FACULTAD DE INFORMÁTICA

UNlVERSlDAD NAClONAL DE LA PLATA

**TESINA DE LICENCIATURA**

**Título:** Mecanismos de rastreo en el desarrollo de software dirigido por modelos

**Autores:** Mariano Gabriel Gili

**Director:** Prof. Dra. Claudia Pons

**Codirector:** -

**Asesor profesional:** -

**Carrera:** Licenciatura en Informática – Plan 90

Resumen

*Completar con tipografía Arial – tamaño 10 - justificado*

Palabras Claves Conclusiones

*Completar con tipografía Arial – tamaño 10 - justificado Completar con tipografía Arial – tamaño 10 - justificado*

Trabajos Realizados Trabajos Futuros

*Completar con tipografía Arial – tamaño 10 - justificado Completar con tipografía Arial – tamaño 10 - justificado*

Fecha de la presentación:

**Octubre de 2014**

´Indice general

P´agina

´Indice de im´agenes IV Agradecimientos V Introduccion VI Ob jetivos VII

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. | Traceability |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
|  | 1.1. Introduccio | n . . . . . . . | . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 2 |
| 1.1.1. Beneficios . . . . . . . | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . 3 | |
| 1.1.2. Algunos inconvenientes | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . 4 | |

2.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Teor´ıa de traceability | | | | | | | | | | | | | | | | | | | | | | 5 |
| 2.1. | Nivel de Granularidad . . . . . . . . . . | | | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | 7 |
| 2.2. | Meta-modelos de traceability . . . . . . . | | | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | 7 |
|  | 2.2.1. Meta-modelo de prop´osito general | | | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | 7 |
|  | 2.2.2. Meta-modelo de caso espec´ıfico . | | | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | 8 |
| 2.3. | Tipos de tracelinks . . . . . . . . . . . . | | | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | 9 |
| 2.3.1. Una clasificacion gen´erica de traceability . . . . . . . . . 10 | | | | | | | | | | | | | | | | | | | | | | |
| 2.4. | Generacion de tracelinks . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 12 |
|  | 2.4.1. Generaci´on impl´ıcita | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 13 |
|  | 2.4.2. Generaci´on Expl´ıcita | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 14 |
| 2.5. | Estrategias de almacenamiento . . . . . . . . . . . | | | | | | | | | | | | | | . | . | . | . | . | . | . | 14 |
|  | 2.5.1. Almacenamiento intra-modelo de tracealinks | | | | | | | | | | | | | | . | . | . | . | . | . | . | 15 |
|  | 2.5.2. Almacenamiento externo de tracelinks . . . | | | | | | | | | | | | | | . | . | . | . | . | . | . | 15 |

3.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problemas y Desaf´ıos | | | | | | | | | | | | | | | | | | | | | 17 |
| 3.1. | Conocimiento de traceability . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 18 |
| 3.2. | Capacitacion y certificacion . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 18 |
| 3.3. | Soporte y evoluci´on . . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 19 |
| 3.4. | Semantica de los tracelinks . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 20 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3.5. | | Escalabilidad . . . . . . . | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 20 |
| 3.6. | | Factores humanos . . . . . | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 21 |
| 3.7. | | Analisis de costo-beneficio | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 22 |
| 3.8. | | M´etodos y herramientas . | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 22 |
| 3.9. | | Procesos . . . . . . . . . . | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 23 |
| 3.10. | | Conformidad . . . . . . . | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 24 |
| 3.11. | | Mediciones y Benchmarks | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 24 |
|  | 3.12. Transferencia de tecnolog´ıa . . . . . . | | | | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 25 |
| 4. | Manos a la obra | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 27 |
|  | 4.1. Requerimientos . . . . . . . . . . . . | | | | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 28 |
|  | 4.1.1. Introducci´on . . . . . . . . . . | | | | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 28 |
|  | 4.2. El esquema de traceability propuesto | | | | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 28 |
|  | 4.2.1. Lo que no ofrece . . . . . . . | | | | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 30 |
|  | 4.3. El prototipo . . . . . . . . . . . . . . | | | | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 30 |
|  | 4.3.1. La herramienta . . . . . . . . | | | | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 30 |
|  | 4.3.2. Arquitectura . . . . . . . . . . | | | | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 31 |
| 4.3.3. | | | Transformacion QvtoTrace To Trace | | | | | | | | | |  | . | . | . | . | . | . | . | . | . | . | . | 33 |
| 4.3.4. | | | Editor grafico y tabular TraceEditor | | | | | | | | | |  | . | . | . | . | . | . | . | . | . | . | . | 33 |

