      (TraceabilityPackage.Literals.TRACE_MODEL__TRACE_LINKS,
            TraceabilityFactory.eINSTANCE.createTraceLink()));

//To hide model creation in traceability editor:
//newChildDescriptors.add(createChildParameter
//(TraceabilityPackage.Literals.TRACE_MODEL__SOURCE_MODELS,
//TraceabilityFactory.eINSTANCE.createInModel()));

//newChildDescriptors.add(createChildParameter
//(TraceabilityPackage.Literals.TRACE_MODEL__TARGET_MODELS,
//TraceabilityFactory.eINSTANCE.createOutModel()));
      }
}
```

Figura 5-13. Método *collectNewChildDescriptors* modificado

En este punto, se dispone de dos editores que muestran diferente información pero que mantienen la estructura del editor en forma de árbol.

5.3.2.1.2 Estructura de los paneles

El siguiente paso consiste en modificar el código que implementa el nuevo editor (proyecto *editor*) para que tenga una apariencia similar a la estructura mostrada anteriormente en la Figura 5-10. En primer lugar, se ha creado una clase *Actions* que contiene un conjunto de métodos estáticos para llevar a cabo tareas comunes como obtener el nombre del modelo de trazas, obtener los modelos origen y los modelos destino y registrar un metamodelo cualquiera, etc. El código de dichos métodos se encuentra en el CD que acompaña a esta tesis doctoral.

A continuación, se procede a modificar la clase *TraceabilityEditorTrace* que contiene la apariencia y funcionalidad del editor multipanel de trazas. En primer lugar, se añaden a dicha clase los atributos mostrados en la Figura 5-14:

```
protected ResourceSet sourceRs;
protected ResourceSet targetRs;
protected TreeViewer sourceViewer;
protected TreeViewer traceabilityViewer;
protected TreeViewer targetViewer;
```

Figura 5-14. Atributos añadidos a la clase *TraceabilityEditorTrace*

Posteriormente, se procede a modificar los siguientes métodos de dicha clase: *createPages*, *createContextMenuFor*, *handleContentOutlineSelection*,

handleContentTargetSelection, handleContentSourceSelection y handleContentTargetSelection.

El método *createPages* contiene el código que implementa la apariencia estética del editor. Sobre este método se llevan a cabo las siguientes tareas:

1. En primer lugar, tras la llamada que invoca al método *createModel*, se añaden las siguientes líneas de código:

```
createModel();

Composite container = getContainer();
final SashForm topSashForm = new SashForm(container,SWT.HORIZONTAL);
```

Figura 5-15. Estableciendo la forma del contenedor del editor multipanel.

```
// Only creates the other pages if there is something that can be edited

if (!getEditingDomain().getResourceSet().getResources().isEmpty()) {
ArrayList<SourceModelImpl> sources =
Actions.getSourceModels(getEditingDomain().getResourceSet());
ArrayList<TargetModelImpl> targets =
Actions.getTargetModels(getEditingDomain().getResourceSet());
```

Figura 5-16. Cargando los modelos de origen y destino en el editor multipanel

2. El siguiente paso consiste en eliminar los visores (*viewers*) existentes en el método y crear los visores para los modelos origen y destino. La Figura 5-18 muestra el código que implementa el visor para los modelos origen. En el caso del visor destino, la implementación es muy similar y puede consultarse en el código fuente de la herramienta MeTAGeM-Trace.
3. Por último, se crea el visor para el modelo de trazas. La implementación es similar a los anteriores, pero contiene cambios que merecen ser mostrados. Esta implementación se muestra en la Figura 5-17.

```
viewerPane2.createControl(topSashForm);
traceabilityViewer = (TreeViewer) viewerPane2.getViewer();
traceabilityViewer.addDragSupport(DND.DROP_COPY | DND.DROP_LINK, transfers, new
ViewerDragAdapter(traceabilityViewer));
traceabilityViewer.setContentProvider(
      new AdapterFactoryContentProvider(adapterFactory));
traceabilityViewer.setLabelProvider(
      new AdapterFactoryLabelProvider(adapterFactory));
traceabilityViewer.setInput(editingDomain.getResourceSet());
traceabilityViewer.setSelection(
      new StructuredSelection(editingDomain.getResourceSet().getResources().get(0)), true);
viewerPane2.setTitle("Traceability Model",Actions.getImage("TraceModel"));
traceabilityViewer.addSelectionChangedListener(
      new ISelectionChangedListener() {
            public void selectionChanged(SelectionChangedEvent event) {
                  handleContentTraceabilitySelection(event.getSelection());
            }
      });
new AdapterFactoryTreeEditor(traceabilityViewer.getTree(),adapterFactory);
createContextMenuFor(traceabilityViewer);
```

Figura 5-17. Código que implementa el visor del modelo de Trazas

```
// Create view for sources models.
ViewerPane viewerPane = new
ViewerPane(getSite().getPage(),TraceabilityEditorTrace2.this)
{
 @Override
 public Viewer createViewer(Composite composite) {
      Tree tree = new Tree(composite, SWT.MULTI);
      TreeViewer newTreeViewer = new TreeViewer(tree);
      return newTreeViewer;
 }
 @Override
 public void requestActivation() {
      super.requestActivation();
      setCurrentViewerPane(this);
 }
};
viewerPane.createControl(topSashForm);
sourceViewer=(TreeViewer)viewerPane.getViewer();
sourceViewer.setContentProvider(new AdapterFactoryContentProvider(adapterFactory));
sourceViewer.setLabelProvider(new AdapterFactoryLabelProvider(adapterFactory));
Transfer[] transfers = new Transfer[] { LocalTransfer.getInstance() };
sourceViewer.addDragSupport(DND.DROP_LINK , transfers, new
ViewerDragAdapter(sourceViewer));
for(int i=0;i<sources.size();i++){
      if (sources.get(i).getMetamodel() != null) {
      //If metamodel attribute is not null
            if (!sources.get(i).getMetamodel().equals("")) {
            //If metamodel is not empty
            String metamodelRegistered = Actions.registerMetamodel(
                  sources.get(i).getMetamodel(), sources.get(i).getName());
            sources.get(i).setMetamodel(metamodelRegistered);
            }
      }
}
sourceRs= Actions.createResourceSet_Sources(sources);
Actions.setSourceModels(getEditingDomain().getResourceSet(),sources);
sourceViewer.setInput(sourceRs);
sourceViewer.setSelection(new StructuredSelection(sourceRs.getResources().get(0)),
true);

viewerPane.setTitle("Source Models");
sourceViewer.addSelectionChangedListener
(new ISelectionChangedListener() {
 // This ensures that we handle selections correctly.
 public void selectionChanged(SelectionChangedEvent event) {
       handleContentSourceSelection(event.getSelection());
 }
});
new AdapterFactoryTreeEditor(sourceViewer.getTree(), adapterFactory);
//To show the contextual menu (disabled for input and output models)
//createContextMenuFor(inputViewer.get(i));
```

Figura 5-18. Código que implementa el visor de los modelos origen

El método *createContextMenuFor* es el encargado de crear el menú contextual de los visores y de registrar las operaciones permitidas en dicho visor. Dado que para cada uno de los visores, se ha definido en su implementación las acciones permitidas, es necesario suprimir en este método las líneas de código mostradas en la Figura 5-19.

```
int dndOperations = DND.DROP_COPY | DND.DROP_MOVE | DND.DROP_LINK;
Transfer[] transfers = new Transfer[] { LocalTransfer.getInstance() };
viewer.addDragSupport(dndOperations, transfers, new
ViewerDragAdapter(viewer));
viewer.addDropSupport(dndOperations, transfers, new
EditingDomainViewerDropAdapter(editingDomain, viewer));
```

Figura 5-19. Líneas a suprimir en el método *createContextMenuFor*

En este punto, el editor multipanel ya tiene la estructura deseada, es decir, tres paneles: modelos origen, modelo de trazas y modelos destino. El siguiente paso es interconectar dichos paneles para permitir que el usuario seleccione una traza y automáticamente se identifiquen los elementos de dicha traza en sus modelos originales y para que al seleccionar un elemento de los modelos, se identifiquen todas las trazas en las que participa el elemento seleccionado.

5.3.2.1.3 Identificación automática de los elementos seleccionados

El método *handleContentOutlineSelection* se encarga de mostrar en el panel del modelo de trazas el elemento seleccionado en la ventana *Outline* del entorno Eclipse. Para ofrecer la funcionalidad mencionada, es necesario eliminar o comentar las líneas de código que aparecen en negrita en la Figura 5-20.

```
public void handleContentOutlineSelection(ISelection selection) {
    if (currentViewerPane != null && !selection.isEmpty() && selection instanceof
IStructuredSelection) {
        Iterator<?> selectedElements = ((IStructuredSelection)selection).iterator();
        if (selectedElements.hasNext()) {
            // Get the first selected element.
            Object selectedElement = selectedElements.next();
            // If it's the selection viewer, then we want it to select the same
selection as this selection.
            //if (currentViewerPane.getViewer() == selectionViewer) {
                ArrayList<Object> selectionList = new ArrayList<Object>();
                selectionList.add(selectedElement);
                while (selectedElements.hasNext()) {
                    selectionList.add(selectedElements.next());
                }
                // Set the selection to the widget.
                selectionViewer.setSelection(new StructuredSelection(selectionList));
            //}
            //else {
                // Set the input to the widget.
                //
            //  if (currentViewerPane.getViewer().getInput() != selectedElement) {
            //      currentViewerPane.getViewer().setInput(selectedElement);
            //      currentViewerPane.setTitle(selectedElement);
            //  }
            //}
        }
    }
}
```

Figura 5-20. Método *handleContentOutlineSelection*

Así mismo, también es necesario modificar la línea de código:

```
selectionViewer.setSelection(new StructuredSelection(selectionList));
```

por la línea:

```
traceabilityViewer.setSelection(new StructuredSelection(selectionList));
```

Siguiendo el método anterior, se implementan los métodos se encargan de establecer las relaciones entre las trazas y los elementos que forman parte de ellas. Se crea un método *handleContentTraceabilitySelection* que se ejecuta si el usuario selecciona una traza. Por cada elemento de tipo *TraceLink* seleccionado, se buscan entre todos los modelos origen y destino, todos los elementos origen y destino incluidos en las trazas. A continuación, dichos elementos son seleccionados en sus correspondientes modelos. Por cada elemento de tipo *SourceElement* seleccionado, se busca su identificador en los modelos origen y si es encontrado, se selecciona en su modelo correspondiente. Para los elementos de tipo *TargetElement*, realiza la misma función pero buscando entre los modelos destino.

Cuando se selecciona un elemento de un modelo origen o destino, los métodos *handleContentSourceSelection* y *handleContentTargetSelection* se encargan de obtener el identificador del elemento para buscar coincidencias entre los elementos de las trazas y de este modo, identificar aquellas en las que participa. El código que implementa estos métodos puede encontrarse en el CD adjunto a esta tesis doctoral.

En este momento el editor multipanel de trazabilidad ya muestra el modelo de trazabilidad, a su izquierda los modelos origen y a su derecha los modelos destino. Además, seleccionando un elemento (elemento origen o elemento destino) nos muestra todas las trazas en las que se encuentra implicado. En el caso de seleccionar una traza se muestran todos los elementos que forman parte de ella.

5.3.2.1.4 Funcionalidad Drag&Drop

En último lugar, solo queda implementar la funcionalidad, conocida como ***drag&drop*** [59], que permite al usuario arrastrar elementos de los modelos origen y destino para establecerlos automáticamente como elementos origen o destino (según corresponda) de la traza sobre la que se arrastran. Para ofrecer esta funcionalidad se crea una nueva clase (*TraceabilityDragDrop*) en el proyecto *editor*. Esta nueva clase debe extender de *EditingDomainViewerDropAdapter*. Su constructor recibe el dominio, el visor, el conjunto de recursos origen (modelos origen) y el conjunto de recursos destino (modelos destino). Esta clase contiene los siguientes métodos:

- *helper*: Recibe el evento drag&drop y selecciona el elemento que ha sido arrastrado (source) y el elemento sobre el cual ha sido arrastrado (target). A partir del source, determina si el evento es sobre un elemento de un modelo de origen o sobre un elemento de un modelo destino (atributo *selectionType*).

- *drop:* Recibe el evento *drop* y a partir del atributo *selectionType* (método *helper*), evalúa si la acción se debe llevar a cabo `(selectionType == SOURCE_ELEMENT)||(selectionType == TARGET_ELEMENT)` o si por el contrario, se debe descartar.
- *createTraceElement*: Recibe el elemento arrastrado, la traza que lo recibe y el tipo de elemento arrastrado (Source o Target). Comprueba si el elemento arrastrado sobre la traza es de tipo *source* o *target* y dependiendo de esta información realiza una llamada al método *handleSetSourceElement* o *handleSetTargetElement*, respectivamente.
- *handleSetSourceElement*: Añade el elemento arrastrado a la traza como elemento de origen.
- *handleSetTargetElement*: Igual que el método anterior, pero en este caso para elementos de tipo destino.

Una vez que se ha creado la clase *TraceabilityDragDrop* es necesario instanciarla desde el método *createPages* de la clase *TraceabilityEditorTrace* del proyecto *editor*, justo después del código que implementa el visor del panel de trazabilidad. La instanciación de esta clase se muestra en la Figura 5-21.

```
TraceabilityDragDrop dragdrop=new TraceabilityDragDrop(
    getEditingDomain(),traceabilityViewer, this.sourceRs, this.targetRs);
traceabilityViewer.addDropSupport(
    DND.DROP_COPY | DND.DROP_LINK, transfers, dragdrop);
```

Figura 5-21. Instanciando la clase *TraceabilityDragDrop*

Después de haber seguido todos los pasos anteriores, el editor multipanel para el DSL de trazabilidad (mostrado en la Figura 5-22) ya tiene la estructura y funcionalidad requerida.

5.3.2.2 Personalización de los editores

Con el objetivo de mejorar la experiencia del usuario a la hora de usar estos editores, se ha llevado a cabo un conjunto de tareas para mejorar aspectos concretos de los diferentes editores. En esta sección se presentan dichas mejoras.

5.3.2.2.1 Personalización de Iconos

Los editores creados con EMF (editor en forma de árbol y editor multipanel) representan los elementos en los modelos mediante iconos genéricos. Así, una de las mejoras realizadas sobre los editores de MeTAGeM-Trace es personalizar dichos iconos con imágenes representativas de cada elemento. Para

ello, en primer lugar se ha seleccionado para cada elemento una imagen adecuada y luego se ha procedido a cambiar la imagen original por la seleccionada.

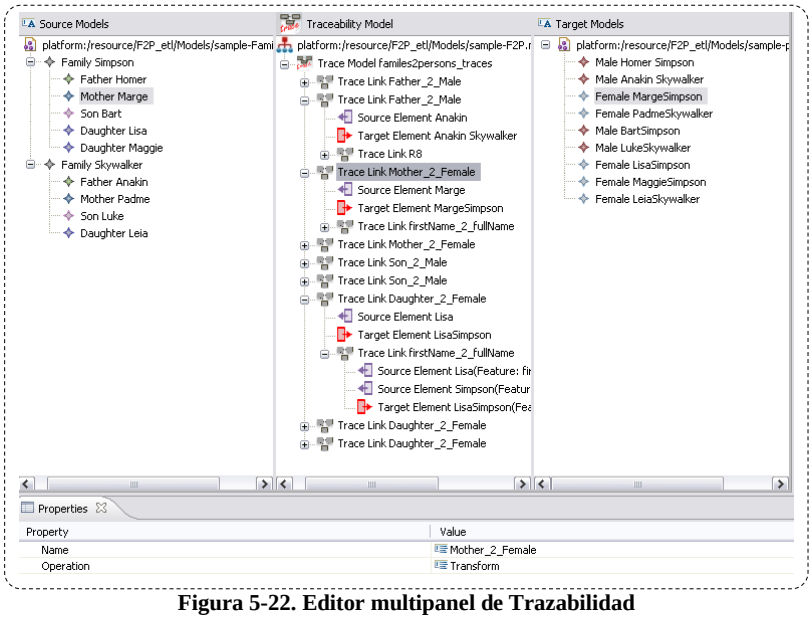

Figura 5-22. Editor multipanel de Trazabilidad

A modo de ejemplo, para mostrar la personalización de los iconos, la Figura 5-23(a) presenta un modelo conforme al metamodelo EMF antes de incluir los iconos personalizados y la Figura 5-23(b) muestra el mismo modelo, una vez incluidos. De igual forma, se han personalizado los iconos del resto de editores.

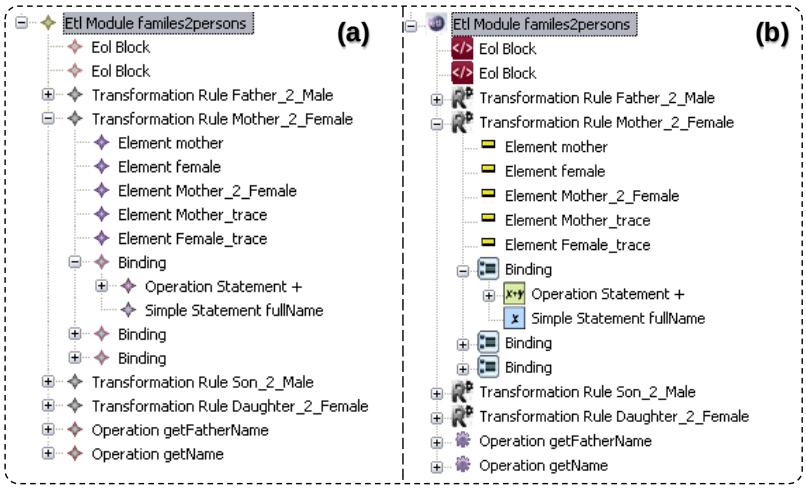

Figura 5-23. Modelo ETL: (a) iconos genéricos; (b) iconos personalizados

5.3.2.2.2 Actualización de menús despegables en tiempo de ejecución

En los metamodelos de los DSLs que componen la herramienta MeTAGeM-Trace, existen algunas referencias con cardinalidad 0..1. Dichas referencias resultan, en tiempo de ejecución, en menús despegables en los que el usuario puede elegir una opción entre todas las instancias del elemento al que se puede hacer referencia. Sin embargo en muchas ocasiones, debido a la semántica del metamodelo, no es correcto relacionarse con cualquiera de las instancias de ese tipo, sino que la relación solo tiene sentido con algunas de ellas.

A modo de ejemplo, en el metamodelo MeTAGeM (nivel PIM), la metaclase *RelationElement*, con el objetivo de representar que un elemento de una relación de transformación (generalmente, un atributo) es interno a otro, contiene una referencia de este tipo (*belongsTo*) cuyo destino es la misma metaclase. Sin embargo, esta referencia solo tiene sentido cuando ambos elementos forman parte de la misma relación de transformación. En la Figura 5-24, se muestra una relación de transformación entre un elemento padre (*Father*) y un elemento hombre (*male*) que además contiene una subrelación en la que se crea el nombre (*fullname*) del elemento destino. En tal caso, no tiene sentido poder indicar que el atributo *fullname* destino pertenece a cualquier elemento (Figura 5-24a) sino que como muestra la Figura 5-24(b) solo tiene cabida la relación con el elemento *Male* de destino.

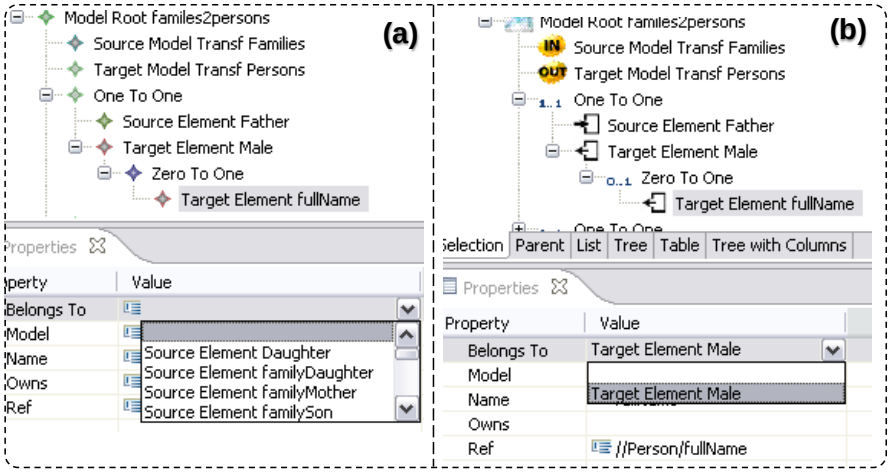

Figura 5-24. Modelo MeTAGeM: (a) todas las instancias disponibles; (b) valores actualizados en tiempo de ejecución

Para ofrecer esta funcionalidad en tiempo de ejecución, se debe modificar la clase *XXXItemProvider* del proyecto *edit*, siendo XXX el nombre de la

metaclase que contiene la relación (en este caso, *RelationElement*). En dicha clase Java, se debe modificar el método *addYYYPropertyDescriptor*, siendo YYY el nombre de la referencia (en este caso, *belongsTo*). Por defecto, dicho método incluirá en los posibles valores del menú desplegable todas aquellas instancias del tipo destino de la referencia.

Para mostrar los valores deseados, dicho método debe implementar el código mostrado en la Figura 5-25, donde en el método *getChoiceOfValue* se debe devolver la colección de los elementos que aparecerán en la lista de opciones del menú.

```
	/**
	 * This adds a property descriptor for the Belongs To feature.
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @NOT generated
	 */
	protected void addBelongsToPropertyDescriptor(Object object) {
		itemPropertyDescriptors.add
			(new ItemPropertyDescriptor
				(((ComposeableAdapterFactory)adapterFactory).getRootAdapterFactory(),
				 getResourceLocator(),
				 getString("_UI_RelationElement_belongsTo_feature"),
				 getString("_UI_PropertyDescriptor_description",
"_UI_RelationElement_belongsTo_feature", "_UI_RelationElement_type"),
				 MetagemPackage.Literals.RELATION_ELEMENT__BELONGS_TO,
				 true,
				 false,
				 true,
				 null,
				 null,
				 null)
				 {
		    @Override
		    public Collection<?> getChoiceOfValues(Object object)
		    {
		        Collection<Object> result = new ArrayList<Object>();
		        result.add(null);

		      //Se seleccionan los elementos adecuados y se insertan en la
		      //collection result
		      [...]

		      return result;
		    }

		    @Override
		    public void resetPropertyValue(Object object)
		    {
		      setPropertyValue(object, null);
		    }
		      });
	}
```

Figura 5-25. Código Java para actualización automática de los menús desplegables de las referencias

Esta funcionalidad de actualización automática en tiempo de ejecución de los valores de los menús desplegables ha sido implementada en todas aquellas referencias de todos los DSLs de MeTAGeM-Trace cuyas características coinciden con las mencionadas en esta sección.

5.3.2.2.3 Establecimiento automático de relaciones

En el modelado de la transformación y de las relaciones de trazabilidad a nivel PSM (modelo de aproximación híbrida), el usuario tiene la posibilidad de crear o modificar las reglas de la transformación y las relaciones de trazabilidad. Como se muestra en la sección 4.2.3, ambos tipos de elementos se encuentran muy relacionados: una regla de transformación puede contener a una relación de trazabilidad (*TraceRule*), en consecuencia, los elementos origen y destino de dicha relación de trazabilidad se corresponden con los elementos origen y destino de la regla de transformación. Este hecho también tiene lugar entre los elementos *Binding* y las relaciones de trazabilidad de tipo *TraceBinding*.

Para facilitar al usuario la construcción del modelo de aproximación híbrida, cada vez que se genera una relación de trazabilidad contenida en una regla de transformación, automáticamente se asocian los elementos origen y destino de la regla a la relación de trazabilidad.

Para implementar esta funcionalidad, se han modificado las clases Java correspondientes a los elementos *Rule* y *Binding* del paquete *hybrid.impl* (*BindingImpl* y *RuleImpl*, respectivamente), del proyecto *Hybrid*. En dichas clases se han modificado los métodos *getLeft* y *getRight* en el caso de *BindingImpl* y *getSources* y *getTargets* en el caso de *RuleImpl*, de forma que, como muestra la Figura 5-26, cada vez que se invoquen estos métodos se actualicen los elementos origen y destino de las relaciones de trazabilidad contenidas en la regla o relacionadas con el *binding*. Es necesario destacar que dichos métodos se invocan cada vez que se carga el modelo o se selecciona un elemento de este tipo, por tanto los elementos de la relación de trazabilidad se actualizan continuamente.

```
/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @NOT generated
*/
public EList<Source> getSources() {
      if (sources == null) {
          sources = new EObjectContainmentWithInverseEList<Source>(
            Source.class, this, HybridPackage.RULE__SOURCES, HybridPackage.SOURCE__RULE);
      }
      //Update TraceRule Sources
      TraceRule traceRule = this.getTrace();
      if(traceRule!=null){
            for(int i=0;i<sources.size();i++){
                  if(!traceRule.getSource().contains(sources.get(i))){
                        traceRule.getSource().add(sources.get(i));
                  }
            }

      }
      …
}
```

Figura 5-26. Código que implementa la actualización automática de los elementos de las relaciones de trazabilidad

5.3.3 *Implementación de Asistentes de Creación de modelos*

Para la creación de nuevos modelos desde cero, EMF proporciona un asistente de creación por defecto. En dicho asistente, el usuario debe elegir la ubicación física y el nombre del fichero que contendrá el modelo y cuál será el elemento raíz de dicho modelo (generalmente debe coincidir con el elemento raíz del metamodelo al que es conforme el nuevo modelo).

Sin embargo, estos asistentes de creación son demasiado básicos y genéricos. Además, dado que en la implementación de los editores multipanel eliminamos la visualización y creación de los elementos que representan a los modelos de origen y destino, es deseable especificar qué metamodelos intervienen como origen o destino de la transformación a modelar o también qué modelos, en el caso de crear un nuevo modelo de trazas.

Por todo ello, en el contexto de la implementación de la herramienta MeTAGeM-Trace se ha construido un conjunto de asistentes de creación de modelos. En concreto, se han creado asistentes de creación para los DSLs que modelan la transformación a nivel PIM y PSM y para el DSL de trazabilidad.

En la Figura 5-27 se muestra parte del asistente para la creación nuevos modelos de trazas. En la siguiente sub-sección, se presenta la implementación del asistente de creación de modelos de trazas.

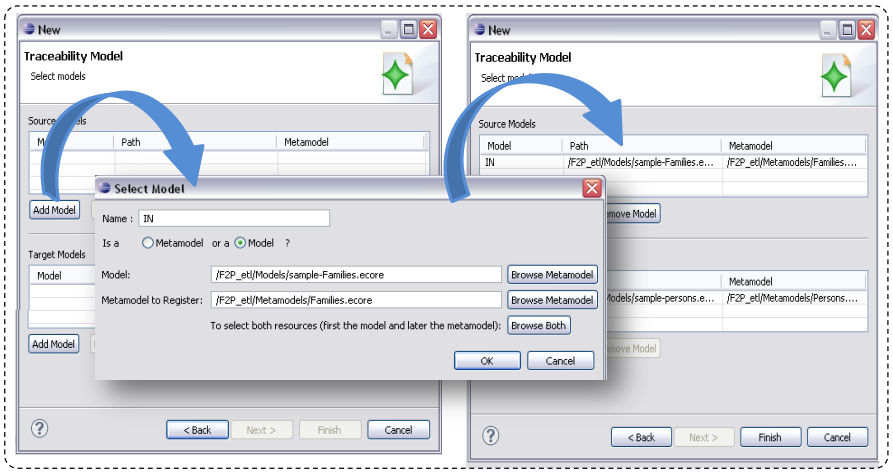

Figura 5-27. Asistente de creación de modelos de Trazas

Todos los asistentes de creación han sido implementados de forma similar, con pequeñas diferencias derivadas de los metamodelos, por lo que tan solo se

muestra la implementación de uno de ellos. Sin embargo, se recuerda que todo el código fuente de la herramienta se encuentra en el CD adjunto.

5.3.3.1 Implementación del Asistente de creación de modelos de Trazas

Como ya se ha comentado anteriormente, EMF proporciona por defecto un asistente (*wizard*) para la creación de nuevos modelos de los DSLs construidos. Sin embargo el asistente que proporciona, en este caso no cumple con necesidades de los DSLs de MeTAGeM-Trace. Por ello, en esta sección se muestra cómo modificar dicho asistente para adaptarlo a las características deseadas.

5.3.3.1.1 Modificando la extensión org.eclipse.ui.newWizards

Para la definición del asistente de creación por defecto, EMF ha definido una extensión de tipo *org.eclipse.ui.newWizards* en el fichero *plugin.xml* ubicado en el proyecto *editor*. En dicha extensión, como muestra la Figura 5-28, se pueden definir características del asistente de creación como el identificador, el nombre, el icono, la clase que lo implementa o la categoría en la que se insertará, entre otros.

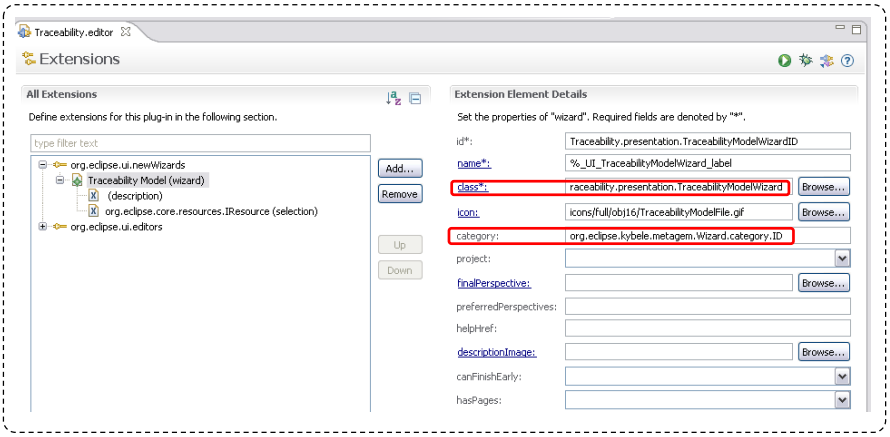

Figura 5-28. Extensión *org.eclipse.ui.newWizards*

En la Figura 5-28 se han resaltado dos parámetros de la extensión: *class* y *category*. El primero de ellos corresponde a la clase Java que implementa el asistente y que por tanto, se debe modificar para obtener el asistente deseado, como se detalla más adelante. El parámetro *category* permite especificar la categoría en la que se alojará nuestro asistente de creación. En este caso, hemos empleado una categoría (*org.eclipse.kybele.metagem.Wizard.category.ID*) que ha sido definida en el contexto del DSL de MeTAGeM con el objetivo de agrupar a todos los asistentes de creación de la herramienta. De este modo, el asistente de

creación para los nuevos modelos de trazabilidad podrá ser invocado desde la plataforma Eclipse como muestra la Figura 5-29.

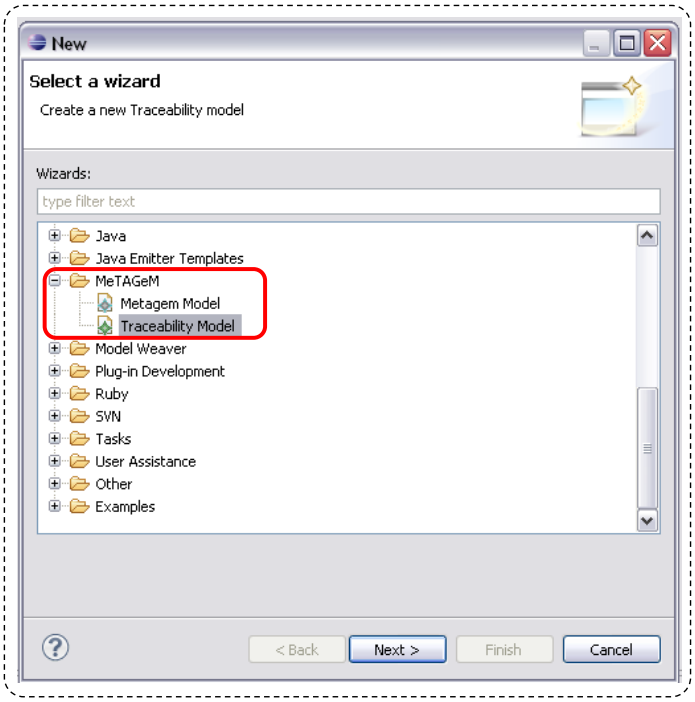

Figura 5-29. Categoría MeTAGeM para agrupar a los asistentes de creación

5.3.3.1.2 Modificando el código Java que implementa el asistente de creación

La clase Java *TraceabilityModelWizard* es la encargada de implementar la apariencia y funcionalidad del asistente de creación propuesto por EMF. A continuación se muestran los cambios realizados sobre dicha clase para obtener el asistente deseado.

En primer lugar, se procede a modificar algunos atributos de la clase. Así, se elimina el atributo *initialObjectCreationPage*, ya que no será necesario disponer de una página para la elección del elemento raíza del modelo porque se definirá de forma automática. Por otro lado, el atributo *newFileCreationPage* es renombrado a *modelsCreationPage* con el objetivo de disponer de un nombre más representativo. También se han creado los atributos *sourceModels* y *targetModels*. Estos atributos son listas que contienen elementos *ModelData*, una clase creada específicamente para almacenar el nombre, la ruta, el metamodelo de cada modelo así como un atributo que permite conocer si dicho modelo es un metamodelo. Más adelante se presenta la implementación de esta clase.

El siguiente paso consiste en modificar los siguientes métodos de la clase *TraceabilityModelWizard*: *init, createInitialModel y addPages.*

En el método *init* tan solo se añaden las sentencias de inicialización de los atributos *sourceModels* y *targetModels* que muestra la Figura 5-30.

```
sourceModels = new ArrayList<ModelData>();
targetModels = new ArrayList<ModelData>();
```

Figura 5-30. Inicializando los atributos *sourceModels* y *targetModels*

El método *createInitialModel* es el encargado de obtener la raíz del modelo a crear. En este caso, para aislar al usuario de llevar a cabo la elección del elemento de forma manual, como muestra la Figura 5-31 se selecciona automáticamente el elemento *TraceModel* como raíz de los nuevos modelos conformes al metamodelo de trazabilidad.

```
protected EObject createInitialModel() {
      EClass eClass = (EClass)traceabilityPackage.getEClassifier("TraceModel");
      EObject rootObject = traceabilityFactory.create(eClass);
      return rootObject;
}
```

Figura 5-31. Método *createInitialModel* modificado

Por último el método *addPages* tiene por objetivo crear y añadir cada una de las páginas del asistente, entendiendo por "páginas" cada una de las pantallas que forman el asistente. En este caso, se ha creado un asistente con dos páginas: una seleccionar el nombre del nuevo modelo y su ubicación y otra para seleccionar los modelos de origen y destino que intervienen en el modelo de trazas que se va a construir. Esta segunda pantalla se corresponde con la mostrada anteriormente en la Figura 5-27.

La primera página es proporcionada por el asistente por defecto y dado que se están realizando las modificaciones directamente sobre él, no es necesario modificar el código que añade esta página. Para añadir la segunda, como muestra la Figura 5-32, es necesario comentar las líneas correspondientes a la página de elección de la raíz del modelo (proporcionada por el asistente por defecto) y añadir el código correspondiente a la nueva pantalla. Para ello, se crea una instancia de la clase *TraceabilityModelWizardModelsCreationPage*, que contiene la implementación de la nueva página, definir el texto del título y descripción de la página y por último añadirla al conjunto de páginas.

```
//initialObjectCreationPage = new
//TraceabilityModelWizardInitialObjectCreationPage("Source Models");
//initialObjectCreationPage.setTitle(
//TraceabilityEditorPlugin.INSTANCE.getString("_UI_TraceabilityModelWizard_label"));
//initialObjectCreationPage.setDescription(
//TraceabilityEditorPlugin.INSTANCE.getString("_UI_Wizard_initial_object_description"));
//addPage(initialObjectCreationPage);

modelsCreationPage = new TraceabilityModelWizardModelsCreationPage(
                     "Models",sourceModels,targetModels);
modelsCreationPage.setTitle(TraceabilityEditorPlugin.INSTANCE.getString(
                     "_UI_TraceabilityModelWizard_label"));
modelsCreationPage.setDescription(TraceabilityEditorPlugin.INSTANCE.getString(
                     "_UI_Wizard_models_description"));
addPage(modelsCreationPage);
```

Figura 5-32. Método *addPages* modificado

5.3.3.1.3 Clase TraceabilityModelWizardModelsCreationPage

Como se ha indicado anteriormente, la implementación de la nueva página que se construye en el asistente para especificar los modelos origen y destino se incluye en la clase *TraceabilityModelWizardModelsCreationPage*. Esta clase se ha implementado en el mismo fichero *TraceabilityModelWizard.java*. El objetivo buscado es disponer de una página del asistente cuya apariencia sea la mostrada por la Figura 5-33.

Figura 5-33. Pantalla de selección de modelos (Asistente de creación de modelos de Trazas)

La página de selección de modelos se compone de dos partes principales: una para los modelos de origen y otra para los modelos destino. Cada una de ellas se compone de una tabla y de tres botones (*add*, *edit*, *remove*). Para crear y definir estos elementos, se define un conjunto de atributos:

```
public abstract class TraceabilityModelWizardModelsCreationPage
extends WizardPage {

      private Button addSourceModel;
      private Button editSourceModel;
      private Button removeSourceModel;
      private Button addTargetModel;
      private Button editTargetModel;
      private Button removeTargetModel;
      private Table modelSourceTable;
      private Table modelTargetTable;
      private ArrayList<ModelData> sourceModels;
      private ArrayList<ModelData> targetModels;
```

Figura 5-34. Atributos de *TraceabilityModelWizardModelsCreationPage*

Esta clase se compone básicamente de un método llamado *createControl* que se encarga de definir los elementos que forman parte de la página del asistente así como su comportamiento. Debido al tamaño de este método (más de 200 líneas de código) se invita al lector a consultarlo en el CD adjunto a esta tesis doctoral. Este método permite al usuario crear, modificar y eliminar los modelos que participan en el modelo de trazas. Para ello, hace uso del manejador *TraceabilityWizardHandleModel*, que se explica a continuación.

5.3.3.1.4 Clase TraceabilityWizardHandleModel

Esta clase ha sido implementada con el objetivo de ofrecer un manejador visual para establecer y modificar los modelos de origen y destino que intervienen en el modelo de relaciones, en este caso, el modelo de trazas.

Para ello, su implementación consiste, básicamente, en la definición de un conjunto de elementos visuales (cuadros de texto, botones, etiquetas, etc.). La Figura 5-35 muestra la ventana ofrecida al usuario para llevar a cabo estas tareas.

Una vez que el usuario ha seleccionado correctamente un modelo de origen o destino, según corresponda, la información relativa a dicho modelo (nombre, ruta, metamodelo) es traslada al método *createControl*, descrito anteriormente, para que este se encargue de rellenar las tablas mostradas en la Figura 5-33.

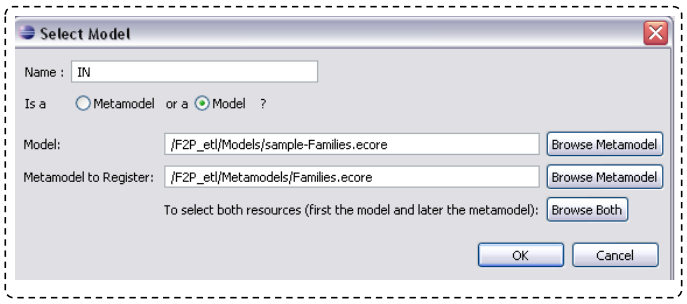

Figura 5-35. Ventana de definición de modelos participantes

5.3.3.1.5 Clase ModelData

Esta clase, implementada en el fichero del mismo nombre, permite almacenar información de un modelo (o metamodelo). En concreto, almacena su nombre, su ruta y la ruta de su metamodelo (en caso de ser un modelo). Además, contiene un atributo de tipo booleano que indica si se trata un metamodelo. Ha sido creada, principalmente, con el objetivo de poder definir estructuras de datos que contengan al mismo tiempo toda la información relativa a los modelos.

Esta clase se compone de los constructores de la clase, los métodos *set* para dar valor a estos atributos y los métodos *get* para obtener su valor.

5.3.4 Implementación de Transformaciones de modelos

Hasta el momento se ha presentado la implementación de los metamodelos de los DSLs que componen la herramienta MeTAGeM-Trace y los mecanismos para crear, editar y visualizar los modelos terminales conformes a dichos metamodelos.

El siguiente paso es, de acuerdo con la arquitectura y diseño de la herramienta, implementar las tareas que forman el procesador de modelos y permiten interconectar dichos DSLs para alcanzar un objetivo común. Así, en esta sección se presenta la implementación de las transformaciones de modelos que forman parte de la herramienta MeTAGeM-Trace, que han sido especificadas en lenguaje natural en la sección 4.3.

Las transformaciones de modelos que forman parte de MeTAGeM-Trace han sido implementadas mediante el lenguaje de transformación ATL. Para desarrollar estas transformaciones, ATL proporciona un *plug-in* construido sobre la plataforma de Eclipse, que proporciona un conjunto de funcionalidades (resaltado de sintaxis, depurador, editor, etc.) que facilita el desarrollo de las transformaciones de modelos. ATL está basado principalmente en el estándar de

OCL y soporta la aproximación declarativa y la imperativa conjuntamente, es decir la aproximación híbrida, aunque sus autores recomiendan el uso de la aproximación declarativa.

Las transformaciones en ATL se implementan definiendo un conjunto de reglas: cada regla especifica un patrón origen y un patrón destino, a nivel de metamodelo. Cuando se ejecuta la transformación, el motor de ATL establece las relaciones entre los patrones origen y cada uno de los elementos del modelo origen. Por cada relación se instancia el patrón destino en elementos del modelo destino.

Una de las ventajas de ATL es que permite la herencia de reglas y la programación implícita y explícita de las mismas. La programación implícita se soporta por medio de las construcciones imperativas. Cuando comienza la ejecución de la transformación, el algoritmo empieza a ejecutar las reglas definidas como puntos de entrada (*entrypoint*), las cuales pueden llamar a otras reglas. Una vez terminada esta fase, el motor verifica de forma automática las coincidencias entre los patrones origen y los elementos del modelo origen y ejecuta las reglas correspondientes. Por último, se ejecutan las reglas definidas como puntos de salida (*endpoint*). La programación explícita se soporta por medio de la llamada a reglas desde el bloque imperativo de otras reglas. Estas reglas se transforman en instrucciones para la máquina virtual de ATL, que ejecuta la transformación.

En las siguientes sub-secciones, se muestra la implementación de las reglas de transformación de modelos que forman parte de la herramienta presentada.

5.3.4.1 Transformación de modelos PIM a modelos PSM

A partir de la especificación de las reglas de transformación entre los metamodelos PIM y PSM (Tabla 4-1), en esta sección se presenta su implementación en el lenguaje ATL. Esta implementación ha sido denominada *MeTAGeM2Hybrid*, de acuerdo al nombre de los metamodelos que participan en la transformación. En la Figura 5-36 se muestra la cabecera de la transformación que define que la entrada de la transformación es un modelo MeTAGeM (nivel PIM) y la salida un modelo Hybrid (nivel PSM).

