Tesis

Mariano Gabriel Gili

25 de septiembre de 2012

Índice general

Índice de figuras

Agradecimientos

Agradezco a todos...

Introducción

La Ingeniería de Software Dirigida por Modelos (MDE - Model-Driven Engineering) define al modelo como artefacto principal a lo largo de todas las tareas de la ingeniería de software – desarrollo, mantenimiento, etc. Un caso particular de ingeniería que sigue esta idea es la Arquitectura Dirigida por Modelos (MDA - Model Driven Architecture), definida por el Object Management Group (OMG), cuyo ciclo de vida del proceso de desarrollo propuesto 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 (MDD – Model-Driven Development) es lo que se conoce como "posibilidad de rastreo" (o de ahora en más en inglés traceability), que ayuda o 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 y ésto 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, el análisis de los problemas de trazabilidad que aun se encuentran abiertos en el ámbito de MDE, seguido de la elaboración de un esquema de trazabilidad que contribuya a la solución de dichos problemas. 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

Según el Glosario Estándar de Términos de la Ingeniería de Software del 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 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, entre los requerimientos y su representación en los modelos, traceability es crucial 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 del 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 manual como una relación de implementación entre un requerimiento y un componente de software. En la ingeniería de software encontramos dos usos principales (o semánticas) que dependen del contexto de traceability:

- Traceability en la ingeniería de requerimientos: donde se guarda un requerimiento desde su expresión 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. Una clasificación genérica de traceability

Conforme un proyecto de desarrollo crece, la administración de la información de traceability que se va generando se vuelve extramadamente compleja, una jerarquía de clasificación de esta información de traceability resulta 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 clasificación comienza con una simple clasificación o meta-modelo de traceability inicial, luego de forma incremental e iterativa se refina la clasificación siguiendo las siguientes tareas: obtención, análisis y clasificación. En la obtención se identifica un trace link y sus relaciones. En el análisis, se abstraen las principales características del trace link obtenido identificando restricciones, relaciones y generalizaciones. Por último, se construye la clasificación.

El meta-modelo inicial propuesto se muestra en el dibujo 1.1 en el que se encuentran los conceptos fundamentales de artefacts, trace links y operations. Artefact refiere tanto a artefactos MDE (ejemplo modelos específicos de dominio) como a no MDE (ejemplo hojas de cálculo), Operation encierra las operaciones bien manuales como automáticas que determina qué información de traceability debe almacenarse, y por último Trace Link engloba la clasificación que comienza con la idea de traceability implícita (Implicit Link) y explícita (Explicit Link). En este contexto, traceability implícita refiere a trace links que son creados y manipulados por la aplicación de operaciones MDE, en cambio explícita refiere a trace links que se encuentran ya concretamente representados en los modelos.

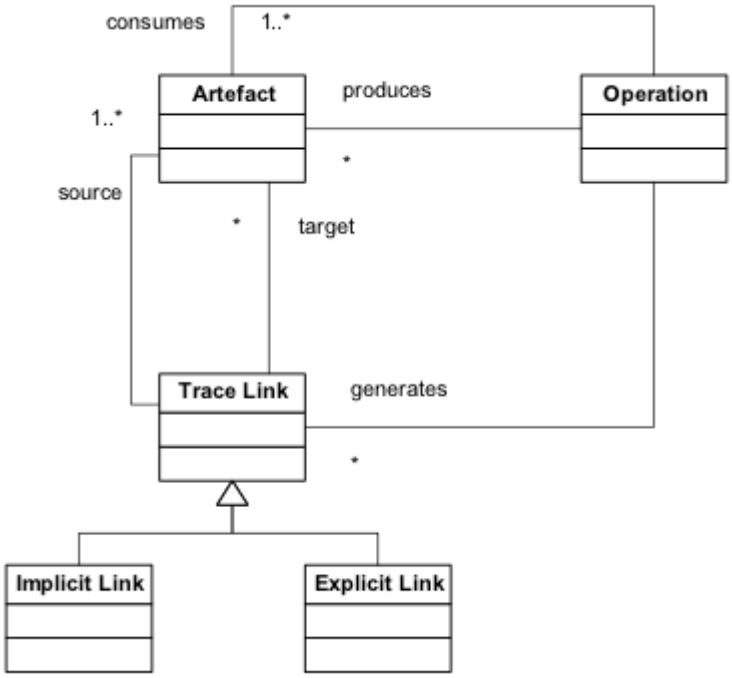

Figura 1.1: Clasificación de traceability inicial

1.1.1. Clasificación de trace links implícita

En esta jerarquía que se muestra en el dibujo 1.2 se presentan un 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.

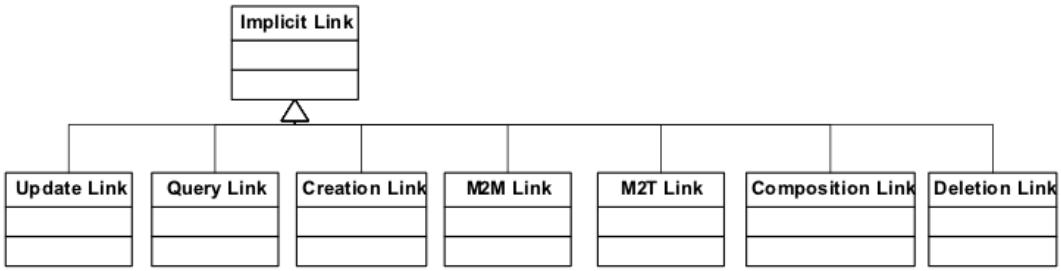

Figura 1.2: Jerarquía de links implícitos

1.1.2. Clasificación de trace links explícita

En esta jerarquía se clasifican los trace links explícitos, por ejemplo una dependencia UML es uno de ellos. En principio se identifican dos clases básicas: links modelo-modelo (ejemplo dependencias UML) y links modelo-artefacto (entre un modelo y un documento de texto que dicta un requerimiento), que en el dibujo 1.3 aparecen como Model-Model Link y Model-Artefact Link.

