Tesis

Mariano Gabriel Gili

25 de marzo de 2013

Índice general

Índice de figuras

Agradecimientos

Agradezco a todos... holaaaaaaaa.
puede ser que hola

Introducción

En la Ingeniería de Software Dirigida por Modelos o Model-Driven Engineering (MDE), se define al modelo como principal artefacto que toma participación a lo largo de todas las tareas/procesos de la ingeniería de software – análisis, diseño, desarrollo, pruebas, mantenimiento, etc. Una implementación particular propuesta que sigue esta idea es la Arquitectura Dirigida por Modelos o Model-Driven Architecture (MDA), definida por el Object Management Group (OMG), cuyo ciclo de proceso de desarrollo está basado enteramente en el uso de modelos formales y transformaciones que se realizan sobre dichos modelos. Una característica muy importante de todo proceso de Desarrollo Dirigido por Modelos o Model-Driven Development (MDD), es lo que se conoce como "posibilidad de rastreo" (o de ahora en más en inglés traceability), que ayuda y toma parte en todo lo que respecta a las relaciones que existen entre cada uno de los artefactos productos del proceso de desarrollo.

Cuando nos referimos al término artefacto, hablamos por ejemplo de un requerimiento de sistema, un componente de software, un caso de prueba, entre otros. El mantenimiento y la definición de las relaciones y dependencias que existen entre los artefactos no es una tarea fácil, el mismo ha sido un desafío desde principios de 1970.

En el presente documento de tesis se abordará el tema de traceability, luego se presentará un análisis de los distintos problemas que aún se encuentran abiertos como así también un conjunto de soluciones encontradas a lo largo de la investigación. Finalmente, se elaborará un esquema de traceability con el fin de contribuir en la solución de alguno de los problemas nombrados.

Objetivos

El objetivo del trabajo de tesis propuesto comprende, por un lado, una introducción al tema de traceability conjunto con un análisis de los problemas que aún se encuentran abiertos en su ámbito en MDE, seguido de la elaboración de un esquema de traceability que contribuya en la solución de dichos problemas.

También se diseñará e implementará una herramienta que pueda ser integrada a otra de desarrollo MDE, y asista al desarrollador automatizando el proceso de definición de trazas o links entre elementos de los modelos origen y destino. Esta solución proveerá un mapa de transformaciones que permitirá determinar la procedencia de cada ítem del modelo destino, y su correspondiente origen en el modelo fuente.

Capítulo 1

Traceability

1.1. Introducción

Según el Glosario Estándar de Términos de la Ingeniería de Software del Institute of Electrical and Electronics Engineers (IEEE) [1] la noción de traceability se define como: El grado o nivel en el cual una relación puede ser establecida entre dos o más productos del proceso de desarrollo, especialmente entre productos que tengan una relación de predecesor-sucesor o principal-secundario; por ejemplo el grado en el cual el requerimiento y el diseño de un componente de software se corresponden. La definición anterior fue dada por "the requirements management community", para nosotros, que necesitamos un punto de vista maś cercano al contexto de MDD, traceability es un término usado para describir cualquiera de las complejas relaciones lógicas que existen entre los distintos artefactos que se presentan en cualquier momento del ciclo de vida del desarrollo de software, el establecimiento de estas relaciones y/o el mantenimiento de las mismas.

Entre varios beneficios de traceability que enumeraremos mejor más adelante, podemos encontrar que ayuda a identificar las relaciones y dependencias que existen entre los artefactos de software. También traceability es crucial entre los requerimientos y su representación en los modelos para asegurar que el conjunto relevante de requerimientos fueron debidamente implementados en el código. Pero no solo traceability asegura la identificación de objetos y elementos relacionados, también puede facilitar el análisis de impactos de cambios durante el desarrollo de software.

Las relaciones de traceability pueden ser definidas de forma automática, por ejemplo producto de una transformación de modelos, o de forma manual como el caso de una relación de implementación entre un requerimiento y un componente de software.

En la ingeniería de software encontramos dos usos o semánticas principales que dependen del contexto de traceability:

- Traceability en la ingeniería de requerimientos: donde se guarda un requerimiento desde su definición hasta su implementación. En más detalle según [6] se refiere a la habilidad de describir y seguir la vida de los requerimientos en ambas direcciones, hacia delante y hacia atrás (forward and backward traceability). Desde los orígenes, pasando por el desarrollo y la especificación, hacia su posterior entrega y uso, y a través de todos los períodos de refinamiento e iteración de cualquiera de estas etapas.

- Traceability en el Desarrollo Dirigido por Modelos: donde se almacenan principalmente las relaciones existentes entre los artefactos producto de las transformaciones de modelos.

1.1.1. Beneficios

A continuación se listan un conjunto de actividades de diferentes dominios de la ingeniería en las cuales el uso de traceability es muy beneficioso según [2] y [3]:

- En el Análisis de sistemas: nos ayuda a entender la complejidad de un sistema navegando a lo largo del modelo de enlaces obtenidos por la ejecución de las distintas cadenas de transformación.

- En el Análisis de cobertura: por ejemplo en el momento de ejecución de los casos de prueba, el uso de traceability es crucial para la hora de determinar si todos los requerimientos fueron cubiertos o tenidos en cuenta.

- En el Análisis de impacto de cambios: traceability nos ayuda a ver cómo los cambios en un modelo repercutirán en los otros modelos relacionados; también el uso de traceability nos permite saber en cualquier momento el tipo de dependencia que existe entre las entidades relacionadas, lo cual ayuda a determinar la necesidad de un cambio.

- En el Análisis de huérfanos: nos permitirá encontrar fácilmente los elementos huérfanos de un modelo dado que serán los artefactos que no se encuentren enlazados a ninguna traza.

- En la Comprensión del software y la ingeniería inversa: crucial cuando sea necesario identificar todas las entidades relacionadas a una en particular, entender el tipo de relación existente, identificar las abstracciones, es decir los patrones de diseño, estilos de arquitectura, principios.
- En el análisis de requerimiento: por ejemplo para identificar el artefacto particular que demanda una propiedad específica; encontrar y resolver un conjunto de requerimientos que se contradicen.
- Apoyo en la toma de decisiones: para justificar una decisión dado que nos facilita entender qué factores y metas influyen en la misma; también traceability nos será muy útil en los momentos que se nos presenten distintas propuestas de solución en su análisis y evaluación.
- En la Configuración del sistema y versionado: en este momento el uso de traceability es beneficioso para identificar las restricciones entre los componentes, identificar los cambios necesarios para resolver una restricción, identificar las diferencias entre dos versiones distintas del mismo artefacto y su impacto en otros artefactos.

1.1.2. Algunos inconvenientes

Aunque las ventajas de traceability hoy día ya han sido identificadas, su puesta en práctica apenas ha quedado establecida. Las principales razones de lo anterior según [3] puede ser por lo siguiente:

- El alto costo de la creación y mantenimiento manual de la información de traceability.
- La falta de heurísticas que determinen qué información de los enlaces deben ser grabados.
- Discrepancias entre los distintos roles de usuarios de traceability, por ejemplo entre quienes crean los enlaces y quienes los usan.
- Carencia de soporte adecuado en las herramientas.
- Los artefactos son escritos en diferentes lenguajes, por ejemplo los requerimientos se escriben en lenguaje natural mientras que los programas en algún lenguaje de programación.
- Los artefactos describen el sistema de software en diferentes niveles de abstracción, los artefactos usados durante el diseño difiere de los usados en su implementación.

Más adelante en otro capítulo se presentará con más detalle los desafíos presentes en la implementación y el uso de traceability.

Capítulo 2

Problemas y Desafíos

En este capítulo vamos a describir los principales problemas y grandes desafíos que se encuentran abiertos en el ámbito de traceability a lo largo de distintos aspectos definidos por [5].

El capítulo se encuentra organizado principalmente por temas. Luego, para cada tema, se describen sus problemas y a continuación se listan los desafíos que resultan de los mismos:

2.1. Sobre el conocimiento de traceability

Problemas

- Existe poco consenso respecto a cuáles son las mejores técnicas y métodos para la aplicación de traceability, pocas anotaciones y documentación sobre las mejores prácticas, sumado a una falta de recursos que provean una buena base de conocimiento.

- Las definiciones semánticas no coinciden y las terminologías son dispares o distintas, todo ésto crea barreras de comunicación.

Desafíos

+ Crear una base de conocimiento en la que se vuelquen las mejores prácticas de traceability, una terminología estándar y mucha información adicional, como por ejemplo casos de estudio.

2.2. Sobre la capacitación y la certificación

Problemas

- Muy poca gente es competente en la definición de trazas y, por otro lado, existen disponibles pocos programas educativos.

- Existen pocos programas de certificación, y de ellos, pocos incluyen componentes de traceability.

- No hay definido un conjunto estándar de estrategias de traceability.

Desafíos

+ Identificar las áreas de conocimientos centrales y las habilidades y/o estrategias asociadas a traceability.

+ Desarrollar buenos componentes educativos para la puesta en práctica de traceability.

+ Desarrollar materiales pedagógicos efectivos para educar con énfasis en la importancia y administración de los costos-beneficios del uso de traceability.

2.3. Sobre el soporte a la evolución

Problemas

- La información precisa, coherente, completa y actualizada sobre traceability es fundamental para diversos ámbitos y aplicaciones. Sin embargo, las técnicas actuales de "recuperación de trazas" aún son realizadas de forma manual y por lo tanto son propensas a errores.

- Para que los enlaces de traceability sean útiles, éstos deben reflejar la dependencia actual entre los artefactos. Dado que el costo y esfuerzo para mantenerlos durante la evolución del sistema es inmenso, a menudo los enlaces pasan a encontrarse en un estado erróneo o incorrecto.

- Las herramientas actuales de administración de requerimientos incluyen características como "suspect trace links" para ayudar a los analistas a administrar la evolución de los enlaces, pero en la mayoría de los proyectos complejos el número de "suspect trace links" se vuelve rápidamente excesivo, minimizando drásticamente la utilidad de tal característica.

- Los enlaces de traceability tienen que evolucionar de forma sincrónica con los artefactos relacionados, sin embargo los sistemas actuales de gestión de cambios y la semántica de los enlaces no son lo suficientemente sofisticados como para apoyar esta evolución.

- Los métodos para transformar y reusar los enlaces sincrónicamente con los productos de desarrollo en línea son inmaduros.

Desafíos

+ Desarrollar técnicas de "recuperación de trazas" para artefactos textuales que sean tan precisos como el proceso manual y a la vez mucho más efectivos en tiempo y costo.

+ Desarrollar la recuperación de trazas para que se encuentren integrados a los Entornos de Desarrollo Integrados o Integrated Development Environment (IDE).

+ Desarrollar sistemas de administración de cambios que efectivamente soporten la evolución de los enlaces de trazas sobre múltiples tipos de artefactos.

+ Desarrollar técnicas que soporten traceability en todos los componentes de una línea de productos maximizando la reutilización y la disponibilidad de traceability entre diferentes versiones de los mismos.

+ Desarrollar técnicas para maximizar la reutilización de los enlaces de trazas cuando el código existente se reutiliza en un nuevo producto.

2.4. Sobre la semántica de los enlaces

Problemas

- Para efectivamente utilizar los enlaces de trazas y entender las relaciones por debajo de traceability, es necesario definir la semántica de los enlaces, sin embargo definir una formalidad para representar esta semántica no es una tarea fácil y puede llegar a ser acotada a un dominio específico, cosa que no es conveniente.

- Es muy importante para la consistencia de traceability conocer y establecer la granularidad de los elementos a ser enlazados, pero no existe un modelo claro de costo-beneficio para determinar consistentemente cuál es la granularidad correcta ("trace granularity").

Desafíos

+ Definir meta-modelos para representar la información semántica de los enlaces de trazas y proveer ejemplos de instanciación para distintos dominios específicos.

+ Desarrollar técnicas y procesos para determinar la correcta granularidad de los enlaces de trazas en un proyecto.

2.5. Sobre la escalabilidad

Problemas

- Las técnicas corrientes de traceability no escalan adecuadamente en proyectos largos.

- Las herramientas de visualización son esenciales para dar ayuda en la compresión y el uso de la gran cantidad de información de los enlaces de las trazas. Sin embargo, las técnicas de visualización actuales no escalan bien y no son efectivas al presentar información compleja porque carecen de características sofisticadas de filtrado, navegación, consultas, etc.

- Muchos conjuntos de datos industriales son compuestos por largos e inestructurados documentos que son difíciles de enlazar mediante trazas.

Desafíos

+ Obtener conjuntos de datos de escala industrial desde varios dominios y usarlos para investigar la escalabilidad de las técnicas disponibles actualmente y, si es necesario, crear nuevas aproximaciones que escalen más eficientemente.

+ Desarrollar mecanismos visuales efectivos para soportar la navegación y consulta de un gran número de enlaces de traceability y sus artefactos asociados.

+ Desarrollar técnicas escalables para marcar las trazas tanto para conjunto de datos heterogéneos y/o grandes y débilmente estructurados.

2.6. Sobre los factores humanos

Problemas

- Los métodos automáticos de traceability producen enlaces de trazas candidatos; sin embargo, el proceso es inútil si el analista no es capaz de evaluarlos correctamente para diferenciar los buenos de los malos, o si es incapaz de confiar en la completitud y precisión de los resultados.

- Idealmente la tarea de encontrar las trazas debería ser invisible durante el proceso de desarrollo, desafortunadamente la generación de trazas y el uso es interrumpido por interacciones humanas porque en los ambientes de desarrollo actuales aún no es posible automatizar todo el proceso.