```
module MeTAGeM2Hybrid;
create OUT : Hybrid from IN : MeTAGeM;
```

Figura 5-36. Cabecera Transformación MeTAGeM2Hybrid

Una vez definida la cabecera del módulo de la transformación comienza la implementación de cada una de las reglas. En primer lugar se realiza la implementación de la regla que transforma los elementos de tipo *ModelRoot* a elementos de tipo *Module*. Como muestra la Figura 5-37, en esta regla se genera el nombre del módulo de la transformación a construir (*name*) y se establecen las referencias correspondientes a los modelos de origen y destino y a las reglas dicha transformación.

```
rule ModelRoot2Module {
from
      mr: MeTAGeM!ModelRoot
to
      m: Hybrid!Module (
            name <- mr.name,
            sourceModels <- mr.sourceModels,
            targetModels <- mr.targetModels,
            rules <- mr.relations
      )
}
```

Figura 5-37. Regla *ModelRoot2Module* (Transformación MeTAGeM2Hybrid)

A continuación, se implementan las reglas de transformación que generan los modelos origen (*SourceModel*) y destino (*TargetModel*) participantes en la transformación final (Figura 5-38).

```
rule SourceModelTransf2SourceModel{
from
      smt: MeTAGeM!SourceModelTransf
to
      sm: Hybrid!SourceModel(
            name <- smt.name+'_model',
            path <- smt.path,
            type_mm <- smt.name
      )
}
rule TargetModelTransf2TargetModel{
from
      tmt: MeTAGeM!TargetModelTransf
to
      tm: Hybrid!TargetModel(
            name <- tmt.name+'_model',
            path <- tmt.path,
            type_mm <- tmt.name
      )
}
```

Figura 5-38. Reglas de transformación de los modelos origen y destino (Transformación MeTAGeM2Hybrid)

El siguiente paso consiste en la implementación de las reglas de transformación que convierten las relaciones definidas a nivel PIM en reglas de transformación y relaciones de trazabilidad, a nivel PSM.

Las metaclases que describen las relaciones a nivel PIM, heredan de una metaclase abstracta y la principal diferencia entre dichas relaciones es la cardinalidad que admiten sus atributos *source* y *target*. Por ello, se ha decidido implementar una regla de transformación abstracta (Figura 5-39) que es extendida por las reglas relativas a los tipos de relaciones definidos. En dicha regla, a partir de una relación (*Relations*) no contenida en otra relación, se genera una regla de transformación (*Rule*) y una relación de trazabilidad (*TraceRule*) en el modelo híbrido. Los elementos generados se relacionan entre sí mediante el atributo *trace*.

Para mejorar la modularidad de las reglas se han definido algunas funciones auxiliares *helpers* como *getRuleName* que permite recuperar el nombre de la regla que se está generando. Dicho nombre se genera a partir de la concatenación de los nombres de los elementos origen y destino que participan en la relación. En caso de que el usuario no especifique el nombre de la relación en el modelo a nivel PIM, por medio de otro *helper* (*getInOutPatternName*) se genera un nombre por defecto. De esta manera se asegura que todas las reglas tengan un nombre, lo que evita errores en la generación de los siguientes modelos.

```
abstract rule Relations2Rule{
from relation:MeTAGeM!Relations(relation.isNotIncluded())
to
      r_hybrid:Hybrid!Rule(
            name <- relation.getRuleName(),
            isAbstract <- relation.role=#IsAbstract,
            isMain <- relation.role=#IsMain,
            isUnique <- false,-- is the default value
            "extends" <- relation.extends,
            isExtended <- relation.isExtended,
            typeRelation <- relation.typeRelation,
            typeElement <- relation.typeElement,
            guard <- if relation.guardCondition.oclIsUndefined()
                  then OclUndefined else
                        thisModule.getGuard(relation)
                  endif,
            trace <- trace_rule
            ),
      trace_rule: Hybrid!TraceRule(
            name <- relation.getRuleName(),
            source <- if relation.haveSource() then
            relation.source else OclUndefined endif,
            target <- if relation.haveTarget() then
            relation.target else OclUndefined endif
            )
}
```

Figura 5-39. Regla abstracta *Relations2Rule* (Transformación MeTAGeM2Hybrid)

Para entender más fácilmente el funcionamiento de las reglas que extienden *Relations2Rule*, la Figura 5-40 presenta un ejemplo gráfico.

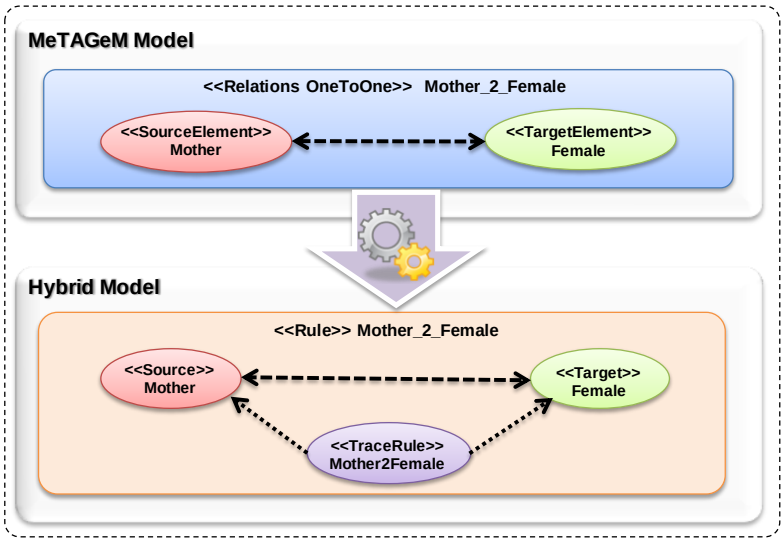

Figura 5-40. Ejemplo gráfico de regla *Relations2Rule* (Transformación MeTAGeM2Hybrid)

Para mostrar cómo se extiende la regla anterior, la Figura 5-41 presenta la regla de transformación que genera las reglas y las relaciones de trazabilidad a partir de una relación de tipo *ManyToMany*. Como se puede observar, la regla *Relations2Rule* se extiende mediante la palabra reservada *extends* y se añade la generación de las propiedades *sources* y *targets* asignándoles los valores de las propiedades *source* y *target* del elemento *ManyToMany*.

```
rule ManyToMany2rule extends Relations2Rule{
from
      relation: MeTAGeM!ManyToMany
to
      r_hybrid: Hybrid!Rule(
            sources <- relation.source.asSequence(),
            targets <- relation.target.asSequence()
      )
do {
 thisModule.countRules <- thisModule.countRules + 1;
}
}
```

Figura 5-41. Regla *ManyToMany2Rule* (Transformación MeTAGeM2Hybrid)

El resto de los tipos de relaciones (*OneToOne*, *OneToZero*, *ZeroToOne*, *OneToMany* y *ManyToOne*), se codifican de manera similar, variando para cada uno de los casos la generación de las propiedades *sources* y *targets*.

```
abstract rule Relations2Binding{
from relation:MeTAGeM!Relations(relation.isIncluded())
to
      binding:Hybrid!Binding(
            name <- relation.getRuleName(),
            typeRelation <- relation.typeRelation,
            typeElement <- relation.typeElement,
            owned <- relation.refImmediateComposite(),
            trace <- trace_binding
      ),
      trace_binding: Hybrid!TraceBinding(
            name <- relation.getRuleName(),
            source <- if relation.haveSource()then
                        relation.source
                      else OclUndefined
                      endif,
            target <- relation.target,
            parent <- thisModule.resolveTemp(
                        relation.refImmediateComposite().
                        refImmediateComposite(),'trace_rule')
      )
}
```

Figura 5-42. Regla abstracta *Relations2Binding* (Transformación MeTAGeM2Hybrid)

En el caso de las relaciones contenidas en otras relaciones, la implementación se ha realizado de forma similar, empleando una regla abstracta que ha sido extendida por los tipos de relación que pueden estar contenido (*OneToOne*, *ManyToOne* y *ZeroToOne*). En la Figura 5-42, se muestra la regla abstracta que, a partir de una relación contenida en otra relación, genera un elemento de tipo *Binding* y una relación de trazabilidad de tipo *TraceBinding*, relacionados entre sí mediante el atributo *trace*.

El resto del código que implementa esta transformación puede consultarse en el CD adjunto a esta tesis doctoral.

5.3.4.2 Transformación de modelos PSM a modelos PDM

En esta sección se presenta la implementación de los modelos PSM (aproximación híbrida) a los modelos dependientes de plataforma: ATL y ETL.

Como ya se ha indicado en la sección 4.3.2, estas transformaciones son las más complejas entre todas las que forman parte de MeTAGeM-Trace, ya que el modelo resultante debe contener el modelado de la transformación a nivel PDM y las estructuras que forman el generador de trazas (ver Figura 4-1).

5.3.4.2.1 Transformación de modelos PSM a modelos ATL

A partir de la especificación de la transformación presentada en la Tabla 4-2, se lleva a cabo la implementación de las reglas de la transformación en el lenguaje ATL. La implementación de esta transformación se compone de 18 reglas tipo *matched*, 17 reglas tipo *lazy*, 2 reglas tipo *called* y 13 *helpers*. Debido al gran tamaño de la implementación de la transformación, en esta sección solo se presentan aquellas partes relevantes y se invita al lector a consultar la transformación completa en el CD adjunto.

La regla de transformación que genera el módulo de la transformación a nivel PDM a partir del módulo de la transformación a nivel PSM, además debe crear un conjunto de estructuras que permitan generar, posteriormente, el modelo de trazas. Así, en primer lugar es necesario crear un nuevo elemento que represente al modelo de trazas como otro modelo destino de la transformación (Figura 5-43(a)). Para facilitar la comprensión al lector, la Figura 5-43(b) presenta un ejemplo del código ATL obtenido tras serializar el modelo destino de la transformación *hybrid2atl*.

(a)

```
-- Create module header
rule Module {
      from
            hybrid : Hybrid!Module
      to
            atl : ATL!Module (
                  isRefining <- false,
                  name <- hybrid.name.debug('Module'),
                inModels <- hybrid.sourceModels,
                  outModels <- hybrid.targetModels.append(traceModel),
                  elements <- hybrid.rules,
                  commentsBefore <- Set {'-- @atlcompiler atl2006'}
            ),

            traceModel: ATL!OclModel(
                  name <- 'in2out_trace',
                  metamodel <- ametamodelTrace
            ),
            ametamodelTrace : ATL!OclModel (
                  name <- 'TRACE'
            ),
```

(b)

```
module familes2persons;
create Persons_model : Persons, in2out_trace : TRACE
from Families_model : Families;
```

Figura 5-43. Fragmento regla *Module*: creando el modelo de Trazas (Transformación Hybrid2ATL)

El modelo de trazas final, como todos los modelos EMF, debe tener un elemento raíz (*TraceModel*). Para que este elemento se genere automáticamente en la ejecución de la transformación final y dado que se debe crear antes que el resto de elementos del modelo de trazas, la regla *Module* de la transformación *hybrid2atl* también genera una regla de tipo *entrypoint* (*calledRule*). Dicha regla comienza a rellenar el modelo de trazas, creando los modelos origen y destino. Además, ya que este elemento raíz se crea automáticamente y será necesario acceder a él durante la ejecución de la transformación final, esta regla también genera una operación ATL (*Helper*) que devuelve dicho elemento (Figura 5-44).

(a)

```
rule Module {
      from
            hybrid : Hybrid!Module
      to
            ...
      -- this called rule creates the root of the trace model
      createTraceModelRoot : ATL!CalledRule (
            name <- 'CreateTraceModelRoot'.debug('CalledRule'),
            outPattern <- outPattern,
            "module" <- hybrid,
            actionBlock <- anAction,
            --To create first the trace model root:
            isEntrypoint <- true,
            commentsBefore <- Set {'-- Comments -> This is a CalledRule
                  to create the root of Trace Model'}
            ),
      ...
      --creates a helper to get the root of the model trace
      modelRoot_var : ATL!Helper (
            "module" <- hybrid,
            definition <- adefinition,
            commentsBefore <- Set {'-- Comments -> This is a Helper to
                  get the root of model trace '}
            ),
      ...
```

(b)

```
-- Comments -> This is a CalledRule to create the root of Trace Model
entrypoint rule CreateTraceModelRoot() {
 to
  root : TRACE!TraceModel
 -- ActionBlock:
 do {
  thisModule.getTraceModelRoot <- root;
  thisModule.createSourceModel_c2_Families();
  thisModule.createTargetModel_c2_Persons();
 }
}
-- Comments -> This is a Helper to get the root of model trace
helper def: getTraceModelRoot() : TRACE!TraceModel =
 OclUndefined;
```

Figura 5-44. Fragmento regla *Module*: creando un *entrypoint* (Transformación Hybrid2ATL)

Las siguientes reglas implementadas son las que se corresponden con los modelos origen y destino de la transformación (*inMM* y *OutMM*). En dichas reglas, además de generar el elemento que los representa en el modelo ATL (*OclModel*), hay que generar la estructura correspondiente en el generador de trazas para que este pueda crear la representación de los modelos origen y destino en el modelo de trazas final. Así, dichas reglas también generan una regla tipo *called* y un *helper* para cada uno de los modelos origen y destino. Las reglas *called* serán invocadas por la parte imperativa de la regla *entrypoint* de la transformación final (Figura 5-44(b)) y los *helper* permitirán acceder a la representación de los modelos en el modelo de trazas.

En cuanto a las reglas de la transformación final, el lenguaje ATL permite la definición de varios tipos de reglas: *matched*, *lazy*, *unique lazy* y *called*. A nivel PSM, las reglas de la transformación contienen diferentes atributos que permiten establecer de forma inequívoca a qué reglas ATL deben transformarse. Por ello, se define una regla de transformación por cada uno de los tipos mencionados, empleando estos atributos como pre-condición (*guard*) de cada una de ellas.

Siguiendo la propuesta presentada por Jouault en [97] para la generación de trazas a partir de transformaciones ATL, las instancias de las relaciones de trazabilidad (enlaces de trazas) asociadas a las reglas de la transformación serán generadas como elementos destino de dichas reglas, como muestra la Figura 5-45.

```
rule A2B_Trace {
 from
  a : MM_Source!A
 to
  b : MM_Target!B (),
  a_2_b_TL1 : TRACE!TraceLink (
   name <- a2b_trace,
   source <- a_trace,
   target <- b_trace
  ),
  a_trace: Trace!Source(),
  b_trace: Trace!Target()
}
```

Figura 5-45. Generando trazas como elementos destino de una regla

Para aplicar esta idea a la transformación *hybrid2atl* es necesario realizar dos tareas: por un lado generar un elemento destino por cada relación de trazabilidad encontrada en el modelo PSM (reglas *TraceRule2CreateTraceLink* y *TraceBinding2CreateTraceLink*) y por otro, crear los *bindings* adecuados en dichos elementos destino por cada uno de los elementos origen y destino de la

relación de trazabilidad. Para llevar a cabo esta segunda tarea, se han implementado seis reglas distintas que son invocadas por cada elemento origen o destino (tres reglas para elementos origen y tres para elementos destino). En dichas reglas se comprueba si la regla a la que pertenece el elemento contiene una relación de trazabilidad o no y si el elemento está contenido en una regla o en un *binding*. Con esta información estas reglas crean los elementos adecuados en el modelo ATL, por ejemplo en la Figura 5-46, se muestra un fragmento de la regla de transformación que es invocada por los elementos origen, contenidos en una regla que tiene asociada una traza. En dicha regla, se genera un elemento origen de la transformación ATL (*SimpleInPatternElement*) y un elemento de salida que representa el elemento de la traza (*SimpleOutPatternElement*). Respecto de la Figura 5-45, estos elementos se corresponden con *a* y *a_trace*.

```
--InputPattern (from part)
rule InPatternElement_withTrace {
from
      inPattern : Hybrid!Source (inPattern.refImmediateComposite().
            oclIsTypeOf(Hybrid!Rule)and inPattern.traceLink.size()>0)
to
      atl : ATL!SimpleInPatternElement (
            varName <- inPattern.name.toLower()+'_in',
            type <- aType
      ),
      aType : ATL!OclModelElement(
            name <- inPattern.name,
            model <- thisModule.resolveTemp(inPattern.model,'ametamodelMM')
      ),
      trace_element: ATL!SimpleOutPatternElement (
            varName <- inPattern.name.toLower()+
                        '_in_Trace_TE'+thisModule.countTE.toString(),
            outPattern <- thisModule.resolveTemp(inPattern."rule",'outPattern'),
            type <- aType_trace,
            bindings <- Sequence{nameBinding, refBinding, modelBinding}
      ),
      aType_trace : ATL!OclModelElement(
            name <- 'SourceElement',
            model <- thisModule.resolveTemp(
                  inPattern.refImmediateComposite().
                  refImmediateComposite(),'ametamodelTrace')
            ),
```

Figura 5-46. Fragmento regla *InPatternElement_withTrace* (Transformación Hybrid2ATL)

A partir de las reglas de transformación descritas en esta sección, se genera la mayor parte del generador de trazas embebido en la transformación final. Una vez más, se recuerda que todo el código implementado durante el desarrollo de la herramienta MeTAGeM-Trace se encuentra en el CD adjunto.

5.3.4.2.2 Transformación de modelos PSM a modelos ETL

En la Tabla 4-3, se ha presentado la especificación de las reglas de transformación entre los modelos PSM (aproximación híbrida) y los modelos

dependientes del lenguaje ETL que contienen embebido el generador de trazas. En esta sección, se presenta la implementación de dichas reglas de transformación. Esta implementación resulta algo más sencilla que la transformación de PSM a ATL, debido a que el metamodelo definido para el lenguaje ETL (Figura 4-8) es más sencillo que el proporcionado por el lenguaje ATL (Figura 4-7). Así, se compone de 20 reglas *matched*, 9 reglas *lazy*, 5 reglas *called* y 10 operaciones *helpers*. Al igual que en el caso anterior, en esta sección solo se muestra la implementación de las reglas de transformación más relevantes.

Las decisiones tomadas en la implementación de esta transformación son similares a las tomadas en la implementación de la transformación de PSM a ATL. Así, una de las prioridades es definir el mecanismo para que en la ejecución de la transformación final, el elemento raíz del modelo de trazas se genere en primer lugar. Esta tarea se llevará a cabo en el bloque *pre* de la transformación final, implementada en ETL. Para desarrollar esta funcionalidad se ha implementado un *helper* (*getPre*) que contiene el código de dicho bloque. En primer lugar, contiene el código ETL que genera el elemento raíz del modelo de trazas (*TraceModel*) y posteriormente se irán añadiendo las líneas de código requeridas por el generador de trazas. Así, como muestra la Figura 5-47(a), la regla que genera el módulo de la transformación final a nivel PDM, además de implementar esta transformación, incluye el código que genera el nombre del elemento *TraceModel* en el bloque *pre* de la transformación final. Del mismo modo que en la sección anterior, para facilitar la comprensión, la Figura 5-47(b) muestra el código ETL de la transformación final que se ha generado (tras la serialización del modelo) a partir del fragmento de código mostrado en la parte (a) de dicha figura.

(a)

```
--Helper -> Pre text
helper def:getPre:String='var traceModel: new Traceability!TraceModel;';
-- Create module header
rule Module {
 from
      hybrid : Hybrid!Module
 to
      etl : ETL!EtlModule (
            name <- hybrid.name.debug('Module'),
            rules <- hybrid.rules,
            operations <- hybrid.operations
      )
 do{
      thisModule.getETLmodule <- etl;
      thisModule.getPre <- thisModule.getPre + '
      traceModel.name = \''+hybrid.name+'_traces\';
            ';
 }
}
```

(b)

```
pre  {
 var traceModel: new Traceability!TraceModel;
 traceModel.name = 'familes2persons_traces';
 ...
}
```

Figura 5-47. *Helper getPre* y regla *Module*: creando bloque *pre* (Transformación Hybrid2ETL)

El metamodelo de ETL no almacena información acerca de los modelos de origen y destino, ya que en el código de la transformación los modelos son tratados como otras variables más y posteriormente, en la configuración de la ejecución se relacionan dichas variables con los modelos origen y destino. Sin embargo, el metamodelo de trazas sí requiere almacenar cierta información sobre los modelos, por ello, el generador de trazas embebido en la transformación ETL debe contener mecanismos para la generación de los elementos que representan a los modelos en el modelo de trazas. Para esta tarea, se han definido en la transformación *hybrid2etl* dos reglas, una para los modelos origen (*createSourceModel*) y una para los modelos destino (*createTargetModel*). Del mismo modo que en la transformación PSM a ATL, los modelos origen y destino del modelo de trazas se deben crear al inicio de la transformación, por ello, estas reglas añaden al bloque *pre* de la transformación la información necesaria para generar en el modelo de trazas cada uno de los modelos origen y destino participantes en la transformación final.

El siguiente paso consiste en la implementación de las reglas de transformación que generan reglas ETL a partir de reglas definidas a nivel PSM. El lenguaje ETL define varios tipos de reglas, que han sido descritas en la sección 4.2.4.2. Sin embargo, tanto en el metamodelo PSM como en el metamodelo ETL

todas ellas son representadas mediante la metaclase *Rule* (*TransformationRule* en ETL) y se diferencian por el valor de un conjunto de atributos. Así, para generar los elementos *TransformationRule* en los modelos ETL se ha implementado la regla *createRule* (Figura 5-48).

```
rule createRule{
from
 r: Hybrid!Rule(r.sources.size()=1 and r.targets.size()>0)
to
 etl_r: ETL!TransformationRule(
      name <- r.name,
      isAbstract <- r.isAbstract,
      "extends" <- r."extends",
      "lazy" <- not r.isMain,
      guard <- r.guard,
      source <- r.sources.first(),
      targets <-r.targets.append(thisModule.resolveTemp(
                  r.trace,'trace_as_element')).append(
                  r.trace.traceBindings->collect(
                  b|thisModule.resolveTemp(
                  b,'trace_as_element'))),
      bindings <- r.targets->collect(t|t.bindings).asSequence()
)
```

Figura 5-48. Regla *createRule* (Transformación Hybrid2ETL)

En dicha regla se crean los valores de los atributos *name*, *isAbstract*, *extends*, *lazy*, *guard* de las reglas ETL así como los elementos origen, destino y los *bindings*. Como se ha indicado en la implementación de la transformación PSM a ATL, las relaciones de trazabilidad definidas a nivel PSM son transformadas en elementos de salida de las reglas PDM. Por ello, en el atributo *target* se concatenan los elementos de salida de la regla ETL con los elementos que se generan a partir de las relaciones de trazabilidad asociadas a dicha regla (*TraceRule* y *TraceBinding*).

Para convertir las relaciones de trazabilidad (*TraceRule* y *TraceBinding*) en elementos destino de las reglas, se han implementado dos reglas de transformación. Ambas reglas son similares, pues generan un elemento ETL (*Element*) y en su parte imperativa llaman a una regla tipo *lazy* que se encarga de definir los atributos de la futura traza. La principal diferencia entre estas reglas es el elemento que las invoca. En la Figura 5-49 (a), se presenta la regla ATL que genera los elementos ETL a partir de un elemento *TraceRule* (nivel PSM) y en la Figura 5-49 (b), un ejemplo del resultado de ejecutar dicha regla (en negrita).

(a)

```
--TraceLink_Rule
rule TraceRule2CreateTraceLink{
 from
      traceRule: Hybrid!TraceRule
 to
      trace_as_element: ETL!Element(
            name <- traceRule.name,
            className <- 'TraceLink',
            metamodel <- 'Traceability'
      )
 do{
      thisModule.generateBindings_TraceRule(traceRule,
            thisModule.resolveTemp(
            traceRule.refImmediateComposite(),'etl_r'));
 }
}
```

(b)

```
rule Father_2_Male
 transform father : Families!Father
 to male : Persons!Male,
      Father_2_Male : Traceability!TraceLink,
      Father_trace : Traceability!SourceElement,
      Male_trace : Traceability!TargetElement,
{
  male.fullName := father.getFatherName();
  Father_2_Male.name := 'Father_2_Male';
  Father_2_Male.`operation` := Traceability!Operations#Transform;
  Father_2_Male.source := Sequence{Father_trace};
  Father_2_Male.target := Sequence{Male_trace};
  Father_2_Male.traceModel := traceModel;
  ...
}
```

Figura 5-49. Regla *TraceRule2CreateTraceLink* (Transformación Hybrid2ETL)

El siguiente paso es construir las reglas para definir los elementos origen y destino de las reglas de la transformación modelada. Para ello, se han implementado ocho tipos de reglas distintas, de acuerdo a las características de los elementos (pre-condiciones o *guards* de las reglas). La Tabla 5-1 muestra cuando se debe ejecutar cada una de ellas:

Tabla 5-1. Reglas de Transformación implementadas para elementos origen y destino (Transformación Hybrid2ETL)

Elemento Origen	Elemento de Regla	No implicado en relación de trazabilidad	*Source2Element_withoutTrace*
		Implicado en relación de trazabilidad	*Source2Element_withTrace*

	Elemento de *Binding*	No implicado en relación de trazabilidad	*Source2Feature_withoutTrace*
		Implicado en relación de trazabilidad	*Source2Feature_withTrace*
Elemento Destino	Elemento de Regla	No implicado en relación de trazabilidad	*Target2Element_withoutTrace*
		Implicado en relación de trazabilidad	*Target 2Element_withTrace*
	Elemento de *Binding*	No implicado en relación de trazabilidad	*Target 2Feature_withoutTrace*
		Implicado en relación de trazabilidad	*Target 2Feature_withTrace*

Estas reglas, básicamente, se encargan de generar los elementos origen y destino en las reglas de la transformación ETL y en el caso de que estén implicados en alguna relación de trazabilidad, generan los elementos de destino necesarios para que el generador de trazas pueda completar adecuadamente el modelo de trazas final. El código que implementa estas reglas y el resto de reglas no detalladas aquí, puede consultarse en el CD adjunto a esta tesis.

5.3.5 Implementación de la Generación de Código

En esta sección se presenta la implementación de los mecanismos que permiten la serialización de los modelos PDM (ATL y ETL) en el código que implementa la transformación.

Como se indica en la sección 4.3.3, el *framework* de ATL para Eclipse proporciona un *parser* implementado en el lenguaje TCS (*Textual Concrete Syntax* [100]) que permite obtener el código de la transformación en ATL a partir de un modelo conforme al metamodelo del lenguaje y viceversa. Así pues, dado que ya existe una implementación para la generación de código ATL, proporcionada por sus propios desarrolladores, no es necesario implementar el generador de código para dicho lenguaje. Solamente, como se muestra en la sección 5.3.7, es necesario integrar dicha funcionalidad con el resto de módulos de MeTAGeM-Trace.

En el caso del lenguaje ETL, para el cual se ha construido un DSL completo, sí es necesario implementar el generador de código. En la siguiente sub-sección se detalla su implementación.

5.3.5.1 Generación de Código ETL

De acuerdo a la especificación de las reglas de generación de código ETL, presentada en la sección 4.3.3 y al diseño técnico de MeTAGeM-Trace (sección 5.2), en esta sección se presenta la implementación, en términos del lenguaje MOFScript [144], del generador de código ETL a partir de los modelos de transformación.

En primer lugar se crea un fichero MOFScript (extensión *.m2t*), se declara su cabecera, indicando la URI del metamodelo (Figura 5-50) y se procede a implementar el código que se generará a partir de cada una de las metaclases del metamodelo del DSL, en este caso el de ETL.

En la generación de código mediante el lenguaje MOFScript se debe definir qué regla se ejecuta en primer lugar (palabra reservada *main*). En este caso, la generación de código comienza por el elemento raíz del metamodelo del lenguaje ETL (*EtlModule*). En la Figura 5-50 se presenta la regla generadora de código para dicha metaclase.

Dado que se trata de la regla principal del generador de código, en esta regla se define dónde se almacenará el código resultante de la generación. Para ello, se emplea la función *file* y en este caso, se ha decidido que el fichero de código se guarde con el nombre del elemento raíz y la extensión propia de los ficheros de transformación ETL (*.etl*). En caso de que no se haya definido el nombre del módulo de la transformación, el fichero de código obtenido será *generated.etl*.

```
texttransformation ETLGenerator
(in eco:"http://org.eclipse.kybele.metagem.ETL") {

eco.EtlModule::main () {
 var name:String

 if (self.name.size()=0)
      name="generated.etl"
 else
      name=self.name + ".etl"

 file (name)
 println("pre "+self.pre.name+" {")
 if (self.name.size()!=0)
      println("'Running ETL: "+self.name+" Transformation'.println();")
      self.pre.body
      println("")
     println("}")
     println("")

     println("post "+self.post.name+" {")
      self.post.body
      println("")
     println("}")
     println("")

     self.rules->forEach(r:eco.TransformationRule){
        r.generateRule()
      println("")
     }

     self.operations->forEach(o:eco.Operation){
        o.generateOperation()
      println("")
     }
  }
```

Figura 5-50. Cabecera y Generación de código de la metaclase EtlModule

En la regla anterior se puede observar que en primer lugar se genera el código relativo al bloque *pre* y posteriormente el del bloque *post*. Debido a que los atributos de la metaclase *EolBlock* (*name* y *body*) son de tipo *String* tan solo es necesario incluir su nombre en la regla para generar su valor en el código resultante. El caso de las reglas y las operaciones es distinto ya que es necesario generar estructuras de código más complejas. Por ello, por cada regla y operación contenida en el modelo de la transformación se invocan, respectivamente, las reglas generadoras *generateRule* (Figura 5-51) y *generateOperation* (Figura 5-52), que se muestran a continuación.

```
eco.TransformationRule::generateRule(){
  if(self.greedy)
    println("@greedy")
  if(self.isAbstract)
    println("@abstract")
  if(self.lazy)
    println("@lazy")
  if(self.primary)
    println("@primary")
  println("rule "+ self.name)
  print(" transform ") self.source.generateElementRule()
  println("")
  print(" to ") self.targets->forEach(e:eco.Element){
          e.generateElementRule()
          if(self.targets.indexOf(e)!=self.targets.size()-1)
                print(", ")
   }
  println("")

  if(self._getFeature("extends").size()>0){
    println("extends")
    self._getFeature("extends")->forEach(ex:eco.TransformationRule){
    ex.name print(",")}
  }
  println("{")

  if(self._getFeature("guard").size()>0)
    println("guard : " + self._getFeature("guard").body)

  self.bindings->forEach(b:eco.Binding){b.generateBinding()}

  println("}")
}
```

Figura 5-51. Generación de código de la metaclase T*ransformationRule*

En la regla *generateRule*, en primer lugar se generan las etiquetas o *flags* que definen el tipo de regla ETL (*greedy*, *abstract*, *lazy* o *primary*). Posteriormente se crea la cabecera de la regla que incluye el nombre de la misma y los elementos de origen y destino que participan, separados por comas. En caso de que la regla extienda de otras reglas, se genera la palabra reservada *extends* del lenguaje ETL y el nombre de las reglas extendidas. A continuación, se genera el código relativo a la pre-condición de la regla (*guard*) y finalmente, por cada una de las asignaciones se invoca a la regla generadora *generateBinding*.

A continuación, la Figura 5-52 muestra la generación del código de la metaclase *Operation* de ETL. En esta regla, se genera el código de las pre y post-condiciones de la operación, la cabecera de la operación que incluye el contexto de la operación, el nombre, los parámetros y el tipo de retorno y por último, el cuerpo de la operación.

```
eco.Operation::generateOperation(){
  if(self.annotation!=null and self.annotation.size()>0)
    println("@"+self.annotation)
  if(self.pre!=null and self.pre.body.size()>0)
    println("@pre "+self.pre.body)
  if(self.post!=null and self.post.body.size()>0)
    println("@post "+self.post.body)
  print("operation ") self.context.generateOPstatement()
  print(" "+self.name+"(")
  self.parameters->forEach(p:eco.OperationParameter){
    p.generateOpParameter()
    if(p!=self.parameters.last())print(",")
    }
  print(") : ")
  self._getFeature("return").generateOPstatement()
  println(" {")
  println("\t"+self.body)
  println("}")
}
```

Figura 5-52. Generación de código de la metaclase *Operation*

Para facilitar el entendimiento de esta regla generadora de código, en la Figura 5-53 se presenta un ejemplo de una operación ETL generada mediante la regla *generateOperation*.

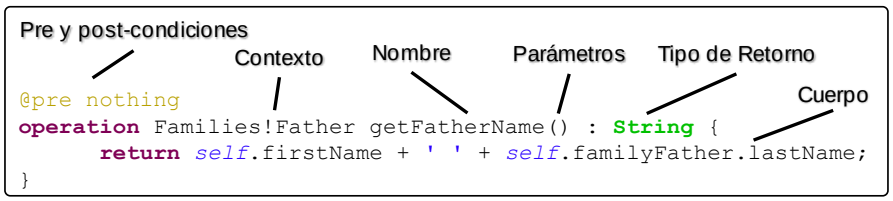

Figura 5-53. Operación ETL generada

Del mismo modo, se han implementado otras reglas generadoras para el resto de las metaclases del metamodelo de ETL: *generateOpParameter*, *generateOpStatement* *generateStatement*, *generateElementRule* y *generateBinding*. La implementación de estas reglas se puede consultar en el código fuente que se encuentra en el CD adjunto.

5.3.6 Validación automática de modelos

La tercera tarea que compone el procesador de modelos de MeTAGeM-Trace es la validación de los modelos que describen la transformación a alto nivel (niveles PIM y PSM). Para llevar a cabo esta validación, en la sección 4.4, se ha presentado la especificación de las reglas de validación que se deben

comprobar para cada uno los elementos de los modelos PIM y PSM para verificar que se trata de un modelo válido. Dicho control permite asegurar que los errores que se dan en la construcción de los modelos de las transformaciones no sean propagados a los niveles de abstracción inferiores y así de este modo, se mejora la calidad de la transformación final generada.

Para implementar dichas reglas de validación en MeTAGeM-Trace, de acuerdo con el diseño técnico presentado en la sección 5.2, se ha empleado el lenguaje de validación EVL (*Epsilon Validation Language*, [116]), perteneciente a la familia de lenguajes de Epsilon [114, 115]. Epsilon es un componente de Eclipse que soporta algunas de las tareas relacionadas con el desarrollo dirigido por modelos. Para ello, proporciona un conjunto de lenguajes y herramientas especializados en las diferentes tareas como comparación, transformación o *merging* de modelos.

EVL es un lenguaje que permite especificar y evaluar restricciones sobre los modelos. Para ello, un módulo EVL (ficheros con extensión *.evl*) se compone de un conjunto de restricciones o reglas de validación, que deben ser verificadas, a nivel de metamodelo. Posteriormente, estas reglas de validación son evaluadas a petición del usuario en los modelos conformes a dicho metamodelo.

En el desarrollo de la herramienta MeTAGeM-Trace se han implementado cuatro módulos de validación correspondientes a las cuatro especificaciones de validación presentadas en la sección 4.4: MeTAGeM, Hybrid, Hybrid_to_ATL y Hybrid_to_ETL. Mientras los dos primeros han sido desarrollados para la validación independiente de los modelos a nivel PIM y PSM, los dos últimos completan la validación Hybrid teniendo en cuenta el tipo de modelo que se generará (ATL o ETL) a partir del modelo PSM.

A continuación, se muestra la implementación de algunas de las restricciones más representativas de los módulos de validación desarrollados.

Así, en la Figura 5-54 se presenta una regla de validación que comprueba (*check*) si el atributo *name* del elemento *ModelRoot* (*context*) está definido. Si un elemento de este tipo no cumple esta restricción, el usuario recibirá un mensaje por pantalla que se genera mediante la operación *getMessageNotEmptyName* (*message*) y se le proporcionarán los medios para solucionar este error de validación (parte *fix*).

```
context ModelRoot {
-- ModelRoot name cannot be empty
constraint notEmptyModelRootName {
      check : self.name.isDefined()
      message : getMessageNotEmptyName(self.type().name.asString())
      fix {
       title : getTitleNotEmptyName(self.type().name.asString())
       do {
            self.name := getInputNotEmptyName(self.type().name.asString());
       }
      }
    }
```

Figura 5-54. Regla de validación *notEmptyModelRootName* (MeTAGeM)

El resto de reglas de validación siguen una estructura similar: *context*, *constraint*, *check*, *message* y *fix* (solo en algunos casos). Además, algunas incluyen una comprobación adicional (*guard*) que permite comprobar una pre-condición antes de validar la restricción. Por ejemplo, en el módulo de validación del metamodelo de aproximación híbrida, para comprobar que el atributo *name* del elemento raíz (*Module*) se ha definido correctamente es necesario comprobar previamente que dicho atributo ha sido definido (Figura 5-55).