Model-Model Link es dividido en links estáticos (Static Link) y dinámicos (Dynamic Link). Los primeros representan relaciones estructurales que no cambian con el tiempo, los segundos representan información de los modelos que pueden variar a través del tiempo. Ejemplos de links estáticos, links consistentes (Consistent-With) donde dos modelos deben mantenerse de acuerdo o consistentes entre si, y links de dependencias (Dependency) donde la estructura y/o la semántica de un modelo depende de otro. Los links de dependencia pueden ser divididos: links de relación de subtipo (Is-A), links de referencias, links de subconjuntos, de importación y exportación, de uso, de refinamiento (Refines), etcétera. Ejemplos de links dinámicos incluyen links de llamadas (Calls) donde un modelo hace uso de métodos provisto por otro, links de notificación (Notifies) donde es necesario almacenar información que puede ser manejada automáticamente. También relaciones en tiempo de diseño como links de generación o construcción (Generates) que indica qué información de un modelo es usada para producir o deducir otro, y relaciones de sincronización (Synchronized With) donde el comportamiento de un conjunto de modelos es sincronizado.

El alcance de Model-Artefact Link es muy amplio, es esta clasificación solo se resumen un subconjunto de ejemplos, entre ellos: la relación satisface (Satisfies) que indican que propiedad o requerimiento de un artefacto es satisfecha por un modelo; 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 que 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 es explicado/respaldado por un artefacto no modelo.

1.2. Beneficios y problemas de traceability

1.2.1. Beneficios

A continuación se listan algunos de los beneficios del uso de relaciones o enlaces de traceability en las siguientes actividades o dominios de la ingeniería

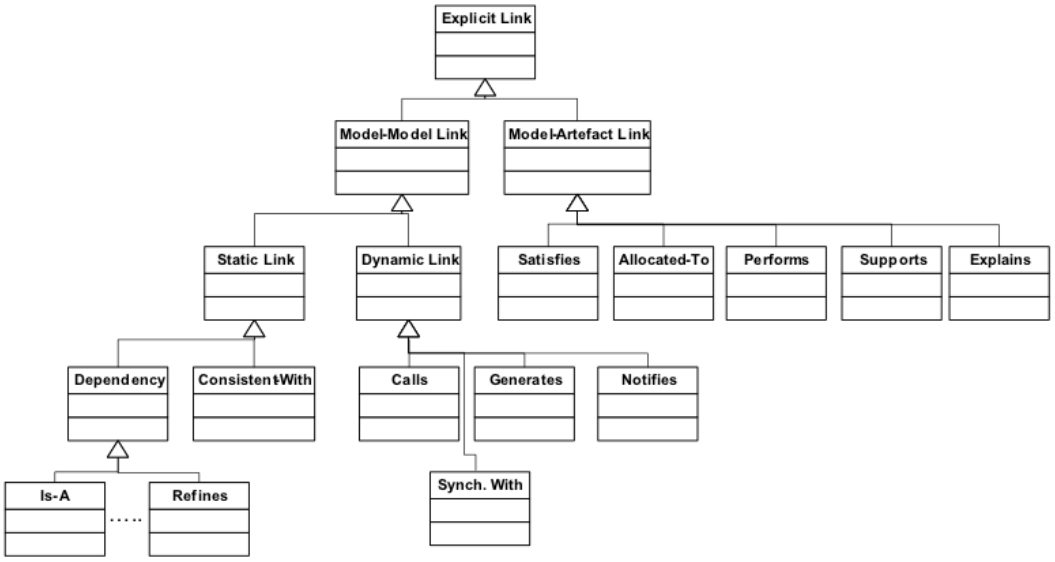

Figura 1.3: Jerarquía de links explícitos

según [2] y [3]:

- Análisis de sistemas: para entender la complejidad del mismo navegando los enlaces a lo largo del modelo de las cadenas de transformación.
- Análisis de cobertura: para determinar si todos los requerimientos fueron cubiertos en los casos de prueba del ciclo de vida del desarrollo, donde la traceability de los respectivos modelos es crucial.
- Análisis de impactos de un cambio: para ver cómo los cambios en un modelo afectarán los otros relacionados; saber el tipo de dependencia que existe entre las entidades relacionadas ayudará a determinar el cambio necesario.
- Análisis de huérfanos: para encontrar los elementos huérfanos de un modelo con respecto a un enlace específico.
- Comprensión del software e ingeniería inversa: para 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.
- Origen de una desición o requerimiento: para identificar el artefacto particular que demanda una propiedad específica; justificar una decisión; resolver un conjunto de requerimientos que se contradicen.

- Apoyo en la toma de decisiones: para entender qué factores y metas influyen en una decisión; para ayudar en las propuesta de soluciones y en la evaluación de las mismas.
- Configuración del sistema y versionado: 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.

1.2.2. Problemas

Aunque las ventajas de traceability ya han sido identificadas, su puesta en práctica apenas ha quedado establecida. Las principales razones son [3]:

- 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 usuarios de traceability, quienes crean los enlaces y quienes los usan.
- Carencia de soporte adecuado en las herramientas.
- Los artefactos son escritos en diferentes lenguajes, lenguaje natural versus lenguaje de programación.
- Los artefactos describen el sistema de software en diferentes niveles de abstracción (diseño versus implementación).