- Los enlaces de trazas comunican artefactos semánticamente diferentes, a su vez estos artefactos son creados por diferentes personas y frecuentemente escritos en diferentes documentos. Como resultado, los usuarios de un lado de los enlaces de trazas no entienden bien los artefactos del otro lado de la relación.

Desafíos

+ Basado en el estudio del uso de herramientas de traceability, crear nuevas herramientas que reúnan las necesidades prácticas que vayan surgiendo.

+ Entender el impacto y las vulnerabilidades de la fallas humanas sobre el proceso de traceability y desarrollar técnicas para ayudar a los analistas a prevenir errores y minimizar el impacto de los mismos cuando ocurran.

+ Desarrollar técnicas para ayudar a los humanos a superar las barreras semánticas del proceso de desarrollo completo.

2.7. Análisis de costo-beneficio

Problemas

- En un escenario de traceability completo, los enlaces son creados entre artefactos en un nivel bajo de abstracción, ésto puede ser deseable para propósitos de comprensión, sin embargo este nivel tan bajo no es frecuentemente práctico y efectivo en costo.

- Existe una carencia de un modelo de costo-beneficio para analizar entre las distintas necesidades de traceability sobre varios proyectos y para diferentes enlaces potenciales dentro de un proyecto.

Desafíos

+ Definir y desarrollar técnicas efectivas en costo para generar y mantener información de traceability.

+ Definir un modelo de costo práctico para generar y mantener los enlaces de trazas que tomen en consideración factores tales como el tamaño del proyecto, el tiempo, el esfuerzo y la calidad del sistema.

+ Definir un modelo de beneficios para usar enlaces de trazas que tomen en consideración factores como la crítica y la volatilidad, e incorpore el valor logrado mediante el uso de traceability.

2.8. Sobre los métodos y las herramientas

Problemas

- Los métodos de recuperación multimedia no son suficientemente sofisticados y soportados, y se ha realizado poco para incorporar tales técnicas multimedia en las herramientas de traceability.

- Traceability automático es esencial; sin embargo, se hace difícil por la falta de consistencia entre los artefactos, y la imprecisión de los modelos.

- Traceability implica todas las siguientes actividades: construcción o generación, evaluación, mantenimiento y el uso de los enlaces; sin embargo, no existe una sola herramienta que pueda cubrir todas estas tareas.

Desafíos

+ Desarrollar métodos efectivos para enlazar artefactos multimedia.

+ Construir métodos y herramientas con altos niveles de automatización para soportar el ciclo de vida entero, que incluya la construcción, la evaluación, el mantenimiento y el uso de los enlaces de trazas.

+ Desarrollar métodos para trazar requerimientos no funcionales.

2.9. Sobre los procesos

Problemas

- Traceability no es incluida frecuentemente como una parte integral del ciclo de vida del desarrollo.

- Traceability automático puede proveer una alternativa efectiva en costo en comparación a la manual, pero la práctica ha mostrado que algunos conjuntos de datos son difíciles de enlazar usando métodos automáticos debido a las inconsistencias en terminología, los estándares, la carencia de estructuras, los formatos heterogéneos, etc.

Desafíos

+ Construir modelos de proceso que definan el ciclo de vida del marcado de trazas.

+ Desarrollar técnicas para evaluar la habilidad de un conjunto de datos dado para soportar los métodos automáticos de traceability.

2.10. Sobre el cumplimiento

Problemas

- Los estándares pueden ayudar a asegurar procesos consistentes y completos, aunque abundan los estándares, no está claro si los investigadores o profesionales están enterados de la existencia de los mismos.

- La comunidad que engloba traceability son eruditos sobre técnicas y procesos de este tema, pero tienen poca influencia sobre los contenidos relacionados a traceability en los procesos estándar de ingeniería de software.

- No es claro cómo se puede demostrar el cumplimiento de los estándares y regulaciones .

Desafíos

+ Establecer un mecanismo de comunicación para hacer que la comunidad de expertos de traceability dictamine los estándares relacionados con la tecnología.

+ Lograr una presencia en la comunidad de estándares para influir y/o desarrollar estándares de traceability.

+ Como comunidad, desarrollar y promover escenarios válidos para probar que las herramientas, las técnicas y las metodologías de traceability cumplen con los estándares.

2.11. Sobre las mediciones y los Benchmarks

Problemas

- Los estudios empíricos son necesarios para demostrar la eficacia de los métodos de traceability y así, facilitar el trabajo colaborativo y evolutivo entre los investigadores y profesionales, sin embargo, hay una falta de diseños experimentales, metodologías y benchmarks comunes.

- Las medidas, métodos y métricas propuestas actuales no han sido validadas a través de estudios o pruebas empíricas.

- No existen o no se han realizado buenas pruebas o "benchmarks" de traceability y/o no son compatibles.

- No existen pruebas estándares de comparación para aplicar sobre los métodos y técnicas desarrolladas de traceability.

- La detección de errores en los enlaces de trazas es necesaria para determinar la eficacia del producto y el proceso, sin embargo los modelos de detección de errores actuales aún son primitivos e inválidos.

Desafíos

+ Definir procesos estándares para la realización de estudios empíricos durante la investigación de traceability.

+ Construir pruebas (benchmarks) para evaluar los métodos y las técnicas de traceability.

+ Definir medidas para evaluar la calidad de los enlaces de trazas tanto forma individual como la de un conjunto.

+ Desarrollar técnicas de evaluación de métodos y procesos de traceability.

2.12. Sobre la transferencia de tecnología

Problemas

- El objetivo de la investigación de traceability es la transferencia de soluciones eficaces para la industria, sin embargo en la realidad en la industria son reacios a probar técnicas nuevas donde la eficacia aún no fue demostrada.

- La carencia de diálogo entre los dos grupos, investigadores y profesionales, limita la accesibilidad de los investigadores a un conjunto de datos reales para testear nuevas técnicas e inhibe la retroalimentación de la industria a los investigadores.

- Los prototipos de Traceability son generalmente diseñados para mostrar demostraciones de conceptos, sin embargo no son suficientemente rigurosos para el campo de prueba de la industria.

Desafíos

+ Crear una infraestructura y un conjunto de métodos relacionados para organizar el proceso de transferencia de tecnología.

+ Identificar los casos de estudio exitosos y darle publicidad, para demostrar la efectividad de los costos de las técnicas de traceability en el ámbito industrial.

+ Identificar los usuarios de traceability y definir sus necesidades en términos de calidad, ciclo de vida, comunicación, etc.

+ Incorporar las herramientas de traceability que se encuentren a la vanguardia en los IDE estándares (tal como Eclipse) y las herramientas de administración de requerimientos industriales.

Capítulo 3

Caminando hacia la propuesta

A continuación se presenta una pequeña introducción a un conjunto de temas que se tuvieron en cuenta a la hora de definir el esquema de traceability propuesto que se presenta en el capítulo 4.

3.1. Generación de trazas

Uno de los desafíos presentados en [3] trata sobre cuál de los dos enfoques de generación de trazas que existen durante una transformación, implícita o explícita, es conveniente utilizar. En la generación implícita la transformación provee un soporte integral de traceability, en cambio en la explícita, queda en manos del desarrollador codificar las reglas de traceability como un modelo más de salida.

A continuación se detallan las ventajas y desventajas de cada enfoque:

3.1.1. Implícita

Ventajas

1. La mayor ventaja de la generación de trazas implícita es que no es necesario ningún esfuerzo adicional para obtener los enlaces de trazas entre los modelos de entrada y salida, dado que son generados en paralelo automáticamente con el modelo actual de la transformación.

Desventajas

1. El meta-modelo de traceability es fijo: como la mayoría de los enfoques de transformación tienen diferentes meta-modelos, lograr estandarizar entre estos diferentes enfoques es muy complejo.

2. Poca flexibilidad para controlar los datos de traceability, ésto incluye:

   a) El tipo de información guardada: cuando se trazan todos los elementos del modelo referenciado, el número de enlaces puede volverse incomprensible y por lo tanto menos útil. También al presentarse un modelo de transformación grande y complejo puede acarrear un problema en lo que respecta a rendimiento.

   b) El nivel de granularidad de la información de traceability: (esto es por ejemplo realizar trazas solo a nivel de archivo o bajar a nivel del contenido del mismo) esta configuración de los enlaces varía de un escenario de traceability a otro.

   c) Contexto de la información: por ejemplo por necesidades de un cliente que presente motivos de seguridad, puede requerir que no toda la información de un modelo tiene permitido ser trazada.

3.1.2. Explícita

Ventajas

1. Es posible lidiar con traceability como un modelo regular de salida de la transformación e incorporar reglas de transformación adicionales para generarlo. La elección del meta-modelo es entonces completa del programador y no depende del motor de transformaciones. Por lo tanto, la granulación de los enlaces es adaptable.

Desventajas

1. Se requiere un esfuerzo adicional para agregar reglas de transformación específicas para traceability, que pueden en consecuencia contaminar la implementación.

2. Como esta tarea es por lo general manual, es propensa a errores y consume mucho tiempo, más aun si pensamos que esta tarea se tiene que repetir para cada transformación que se realice.

3.2. Tipos de enlaces

Otro de los desafíos refiere a la semántica de los enlaces, es necesario frecuentemente distinguir entre los distintos tipos de enlaces que se pueden

presentar. Por ejemplo un enlace entre un requerimiento textual y un elemento de modelo, tiene una semántica distinta que una relación de refinamiento dentro de un modelo.

Los tipos de enlaces requeridos por lo general son fuertemente dependientes al proyecto. Por lo cual definir fijamente los tipos de enlaces semánticos tiene la consecuencia de menor flexibilidad para los enlaces que desee trazar el usuario, los cuales serán definidos por las necesidades del proyecto o la compañía.

Es importante remarcar que la determinación de la semántica de un enlace es guiada por la razón del usuario sobre qué quiere realizar o representar con dicho enlace. No predefinir la semántica correctamente de un enlace puede resultar en fallas de razonamientos.

3.3. Estrategias de almacenamiento

Según [7] hay dos tipos principales de estrategias para almacenar y administrar la información de traceability. En la primera, la información de traceability se encuentra embebida en los modelos a los que ella refiere. En cambio en la segunda, dicha información se encuentra almacenada de forma separada de los modelos:

3.3.1. Almacenamiento de enlaces Intra-Modelo

Como ya se dijo, bajo esta estrategia la información de traceability es almacenada dentro de los artefactos a los que refiere, esto puede ser mediante elementos del modelo o mediante atributos de los elementos del modelo (como etiquetas o propiedades).

Es una estrategia sencilla y amigable, pero puede ser muy problemática por varias razones. Si los enlaces son dirigidos y almacenados solamente en el modelo origen, éstos no son visibles en el modelo destino, a la inversa (almacenados en el destino) nos encontramos con el mismo problema pero en el origen. Por otro lado, si la información de traceability es almacenada en ambos modelos, entonces nos encontramos con el problema de que dicha información se mantenga consistente por cada vez que se realice un cambio.

A todo lo anterior se suma el problema de la polución que se genera en el modelo con la información de traceability que es ajena a su contexto o fin original, dicha polución puede lograr que el modelo se vuelva muy difícil de comprender y mantener.

Por otro lado, en un entorno MDE es común que los modelos tengan sus propias representaciones y semánticas, lo cual puede volver más complejo

diferenciar la información de traceability de los objetos que representan el modelo del dominio.

Como resultado a los inconvenientes anteriores, el análisis automatizado de la información de traceability se hace muy difícil. Los enfoques principales que hacen uso de esta estrategia utilizan construcciones de lenguajes específicas, por ejemplo determinados tipos de enlaces de traceability están representados en los diagramas Unified Modeling Language (UML) mediante el uso de las estereotipos como «refines».

3.3.2. Almacenamiento externo de los enlaces

En esta estrategia la información de traceability se encuentra almacenada de forma separada a los modelos a los que refiere, esto es en un modelo aparte. Esta propuesta tiene dos claras ventajas, la primera es que los modelos origen y destino se mantienen totalmente limpios, con lo cual la polución nombrada en el almacenamiento intra-modelo no sucede. Y la segunda, dado que el modelo en donde se almacena los traceability links se encuentra definido por un meta-modelo con una clara semántica, logra que el proceso de análisis de la información sea mucho más fácil que en la otra estrategia.

Un requisito previo para el almacenamiento externo de los enlaces de traceability, es que los diferentes elementos del modelo tengan identificadores únicos, de modo que las trazas que los relacionan se pueden resolver inequívocamente. Un ejemplo es el mecanismo propuesto por MetaObject Facility (MOF) y por Eclipse Modeling Framework (EMF) en la forma de un identificador xmi.id .

3.4. Meta-modelos

Los modelos que determinan los enlaces de traceability se encuentran definidos cada uno por un meta-modelo, éste puede ser clasificado como un "meta-modelo de traceability de propósito general" o un "meta-modelo de traceability de caso específico".

3.4.1. Meta-modelo de propósito general

En este caso, nos encontramos con un meta-modelo genérico que permite la captura de las relaciones entre cualquier tipo de elementos de modelo. En este meta-modelo, un enlace de traceability se puede conectar con cualquier número de elementos, de cualquier tipo y de cualquier modelo. Las principales ventajas de este tipo de meta-modelo de propósito general son la simplicidad

y la uniformidad (dado que todos los modelos conforman el mismo meta-modelo) con lo cual se mejora la interoperabilidad de las herramientas con capacidades de importar, exportar y gestionar traceability en un formato común.

Por otro lado, como el meta-modelo de propósito general no capta casos específicos de enlaces de trazas fuertemente tipados, o sea con semántica y restricciones definidas rigurosamente, se abre la puerta a establecimientos de enlaces ilegítimos. Como por ejemplo en el caso que se quiera representar trazas entre un diagrama de clases y un modelo de base de datos relacional, sabemos que existen vínculos entre las clases del primer modelo y las tablas del segundo, un meta-modelo de traceability genérico permite el establecimiento de enlaces ilegítimos tales como una clase relacionada con una columna.