```
context Module{

 [...]

 -- Module name  cannot be empty and should start with a letter
 -- following letters, numbers, dashes and underscores
 constraint validModuleName {
      guard : self.satisfies('notEmptyModuleName')
      check : self.name.isValidName()
      message : getMessageValidName(self.type().name.asString())
      fix {
            title : getTitleValidName("Module", self.name_module)
            do {
             self.name := getInputValidName(
                  self.type().name.asString(), self.name_module);
            }
      }
 }
```

Figura 5-55. Regla de validación *validModuleName* (Hybrid)

Con el objetivo de ilustrar la motivación que ha llevado a implementar los módulos de validación Hybrid_to_ATL y Hybrid_to_ETL se presenta la Figura 5-56. El lenguaje de transformación ETL no permite la definición de reglas con más de un elemento origen, sin embargo, a nivel PSM sí es posible definir reglas con N orígenes. Por ello, en el módulo de validación Hybrid_To_ETL se ha definido una restricción que no permite la existencia de reglas con más de un elemento origen.

```
context Rule{

 [...]

-- Validation to transform to ETL:
-- A rule cannot have more than one source
constraint manySources{
 check: (self.sources.size()<2)
 message: 'Due to current ETL version implemented: '
            +self.type().name+' only must have one source'
}
```

Figura 5-56. Regla de validación *manySources* (Hybrid_to_ETL)

La implementación completa de las reglas de validación se puede consultar en el código fuente que se encuentra en el CD adjunto.

5.3.7 Integración de los módulos

Una de las debilidades detectadas tradicionalmente en las herramientas MDE es la **usabilidad** [180]. En este sentido, la plataforma Eclipse ayuda a proporcionar una interfaz común, fácilmente extensible y adaptable a las necesidades específicas [44].

La herramienta MeTAGeM-Trace que se presenta en esta tesis doctoral, además de proporcionar un conjunto de editores para el modelado y visualización de los modelos terminales conformes a los DSL definidos, tiene por objetivo mejorar la experiencia del usuario en lo referente a la ejecución de los procesos de validación de modelos, transformación de modelos y generación de código. De forma que estas tareas resulten sencillas y en algunos casos (semi-)automáticas para el usuario de la herramienta.

Por todo ello, en esta sección se presenta cómo, a través del uso de los componentes existentes y la documentación disponible de la plataforma Eclipse [44, 70, 72], se ha implementado un conjunto de mecanismos o componentes (lanzadores, entradas en menús contextuales, etc.) para proporcionar una interfaz de usuario integrada que permita la ejecución sencilla de las tareas proporcionadas por MeTAGeM-Trace.

Dado que en el CD adjunto a esta tesis se encuentra el código completo de la herramienta, esta sección se centra en presentar los resultados obtenidos más importantes, sin profundizar en el código implementado.

5.3.7.1 Desarrollo de un plug-in de Integración

Uno de los objetivos de esta tesis doctoral es proporcionar una herramienta empaquetada que de soporte al desarrollo de transformaciones de modelos dirigido por modelos, que permitan la generación de trazas. Hasta el momento se ha presentado un conjunto de módulos que proporcionan la funcionalidad deseada, sin embargo para empaquetar todos estos módulos en una misma herramienta se hace necesario el desarrollo de un módulo o *plug-in* de integración.

Dicho *plug-in* proporciona los medios necesarios (lanzadores y entradas de menús contextuales) para que el usuario ejecute las transformaciones de modelos, la validación de los modelos y la generación de código, de forma sencilla sobre la plataforma Eclipse.

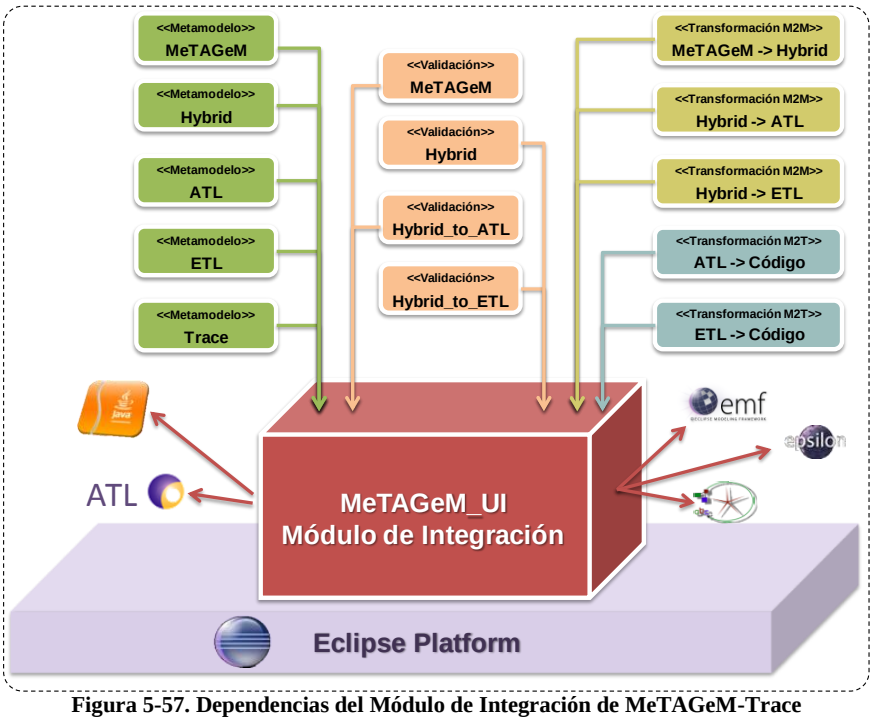

Figura 5-57. Dependencias del Módulo de Integración de MeTAGeM-Trace

Como muestra la Figura 5-57, para llevar a cabo el desarrollo del módulo de integración es necesario hacer uso de las tecnologías empleadas para el desarrollo de cada uno de los módulos anteriores (EMF, ATL, Epsilon, MOFScript y Java) y disponer de los metamodelos de los DSLs, las reglas de validación y las reglas de transformación que componen la herramienta. Así, para

añadir nuevos módulos o DSLs a la herramienta tan solo es necesario crearlos y actualizar el módulo de integración.

El *plug-in* de integración de MeTAGeM-Trace está compuesto por cinco metamodelos (MeTAGeM, Hybrid, ATL, ETL y Trace), cuatro módulos de validación (MeTAGeM, Hybrid, Hybrid_to_ATL y Hybrid_to_ETL), tres transformaciones de modelos (MeTAGeM2Hybrid, Hybrid2ATL y Hybrid2ETL) y dos generadores de código (ATL y ETL).

A continuación se muestra cómo se han integrado todos estos componentes para llevar a cabo la validación de los modelos, la ejecución de las transformaciones de modelos y la generación de código en el entorno Eclipse.

5.3.7.2 Integración de la Validación de Modelos

El *framework* EMF proporciona un lanzador de validaciones para modelos terminales construidos a partir de esta herramienta (Figura 5-59(1)). Para que dicho lanzador ejecute las restricciones EVL definidas, es necesario asociar los ficheros *.evl* con los *plug-in* que contienen que manejan los modelos terminales conformes al DSL, es decir en este caso, los proyectos MeTAGeM y Hybrid. Para ello, se debe incluir la extensión *org.eclipse.evl.emf.validation* en el fichero de configuración *plugin.xml* de la forma que muestra la Figura 5-58.

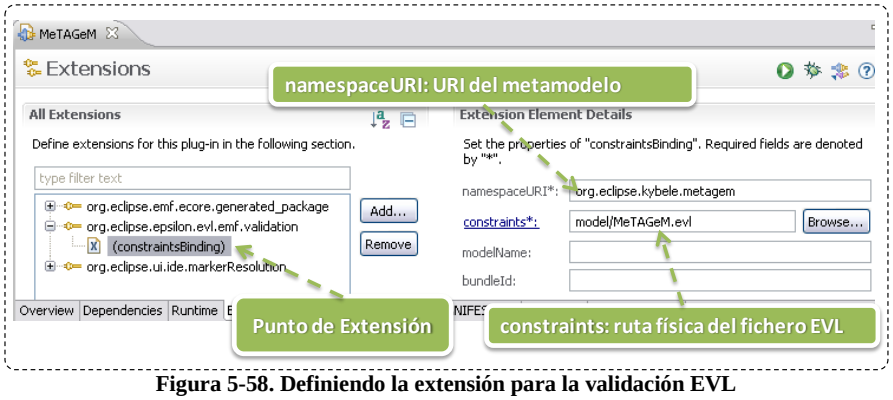

Figura 5-58. Definiendo la extensión para la validación EVL

Una vez definida la extensión, es posible ejecutar la comprobación de las reglas de validación a través del menú contextual del elemento raíz del modelo como muestra la Figura 5-59(1). En caso de que alguna restricción no sea satisfecha el usuario recibirá un mensaje por pantalla (2) y en consecuencia el modelo será marcado como invalido (3).

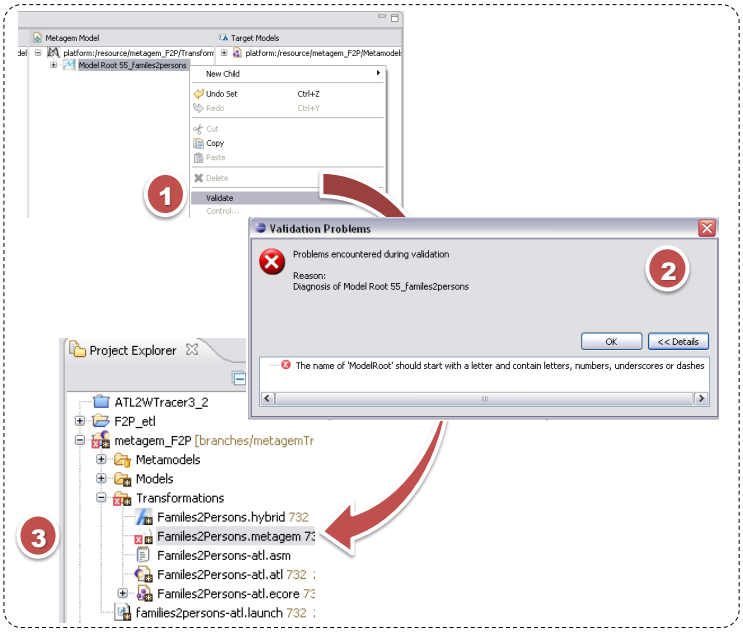

Figura 5-59. Lanzando una validación de modelos en MeTAGeM-Trace

De acuerdo a la información anterior, *a priori*, parece que la validación de los modelos no se debe incluir en el módulo de integración de MeTAGeM-Trace. Sin embargo, dado que a partir de los modelos se ejecutarán transformaciones para generar nuevos modelos, resulta deseable validarlos antes de ejecutar la transformación para asegurar que los modelos que se generen sean correctos. Por ello, como se indica en la siguiente sección, la validación de los modelos se integra en los mecanismos de lanzamiento de transformaciones para que la comprobación de los modelos se realice automáticamente.

5.3.7.3 Lanzamiento de Transformaciones de Modelos

Las transformaciones desarrolladas en la implementación de la herramienta MeTAGeM-Trace han sido implementadas mediante el lenguaje de transformaciones ATL. El *framework* de ATL para Eclipse proporciona a los usuarios un asistente para la configuración de la ejecución de las transformaciones como el que se muestra en la Figura 5-60. De esta manera, el usuario puede crear un lanzador que permita ejecutar la transformación, recuperarla y utilizarla en cualquier momento. Para ello es necesario disponer, en el espacio de trabajo, del código de la transformación, el modelo de origen y los metamodelos participantes en la transformación (metamodelos de los DSLs de MeTAGeM-Trace).

Para evitar que el usuario gestione directamente toda esta información, se proporcionan lanzadores específicos que por un lado, encapsulan esta información y por otro, simplifican la definición de las configuraciones de ejecución.

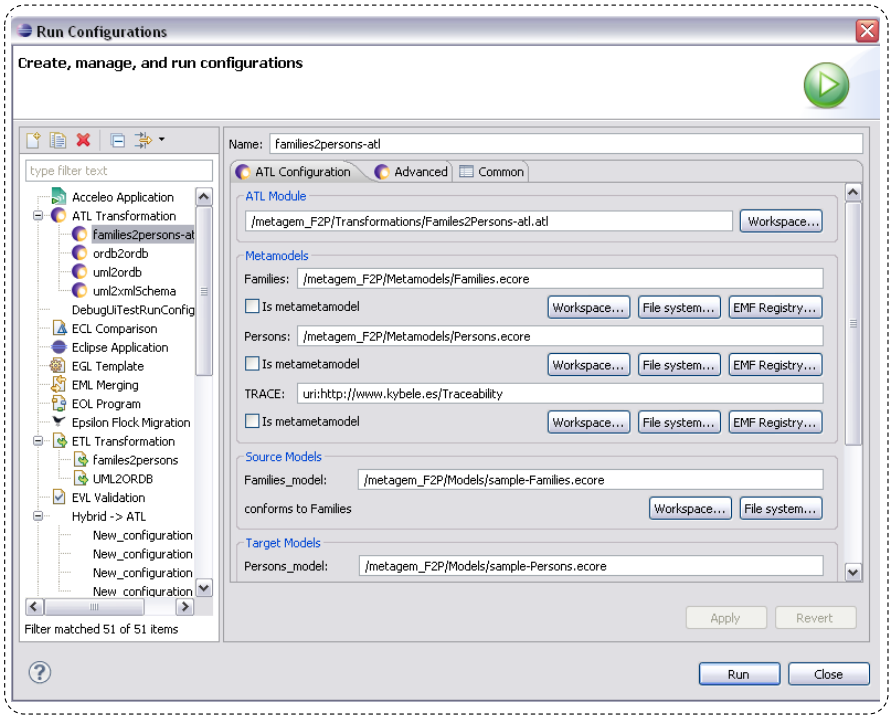

Figura 5-60. Configurando la ejecución de una transformación ATL

Para llevar a cabo esta tarea, en primer lugar, se deben definir, por cada transformación que se desee crear un lanzador específico, las extensiones adecuadas sobre el fichero *plugin.xml* del proyecto correspondiente al módulo de integración (MeTAGeM.ui). En este caso se crea una extensión de tipo *launchConfigurationTypes*, una de tipo *launchConfigurationTabGroups* y una de tipo *launchShorcuts* (Figura 5-61). La primera permite la creación de tipos de configuraciones, un ejemplo puede ser la configuración para la ejecución de la transformación de la Figura 5-60. La extensión *launchConfigurationTabGroups* permite la agrupación de pestañas en la ventana mostrada al usuario. Y por último, *launchShorcuts* proporciona un lanzador que se añade a los menús *Run*/*Debug* de Eclipse.

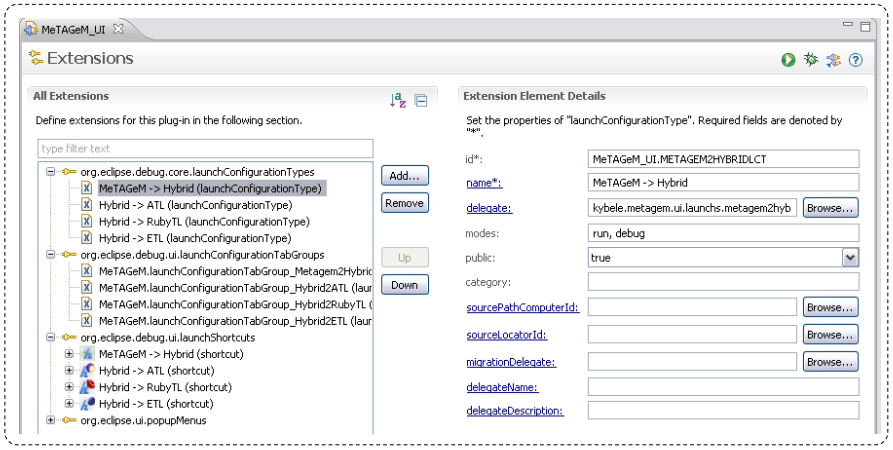

Figura 5-61. Definición de extensiones para los lanzadores de las transformaciones de modelos

Una vez definidas las extensiones, se debe implementar el código Java correspondiente a las clases de cada una de las extensiones. La clase Java relativa a la extensión *launchShorcuts* permite al usuario seleccionar un modelo de la extensión adecuada (por ejemplo *.metagem*) y ejecutar el proceso para la configuración del lanzador (Figura 5-62(a)). Si para dicho modelo ya se han definido configuraciones de lanzamiento, le mostrará al usuario una ventana en la que debe elegir una de las configuraciones definidas (Figura 5-62(b)). En el caso de que solo se haya definido una, automáticamente se prepara la ejecución de la transformación. En cambio, si hasta el momento no se ha definido ninguna configuración para el modelo seleccionado se procede a mostrar la ventana de configuración de la ejecución (Figura 5-62(c)). Dado que todas las transformaciones definidas en MeTAGeM-Trace tienen un modelo de entrada y uno de salida, para facilitar al usuario la configuración del lanzamiento de la transformación, se establecen automáticamente, a partir del fichero seleccionado, las rutas físicas de los modelos de entrada y salida. Así por ejemplo, si el usuario selecciona el modelo *families2persons.metagem* y su objetivo es generar un modelo PSM (*hybrid*), automáticamente se le sugerirá que la ubicación del modelo de salida sea la misma que la del modelo de entrada cambiando la extensión, es decir, *families2persons.hybrid* (Figura 5-62(c)).

La ventana de configuración del lanzamiento de la transformación se corresponde con la implementación de la clase relativa a la extensión *launchConfigurationTabGroups*. Esta implementación crea un grupo de dos pestañas: la pestaña propia de la configuración de la transformación y la pestaña común a todos los lanzadores de Eclipse.

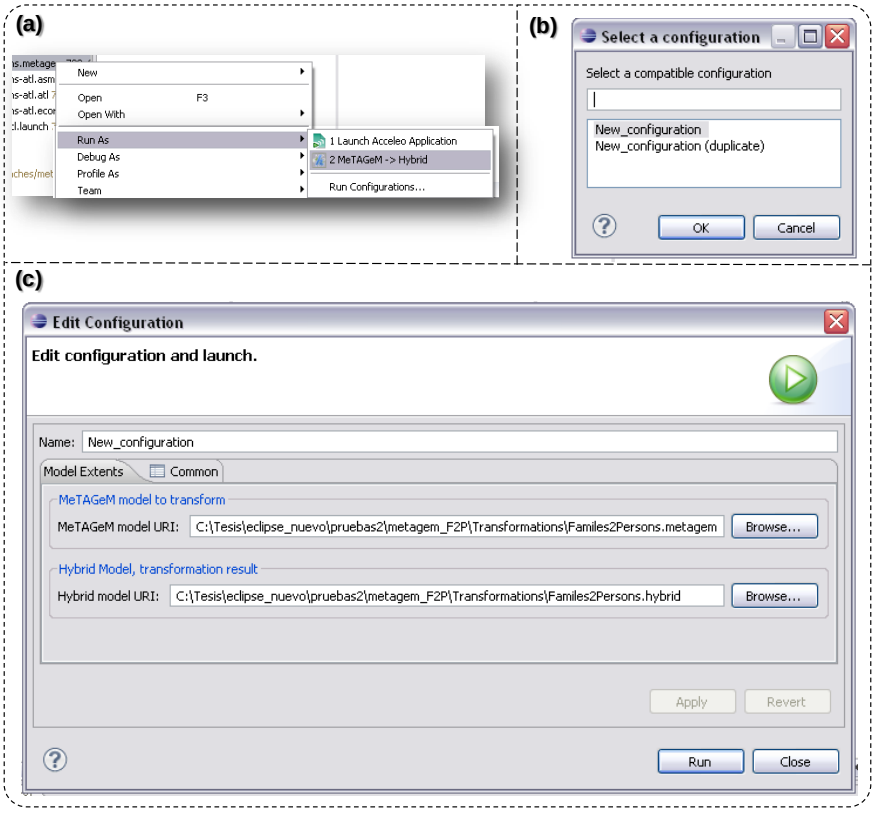

Figura 5-62. Posibles comportamientos del lanzador de transformaciones: a) selección de configuración ya establecida; b) creación de una nueva configuración

La pestaña de la configuración de la transformación es implementada en una nueva clase en la que se crea la configuración de lanzamiento y se definen los componentes visuales de la pestaña (botones, campos de texto, etiquetas, etc.). Además, cada vez que se realiza un cambio en la configuración se comprueba la existencia del modelo de entrada, que su extensión sea la correcta y que se trata de un modelo válido, es decir, cumple con las reglas de validación. Si no se cumplen todas estas condiciones, el usuario recibirá por pantalla el mensaje mostrado en la Figura 5-63. Del mismo modo, si no se ha seleccionado la ubicación del modelo destino, también se muestra un error por pantalla.

Cuando el usuario pulsa sobre el botón “Run” de la ventana de configuración, se asume automáticamente que la configuración a ejecutar es la definida, es decir se vuelve al mismo punto que si se ejecutara la implementación de la extensión *launchShortcuts* con una sola configuración para el modelo de entrada. En este punto, en primer lugar se ejecuta automáticamente la validación

del modelo de entrada y si se trata de un modelo válido, se ejecuta la transformación.

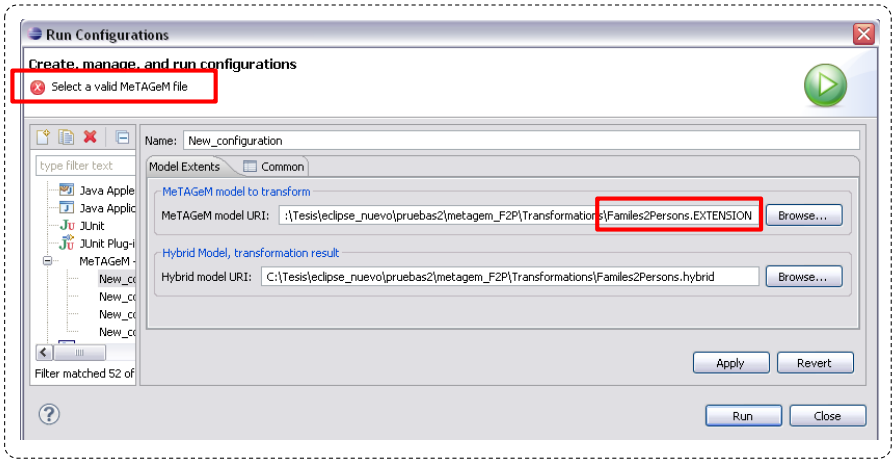

Figura 5-63. Mensaje de error en la ventana de configuración

Para llevar a cabo la validación de los modelos, se ha implementado la clase *ValidationExecution*. Esta clase contiene un método *isValid* que recibe el modelo origen, el metamodelo origen y el metamodelo destino (en caso de ser necesario). A partir de esta información crea una instancia de un módulo de EVL, ejecuta el proceso de validación y posteriormente, analiza el resultado obtenido. En caso de no cumplir con alguna de las reglas de validación implementadas, el usuario recibirá un mensaje por pantalla como muestra la Figura 5-64.

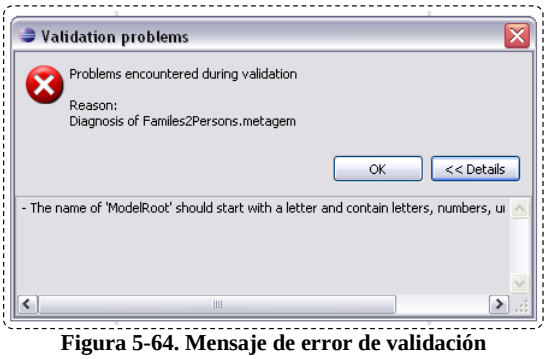

Figura 5-64. Mensaje de error de validación

Una vez comprobado que el modelo origen es válido respecto de las reglas de validación implementadas en EVL, se procede a ejecutar la transformación configurada.

El principal elemento a la hora de ejecutar las transformaciones en MeTAGeM-Trace es la clase *Transformations*. Una de las principales responsabilidades de esta clase es obtener los diferentes módulos de las transformaciones que soporta MeTAGeM-Trace. Dado que se trata de una tarea costosa en términos de memoria y tiempo de procesamiento, la clase *Transformations* ha sido implementada siguiendo el patrón *singleton* [71] para evitar duplicar la carga de los metamodelos que intervienen en la transformación. De esta forma, el proceso de carga de las transformaciones se realiza la primera vez y las transformaciones permanecen disponibles hasta el final de la ejecución de la herramienta.

Para ejecutar las transformaciones de modelos implementadas en MeTAGeM-Trace, esta clase contiene un método por cada una de estas transformaciones (*metagem2hybrid, hybrid2atl* y *hybrid2etl*). Disponer de un método por cada transformación permite que dichos métodos tan solo reciban el *path* de los modelos de origen y destino, dado que los metamodelos siempre serán los mismos en la ejecución de cada uno de estos métodos.

Para llevar a cabo la ejecución de una transformación, en primer lugar, a partir de la ejecución de un método *loadModelsXXX* (siendo XXX el nombre de la transformación, por ejemplo *loadModelsMeTAGeM2Hybrid*), se cargan los modelos, de acuerdo a sus metamodelos, en una estructura de tipo *Map*. Posteriormente, se crea una instancia de la máquina virtual de ATL específica para modelos EMF en la que se cargan los modelos. Finalmente, tras introducir las opciones de la ejecución, se procede a la ejecución de la transformación, cuyo resultado es almacenado en el modelo destino.

A modo de ejemplo, en la Figura 5-65 se presentan los métodos que permiten la ejecución programática de la transformación *metagem2hybrid*.

```
public void metagem2hybrid(String inFilePath, String outFilePath)
      throws Exception {

      try{
            Map<String, Object> models=loadModelsMeTAGeM2Hybrid(inFilePath);
            doMeTAGeM2Hybrid(models,new NullProgressMonitor());
            saveModels(((IModel)models.get("OUT")),outFilePath);
      }catch(Exception e){
            e.printStackTrace();
      }
}

private void doMeTAGeM2Hybrid(Map<String, Object> models,
      NullProgressMonitor nullProgressMonitor) throws Exception {

      ILauncher launcher = new EMFVMLauncher();
      Map<String, Object> launcherOptions = getOptions(_METAGEM2HYBRID);
      launcher.initialize(launcherOptions);
      launcher.addInModel(((IModel)models.get("IN")), "IN", "MeTAGeM");
      launcher.addOutModel(((IModel)models.get("OUT")), "OUT", "Hybrid");
      launcher.launch("run", nullProgressMonitor, launcherOptions, (Object[])
                                          getModulesList(_METAGEM2HYBRID));
}

private Map<String, Object> loadModelsMeTAGeM2Hybrid(String inFilePath)
      throws Exception {

      Map<String, Object> models = new HashMap<String, Object>();
      ModelFactory factory = new EMFModelFactory();
      IInjector injector = new EMFInjector();
      IReferenceModel hybridMetamodel = factory.newReferenceModel();
      injector.inject(hybridMetamodel,
                  Utils.getFileURL("resources/Hybrid.ecore").toString());
      IReferenceModel metagemMetamodel = factory.newReferenceModel();
      injector.inject(metagemMetamodel,
                  Utils.getFileURL("resources/MeTAGeM.ecore").toString());
      IModel metagemInputModel = factory.newModel(metagemMetamodel);
      injector.inject(metagemInputModel, inFilePath);
      models.put("IN", metagemInputModel);