1.3. Desafíos de traceability

Vamos a describir los principales problemas y grandes desafíos relacionados a varios aspectos de traceability en sistemas de software según [5] de la siguiente forma, agrupando por tema los problemas seguido de los grandes desafíos para dichos problemas:

1.3.1. Sobre el conocimiento de traceability

+ Existe poco consenso respecto a cuáles son las mejores técnicas y métodos de traceability, pocas anotaciones o documentación sobre las mejores

prácticas, y una falta de recursos que provean una buena base de conocimiento.

+ Definiciones semánticas que no coinciden y uso de terminologías dispares o distintas crean barreras de comunicación.

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

1.3.2. Sobre la capacitación y la certificación

+ Muy poca gente es competente en tracing y por otro lado existen disponibles pocos programas educativos.

+ Hay pocos programas de certificación, y pocos de ellos incluyen componentes de traceability.

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

- Identificar áreas de conocimientos centrales y habilidades o estrategias asociadas a traceability.

- Desarrollar componentes educativos buenos para la práctica de traceability.

- Desarrollar materiales pedagógicos efectivos para educar en la importancia y en la administración de costos-beneficios de traceability.

1.3.3. Sobre el soporte a la evolución

+ La información precisa, coherente, completa y actualizada sobre traceability es fundamental para diversos ámbitos y aplicaciones. Sin embargo, las técnicas actuales de "link recovery" son aún 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 links 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 sus artefactos relacionados, sin embargo los sistemas actuales de gestión del cambios y la semántica de los enlaces no son lo suficientemente sofisticadas para apoyar esta evolución.

+ Los métodos para transformar y reusar los "trace links" sincrónicamente con los productos de desarrollo en línea son inmaduros.

- Desarrollar técnicas de "link recovery" para artefactos textuales que sean tan precisos como el proceso manual y mucho más efectivos en tiempo y costo.

- Desarrollar "traceability recovery" con el fin de ser integrado en los "IDE".

- Desarrollar sistemas de administración de cambios que efectivamente soporten la evolución de los "traceability links" sobre múltiples tipos de artefactos.

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

- Desarrollar técnicas para maximizar la reutilización de los "traceability links" cuando el código existente se reutiliza en un nuevo producto.

1.3.4. Sobre la semántica de los enlaces

+ Para efectivamente utilizar links y entender las relaciones por debajo de traceability, es necesario definir la semántica de los links, 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 la correcta granularidad o "trace granularity".

- Definir meta-modelos para representar información semántica de los "traceability links" y proveer ejemplos de instanciación para dominios específicos.

- Desarrollar técnicas y procesos para determinar la correcta granularidad de links en un proyecto.

1.3.5. Sobre la escalabilidad

+ 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 gran cantidad de información de "trace links". 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 "trace".

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

- Desarrollar mecanismos visuales efectivos para soportar navegación y consulta de un gran número de "traceability links" y sus artefactos asociados.

- Desarrollar técnicas escalables para "tracing" tanto para conjunto de datos heterogéneos y/o grandes y débilmente estructurados.

1.3.6. Sobre los factores humanos

+ Los métodos automáticos de traceability producen "trace links" candidatos; sin embargo, el proceso es inútil si el analista no es capaz de evaluar correctamente los links 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 "tracing" 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 "trace links" comunican artefactos semánticamente diferentes, pero éstos artefactos son creados por diferentes personas, frecuentemente escritos en diferentes documentos. Como resultado, los usuarios de un lado de los "trace links" no entienden bien los artefactos del otro lado relacionado.

- Basado en el estudio del uso de herramientas de traceability, crear nuevas herramientas de traceability que reunan las necesidades prácticas que salgan a la luz.

- 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 éstos cuando ocurran.

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

1.3.7. Análisis de costo-beneficio

+ En una traceability completa, links 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 modelo de costo-beneficio para diferenciar entre diferentes necesidades de traceability sobre varios proyectos y para diferentes links potenciales dentro de un proyecto.

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

- Definir un modelo de costo práctico para generar y mantener "trace links" que tomen en consideración factores tales como el tamaño del proyecto, tiempo, esfuerzo y calidad del sistema.

- Definir un modelo de beneficios para usar "trace links" que tomen en consideración factores como crítica y volatilidad, e incorpore el valor logrado desde el uso de las traces.

1.3.8. Sobre los métodos y las herramientas

+ 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.

+ Trace automático es esencial; sim embargo, se hace difícil por la escases de consistencia entre los artefactos, y la impresición de los modelos.

+ Traceability implica las actividades de construcción o generación, evaluación, mantenimiento y uso de links; sin embargo, no hay una sola herramienta que pueda cubrir todas estas tareas.

- Desarrollar métodos efectivos para "trace" artefactos multimedia.

- Construir métodos y herramientas con altos niveles de automatización para soportar el ciclo de vida entero que incluya la construcción, evaluación, mantenimiento y uso de los "trace links".

- Desarrollar métodos para trazar requerimientos no funcionales.

1.3.9. Sobre los procesos

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

+ El "tracing" automático puede proveer una alternativa efectiva en costo al manual, pero la práctica ha mostrado que algunos conjuntos de datos son difíciles de "trace" usando métodos automáticos debido a las inconsistencias en terminología, estándares, carencia de estructura, formatos heterogéneos, etc.

- Construir modelos de proceso que definan el ciclo de vida del "tracing".

- Desarrollar técnicas para evaluar la habilidad de un conjunto de datos dado para soportar los métodos automáticos de "tracing".

1.3.10. Sobre el cumplimiento

+ 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 éstos.

+ La comunidad de traceability son eruditos sobre técnicas y procesos de traceability, pero tienen poca influencia sobre los contenidos relacionados de traceability en los procesos estándar de ingeniería de software.

+ No es claro cómo el cumplimento de los estándares y regulaciones se puede demostrar.