La provisión de mecanismos de extensión junto con el meta-modelo de propósito general es un método de uso frecuente para permitir un mejor apoyo para el caso de los requisitos específicos. Sin embargo, todavía carecen de la eficacia de los meta-modelos de casos específicos para capturar estos tipos de casos que requieren tal legitimidad entre la información y su semántica.

3.4.2. Meta-modelo de caso específico

En este caso, para cada escenario de traceability se define un meta-modelo específico. Este meta-modelo de traceability captura enlaces fuertemente tipados para casos específicos con una semántica bien definida, que pueden o no incluir restricciones de corrección. Debido a su naturaleza de tipado fuerte y las restricciones asociadas, restringe a los usuarios y las herramientas para que sólo puedan establecer enlaces legítimos. Por otro lado, la definición de un meta-modelo para cada caso específico requiere mucho esfuerzo en su construcción, así como herramientas que soporten, o mejor dicho, ofrezcan la posibilidad de aceptar diferentes meta-modelos de traceability.

Para ser fuertemente tipado el meta-modelo de traceability necesita referir explícitamente a los tipos de elementos que se encuentran definidos en otros meta-modelos. Por ejemplo, consideremos que es necesario definir un meta-modelo de traceability que permita el establecimiento de enlaces entre instancias de A (del meta-modelo MMa) e instancias de B (a partir de MMb), pero no entre dos instancias de A o dos de B. Para capturar tal meta-modelo, la tecnología de modelado que se use no debe tomar cada meta-modelo como un espacio cerrado, sino que por el contrario debe permitir referencias inter-meta-modelo. Un ejemplo de tecnología que soporta referencias inter-meta-modelo es el framework EMF.

Mas allá de que un meta-modelo de traceability definido utilizando una

tecnología que permita referencias inter-meta-modelo puede brindar tipos seguros, encontramos frecuentemente otras restricciones que necesitan especificarse y que dicho meta-modelo no puede capturarlas. Por ejemplo, tomando como referencia el ejemplo anterior, podríamos precisar que cada instancia A de MMa sólo se puede vincular a no más de una instancia B de MMb. Para especificar tales restricciones, se requiere un lenguaje de especificación de restricciones que pueda expresar restricciones que abarquen elementos que pertenezcan a modelos definidos por diferentes meta-modelos. En la actualidad el Object Constraint Language (OCL) carece de esta capacidad, ya que no proporciona las construcciones para la expresión de restricciones que atraviese modelos (cross-model). Ejemplos de lenguajes de restricción que soportan el establecimiento con tales restricciones incluyen el Epsilon Validation Language (EVL) y el XLinkit toolkit.

La combinación de un meta-modelo de traceability fuertemente tipado conjunto con la verificación de restricciones inter-modelo restringe a los usuarios y a las herramientas a establecer y mantener sólo enlaces de trazas con sentido, que pueden ser automáticamente validadas para descubrir posibles omisiones e inconsistencias. Estas cuestiones pueden surgir, ya sea durante el establecimiento de las trazas o más tarde en el ciclo de vida de los modelos donde ya los enlaces de trazas ya han sido establecidos.

3.5. Una clasificación genérica de traceability

Conforme un proyecto de desarrollo crece, la administración de la información de traceability que en consecuencia se va generando se vuelve extremadamente compleja, una jerarquía de clasificación de dicha información resultará esencial para poder entenderla y administrarla mejor. En [10] se encuentra una propuesta de clasificación de traceability conjunto con la descripción del proceso usado para su obtención al que llamaron Traceability Elicitation and Analysis Process (TEAP).

Esta propuesta comienza con una clasificación o meta-modelo de traceability inicial, luego de forma iterativa e incremental esta clasificación se va refinando siguiendo las siguientes tareas: obtención, análisis y clasificación. En la obtención se identifica un enlace de traza y sus relaciones. En el análisis, se abstraen las principales características del enlace obtenido identificando las restricciones, sus relaciones y generalizaciones. Por último, se define la clasificación a la que pertenece.

El meta-modelo inicial se puede ver en el dibujo 3.1, en el que se encuentran los siguientes conceptos fundamentales: artefacts, trace links y operations. Artefact refiere tanto a artefactos MDE (por ejemplo modelos es-

pecíficos de dominio) como a no MDE (por ejemplo hojas de cálculo), en Operation se encierran las operaciones bien manuales como automáticas que determinan qué información de traceability debe almacenarse, y por último Trace Link engloba la clasificación que da inicio con la idea de traceability implícita (Implicit Link) y explícita (Explicit Link). En este contexto, traceability implícita refiere a los enlaces de trazas que son creados y manipulados por la aplicación de operaciones MDE, y por su parte explícita refiere a los enlaces que se encuentran concretamente ya representados en los modelos.

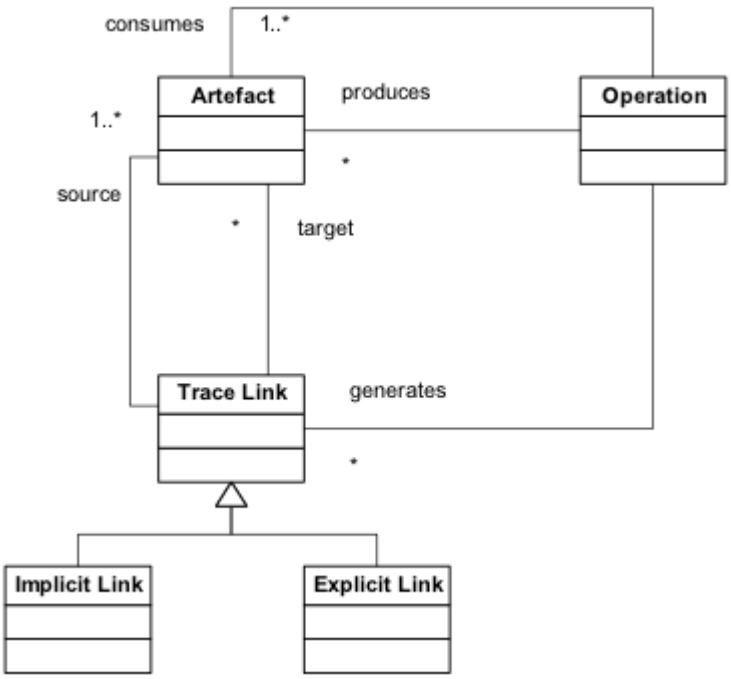

Figura 3.1: Clasificación de traceability inicial

3.5.1. Clasificación de enlaces de trazas implícita

En la jerarquía de esta sección que se muestra en el dibujo 3.2 se representan el conjunto de operaciones de MDE principales: consulta (Query Link), transformación (M2M Link), transformación modelo a texto (M2T Link), composición (Composition Link), actualización (Update Link), creación (Creation Link), eliminación (Delete Link), entre otras.

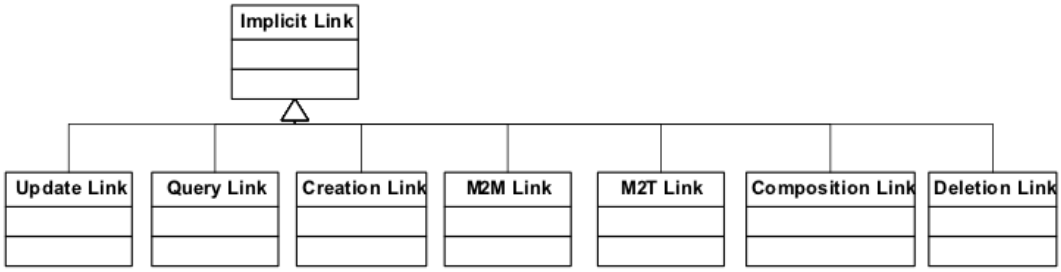

Figura 3.2: Jerarquía de links implícitos

3.5.2. Clasificación de enlaces de trazas explícitos

En un principio esta clasificación se encuentra dividida en dos grandes grupos representados por las siguientes clases básicas: links modelo-modelo (por ejemplo las relaciones de dependencias UML) y links modelo-artefacto (por ejemplo el enlace entre un modelo y un documento de texto que dicta un requerimiento), en el dibujo 3.3 aparecen como Model-Model Link y Model-Artefact Link respectivamente.

Model-Model Link se encuentra a su vez subdividido en links estáticos (Static Link) y dinámicos (Dynamic Link). Los primeros representan relaciones estructurales que no cambian con el tiempo, en cambio los segundos representan información de los modelos que si pueden llegar a variar.

Los links estáticos pueden ser, o links consistentes (Consistent-With) donde dos modelos deben mantenerse de acuerdo o consistentes entre si, o links de dependencias (Dependency) donde la estructura y/o la semántica de un modelo depende de otro. Por su parte, los links de dependencia pueden ser: links de relación de subtipo (Is-A), links de referencias, links de subconjuntos, de importación y exportación, de uso, de refinamiento (Refines), etcétera.

Entre los links dinámicos se incluyen los links de llamadas (Calls) donde un modelo hace uso de métodos provisto por otro, los links de notificación (Notifies) donde es necesario almacenar información que puede ser manejada automáticamente. También las relaciones en tiempo de diseño como los links de generación o construcción (Generates), que indican qué información de un modelo es usada para producir o deducir otro, y las relaciones de sincronización (Synchronized With), donde el comportamiento de un conjunto de modelos se realiza de forma sincronizada.

El alcance de Model-Artefact Link es muy amplio, por tal motivo es esta clasificación solo se resumen un subconjunto de ejemplos, entre ellos: la relación satisface (Satisfies), que indica qué propiedad o requerimiento de un artefacto es satisfecha por un modelo; los links de asignación (Allocated-To), usados cuando la información de un artefacto no modelo es asignada a un

modelo específico que la representa; la relación de realización (Performs), que indica qué tarea descripta por un artefacto es llevada a cabo por el modelo; las relaciones explica y respalda (Explains y Supports respectivamente) que dictan qué modelo se encuentra explicado/respaldado por un artefacto no modelo.

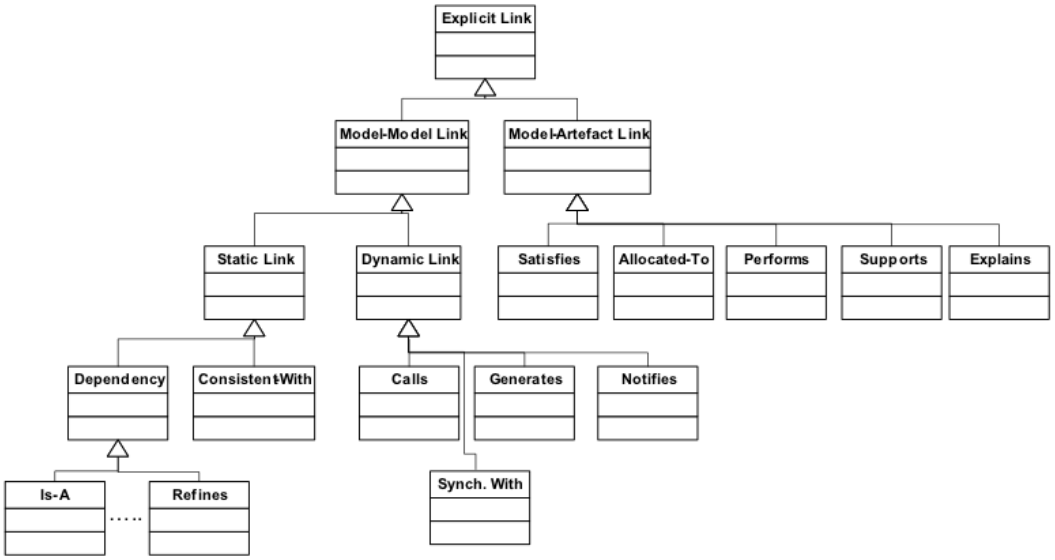

Figura 3.3: Jerarquía de links explícitos

Capítulo 4

Manos a la obra

En este capítulo se introduce, desarrolla y fundamenta el esquema de traceability propuesto en la presente tesis.

4.1. Esquema de traceability

4.1.1. Introducción

La idea fundamental es lograr un meta-modelo bien simple en el que se capture solamente las ideas más relevantes de traceability y se almacene la información mínima necesaria. Gracias a todo lo anterior, y solo así, logramos obtener el concepto deseable de independencia del modelo con respecto al dominio en el que se quiera utilizar la funcionalidad de traceability. Concepto necesario para poder llegar a definir un esquema de propósito general que apunte principalmente a dar solución a los siguientes desafíos:

- Lograr un esquema que represente todas las trazas independientemente de como fueron generadas, si de forma explícita o implícitas, su granularidad y semántica.
- Mantener la comunicación lo más simple posible, tanto la que existe entre los distintos usuarios en sus respectivos roles como la de los distintos tipos de herramientas que necesiten traceability.
- La adaptación de una herramienta tiene que ser una tarea sencilla, cualquiera sea cuando desee agregar la funcionalidad de traceability mediante el uso de este esquema.
- Aún dada la infinidad de artefactos que se pueden presentar, el esquema siempre tiene que ofrecer información semántica de los mismos, con el

fin de facilitar la validación de enlaces legítimos.

- Lograr identificar unívocamente cada artefacto sobre el que se desee realizar traceability.

4.2. El esquema de traceability

Con el fin de lograr respetar todas las consignas fundamentales listadas en 4.1.1, se definió una propuesta de un esquema de traceability que se encuentra representado mediante un diagrama de clases que se puede observar en el dibujo 4.1.