      IModel hybridOutputModel = factory.newModel(hybridMetamodel);
      models.put("OUT", hybridOutputModel);
      return models;
}
```

Figura 5-65. Métodos para la ejecución programática de la transformación *metagem2hybrid*

5.3.7.4 Lanzamiento de Generación de Código

Ya que para la integración de los generadores de código no es necesario definir una configuración de lanzamiento, se ha empleado otro tipo de interfaz. En concreto se han definido entradas en el menú contextual que aparece sobre los modelos PDM (ATL y ETL) como muestra la Figura 5-66.

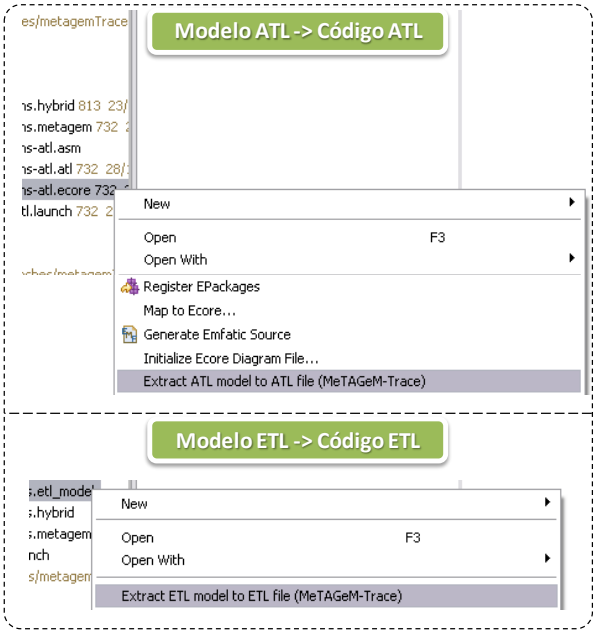

Figura 5-66. Entradas del menú contextual para generar el código de la transformación

Para implementar cada una de estas entradas de menú contextual es necesario definir un punto de extensión sobre la extensión *org.eclipse.ui.popupMenus* en el fichero *plugin.xml* como muestra la Figura 5-67.

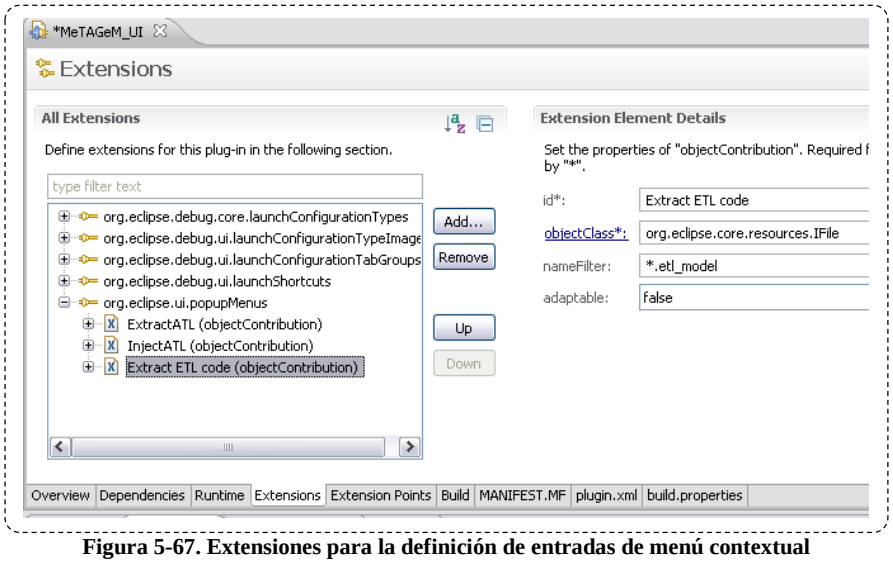

Figura 5-67. Extensiones para la definición de entradas de menú contextual

Cada elemento de tipo *objectContribution* tiene asociada una acción (*action*) que se implementa mediante una clase Java: *ATLExtract* y *ETLExtract*.

La primera de ellas, contiene un método *ExtractATL* que carga el modelo de la transformación ATL seleccionado por el usuario, crea una instancia del *parser* TCS que proporciona el *framework* de ATL y ejecuta el proceso de extracción del código. Dicho código es almacenado en un fichero con el mismo nombre que el modelo de entrada, pero cambiando su extensión por *.atl*. La clase *ETLExtract* proporciona la misma funcionalidad, pero para ello emplea el *parser* genérico que proporciona el *framework* de MOFScript, el lenguaje empleado para implementar las reglas de generación de código. El código Java que implementan estas clases puede consultarse en el CD adjunto a esta tesis.

Validación

En los capítulos anteriores se ha presentado el entorno MeTAGeM-Trace para el desarrollo de transformaciones de modelos que incluyen de forma embebida un generador de modelos de trazas. Para la definición de dicho entorno se ha presentado por un lado, la propuesta metodológica y por otro lado, la herramienta que da soporte tecnológico a dicha propuesta metodológica.

En este capítulo se presenta el proceso que permite comprobar la validez del entorno MeTAGeM-Trace. Para ello, se ha seguido el método de validación presentado en la sección 2.4 de esta tesis doctoral, correspondiente al empleo de **casos de estudio de laboratorio**. Así, en las siguientes secciones se detallarán las decisiones tomadas en la definición del protocolo de validación (sección 6.1), la ejecución de los casos de estudio seleccionados (secciones 6.2 y 6.3). Por último, se realiza una discusión acerca de los datos obtenidos y se presentan las conclusiones del proceso de validación (sección 6.4).

6.1 Protocolo de Validación

De acuerdo al método de validación presentado en la sección 2.4, el primer paso del proceso de validación es definir los casos de estudio a emplear y un protocolo para la recogida de datos. Esta información es descrita en las siguientes sub-secciones.

6.1.1 Definición de los Casos de Estudio

Como ya se ha indicado en la introducción de esta tesis doctoral (sección 1.3), el entorno MeTAGeM-Trace forma parte de MIDAS, una metodología para el desarrollo dirigido por modelos de sistemas de información [49, 50, 131, 204, 209]. Hasta el momento, se han propuesto dos herramientas para ofrecer soporte tecnológico a MIDAS: M2DAT y MeTAGeM-Trace (presentada en esta tesis doctoral). M2DAT es presentada como una herramienta MDA que soporta cada uno de los métodos propuestos por MIDAS, aunque hasta el momento solo se centra en el módulo de contenido de MIDAS (M2DAT-DB) [199]. MeTAGeM-Trace ha sido desarrollada, entre otras cuestiones, para el desarrollo de las transformaciones embebidas en los módulos de M2DAT y para generar las trazas del proceso de desarrollo de sistemas de información.

Teniendo en cuenta este contexto, para validar el entorno MeTAGeM-Trace se propone el desarrollo de un **meta-caso de estudio** para la implementación de un subconjunto de las reglas de transformación de uno de los

módulos propuestos en M2DAT-DB [199]. Dado que M2DAT-DB ya ha sido implementada por completo [199], se dispone de una implementación de referencia completa y validada de cada uno de los módulos de transformación contenidos en la herramienta. Por ello, será posible comparar y evaluar los resultados obtenidos con MeTAGeM-Trace. En concreto, de las transformaciones contenidas en M2DAT-DB, se ha escogido el módulo de transformación que permite obtener esquemas XML a partir del modelo conceptual en UML (**UML2XMLSchema**).

Como resultado de la ejecución de dicho meta-caso de estudio se obtendrá una transformación entre los metamodelos UML (origen) y XML (destino), que contiene de forma embebida un generador de trazas para la creación de un modelo de trazas entre los elementos de los modelos UML y XML participantes en la transformación. Para comprobar que dicha transformación generada proporciona la funcionalidad requerida, se ha optado por emplear otro **caso de estudio** adicional.

El caso de estudio que se utiliza para probar las transformaciones implementadas se ha tomado de [65]. Concretamente, se trata de la construcción de una **base de datos de películas online** (*OMDB, Online Movie Database*). Este caso de estudio proporciona dos ventajas principales al proceso de validación que se está realizando. Por un lado, se trata de un caso "externo" al contexto de la propuesta que se desea validar y de esta manera, se evita que el caso de estudio se construya de acuerdo a las características del entorno a validar. Por otro lado, dicho caso de estudio fue empleado en la validación de las transformaciones contenidas en la herramienta M2DAT-DB [199], lo que hace posible comparar los resultados obtenidos en este proceso de validación.

Dado que para la implementación de la herramienta M2DAT-DB se ha empleado ATL como lenguaje de transformación de modelos, estos casos de estudio se centran en validar el entorno MeTAGeM-Trace para la generación de transformaciones ATL.

Así mismo, es necesario destacar que al margen de esta validación, se ha realizado un conjunto de pruebas unitarias durante el desarrollo de la herramienta para probar y asegurar el funcionamiento del entorno para la generación de transformaciones ETL.

6.1.2 Protocolo para la Recogida de Datos

El siguiente paso del protocolo de validación consiste en la definición del protocolo para la recogida de datos. De acuerdo con Yin [223], este protocolo debe proporcionar una visión global del proyecto de los casos de estudio, haciendo

uso de múltiples fuentes de información, siempre que sea posible y definir un conjunto de cuestiones que se deben resolver con la ejecución de cada caso de estudio. Así, en las siguientes sub-secciones se proporciona una visión global de cada uno de los casos de estudio seleccionados y se establece un conjunto de preguntas a las que se dará respuesta en secciones posteriores.

6.1.2.1 Meta-caso de estudio UML2XMLSchema

M2DAT-DB es una herramienta para el desarrollo dirigido por modelos de esquemas de Bases de Datos (BD) modernas. M2DAT-DB ofrece soporte para el desarrollo de BD, desde la definición de los modelos a nivel PIM hasta la generación del código final. En particular, a partir de un modelo conceptual de datos definido a nivel PIM y representado por medio de un diagrama de clases de UML [33, 156], permite la generación de esquemas de Bases de Datos Objeto-Relacionales (para el producto comercial Oracle 10g [79, 205] y el estándar SQL:2003 [91]) y esquemas XML [36, 157, 208, 209, 210, 216] a nivel PSM y posteriormente, la generación del código que implementa dicha BD.

Para obtener los modelos PSM (Oracle, SQL:2003 y XMLSchema) a partir de los modelos PIM (diagrama de clases UML), M2DAT-DB implementa, mediante el lenguaje de transformación ATL, tres módulos de transformación de modelo a modelo, uno por cada DSL propuesto a nivel PSM: UML2ORA (UML a Oracle 10g), UML2SQL2003 (UML a SQL:2003) y UML2XMLSchema (UML a XMLSchema). En la Figura 6-1 se presenta el proceso de desarrollo de BD que proporciona la herramienta M2DAT-DB.

Como se ha indicado anteriormente, de las transformaciones de modelos que forman parte de la herramienta M2DAT-DB, para la validación del entorno MeTAGeM-Trace se ha decidido desarrollar el módulo de transformación UML2XMLSchema.

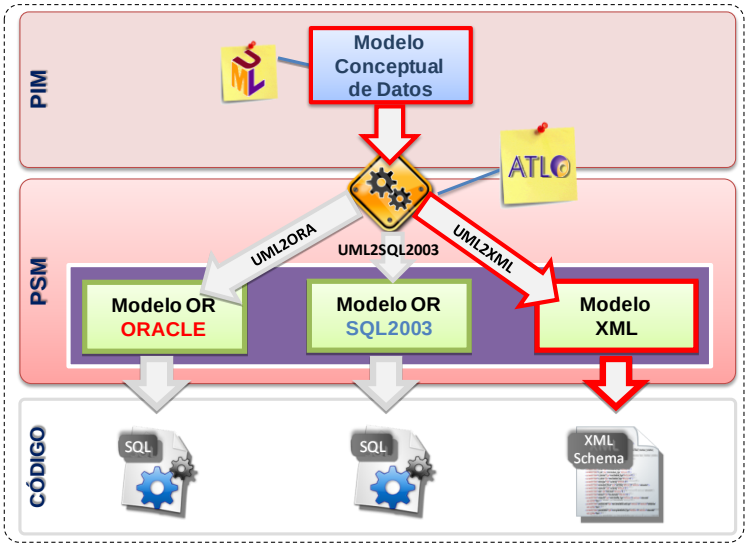

Figura 6-1. Proceso de Desarrollo de Bases de Datos soportado por M2DAT-DB

Una vez que se ha presentado una visión global del meta-caso de estudio que se va a desarrollar, se definen las siguientes cuestiones a las que se deberá dar respuesta a partir de este meta-caso de estudio (CMCE, Cuestión del Meta-Caso de Estudio):

- **CMCE-1.** ¿Ha sido posible completar el desarrollo de la transformación hasta obtener una transformación en código ATL?
- **CMCE-2.** Durante el proceso de desarrollo dirigido por modelos, ¿el usuario ha tenido que modificar manualmente los modelos generados (PSM y PDM)?
- **CMCE-3.** ¿El código de la transformación obtenido es totalmente ejecutable? Si no es así, ¿qué elementos no se han generado (semi-)automáticamente?
- **CMCE-4.** En cuanto al número de líneas de código obtenidas, ¿es menor, igual o mayor al de la transformación UML2XMLSchema contenida en M2DAT-DB? Es necesario recordar que la transformación incluida en la herramienta M2DAT-DB no proporciona generación de trazas.

En cuanto a la recolección de datos que permitirá obtener un conjunto de evidencias para dar respuesta a las preguntas anteriores se ha decidido emplear como fuentes de información: la observación directa del desarrollo del meta-caso de estudio, el código de la implementación de la herramienta M2DAT-DB y la tesis doctoral en la que se presenta dicha herramienta [199].

6.1.2.2 Caso de estudio OMDB (UML2XMLSchema)

El desarrollo del meta-caso de estudio UML2XMLSchema debe proporcionar como resultado una transformación entre los metamodelos UML y XMLSchema que incluye de forma embebida un generador de trazas. Con el objetivo de validar la funcionalidad de dicha transformación y del generador de trazas, se ha decidido desarrollar un caso de estudio que requiera la ejecución de dicha transformación. El caso de estudio seleccionado se corresponde al desarrollo dirigido por modelos de una BD para la gestión de un catálogo de películas online (OMDB, *Online Movie DataBase* [65]).

La BD OMDB está diseñada para manejar información referente a películas, actores, directores, guionistas y en general, información relacionada con las películas. Los usuarios pueden acceder de forma *online* a esta información en la página web de OMDB y comprar productos (por ejemplo, videos de películas, DVDs, libros, CDs y otros productos relacionados con las películas).

De cada una de las películas almacenadas en la BD se registra su título, director, sitio web oficial, género, estudio, una breve sinopsis y el reparto de actores. Además, cada película tiene un máximo de cinco comentarios de editores externos, y el número ilimitado de comentarios de los usuarios. En la web OMDB se ofrecen productos para la venta incluyendo vídeos de películas y DVDs. De estos productos se almacena su información más relevante como el título, la categoría, el precio de lista, fecha de lanzamiento, etc.

Una vez que se ha presentado una visión global del caso de estudio que se va a desarrollar, se definen las siguientes cuestiones a las que se deberá dar respuesta a partir del desarrollo de este caso de estudio (CCE, Cuestión del Caso de Estudio):

- **CCE-1.** ¿La transformación generada mediante el meta-caso de estudio se ejecuta correctamente sobre el modelo *OMDB.uml*?
- **CCE-2.** ¿El modelo *OMDB.xmlschema* que se ha obtenido es igual al modelo obtenido en el proceso de validación de la herramienta M2DAT-DB? Si no son iguales, ¿qué diferencias existen?
- **CCE-3.** En cuanto al tiempo de ejecución de la transformación obtenida como resultado del meta-caso de estudio para generar el modelo *OMDB.xmlschema*, ¿es menor, igual o mayor que ejecutando la transformación contenida en M2DAT-DB?
- **CCE-4.** ¿Se ha generado un modelo que contiene las trazas entre los modelos que describen OMDB a nivel PIM (*OMDB.uml*) y nivel PSM (*OMDB.xmlschema*)?

En cuanto a la recolección de datos que permitirá obtener un conjunto de evidencias para dar respuesta a las preguntas anteriores se ha decidido emplear como fuentes de información: la observación directa del desarrollo del caso de estudio, el código implementado para la validación de la herramienta M2DAT-DB, la tesis doctoral en la que se aplica este caso de estudio a la herramienta M2DAT-DB [199] y el estudio del cual se ha obtenido el caso de estudio [65].

6.2 Diseño y Ejecución del Meta-Caso de Estudio

Como se ha mencionado anteriormente, el meta-caso de estudio seleccionado para la validación del entorno MeTAGeM-Trace consiste en usar este entorno para el desarrollo de la transformación UML2XMLSchema, que forma parte de la implementación de la herramienta M2DAT-DB [199]. El desarrollo de este **meta-caso de estudio de laboratorio** se lleva a cabo por el doctorando en un entorno controlado.

Mediante el desarrollo de este meta-caso de estudio se comprueba el funcionamiento y la aplicabilidad del entorno MeTAGeM-Trace en un desarrollo "externo". En otras palabras, el objetivo es evaluar si MeTAGeM-Trace produce transformaciones de modelos correctas que puedan aplicarse en otros desarrollos que impliquen la ejecución de transformaciones de modelos.

Como se describe en la Figura 6-2, el proceso de desarrollo de las reglas de la transformación UML2XMLSchema consiste en el modelado a nivel PIM de las relaciones de transformación entre los elementos del metamodelo de UML y los elementos del metamodelo XMLSchema. Debido al gran tamaño de ambos metamodelos, se ha decidido no incluirlos en esta memoria de tesis. No obstante, la implementación de estos metamodelos se puede consultar en el CD adjunto.

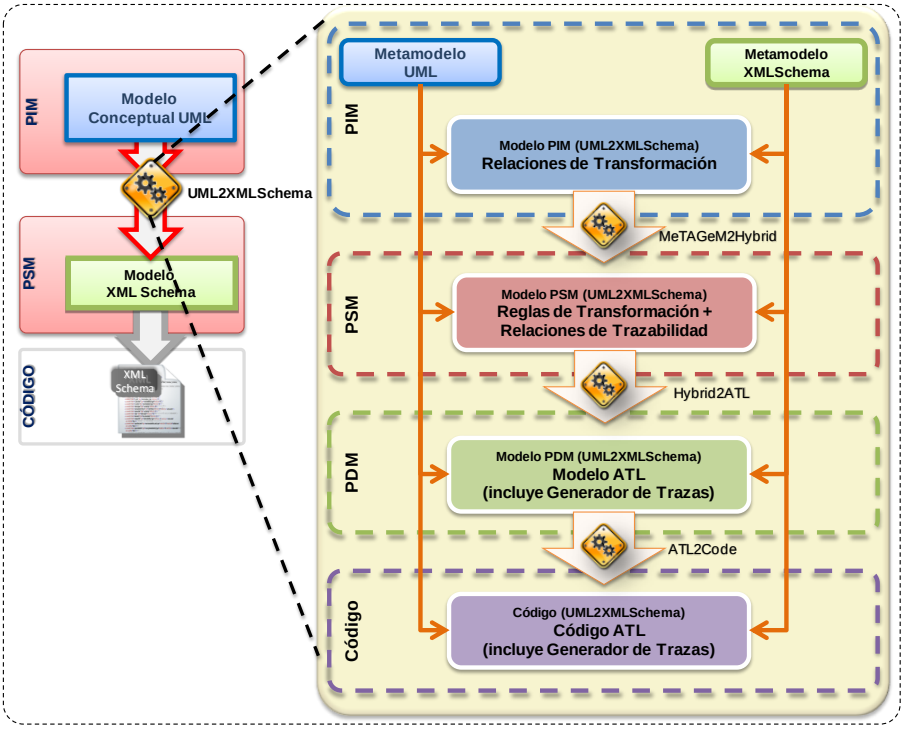

Figura 6-2. Proceso de Desarrollo de la Transformación *UML2XMLSchema* con MeTAGeM-Trace

A partir del modelo de transformación a nivel PIM y ejecutando la transformación *MeTAGeM2Hybrid*, proporcionada por el entorno MeTAGeM, se obtiene el modelo de la transformación a nivel PSM que sigue la aproximación híbrida. De la misma manera, a partir del modelo de transformación de nivel PSM y aplicando la transformación *Hybrid2ATL*, se obtiene el modelo de transformación a nivel PDM conforme al lenguaje de ATL. Por último, por medio de los mecanismos de extracción de ATL, se obtiene el código que implementa la transformación en dicho lenguaje.

6.2.1 Especificación de las Relaciones de Transformación

Para llevar a cabo el desarrollo de la transformación UML2XMLSchema mediante MeTAGeM-Trace, el primer paso consiste en la definición de las relaciones de transformación entre los elementos de los metamodelos de UML (origen) y de XMLSchema (destino). Para ello, en la Tabla 6-1 se muestran las principales relaciones de transformación entre los elementos existentes en la transformación UML2XMLSchema.

Tabla 6-1. Relaciones de Transformación en UML2XMLSchema

Metamodelo UML (PIM)	**Metamodelo XMLSchema (PSM)**
Model	*Schema*
Package	*Schema*
Class *(mapTo==Sequence and not getGeneralization)*	*ElementGlobal*
	ComplexTypeLocal
	Other
	Sequences
Class *(getGeneralization)*	*ElementGlobal*
	ComplexTypeLocal
	ComplexContent
	Extension ComplexContent
	Choice
Class *(mapTo==Choice and not getGeneralization)*	*ElementGlobal*
	ComplexTypeLocal
	Other
	Choice
Class *(mapTo==All and not getGeneralization)*	*ElementGlobal*
	ComplexTypeLocal
	Other
	All
Property (contenido en un elemento *Class*)	*ElementLocal*
Association (*isNM and not isAgregation and not isComposite*)	*ElementLocal* - *minOccurs:N* - *maxOccurs: N*
Association (*is1N and not isAgregation and not isComposite*)	*ElementLocal* - *minOccurs:1* - *maxOccurs: N*
Association (*is11 and not isAgregation and not isComposite*)	*ElementLocal* - *minOccurs:1* - *maxOccurs: 1*
Association (*isAgregation*)	*ElementLocal*

Metamodelo UML (PIM)	Metamodelo XMLSchema (PSM)
Association (*isComposite and not isAgregation and mapTo==Sequences*)	*ElementLocal*
	ComplexTypeLocal
	Other
	Sequences
	ElementLocal
Association (*isComposite and not isAgregation and mapTo==All*)	*ElementLocal*
	ComplexTypeLocal
	Other
	All
	ElementLocal

A continuación, como se muestra en la siguiente sección, se procede a implementar las relaciones anteriores en términos del modelo PIM que define MeTAGeM-Trace.

6.2.2 Ejecución del Meta-Caso de Estudio con MeTAGeM-Trace

Una vez especificadas las relaciones de transformación (Tabla 6-1), se procede al desarrollo de la transformación con la herramienta MeTAGeM-Trace. Para ello, se deben seguir los pasos indicados en la Figura 6-2:

1. Crear modelo PIM (modelo MeTAGeM).
2. Generación del modelo PSM (modelo *hybrid*), a partir de la ejecución de la transformación *MeTAGeM2Hybrid.*
3. Generación del modelo PDM (modelo *ATL*) a partir de la ejecución de la transformación *MeTAGeM2ATL*.
4. El último paso es la generación del código que implementa la transformación.

Es necesario mencionar que cada uno de los artefactos resultantes de las transformaciones (los modelos PSM y PDM y el código de la transformación) pueden ser refinados manualmente por el desarrollador.

A continuación, en las siguientes sub-secciones, se detallan cada uno de los pasos descritos.

6.2.2.1 Definición del Modelo PIM

Para definir el modelo de la transformación UML2XMLSchema a nivel PIM se debe crear un nuevo modelo MeTAGeM empleando, para ello, el asistente de creación de modelos implementado en MeTAGeM-Trace (sección 5.3.3.).

Una vez creado el modelo vacío, se deben modelar en él todas las relaciones descritas en la Tabla 6-1 así como sus relaciones internas, en términos de elementos del metamodelo de MeTAGeM (Figura 5-3). Así, por ejemplo, la relación de transformación entre un elemento origen *Model* y un elemento destino *Schema*, se debe modelar mediante una relación de tipo *OneToOne* (uno-a-uno) que a su vez, contiene otra relación *OneToOne* que define que el atributo *id* del *Schema* se generará a partir del atributo *name* del elemento *Model*. De la misma forma, las relaciones cuyo elemento origen es *Class* deben ser definidas mediante relaciones *OneToMany*, ya que a partir de este elemento origen, se obtienen varios elementos destino. En concreto, en este caso se obtienen cuatro o cinco elementos destino, dependiendo de si es el elemento *Class* representa una generalización y del tipo de elemento al que debe mapearse (*mapTo*).

En la Figura 6-3 se muestra el modelo de transformación de nivel PIM definido. Como se puede observar, en la parte izquierda se muestran los elementos del metamodelo origen (UML), en la parte derecha los elementos del meta-modelo destino (XMLSchema), y en el centro, el modelo de la transformación donde se han especificado todas las relaciones.

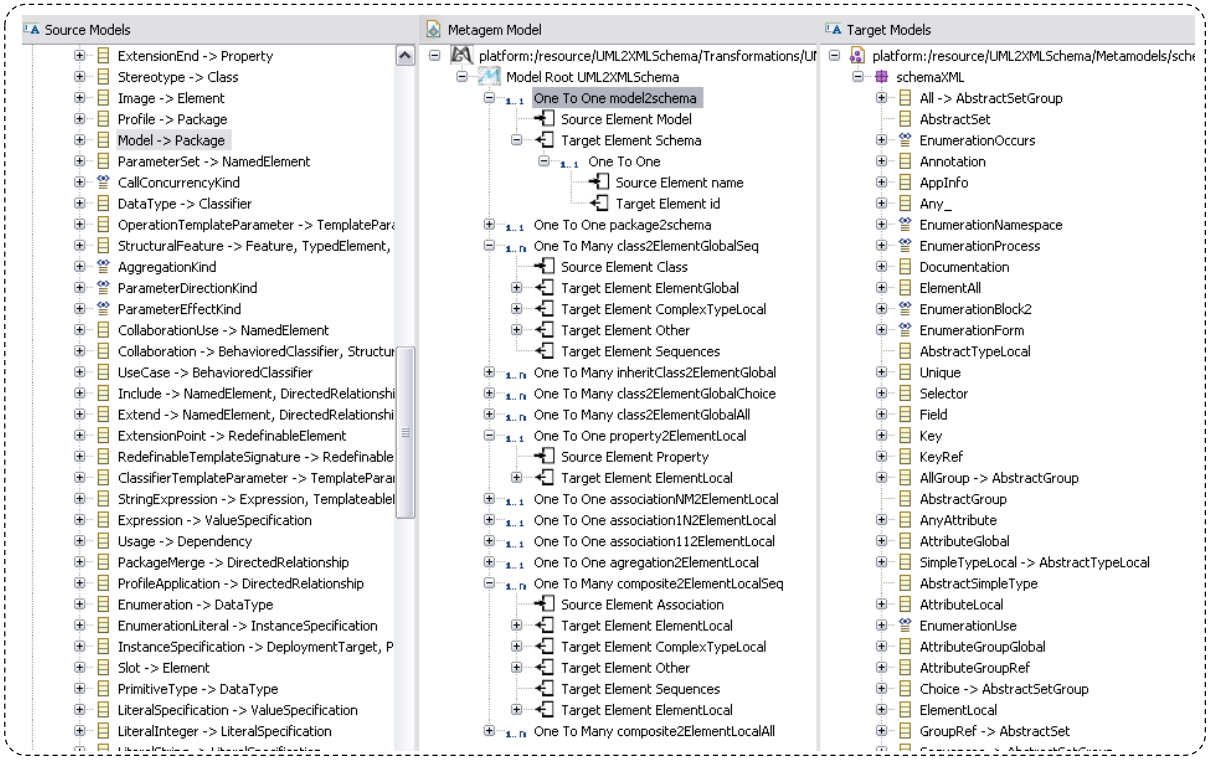

Figura 6-3. Modelo de Transformación a nivel PIM (UML2XMLSchema)

6.2.2.2 Definición del Modelo PSM

Una vez que se ha definido por completo el modelo que describe la transformación a nivel PIM, se puede generar de forma (semi-)automática, usando las facilidades que ofrece la herramienta MeTAGeM-Trace, el modelo PSM de la transformación de acuerdo a las características de la aproximación híbrida. Como se ha indicado en el capítulo anterior, además de generar las reglas de la transformación, se generan las relaciones de trazabilidad identificadas automáticamente a partir del modelo PIM.

El modelo PSM puede ser modificado manualmente por el usuario para adaptarlo a las necesidades concretas del desarrollo. Así, puede modificar las reglas de transformación y sus elementos origen y destino, las relaciones de trazabilidad, crear operaciones en el ámbito de la transformación, etc.

En el desarrollo de la transformación *UML2XMLSchema* ha sido necesario incluir un conjunto de operaciones que se deben ejecutar en el ámbito de la transformación. Para definir estas operaciones se ha consultado directamente la implementación de la transformación contenida en la herramienta M2DAT-DB. Además, se han modificado diversos elementos de tipo *RightPattern* para establecer llamadas a dichas operaciones así como para establecer valores concretos que se asignarán al elemento destino (atributos *minOccurs* y *maxOccurs* de elementos *ElementLocal*).

En la Figura 6-4 se muestra parte del modelo PSM final que describe la transformación *UML2XMLSchema*.A modo de ejemplo, al igual que en la figura del modelo PIM, se muestra la relación *model2schema*.

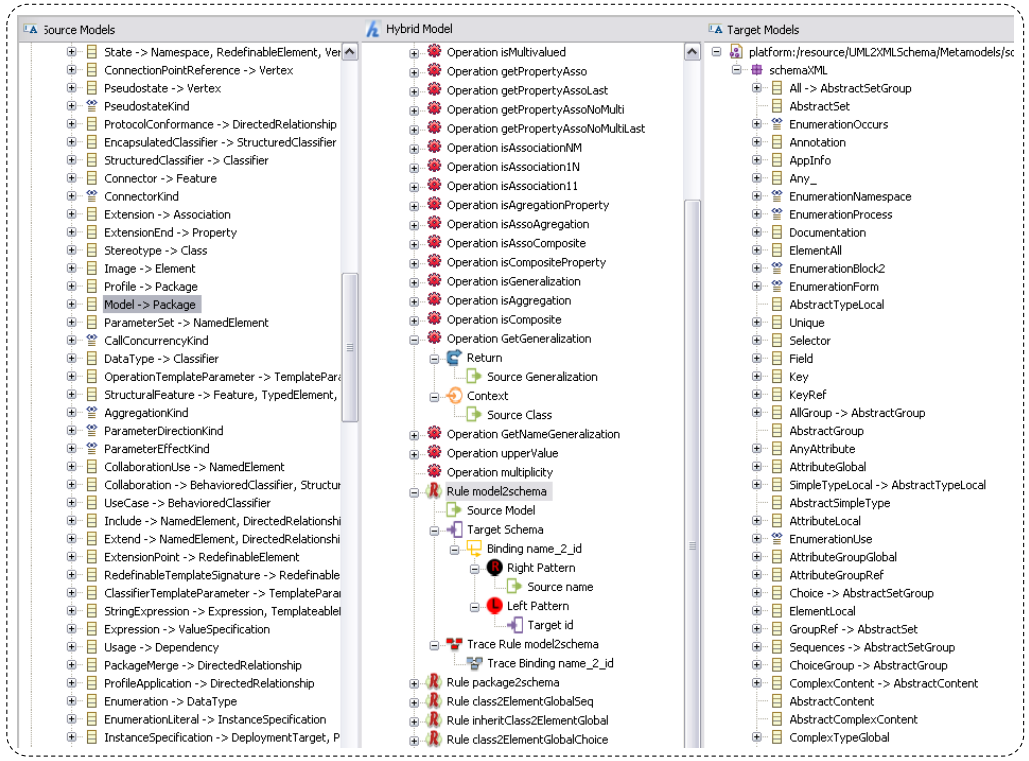

Figura 6-4. Modelo de Transformación a nivel PSM (UML2XMLSchema)

6.2.2.3 Definición del Modelo PDM

Una vez que se ha refinado el modelo PSM, es posible obtener (semi-)automáticamente el modelo de la transformación a nivel PDM, de acuerdo a los metamodelos que describen los lenguajes de transformación ATL y ETL. Como se ha comentado anteriormente, la implementación de referencia con la que se compararán los resultados de este proceso de validación se encuentra implementada en ATL. Por este motivo, en este caso, se generará el modelo ATL (extensión *.atl.ecore*).

Para obtener dicho modelo, se ejecuta la transformación *Hybrid2ATL*, embebida en la herramienta MeTAGeM-Trace. El modelo obtenido puede ser modificado por el desarrollador de la transformación, para ello, se puede emplear el editor reflexivo que proporciona EMF. En este caso, no ha sido necesario refinar el modelo ATL obtenido. En la Figura 6-5 se muestra parte del modelo de la transformación conforme al metamodelo de ATL.

Figura 6-5. Modelo de Transformación a nivel PDM: ATL (UML2XMLSchema)

En la figura anterior, una vez más, puede verse la relación *model2schema*. Si bien en el modelo PSM era más compleja que en su definición a nivel PIM, en

el caso del modelo PDM, el tamaño y complejidad de la relación aumenta de forma muy notable. A la vista de esta información, como se defiende en esta tesis doctoral, parece recomendable emplear modelos de alto nivel para desarrollar las transformaciones que incluyen generación de trazas.

6.2.2.4 Generación de Código

El siguiente paso en el proceso de desarrollo de la transformación *UML2XMLSchema* usando MeTAGeM-Trace es generar el código de la transformación a partir del modelo PDM. Dado que el modelo PDM es descrito en términos del metamodelo del lenguaje ATL, el código que implementa la transformación será generado de acuerdo a la sintaxis de dicho lenguaje (fichero *.atl*).

Como se ha indicado en varias ocasiones a lo largo de esta memoria de tesis doctoral, para realizar la generación de código, la herramienta MeTAGeM-Trace proporciona una entrada en el menú contextual de los modelos ATL. Dicha entrada ejecuta la extracción de código proporcionada por el *plug-in* de ATL para el entorno Eclipse. Por cada elemento definido en el modelo de la transformación conforme al metamodelo de ATL (nivel PDM), se genera el código que lo implementa.

En la Figura 6-6 se muestra un fragmento del código de la transformación en el lenguaje ATL que se ha obtenido como resultado del proceso de desarrollo de la transformación, utilizando MeTAGeM-Trace. En concreto, en esta figura se presenta el código que implementa la regla de transformación *model2schema*. El resto del código de la transformación *UML2XMLSchema* puede consultarse en el CD adjunto.

Con el objetivo de comparar el código generado por la herramienta MeTAGeM-Trace con el código de la transformación implementado de forma manual, en la Figura 6-7 se presenta la regla de transformación *model2schema* implementada manualmente, incluida en la herramienta M2DAT-DB.

```
-- @atlcompiler atl2006
module UML2XMLSchema;
create schemaXML_model : schemaXML, in2out_trace : TRACE from UML_model : UML,
AnnotationMetamodel_model : AnnotationMetamodel;

-- Comments -> This is a MatchedRule: model2schema -> do{thisModule.model;}
rule model2schema {
 from
  model_in : UML!Model
 to
  schema_out : schemaXML!Schema (
   id <- model_in.name
  ),
  model2schema_TL1 : TRACE!TraceLink (
   name <- 'model2schema',
   traceModel <- thisModule.getTraceModelRoot,
   operation <- #Transform,
   source <- Sequence {model_in_Trace_TE1},
   target <- Sequence {schema_out_Trace_TE35},
   childLinks <- Sequence {name_2_id_TL14}
  ),
  name_2_id_TL14 : TRACE!TraceLink (
   name <- 'name_2_id',
   operation <- #Transform,
   source <- Sequence {name_in_Trace_TE14},
   target <- Sequence {id_out_Trace_TE69}
  ),
  model_in_Trace_TE1 : TRACE!SourceElement (
   name <- model_in.getName(),
   ref <- model_in.__xmiID__,
   model <- thisModule.getModel_UML_model
  ),
  name_in_Trace_TE14 : TRACE!SourceElement (
   name <- model_in.name.toString() + '(Feature: name)',
   belongsTo <- model_in_Trace_TE1,
   model <- thisModule.getModel_UML_model
  ),
  schema_out_Trace_TE35 : TRACE!TargetElement (
   name <- schema_out.getName(),
   ref <- schema_out.__xmiID__,
   model <- thisModule.getModel_schemaXML_model
  ),
  id_out_Trace_TE69 : TRACE!TargetElement (
   name <- schema_out.id.toString() + '(Feature: id)',
   belongsTo <- schema_out_Trace_TE35,
   model <- thisModule.getModel_schemaXML_model
  )
 -- ActionBlock:
 do {
  thisModule.model;
 }
}
```

Figura 6-6. Código ATL de la regla *model2schema* generada con MeTAGeM-Trace (UML2XMLSchema)

```
module UML2XMLW; -- Module Template
create OUT : schemaXML from IN : UML, AMW : AnnotationMetamodel;

rule model2schema {
      from
            c: UML!Model
      to
            xml: schemaXML!Schema (
                  id<-c.name)
      do {thisModule.model;}