- Establecer un mecanismo de comunicación para hacer que la comunidad conocedora de traceability de los estándares relacionados con traceability.

- 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, técnicas y metodologías de traceability cumplen con estándares.

1.3.11. Sobre las mediciones y los Benchmarks

+ Los estudios empíricos son necesarios para demostrar la eficacia de los métodos de traceability y 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 o no son compatibles.

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

+ La detección errores en los "traceability links" es necesaria para determinar la eficacia del producto y proceso, sin embargo los modelos de detección de errores actuales en traceability son primitivos y no válidos.

- Definir procesos estándares y procedimientos relacionados para realizar estudios empíricos durante la investigación de traceability.

- Construir pruebas (benchmarks) para evaluar métodos y técnicas de traceability.

- Definir medidas para evaluar la calidad tanto individual como la de un conjunto de "traceability links".

- Desarrollar técnicas para evaluar métodos y procesos de traceability.

1.3.12. Sobre la transferencia de tecnología

+ El objetivo de la investigación de traceability es la transferencia de soluciones eficaces de traceability para la industria, sin embargo en la industria son reacios a probar nuevas y no demostradas técnicas.

+ 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.

- Crear una infraestructura y métodos relacionados para organizar el proceso de transferencia de tecnología.

- Identificar los casos de estudio exitosos y publicitarlos para demostrar la efectividad de los costos y las técnicas de traceability para la industria.

- Identificar los usuarios de traceability y definir sus necesidades en términos de calidad, ciclo de vida, grupos de usuarios, 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 2

Trabajos relacionados

2.1. Un motor de traceability de transformación de modelos en Ingeniería de Software

En [14] describen un motor de traceability al que llaman 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 de la transformación. El mismo trabaja sobre transformaciones escritas en Java usando la API EMF [15] y a continuación se listan las principales características:

- El código de generación de trazas no es intrusivo en el código de la transformación;
- La generación de trazas es activada explícitamente por el diseñador de la transformación;
- Los modelos de las trazas se encuentran aisladas o aparte 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.

La arquitectura se presenta en el dibujo 2.1 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 o ajusta al meta-modelo de trazas anidado que se muestra en el dibujo 2.2.

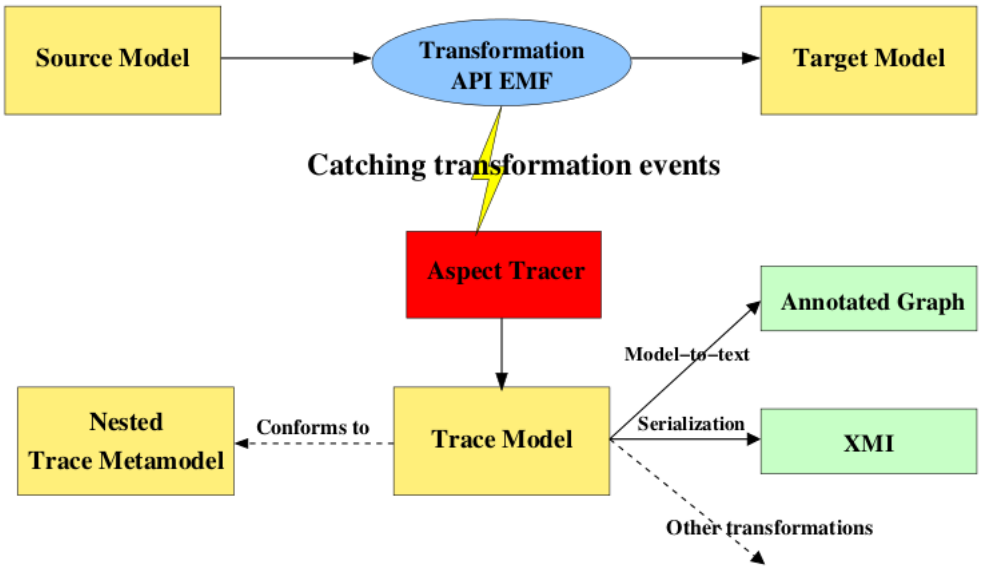

Figura 2.1: Arquitectura de la herramienta ETraceTool

Al final, el modelo de trazas generado puede ser serializado en un archivo XMI o transformado a cualquier otra lenguaje mediante una transformación modelo-texto.

El fundamento para el diseño del meta-modelo de trazas anidado propuesto es dado 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).

2.2. 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 SPL, aunque dado que el diseño del framework es genérico, éste también puede aplicarse o usarse fuera 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 2.3 permite definir distintos tipos de enlaces de trazas entre los artefactos.