Como se puede observar, el meta-modelo se puede dividir lógicamente en dos partes principales, la primordial en la que se encuentran las abstracciones fundamentales de los elementos mínimos necesarios en cualquier herramienta que desee ofrecer traceability, y una secundaria en la que se engloba la configuración del esquema. Esta configuración permite ofrecer un meta-modelo flexible a cualquier escenario/dominio en el que se desee hacer uso de traceability.

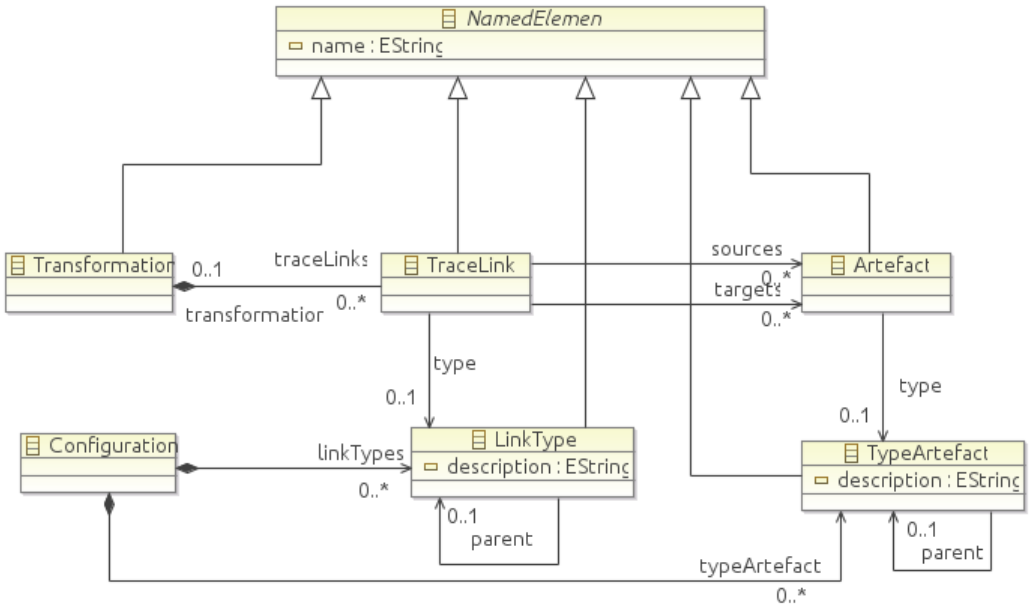

Figura 4.1: Esquema de traceability propuesto

Clases fundamentales

Entre los elementos fundamentales se encuentra la clase TraceLink, la cual representa una traza. Esta clase está compuesta por un nombre que sirve como identificador (name), un tipo con el cual se determina su semántica (type), una referencia de la transformación a la que pertenece la cual es optativa porque una traza puede o no ser producto de una transformación, y por último dos conjuntos de artefactos: el de los orígenes/fuentes (sources) y el de los objetivos/resultados (targets).

Los artefactos se encuentran representados por la clase Artefact, cuya identificación también pasa por su nombre (name), su descripción también por un diccionario de clave-valor, y por último su semántica también puede ser definida por un tipo definido en la configuración.

Por último entre los elementos fundamentales, tenemos la clase Transformation que abstrae de las transformaciones un nombre para su identificación, una descripción vía un diccionario y la colección de TraceLinks que son producto de su ejecución.

Configuración del esquema

Esta parte refleja la configuración necesaria para lograr un esquema adaptable a la infinidad de escenarios en los que puede aplicarse el mismo. Dicha configuración consiste en la definición de dos jerarquías, una que define la semántica de los artefactos que se van a tracear, y la otra para determinar la semántica de las trazas que pueden llegar a crearse. Ambas jerarquías están determinadas mediante las clases LinkType y TypeArtefact respectivamente, éstas comparten la siguiente estructura: un nombre (name) que las identifica, un texto que posibilita describirlas (description) y una relación (parent) que asocia una clasificación con su padre, gracias a dicha relación se logra la estructura jerárquica.

Un ejemplo de una configuración de tipos de trazas puede ser la jerarquía propuesta en la sección 3.5, que se puede ver siguiendo los diagramas 3.1, 3.2 y 3.3.

4.2.1. Lo que no ofrece

A continuación se lista un conjunto de aspectos y/o funcionalidades que el esquema propuesto actual no ofrece o tiene en cuenta:

- funcionalidad de versionado: que permita navegar por las trazas y refleje las modificaciones conjunto con las que se fueron realizando sobre los

artefactos relacionados en el tiempo, a lo largo de la evolución de todas las tareas que forman parte de la ingeniería de un software.

- métodos de detección o información de errores: que determine o informe cuando una traza es inválida.

4.3. El prototipo

En la presente sección se explica y presenta el prototipo desarrollado para presentar la idea de la herramienta de traceability propuesta en la tesis.

4.3.1. La herramienta

La idea es lograr una herramienta que dada la definición de una transformación en conjunto con sus modelos de entrada y modelos resultados de su ejecución, retorne y muestre el mapa de trazas que se generará implícitamente producto de la transformación definida. El modelo del mapa de trazas se encuentra definido según el esquema de traceability propuesto y presentado en 4.2.

El prototipo de la herramienta propuesta trabaja sobre transformaciones Query/View/Transformation (QVT), pero en la sección

Arquitectura

4.4. Traceability en ATL

La información de traceability forma parte del lenguaje Atlas Transformation Language (ATL), ésta ayuda en la interacción de cada regla de transformación cuando necesita tomar la salida de alguna otra regla por medio del mecanismo interno de traceability implícito. Sin embargo, actualmente ATL ofrece un acceso muy limitado a dicha información de traceability sólo por medio del método resolveTemp(). Además, ésta información es eliminada ni bien la ejecución de la transformación termina. Dada las carencias enumeradas antes, para lograr obtener alguna información de traceability en esta tecnología es necesario implementar de dicha funcionalidad.

Existen varias propuestas de implementación, una de ellas, la que se presenta en [9] en donde propone considerar tanto a los programas de transformación como a la información de traceability como modelos, entonces mediante la definición de una nueva transformación modificar los programas para que generen la información de traceability como una salida más

al modelo resultado original. Con este enfoque, la generación de código de traceability está claramente separada de la lógica de la transformación y la misma se puede agregar después de que un programa haya sido escrito. Por lo cual, es posible añadir el soporte para nuevos formatos o modificar la granularidad o rango de las trazas sin interferir con la lógica del programa.

Otras propuestas hacen uso del mecanismo de traceability implícito de ATL, como copiar toda la información de trazas durante la ejecución de la transformación, o directamente persistirla y dejarla accesible en un archivo aparte, pero para todas estas implementaciones se requiere de modificaciones del motor de ejecución de ATL.

Capítulo 5

Descripción de tecnologías

En el presente capítulo se realiza una breve introducción sobre cada una de las tecnologías que fueron utilizadas para la implementación del prototipo que hace uso del esquema propuesto de traceability.

5.1. Eclipse

Empezamos con el proyecto base sobre el cual el prototipo va a funcionar/ejecutarse, en la presente sección daremos una introducción a Eclipse en la que se presenta el proyecto y se detalla en breve la plataforma con sus principales componentes y/o funcionalidades.

5.1.1. El proyecto

Según se presenta en [19, 17] Eclipse es un proyecto de desarrollo de software de código abierto, cuyo propósito es proveer una plataforma de herramientas altamente integradas para la construcción, implementación y administración de software a lo largo de todo su ciclo de vida.

El proyecto núcleo es un framework genérico para la integración de herramientas conjunto con un entorno de desarrollo Java que ya se incluye para usarlo. Otros proyectos extienden el framework núcleo para soportar distintos tipos de herramientas y ambientes de desarrollo específicos. Estos proyectos en Eclipse están implementados en Java y pueden ser ejecutados en muchos sistemas operativos.

La comunidad Eclipse tiene más de 200 proyectos, los cuales pueden conceptualmente organizarse dentro de las siguientes 7 categorías:

1. Enterprise Development

2. Embedded and Device Development
3. Rich Client Platform (RCP)
4. Rich Internet Applications (RIA)
5. Application Frameworks
6. Application Lifecycle Management (ALM)
7. Service Oriented Architecture (SOA)

Eclipse hace uso de la Eclipse Public License (EPL), dicha licencia comercial permite a las organizaciones incluir software Eclipse en sus productos comerciales, mientras que al mismo tiempo les solicita en retorno un aporte a la comunidad con algo del producto derivado comercializado.

5.1.2. La plataforma Eclipse

La plataforma Eclipse es un framework para la construcción de IDEs, el mismo ha sido descripto como "un ambiente para cualquier cosa y nada en particular". La plataforma define simplemente la estructura básica del IDE, luego mediante la definición de herramientas específicas que amplían y se conectan al framework terminan definiendo un IDE particular de forma colectiva.

Arquitectura de plugins

En Eclipse la unidad básica de funcionamiento, o sencillamente un componente, es llamado plugin-in. Tanto la plataforma misma de Eclipse como las herramientas que la extienden están compuestas por éstos plug-ins. Una herramienta sencilla puede consistir de un simple plug-in, pero las más complejas por lo general están divididas en varios de éstos.

Un plug-in incluye todo lo necesario para la ejecución del mismo, ésto puede ser: código Java, imágenes, textos, etc. También incluye un archivo manifiesto (plugin.xml), en el que se declaran las interconexiones con otros plug-ins.

Durante el arranque la plataforma Eclipse descubre todos los plug-ins disponibles, sin embargo éstos son sólo activados cuando es necesaria su ejecución con el fin de no ralentizar el arranque.

Workspace

Las herramientas integradas en Eclipse trabajan con archivos y carpetas ordinarias, pero también disponen de una Interfaz de programación de aplicaciones o Application Programming Interface (API) de alto nivel que define los siguientes componentes: recursos (resources), proyectos y un espacio de trabajo (workspace).

Resource Un recurso es la representación que da Eclipse de un archivo y/o una carpeta, a los que provee de capacidades adicionales como detectores de cambios (change listeners), marcadores como las listas por hacer (to-do list) y/o mensajes de errores, y un registro de historia de cambios.

Project Un proyecto es un recurso especial de tipo carpeta que es asignada por el usuario a una carpeta del sistema de archivos, es la que contiene todos los recursos del proyecto y la que define el tipo del mismo.

Workspace El workspace es el espacio de trabajo o contenedor virtual en el que se encuentran todos los proyectos del usuario.

Workbench

El Workbench es la ventana principal que se le presenta al usuario cuando ejecuta la plataforma Eclipse, se encuentra implementada usando Standard Widget Toolkit (SWT) y JFace. Esta ventana principal, que se puede apreciar en la imagen 5.1 está compuesta de vistas, editores y perspectivas.

Editores Éstos permiten al usuario abrir, editar y guardar distintos tipos de objectos.

Vistas proveen información de algún objeto sobre el que el usuario se encuentra trabajando en el Workbench. Por ejemplo, una vista puede asistir un editor dando información sobre el documento que se está editando.

Perspectivas Una perspectiva es sencillamente una agrupación de vistas y editores de manera que en su conjunto dan apoyo en una actividad completa.

5.1.3. Resumen y más información

Para cerrar y en resumen, la plataforma Eclipse proporciona un núcleo de elementos básicos y un conjunto de APIs genéricas, como el workspace

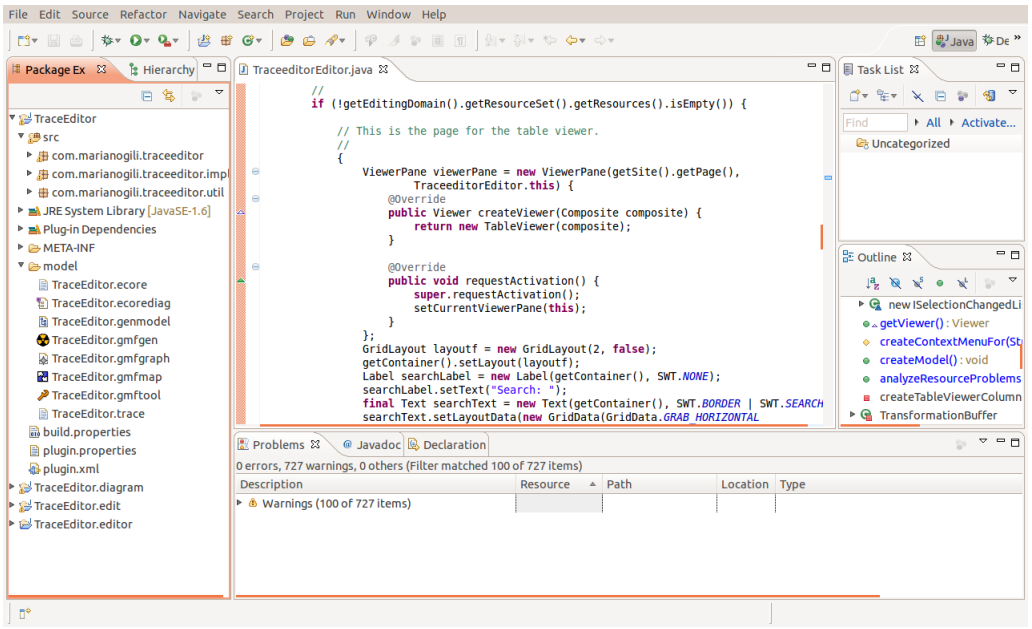

Figura 5.1: Elcipse Workbench

y el workbench, y varios puntos de extensión a través del cual se pueden integrar nuevas funcionalidades. A través de estos puntos de extensión, las herramientas escritas como plug-ins independientes pueden extender la plataforma Eclipse.