}
```

Figura 6-7. Código ATL de la regla *model2schema* implementada manualmente para M2DAT-DB (UML2XMLSchema)

Como se puede ver, la regla de transformación generada con MeTAGeM-Trace es mucho mayor que la implementación manual. En este punto, es necesario recordar que MeTAGeM-Trace proporciona un generador de trazas en cada transformación, de forma que además de generar los modelos destino, se genera un modelo que contiene las trazas entre los elementos de los modelos origen y los elementos de los modelos destino. Teniendo en cuenta esta información, se puede comprobar que el código de la Figura 6-7 se encuentra implementado de forma muy similar en el proporcionado por la herramienta. Por tanto, la principal diferencia entre ambas implementaciones radica en la generación de las trazas.

Por todo ello, se puede afirmar que incluir generadores de trazas en la codificación de las transformaciones aumenta la complejidad de la misma, de forma considerable. En consecuencia, aumentaría proporcionalmente el esfuerzo de los desarrolladores si tuvieran que realizar esta implementación de forma manual.

6.3 Diseño y Ejecución del Caso de Estudio

Mediante el desarrollo del meta-caso de estudio se ha obtenido el código que implementa la transformación *UML2XMLSchema*, incluyendo el generador de trazas. Con el objetivo de probar si dicha transformación proporciona la funcionalidad requerida, se procede al desarrollo de un caso de estudio que requiera la ejecución de dicha transformación. En concreto, como se ha comentado en la sección 6.1.1, se ha optado por desarrollar el caso de estudio OMDB [65], empleado en la validación de las transformaciones contenidas en la herramienta M2DAT-DB [199]. Así, será posible comparar los resultados de la ejecución de la transformación desarrollada con MeTAGeM-Trace con los resultados de ejecutar

la transformación implementada de forma manual. Al igual que en el meta-caso de estudio, el desarrollo de este **caso de estudio de laboratorio** se lleva a cabo por el doctorando en un entorno controlado.

Con el desarrollo de este caso de estudio se comprueba si las transformaciones generadas con la herramienta MeTAGeM-Trace proporcionan el código adecuado para generar correctamente los modelos destino.

Para desarrollar este caso de estudio, se parte del modelado conceptual de la base de datos OMDB a nivel PIM (fichero *OMDB.uml*). Este modelo se describe en términos de un diagrama de clases UML, implementado mediante el *plug-in* UML2 de Eclipse (Figura 6-8).

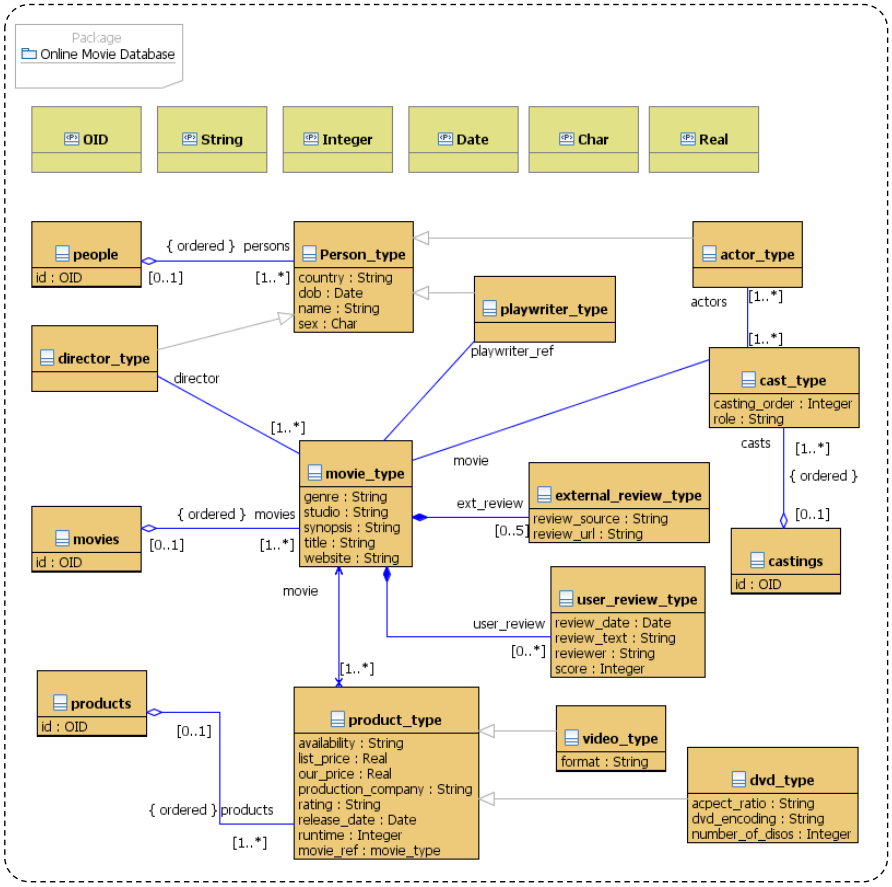

Figura 6-8. Modelo Conceptual de Datos (Caso de Estudio OMDB)

A partir de este modelo origen, como muestra la Figura 6-9, al ejecutar la transformación *UML2XMLSchema* generada, se deben obtener dos modelos: un modelo de salida que describa la base de datos OMDB en términos de un modelo

a nivel PSM, conforme al metamodelo XMLSchema (fichero *OMDB.schemaxml*) y un modelo que contenga las trazas entre los elementos definidos a nivel PIM y los elementos a nivel PSM.

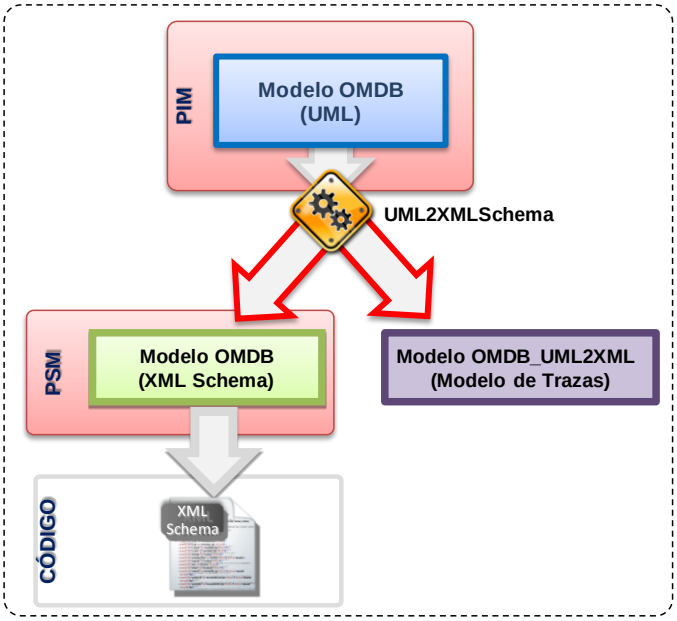

Figura 6-9. Proceso de Desarrollo del Caso de Estudio

Con el objetivo de comprobar que la transformación desarrollada con MeTAGeM-Trace genera los modelos destinos correctos, se evalúan dos factores:

1. Usando como entrada el mismo modelo origen *OMDB.uml*, el modelo destino *OMDB.schemaxml* que se obtiene a partir de la ejecución de la transformación *UML2XMLSchema* desarrollada mediante MeTAGeM-Trace debe ser igual o equivalente al modelo destino que se obtiene ejecutando la transformación implementada de forma manual. Para ello se emplea la observación directa de los modelos y el comparador de modelos EMFCompare [191].
2. Además del modelo destino *OMDB.schemaxml*, se debe generar un modelo de trazas entre los elementos del modelo OMBD a nivel PIM y los elementos del modelo OMBD a nivel PSM.

Para evaluar el primer factor, la Figura 6-10 presenta el modelo obtenido a partir de la ejecución de la transformación desarrollada mediante MeTAGeM-Trace (Figura 6-10 (a)) y el modelo obtenido ejecutando la transformación implementada de forma manual, contenida en la implementación de la herramienta M2DAT-DB (Figura 6-10 (b)). Como se puede comprobar en

dicha figura, ambos modelos son iguales. Debido a pequeñas diferencias en la implementación, tan solo cambia el nombre de algunos elementos. Por ejemplo en el caso del modelo (b), los elementos de tipo *ElementGlobal* incluyen la cadena "<<ElementGlobal>>" al final de su nombre.

Además usando el comparador de modelos EMFCompare, disponible en el entorno Eclipse, se verifica que no existen más diferencias entre dichos modelos. Por tanto, se asume que la transformación desarrollada con MeTAGeM-Trace ha proporcionado el mismo modelo de salida.

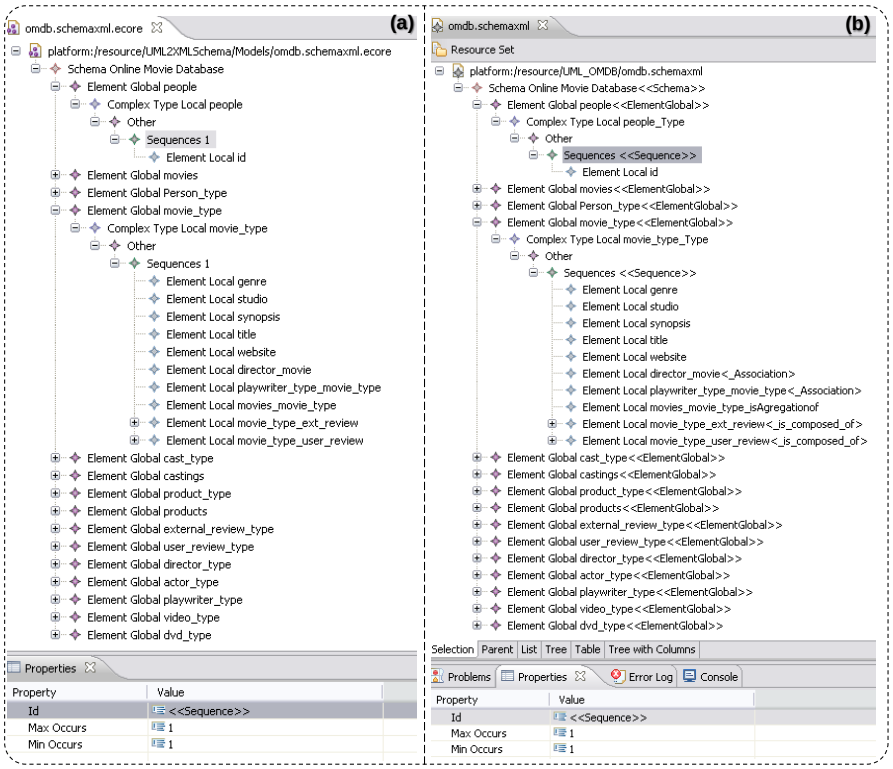

Figura 6-10. Modelo OMDB.schemaxml: (a) obtenido a partir de la transformación generada con MeTAGeM-Trace; (b) obtenido a partir de la transformación manual

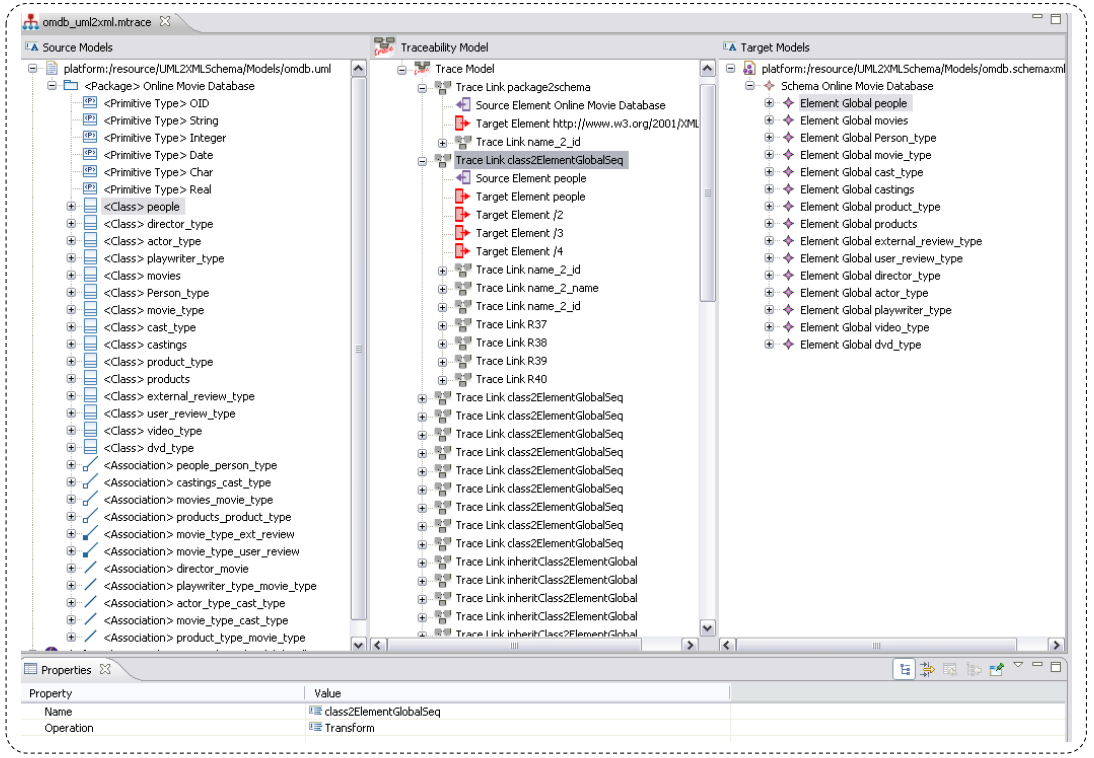

Figura 6-11. Modelo de Trazas generado (Caso de Estudio OMDB)

La Figura 6-11 valida que se ha cumplido el segundo factor, es decir, se ha obtenido un modelo que contiene las trazas que se derivan de la transformación entre los elementos del modelo OMDB a nivel PIM y los elementos del modelo OMDB a nivel PSM. Adicionalmente, se ha comprobado dicho modelo de trazas por observación directa.

6.4 Conclusiones de la Validación

Finalmente, en esta sección se presentan las conclusiones obtenidas a partir del proceso de validación del entorno MeTAGeM-Trace. Para ello, se dará respuesta a cada una de las preguntas establecidas en el protocolo de validación (sección 6.1.2).

En cuanto a las cuestiones establecidas para el meta-caso de estudio (CMCE):

- **CMCE-1.** ¿Ha sido posible completar el desarrollo de la transformación hasta obtener una transformación en código ATL?

  Sí, aunque en las primeras ejecuciones del proceso de validación se han detectado errores de implementación en la herramienta MeTAGeM-Trace que han sido reparados. Así, como muestra la Figura 2-2, el proceso de validación también ha servido como fase de pruebas de la herramienta presentada.

  Una vez corregidos todos los errores detectados, como se ha mostrado en la sección 6.2.2, el proceso de desarrollo se ha completado y se ha obtenido la transformación en código ATL.

- **CMCE-2.** Durante el proceso de desarrollo dirigido por modelos, ¿el usuario ha tenido que modificar manualmente los modelos generados (PSM y PDM)?

  Sí, como se ha indicado en la sección 6.2.2.2, para satisfacer las necesidades de la transformación que se ha desarrollado, ha sido necesario refinar el modelo PSM (modelo de aproximación híbrida). En dicho modelo se han incluido operaciones que se ejecutan en el ámbito de la transformación y que no pueden ser recogidas en el modelo PIM porque en dicho modelo solo se definen relaciones. Además, se ha modificado el valor de varios elementos de tipo *RightPattern* para invocar a las operaciones definidas y para establecer valores concretos de destino.

- **CMCE-3.** ¿El código de la transformación obtenido es totalmente ejecutable? Si no es así, ¿qué elementos no se han generado semi-automáticamente?

  No, aunque la mayor parte del mismo sí es ejecutable.

  Por defecto, el cuerpo de las operaciones (atributo *body*) definidas a nivel PSM es implementado en forma de comentario. En el caso del desarrollo de este meta-caso de estudio, estos atributos contienen código ATL, sin embargo, en muchos casos puede que el cuerpo de las operaciones se describa en lenguaje natural (se recuerda que el modelo de aproximación híbrida puede transformarse en modelos conformes a distintos lenguajes de transformación).

  En las reglas *composite2ElementLocalSeq* y *composite2ElementLocalAll* se generan dos errores de compilación. Estos errores se deben a que ambas reglas generan elementos destino del mismo tipo. Estos elementos son representados por variables que comparten nombre porque obtienen su nombre a partir del tipo del elemento.

- **CMCE-4.** En cuanto al número de líneas de código obtenidas, ¿es menor, igual o mayor al de la transformación UML2XMLSchema contenida en M2DAT-DB? Es necesario recordar que la transformación incluida en la herramienta M2DAT-DB no proporciona generación de trazas.

  Mayor. Una vez resueltos los errores que no permitían la ejecución de la transformación, esta se compone de un solo fichero que contiene 1731 líneas de código ATL. En cambio, la transformación implementada de forma manual, contenida en la herramienta M2DAT-DB, se compone de dos ficheros de código que suman un total de 460 líneas de código ATL. Es decir, la transformación generada con MeTAGeM-Trace ocupa casi 4 veces más líneas de código.

En cuanto a las cuestiones establecidas para el caso de estudio (CCE):

- **CCE-1.** ¿La transformación generada mediante el meta-caso de estudio se ejecuta correctamente sobre el modelo *OMDB.uml*?

  Sí, una vez corregidos los errores que no permitían ejecutar la transformación *UML2XMLSchema*, esta se ha ejecutado sobre el modelo de entrada *OMDB.uml* y el proceso ha concluido de forma satisfactoria.

- **CCE-2.** ¿El modelo *OMDB.xmlschema* que se ha obtenido es igual al modelo obtenido en el proceso de validación de la herramienta M2DAT-DB? Si no son iguales, ¿qué diferencias existen?

Son iguales. Como se ha indicado en la sección 6.3 se han realizado dos tipos de pruebas para comparar ambos modelos: observación directa y el uso de la herramienta EMFCompare. Como resultado de ambas pruebas, se ha comprobado que ambos modelos son iguales y solo difieren en el nombre de algunos elementos porque en el desarrollo del meta-caso de estudio no se ha tenido en cuenta cómo se generan dichos nombres en la transformación UML2Schema embebida en M2DAT-DB.

- **CCE-3.** En cuanto al tiempo de ejecución de la transformación obtenida como resultado del meta-caso de estudio para generar el modelo *OMDB.xmlschema*, ¿es menor, igual o mayor que ejecutando la transformación contenida en M2DAT-DB?

  Similar (algo mayor). Para evaluar esta pregunta se han realizado diez ejecuciones de cada transformación y se ha medido los tiempos mediante el medidor de tiempos que proporciona el *plug-in* de ATL. En la Tabla 6-2 se muestran los tiempos obtenidos en cada ejecución y la media total.

Tabla 6-2. Comparativa de tiempos de ejecución
(Transformación MeTAGeM-Trace vs. Transformación manual)

	Transformación generada con MeTAGeM-Trace	**Transformación implementada manualmente**
Nº Ejec.	**Tiempo (s.)**	**Tiempo (s.)**
1	0,198	0,094
2	0,079	0,094
3	0,093	0,109
4	0,078	0,093
5	0,094	0,125
6	0,093	0,094
7	0,078	0,093
8	0,093	0,094
9	0,074	0,094
10	0,094	0,079
Media	**0,0974**	**0,0969**

Como se puede observar en la tabla anterior, el tiempo de ejecución de la transformación desarrollada con MeTAGeM-Trace es ligeramente superior, del orden de 0,001 segundos. Esta diferencia puede considerarse despreciable.

Es necesario destacar que MeTAGeM-Trace genera transformaciones que deben ser ejecutadas con la máquina virtual EMFVM de ATL, mientras que la transformación contenida en M2DAT-DB, implementada a mano, está desarrollada para ejecutarse sobre la máquina virtual Regular de ATL. EMFVM es una máquina virtual optimizada por ello, aun generando trazas la transformación desarrollada con MeTAGeM-Trace obtiene tiempos de ejecución similares.

- **CCE-4.** ¿Se ha generado un modelo que contiene las trazas entre los modelos que describen OMDB a nivel PIM (*OMDB.uml*) y nivel PSM (*OMDB.xmlschema*)?

  Sí, como muestra la Figura 6-11, se ha obtenido un modelo de trazas conforme al metamodelo de trazabilidad propuesto por MeTAGeM-Trace. Dicho modelo contiene las trazas derivadas de la transformación entre los elementos del modelo *OMDB.uml* y los elementos del modelo *OMDB.schemaxml*.

A partir de los resultados obtenidos del desarrollo del meta-caso y del caso de estudio y de las respuestas a las preguntas anteriores se puede concluir que el entorno MeTAGeM-Trace ha podido ser aplicado por completo y de forma correcta al desarrollo de un caso real.

Mediante la herramienta MeTAGeM-Trace que da soporte a la propuesta metodológica presentada se ha realizado el modelado a alto nivel de las relaciones de transformación entre los elementos de los metamodelos UML y XMLSchema. A partir de este modelo de la transformación a alto nivel y la ejecución de varios módulos de transformación implementados en la herramienta, se ha obtenido el código ATL que implementa las reglas de transformación y que además incluye un generador de trazas. En el proceso de validación se ha mostrado que el código obtenido no es totalmente ejecutable, aunque sí la mayor parte del mismo. Sin embargo, se ha comprobado que, una vez resueltos los problemas que impiden la ejecución de la transformación, esta puede ejecutarse y genera los modelos destino correctos.

En la pregunta CMCE-4 se ha detectado que el tamaño de la transformación ATL generada con MeTAGeM-Trace es varias veces superior a la implementación manual. Esta diferencia de tamaño se debe a que la

transformación que genera MeTAGeM-Trace contiene además el generador del modelo de trazas. Sin embargo, dicha implementación se genera para ser ejecutada por la máquina virtual optimizada de ATL (*EMF-Specific VM*). Por esta razón, a pesar de ser una transformación mayor en cuanto a líneas de código, como se ha mostrado en la respuesta a la cuestión CCE-3, el tiempo de ejecución es similar al de la transformación implementada manualmente.

Por todo lo anterior, usando MeTaGeM-Trace el usuario solo tiene que modelar las relaciones de transformación a alto nivel, refinar los modelos intermedios y corregir un número pequeño de líneas del código final. Una vez realizado este trabajo de desarrollo, el usuario habrá obtenido una implementación de la transformación que generará los modelos destino adecuados y además, un modelo que contiene las trazas entre los elementos de los modelos implicados en la transformación.

Conclusiones

Para concluir esta memoria de tesis doctoral, en este capítulo se resumen las principales contribuciones realizadas y comprueba el cumplimiento de los objetivos establecidos en el capítulo 1. Además, se realiza un análisis de los resultados obtenidos proporcionando una enumeración de las publicaciones que sirven para contrastarlos, tanto en foros nacionales como en foros internacionales. Por último, se presentan las líneas de investigación futuras que permitirán seguir trabajando en la mejora de la propuesta presentada.

7.1 Análisis de la Consecución de Objetivos

En el primer capítulo de esta tesis doctoral, concretamente en la sección 1.2, se establecieron un conjunto de objetivos parciales para completar el objetivo principal de la tesis: la definición y construcción de un entorno de desarrollo de transformaciones de modelos que incorpore generación de información de trazas y permita generar trazas entre los elementos de los modelos origen y destino de una transformación.

A continuación se presentan los objetivos parciales establecidos y su grado de cumplimiento:

O1. Estudio de trabajos previos.

Para identificar el cuerpo del conocimiento en el que se basa la investigación de esta tesis doctoral, se han realizado dos estudios diferentes.

Por un lado, se ha realizado el análisis y evaluación de propuestas para el desarrollo dirigido por modelos de transformaciones de modelos (sección 3.2) y por otro lado, se ha realizado el análisis y evaluación de propuestas centradas en la gestión y generación de información de trazabilidad a partir del desarrollo de transformaciones de modelos (sección 3.3).

En la sección 3.2 se ha presentado el análisis y evaluación de propuestas para el desarrollo dirigido por modelos de transformaciones de modelos. Para realizar esta tarea se ha dado respuesta a un conjunto de preguntas de investigación definidas. Además, se ha establecido un conjunto de criterios de evaluación para comparar las propuestas analizadas. Este estudio surge como una evolución del estudio de trabajos previos, presentado en [30] como resultado de un proceso de revisión sistemática de la literatura [27, 105].

Como resultado de este estudio se destaca que la mayoría de las propuestas coinciden en los tipos de niveles de abstracción a los que se debe definir las transformaciones de modelos: niveles independientes del lenguaje o motor de transformación y niveles dependientes. También es necesario destacar que ninguna de las propuestas analizadas, centradas en aplicar MDE al desarrollo de transformaciones, considera la gestión y generación de información de trazabilidad.

En la sección 3.3 se presenta el análisis y estudio de diferentes propuestas centradas en la generación de la trazabilidad a partir del desarrollo de transformaciones de modelos. Este estudio es el resultado de un proceso completo de revisión sistemática de la literatura, basado en las guías propuestas por Kitchenham y Charters [105] y Biolchini [27]. Al igual que en el estudio anterior, para conocer el estado del arte actual en este tema, se ha dado respuesta a un conjunto de cuestiones de investigación y con el objetivo de evaluar y analizar las propuestas encontradas en la literatura, se ha definido un conjunto de criterios de evaluación.

Entre las conclusiones obtenidas en este estudio se destaca que, a pesar de enmarcarse en el contexto de MDE, el nivel de automatización de los procesos propuestos es bastante limitado. También llama la atención que la mayoría de las propuestas se desarrollen de acuerdo a las características de una aproximación o lenguaje de transformación concreto.

Por otro lado, ambos estudios coinciden en poner de manifiesto la brecha o *gap* existente entre la teoría y la práctica: mientras la mayoría de las propuestas metodológicas son muy completas, las herramientas para dar soporte a dichas ideas teóricas son incompletas e incluso en algunos casos, ni siquiera existe tal soporte tecnológico.

A partir de estos estudios previos, en esta tesis doctoral se propone la incorporación de mecanismos generadores de información de trazabilidad en el desarrollo dirigido por modelos de transformaciones de modelos. Así, como se detalla en la sección 7.2, las principales contribuciones de este trabajo son: la identificación automática de relaciones de trazabilidad a partir de la definición de relaciones de transformación, la inclusión de construcciones, que incorporadas a los metamodelos de transformación, permiten la generación de trazas y el desarrollo de una herramienta de soporte completa para el desarrollo de modelos dirigido por modelos de transformaciones que incorporen, de forma embebida, generadores de trazas.

O2. Especificación de un metamodelo de trazabilidad genérico.

Uno de los principales objetivos de la propuesta MeTAGeM-Trace, que se presenta en esta tesis doctoral, es la generación (semi-)automática de modelos de trazas. En consecuencia, es necesario definir un metamodelo de trazabilidad que describa la sintaxis abstracta de dichos modelos.

La mayoría de las propuestas analizadas para la gestión de la trazabilidad, en la sección 3.3, coinciden en la necesidad del uso de metamodelos genéricos que permitan modelar las trazas en cualquier dominio de aplicación. Para ello, proponen metamodelos de trazabilidad [34, 97, 112, 128, 138, 175, 217], que permiten el modelado de elementos de los modelos y las relaciones entre ellos.

A partir de las conclusiones extraídas del análisis de dichos metamodelos, en la sección 4.2.1 se presenta la especificación del metamodelo de trazabilidad de MeTAGeM-Trace. Dado que MeTAGeM-Trace permite la generación de modelos de trazas a partir de la definición a alto nivel de transformaciones entre diferentes metamodelos origen y destino, el metamodelo de trazabilidad presentado debe ser genérico, para poder ser aplicado en cualquier escenario de trazabilidad.

Al igual que los metamodelos de trazabilidad encontrados en la literatura, este metamodelo permite la definición de elementos de los modelos y las trazas existentes entre ellos. Pero, además, con el objetivo de dar soporte a otras tareas como la visualización, permite la representación de los modelos origen y destino participantes en el modelo de trazas.

O3. Estudio y mejora (si procede) de la propuesta MeTAGeM para el desarrollo de transformaciones de modelos [30].

En [30] se presentó MeTAGeM, un entorno para el desarrollo de transformaciones de modelos dirigido por modelos. Dicho entorno propone un conjunto de buenas prácticas para la aplicación de los principios de MDE al desarrollo de transformaciones. Así, dado que en esta tesis doctoral el objetivo principal es proporcionar un entorno de DDM de transformaciones de modelos que incluyan generadores de trazas, se han aprovechado las contribuciones proporcionadas, en este sentido, por MeTAGeM.

Para ello, ha sido necesario realizar un estudio completo de la propuesta metodológica presentada en [30]. En dicho propuesta, destaca la clasificación de los niveles de abstracción que se deberían usar para la especificación de las transformaciones de modelos (PIM, PSM, PDM y Código), que se corresponde con los niveles definidos por MDA [153].

En este sentido, se ha aceptado como válido el empleo de dichos niveles de abstracción y se han analizado cada uno de los metamodelos propuestos por MeTAGeM con el objetivo de encontrar debilidades y puntos de mejora.

Tras el análisis de dichos metamodelos, como se presenta en la sección 4.2, a nivel PIM se han realizado cambios menores relacionados con la nomenclatura de las metaclases y sus atributos; se han redefinido los tipos de datos enumerados; y se han añadido dos nuevas metaclases abstractas con el objetivo de incluir mecanismos de herencia que mejoren la calidad de dicho metamodelo.

A nivel PSM se define un metamodelo que describe las transformaciones de modelos que siguen la aproximación híbrida (imperativa + declarativa). En este metamodelo se han realizado los mismos cambios que en el anterior: nomenclatura de las metaclases, redefinición de los tipos enumerados e inclusión de metaclases abstractas. Además, se han incorporado nuevas metaclases que han resuelto un conjunto de errores relacionados con las operaciones incluidas en el módulo de la transformación. Por otro lado, la metaclase que define la pre-condición de las reglas de transformación (*guard*) anteriormente estaba contenida en los elementos origen de la regla y en este nuevo metamodelo PSM, se encuentra contenida en la metaclase que describe a las regla de transformación (*Rule*).

Dado que a nivel PSM se ha definido un metamodelo que describe las transformaciones que siguen una aproximación híbrida, a nivel PDM se han de definir metamodelos que describan lenguajes de transformación que sigan esta aproximación. Así, en [30] se propuso el empleo de los lenguajes de transformación ATL y RubyTL. A este nivel, MeTAGeM-Trace asume la conveniencia de centrarse en el lenguaje ATL, principalmente por dos motivos: en el momento de desarrollar esta tesis doctoral, es considerado por la comunidad de investigadores como el estándar *de-facto* para el desarrollo de transformaciones de modelos y los desarrolladores del lenguaje proporcionan una especificación e implementación completa del metamodelo que describe el lenguaje.

O4. Especificación de un metamodelo de transformación a nivel PDM de acuerdo a las características del lenguaje híbrido ETL.

En los últimos tiempos, el *framework* Epsilon [114, 116] está adquiriendo una notable importancia entre la comunidad de investigadores y desarrolladores. Epsilon se compone de una familia de lenguajes para llevar a cabo la implementación de tareas propias de la Ingeniería Dirigida por Modelos. Concretamente, ETL (*Epsilon Transformation Language*, [113]) es un lenguaje

híbrido que proporciona Epsilon para el desarrollo de reglas de transformación entre modelos.

A diferencia de ATL, los desarrolladores de ETL no proporcionan una especificación completa del metamodelo del lenguaje, tan solo presentan una aproximación de su sintaxis abstracta. Con el objetivo de considerar en MeTAGeM-Trace el desarrollo de transformaciones en el lenguaje ETL, se realiza la especificación y posteriormente, la implementación del metamodelo de dicho lenguaje. Para realizar esta tarea, se ha analizado en profundidad la sintaxis abstracta de ETL y los ejemplos disponibles en el sitio web de la herramienta Epsilon. Como resultado, en la sección 4.2.4.2, se ha presentado una especificación completa del metamodelo del lenguaje de transformación híbrido ETL, que forma parte del nivel PDM de MeTAGeM-Trace.

O5. Identificación de construcciones que, incorporadas a los metamodelos de transformación definidos a nivel PIM, PSM y PDM, permitirán generar modelos de traza conformes al metamodelo definido en O2.

Este quinto objetivo se centra en la identificación de aquellas construcciones que, embebidas en el entorno de DDM de transformaciones de modelos, permiten generar las trazas del sistema que se derivan de la ejecución de las transformaciones construidas.

Como ya se ha mencionado en el O2, en la sección 4.2.1, se ha especificado el metamodelo de trazabilidad que describe como serán los modelos de trazas generados por el entorno MeTAGeM-Trace. A partir de los elementos descritos en dicho metamodelo, se han identificado las construcciones que se deben incorporar en los distintos niveles de abstracción en los que se modelan las transformaciones para la creación de dichos elementos.

Así, a nivel PIM, donde se describen de forma genérica las relaciones entre los elementos de los modelos origen y destino, no es necesario incorporar nuevas construcciones, debido a dicha genericidad.

A partir de cada elemento de tipo *Relations* definido a nivel PIM, se genera una regla de transformación (*Rule*) a nivel PSM. Puesto que en este caso, las reglas de transformación contienen un significado propio, es necesario establecer a nivel PSM los elementos que describan las relaciones de trazabilidad que se derivan de las relaciones modeladas a nivel PIM. Por ello, en el metamodelo PSM se incluye una nueva metaclase (*TraceLink*). Sin embargo, dado que las relaciones de trazabilidad pueden darse como resultado de una regla de transformación (a partir del elemento *a*, se genera un elemento *b*) o como resultado de una

asignación contenida en una regla de transformación (el atributo *name* del elemento *b*, se genera a partir del atributo *nombre* del elemento *a*), la metaclase *TraceLink* se ha especializado (mecanismo de herencia) en dos subclases, *TraceRule* y *TraceBinding*, que representan las relaciones de trazabilidad derivadas de cada uno de estos tipos de relaciones. Estas construcciones han sido incorporadas a las especificaciones de los metamodelos, presentadas en la sección 4.2.

A nivel PDM se describen las transformaciones de acuerdo a las características propias de los lenguajes de transformación, por ello no es posible definir construcciones específicas para la generación de modelos de trazas. Sin embargo, sí es necesario definir qué elementos de los proporcionados por el lenguaje se usarán para la creación de los modelos de trazas.

En este caso, siguiendo la propuesta presentada por Jouault en [97], se ha decidido que tanto las trazas de salida como los elementos del modelo de trazas que representan los elementos origen y destino se construyan como elementos destino de las reglas de la transformación final generada. De igual manera, para la creación del elemento raíz del modelo de trazas y los elementos que representan a los modelos de origen y destino participantes en el modelo de trazas, se ha decidido aprovechar las ventajas que ofrecen los lenguajes híbridos en cuanto a la ejecución imperativa de reglas de transformación. Concretamente los lenguajes de transformación seleccionados a nivel PDM (ATL y ETL) permiten la definición de acciones que se ejecutarán al principio de la transformación. De esta forma, para crear el modelo de trazas correctamente, en primer lugar se genera la raíz del modelo y los elementos que representan los modelos origen y destino implicados en las trazas y posteriormente, se generan las trazas y los elementos.

O6. Especificación de transformaciones de modelos.

Para automatizar el DDM de transformaciones de modelos, se ha definido un conjunto de transformaciones que permite el paso (semi-)automático entre los modelos de los distintos niveles de abstracción presentados en MeTAGeM-Trace. Concretamente, se propone el uso de tres transformaciones de modelos: PIM-PSM, PSM-ATL (PDM) y PSM-ETL (PDM); y dos transformaciones de modelo a texto o generadoras de código: PDM-Código ATL y PDM-Código ETL.

La especificación de las reglas de transformación que componen dichos módulos ha sido descrita en términos de lenguaje natural en la sección 4.3.

O7. Construcción de la herramienta de soporte.

Uno de los principales objetivos de esta tesis doctoral es la construcción de una herramienta que ofrezca soporte tecnológico a la propuesta metodológica MeTAGeM-Trace. La implementación de dicha herramienta se presenta en el capítulo 5.

Para la consecución de este objetivo, en la sección 1.2 se ha planteado un conjunto de sub-objetivos que se deben cumplir. Así el primero de ellos, que consiste en la especificación de la arquitectura de la herramienta de soporte, ha sido satisfecho en la sección 5.1, donde se presenta una arquitectura basada en capas (presentación, lógica y persistencia) que se compone de varios módulos ortogonales a dichas capas. Cada uno de los módulos representa los DSLs que componen MeTAGeM-Trace: DSL MeTAGeM (nivel PIM), DSL de aproximación híbrida (nivel PSM), DSL del lenguaje ATL (nivel PDM), DSL del lenguaje ETL (nivel PDM) y el DSL de Trazabilidad.

El siguiente paso, que satisface los sub-objetivos O7.2 y O7.3 es la implementación de los DSLs mencionados. Para ello se ha seguido el método de construcción detallado en la sección 2.3. En cuanto a los medios tecnológicos usados para la implementación de estos DSLs, como se presenta en la sección 5.2, se ha empleado como plataforma el entorno de desarrollo Eclipse, concretamente, el *framework* de modelado EMF. Con el objetivo de mejorar la experiencia del usuario a la hora de interactuar con estos DSLs se han implementado editores *ad-hoc* de acuerdo a la naturaleza relacional de estos modelos (sección 5.3.2). Además, para facilitar la creación de nuevos modelos conformes a los metamodelos de dichos DSLs se ha construido un conjunto de asistentes (sección 5.3.3).

Como se ha mencionado en el O6, la propuesta metodológica define un conjunto de transformaciones de modelo a modelo y un conjunto de transformaciones de modelo a texto. Dichas transformaciones han sido implementadas mediante el lenguaje de transformación ATL en el caso de las transformaciones M2M y en el caso de las M2T, mediante MOFScript. Estas implementaciones que han satisfecho el sub-objetivo O7.4 han sido presentadas en las secciones 5.3.4 y 5.3.5.

Finalmente, se ha llevado a cabo la implementación de un módulo denominado *procesador de modelos* [215] que ha sido presentado en la sección 5.3.7. En dicha sección se presenta cómo se ha construido un conjunto de componentes (lanzadores, entradas en menús contextuales, etc.) para proporcionar

una interfaz de usuario integrada que permita la ejecución sencilla de las tareas proporcionadas por MeTAGeM-Trace: ejecución de transformaciones, generación de código y validación de modelos. De esta forma se ha satisfecho el sub-objetivo O7.5

O8. Validación del entorno de desarrollo propuesto.

Para cumplir este objetivo, en el capítulo 6 se presenta cómo se ha aplicado el método de validación que se ha descrito en la sección 2.4. Así, la validación del entorno se ha realizado mediante el desarrollo de un meta-caso de estudio que consiste en seguir la propuesta y hacer uso de la herramienta presentada para llevar a cabo la implementación de una transformación de modelos que permite pasar de un modelo conceptual de datos, en términos de un diagrama de clases UML, a un modelo lógico que define dichos datos en términos de un esquema XML. Además, para comprobar que la transformación generada es correcta, se ha desarrollado un caso de estudio centrado en el desarrollo de una base de datos de películas online (OMDB).

El proceso de validación se ha desarrollado satisfactoriamente, por tanto, se concluye que el entorno MeTAGeM-Trace puede aplicarse al desarrollo de transformaciones de modelos. El meta-caso de estudio ha proporcionado una transformación en código ATL que ejecutada sobre un modelo UML, genera un modelo destino conforme al metamodelo *XMLSchema* y un modelo que contiene las trazas entre los elementos del modelo origen y los elementos del modelo destino.

Por todo lo anterior, se puede concluir que se han cumplido satisfactoriamente todos los objetivos planteados y se ha probado la hipótesis propuesta.

7.2 Principales Contribuciones

Entre los resultados obtenidos en esta tesis doctoral se encuentra un conjunto de contribuciones que se detallan a continuación. La mayor parte de ellas se alinean directamente con los objetivos satisfechos.

- **Una evolución y actualización de un estudio y análisis de propuestas centradas en el desarrollo dirigido por modelos de transformaciones de modelos**.

Una de las contribuciones de la tesis doctoral presentada en [30] es un análisis de las propuestas existentes sobre la utilización de los principios de MDE en el ámbito de las transformaciones de modelos. A partir de las propuestas identificadas en dicho análisis y otras más recientes, en esta tesis doctoral se ha realizado un estudio para conocer cómo se aplican los principios de MDE al desarrollo de transformaciones de modelos y si estas propuestas consideran la gestión de la trazabilidad a partir del desarrollo de las transformaciones (sección 3.2).

Para dar respuesta al objetivo principal del estudio, se ha planteado un conjunto de preguntas de investigación a las que se ha dado respuesta en la sección 3.2.5. En dicha sección además, se presenta una comparativa entre las diferentes propuestas analizadas. Para realizar esta comparativa, se ha establecido un conjunto de criterios de evaluación y los posibles valores que se le pueden asignar a cada propuesta en cada uno de los criterios.

Como resultado de este estudio se ha determinado la conveniencia de: aplicar los principios de MDE al desarrollo de transformaciones de modelos, definir niveles de abstracción para el modelado de las transformaciones y potenciar la automatización en el proceso de construcción de las transformaciones. Así mismo, se ha puesto de manifiesto que la gestión de la información de trazabilidad no es afrontada por las propuestas centradas en el DDM de transformaciones de modelos.

Estos resultados han permitido establecer parte de los objetivos a cumplir por la propuesta MeTAGeM-Trace, que se presenta en esta tesis doctoral.

- **Un estudio y análisis completo de propuestas para la generación de trazabilidad a partir del desarrollo de transformaciones de modelos.**

Dado que en el estudio anterior se identificó que ninguna de las propuestas analizadas para el DDM de transformaciones de modelos consideran la generación de la trazabilidad, se ha realizado un estudio y análisis completo de las propuestas existentes para la generación de trazabilidad a partir del desarrollo de transformaciones de modelos, sin tener en cuenta si aplican los principios de MDE a la construcción de dichas transformaciones. Dicho estudio y análisis, ha sido presentado en la sección 3.3.

La identificación de las propuestas se ha llevado a cabo mediante un proceso de revisión sistemática de la literatura. Al igual que en el caso anterior, se ha definido un conjunto de cuestiones de investigación a las que se ha dado

respuesta así como un conjunto de criterios de evaluación que permiten comparar las propuestas analizadas.

Como se ha comentado en varias ocasiones a lo largo de esta tesis doctoral, este estudio ha puesto de manifiesto varias debilidades o cuestiones inmaduras que se dan en el estado del arte de esta cuestión. Entre estas debilidades se ha identificado: ausencia de la aplicación de los principios de MDE al desarrollo de las transformaciones, dependencia de lenguajes de transformación o aproximaciones concretas, falta de mecanismos específicos de visualización y análisis de las trazas generadas y ausencia de herramientas completas que den soporte a las propuestas metodológicas presentadas.

Por otro lado, también se ha obtenido información relevante acerca de cómo almacenar las trazas generadas así como los componentes más comunes de los metamodelos de trazabilidad.