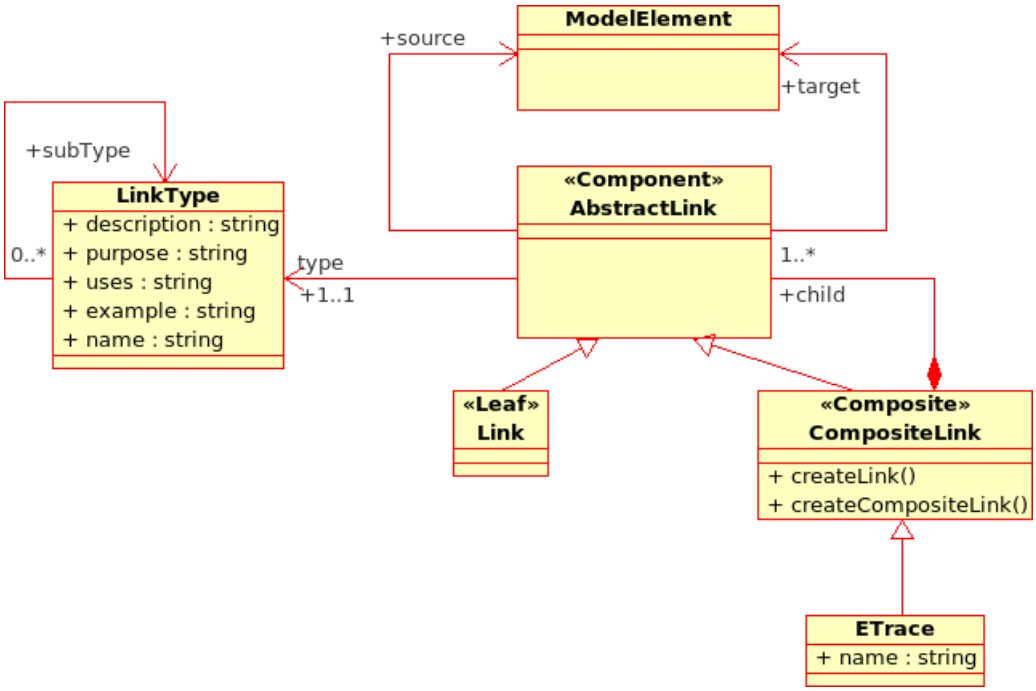

Figura 2.2: Meta-modelo de trazas anidado

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.

2.2.1. Meta-modelo de traceability

Los elementos principales del meta-modelo son los siguientes:

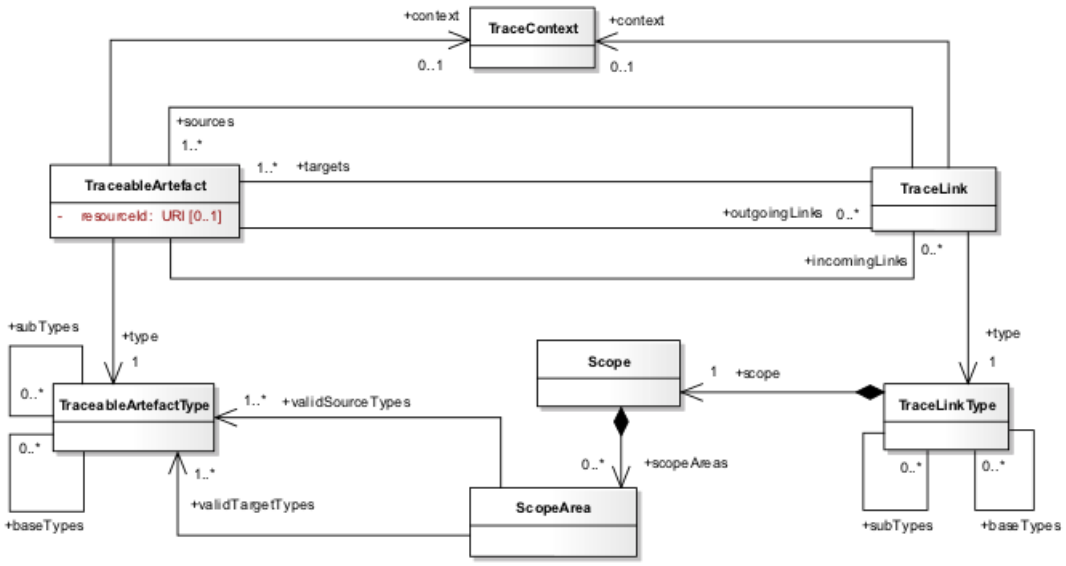

Figura 2.3: Meta-modelo de traceability

- 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. Dicho artefacto es identificado mediante resourceId.
- Un TraceLink es la abstracción de una transición de un artefacto a otro.
- Cada TraceableArtefact tiene asignada una instancia de TraceableArtefactType, éstos pueden ser agrupados jerárquicamente.
- Análogo a los tipos de los artefactos los TraceLinks también tienen un tipo, TraceLinkType, dado que la semántica de una relación entre dos artefactos puede variar.
- Información adicional de artefactos y enlaces puede ser modelada mediante el contexto representado por TraceContext.
- Las restricciones sobre el conjunto válido de artefactos sobre los cuales los tipos de enlaces son válidos es modelado mediante los elementos ScopeArea y Scope.

2.2.2. Arquitectura

Como se muestra en el dibujo 2.4, la arquitectura ha sido definida en término de cuatro módulos principales. Cada uno de los cuales implementa una de las funcionalidades principales del framework antes nombradas como se detalla a continuación:

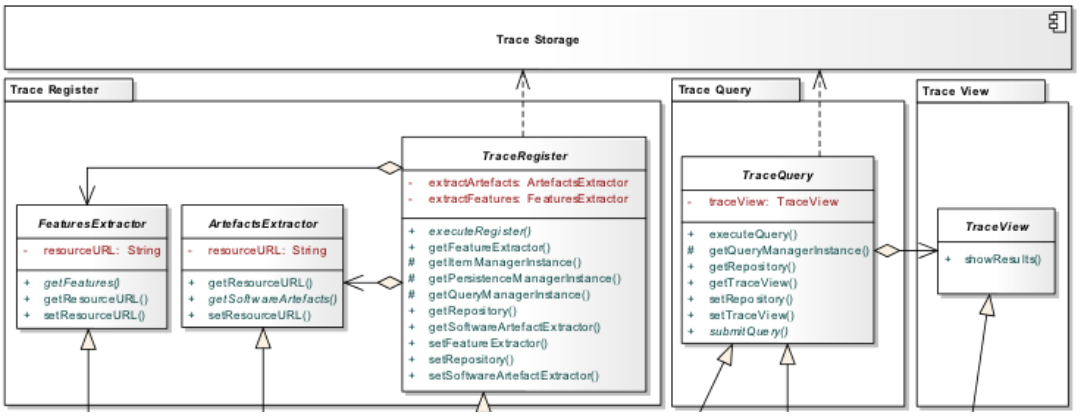

Figura 2.4: Arquitectura del Framework de Traceability

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

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

3. Trace Query: éste permite crear y ejecutar consultas para buscar enlaces de trazas específicos previamente almacenados;