Para obtener más información sobre el proyecto Eclipse ver [17, www.eclipse.org], y para información técnica consultar [18, Eclipse Platform Technical Overview].

5.2. Eclipse Modeling Framework

A continuación proseguimos con una de las tecnologías del proyecto Eclipse usada para la implementación de una parte del prototipo, el framework EMF [15].

5.2.1. El framework EMF

El EMF es un framework de modelado y generación de código para herramientas de construcción y otras aplicaciones basadas en un modelo de datos estructurados. A partir de una especificación de un modelo descrito en XML Metadata Interchange (XMI), EMF provee herramientas y un entorno de ejecución para producir un conjunto de clases Java para el modelo y un

conjunto de clases adaptadoras que permiten la visualización y la edición de dicho modelo.

EMF consiste de tres partes fundamentales:

EMF El framework EMF base incluye un meta-modelo (Ecore) para la descripción de modelos y soporte para un entorno de ejecución para dichos modelos que incluye las funcionalidades de notificación de cambios, persistencia por defecto vía serialización XMI y una librería muy eficiente para la manipulación de objectos EMF genérica.

EMF.Edit El framework EMF.Edit incluye clases genéricas reusables para la construcción de editores de modelos EMF. Este provee:

- Clases proveedoras de contenido y etiquetas, acceso a propiedades orígenes y otras clases convenientes que permiten a los modelos EMF ser visualizados mediante entornos visuales estándares de escritorio (vía JFace) y/o hojas de propiedades.
- Un framework de comandos que incluye un conjunto de clases que implementan comandos para la construcción de editores con soporte íntegro de deshacer y rehacer (undo/redo).

EMF.Codegen La funcionalidad de generación de código EMF es capaz de generar todo lo necesario para la construcción de un editor de un modelo EMF completo. Ésta incluye una Interfaz de Usuario Gráfica o Graphical User Interface (GUI) desde la cual las opciones de generación pueden ser especificadas y los generadores son invocados. Dicha funcionalidad de generación aprovecha el componente de Eclipse Java Development Tooling (JDT).

Tres niveles de generación de código son soportados:

Model Proporciona la implementación de interfaces y clases Java para todas las clases del modelo, además la implementación de una clase fábrica y el paquete.

Adapters Genera la implementación de las clases, llamadas ItemProviders, que se adaptan a las clases del modelo para su edición y visualización.

Editor Produce un editor estructurado apropiadamente que se ajusta al estilo recomendado para los editores de modelo EMF de Eclipse y sirve como punto de partida para comenzar con la personalización.

5.2.2. El (Meta) modelo Ecore

Ecore es el modelo usado para representar modelos en EMF. Ecore es en sí mismo un modelo EMF, por lo cual es su propio meta-modelo.

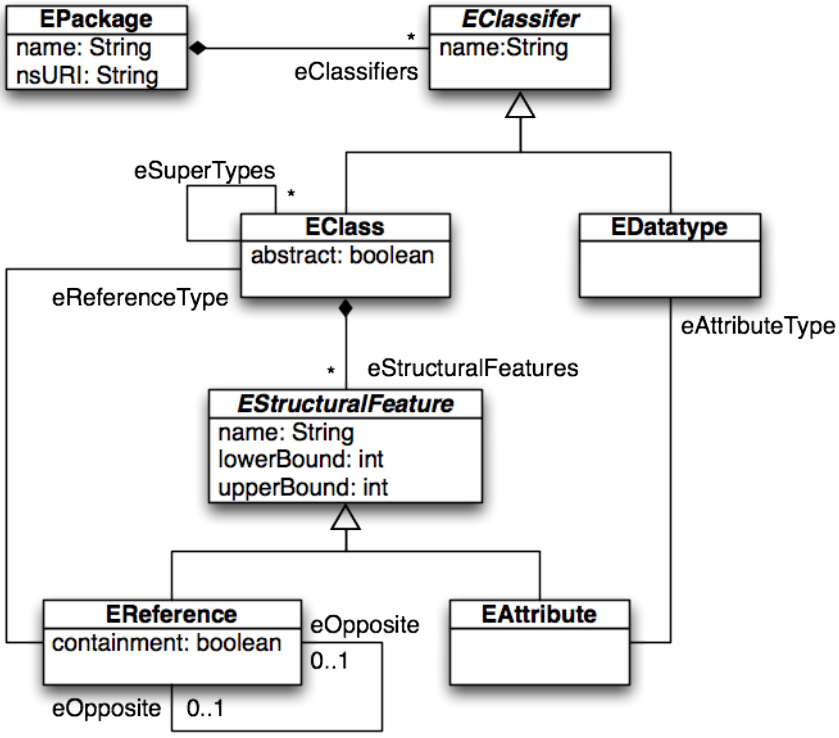

Figura 5.2: Modelo Ecore simplificado

Un modelo simplificado de Ecore se puede observar en la imagen 5.2, en particular lo conforman las siguientes cuatro clases Ecore principales:

EClass Usada para representar una clase en el modelo. Tiene un nombre, atributos y referencias.

EAttribute Para representar en el modelo un atributo. Tiene un nombre y un tipo.

EReference Para representar una relación entre clases. Tiene un nombre, una marca que indica si es una composición, y el tipo de clase referenciada u objetivo que es otra clase.

EDataType Representa el tipo de un atributo. El mismo puede ser un tipo primitivo (int, float) o un tipo objeto.

5.2.3. Beneficios y más información

Además de incrementar la productividad gracias a la generación automática de código, el uso del framework EMF provee otros beneficios: la notificación de cambios, una funcionalidad de persistencia (como la serialización XMI entre otras), y una API genérica reflexiva muy eficiente que sirve para la manipulación de objetos EMF. Sin embargo, uno de los beneficios más importantes de este framework es que EMF provee las bases para la interoperabilidad con otras herramientas y aplicaciones que se basan en él.

Más información sobre el framework EMF se puede obtener en [15] y [19].

5.3. Graphical Modeling Framework

En la presente sección analizamos el framework Graphical Modeling Framework (GMF), el mismo fue usado para el desarrollo de la parte gráfica del editor de trazas del prototipo.

5.3.1. El framework GMF

GMF es un framework para la construcción de editores gráficos de modelado para la plataforma Eclipse, por ejemplo editores UML, ECore, de procesos de negocios, diagramas de flujo, etc. Este framework está compuesto por un componente de generación (GMF Tooling) y un entorno de ejecución (GMF Runtime) que ayudan en el desarrollo de editores gráficos que se basan en EMF y Graphical Editing Framework (GEF).

GMF Tooling incluye por un lado editores para crear y/o modificar los modelos que describen los aspectos de notación, semántica y utilidad de un editor gráfico, y por otro, un generador que da como resultado la implementación del editor definido.

GMF Runtime Los plug-ins generados con GFM-Tooling dependen de este componente que es el entorno de ejecución sobre el que va a correr el editor.

5.3.2. Arquitectura

En el diagrama de componentes que podemos ver en la figura 5.3 se muestran las dependencias existentes entre el editor gráfico generado, el entorno GMF Runtime, EMF, GEF y la plataforma Eclipse. Como se puede ver, el editor gráfico GMF depende del componente GMF runtime, y a su vez hace uso directo de EMF, GEF y la plataforma Eclipse.

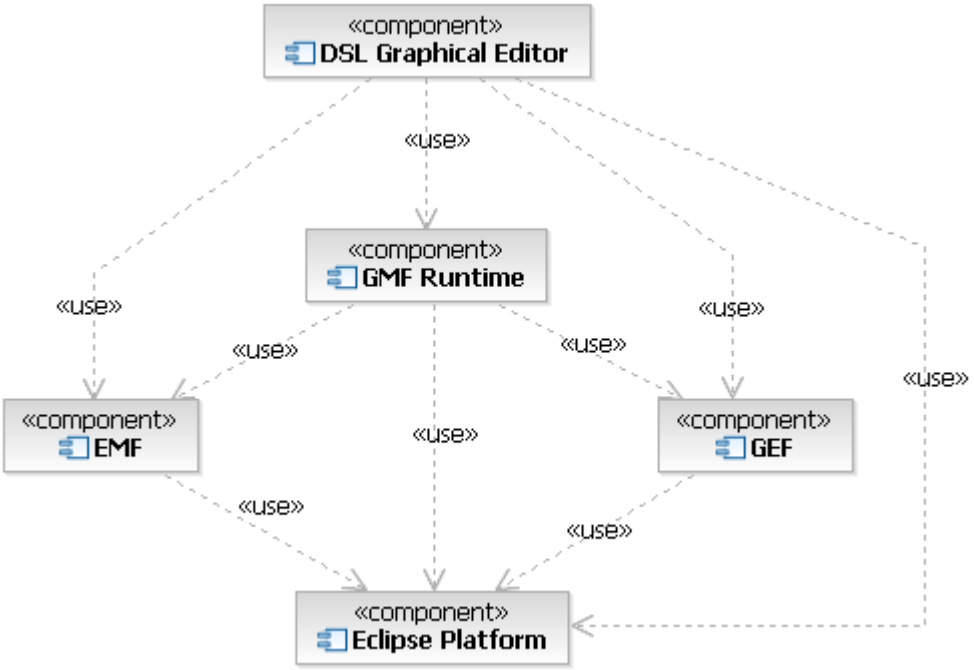

Figura 5.3: Arquitectura GMF

5.3.3. Modelos y flujo de trabajo

En el diagrama 5.4 se encuentran los principales componentes y modelos usados durante el desarrollo de un editor gráfico mediante el framework GMF. Entre ellos se encuentra el concepto de **modelo de definición gráfica**, el cual contiene toda la información relacionada con los elementos gráficos que formarán parte del editor (figuras, nodos, enlaces, etc), pero que no tiene ninguna conexión directa o es dependiente con ninguno de los componentes del modelo de dominio para el cual ofrecerá la representación y/o edición. También tenemos el **modelo de definición de herramientas** que es opcional y usado para el diseño de la paleta, el menú y las barras de herramientas.

Tanto la definición gráfica como la de las herramientas pueden funcionar para modelos variados de dominios, éste es uno de los objetivos de GMF, lograr que estas definiciones sean reusables para distintos dominios que se puedan presentar. Lo anterior se logra gracias al uso de un **modelo de asignación o mapeo**, que se encuentra separado y vincula las definiciones gráficas y las herramientas con los correspondientes modelos de dominios seleccionados.

Una vez fueron definidas los mapeos o asignaciones, GMF dispone de un **modelo de generación** el cual posibilita la definición de los detalles para la implementación de la siguiente y última fase, la generación del editor.

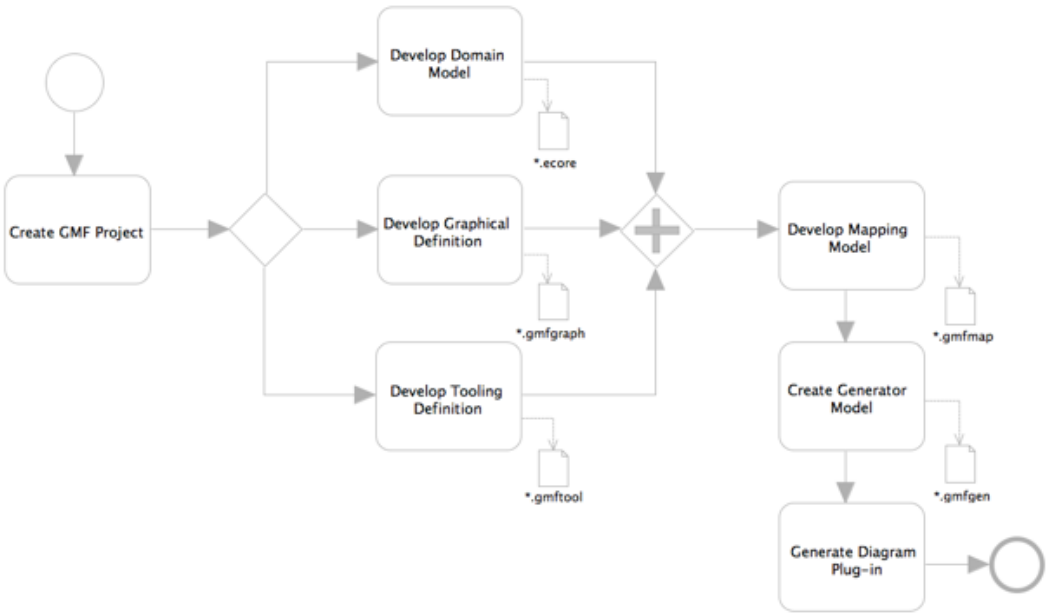

Figura 5.4: Flujo de trabajo de GMF

Flujo de trabajo

1. Creación del modelo del dominio, en este modelo se define la información no gráfica que gestiona el editor.

2. Creación del modelo de definición gráfica, en el que se definen los elementos gráficos que se mostrarán/presentarán en el editor.

3. Creación del modelo de asignación gráfica, que es el que define la correspondencia entre los elementos del modelo del dominio con los elementos gráficos del modelo de definición gráfica.

4. Generación del editor gráfico.

5. Mejorar el editor gráfico por medio de la edición del código del plug-in generado.

5.3.4. Más información

Más información sobre el framework GMF se puede obtener en [20], [21] y [22].

5.4. Atlas Transformation Language

A continuación se da una introducción a la tecnología ATL, que no solo refiere a un lenguaje de transformación de modelos como su nombre puede confundir, sino que además trata de un conjunto de herramientas de desarrollo construidas para ser ejecutadas sobre la Plataforma Eclipse (introducida en 5.1.2).

5.4.1. ¿Qué es ATL?

En el campo de MDE, ATL nos ofrece un medio para especificar la forma de producir un conjunto de modelos resultados/destinos a partir de un conjunto de modelos fuentes.