Estos resultados, junto con los del estudio anterior, han permitido llevar a cabo la especificación y construcción del entorno MeTAGeM-Trace.

- **Una propuesta metodológica para el DDM de transformaciones de modelos que incluyan generadores de trazas.**

Una de las principales contribuciones de esta tesis doctoral es la especificación de la metodología MeTAGeM-Trace para el desarrollo de transformaciones de modelos que incluyen generadores de modelos de trazas siguiendo un proceso dirigido por modelos (capítulo 4). Dicha metodología surge como una evolución de la metodología MeTAGeM [30].

La propuesta metodológica MeTAGeM-Trace propone la incorporación de un meta-generador de trazabilidad en el proceso de DDM de transformaciones de modelos. Para ello, se especifica un metamodelo de trazabilidad y se identifica un conjunto de construcciones que se deben tener en cuenta a la hora de definir los metamodelos que describen la transformación a diferentes niveles de abstracción. Además, también se especifican un conjunto de reglas de transformación de modelos que permitirán automatizar el desarrollo de las transformaciones de modelos incluyendo mecanismos de generación de trazas. Con el objetivo de mejorar la calidad de las transformaciones generadas y evitar la propagación de errores de modelado entre los modelos de transformación de diferentes niveles de abstracción, la propuesta contiene un conjunto de restricciones o reglas de validación que deben cumplir los modelos para considerarse válidos.

Finalmente, para serializar los modelos en el código que implementa la transformación que contiene el generador de trazas, se especifican un conjunto de reglas de transformación de modelo a texto.

- **Desarrollo de un entorno de DDM de transformaciones de modelos que incluye generadores de modelos de trazas.**

Otra de las principales contribuciones de esta tesis es la construcción de la herramienta MeTAGeM-Trace que ofrece soporte tecnológico para la propuesta metodológica del mismo nombre (capítulo 5).

Para ello, se ha definido una arquitectura conceptual basada en capas compuesta por un conjunto de módulos ortogonales a dichas capas. Cada uno de estos módulos se corresponde con la implementación completa de uno de los DSLs que forman la herramienta. Con el objetivo de permitir la interoperabilidad entre estos módulos, se ha implementado un procesador de modelos que permite ejecutar transformaciones entre modelos, validación de modelos y generación de código.

Por otro lado, se ha definido un diseño técnico que establece la relación entre los componentes de la arquitectura conceptual y los medios tecnológicos empleados para su construcción.

La principal característica de la herramienta MeTAGeM-Trace es su naturaleza extensible e interoperable. La extensibilidad facilita la incorporación de nuevas funcionalidades, como por ejemplo, el soporte de nuevas aproximaciones a nivel PSM o de nuevos lenguaje de transformación a nivel PDM y código.

Estas nuevas funcionalidades se soportarán mediante la creación de nuevos DSLs en MeTAGeM-Trace, que serán directamente compatibles con los módulos ya existentes sin requerir ningún esfuerzo adicional, es decir, sin la necesidad de construir puentes entre diferentes espacios tecnológicos, ya que se utiliza la misma tecnología y el mismo proceso para desarrollarlos.

- **Implementación de editores/visualizadores específicos para la representación de modelos de transformaciones y modelos de trazas.**

Finalmente, otra de las contribuciones a destacar de esta tesis doctoral es la implementación de editores/visualizadores construidos *ad-hoc* a partir de los editores genéricos proporcionados por EMF. Estos editores específicos permiten la edición y visualización de los modelos de transformación y de los modelos de trazas, aprovechando la naturaleza relacional de estos modelos.

En concreto, permiten la visualización, al mismo tiempo, de los modelos origen, los modelos destino y el modelo de transformación o modelo de trazas que contiene las relaciones entre los elementos de los modelos anteriores.

7.3 Resultados Científicos

Algunos de los resultados relacionados con el desarrollo de esta tesis se han publicado en diferentes congresos, tanto nacionales como internacionales. A continuación se presentan las diferentes publicaciones, agrupadas por tipo de publicación. Dada la naturaleza técnica de esta tesis doctoral, para la publicación en foros de impacto, ha sido necesario esperar a finalizar la implementación de la herramienta. Por ello, en el momento de redactar esta memoria de tesis se cuenta con publicaciones de menor impacto que recogen resultados parciales de la misma. En la actualidad se está trabajando en la publicación en foros de mayor impacto.

- **Artículos en Conferencias Internacionales**
  - Jiménez, Á., Granada, D., Bollati, V.A., Vara, J.M. (2011). *Using ATL to support Model-Driven Development of RubyTL Model Transformations*. Proceedings of the 3rd International Workshop on Model Transformation with ATL (MtATL 2011), Zurich, Suiza (01 de Julio, 2011). Publicado en CEUR-WS Workshop Proceedings, vol. 742, pp. 35-48.
- **Artículos en Conferencias Nacionales**
  - Jiménez, Á., Vara, J.M., Bollati, V.A., Marcos, E. (2011). *Gestión de la Trazabilidad en el Desarrollo Dirigido por Modelos de Transformaciones de Modelos: una revisión de la literatura.* **XVI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2011)**. A Coruña, España. (5 al 7 de Septiembre, 2011).
  - Jiménez, Á., Vara, J. M., Bollati, V. A., Marcos, E. (2010). *Mejorando el nivel de automatización en el desarrollo dirigido por modelos de editores gráficos***.** VII Taller sobre Desarrollo de Software Dirigido por Modelos. DSDM'10. Valencia, España (7 de Septiembre, 2010).
- **Artículos en Revistas Internacionales**
  - Santiago, I., Jiménez, Á., Vara, J.M., De Castro, V., Bollati, V. A., Marcos, E. (Pendiente de Revisión). *Model-Driven Engineering as a new landscape for Traceability Management: a Systematic Review.* **Information and**

Software Technology (JCR-Q1, Factor de impacto: 1,507). Enviado 05 de enero, 2012.

- **Capítulos de Libros Nacionales**
  - Jiménez, Á., Granada, D., García, A. (Pendiente de publicación). *EuGENia*. Capítulo del libro: "Desarrollo de Software Dirigido por Modelos: Conceptos, Métodos y Herramientas". Red Española de DSDM. Ed. Ra-Ma.
- **Registro de la Propiedad Intelectual**
  - Título: *SOD-M Tool: Una herramienta para el Modelado de Sistemas de Información siguiendo la Metodología SOD-M.*
    - Inventores: E. Marcos, V. De Castro, Á. Jiménez
    - Nº de Aplicación: M-007589/2011
    - País: España
    - Fecha: 30/09/2011
    - Entidad Titular: Universidad Rey Juan Carlos
    - Países Participantes: España

7.4 Trabajos Futuros

A partir de las contribuciones realizadas en esta tesis, se han detectado varias líneas de investigación en las que seguir trabajando. Algunas de ellas simplemente no se habían considerado como objetivos de esta tesis, mientras que otras han surgido durante el desarrollo de la misma. A continuación, se resumen las principales líneas futuras de investigación.

7.4.1 Trabajos Futuros en la Generación y Gestión de Trazas

Como se ha visto a lo largo de esta tesis doctoral, la trazabilidad es un tema importante en el ámbito de la Ingeniería de Software y con el surgimiento de MDE, y más específicamente del DDM, la generación y gestión de las trazas del sistema ha cobrado cada vez mayor importancia.

En esta tesis doctoral se presenta una metodología para el desarrollo (semi-)automático de modelos de trazas a partir del desarrollo de transformaciones de modelos. Sin embargo, dado que este campo de investigación aún se encuentra

inmaduro, a continuación en las siguientes sub-secciones se presentan algunas líneas futuras de investigación en este campo. Estas líneas de investigación están siendo afrontadas en el marco de la tesis doctoral de Iván Santiago, miembro del Grupo Kybele.

7.4.1.1 Generación de Trazas a partir de Transformaciones M2T

Del mismo modo que las transformaciones de modelos definen implícitamente relaciones de trazabilidad que pueden aprovecharse para la generación (semi-)automática de modelos de trazas, esta idea puede ser adaptada al desarrollo de transformaciones de modelo a texto.

En este sentido existe una propuesta que ha sido analizada en la sección 3.3.4.8 de esta tesis doctoral. Para afrontar esta línea de investigación, los autores proponen dos caminos alternativos: emplear la trazabilidad implícita del motor de transformaciones M2T de MOFScript y la definición explicita de las relaciones de trazabilidad en las reglas de la transformación de modelo a texto.

La principal debilidad de esta aproximación es que no aplica los principios de MDE al desarrollo de las transformaciones de modelo a texto. Sin embargo, *a priori* parece bastante complejo aplicar técnicas MDE al desarrollo de este tipo de transformaciones en tanto en cuanto no se disponga de un metamodelo que describa la sintaxis del lenguaje que implementa el código, de forma similar a como lo hacen las gramáticas.

Por todo ello, uno de los trabajos futuros que se plantea es el desarrollo de métodos o técnicas para la generación de modelos de trazas a partir de las relaciones implícitas que se derivan de la ejecución de una transformación de modelo a texto, aplicando si fuera posible los principios de MDE a la construcción de dichas transformaciones.

7.4.1.2 Análisis de las trazas generadas

Como se ha comentado en varias ocasiones a lo largo de esta tesis doctoral, en la literatura existen evidencias acerca de la importancia de analizar la información de trazabilidad para llevar a cabo otras actividades software como el análisis del impacto del cambio, pruebas de regresión, validación de requisitos, análisis de coberturas, toma de decisiones, gestión de la configuración y en general, todas las tareas de mantenimiento del software [5, 9, 37, 38, 42, 107, 142, 148, 160, 177, 182, 193, 218].

Sin embargo, como se ha mostrado en la discusión del estudio sobre las propuestas para la generación de trazabilidad a partir del desarrollo de transformaciones de modelos (sección 3.3.5) se han identificado pocas propuestas que ofrezcan facilidades para el uso de la información de trazabilidad una vez que ha sido generada.

A pesar de haberse detectado como un punto de mejora en el estado del arte de la gestión de la trazabilidad en el ámbito de MDE, esta cuestión no ha formado parte de los objetivos de esta tesis doctoral. Es por ello que se plantea como línea de investigación futura la definición de una propuesta para la generación de informes técnicos de trazabilidad y métodos para facilitar el análisis de las trazas del sistema. Esta línea de investigación ya está siendo afrontada en el contexto de la tesis doctoral de Iván Santiago. En este sentido se ha empezado a trabajar en el desarrollo de informes técnicos que proporcionen información acerca de los tipos de trazas que se generan y en qué fases del ciclo de desarrollo.

7.4.1.3 Control sobre los tipos de trazas generadas

De acuerdo con Aizenbud-Reshef *et al*. [5], los diferentes *stakeholders* implicados en el proceso de desarrollo tienen diferentes objetivos a la hora de acceder a la información de trazabilidad. Diseñadores, programadores, *testers*, jefes de proyecto, ingenieros de sistemas o clientes son algunos de los posibles *stakeholders* que pueden estar implicados en el proceso de desarrollo del software [224]. Así, por ejemplo el cliente o el jefe de proyecto necesitarán acceder a las trazas del sistema relacionadas con los requisitos del mismo para comprobar que son satisfechos por el sistema final. En cambio los encargados del mantenimiento del sistema requerirán consultar las trazas entre los artefactos a bajo nivel. Por todo ello, resulta útil proporcionar distinto tipo información de trazabilidad, dependiendo de quién la requiera.

En MeTAGeM-Trace, por defecto, se genera un modelo de trazas que contiene todas las trazas del sistema, derivadas de la ejecución de una transformación de modelos. El usuario tiene la posibilidad de determinar qué relaciones de trazabilidad quiere registrar pero para ello, debe modificar manualmente los modelos que definen la transformación a nivel PSM o PDM. Esto implica que se debe disponer de una copia modificada de los modelos PSM y PDM por cada *stakeholder* que requiera generar modelos de trazas. Esta duplicidad puede acarrear por un lado, problemas de coherencia entre las versiones y por otro, implica un mayor coste en cuanto al espacio de almacenamiento.

Así, en la línea de la propuesta anterior, se está trabajando para proporcionar distintos tipos de análisis de las trazas del sistema, dependiendo del usuario. Concretamente, en estos momentos se ha establecido una distinción a la hora de proporcionar los análisis de trazas, de acuerdo a dos tipos de perfiles: perfil analítico (clientes, jefes de proyecto, analistas, etc.) y perfil operativo (desarrolladores, encargados de mantenimiento, diseñadores, *testers*, etc.).

7.4.2 Mejoras en el Desarrollo de Transformaciones de Modelos

Otro de los aspectos clave del entorno MeTAGeM-Trace es el desarrollo de las transformaciones de modelos. Este campo de investigación, siendo más maduro que la gestión de la trazabilidad en MDE, aun dispone de un amplio margen de mejora y evolución respecto de los avances en Ingeniería del Software [78, 214]. Así, en las siguientes sub-secciones se presentan algunas líneas de investigación futuras relacionadas con el desarrollo de transformaciones de modelos.

7.4.2.1 Transformaciones bidireccionales

La bidireccionalidad en las transformaciones de modelos es un mecanismo que facilita el mantenimiento de la coherencia entre dos (o más) fuentes de información relacionadas [45]. En el contexto de MDE, la bidireccionalidad permite sincronizar las diferentes vistas de los modelos, y debería tenerse en cuenta en el desarrollo de cualquier herramienta de soporte a las actividades del MDE si se quiere contemplar cuestiones como la evolución de metamodelos y modelos, o el soporte a los mecanismos de ingeniería inversa.

A pesar de la utilidad e importancia de soportar el modelado de las transformaciones bidireccionales, se trata de uno de los aspectos menos afrontados por la comunidad de investigadores [45, 189]. Por esta razón, se considera importante permitir el modelado de las transformaciones en forma bidireccional. Para ello, el entorno MeTAGeM-Trace puede ser adaptado por medio de nuevas reglas de transformación para soportar la bidireccionalidad de las transformaciones. Estas nuevas reglas de transformación deberían ser inversas a las que ya forman parte del entorno MeTAGeM-Trace, es decir, se debe disponer de las transformaciones: de PSM a PIM y de PDM a PSM. Además, debería ser posible crear los modelos PDM a partir del código de la transformación (para el lenguaje ATL esta funcionalidad ya es soportada).

7.4.2.2 Parametrización de las transformaciones

Las transformaciones que forman parte de la implementación de MeTAGeM-Trace (PIM-PSM, PSM-PDM) son cerradas, en el sentido de que su comportamiento siempre es el mismo. Es otras palabras, para un mismo modelo origen siempre se va a obtener el mismo modelo destino.

Sin embargo, en muchas ocasiones resulta deseable introducir algunas decisiones de diseño que guíen el proceso de desarrollo, especialmente cuando se baja a niveles dependientes de plataforma y es necesario indicar cómo los conceptos abstractos descritos en los niveles superiores se convierten en objetos concretos de la plataforma. En otros casos, debido a los dominios de aplicación de los modelos implicados en la transformación, pueden aparecer ambigüedades en el proceso de transformación que no puedan ser resueltas de forma automática en tiempo de ejecución [10].

Por todo lo anterior, resulta recomendable poder introducir información adicional en el proceso de desarrollo de las transformaciones. Para soportar dicho comportamiento se propone parametrizar las transformaciones, es decir, proporcionar mecanismos para que el usuario encargado de ejecutar la transformación pueda introducir decisiones de diseño en la transformación, en forma de parámetros.

Esta aproximación ya ha sido afrontada en trabajos anteriores [199, 200]. En dichos trabajos, se propone emplear modelos de *weaving* [54] como modelos de anotación de las transformaciones. En estos modelos de anotación se podrán introducir parámetros o decisiones de diseño que modifiquen ciertos aspectos del comportamiento de la transformación. De esta forma, a partir de un modelo origen se podrían generar diferentes modelos destino, dependiendo de la información contenida en modelo de *weaving*.

En cuanto a la aplicación de esta idea al entorno MeTAGeM-Trace, dado que los modelos de *weaving* son implementados en la plataforma Eclipse mediante el *framework* AMW [54, 96], es posible parametrizar las transformaciones que forman parte de MeTAGeM-Trace haciendo uso de dicho *framework*. Sin embargo, como se ha comentado en varias ocasiones a lo largo de esta tesis, uno de los avances de este trabajo respecto de trabajos anteriores ha sido prescindir de la dependencia de la herramienta AMW, debido a que no es actualizada al mismo ritmo que otros *framework* de Eclipse, como EMF, lo que puede dar lugar a algunos problemas de compatibilidad.

Por ello, igual que en esta tesis, se ha simulado el comportamiento del editor de AMW mediante el uso de EMF y modificando el código Java que implementa los editores que ofrece EMF, una posible línea de investigación futura es el desarrollo de un DSL para la parametrización de transformaciones de modelos, siguiendo una aproximación similar a los modelos de *weaving*.

7.4.2.3 Análisis de calidad de las transformaciones

La calidad del software que se construye es uno de los aspectos que más se tienen en cuenta por parte de las organizaciones [55, 87, 171].

En el caso del Desarrollo de Software Dirigido por Modelos, las transformaciones de modelos son elementos clave debido a que los modelos que definen el sistema evolucionan mediante la ejecución de transformaciones [22, 73, 179, 192]. Como consecuencia, la calidad de las transformaciones de modelos afecta directamente a los artefactos obtenidos [182]. Así, en este tipo de desarrollos, sería deseable asegurar la calidad de las transformaciones de modelos que guían el proceso de desarrollo de software.

Para llevar a cabo la evaluación de la calidad del software, en general, uno de los métodos más empleados es el uso de métricas de software. En el campo de las transformaciones de modelos ya existen algunos trabajos en esta dirección [195, 196, 197, 198]. En este sentido, se plantea el análisis de los trabajos anteriores con el objetivo de determinar su aplicabilidad a MeTAGeM-Trace.

7.4.3 Aplicación de los principios de MDE a la construcción de artefactos MDE

Del mismo modo que en esta tesis se defienden las ventajas de aplicar los principios de MDE al desarrollo de las transformaciones de modelos, sería recomendable y deseable aplicar estos mismos principios al desarrollo de cualquier artefacto software que forme parte de un desarrollo en el ámbito de MDE. Por este motivo, en las siguientes sub-secciones se presentan las líneas de investigación en esta dirección.

7.4.3.1 Desarrollo Dirigido por Modelos de Validación de modelos

La validación de los modelos implicados en un proceso de desarrollo dirigido por modelos puede considerarse como una medida de calidad, dado que permite detectar errores de modelado y establecer restricciones que no pueden contemplarse en la definición de los metamodelos [141, 170]. En general, las

reglas de validación se establecen a nivel de metamodelo, especificando que condiciones deben cumplir cada una de las instancias de las metaclases, por ello, el esfuerzo que se debe realizar a la hora de implementar las reglas de validación es directamente proporcional al número de metaclases contenidas en el metamodelo y al número de restricciones que deben cumplir cada una de dichas metaclases.

Así, si la validación de modelos se considera un medio para mejorar la calidad de los modelos, se debe establecer algún método que facilite la implementación de los módulos que contienen a las reglas de validación, especialmente si estas son complejas y numerosas. A lo largo de esta tesis se ha mostrado cómo el DDM de transformaciones de modelos proporciona una serie de nuevas ventajas y facilidades. Del mismo modo, se pueden obtener ventajas a la hora aplicar los principios de MDE al desarrollo de la validación de los modelos. Entre estas ventajas se encuentran: desarrollos más rápidos ya que al alejar al desarrollador de la complejidad de codificar en un lenguaje concreto, el tiempo de aprendizaje para los nuevos desarrolladores es bastante inferior y es posible crear de forma rápida módulos de validación para distintos lenguajes (EVL, OCL, etc.).

7.4.3.2 Desarrollo Dirigido por Modelos de Editores de modelos

En general, como se ha mostrado en esta tesis y puede comprobarse en otros trabajos anteriores [94, 199], la construcción de editores de modelos, a pesar de la existencia de *frameworks* que facilitan esta tarea, suele implicar bastante trabajo de refinamiento y codificación por parte de los desarrolladores. Por ello, teniendo en cuenta que los editores de modelos son otro artefacto más del sistema y que su construcción resulta compleja, una futura línea de investigación es el desarrollo dirigido por modelos de editores de modelos.

Siguiendo un proceso similar al propuesto por MeTAGeM-Trace, sería posible definir un modelo a nivel PIM que describa la representación de los elementos de un metamodelo, de forma independiente de plataforma. Dicho modelo podría ser transformado en un modelo a nivel PSM que describa dicha representación de acuerdo a las características de las aproximaciones más comunes (representación gráfica y representación textual) y finalmente, se podría definir un nivel PDM que defina la representación de los elementos de acuerdo a las características de los *framework* que facilitan el desarrollo de editores (por ejemplo, EMF y GMF).

Actualmente se está trabajando en esta línea, en el marco de la tesis doctoral de David Granada, becario FPI del Grupo Kybele. En concreto, se está trabajando en el desarrollo de un meta-editor que permita realizar la implementación de editores, tanto gráficos como textuales.

7.4.4 Mejoras en el contexto de la herramienta MeTAGeM-Trace

En esta sección se presentan las futuras líneas de investigación en el contexto de la herramienta MeTAGeM-Trace, presentada en esta tesis.

- **Implementar otros metamodelos de trazabilidad.** Actualmente, la herramienta ofrece soporte para la generación de modelos de trazas conformes a un metamodelo de trazabilidad particular. En dichos modelos se pueden registrar las trazas derivadas de la ejecución de la transformación, los elementos implicados en dichas trazas y el tipo de operación que ha generado la traza (transformación, borrado y creación). Sin embargo, es posible que en algunas ocasiones los usuarios deseen almacenar otro tipo de información adicional, como por ejemplo la fecha de creación de la traza o el usuario que ejecutado la transformación. Del mismo modo, en otros casos puede que se requiera menor nivel detalle. Por ello, como posible trabajo futuro se plantea la implementación de otros metamodelos de trazabilidad que permitan registrar diferente información acerca de las trazas generadas. Otra posibilidad puede ser la definición de un metamodelo base y permitir que sea extendido para generar metamodelos de trazabilidad específicos.
- **Implementar otros metamodelos a nivel PSM para dar otras aproximaciones de desarrollo de transformaciones.** Como se especifica a lo largo de esta tesis, a nivel PSM, MeTAGeM-Trace propone el modelado de las transformaciones siguiendo las diferentes aproximaciones de transformación de modelos existentes (declarativa, imperativa, híbrida, basada en grafos, etc.). Actualmente, en este nivel se soporta el modelado siguiendo la aproximación híbrida. Así, como futuro trabajo se propone el desarrollo de otros DSLs para el resto de las aproximaciones. Esto permitirá que en el momento de realizar el modelado de la transformación, el desarrollador pueda seleccionar la aproximación que considere más adecuada para implementar su transformación. Para especificar los metamodelos de los DSLs de dichas aproximaciones, se deberán analizar diferentes propuestas que siguen tales aproximaciones.

- **Implementar transformaciones entre los metamodelos definidos a nivel PSM.** Al tener implementado varios DSLs a nivel PSM, será interesante poder realizar transformaciones horizontales entre los modelos definidos al mismo nivel pero en diferentes aproximaciones, de forma que, por ejemplo, sea posible pasar (semi-)automáticamente de un modelo de aproximación híbrida a un modelo de aproximación declarativa. Para llevar a cabo esta tarea, se debe especificar e implementar un conjunto de reglas de transformación entre los metamodelos definidos a nivel PSM.
- **Implementar otros lenguajes de transformación a nivel PDM y código.** Una vez se hayan implementado nuevos metamodelos a nivel PSM que describan otras aproximaciones para el desarrollo de transformaciones de modelos, sería deseable definir a nivel PDM y a nivel de código, al menos, un lenguaje de transformación que implemente dicha aproximación. Para ello, dependiendo de las facilidades que proporcionen los desarrolladores, será necesario especificar e implementar un metamodelo que describa la transformación de acuerdo a las características propias del lenguaje, un conjunto de reglas de transformación entre el metamodelo a nivel PSM y el metamodelo a nivel PDM, un conjunto de reglas de validación para los modelos PSM que permite comprobar su validez antes de ser transformados a nivel PDM y un generador de código que serialice los modelos PDM en el código que implementa la transformación.
- **Implementar transformaciones de modelos inversas**. Como se ha comentado anteriormente, una futura línea de investigación es dar soporte a las transformaciones de modelos bidireccionales. Para ello, es necesario especificar e implementar transformaciones de modelos inversas a las ya existentes en MeTAGeM-Trace, es decir, se debe especificar e implementar un conjunto de reglas de transformación para pasar de modelos PDM a modelos PSM y de modelos PSM a modelos PIM.
- **Implementar inyectores de modelos**. Del mismo modo que es posible obtener el código de la transformación a partir de los modelos PDM, es posible obtener los modelos a partir del código. De hecho, el *plug-in* de ATL para Eclipse da soporte para realizar esta tarea. Para implementar estos inyectores, se está realizando un análisis de herramientas como EMFText (http://www.emftext.org) o xText (http://www.eclipse.org/Xtext/).
- **Actualizar la herramienta a las últimas versiones.** Cuando se comenzó a implementar la herramienta MeTAGeM-Trace, se realizó sobre las últimas

versiones disponibles en ese momento para Eclipse, EMF, ATL, Epsilon y MOFScript. Concretamente se han usado las versiones: Eclipse 3.6.1, EMF 2.6, ATL 3.2, Epsilon 0.9 y MOFScript 1.4.0.3. Sin embargo, durante el proceso de construcción de MeTAGeM se han publicado nuevas versiones de Eclipse (3.7), EMF (2.7), ATL (3.2.1) y Epsilon (0.9.1). Debido a la aparición de estas nuevas versiones, uno de los trabajos futuros es la migración de la herramienta MeTAGeM-Trace a dichas versiones.

Apéndice A: Detalles de las Revisiones Sistemáticas

En la sección 2.2 de esta memoria de tesis doctoral se ha presentado el método de identificación del cuerpo de conocimiento el cual consiste en un proceso de revisión sistemática de la literatura [27, 105].

Posteriormente, en la sección 3.3 se ha presentado **una revisión sistemática para la identificación y análisis de propuestas de generación de trazabilidad a partir del desarrollo de transformaciones de modelos**. En dicha sección se presentan los aspectos clave del proceso y el análisis de las propuestas identificadas. Como complemento, en este apéndice se presentan algunos detalles del proceso de revisión sistemática (RS) realizado. En concreto, se presentan las decisiones tomadas en la fase de planificación (sección A.1), los resultados obtenidos y los estudios primarios seleccionados durante la fase de ejecución (sección A.2) y las limitaciones o debilidades del proceso de revisión sistemática realizado (sección A.3).

A.1 Planificación

La fase de planificación consiste en la definición del objetivo de la investigación y del protocolo para llevar a cabo la revisión [27]. En concreto, se deben definir aspectos tales como: preguntas de investigación, estrategias de búsqueda (fuentes de información y términos clave), criterios de inclusión y exclusión y los datos que se extraerán de cada estudio. Así, en esta sección se presentarán estos aspectos.

A.1.1 Cuestiones de Investigación

Uno de los primeros pasos del proceso de RS es definir el objetivo principal de la investigación. En este caso se trata de: **identificar y analizar propuestas centradas en generar y gestionar información de trazabilidad a partir del desarrollo de transformaciones de modelos**. Para alcanzar este objetivo principal, se define un conjunto de cuestiones de investigación (CI). En esta RS se han definido las siguientes CI:

- **CI-1.** ¿Qué nivel de automatización ofrecen o sugieren las propuestas metodológicas para generar información de trazabilidad?

  Aunque en este estudio de la literatura el objetivo principal no es analizar si las propuestas aplican los principios de MDE al desarrollo de las transformaciones, sí que resulta interesante conocer qué tareas es posible automatizar y hasta qué nivel. Así será posible conocer si, en el estado del arte actual, la generación de información de trazabilidad se considera una tarea manual o si por el contrario, se proporcionan métodos que la automaticen (completa o parcialmente).

- **CI-2.** Según las propuestas metodológicas identificadas, ¿cómo se deberían almacenar, visualizar y analizar las trazas generadas?

  Como se ha indicado en la introducción de este documento, la información de trazabilidad puede ser muy útil para llevar a cabo otras actividades del ciclo de vida software, como por ejemplo, el análisis del impacto del cambio, la toma de decisiones o el mantenimiento general del sistema [5, 37, 38, 107, 142]. Por tanto, resulta interesante conocer cómo, según las propuestas metodológicas, deberían llevarse a cabo ciertas tareas relacionadas con la gestión de las trazas como el almacenamiento (en modelos de trazas, en repositorios de trazas, embebidas en los modelos terminales, etc.), la visualización (de forma gráfica o textual) y las técnicas para facilitar su análisis (informes técnicos, estudios estadísticos, clasificaciones, etc.).

- **CI-3.** ¿Existen herramientas o marcos de trabajo que den soporte tecnológico para la gestión de la trazabilidad en el contexto del desarrollo de transformaciones de modelos?

  Una de las principales metas de este estudio de la literatura es identificar si las propuestas para la generación y gestión de la trazabilidad a partir del desarrollo de transformaciones de modelos han sido puestas en práctica por medio de herramientas MDE. Por tanto, el objetivo de esta cuestión de investigación es analizar si las propuestas ofrecen a los usuarios algún tipo de soporte tecnológico.

- **CE-4**. ¿Cuáles son las limitaciones encontradas en las propuestas para la generación de trazabilidad a partir del desarrollo de transformaciones de modelos?

Finalmente, en la última cuestión se combinarán las respuestas a las preguntas anteriores (CI-1, CI-2 y CI-3) para identificar problemas o limitaciones en el estado del arte actual del tema objetivo.

A.1.2 Fuentes de Información y cadenas de búsqueda

Como se ha comentado al inicio de esta sección, en la fase de planificación también se debe definir una estrategia de búsqueda para los estudios que contribuirán a la revisión sistemática. Dicha estrategia consiste, básicamente, en definir dónde se buscarán los estudios y qué palabras clave se usarán para realizar las búsquedas [27, 105].

Debido a las facilidades que ofrece Internet, en cuanto a cantidad de información disponible, para llevar a cabo esta RS se ha decidido realizar las búsquedas en **fuentes de información digitales** que se encuentran disponibles de forma *online*. En concreto, se han seleccionado las siguientes bibliotecas de datos:

- ACM Digital Library (ACM).
- CiteSeerX (CSX).
- IEEEXplore (IEEEX).
- Google Scholar (GS).
- ISI Web of Knowledge (ISI).
- Science Direct (SD).
- SpringerLink (SL).
- The Collection of Computer Science Bibliographies (CSB).

Para realizar las búsquedas de los estudios en estas fuentes de información se ha establecido un conjunto de palabras clave relacionadas con el objetivo principal de la investigación y combinadas mediante operadores lógicos. Así, la **cadena de búsqueda** empleada en esta RS es la siguiente:

(meta-traceability or traceability) AND (mda OR mdsd OR mdd OR mde OR model driven OR model transformation).

Es necesario mencionar que los motores de búsqueda seleccionados funcionan bajo una sintaxis concreta, por ello, esta cadena de búsqueda ha tenido que ser adaptada a la sintaxis de cada uno de los motores de búsqueda.

A.1.3 Criterios de Inclusión y Exclusión

Aunque la cadena de búsqueda se ha definido teniendo en cuenta el objetivo principal de investigación, es posible que un número considerable de los estudios obtenidos a partir de las búsquedas realizadas no proporcionen información relevante acerca de dicho objetivo. Así, de acuerdo con [105], es necesario definir criterios de inclusión y exclusión para filtrar a los estudios obtenidos.

En este proceso de RS se han incluido aquellos estudios que hayan sido publicados *online* antes de Marzo de 2011 que cumplan al menos uno de los siguientes **criterios de inclusión**:

- Su resumen (*abstract*) indica que el objetivo del trabajo es la gestión o generación de la trazabilidad en el ámbito de MDE.
- Su título o palabras clave incluyen los términos (o sus derivados): "trazabilidad" y "dirigido por modelos" o "trazabilidad" y "transformación de modelos".

A partir de los criterios de inclusión definidos es posible realizar un primer filtrado de los estudios obtenidos, sin embargo, es posible que muchos de ellos no proporcionen información suficientemente relevante para el objetivo buscado. Así, se procede a leer por completo estos trabajos y se excluye aquellos que cumplan alguno de los siguientes **criterios de exclusión**:

- Consideren la gestión o generación de la trazabilidad como una tarea deseable pero no proporcionen información acerca de cómo abordarla.
- Estén dirigidos a la gestión o generación de la trazabilidad sin tener en cuenta a los principales artefactos de MDE: modelos y transformaciones de modelos.
- Su principal objetivo es clasificar otros trabajos o sean revisiones sistemáticas en sí mismos.

A.1.4 Criterios de Evaluación

Una vez que se han seleccionado aquellos trabajos que cumplen con los criterios de inclusión y no han sido excluidos por los de exclusión, sería deseable disponer de mecanismos que permitan evaluar y comparar dichos estudios. Por tanto, de acuerdo a la guía propuesta por Kitchenham y Charters [105], para evaluar las propuestas y proporcionar una comparativa cuantitativa entre ellas, en

esta RS se ha definido un conjunto de **criterios de evaluación (CE)** y un conjunto de **valores** que pueden adoptar dichos criterios. Los valores que se les pueden asignar para cada uno de los criterios de evaluación son: Sí (S) = 1; Parcial = 0.5; y No (N) = 0. Los criterios de evaluación para las propuestas centradas en el DDM de transformaciones de modelos son:

- **CE-1**. ¿La propuesta metodológica propone especificar la transformación en los niveles PIM y PSM definidos por MDA [153] antes de ser serializada en código?

  Valores: Sí (S), la propuesta define explícitamente que una transformación de modelos debe ser especificada al menos en estos dos niveles o equivalentes; Parcial (P), la propuesta contempla que la transformación sea definida, al menos, a nivel PIM o PSM o equivalentes; No (N), la propuesta no contempla el modelado de la transformación a ninguno de estos niveles.

- **CE-2.** ¿La propuesta metodológica sugiere cómo automatizar el paso entre los diferentes niveles de abstracción a los que se especifica la transformación?