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

2.3. Integración de herramientas Case

En [11] se presenta el problema real que sufre cualquier proceso de desarrollo actual, en el que en su conjunto de actividades o pasos 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) que aunque en esencia éstos se relacionan lógicamente, al ser creados y

manipulados por herramientas muy distintas que no fueron pensadas para interactuar (editores de textos, editores de modelo UML, etc), estas relaciones entre dichos artefactos se pierden o mejor dicho no existen en la práctica. En otras palabras, presenta el problema de la imposibilidad de traceability entre la mayoría de las herramientas CASE actuales. Como solución a este problema, propusieron un ambiente de integración para estas herramientas al que llamaron TiE - Tool Integration Environment, el mismo basa su integración por medio de la creación de traceability links entre los artefactos que pertenecen a las distintas herramientas.

2.4. Framework de extracción de datos de traceability genérico

En [13] proponen un framework genérico de traceability que toma una transformación de modelo y aumenta arbitrariamente su funcionalidad con un mecanismo de traceability. En el dibujo 2.5 se muestra un panorama de alto nivel de la arquitectura propuesta. 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 que llaman Trace-DSL.

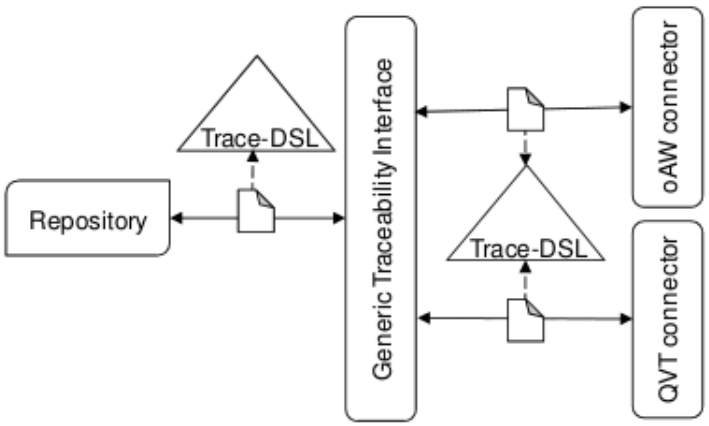

Figura 2.5: Resumen de la arquitectura del Framework Genérico de Traceability

Trace-DSL que se detalla en el dibujo 2.6, incluye como elemento raíz

TraceModel. Un Artefact representa cualquier producto traceable generado en el proceso de desarrollo, como un requerimiento o una clase o un componente como un método dentro de una clase, cualquier artefacto es identificado por un identificador único universal (UUID). Un 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.

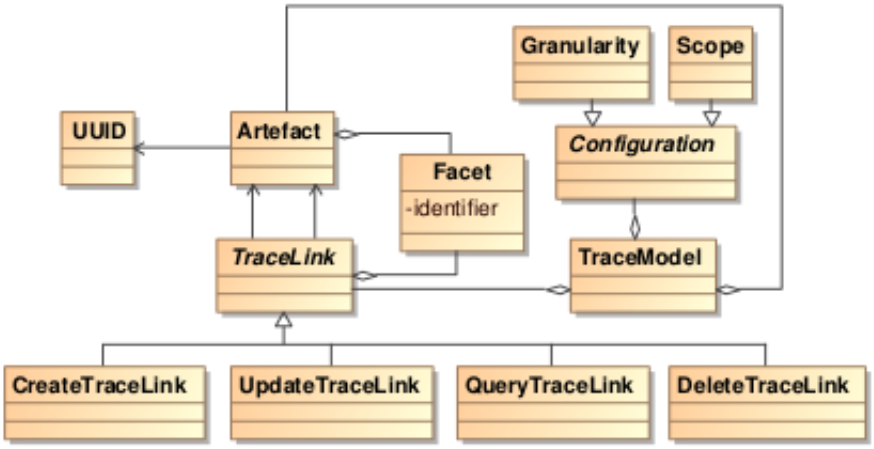

Figura 2.6: Lenguaje específico de dominio para traceability

Para asignar 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 2.7. Además de lograr una solución simple al tipado de artefactos y enlaces, se obtiene un mecanismo fácilmente extensible al contexto donde se necesite aplicar traceability.

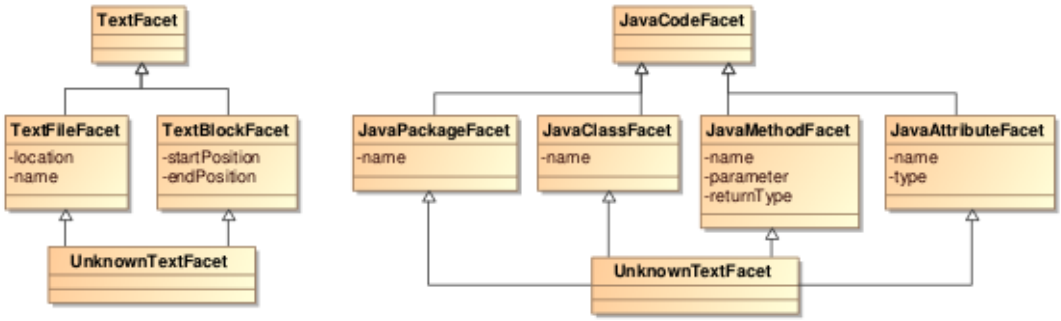

Figura 2.7: Faceta para traceability de código fuente

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 de traceability particular. En cambio la configuración del alcance implica restringir los datos de traceability a tener 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.