5.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Descripci´on de tecnolog´ıas | | | | | | | | | | | | | | | | | | | 39 |
| 5.1. | Eclipse . . . . . . . . . . . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 41 |
|  | 5.1.1. El proyecto . . . . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 41 |
|  | 5.1.2. La plataforma Eclipse . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 42 |
|  | 5.1.3. Resumen y mas informacio | n | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 44 |
| 5.2. | Eclipse Modeling Framework . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 44 |
|  | 5.2.1. El framework EMF . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 44 |
|  | 5.2.2. El (Meta) modelo Ecore . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 46 |
|  | 5.2.3. Beneficios y mas informacio | n |  | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 47 |
| 5.3. | Graphical Modeling Framework . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 47 |
|  | 5.3.1. El framework GMF . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 47 |
|  | 5.3.2. Arquitectura . . . . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 47 |
|  | 5.3.3. Modelos y flujo de trabajo | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 48 |
|  | 5.3.4. M´as informacion . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 50 |
| 5.4. | Atlas Transformation Language . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 50 |
|  | 5.4.1. ¿Qu´e es ATL? . . . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 50 |
|  | 5.4.2. Conceptos de ATL . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 50 |
|  | 5.4.3. El lenguaje ATL . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 51 |
|  | 5.4.4. M´as informacion . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 51 |
| 5.5. | QVT . . . . . . . . . . . . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 52 |
|  | 5.5.1. Introducci´on a QVT . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 52 |
|  | 5.5.2. Lenguajes QVT . . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 52 |
|  | 5.5.3. M´as informacion . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 53 |

6. Traba jos relacionados 54

|  |  |  |
| --- | --- | --- |
| 6.1. | Un motor de traceability de transformacion de modelos en la |  |
|  | Ingenier´ıa de Software . . . . . . . . . . . . . . . . . . . . . . . | 55 |
| 6.2. | Un Framework de Traceability dirigido por modelos para el desa- |  |
|  | rrollo de Software Product Line (SPL) . . . . . . . . . . . . . . | 57 |
|  | 6.2.1. Meta-modelo de traceability . . . . . . . . . . . . . . . . | 57 |
|  | 6.2.2. Arquitectura . . . . . . . . . . . . . . . . . . . . . . . . . | 58 |
| 6.3. | Integracion de herramientas Case . . . . . . . . . . . . . . . . . | 59 |
| 6.4. | Framework gen´erico de extraccion de datos de traceability . . . | 60 |
| 6.5. | Traceability local y global . . . . . . . . . . . . . . . . . . . . . | 62 |
|  | 6.5.1. Meta-modelo de Traceability Local . . . . . . . . . . . . | 62 |
|  | 6.5.2. Meta-modelo de Traceability Global . . . . . . . . . . . . | 63 |
|  | 6.5.3. ¿C´omo trabaja el framework? . . . . . . . . . . . . . . . | 64 |

|  |  |
| --- | --- |
| Conclusi´on | 66 |
| Anexos | 67 |
| Traceability en ATL | 68 |
| QvtoTrace To Trace | 69 |
| Glosario | 77 |
| Siglas | 79 |
| Bibliograf´ıa | 81 |