  Valores: S, proporciona métodos para automatizar o semi-automatizar la conversión de modelos a distintos niveles de abstracción, por ejemplo, mediante transformaciones entre dichos modelos; P, no indica cómo automatizar el proceso, sin embargo, considera que es una característica deseable; N, el paso entre los niveles es totalmente manual.

- **CE-3.** ¿La propuesta metodológica especifica cómo generar el código que implementa la transformación a desarrollar?

  Valores: S, indica cómo llevar a cabo la serialización de los modelos de bajo nivel que describen la transformación; P, no especifica cómo ponerlo en práctica, pero identifica a la generación de código como una característica deseable; N, no considera la generación del código de la transformación.

- **CE-4.** ¿Propone algún mecanismo para visualizar los modelos de transformación?

  Valores: S, propone el desarrollo de un editor o visualizador *ad-hoc*, considerando la naturaleza de los modelos de transformación; P, la representación de los modelos puede no ser la más óptima, ya que considera la visualización a través de métodos genéricos para cualquier tipo de modelo, como por ejemplo el editor en forma de árbol que ofrece EMF

[188]; N, la propuesta no considera la visualización de los modelos o no sugiere ninguna forma de representación.

- **CE-5.** ¿La propuesta metodológica proporciona o sugiere mecanismos de validación de los modelos de transformación, de forma que los errores no se propaguen a los modelos de menor nivel de abstracción?

  Valores: S, indica cómo deberían validarse los modelos de transformación para asegurar que son correctos; P, no indica cómo deben validarse los modelos, pero considera que es una característica deseable; N, no considera en absoluto la validación de modelos.

- **CE-6.** ¿La especificación a alto nivel de la transformación es independiente de cualquier lenguaje de transformación concreto?

  Valores: S, la definición de la transformación a alto nivel es independiente de cualquier lenguaje de transformación; P, la especificación de la transformación no es completamente independiente, o es dependiente de un lenguaje concreto pero consideran deseable que fuera independiente; N, la especificación de la transformación es totalmente dependiente de un lenguaje concreto.

- **CE-7.** ¿La propuesta metodológica considera la gestión de la trazabilidad a partir del DDM de transformaciones de modelos?

  Valores: S, propone cómo gestionar la trazabilidad a partir del desarrollo de transformaciones; P, no indica cómo, pero considera que sería posible gestionar trazabilidad a partir de las transformaciones; N, no considera en absoluto la gestión de la trazabilidad.

- **CE-8.** ¿Proporciona alguna implementación, herramienta, o soporte tecnológico?

  Valores: S, proporciona una herramienta o marco de trabajo completo para dar soporte a la propuesta metodológica; P, proporciona una implementación parcial de la propuesta; N, se trata de una propuesta puramente teórica que no ofrece implementación para ponerla en práctica.

A.1.5 Extracción de Datos y Análisis

Finalmente, en la fase de planificación también se deben definir los datos que se recogerán de cada uno de los trabajos y sus propuestas. A partir de estos datos será posible dar respuesta a los criterios de evaluación y a las cuestiones de

investigación, establecidas anteriormente. Así, en esta RS se ha decidido extraer la siguiente información:

- Título, autores, resumen y año de publicación de cada estudio identificado (perteneciente a la propuesta).
- Niveles de abstracción que considera para el modelado de la transformación.
- Si automatiza el DDM de transformaciones de modelos.
- Si proporciona o sugiere cómo sería posible generar el código que serializa los modelos de la transformación.
- Si sugiere cómo deberían visualizarse los modelos que especifican la transformación.
- Si considera o recomienda la validación de los modelos que especifican la transformación.
- Si considera la gestión de la trazabilidad a partir del DDM de transformaciones de modelos.
- Si ofrece soporte tecnológico y cómo lo hace (si aplica).

Con el objetivo de simplificar el análisis de esta información y de los estudios en general, se ha decidido organizar los estudios en Grupos de Estudios Primarios (GPS, *Group of Primary Studies*). Cada GPS contiene a aquellos estudios que tienen uno o más autores en común y las ideas que se defienden en dichos artículos se corresponden a una misma línea de investigación o al menos, son una evolución de una misma hipótesis inicial.

A.2 Ejecución

La fase de ejecución de la RS consiste básicamente en la ejecución de las búsquedas y la selección de los estudios primarios que contribuyen a dar respuesta al objetivo de investigación planteado. Por ello, en esta sección se presentan los resultados obtenidos en la ejecución de las búsquedas (sección A.2.1) y la lista de los estudios primarios seleccionados (sección A.2.2).

A.2.1 Búsqueda de Estudios Primarios

En el proceso de búsqueda, con el objetivo de maximizar los resultados obtenidos, se ha decidido ejecutar las búsquedas en el mayor ámbito que permiten cada uno de los motores. En este contexto, ámbito es entendido como la parte de los estudios en los que se buscarán las cadenas de búsqueda. En la Tabla A-1, se presentan el ámbito seleccionado para cada motor de búsqueda así como la cadena de búsqueda adaptada.

Tabla A-1. Cadenas de Búsqueda adaptas y ámbito de búsqueda

Fuente	Cadena de Búsqueda Adaptada	Ámbito
ACM	(meta-traceability and traceability) and (mda or mdsd or mdd or mde or "model driven" or “model transformation”)	Title, Abstract, Review
CSX	((meta-traceability AND traceability) AND (mda OR mdsd OR mdd OR mde OR "model driven" OR “model transformation”))	Text
IEEEX	((meta-traceability AND traceability) AND (mda OR mdsd OR mdd OR mde OR "model driven" OR “model transformation”))	All
GS	(meta-traceability + traceability) + (mda OR mdsd OR mdd OR mde OR "model driven" OR “model transformation”)	All
ISI	TS=((meta-traceability AND traceability) AND (mda OR mdsd OR mdd OR mde OR "model driven" OR “model transformation”))	Topic
SD	ALL((meta-traceability AND traceability) AND (mda OR mdsd OR mdd OR mde OR "model driven" OR “model transformation”))	All
SL	((meta-traceability AND traceability) AND (mda OR mdsd OR mdd OR mde OR "model driven" OR “model transformation”))	All
CSB	(meta-traceability + traceability) +(mda mdsd mdd mde "model driven" “model transformation”)	Author, Title

Como resultado de la ejecución de estas búsquedas, se ha obtenido un total de **10.028 resultados**. Las dos primeras columnas de la Tabla A-2 muestran el número de estudios devueltos por cada una de las fuentes de información.

Para cada uno de estos estudios se ha evaluado su comportamiento frente a los criterios de inclusión y tan solo 267 de ellos, es decir el 2,66%, los han cumplido, y en consecuencia, se han convertido en **estudios relevantes** de la RS. En la tercera columna de la Tabla A-2 (“Estudios Relevantes”) puede comprobarse el número de estudios seleccionados de cada fuente de información. Además, en la columna 4, se muestra el porcentaje de estudios relevantes de cada fuente de información respecto del total de resultados proporcionados por dicha fuente. Así, por ejemplo, los 40 estudios relevantes obtenidos de ACM suponen un 9,71% del total de resultados proporcionados por la fuente (412 estudios).

Finalmente, en la última columna se muestra el porcentaje que supone el número de estudios relevantes identificados en una fuente de información respecto del total obtenido, es decir 267. Así, a modo de ejemplo, los 40 estudios relevantes de ACM representan el 14,98% de los 267 estudios seleccionados como relevantes para la RS.

Tabla A-2. Resultados de las búsquedas

Fuente	Resultados	Estudios Relevantes	% de estudios relevantes	% sobre el total de estudios relevantes
ACM	412	40	9.71 %	14.98 %
CSX	451	26	5.76 %	9.74 %
IEEEX	923	21	2.28 %	7.87 %
GS	6980[3]	82	1.17 %	30.71 %
ISI	86	25	29.07 %	9.36 %
SD	259	9	3.47 %	3.37 %
SL	863	47	5.45 %	17.60 %
CSB	54	17	31.48 %	6.37 %
Total	**10028**	**267**	**2.66 %**	**100 %**

Dado que estas fuentes de información pueden compartir un alto número de estudios, es posible que parte de los estudios seleccionados hasta el momento, se hayan encontrado en varias fuentes, es decir entre los 267 estudios relevantes pueden existir varias copias de un mismo trabajo. Por ello, el siguiente paso consiste en identificar dichas copias y eliminarlas.

Como se muestra la Tabla A-3, más del 40% de los estudios relevantes son copias repetidas de otros estudios relevantes. Así, una vez eliminadas estas duplicidades, se obtiene una lista de 157 estudios que son evaluados respecto del criterio de exclusión. Como resultado, solo 20 de ellos (es decir, un 12,78% de los estudios relevantes no duplicados) se convierten en estudios primarios de la RS. La lista completa de estos estudios primarios se presenta en la siguiente sección.

[3] Google Scholar solo muestra los 1000 primeros resultados.

Tabla A-3. Filtrado de estudios duplicados

	Estudios	Porcentaje
Estudios relevantes	267	100 %
Estudios relevantes duplicados	110	41.20 %
Estudios relevantes NO duplicados	157	58.80 %

A.2.2 Estudios Primarios seleccionados

Una vez ejecutadas las búsquedas, evaluado los estudios de acuerdo a los criterios de inclusión y exclusión y eliminado las duplicidades, se han identificado los siguientes estudios primarios que contribuyen a la RS:

Tabla A-4. Lista de Estudios Primarios

Autores	Estudios	Ref.
Allilaire	Towards traceability support in ATL with Obeo Traceability	[7]
Barbero, Del Fabro y Bézivin	Traceability and provenance issues in global model management	[17]
Bonde, Boulet y Dekeyser	Traceability and interoperability at different levels of abstraction in model-driven engineering	[32]
Boronat, Carsí y Ramos	Automatic support for traceability in a generic model management framework	[34]
Drivalos, Kolovos, Paige y Fernandes	A state-based approach to traceability maintenance	[57]
Drivalos, Kolovos, Paige y Fernandes	Engineering a DSL for software traceability	[56]
Drivalos, Paige, Fernandes y Kolovos	Towards Rigorously Defined Model- to-Model Traceability	[58]
Falleri, Huchard y Nebut	Towards a traceability framework for model transformations in kermeta	[63]
Grammel y Kastenholz	A generic traceability framework for facet-based traceability data extraction in model-driven software development	[77]
Guerra, de Lara, Kolovos y Paige	Inter-modelling: From theory to practice	[81]
Jouault	Loosely Coupled Traceability for ATL	[97]
Kolovos, Paige y Polack	On-Demand Merging of Traceability Links with Models	[111]
Kurtev, Van den Berg y Jouault	Evaluation of rule-based modularization in model transformation languages illustrated with ATL	[121]
Levendovszky, Balasubramanian, Smyth, Shi y Karsai	A transformation instance-based approach to traceability	[128]

Melby	Traceability in Model Driven Engineering	[138]
Oldevik y Neple	Traceability in Model to Text Transformations	[146]
Olsen and Oldevik	Scenarios of Traceability in Model to Text Transformations	[149]
Paige, Drivalos, Kolovos, Fernandes, Power, Olsen y Zschaler	Rigorous identification and encoding of trace-links in model-driven engineering	[159]
Sánchez, Alonso, Rosique, Álvarez y Pastor	Introducing safety requirements traceability support in model-driven development of robotic applications	[175]
Valderas y Pelechano	Introducing requirements traceability support in model-driven development of web applications	[193]

A.3 Limitaciones

Como se ha mencionado en varias ocasiones a lo largo de esta memoria de tesis doctoral, para llevar a cabo el proceso de revisión sistemática se han seguido las guías propuestas por Kitchenham y Charters [105] y Biolchini [27]. No obstante, el proceso seguido se desvía ligeramente de las pautas indicadas por dichas guías en los siguientes aspectos:

- No se han realizado búsquedas manuales.

  El proceso de búsqueda de estudios se ha realizado únicamente mediante la ejecución de búsquedas en fuentes de información *online*. Esta desviación puede hacer que no se hayan identificado algunos trabajos publicados en revistas o actas de congresos que no se encuentren *online*.

- No se ha seguido el criterio PICOC (*Population, Intervention, Comparison, Outcome and Context.* En español: población o problema, intervención, comparación, resultados o salidas y contexto) para la definición de las cuestiones de investigación.

  En la guía para el desarrollo de revisiones sistemáticas de Kitchenham y Charters [105], para definir las cuestiones de investigación se recomienda seguir el criterio PICOC, propuesto por Petticrew y Roberts en [162]. A pesar de que no se haya empleado este criterio de forma explícita para la definición de las preguntas de investigación, es cierto que se cumplen algunos de los componentes de este criterio. Por ejemplo, el contexto de las cuestiones definidas es ‘MDE’ y la población (problema) es la ‘generación de trazabilidad’. Aunque esta desviación puede implicar que las cuestiones de investigación definidas pueden ser de alguna manera desestructuradas o

subjetivas, es necesario mencionar que son resultado de un proceso iterativo en el que han sido mejoradas y refinadas en varias ocasiones.

- La revisión no ha sido revisada por expertos externos.

  A pesar de no haber contactado con expertos externos para la evaluación de la revisión sistemática, es necesario mencionar que una versión inicial de algunos de los principales resultados y conclusiones extraídos fue presentada en las Jornadas de Ingeniería del Software y Bases de Datos (JISBD), celebradas en A Coruña en septiembre de 2011 [95].

  Del mismo modo, también es necesario destacar que, durante el desarrollo de la revisión sistemática, el doctorando hizo el papel de investigador mientras que los directores de esta tesis y otros miembros del Grupo Kybele actuaron como revisores proporcionando evaluaciones internas.

Apéndice B: Manual de Usuario de MeTAGeM-Trace

En este apéndice se presenta el manual de usuario completo de la herramienta MeTAGeM-Trace, presentada en el capítulo 5 de esta tesis doctoral.

B.1 Introducción

MeTAGeM-Trace es una meta-herramienta para el desarrollo transformaciones de modelos. Proporciona un proceso dirigido por modelos y su principal característica es que las transformaciones obtenidas incluyen un generador de trazas, es decir, cuando sean ejecutadas, además de los modelos destino, crearán modelos que contienen las trazas entre los elementos implicados en la transformación.

El funcionamiento de la herramienta consiste básicamente en el modelado a alto de nivel de la transformación, realizado por el usuario, y por medio de la generación semi-automática de un conjunto de modelos intermedios que la describen a menor nivel de abstracción, se proporciona el código que implementa la transformación (e incluye el generador de trazas) en un lenguaje de transformación seleccionado por el usuario.

Para ofrecer esta funcionalidad, la herramienta se compone de un conjunto de lenguajes específicos de dominio (DSLs, *Domain Specific Languages* [213, 215]) que permiten el modelado y la visualización de la transformación a los diferentes niveles de abstracción propuestos por MeTAGeM-Trace así como de las trazas generadas. Estos DSLs, para modelar la transformación, se dividen en cuatro niveles de abstracción:

- **Nivel independiente de plataforma (PIM)**, donde el usuario debe definir las relaciones entre los elementos de los metamodelos implicados en la transformación. Este nivel se representa mediante los modelos MeTAGeM (extensión *.metagem*).
- **Nivel específico de plataforma (PSM)**, donde se describen las reglas de la transformación y las relaciones de trazabilidad, de acuerdo a un paradigma o aproximación de desarrollo de transformaciones. En este caso, la herramienta ofrece soporte para la aproximación híbrida. Este nivel se representa mediante los medios de aproximación híbrida (extensión *.hybrid*).
- **Nivel dependiente de plataforma (PDM)**, donde se especifica la transformación en términos del metamodelo que describe un lenguaje de transformación concreto. De acuerdo a la aproximación soportada a nivel

PSM, MeTAGeM-Trace permite la definición de modelos conformes a los metamodelos de dos lenguajes de transformación híbridos: ATL [101] (ficheros –*atl.ecore*) y ETL [113] (ficheros .*etl_model*).

- **Código**. Implementa la transformación modelada en el lenguaje de transformación seleccionado por el usuario. En este caso, ATL o ETL.

B.2 Instalación

En esta sección se detallan los pasos para llevar a cabo la instalación de la herramienta MeTAGeM-Trace:

- **Instalar la Máquina Virtual de Java.** En concreto, para el desarrollo de la herramienta se ha empleado la versión 1.6, por lo que se recomienda utilizar una versión igual o superior.
- **Instalar el entorno Eclipse y la herramienta MeTAGeM-Trace**. Para realizar esta tarea existen dos posibilidades:
  - **Recomendado**: Descargar la distribución *Eclipse-MeTAGeM-Trace* que contiene el entorno *Eclipse Modeling Tools* y todos los *plug-ins* necesarios para el funcionamiento de la herramienta MeTAGeM-Trace. Se encuentra disponible para su descarga en:

    http://www.kybele.es/members/ajimenez/thesis/software/Eclipse-MeTAGeM-Trace.zip

    Una vez finalizada la descarga, sólo se debe descomprimir el archivo.
  - Instalar Eclipse y MeTAGeM-Trace por separado.

    Si no se dispone del entorno Eclipse, está disponible para su descarga en: http://www.eclipse.org/downloads/. De entre las distribuciones disponibles, se recomienda *Eclipse Modeling Tools*.

    Para el correcto funcionamiento de MeTAGeM-Trace, Eclipse debe tener instalados los siguientes *plug-ins*:
      - EMF 2.6
      - ATL 3.2
      - Epsilon 0.9
      - MOFScript 1.4.0.3

    Una vez que se dispone de Eclipse y los *plug-ins* mencionados, puede instalarse el conjunto de *plug-ins* de MeTAGeM-Trace, que se encuentra disponible para su descarga en:

    http://www.kybele.es/members/ajimenez/thesis/software/MeTAGeM-Trace-Plugins.zip

B.3 Modelado de la Transformación Independiente de Plataforma

Para comenzar con el modelado de las transformaciones independientes de plataforma, el primer paso consiste en la creación de un modelo de tipo MeTAGeM. Para ello, MeTAGeM-Trace proporciona un asistente de creación de modelos al que se puede acceder a traves del menú de Eclipse: *File→New→Other→MeTAGeM-Trace→MeTAGeM* (Figura B-1). De la misma forma, pueden crearse el resto de modelos de los DSLs que forman parte de la herramienta MeTAGeM-Trace.

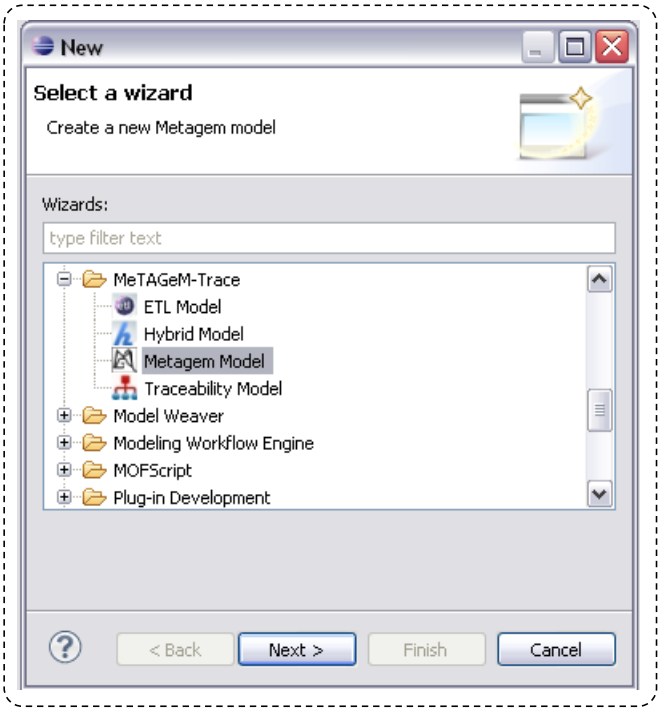

Figura B-1. Selección del Asistente de Creación para modelos MeTAGeM

Una vez seleccionado el tipo de modelo a crear, se debe seleccionar la ruta en la que se almacenará el nuevo modelo y posteriormente, como muestra la Figura B-2, se deben seleccionar los metamodelos origen y destino de la transformación que se va a desarrollar.

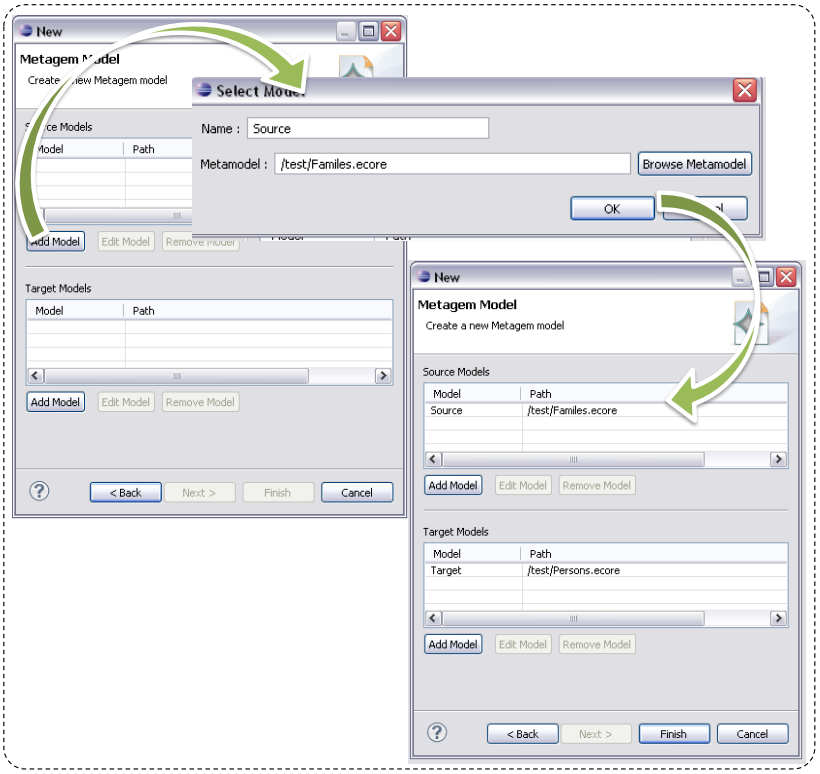

Figura B-2. Creando el modelo de la Transformación Independiente de Plataforma

A partir de la información introducida por el usuario en el asistente, se crea el nuevo modelo MeTAGeM, donde se pueden definir las relaciones de transformación entre los elementos de los metamodelos origen y destino.

Como se muestra en la Figura B-3, MeTAGeM-Trace proporciona un visor/editor compuesto por tres paneles para representar: metamodelos origen (izquierda), modelo que define la transformación a alto nivel (centro) y metamodelos destino (derecha).

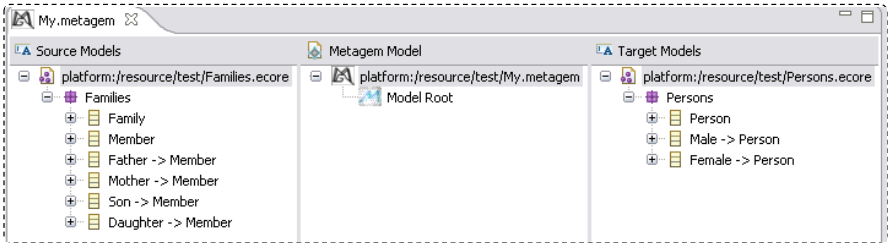

Figura B-3. Nuevo modelo de la Transformación Independiente de Plataforma

El siguiente paso en la definición del modelo de transformación es la especificación de las diferentes relaciones entre los elementos de los metamodelos implicados en la transformación. Para ello, se debe hacer clic con el botón secundario sobre el elemento raíz del modelo (*Model Root*) y en la opción *New child* seleccionar el tipo de relación que se quiere definir (Figura B-4).

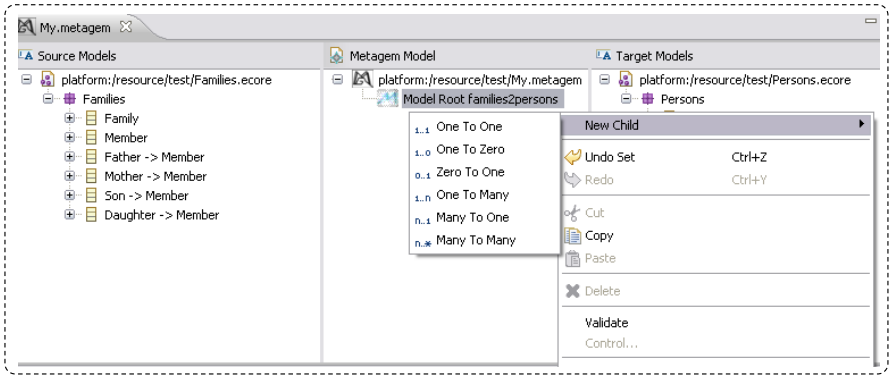

Figura B-4. Tipos de Relaciones

Como se puede comprobar en la Figura B-4 y de acuerdo a la especificación del metamodelo PIM (sección 4.2.2), se pueden crear diferentes tipos de relaciones, dependiendo de su cardinalidad. Así por ejemplo, el tipo de relación *OneToOne* sirve para indicar que un único elemento del metamodelo origen se relaciona con un único elemento del metamodelo destino, de la misma manera, la relación *ManyToOne* sirve para indicar que a partir de varios elementos del metamodelo origen se genera un único elemento en el metamodelo destino.

En la Figura B-5, se muestra cómo especificar los elementos origen y destino que participan en una relación *OneToOne*. En este caso, el elemento origen ya ha sido seleccionado anteriormente (*Father*). Para definir el elemento destino, se selecciona el elemento en el metamodelo destino (*Male*) y se arrastra hasta el elemento *OneToOne* del modelo de la transformación (panel central). El editor de modelos ha sido implementado de forma que el usuario solo tiene que arrastrar el elemento y es la propia herramienta la encargada de identificar si dicho elemento es origen o destino, dependiendo del metamodelo al que pertenezca. Además, es importante mencionar que el editor solo permite arrastrar elementos mientras su cardinalidad no se haya completado, es decir, al definir una relación de tipo *OneToOne* sólo es posible seleccionar un elemento origen y un elemento destino. Así, una vez que se hayan seleccionado estos elementos se deshabilita la posibilidad de seguir agregando más.

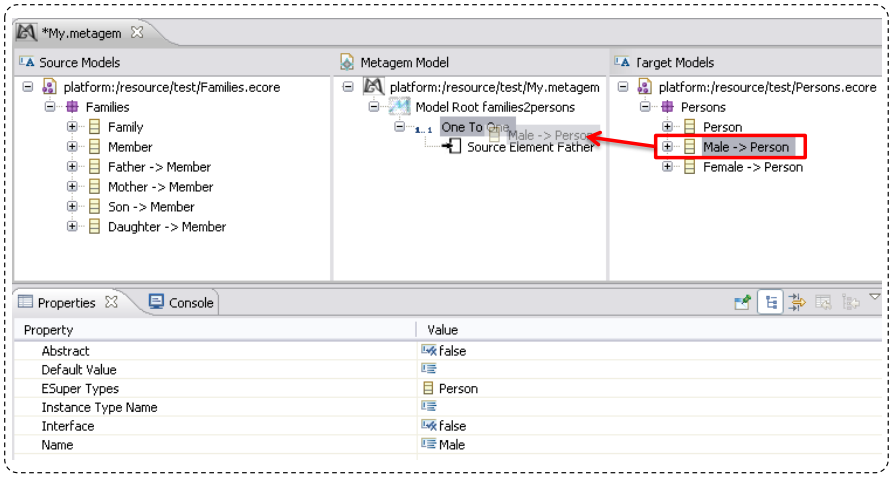

Figura B-5. Especificando el elemento destino en una relación *OneToOne*

De la misma forma se pueden crear relaciones incluidas en elementos destino. El objetivo de estas relaciones es definir transformaciones entre los atributos de los elementos. Así, por ejemplo, como muestra la Figura B-6, se especifica que la propiedad *fullName* del elemento *Male* se genera a partir de la propiedad *firstName* del elemento *Father*.

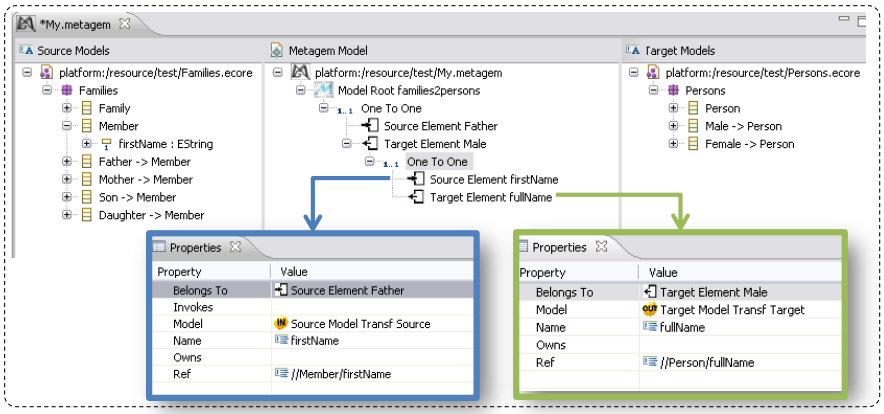

Figura B-6. Creando una relación incluida en un elemento destino

B.4 Modelado de la Transformación Específica de Plataforma

A partir del modelo de nivel independiente de plataforma definido en la sección anterior, es posible obtener de forma semi-automática el modelo de transformación específico de plataforma. Como se ha comentado anteriormente, la implementación actual de la herramienta MeTAGeM-Trace permite el modelado a nivel PSM de las transformaciones siguiendo la aproximación híbrida.

Para obtener el modelo de la transformación a nivel PSM se debe hacer clic con el botón secundario sobre el modelo especificado a nivel PIM (modelo MeTAGeM), y seleccionar *Run As* → *MeTAGeM* →*Hybrid*. En dicho menú contextual, la herramienta MeTAGeM-Trace solo muestra las transformaciones disponibles para el tipo de modelo seleccionado. Así, en este caso, la única transformación disponible para un modelo con *metagem* es *MeTAGeM* → *Hybrid*. Si el modelo MetAGeM es válido (sección 4.4.1), como resultado, se obtiene un modelo conforme al metamodelo de nivel PSM (aproximación híbrida) que contiene las reglas de la transformación y las relaciones de trazabilidad entre los elementos de los metamodelos. En la Figura B-7 se muestra el proceso para obtener el modelo PSM.

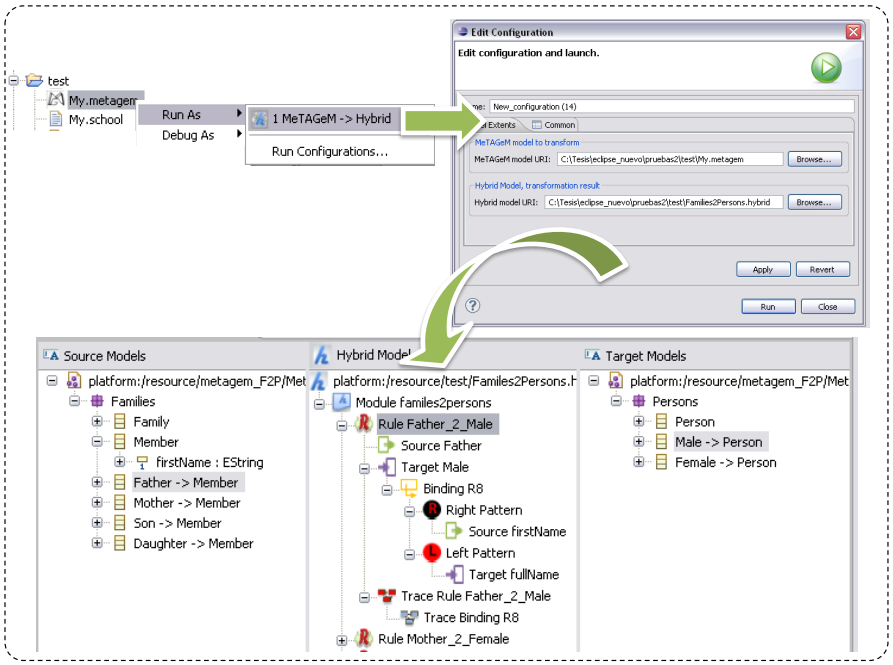

Figura B-7. Generando un modelo Hybrid a partir de un modelo MeTAGeM

La herramienta MeTAGeM-Trace permite refinar el modelo generado, agregando, modificando o eliminando elementos.

El modelo de la transformación a nivel PIM, sólo permite establecer relaciones entre los elementos de los metamodelos, por lo que el modelo generado a nivel PSM sólo contiene elementos de tipo *Rule* y de tipo *TraceLink* (*TraceRule* o *Trace Binding*). Sin embargo, muchas veces es necesario definir algún tipo de función especial que genere o recupere algún tipo de información. Para ello, se pueden definir elementos de tipo *Operation*. En la Figura B-8 se muestra la forma de crear un elemento de este tipo. Para ello se debe hacer clic con el botón secundario sobre el elemento raíz del modelo y seleccionar la opción *New Child* → *Operation*.

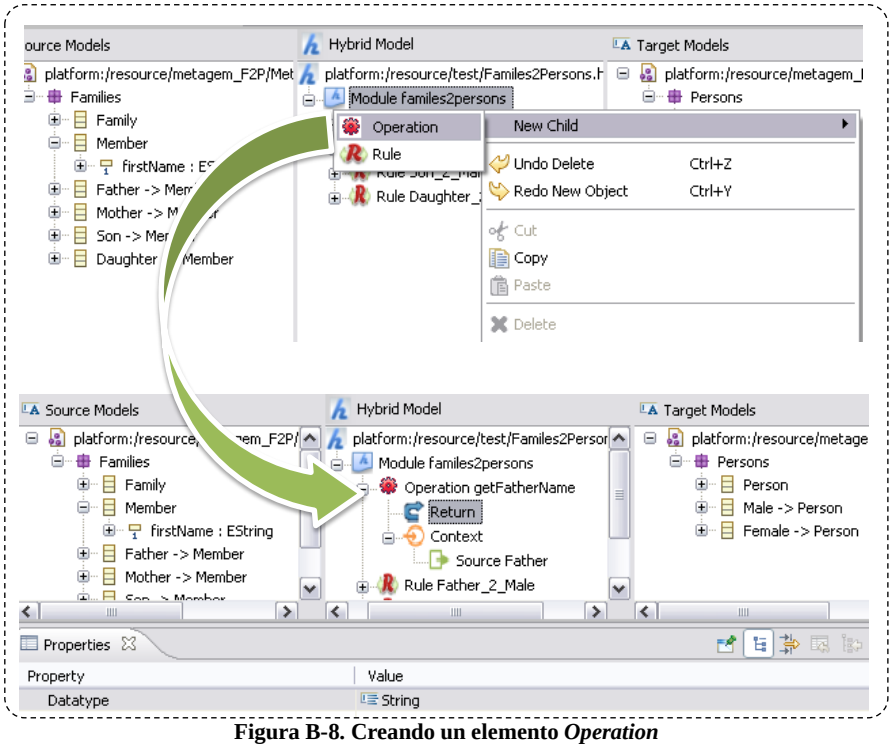

Figura B-8. Creando un elemento *Operation*

Una vez creado el elemento se deben completar sus propiedades y elementos hijos. En el ejemplo que se muestra en la Figura B-8 se ha creado un elemento de tipo *Operation* cuyo atributo *name* es 'getFatherName', su valor de retorno es 'String' y su contexto, el elemento origen *Father*. Del mismo modo, se pueden definir los parámetros de la operación y cuerpo de la misma.

Otro de los aspectos que debe ser modelado de forma manual por el usuario es el valor de los elementos de tipo *RightPattern*. En una relación de tipo *Binding*, el *RightPattern* se encarga de describir el valor que se asigna al elemento *LeftPattern*. Cuando el valor de la asignación se corresponde con un elemento origen como, por ejemplo, en la relación creada en la Figura B-6 (*fullName:=firstName*) es posible definirlo a nivel PIM. En cambio, cuando el valor que se asigna es resultado de una operación, es un valor concreto y se refiere a otro elemento destino, el usuario debe especificarlo a través de los atributos del elemento *RightPattern.*

A modo de ejemplo, en la Figura B-9, se muestra cómo definir una operación creada anteriormente (*getFatherName*) como origen para una relación de tipo *Binding*. Así, a partir de ahora al atributo destino *fullName* se le asignará el resultado de la operación: *fullName:= getFatherName().*

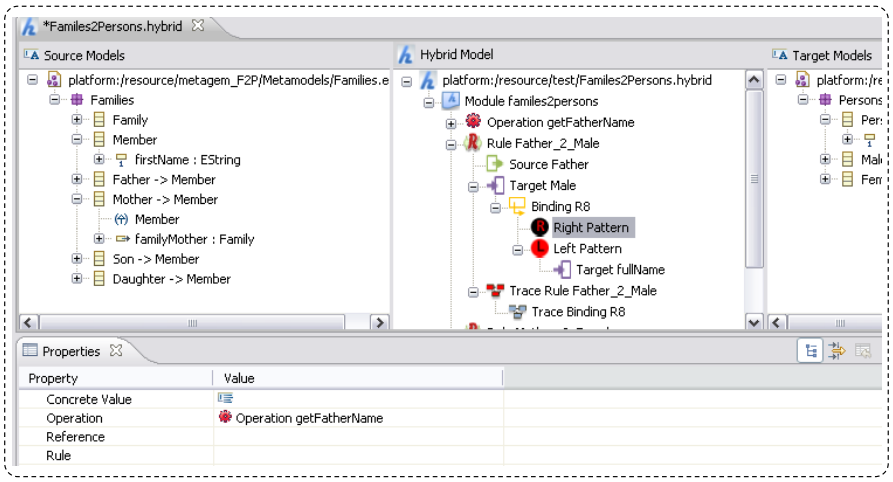

Figura B-9. Definiendo el atributo *Operation* de un elemento *RightPattern*

Continuando con la definición de las relaciones de tipo *Binding*, en los elementos origen de las mismas, es necesario que el usuario defina explícitamente a qué elementos origen de la regla pertenecen (atributo *belongsTo*). En el caso de los elementos destino no es necesario porque cada relación *Binding* pertenece a un elemento destino de la regla, por tanto, los elementos destino del *Binding* pertenecen al él.

A modo de ejemplo, en la Figura B-10 puede verse una relación *mother2female* que contiene dos elementos origen (*member* y *mother*) y un elemento destino (*female*). En el elemento destino se ha definido una relación *binding* para modelar la creación del atributo *fullName* de dicho elemento. Para

definir el valor de este atributo se emplean dos elementos entrada (*firstName* y *familyMother.lastName*). Para obtener, en niveles inferiores, la implementación correcta de esta asociación es necesario indicar la pertenencia de estos sub-elementos. Así, el elemento *familyMother.lastName* pertenece al elemento entrada de la regla *Mother*.

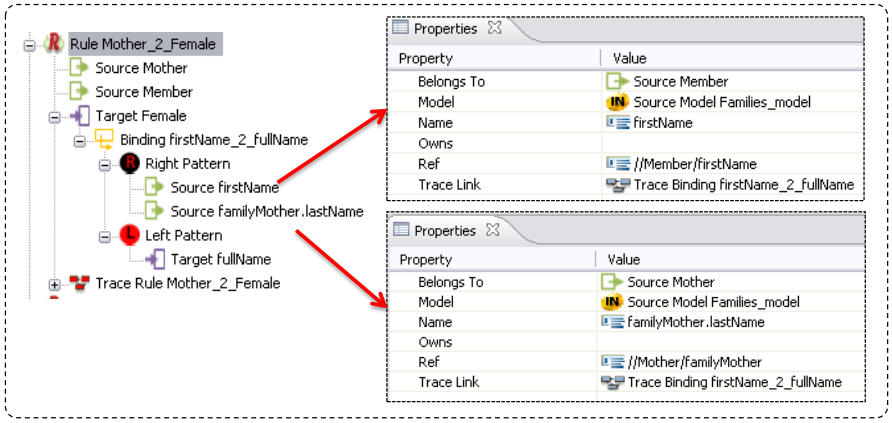

Figura B-10. Definiendo los atributos *BelongsTo*

Es necesario destacar que cuando la regla a la que pertenece el *Binding* contiene un solo elemento origen, la herramienta realiza de forma automática la asignación del atributo *belongsTo*.

B.5 Modelado de la Transformación Dependiente de Plataforma

A nivel PDM, la versión actual de MeTAGeM-Trace propone el modelado de las transformaciones de acuerdo a dos lenguajes de transformación que siguen la aproximación híbrida: ATL y RubyTL. Por lo tanto, el usario debe seleccionar en que lenguaje desea implementar las transformaciones antes de realizar la transformación.

Como muestra la Figura B-11, la acción *Run As* del menú contextual de los modelos PSM, que describen la transformación de acuerdo a las características de la aproximación híbrida (modelos *hybrid*), proporciona dos opciones de transformación: *Hybrid → ATL* y *Hybrid → ETL*. Si se selecciona la primera opción (*Hybrid → ATL*), se obtiene un modelo de transformación conforme al metamodelo del lenguaje ATL. En cambio, si se selecciona la segunda (*Hybrid → ETL*), se obtiene un modelo conforme al metamodelo del lenguaje ETL.

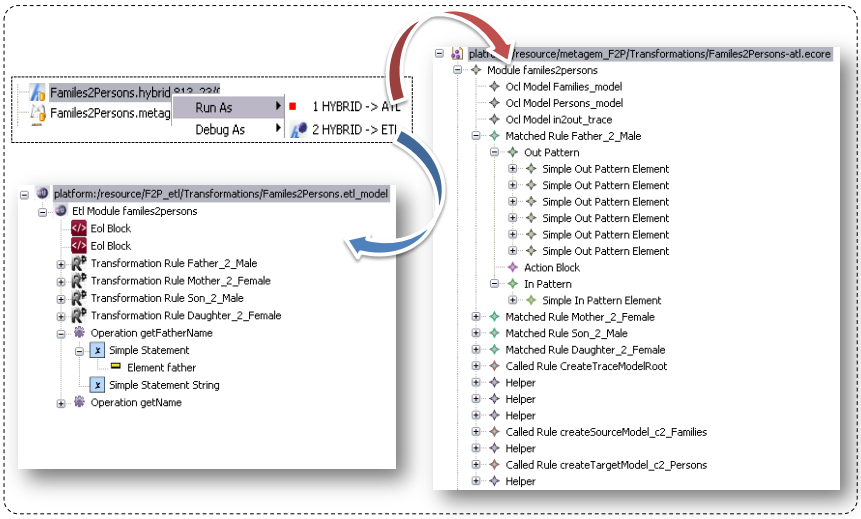

Figura B-11. Transformación de PSM a PDM

En ambos casos, antes de realizar la ejecución, se realiza de forma automática el proceso de validación del modelo definido a nivel PSM, teniendo en cuenta el modelo destino, de la forma que se ha indicado en la sección 4.4.2. Si la validación es correcta se generarán los modelos PDM como muestra la Figura B-11. En caso contrario, el usuario recibirá un mensaje por pantalla que le indicará los errores detectados.

B.6 Generación de Código

Para obtener el código que implementa la transformación a partir del modelo PDM, MeTAGeM-Trace proporciona una entrada en el menú contextual de dichos modelos, ya sean conformes al metamodelo de ATL o al de ETL.

Para obtener el código de la transformación en el lenguaje ATL, se debe hacer clic con el botón secundario sobre el modelo ATL y seleccionar la opción *Extract ATL model to ATL file (MeTAGeM-Trace)*. Como resultado, se obtiene un archivo con extensión *atl* que contiene el código que implementa la transformación en dicho lenguaje (Figura B-12).

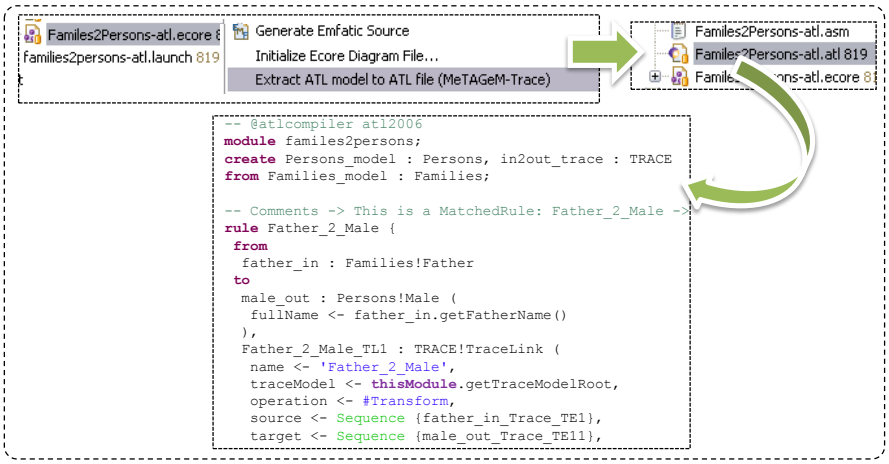

Figura B-12. Generando el código de la transformación en ATL

Del mismo modo, haciendo clic con el botón secundario sobre el modelo ETL y seleccionando la opción *Extract ETL model to ETL file (MeTAGeM-Trace)*, se obtiene el fichero de código que implementa la transformación en ETL (Figura B-13).

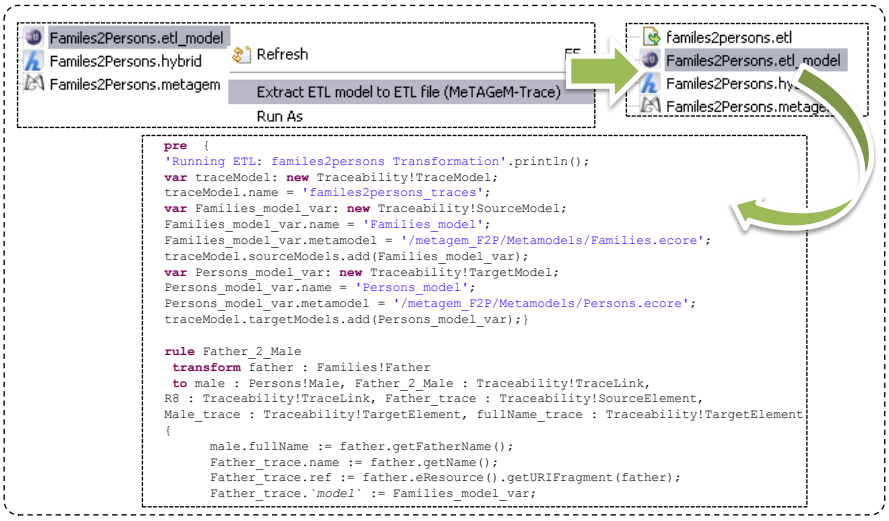

Figura B-13. Generando el código de la transformación en ETL

B.7 Generación del Modelo de Trazas

Siguiendo los pasos anteriores, el usuario habrá obtenido el código de la transformación que contiene de forma embebida un generador de trazas. Así, cuando se ejecute la transformación sobre los modelos origen se generará, además de los modelos destino, un modelo adicional que contendrá las trazas entre los elementos de los modelos anteriores. Este modelo de trazas tendrá una estructura similar al mostrado en la Figura B-14.

Para llevar a cabo la ejecución de las transformaciones, el usuario deberá configurar los lanzadores de ejecución de la forma requerida por el lenguaje en el que está implementada la transformación (ATL o ETL).

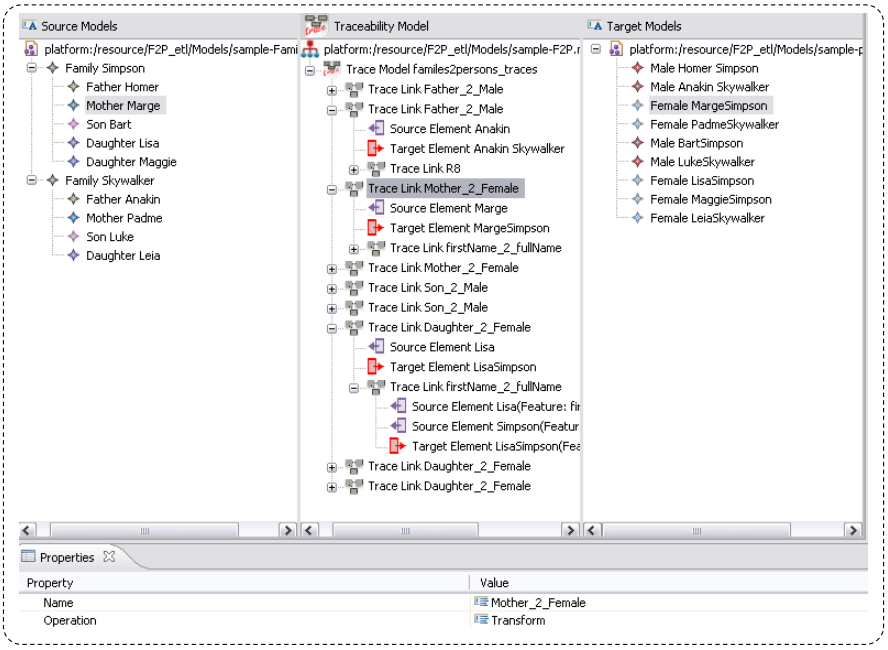

Figura B-14. Ejemplo de Modelo de Trazas

Referencias

1. Abdelhalim, I., Sharp, J., Schneider, S., & Treharne, H. (2010), "Formal Verification of Tokeneer Behaviours Modelled in fUML Using CSP", in *Formal Methods and Software Engineering*. vol. 6447, J. Dong &H. Zhu, Eds., ed: Springer Berlin / Heidelberg, pp. 371-387.
2. Abran, A. & Moore, J. W. (2004), *Guide to the Software Engineering Body of Knowledge*: IEEE Computer Society.
3. Acuña, C., Minoli, M., & Vara, J. M. (2009), "Model Driven Development of Semantic Web Services using Eclipse", in *Mexican International Conference on Computer Science, ENC'09*, Mexico City (Mexico), 2009.
4. Acuña, C. J. (2007), "PISA – Arquitectura de integración de portales Web: un enfoque dirigido por modelos y basado en servicios Web semánticos", PhD thesis, Universidad Rey Juan Carlos (October 2007).
5. Aizenbud-Reshef, N., Nolan, B. T., Rubin, J., & Shaham-Gafni, Y. (2006), "Model traceability", *IBM Systems Journal*, vol. 45 (3), pp. 515-526.
6. Allilaire, F. (2009), "Towards Traceability support in ATL with Obeo Traceability", presented at the 1st International Workshop on Model Transformation with ATL (MtATL2009), Nantes, France, 2009, pp. 150-153.
7. Allilaire, F. (2009), "Towards Traceability support in ATL with Obeo Traceability", *Model Transformation with ATL*, p. 150.
8. Allilaire, F., Bézivin, J., Jouault, F., & Kurtev, I. (2006), "ATL-eclipse support for model transformation", 2006.
9. Anquetil, N., Kulesza, U., Mitschke, R., Moreira, A., Royer, J.-C., Rummler, A., & Sousa, A. (2010), "A model-driven traceability framework for software product lines", *Software and Systems Modeling*, vol. 9 (4), pp. 427-451.
10. Aquino, N., Vanderdonckt, J., & Pastor, O. (2010), "Transformation templates: adding flexibility to model-driven engineering of user interfaces", presented at the Proceedings of the 2010 ACM Symposium on Applied Computing, Sierre, Switzerland, 2010, pp. 1195-1202.
11. Asuncion, H. (2008), "Towards practical software traceability", in *Companion of the 30th international conference on Software engineering*, Leipzig, Germany, 2008, pp. 1023-1026.
12. Asuncion, H. U., Asuncion, A. U., & Taylor, R. N. (2010), "Software traceability with topic modeling", in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, Cape Town, South Africa, 2010, pp. 95-104.
13. Atkins, C. & Louw, G. (2000), "Reclaiming Knowledge: A case for evidence-based information systems", 2000, pp. 39-45.
14. Atkinson, C. & Kuhne, T. (2003), "Model-Driven Development: A Metamodeling Foundation", *IEEE Software*, vol. 20 (5), pp. 36-41.
15. ATL (2006), "ATL: User Manual", version 0.7. Retrieved from http://www.eclipse.org/m2m/atl/doc/ATL_User_Manual[v0.7].pdf.
16. Balasubramanian, D., Narayanan, A., van Buskirk, C., & Karsai, G. (2007), "The graph rewriting and transformation language: GReAT", *Electronic Communications of the EASST*.
17. Barbero, M., Del Fabro, M. D., & Bezivin, J. (2007), "Traceability and provenance issues in global model management", in *Proceedings of International Conference on Systems Engineering and Modelling (ICSEM07)*, 2007.
18. Basili, V. R. (1996), "The role of experimentation in software engineering: past, current, and future", presented at the Proceedings of the 18th international conference on Software engineering, Berlin, Germany, 1996, pp. 442-449.
19. Basili, V. R., Selby, R. W., & Hutchens, D. H. (1986), "Experimentation in software engineering", *IEEE Transactions on Software Engineering*, vol. 12 (7), pp. 733-743.
20. Baxter, P. & Jack, S. (2008), "Qualitative case study methodology: Study design and implementation for novice researchers", *The Qualitative Report*, vol. 13 (4), pp. 544-559.

21. Bernstein, P. (2003), "Applying model management to classical meta data problems", in *First Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, USA, 2003.
22. Bézivin, J. (2004), "In search of a basic principle for model driven engineering", *Novatica Journal, Special Issue*, vol. V (2), pp. 21-24.
23. Bézivin, J. (2005), "On the unification power of models", *Software and Systems Modeling*, vol. 4 (2), pp. 171-188.
24. Bézivin, J., Büttner, F., Gogolla, M., Jouault, F., Kurtev, I., & Lindow, A. (2006), "Model Transformations? Transformation Models!", in *Model Driven Engineering Languages and Systems*. vol. 4199, O. Nierstrasz, J. Whittle, D. Harel , & G. Reggio, Eds., ed: Springer Berlin / Heidelberg, pp. 440-453.
25. Bézivin, J., Farcet, N., Jézéquel, J.-M., Langlois, B., & Pollet, D. (2003), "Reflective Model Driven Engineering", in *«UML» 2003 - The Unified Modeling Language. Modeling Languages and Applications*. vol. 2863, P. Stevens, J. Whittle , & G. Booch, Eds., ed: Springer Berlin / Heidelberg, pp. 175-189.
26. Bezivin, J. & Gerbe, O. (2001), "Towards a precise definition of the OMG/MDA framework", in *Automated Software Engineering, 2001. (ASE 2001). Proceedings. 16th Annual International Conference on*, 2001, pp. 273-280.
27. Biolchini, J., Mian, P. G., Natali, A. C. C., & Travassos, G. H. (2005), "Systematic review in software engineering", *System Engineering and Computer Science Department COPPE/UFRJ, Technical Report ES*, vol. 679 (05).
28. Birrell, N. & Ould, M. A. (1988), *A practical handbook for software development*: Cambridge University Press.
29. Blumberg, R. & Atre, S. (2003), "The problem with unstructured data", *DM REVIEW*, vol. 13 pp. 42-49.
30. Bollati, V. A. (2011), "MeTAGeM: Entorno de Desarrollo de Transformaciones de Modelos Dirigido por Modelos", PhD thesis, Universidad Rey Juan Carlos (February 2011).
31. Bollati, V. A., Sánchez, E. V., Vela, B., & Marcos, E. (2009), "Análisis de QVT Operational Mappings: un caso de estudio", *Actas de los Talleres de las Jornadas de Ing. del Software y BBDD*, vol. 3 (2).
32. Bonde, L., Boulet, P., & Dekeyser, J. L. (2006), "Traceability and interoperability at different levels of abstraction in model-driven engineering", *Applications of specification and design languages for SoCs: selected papers from FDL 2005*, p. 263.
33. Booch, G., Rumbaugh, J., & Jacobson, I. (2000), *El Lenguaje Unificado de Modelado, manual de referencia*: Addison Wesley, Madrid.
34. Boronat, A., Carsí, J., & Ramos, I. (2005), "Automatic Support for Traceability in a Generic Model Management Framework", in *1st European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA'05)*, Nuremberg, Germany, 2005.