2.5. 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.

2.5.1. 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 2.8.

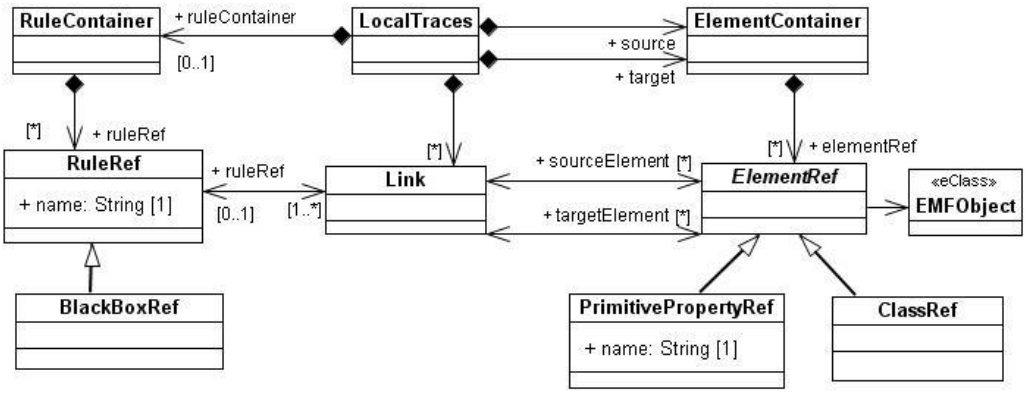

Figura 2.8: Meta-modelo de Trazas Local

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. Ambos casos pueden dar como resultado varios enlaces, por eso la relación uno a varios entre RuleRef y Link. RuleRef y BlackBoxRef son opcionales, en el caso de la primera sólo se usa para el caso de realizar una depuración de las transformaciones, y la segunda si nos encontramos con ciertas partes del sistema que no pueden ser vista 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.

2.5.2. Meta-modelo de Traceability Global

Este meta-modelo enlaza trazas locales de acuerdo a la cadena de 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. El dibujo 2.9 muestra el meta-modelo.

Este meta-modelo engloba todas las trazas locales y 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 sepa-

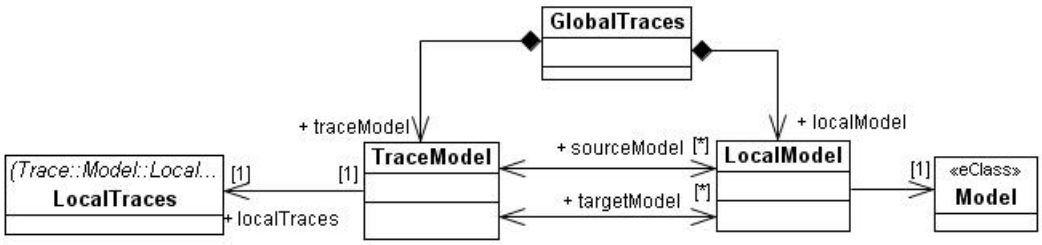

Figura 2.9: Meta-modelo de Trazas Global

ración de la compleja información de traceability lo que permite una mejor flexibilidad para la creación de las trazas y la explotación o uso de las mismas. No utilizar esta idea de trazas globales podría tener como consecuencia el colapso de todos las trazas en un único modelo de trazas para toda la cadena de transformación, más difícil de crear y consultar.

2.5.3. ¿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 2.10 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.

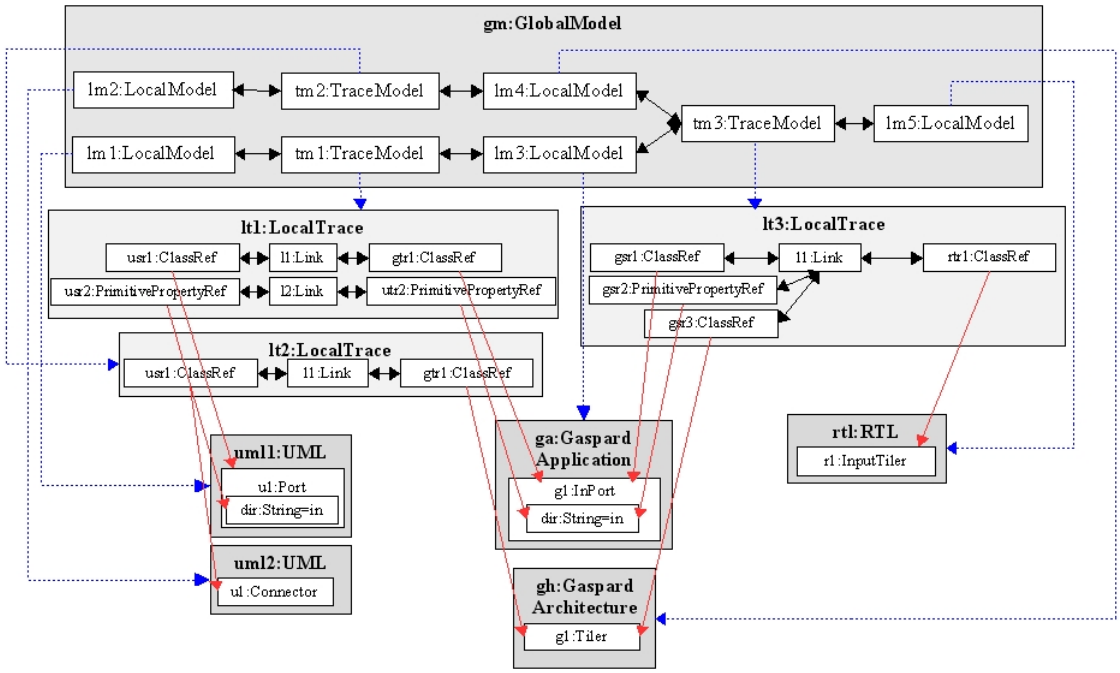

Figura 2.10: Ejemplo de un modelo de trazas local y global

Capítulo 3

Caminando hacia una solución

3.1. Desafíos

Uno de los desafíos presentados en [3] es sobre cuál de los dos enfoques de generación de trazas durante una transformación, implícita o explícita, seguir o tomar. En la implícita la transformación provee un soporte integral para traceability, en cambio en la explícita depende del desarrollador codificar las reglas de traceability como un modelo de salida.