El lenguaje ATL es un híbrido de la programación declarativa e imperativa, dado que aunque el estilo declarativo es el más conveniente para la definición de las transformaciones, ATL también provee la posibilidad de construcciones imperativas con el fin de facilitar la especificación de algunos mapeos que en forma declarativa pueden llegar a resultar muy complejos de expresar.

Por otro lado, el IDE ATL provee un conjunto de herramientas estándar con el fin de facilitar el desarrollo de las transformaciones mediante este lenguaje como el resaltado de sintaxis, el auto-completado de código, un depurador, entre otras.

5.4.2. Conceptos de ATL

Un modelo fuente se transforma en un modelo destino gracias a la definición de una transformación escrita en ATL, la cual también es un modelo. Los modelos fuente, destino y la definición de la transformación, cada uno conforman a sus respectivos meta-modelos y, a su vez, todos los meta-modelos se ajustan a MOF o Ecore. Esta relación se puede observar en la imagen 5.5.

Una transformación ATL es unidireccional, o sea que trabaja sobre un modelo fuente de solo lectura y produce un modelo destino de solo escritura. Durante la ejecución de una transformación, el modelo fuente puede ser navegado pero no cambiado, en cambio el modelo destino no puede ser navegado.

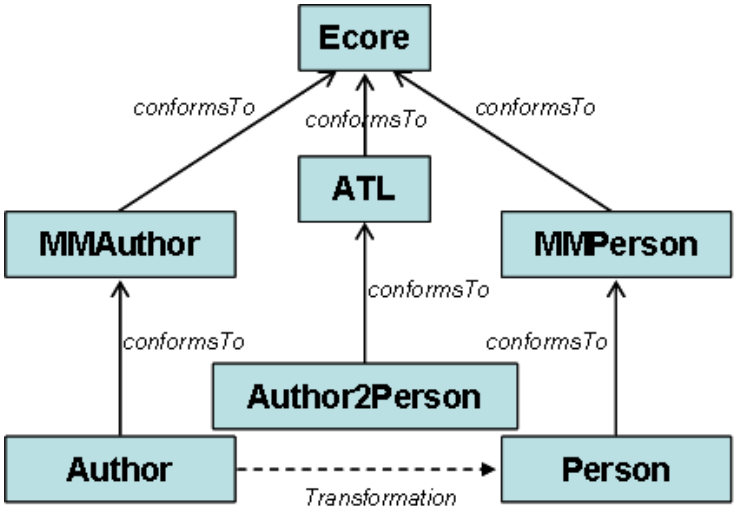

Figura 5.5: Modelos, transformaciones y sus meta-modelos

5.4.3. El lenguaje ATL

En este lenguaje de transformaciones modelo a modelo o Model-to-Model Transformation (MMT), las operaciones de transformación son especificadas mediante módulos ATL. Cada módulo ATL permite a un desarrollador especificar la forma de producir un conjunto de modelos resultados desde un conjunto de modelos fuentes/orígenes.

Además de módulos, este lenguaje permite crear programas que transforman modelos en tipos de datos primitivos (como booleanos, enteros o cadenas), lo cual se logra mediante la especificación de unidades llamadas ATL queries.

Por último ATL ofrece la posibilidad de desarrollar librerías independientes que pueden ser importadas a lo largo de diferentes tipos de unidades ATLs. Ésto nos da una forma conveniente de factorizar el código que va a ser usado por muchas unidades ATL.

5.4.4. Más información

Más información sobre la tecnología ATL se puede encontrar en la sección de documentación de la página del proyecto [23, www.eclipse.org/atl/].

5.5. QVT

Por último en este capítulo dedicado a las tecnologías que forman parte de la presente tesis, se introduce el estándar para transformaciones especificado por la OMG, el lenguaje QVT.

5.5.1. Introducción a QVT

QVT como ya se dijo antes es el estándar de OMG para la definición de transformaciones. La especificación de QVT es híbrida, como en el caso de ATL, relacional (o declarativa) y operacional (o imperativa). Comprende tres diferentes lenguajes MMT: dos lenguajes declarativos llamados Relations y Core, y un tercer lenguaje, de naturaleza imperativa, llamado Operational Mappings. Puede observarse la relación de los meta-modelos de cada uno de los lenguajes en la figura 5.6.

La naturaleza híbrida de este estándar fue introducida para abarcar distintos tipos de usuarios con diferentes necesidades, requisitos y hábitos.

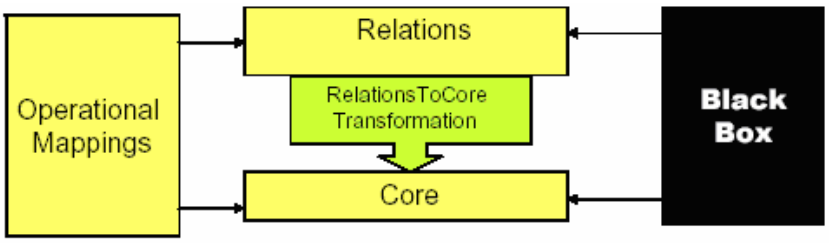

Figura 5.6: Relación entre los meta-modelos QVT

5.5.2. Lenguajes QVT

QVT declarativo

Relations Es un lenguaje amigable para el usuario, que soporta pattern matching complejo y la creación de templates para objetos. Tiene creación de trazas implícita. También incluye la propagación de cambios, ya que provee un mecanismo para identificar elementos del modelo destino.

Core Es un lenguaje pequeño con un soporte de pattern matching acotado. Tiene traceability explícito y las trazas pueden crearse y borrarse como cualquier otro objeto. Es igual de poderoso semánticamente que el lenguaje Relations pero trabaja a un nivel más bajo de abstracción. Esta propuesta absolutamente minimal lleva a que el lenguaje Core sea usado como un "assembler" de los lenguajes de transformación.

QVT imperativo

Operational Mappings Este lenguaje se especificó como una forma estándar para proveer implementaciones imperativas de transformaciones unidireccionales. También como el lenguaje Relations, dispone de creación de trazas implícita sobre el mismo modelo de trazas.

Implementaciones Black-box Estas implementaciones de caja negra, permite escribir transformaciones en otro lenguaje distinto a QVT. Una implementación de este tipo no tiene relación explicita con el lenguaje Relations, aunque una caja negra podría implementar una Relation, la misma debe ser responsable de mantener las trazas entre los elementos relacionados.

5.5.3. Más información

Para obtener más información introductoria consultar [25]. Para más detalles técnicos y ejemplos se puede leer el documento de especificación MOF Query/View/Transformation [24].

Capítulo 6

Trabajos relacionados

En este capítulo se realiza una breve introducción de un conjunto de trabajos que tienen alguna relación con el tema en cuestión de la presente tesis, encontrados a lo largo del aprendizaje y la investigación.

6.1. Un motor de traceability de transformación de modelos en la Ingeniería de Software

En [14] se describe un motor de traceability al que llamaron ETraceTool, que funciona como un plug-in de Eclipse programado mediante el paradigma orientado a aspectos con el fin de mantener aislada la generación de las trazas del código que pertenece a la transformación. El mismo trabaja sobre transformaciones escritas en Java usando la API EMF [15]. A continuación se listan sus principales características:

- El código de generación de trazas no es intrusivo en el código de la transformación;
- La generación de trazas tiene que ser activada explícitamente por el diseñador de la transformación;
- Los modelos de las trazas se encuentran aislados de los modelos origen y destino que forman parte de la transformación;
- Los modelos de las trazas pueden ser usados a diferentes niveles de granularidad.

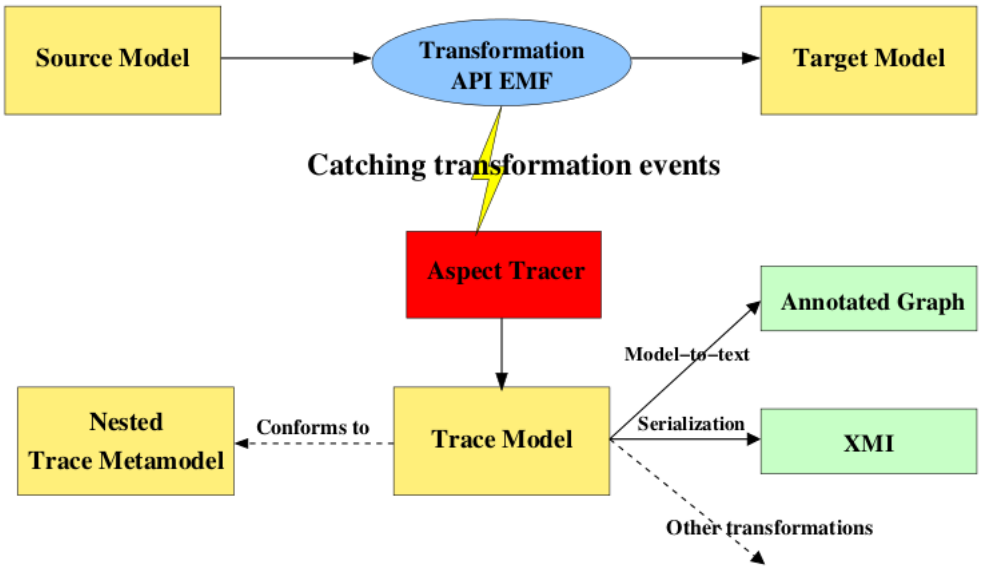

Figura 6.1: Arquitectura de la herramienta ETraceTool

La arquitectura se puede apreciar en el dibujo 6.1 y se explica de la siguiente manera, durante la transformación el plug-in captura un conjunto de eventos previamente identificados y clasificados gracias a la programación orientada a aspectos, luego el Aspect Tracer genera un modelo de trazas que conforma al meta-modelo de trazas anidado que se muestra en el dibujo 6.2. Al final, el modelo de trazas generado puede ser serializado en un archivo XMI o transformado a cualquier otro lenguaje mediante una transformación modelo-texto.

El fundamento del diseño del meta-modelo de trazas anidado propuesto es para el caso en el que se presente una operación de transformación que llama o hace uso de otra transformación. En este caso el enlace compuesto permite separar las operaciones de bajo nivel (creación, eliminación, etc) de las operaciones de alto nivel (como una operación de refactorización).

6.2. Un Framework de Traceability dirigido por modelos para el desarrollo de Software Product Line (SPL)

El framework presentado en [16] provee una plataforma abierta y flexible para crear enlaces de trazas entre distintos artefactos del desarrollo de una Línea de Producto de Software o Software Product Line (SPL). Pero,

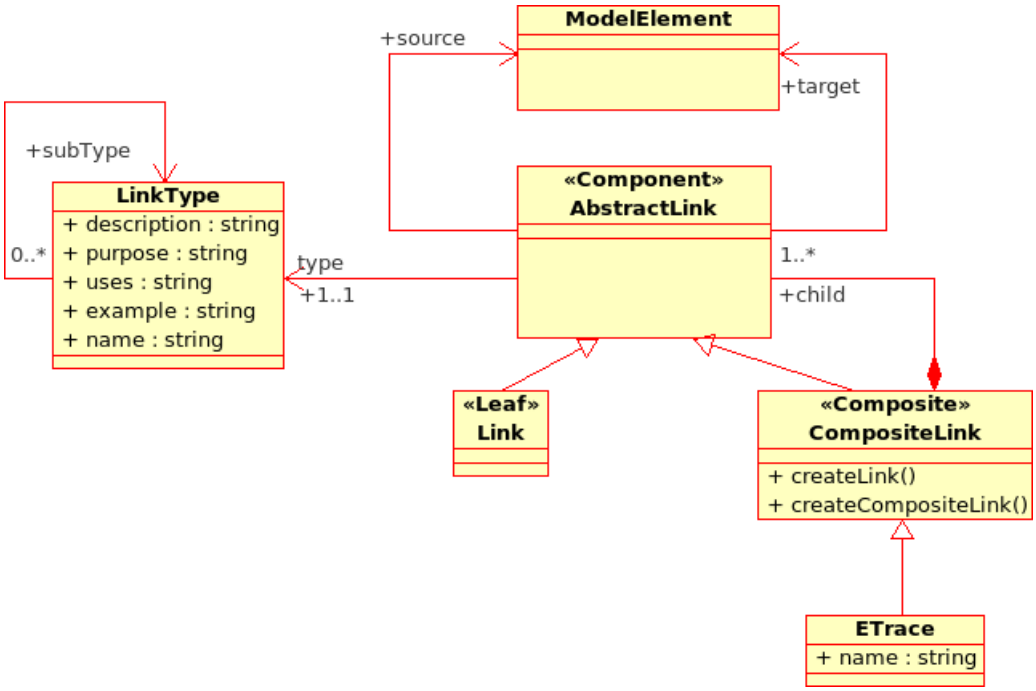

Figura 6.2: Meta-modelo de trazas anidado

dado que el diseño del framework es genérico, éste también puede aplicarse o usarse en otras áreas del desarrollo SPL. El mismo ha sido diseñado e implementado basado en el uso de técnicas dirigidas por modelos. El meta-modelo de traceability descripto en el dibujo 6.3 permite definir distintos tipos de enlaces de trazas entre los artefactos.

Las principales funcionalidades ofrecidas por el framework son las siguientes:

1. Creación y mantenimiento de los enlaces de trazas de artefactos existentes (modelos UML, código fuente, etc);

2. Almacenamiento de los enlaces de trazas mediante el uso de un repositorio;

3. Búsqueda de enlaces de trazas específicos usando consultas de trazas predefinidas o personalizadas;

4. Visualización flexible de los resultados de las consultas de trazas por medio de diferentes tipos de vistas, como vista de árbol, grafo, tabla, etc.