35. Bourque, P. & Dupuis, R. (2004), "Guide to the Software Engineering Body of Knowledge 2004 Version", *Guide to the Software Engineering Body of Knowledge, 2004. SWEBOK*.
36. Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (2000) "Extensible markup language (XML) 1.0", ed: W3C
37. Brcina, R. & Riebisch, M. (2008), "Defining a traceability link semantics for design decision support", in *Proceedings of the 4th European Conference on Model Driven Architecture - Traceability Workshop (ECMDA-TW'08)*, Berlin, Germany, 2008, pp. 39-48.
38. Briand, L. C., Labiche, Y., & O'Sullivan, L. (2003), "Impact analysis and change management of UML models", in *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, 2003, pp. 256-265.
39. Budgen, D. & Brereton, P. (2006), "Performing systematic literature reviews in software engineering", presented at the Proceedings of the 28th international conference on Software engineering, Shanghai, China, 2006, pp. 1051-1052.
40. Budinsky, F. (2004), *Eclipse modeling framework: a developer's guide*: Addison-Wesley Professional.
41. Bunge, M. (1983), *La investigación científica*. Barcelona: Ed. Ariel.
42. Campos, P. & Nunes, N. J. (2007), "Practitioner Tools and Workstyles for User-Interface Design", *IEEE Softw.*, vol. 24 (1), pp. 73-80.

43. Clark, T., Sammut, P., & Willans, J. (2008), "Applied metamodelling: a foundation for language driven development".
44. Clayberg, E. & Rubel, D. (2006), *Eclipse: Building Commercial-Quality Plug-ins (Eclipse)*: Addison-Wesley Professional.
45. Czarnecki, K., Foster, J., Hu, Z., Lämmel, R., Schürr, A., & Terwilliger, J. (2009), "Bidirectional Transformations: A Cross-Discipline Perspective", in *Theory and Practice of Model Transformations*. vol. 5563, R. Paige, Ed., ed: Springer Berlin / Heidelberg, pp. 260-283.
46. Czarnecki, K. & Helsen, S. (2003), "Classification of Model Transformation Approaches", presented at the OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, 2003.
47. Dausend, M. & Raiser, F. (2011), "Model Transformation using Constraint Handling Rules as a basis for Model Interpretation", in *Proceedings of the Eighth International Workshop on Constraint Handling Rules*, El Cairo, Egypt, 2011, pp. 64 - 78.
48. de Castro, M. (2007), "Aproximación MDA para el desarrollo orientado a servicios de sistemas de información web: del modelo de negocio al modelo de composición de servicios web", PhD thesis, Universidad Rey Juan Carlos (March 2007).
49. de Castro, V., Marcos, E., & Cáceres, P. (2004), "A user service oriented method to model web information systems", *Web Information Systems–WISE 2004*, pp. 41-52.
50. de Castro, V., Marcos, E., & Lopez Sanz, M. (2006), "A model driven method for service composition modelling: a case study", *International Journal of Web Engineering and Technology*, vol. 2 (4), pp. 335-353.
51. de Castro, V., Mesa, J. M., Herrmann, E., & Marcos, E. (2008), "A Model Driven Approach for the Alignment of Business and Information Systems Models", in *Mexican International Conference on Computer Science, 2008. ENC '08.* , Baja California, Mexico, 2008, pp. 33-43.
52. de Castro, V., Mesa, J. M., Herrmann, E., & Marcos, E. (2009), "From Real Computational Independent Models to Information System Models: An MDE Approach", in *4th International Workshop on Model-Driven Web Engineering (MDWE 2008)*, Toulouse, France., 2009.
53. de Castro, V., Vara, J. M., & Marcos, E. (2007), "Model transformation for service-oriented web applications development", in *3rd International Workshop on Model-Driven on Web Engineering (MDWE 2007)*, Como, Italy, 2007, pp. 284-198.
54. Didonet Del Fabro, M. (2007), "Metadata management using model weaving and model transformation", PhD thesis, Université de Nantes.
55. Dorling, A. (1993), "SPICE: Software Process Improvement and Capability Determination", *Software Quality Journal*, vol. 2 (4), pp. 209-224.
56. Drivalos, N., Kolovos, D., Paige, R., & Fernandes, K. (2009), "Engineering a DSL for Software Traceability Software Language Engineering", in *Software Language Engineering*. vol. 5452, ed: Springer Berlin / Heidelberg, pp. 151-167.
57. Drivalos, N., Kolovos, D., Paige, R., & Fernandes, K. (2010), "A state-based approach to traceability maintenance", presented at the Proceedings of the 6th ECMFA Traceability Workshop, Paris, France, 2010, pp. 23-30.
58. Drivalos, N., Paige, R., Fernandes, K., & Kolovos, D. (2008), "Towards rigorously defined model-to-model traceability", in *Proceedings of the 4th European Conference on Model Driven Architecture - Traceability Workshop (ECMDA-TW'08)*, Berlin, Germany, 2008, pp. 17-26.
59. Eckstein, R., Loy, M., & Wood, D. (1998), *Java swing*: O'Reilly & Associates, Inc.
60. Edwards, G., Seo, C., & Medvidovic, N. (2008), "Model Interpreter Frameworks: A foundation for the analysis of Domain-specific software architectures", *Journal of Universal Computer Science*, vol. 14 (8), pp. 1182-1206.
61. Egyed, A. (2004), "Resolving uncertainties during trace analysis", *SIGSOFT Softw. Eng. Notes*, vol. 29 (6), pp. 3-12.
62. Eisenhardt, K. M. (1989), "Building Theories from Case Study Research", *The Academy of Management Review*, vol. 14 (4), pp. 532-550.
63. Falleri, J. R., Huchard, M., & Nebut, C. (2006), "Towards a traceability framework for model transformations in kermeta", in *Procedings of the 2th European Conference on Model Driven Architecture - Traceability Workshop (ECMDA-TW'06)*, Bilbao, Spain, 2006.

64. Favre, J. M. (2004), "Towards a basic theory to model model driven engineering", presented at the Workshop on Software Model Engineering, WISME 2004, Lisbon, Portugal, 2004.
65. Feuerlicht, G., Pokorný, J., & Richta, K. (2009), "Object-Relational Database Design: Can Your Application Benefit from SQL:2003?", in *Information Systems Development*, C. Barry, M. Lang, W. Wojtkowski, K. Conboy , & G. Wojtkowski, Eds., ed: Springer US, pp. 975-987.
66. Flore, F. (2003), "MDA: The proof is in automating transformations between models", *OptimalJ White Paper*, pp. 1–4.
67. Fowler, M. (2009), "Code Generation for Dummies", in *METHODS & TOOLS*, ed: Spring, pp. 65 - 89.
68. Fowler, M. & Parsons, R. (2010), *Domain-specific languages*: Addison-Wesley Professional.
69. Frankel, D. S. (2002), *Model Driven Architecture: Applying MDA to Enterprise Computing*. New York, USA: John Wiley & Sons.
70. Gallardo, D., Burnette, E., & McGovern, R. (2003), *Eclipse in action: a guide for Java developers*: Manning Publications.
71. Gamma, E., Acebal, C. F., & Lovelle, J. M. C. (2003), *Patrones de diseño: elementos de software orientado a objetos reutilizable*: Addison-Wesley.
72. Gamma, E. & Beck, K. (2004), *Contributing to Eclipse: principles, patterns, and plug-ins*: Addison-Wesley Professional.
73. Gerber, A., Lawley, M., Raymond, K., Steel, J., & Wood, A. (2002), "Transformation: The Missing Link of MDA ", in *Graph Transformation*. vol. 2505, A. Corradini, H. Ehrig, H. Kreowski , & G. Rozenberg, Eds., ed: Springer Berlin / Heidelberg, pp. 90-105.
74. Gervais, M. P. (2002), "Towards an MDA-oriented methodology", in *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*, 2002, pp. 265-270.
75. Göknil, A., Topaloglu, N. Y., & van den Berg, K. G. (2008) "Operation Composition in Model Transformations with Complex Source Patterns", ed. Enschede: Centre for Telematics and Information Technology, University of Twente.
76. Gotel, O. C. Z. & Finkelstein, C. W. (1994), "An analysis of the requirements traceability problem", in *Proceedings of the First International Conference onRequirements Engineering*, Colorado Springs, CO , USA 1994, pp. 94-101.
77. Grammel, B. & Kastenholz, S. (2010), "A generic traceability framework for facet-based traceability data extraction in model-driven software development", presented at the Proceedings of the 6th ECMFA Traceability Workshop, Paris, France, 2010, pp. 7-14.
78. Gray, J. (2010), "Model comparison: the marrow of model transformation", presented at the Proceedings of the 1st International Workshop on Model Comparison in Practice, Malaga, Spain, 2010, pp. 1-1.
79. Greenwald, R., Stackowiak, R., & Stern, J. (2004), *Oracle Essentials, 3e: Oracle Database 10g*: O'Reilly & Associates, Inc.
80. Gronback, R. C. (2009), *Eclipse modeling project: a domain-specific language toolkit*: Addison-Wesley Professional.
81. Guerra, E., de Lara, J., Kolovos, D., & Paige, R. (2010), "Inter-modelling: From Theory to Practice", in *Model Driven Engineering Languages and Systems*. vol. 6394, D. Petriu, N. Rouquette , & ò. Haugen, Eds., ed: Springer Berlin / Heidelberg, pp. 376-391.
82. Guerra, E., de Lara, J., Kolovos, D., Paige, R., & dos Santos, O. (2010), "transML: A Family of Languages to Model Model Transformations", in *Model Driven Engineering Languages and Systems*. vol. 6394, D. Petriu, N. Rouquette , & Ø. Haugen, Eds., ed: Springer Berlin / Heidelberg, pp. 106-120.
83. Guerra, E., de Lara, J., Kolovos, D., Paige, R., & dos Santos, O. (2011), "Engineering model transformations with transML", *Software and Systems Modeling*, pp. 1-23.
84. Guttman, M. & Parodi, J. (2007), *Real-life MDA: solving business problems with model driven architecture*: Morgan Kaufmann.
85. Hailpern, B. & Tarr, P. (2006), "Model-driven development: The good, the bad, and the ugly", *IBM Systems Journal*, vol. 45 (3), pp. 451-461.

86. Hayes, J. H., Dekhtyar, A., & Sundaram, S. K. (2006), "Advancing candidate link generation for requirements tracing: the study of methods", *Software Engineering, IEEE Transactions on*, vol. 32 (1), pp. 4-19.
87. Herbsleb, J., Zubrow, D., Goldenson, D., Hayes, W., & Paulk, M. (1997), "Software quality and the Capability Maturity Model", *Commun. ACM*, vol. 40 (6), pp. 30-40.
88. IBM. (2011). *Rational Rose Product line*. Available in: http://www-01.ibm.com/software/awdtools/developer/rose/index.html
89. IEEE (1990), "IEEE Standard Glossary of Software Engineering Terminology", *IEEE Std 610.12-1990*.
90. ISO (1995), "Information technology – Software life cycle processes", International Organization for Standardization ISO/IEC 12207:1995.
91. ISO&IEC (2003) "ISO/IEC 9075:2003. Information technology – Database languages – SQL:2003", ed: ISO (International Standards Organization for Standardization) & IEC (International Electrotechnical Commission).
92. Jacobson, I., Booch, G., & Rumbaugh, J. (1999), *The Unified Software Development Process– The complete guide to the Unified Process from the original designers*: Addison-Wesley Professional.
93. Jiménez, Á., Granada, D., Bollati, V. A., & Vara, J. M. (2011), "Using ATL to support Model-Driven Development of RubyTL Model Transformations", in *3rd International Workshop on Model Transformation with ATL (MtATL2011)*, Zürich, Switzerland, 2011, pp. 35 - 48.
94. Jiménez, Á., Vara, J. M., Bollati, V., & Marcos, E. (2010), "Mejorando el nivel de automatización en el desarrollo dirigido por modelos de editores gráficos", in *VII Taller sobre Desarrollo de Software Dirigido por Modelos (DSDM 2010), XV Jornadas de Ingeniería del Software y Bases de Datos (JISBD, 2010)*, Valencia, Spain, 2010, pp. 29 - 37
95. Jiménez, Á., Vara, J. M., Bollati, V. A., & Marcos, E. (2011), "Gestión de la trazabilidad en el desarrollo dirigido por modelos de Transformaciones de Modelos: una revisión de la literatura", in *XVI Jornadas de Ingeniería del Software y Bases de Datos (JISBD2011)*, A Coruña, Spain, 2011, pp. 783-796.
96. Jossic, A., Del Fabro, M. D., Lerat, J. P., Bezivin, J., & Jouault, F. (2007), "Model Integration with Model Weaving: a Case Study in System Architecture", in *Systems Engineering and Modeling, 2007. ICSEM '07. International Conference on*, 2007, pp. 79-84.
97. Jouault, F. (2005), "Loosely coupled traceability for ATL", presented at the 1st European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA'05), Nuremberg, Germany, 2005.
98. Jouault, F., Allilaire, F., Bézivin, J., & Kurtev, I. (2008), "ATL: A model transformation tool", *Science of Computer Programming*, vol. 72 (1-2), pp. 31-39.
99. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., & Valduriez, P. (2006), "ATL: a QVT-like transformation language", presented at the Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, Portland, Oregon, USA, 2006, pp. 719-720.
100. Jouault, F., Bézivin, J., & Kurtev, I. (2006), "TCS: a DSL for the specification of textual concrete syntaxes in model engineering", presented at the Proceedings of the 5th international conference on Generative programming and component engineering, Portland, Oregon, USA, 2006, pp. 249-254.
101. Jouault, F. & Kurtev, I. (2006), "Transforming Models with ATL", in *Satellite Events at the MoDELS 2005 Conference*. vol. 3844, ed: Springer Berlin / Heidelberg, pp. 128-138.
102. Kent, S. (2002), "Model Driven Engineering", in *Integrated Formal Methods*. vol. 2335, M. Butler, L. Petre , & K. Sere, Eds., ed: Springer Berlin / Heidelberg, pp. 286-298.
103. Kish, L. (1959), "Some statistical problems in research design", *American Sociological Review*, pp. 328-338.
104. Kitchenham, B. (2004), "Procedures for performing systematic reviews", *Keele, UK, Keele University, Technical Report* vol. 33.
105. Kitchenham, B. & Charters, S. (2007), "Guidelines for performing systematic literature reviews in software engineering", *Engineering*, vol. 2 (EBSE 2007-001).

106. Kleppe, A. G., Warmer, J., & Bast, W. (2003), *MDA explained: the model driven architecture: practice and promise*: Addison-Wesley
107. Knethen, A. v. & Grund, M. (2003), "QuaTrace: A Tool Environment for (Semi-) Automatic Impact Analysis Based on Traces", presented at the Proceedings of the International Conference on Software Maintenance, 2003, p. 246.
108. Koch, N. (2001), "Software engineering for adaptive hypermedia applications", PhD. Thesis, FAST Reihe Softwaretechnik, Uni-Druck Publishing Company, Munich, Germany.
109. Kolovos, D., Paige, R., & Polack, F. (2006), "The Epsilon Object Language (EOL)", in *Model Driven Architecture – Foundations and Applications*. vol. 4066, A. Rensink &J. Warmer, Eds., ed: Springer Berlin / Heidelberg, pp. 128-142.
110. Kolovos, D., Paige, R., & Polack, F. (2006), "Merging Models with the Epsilon Merging Language (EML)", in *Model Driven Engineering Languages and Systems*. vol. 4199, ed: Springer Berlin / Heidelberg, pp. 215-229.
111. Kolovos, D., Paige, R., & Polack, F. (2006), "On-demand merging of traceability links with models", in *Procedings of the 2nd European Conference on Model Driven Architecture - Traceability Workshop (ECMDA-TW'06)*, Bilbao, Spain, 2006.
112. Kolovos, D., Paige, R., & Polack, F. (2006), "On-demand merging of traceability links with models", in *Procedings of the 2th European Conference on Model Driven Architecture - Traceability Workshop (ECMDA-TW'06)*, Bilbao, Spain, 2006.
113. Kolovos, D., Paige, R., & Polack, F. (2008), "The Epsilon Transformation Language", in *Theory and Practice of Model Transformations*. vol. 5063, A. Vallecillo, J. Gray , & A. Pierantonio, Eds., ed: Springer Berlin / Heidelberg, pp. 46-60.
114. Kolovos, D., Rose, L., Paige, R., & Polack, F. (2010), "The Epsilon Book".
115. Kolovos, D. S. (2008), "An Extensible Platform for Specification of Integrated Languages for Model Management", PhD thesis, University of York.
116. Kolovos, D. S., Paige, R. F., & Polack, F. A. C. (2006), "Eclipse development tools for epsilon", in *Eclipse Summit Europe, Eclipse Modeling Symposium.* , Esslingen, Germany., 2006.
117. Kruchten, P. (2004), *The rational unified process: an introduction*: Addison-Wesley Professional.
118. Kulkarni, V. & Reddy, S. (2003), "Separation of concerns in model-driven development", *Software, IEEE*, vol. 20 (5), pp. 64-69.
119. Kulkarni, V., Venkatesh, R., & Reddy, S. (2002), "Generating Enterprise Applications from Models", in *Advances in Object-Oriented Information Systems*. vol. 2426, J.-M. Bruel &Z. Bellahsene, Eds., ed: Springer Berlin / Heidelberg, pp. 309-315.
120. Kurtev, I. (2005), "Adaptability of model transformations", PhD thesis, University of Twente (May 2005), Twente.
121. Kurtev, I., Van den Berg, K., & Jouault, F. (2006), "Evaluation of rule-based modularization in model transformation languages illustrated with ATL", presented at the Proceedings of the 2006 ACM symposium on Applied computing, Dijon, France, 2006, pp. 1202-1209.
122. Kusel, A. (2009), "TROPIC-a framework for building reusable transformation components", in *Proceedings of the Doctoral Symposium at MODELS 2009*, School of Computing, Queen's University, Denver, 2009.
123. Küster, J. & Abd-El-Razik, M. (2007), "Validation of model transformations–first experiences using a white box approach", *MoDELS Conference*, pp. 193-204.
124. Küster, J., Gschwind, T., & Zimmermann, O. (2009), "Incremental development of model transformation chains using automated testing", in *MoDELS Conference*, 2009, pp. 733-747.
125. Küster, J. M., Ryndina, K., & Hauser, R. (2005), "A Systematic Approach to Designing Model Transformations", Report RZ 3621, IBM. Zurich.
126. Lano, K. & Kolahdouz-Rahimi, S. (2011), "Model-Driven Development of Model Transformations", in *Theory and Practice of Model Transformations*. vol. 6707, J. Cabot &E. Visser, Eds., ed: Springer Berlin / Heidelberg, pp. 47-61.
127. Lemesle, R. (1998), "Transformation rules based on meta-modeling", in *Enterprise Distributed Object Computing Workshop, 1998. EDOC '98. Proceedings. Second International*, 1998, pp. 113-122.

128. Levendovszky, T., Balasubramanian, D., Smyth, K., Shi, F., & Karsai, G. (2010), "A transformation instance-based approach to traceability", presented at the Proceedings of the 6th ECMFA Traceability Workshop, Paris, France, 2010, pp. 55-60.
129. Li, S., Liu, S., Wang, X., & Geng, Z. (2010), "A Research and Implementation of Model Execution Method Based on MOF", presented at the Proceedings of the Third International Symposium on Computer Science and Computational Technology(ISCSCT '10), Jiaozuo, P. R. China, 2010, pp. 91 - 93.
130. Lichter, H., Schneider-Hufschmidt, M., Z\, H., \#252, & llighoven (1993), "Prototyping in industrial software projects\—bridging the gap between theory and practice", presented at the Proceedings of the 15th international conference on Software Engineering, Baltimore, Maryland, United States, 1993, pp. 221-229.
131. López-Sanz, M., Acuña, C. J., Cuesta, C. E., & Marcos, E. (2008), "Modelling of Service-Oriented Architectures with UML", *Electronic Notes in Theoretical Computer Science*, vol. 194 (4), pp. 23-37.
132. Mäder, P., Gotel, O., & Philippow, I. (2009), "Enabling Automated Traceability Maintenance through the Upkeep of Traceability Relations", in *Model Driven Architecture - Foundations and Applications*. vol. 5562, R. Paige, A. Hartman , & A. Rensink, Eds., ed: Springer Berlin / Heidelberg, pp. 174-189.
133. Marcos, E. (2005), "Software engineering research versus software development", *SIGSOFT Softw. Eng. Notes*, vol. 30 (4), pp. 1-7.
134. Marcos, E., Acuña, C., & Cuesta, C. (2006), "Integrating Software Architecture into a MDA Framework", in *Software Architecture, Third European Workshop* vol. 4344, ed Heidelberg, Alemania: Springer Verlag, pp. 127-143.
135. Marcos, E. & Marcos, A. (1998), "An Aristotelian Approach to the Methodological Research: a Method for Data Models Construction", *Information Systems-The Next Generation. L. Brooks & C. Kimble (Eds.). Mc Graw-Hill*, pp. 532-543.
136. Margaria, T. & Steffen, B. (2009), "Continuous Model-Driven Engineering", *IEEE Computer*, vol. 42 (10), pp. 106-109.
137. Mazón, J.-N. & Trujillo, J. (2008), "An MDA approach for the development of data warehouses", *Decision Support Systems*, vol. 45 (1), pp. 41-58.
138. Melby, S. (2007), "Traceability in Model Driven Engineering", Master Thesis. University of Oslo, Norway, http://urn.nb.no/URN:NBN:no-18721.
139. Mellor, S. J., Kendall, S., Uhl, A., & Weise, D. (2004), *MDA distilled*: Addison Wesley.
140. Mernik, M., Heering, J., & Sloane, A. M. (2005), "When and how to develop domain-specific languages", *ACM Comput. Surv.*, vol. 37 (4), pp. 316-344.
141. Molina, F., Lucas, F. J., Toval, A., Vara, J. M., Cáceres, P., & Marcos, E. (2007), "Towards quality web information systems through precise model driven development", *Handbook of research on web information systems quality, Idea Group Publishing*, pp. 317-355.
142. Naslavsky, L., Alspaugh, T. A., Richardson, D. J., & Ziv, H. (2005), "Using scenarios to support traceability", presented at the Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering, Long Beach, California, 2005, pp. 25-30.
143. Nuthall, G. (2004), "Relating classroom teaching to student learning: A critical analysis of why research has failed to bridge the theory-practice gap", *Harvard Educational Review*, vol. 74 (3), pp. 273-306.
144. Oldevik, J. (2006), "MOFScript user guide", *Version 0.6 (MOFScript v 1.1. 11)*.
145. Oldevik, J. & Neple, T. (2006), "Traceability in model to text transformations", in *Procedings of the 2th European Conference on Model Driven Architecture - Traceability Workshop (ECMDA-TW'06)*, Bilbao, Spain, 2006, pp. 17-26.
146. Oldevik, J. & Neple, T. (2006), "Traceability in model to text transformations", in *Procedings of the 2nd European Conference on Model Driven Architecture - Traceability Workshop (ECMDA-TW'06)*, Bilbao, Spain, 2006, pp. 17-26.
147. Oldevik, J., Neple, T., Grønmo, R., Aagedal, J., & Berre, A.-J. (2005), "Toward Standardised Model to Text Transformations", in *Model Driven Architecture – Foundations and Applications*. vol. 3748, A. Hartman &D. Kreische, Eds., ed: Springer Berlin / Heidelberg, pp. 239-253.

148. Oliveto, R. (2008), "Traceability Management meets Information Retrieval Methods - Strengths and Limitations", in *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on*, 2008, pp. 302-305.
149. Olsen, G. K. & Oldevik, J. (2007), "Scenarios of traceability in model to text transformations", presented at the Proceedings of the 3rd European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA'07), Haifa, Israel, 2007, pp. 144-156.
150. OMG. *Architecture Driven Modernization (ADM)*. Available in: http://adm.omg.org/
151. OMG (2002), "MOF 2.0 Query/Views/Transformations RFP", OMG document ad/2002-04-10.
152. OMG (2002), "Software Process Engineering Metamodel (SPEM)", *OMG Document Formal/02-11*, vol. 14.
153. OMG (2003), "MDA Guide, Version 1.0.1", Object Management Group.
154. OMG (2006), "The Meta Object Facility (MOF) Core Specification", version 2.0, OMG Document - formal/06-01-01.
155. OMG (2006), "Object Constraint Language Specification (OCL) ", version 2.0. OMG Document - formal/2006-05-01.
156. OMG (2007), "UML 2.1.1 Formal Specification", OMG Document - formal/07-02-03.
157. OMG (2007), "XML Metadata Interchange (XMI) specification ", version 2.1.1. OMG Document - formal/2007-12-01.
158. OMG (2008), "MOF Model to Text Transformation Language", version 1.0, OMG Document - formal/2008-01-16.
159. Paige, R., Drivalos, N., Kolovos, D., Fernandes, K., Power, C., Olsen, G., & Zschaler, S. (2011), "Rigorous identification and encoding of trace-links in model-driven engineering", *Software and Systems Modeling*, vol. 10 (4), pp. 469-487.
160. Paige, R., Olsen, G. K., Kolovos, D., Zschaler, S., & Power, C. (2008), "Building model-driven engineering traceability classifications", in *Proceedings of the 4th European Conference on Model Driven Architecture - Traceability Workshop (ECMDA-TW'08)*, Berlin, Germany, 2008, pp. 49-58.
161. Parnas, D. L. (1972), "On the criteria to be used in decomposing systems into modules", *Commun. ACM*, vol. 15 (12), pp. 1053-1058.
162. Petticrew, M. & Roberts, H. (2005), *Systematic reviews in the social sciences: A practical guide*: Blackwell Publishing.
163. Pfleeger, S. L. & Atlee, J. M. (1998), *Software engineering: theory and practice*.
164. Pfleeger, S. L. & Kitchenham, B. A. (2001), "Principles of survey research: part 1: turning lemons into lemonade", *SIGSOFT Softw. Eng. Notes*, vol. 26 (6), pp. 16-18.
165. Pilato, C. M., Collins-Sussman, B., & Fitzpatrick, B. W. (2008), *Version control with subversion*: O'Reilly Media.
166. Pino, F., García, F., & Piattini, M. (2008), "Software process improvement in small and medium software enterprises: a systematic review", *Software Quality Journal*, vol. 16 (2), pp. 237-261.
167. Poole, J. D. (2001), "Model-driven architecture: Vision, standards and emerging technologies", in *Workshop on Metamodeling and Adaptive Object Models, ECOOP*, 2001.
168. Pressman, R. S. (2002), *Ingeniería del Software, un enfoque práctico*. Madrid: McGraw-Hill, Interamericana de España.
169. Ramesh, B., Stubbs, C., Powers, T., & Edwards, M. (1997), "Requirements traceability: Theory and practice", *Annals of Software Engineering*, vol. 3 pp. 397-415.
170. Rossini, A., Mughal, K. A., Wolter, U., Rutle, A., & Lamo, Y. "A Formal Approach to Data Validation Constraints in MDE".
171. Rothenberger, M. A., Kao, Y.-C., & Van Wassenhove, L. N. (2010), "Total quality in software development: An empirical study of quality drivers and benefits in Indian software projects", *Information & Management*, vol. 47 (7–8), pp. 372-379.
172. Royce, W. W. (1987), "Managing the development of large software systems: concepts and techniques", presented at the Proceedings of the 9th international conference on Software Engineering, Monterey, California, United States, 1987, pp. 328-338.
173. Rugaber, S. & Stirewalt, K. (2004), "Model-driven reverse engineering", *Software, IEEE*, vol. 21 (4), pp. 45-53.

174. Sánchez Cuadrado, J., García Molina, J., & Menarguez Tortosa, M. (2006), "RubyTL: A Practical, Extensible Transformation Language", in *Model Driven Architecture – Foundations and Applications*. vol. 4066, A. Rensink &J. Warmer, Eds., ed: Springer Berlin / Heidelberg, pp. 158-172.
175. Sánchez, P., Alonso, D., Rosique, F., Álvarez, B., & Pastor, A. P. (2011), "Introducing Safety Requirements Traceability Support in Model-Driven Development of Robotic Applications", *IEEE Transactions on Computers*, vol. 60 pp. 1059-1071.
176. Schmidt, D. C. (2006), "Model-driven engineering", *IEEE Computer*, vol. 39 (2), pp. 25-31.
177. Schwarz, H. (2009), "Taxonomy and definition of the explicit traceability information suppliable for guiding model-driven, ontology-supported development. Project Deliverable ICT216691/UoKL", WP4-D1/D/PU/b1, MOST Project.
178. Seidewitz, E. (2003), "What models mean", *Software, IEEE*, vol. 20 (5), pp. 26-32.
179. Selic, B. (2003), "The pragmatics of model-driven development", *Software, IEEE*, vol. 20 (5), pp. 19-25.
180. Selic, B. (2008), "MDA manifestations", *The European Journal for the Informatics Professional, IX (2)*, pp. 12-16.
181. Selic, B. (2008), "Personal reflections on automation, programming culture, and model-based software engineering", *Automated Software Engineering*, vol. 15 (3), pp. 379-391.
182. Sendall, S. & Kozaczynski, W. (2003), "Model transformation: the heart and soul of model-driven software development", *Software, IEEE*, vol. 20 (5), pp. 42-45.
183. Sjoeberg, D. I. K., Hannay, J. E., Hansen, O., Kampenes, V. B., Karahasanovic, A., Liborg, N. K., & Rekdal, A. C. (2005), "A survey of controlled experiments in software engineering", *Software Engineering, IEEE Transactions on*, vol. 31 (9), pp. 733-753.
184. Sommerville, I. (2005), *Ingeniería del Software*. Madrid: Pearson Educación.
185. Spanoudakis, G. & Zisman, A. (2005), "Software traceability: a roadmap", *Handbook of Software Engineering and Knowledge Engineering*, vol. 3 pp. 395-428.
186. Stahl, T., Völter, M., & Czarnecki, K. (2006), *Model-Driven Software Development: Technology, Engineering, Management*: John Wiley & Sons.
187. Stake, R. E. (1995), *The art of case study research*: Sage Publications, Inc.
188. Steinberg, D., Budinsky, F., Paternostro, M., & Merks, E. (2008), *EMF: Eclipse Modeling Framework*, 2nd Edition ed.: Addison-Wesley Professional.
189. Stevens, P. (2008), "A Landscape of Bidirectional Model Transformations", in *Generative and Transformational Techniques in Software Engineering II*. vol. 5235, R. Lämmel, J. Visser , & J. Saraiva, Eds., ed: Springer Berlin / Heidelberg, pp. 408-424.
190. Tisi, M., Jouault, F., Fraternali, P., Ceri, S., & Bézivin, J. (2009), "On the Use of Higher-Order Model Transformations", in *Model Driven Architecture - Foundations and Applications*. vol. 5562, R. Paige, A. Hartman , & A. Rensink, Eds., ed: Springer Berlin / Heidelberg, pp. 18-33.
191. Toulmé, A. & Inc, I. (2006), "Presentation of EMF compare utility", 2006, p. 2009.
192. Tratt, L. (2005), "Model transformations and tool integration", *Software and Systems Modeling*, vol. 4 (2), pp. 112-122.
193. Valderas, P. & Pelechano, V. (2009), "Introducing requirements traceability support in model-driven development of web applications", *Information and Software Technology*, vol. 51 (4), pp. 749-768.
194. Vallecillo, A. (2008), "A Journey through the Secret Life of Models", in *Perspectives Workshop: Model Engineering of Complex Systems (MECS)* Dagstuhl, Germany, 2008.
195. van Amstel, M., Lange, C., & van den Brand, M. (2009), "Using Metrics for Assessing the Quality of ASF+SDF Model Transformations", in *Theory and Practice of Model Transformations*. vol. 5563, R. Paige, Ed., ed: Springer Berlin / Heidelberg, pp. 239-248.
196. van Amstel, M. & van den Brand, M. (2010), "Quality assessment of ATL model transformations using metrics", in *2nd International Workshop on Model Transformation with ATL (MtATL2010)*, Malaga, Spain, 2010.
197. van Amstel, M. & van den Brand, M. (2011), "Using Metrics for Assessing the Quality of ATL Model Transformations", in *3rd International Workshop on Model Transformation with ATL (MtATL2011)*, Zürich, Switzerland, 2011, pp. 20-34.

198. van Amstel, M., van den Brand, M., & Nguyen, P. H. (2010), "Metrics for model transformations", in *9th Belgian-Netherlands Software Evolution Workshop (BENEVOL 2010)*, Lille, France, 2010.
199. Vara, J. M. (2009), "M2DAT: a Technical Solution for Model-Driven Development of Web Information Systems", PhD thesis, Universidad Rey Juan Carlos (September 2009).
200. Vara, J. M., Bollati, V. A., Vela, B., & Marcos, E. (2008), "Uso de Modelos de Anotación para automatizar el Desarrollo Dirigido por Modelos de Bases de Datos Objeto-Relacionales", *Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos*, vol. 2 (3).
201. Vara, J. M., Bollati, V. A., Vela, B., & Marcos, E. (2009), "Leveraging Model Transformations by means of Annotation Models", in *Proceedings of 1st International Workshop on Model Transformation with ATL (MtATL2009)*, Nantes, France, 2009, pp. 88 -102.
202. Vara, J. M., de Castro, V., & Marcos, E. (2005), "WSDL automatic generation from UML models in a MDA framework", in *Next Generation Web Services Practices, 2005. NWeSP 2005. International Conference on*, 2005.
203. Vara, J. M., Vela, B., Bollati, V. A., & Marcos, E. (2009), "Supporting Model–Driven Development of Object–Relational Database Schemas: A Case Study", in *Theory and Practice of Model Transformations*. vol. 5563, ed: Springer Berlin / Heidelberg, pp. 181-196.
204. Vara, J. M., Vela, B., Cavero, J. M., & Marcos, E. (2007), "Model transformation for object-relational database development", 2007, pp. 1012-1019.
205. Vara, J. M., Vela, B., & Marcos, E. (2006), "Oracle XML DB como repositorio integrado para herramientas CASE. Aplicación al desarrollo de MIDAS-CASE, una herramienta MDA", presented at the XVI Congreso Nacional Usuarios de Oracle (CUORE), 2006.
206. Varró, D. & Pataricza, A. (2003), "VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML (The Mathematics of Metamodeling is Metamodeling Mathematics)", *Software and Systems Modeling*, vol. 2 (3), pp. 187-210.
207. Varró, D. & Pataricza, A. (2004), "Generic and Meta-transformations for Model Transformation Engineering", in *UML2004 - The Unified Modeling Language. Modelling Languages and Applications*. vol. 3273, T. Baar, A. Strohmeier, A. Moreira , & S. Mellor, Eds., ed: Springer Berlin / Heidelberg, pp. 290-304.
208. Vela, B., Acuña, C., & Marcos, E. (2004), "A Model Driven Approach for XML Database Development", in *Conceptual Modeling – ER 2004*. vol. 3288, P. Atzeni, W. Chu, H. Lu, S. Zhou , & T.-W. Ling, Eds., ed: Springer Berlin / Heidelberg, pp. 780-794.
209. Vela, B., Fernández-Medina, E., Marcos, E., & Piattini, M. (2006), "Model driven development of secure XML databases", *ACM Sigmod Record*, vol. 35 (3), pp. 22-27.
210. Vela, B. & Marcos, E. (2003), "Extending UML to represent XML Schemas", presented at the 15th Conference On Advanced Information Systems Engineering. CAISE'03 FORUM, Klagenfurt/Velden (Austria), 2003.
211. Vignaga, A. (2007), "A Methodological Approach to Developing Model Transformations", *Model-Driven Engineering Languages and Systems (MoDELS 2007). Nashville (TN), Estados Unidos*.
212. Vignaga, A., Perovich, D., & Bastarrica, M. C. (2007), "Towards Layered Specifications of Model Transformation", Technical Report.
213. Völter, M. (2009), "Best Practices for DSLs and Model-Driven Development", *Journal of Object Technology*, vol. 8 (6), pp. 79 - 102, September-October 2009.
214. Völter, M. (2011), "From Programming to Modeling-and Back Again", *Software, IEEE*, vol. 28 (6), pp. 20-25.
215. Völter, M. (2011), "MD*/DSL Best Practices (Version 2.0)", Retrieved November, 2011, from: http://voelter.de/data/pub/DSLBestPractices-2011Update.pdf.
216. W3C (1998), "Extensible markup language (XML) 1.0", *W3C XML, February*.
217. Walderhaug, S., Johansen, U., Stav, E., & Aagedal, J. (2006), "Towards a generic solution for traceability in MDD", presented at the 2th European Conference on Model Driven Architecture - Traceability Workshop (ECMDA-TW'06), Bilbao, Spain, 2006.
218. Walderhaug, S., Stav, E., Johansen, U., & Olsen, G. K. (2009), "Traceability in Model-Driven Software Development", in *Designing Software Intensive Systems: Methods and Principles*, P. Tiako, Ed., ed, pp. 133-159.

219. Watson, A. (2008), "A brief history of MDA", *Upgrade, the European Journal for the Informatics Professional*, vol. 9 (2), pp. 7-11.
220. Wimmer, M., Kusel, A., Schoenboeck, J., Kappel, G., Retschitzegger, W., & Schwinger, W. (2009), "Reviving QVT Relations: Model-Based Debugging Using Colored Petri Nets", in *Model Driven Engineering Languages and Systems*. vol. 5795, A. Schürr &B. Selic, Eds., ed: Springer Berlin / Heidelberg, pp. 727-732.
221. Winkler, S. & von Pilgrim, J. (2010), "A survey of traceability in requirements engineering and model-driven development", *Software and Systems Modeling*, vol. 9 (4), pp. 529-565.
222. Wu, G., Liu, W., Zheng, Q., & Zhang, Z. (2009), "Model Interpretation Development: Analysis Design of Automatic Control System", *Interpretation of Information Processing Regulations*.
223. Yin, R. K. (2003), *Case Study Research: Design and Methods* vol. 5: Sage Publications, Thousand Oaks, USA.
224. Yu, E. S. K. & Mylopoulos, J. (1994), "Understanding "why" in software process modelling, analysis, and design", presented at the Proceedings of the 16th international conference on Software engineering, Sorrento, Italy, 1994, pp. 159-168.

Tabla de Acrónimos

Acrónimo	Descripción
AMW	*ATLAS Model Weaver*
ATL	*ATLAS Transformation Language*
BD	Bases de Datos
CCE	Cuestión del Caso de Estudio
CE	Criterio de Evaluación
CI	Cuestión de Investigación
CIM	*Computation Independent Models*
CMCE	Cuestión del Meta-Caso de Estudio
DDM	Desarrollo Dirigido por Modelos
DSDM	Desarrollo de Software Dirigido por Modelos
DSL	*Domain Specific Language*
EMF	*Eclipse Modeling Framework*
EMOF	*Essential MOF*
EMP	*Eclipse Modeling Project*
Epsilon	*Extensible Platform for Specification of Integrated Languages for mOdel Management*
ETL	*Epsilon Transformation Language*
EVL	*Epsilon Validation Language*
GMT	*Generative Modeling Technologies*
HOT	*Higher Order Transformation*
IDM	Ingeniería Dirigida por Modelos
M2DAT	MIDAS MDA Tool

Acrónimo	**Descripción**
M2M	*Model-to-Model*
M2T	*Model-to-Text*
MDA	*Model-Driven Architecture*
MDD	*Model-Driven Development*
MDE	*Model-Driven Engineering*
MDSD	*Model-Driven Software Development*
MeTAGeM	*A Meta-Tool for the Automatic Generation of transformation Models*
MeTAGeM-Trace	*A Meta-Tool for the Automatic Generation of transformation Models and Trace models*
MOF	*Meta-Object Facility*
OCL	*Object Constraint Language*
OMDB	*Online Movies DataBase*
OMG	*Object Management Group*
PDM	*Platform Dependent Models*
PIM	*Platform Independent Models*
PSM	*Platform Specific Models*
QVT	*Query/View/Transformation*
RS	Revisión Sistemática
RUP	*Rational Unified Process*
SI	Sistema de Información
UML	*Unified Modelling Language*
XML	*eXtensible Markup Language*