3.1.1. Implícita

La ventaja mayor de la generación implícita de enlaces 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:

    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 esto puede ser un tema de rendimiento al manejar un modelo de transformación grande y complejo.

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 puede variar de un escenario de traceability a otro.

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

3.1.2. Explícita

Ventajas de la explícita:

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.

Otro desafío refiere a la semántica de los enlaces. Es necesario frecuentemente distinguir entre distintos tipos de enlaces. 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 frecuentemente son fuertemente dependientes al proyecto. Fijar los tipos de enlaces semánticos tiene la consecuencia de menor flexibilidad para los enlaces definidos por el usuario que pueden ser necesarios para juntar diferentes necesidades del proyecto o la compañía. Por otro lado, la selección de la semántica de un enlace es guiada por la razón del usuario sobre qué quiere realizar con el enlace, no predefinir la semántica de un enlace puede resultar en fallas de razonamientos debido al mal uso semántico.

3.2. Estrategias

[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 que ella refiere, en la segunda esta información se encuentra almacenada separada de los modelos:

3.2.1. Almacenamiento de Traceability Links 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 como elementos del modelo o atributos de 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, estos no son visibles en el destino, a la inversa (almacenados en el destino) nos encontramos con el mismo problema. Por otro lado, si la información de traceability es almacenada en ambos modelos, entonces nos encontramos con el problema de que esta información debe mantenerse consistente por cada vez que se realice un cambio. Todo lo anterior sumado a la polución que se genera en el modelo con información ajena a su contexto o fin, pudiendo lograr que dicho modelo se vuelva difícil de comprender y mantener. También, en un entorno MDE es común que los modelos tengan sus propias representaciones y semánticas, por lo cual es muy difícil distinguir la información de traceability de los objetos del modelo. Como resultado, el análisis automatizado de la información de traceability se hace muy difícil. Los enfoques principales que hacen uso de esta estrategia utilizan las construcciones específicas del lenguaje, por ejemplo determinados tipos de enlaces de traceability están representados en los diagramas UML mediante el uso de las estereotipos como «refines».

3.2.2. Almacenamiento externo de Tracebility Links

En esta estrategia la información de traceability se encuentra almacenada separada de los modelos a los que refiere en un modelo aparte. Esta propuesta tiene dos claras ventajas, la primera, los modelos origen y destino se mantienen totalmente limpios, de esta manera la polución antes nombrada es evitada. Y la segunda, dado que el modelo 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 estrategia intra-modelo. Un requisito previo para el almacenamiento externo de los traceability links, es que los diferentes elementos del modelo

tengan identificadores únicos, de modo que los traceability links relacionados se pueden resolver inequívocamente. Un ejemplo es el mecanismo propuesto por MetaObject Facility (MOF) y por el Eclipse Modeling Framework (EMF) en la forma de un identificador xmi.id .

3.3. Meta-modelos

Los modelos que definen los traceability links son determinados cada uno por su meta-modelo, este meta-modelo puede ser clasificado como un "meta-modelo de traceability de propósito general" o un "meta-modelo de de traceability de caso específico".

3.3.1. 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 traceability link se puede conectar con cualquier número de elementos, de cualquier tipo 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 traceability links fuertemente tipados con semántica y restricciones definidas rigurosamente, se abre la puerta a establecimientos de traceability links ilegítimas. Como por ejemplo en el caso que se quiere representar links 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 links 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 casos específicos de información y su semántica.

3.3.2. Meta-modelo de caso específico

En este caso, para cada escenario de traceability se define un meta-modelo de traceability específico. Este meta-modelo de traceability captura trace-

ability links fuertemente tipados de caso específico 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 herramientas para que solo puedan establecer traceability links legítimos. Por otro lado, un meta-modelo de traceability para cada caso específico requiere mucho esfuerzo en su construcción, así como herramientas que soporten diferentes meta-modelos de traceability. Para ser fuertemente tipado el meta-modelo de traceability necesita referir explícitamente a los tipos de elementos definidos en otros meta-modelos. Por ejemplo, considereremos que es necesario definir un meta-modelo de traceability que permita el establecimiento de traceability links entre instancias de A (del meta-modelo MMa) y las 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 debe permitir referencias inter-meta-modelo. Un ejemplo de tecnología que soporta referencias inter-meta-modelo es el Eclipse Modeling Framework (EMF). A pesar de que un meta-modelo capturado utilizando una tecnología que permita referencias inter-meta-modelo puede confiar tipos seguros, encontramos frecuentemente otras restricciones que necesitan especificarse y que el meta-modelo de traceability no puede capturarlas. Por ejemplo tomando como referencia el ejemplo anterior, podríamos querer 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 traceability links con sentido, que pueden ser automáticamente validadas para descubrir posibles omisiones e inconsistencias. Estas cuestiones pueden surgir, ya sea durante el establecimiento de los traceability links o más tarde en el ciclo de vida de los modelos entre los que los traceability links ya han sido establecidos.

Capítulo 4

Descripción de tecnologías

Conclusión

Glosario

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.