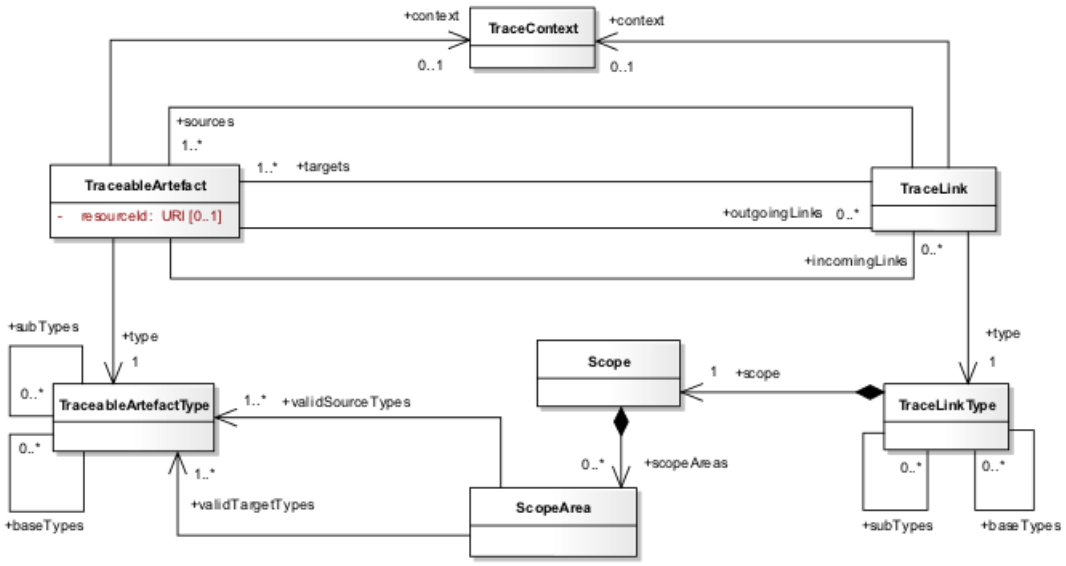

Figura 6.3: Meta-modelo de traceability

6.2.1. Meta-modelo de traceability

Los elementos principales del meta-modelo son los siguientes:

- Un TraceableArtefact representa un artefacto que juega un rol en el ciclo del desarrollo. La granularidad del artefacto es arbitraria, puede ser un requerimiento, un diagrama UML, un elemento de dicho diagrama, una clase o un método de dicha clase. El artefacto es identificado mediante la propiedad resourceId.
- Un TraceLink es la abstracción de una transición de un artefacto a otro.
- Cada TraceableArtefact tiene asignada una instancia de TraceableArtefactType, éstos se pueden encontrar agrupados de forma jerárquica.
- Análogo a los tipos de los artefactos los TraceLinks también tienen un tipo (TraceLinkType), teniendo en cuenta que la semántica de una relación entre dos artefactos puede variar.
- Información adicional de artefactos y enlaces puede ser modelada mediante un contexto, el cual se encuentra representado por la clase TraceContext.
- Las restricciones que determinan el conjunto de artefactos válidos, sobre los cuales los tipos de enlaces también son válidos se encuentran modeladas mediante los elementos ScopeArea y Scope.

6.2.2. Arquitectura

Como se muestra en el dibujo 6.4, la arquitectura ha sido definida en término de cuatro módulos principales. Cada uno de los cuales implementa una de las funcionalidades principales del framework, se detallan a continuación:

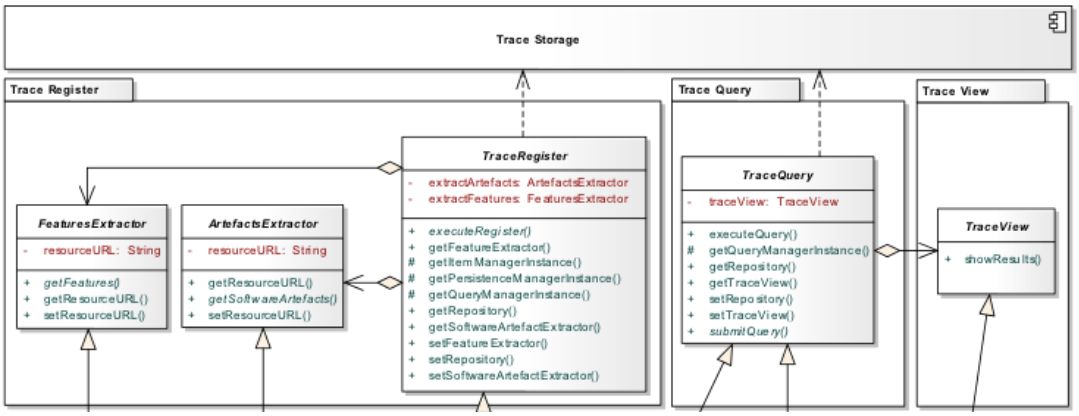

Figura 6.4: Arquitectura del Framework de Traceability

1. Trace Register: este módulo provee mecanismos para crear y mantener (actualizar, eliminar y buscar) los enlaces de trazas;

2. Trace Storage: define los mecanismos de almacenamiento para persistir dichos enlaces;

3. Trace Query: este modulo permite crear y ejecutar consultas para buscar enlaces de trazas específicos que se encuentran previamente almacenados;

4. Trace View: usado específicamente para la representación visual de los enlaces resultados de una consulta realizada.

6.3. Integración de herramientas Case

En [11] se presenta el problema real que sufre cualquier proceso de desarrollo actual, en el que como resultado del conjunto de actividades que lo conforman, se van generando una variedad muy amplia de artefactos de software (documentos de textos, hojas de cálculo, resultado de pruebas, modelos, gráficos, etc). Estos artefactos, aunque en esencia se encuentran relacionados

lógicamente, al ser creados y manipulados por herramientas muy distintas que no fueron pensadas para interactuar (editores de textos, editores de modelo UML, etc), las relaciones lógicas nombradas se pierden, o mejor dicho no existen o pasan desapercibidas en la práctica. En otras palabras, nos presenta el problema de la imposibilidad de traceability que encontramos entre la mayoría de las herramientas Computer Aided Software Engineering (CASE) actuales.

Como solución a este problema, se propone un ambiente de integración para las herramientas CASE al que llamaron TiE - Tool Integration Environment, el cual basa su integración en la creación de enlaces de trazas entre los artefactos de las distintas herramientas.

6.4. Framework genérico de extracción de datos de traceability

En [13] proponen un framework genérico de traceability que toma como entrada una transformación de modelo y le aumenta arbitrariamente su funcionalidad con un mecanismo de traceability. En el dibujo 6.5 se muestra un panorama de alto nivel de la arquitectura propuesta. Ésta se basa en una interfase genérica que provee un punto de conexión para cualquier motor de transformación de modelos, mediante una API que se ofrece al ingeniero que conecta su motor de transformación con el motor de traceability (oAW connector, QVT connector). Como resultado el motor de transformación incluye la funcionalidad de traceability. El modelo de datos que usa el framework es el lenguaje específico de dominio para traceability al que llaman Trace-DSL.

Trace-DSL que se detalla en el dibujo 6.6, tiene como elemento raíz TraceModel. Artefact representa cualquier producto traceable generado durante el proceso de desarrollo, esto puede ser un requerimiento o una clase o un componente, como el método que pertenece a una clase. Todo artefacto es identificado unívocamente mediante un identificador único universal (UUID). TraceLink es una abstracción de una transición de un artefacto a otro dirigida por la relación desde-hacia entre artefactos origen y destino. TraceLink puede ser de uno de las siguientes cuatro instancias: CreateTraceLink, QueryTraceLink, UpdateTraceLink y DeleteTraceLink.

Para la asignación de tipos a los artefactos y a los enlaces se usa el concepto de faceta, donde Trace-DSL asigna un conjunto de facetas (Facet) a cada uno de los mismos. Un ejemplo de faceta se da en el dibujo 6.7. Además de lograr una solución simple al tipado de artefactos y enlaces, se obtiene un mecanismo fácilmente extensible y configurable al contexto donde se necesite

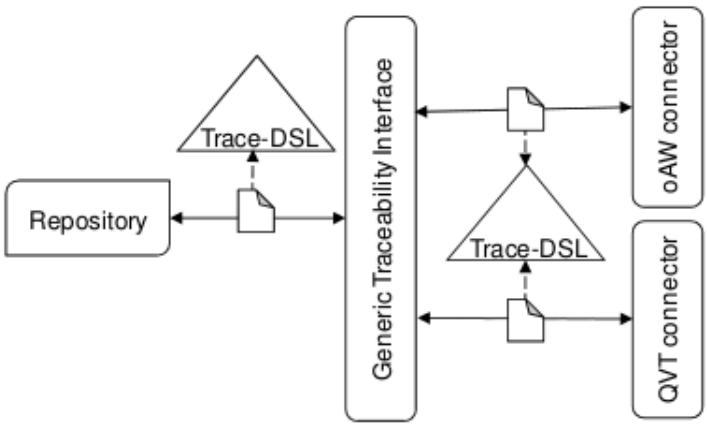

Figura 6.5: Resumen de la arquitectura del Framework Genérico de Traceability

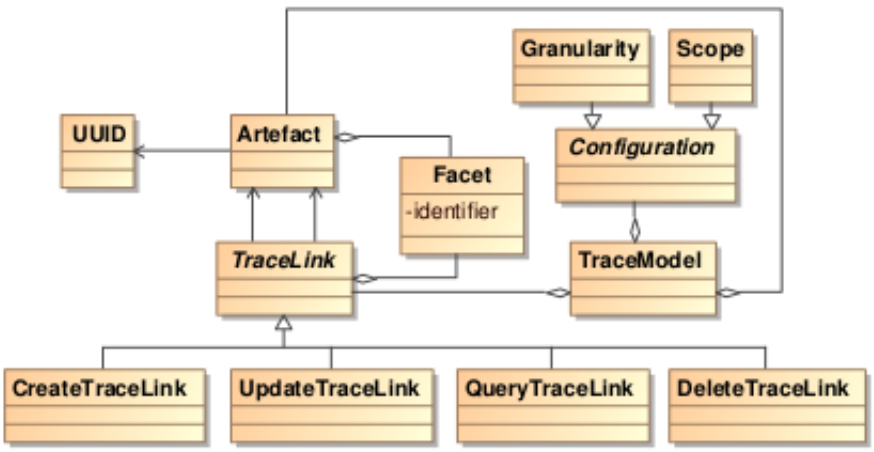

Figura 6.6: Lenguaje específico de dominio para traceability

aplicar traceability.

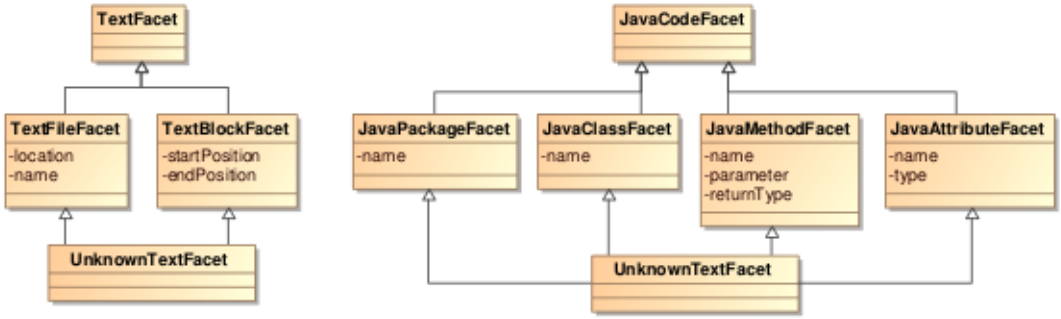

Figura 6.7: Faceta para traceability de código fuente

La configuración necesaria para hacer uso del framework implica:

1. Seleccionar las facetas requeridas para el escenario
2. Configurar la granularidad (Granularity) y el alcance (Scope)

La configuración de la granularidad consiste en la especificación de qué tipos (definidos por las facetas) de artefactos y enlaces serán trazados para un escenario de traceability particular. En cambio la configuración del alcance implica restringir los datos de traceability a una combinación específica de valores. En otras palabras la primera solo chequea la existencia de facetas, mientras que la segunda adicionalmente examina las propiedades especificas de las facetas. Por ejemplo en el caso de TextFileFacet, puede ser necesario trazar solo archivos de textos con cierto nombre.

6.5. Traceability local y global

En la propuesta presentada en [8] usan la idea de separación del proceso de traceability en los siguientes niveles, traceability en lo pequeño y traceability en lo grande refiriéndose a los mismos como traceability local y traceability global respectivamente.

6.5.1. Meta-modelo de Traceability Local

Este meta-modelo toma las trazas de la entrada y la salida de una única transformación. El meta-modelo está basado en el meta-modelo de trazas presentado en [9] y se muestra en el dibujo 6.8.

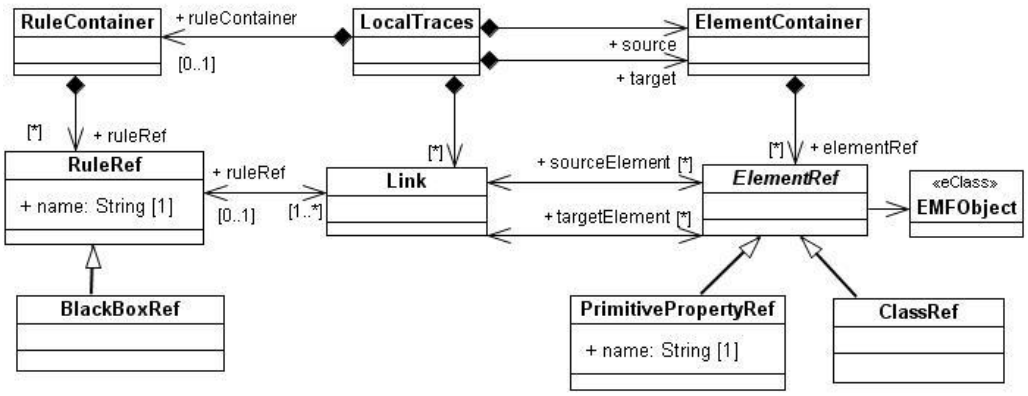

Figura 6.8: Meta-modelo de Trazas Local

El meta-modelo de trazas local contiene dos conceptos principales Link y ElementRef en donde se expresa que uno o más elementos orígenes son enlazados a uno o más elementos objetivos. ElementRef es una clase abstracta que representa elementos que pueden ser traceados: instancias de clases o valores de propiedades. Los valores de las propiedades son traceados usando PrimitivePropertyRef el cual apunta a la instancia contenedora de la propiedad y tiene el nombre de la misma. Este tratamiento especial para los tipos primitivos de Java se debe a que no existen instancias de ellos en el modelo. Por otro lado las propiedades que son tipadas mediante una clase normal, son traceadas mediante ClassRef.

Para almacenar la información sobre las reglas de transformación aplicadas así como el caso particular de las cajas negras, se hace uso de los conceptos RuleRef y BlackBoxRef. En ambos casos, dado que pueden dar como resultado de su ejecución varios enlaces, se define la relación como uno a varios entre RuleRef y Link. RuleRef y BlackBoxRef son opcionales, en el caso de la primera solo se usa para realizar una depuración de las transformaciones, y la segunda si nos encontramos en la situación en la que ciertas partes del sistema no pueden ser pública su implementación.

ElementRef tiene una referencia al objeto real de los modelos origen y destinos. Como estos modelos están implementados mediante EMF, la referencia EMFObject es un EObject del meta-modelo Ecore. La clase LocalTraces representa la raiz del modelo de trazas local y tiene un RuleContainer que se usa como contenedor de las reglas y dos ElementContainer usados para agrupar los ElementRef origen y destino respectivamente. Separar los elemento orígenes y destinos permite reducir los costos de búsquedas de elementos de entrada o salida.

6.5.2. Meta-modelo de Traceability Global

Este meta-modelo enlaza trazas locales de acuerdo a la cadena que define la transformación. Un modelo de trazas global es el punto de entrada principal en el cual todas los modelos de trazas locales se encuentran, y describe qué modelo origen/objetivo de una transformación es el modelo objetivo/origen de la siguiente/previa transformación. En el dibujo 6.9 se puede observar el meta-modelo.

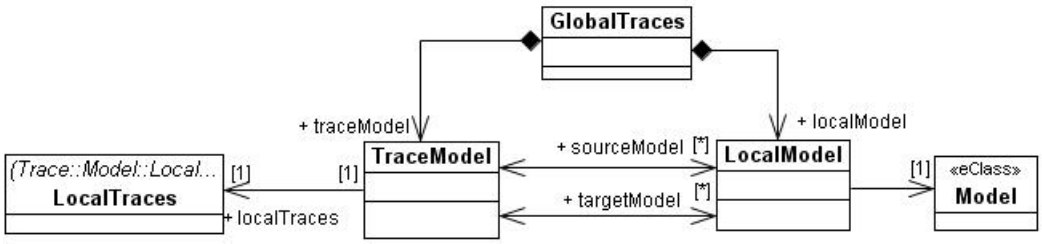

Figura 6.9: Meta-modelo de Trazas Global

Este meta-modelo se engloban todas las trazas locales, los modelos de una cadena de transformación y la forma en que éstos se encuentran enlazados mediante TraceModel y LocalModel. Los modelos pueden ser compartidos entre distintas transformaciones, es decir uno puede ser producto de una transformación y también ser consumido por otra transformación.

Introducir este nivel global de trazas permite la navegación entre los modelos transformados y sus modelos de trazas locales, dando una mejor separación de lo que es en verdad la compleja información de traceability, lo que permite además una mejor flexibilidad para la creación de las trazas y la explotación de las mismas. No utilizar esta idea de trazas globales tiene como consecuencia disponer de todas las trazas en un único modelo de toda una cadena de transformación, lo cual desencadenaría en el colapso para la creación y la consulta de dicho modelo.

6.5.3. ¿Cómo trabaja el framework?

Una de los principales objetivos de recolectar las trazas es dar luego la posibilidad a un usuario de inspeccionarlas realizando distintas consultas, una puede ser por ejemplo obtener los elementos relacionados a uno seleccionado. Este meta-modelo permite desde el modelo de trazas global navegar hacia los modelos de trazas locales y/o hacia a los modelos envueltos en cada una de las transformaciones. También desde el modelo de trazas local se puede navegar entre los elementos del modelo parte de la transformación.

En el dibujo 6.10 se muestra un ejemplo de un modelo de trazas locales y globales producto de una cadena de transformación. En el ejemplo se representan los enlaces (Link) entre los elementos sin tener en cuenta las instancias de RuleRef para no sobrecargarlo.

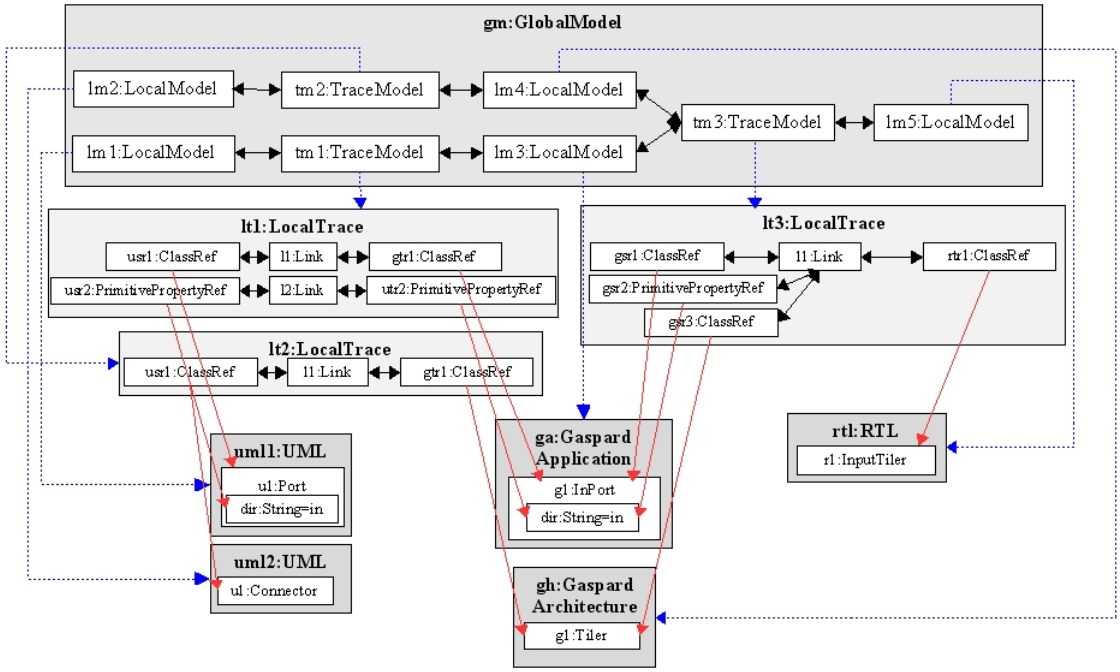

Figura 6.10: Ejemplo de un modelo de trazas local y global

Conclusión

Glosario

hola Esto es una prueba. v

Siglas

ALM Application Lifecycle Management. 28

API Interfaz de programación de aplicaciones o Application Programming Interface. 28, 29, 33, 40, 45

ATL Atlas Transformation Language. 36–38

CASE Computer Aided Software Engineering. 44, 45

EMF Eclipse Modeling Framework. 17, 18, 30, 31, 33, 40, 48

EPL Eclipse Public License. 28

EVL Epsilon Validation Language. 18

GEF Graphical Editing Framework. 33

GMF Graphical Modeling Framework. 33–36

GUI Interfaz de Usuario Gráfica o Graphical User Interface. 31

IDE Entornos de Desarrollo Integrados o Integrated Development Environment. 7, 13, 28, 36

IEEE Institute of Electrical and Electronics Engineers. 1

JDT Java Development Tooling. 31

MDA Arquitectura Dirigida por Modelos o Model-Driven Architecture. VI

MDD Desarrollo Dirigido por Modelos o Model-Driven Development. VI, 1

MDE Ingeniería de Software Dirigida por Modelos o Model-Driven Engineering. VI, VII, 16, 19, 20, 36

MMT Model-to-Model Transformation. 37, 38

MOF MetaObject Facility. 17, 36, 39

OCL Object Constraint Language. 18

OMG Object Management Group. VI, 38

QVT Query/View/Transformation. 38, 39, 45

RCP Rich Client Platform. 27

RIA Rich Internet Applications. 28

SOA Service Oriented Architecture. 28

SPL Línea de Producto de Software o Software Product Line. 41

SWT Standard Widget Toolkit. 29

TEAP Traceability Elicitation and Analysis Process. 19

UML Unified Modeling Language. 17, 20, 33, 42–44

XMI XML Metadata Interchange. 30, 31, 33, 40

Bibliografía

[1] IEEE Standard Glossary of Software Engineering Terminology. Number Std 610.12-1990, IEEE (1990).

[2] R. Brcina and M. Riebisch: Defining a Traceability Link Semantics for Design Decision Support. In: ECMDA Traceability Workshop (ECMDA-TW) 2008 Proceedings.

[3] B. Grammel and K. Voigt: Foundations for a Generic Traceability Framework in Model-Driven Software Engineering. In: ECMDA Traceability Workshop (ECMDA-TW) 2009 Proceedings.

[4] Glossary of Center of Excellence for Software Traceability (CoEST) http://www.coest.org/index.php/traceability/glossary.

[5] Center of Excellence for Traceability - Problem Statements and Grand Challenges. In: Center of Excellence of Traceability Technical Report (COET-GCT-06-01-0.9) September 10, 2006.

[6] Gotel, O.C.Z., Finkelstein, A.C.W., "An Analysis of the Requirements Traceability Problem", International Conference on Requirements Engineering, ICRE'94, Los Alamitos, California, Abril, 1994, pp 94-101.

[7] N. Drivalos, R. F. Paige, K. J. Fernandes, D. S. Kolovos: Towards Rigorously Defined Model-to-Model Traceability. In: ECMDA Traceability Workshop (ECMDA-TW) 2008 Proceedings.

[8] F. Glitia, A. Etien and C. Dumoulin: Fine Grained Traceability for an MDE Approach of Embedded System Conception. In: ECMDA Traceability Workshop (ECMDA-TW) 2008 Proceedings.

[9] F. Jouault: Loosely Coupled Traceability for ATL, In: Proceedings of the European Conference on MDA Traceability Workshop, Nurnberg, Germany (2005).

[10] R. Paige, G. Olsen, D. Kolovos, S. Zschaler, C. Power: Building Model-Driven Engineering Traceability Classifications, In: ECMDA Traceability Workshop (ECMDA-TW) 2008 Proceedings.

[11] F. Klar, S. Rose, A. Schurr: TiE - A Tool Integration Environment, In: ECMDA Traceability Workshop (ECMDA-TW) 2009 Proceedings.

[12] S. B. Abid, G. Botterweck: Resolving Product Derivation Tasks using Traceability in Software Product Lines, en: ECMDA Traceability Workshop (ECMDA-TW) 2009 Proceedings.

[13] B. Grammel, S. Kastenholz: A Generic Traceability Framework for Facet-based Traceability Data Extraction in Model-driven Software Development, en: Proceedings of the 6°ECMFA Traceability Workshop (ECMFA-TW), 15 de junio de 2010, Paris, Francia.

[14] B. Amar, H. Leblanc, B. Coulette: A Traceability Engine Dedicated to Model Transformation for Software Engineering. In: ECMDA Traceability Workshop (ECMDA-TW) 2008 Proceedings.

[15] Eclipse Modeling Framework Project (EMF) http://www.eclipse.org/modeling/emf/.

[16] A. Sousa, U. Kulesza, A. Rummler, N. Anquetil, R. Mitschke, A. Moreira, V. Amaral, J. Araújo: A Model-Driven Traceability Framework to Software Product Line Development. In: ECMDA Traceability Workshop (ECMDA-TW) 2008 Proceedings.

[17] Eclipse Project http://www.eclipse.org.

[18] Eclipse Platform Technical Overview. Object Technology International, Inc., February 2003, http://www.eclipse.org/whitepapers/eclipse-overview.pdf.

[19] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, T. J. Grose: Eclipse Modeling Framework - A Developer's Guide. Addison Wesley.

[20] Graphical Modeling Project http://www.eclipse.org/modeling/gmp/.

[21] Frederic Plante, IBM: Introducing the GMF Runtime, January 2006 http://www.eclipse.org/articles/Article-Introducing-GMF/article.html

[22] Graphical Modeling Framework - Tutorial - Part 1 http://wiki.eclipse.org/Graphical_Modeling_Framework/Tutorial/Part_1

[23] ATL Eclipse Project http://www.eclipse.org/atl/.

[24] Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Version 1.1, January 2011 - Version 1.0, April 2008. http://www.omg.org/spec/QVT/index.htm.

[25] C. Pons, R. Giandini y G. Pérez: Desarrollo de Software Dirigido por Modelos. Conceptos teóricos y su aplicación práctica. Facultad de Informática, Universidad Nacional de La Plata. Octubre de 2008.