´Indice de im´agenes

P´agina

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2.1. | Clasificacion de traceability inicial . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 11 |
| 2.2. | Jerarqu´ıa de links impl´ıcitos . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 11 |
| 2.3. | Jerarqu´ıa de links expl´ıcitos . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 13 |
| 4.1. | Esquema de traceability propuesto | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 29 |
| 4.2. | Secuencia del prototipo . . . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 31 |
| 4.3. | Modelo de tracelinks de QVT . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 32 |
| 4.4. | Meta-modelo TraceEditor . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 33 |
| 4.5. | Editor gr´afico y tabular TraceEditor |  | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 34 |
| 4.6. | El editor gr´afico . . . . . . . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 35 |
| 4.7. | El editor tabular . . . . . . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 36 |
| 4.8. | Perspectiva del editor . . . . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 37 |
| 5.1. | Elcipse Workbench . . . . . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 43 |
| 5.2. | Modelo Ecore simplificado . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 46 |
| 5.3. | Arquitectura GMF . . . . . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 48 |
| 5.4. | Flujo de trabajo de GMF . . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 49 |
| 5.5. Modelos, transformaciones y sus meta-modelos . . . . . . . . . . 51 | | | | | | | | | | | | | | | | | | |
| 5.6. | Relacion entre los meta-modelos QVT . . . | | | | | | . | . | . | . | . | . | . | . | . | . | . | 52 |
| 6.1. | Arquitectura de la herramienta ETraceTool . | | | | | | . | . | . | . | . | . | . | . | . | . | . | 55 |
| 6.2. | Meta-modelo de tracelinks anidado . . . . . | | | | | | . | . | . | . | . | . | . | . | . | . | . | 56 |
| 6.3. | Meta-modelo de traceability . . . . . . . . . | | | | | | . | . | . | . | . | . | . | . | . | . | . | 58 |
| 6.4. | Arquitectura del Framework de Traceability | | | | | | . | . | . | . | . | . | . | . | . | . | . | 59 |

6.5. Resumen de la arquitectura del Framework Gen´erico de Traceability 60

6.6. Lenguaje espec´ıfico de dominio para traceability . . . . . . . . . 61

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6.7. | Faceta para traceability de c´odigo fuente | . | . | . | . | . | . | . | . | . | . | . | . | . | 61 |
| 6.8. | Meta-modelo de tracelinks Local . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | 62 |
| 6.9. | Meta-modelo de tracelinks Global . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | 63 |
| 6.10. Ejemplo de un modelo de tracelinks local y global . . . . . . . . 65 | | | | | | | | | | | | | | | |

Agradecimientos

A mis padres, Ana Mar´ıa Bloga y Nestor Hugo Gili, quienes nunca dudaron en ofrecerme el soporte necesario para afrontar esta carrera de grado. A

mi mujer Mar´ıa Jimena Reimundo, por brindarme el apoyo que me ayud a

terminar el trabajo de tesis al mismo tiempo que empezamos a construir nuestro hermosa familia y hogar.

A la Universidad Nacional de La Plata y en especial a la Facultad de Inform´atica, por el excelente nivel educativo al que me dio acceso. A las catedras de Programacion y Programacion Funcional, que me abrieron las puertas para poder desarrollar la gratificante tarea de ayudante y colaborador de c´atedra respect´ıvamente. Al laboratorio LIFIA, por haberme brindado la posibilidad de realizar trabajos de investigaci´on y de asistir a cursos extracurri- culares en los que adquir´ı conocimientos sobre diversos temas que me resultaron apasionantes.

A la profesora Claudia Pons, por haberme introducido en el grupo de investigaci´on de Desarrollo Dirigido por Modelos, desde donde definimos el tema que se presentar´a a lo largo de este trabajo de tesis, y por haber sido mi gu´ıa en el desarollo del mismo, demostrandome siempre su buena predisposicion en cada consulta realizada.

A mi familia, amigos y compan˜eros de estudio y del trabajo, por su acom- pan˜amiento y empujon en este u´ltimo trayecto del camino que con tanto esfuerzo recorr´ı.

¡¡¡Muchas gracias a todos!!!

Introduccio´n

En la Ingenier´ıa de Software Dirigida por Modelos (MDE de Model-Driven Engineering), se define al modelo como artefacto principal que toma participa- cion a lo largo de todas las tareas o procesos que conforman dicha ingenier´ıa,

´esto es, el analisis, el disen˜o, el desarrollo, las pruebas, el mantenimiento, etc. Una implementacion particular propuesta por el Object Management Group (OMG ) que acompan˜a esta idea es la Arquitectura Dirigida por Modelos (MDA de Model-Driven Architecture), cuyo ciclo de proceso de desarrollo est´a basado enteramente en el uso de modelos formales y transformaciones que se realizan sobre dichos modelos. Una caracter´ıstica muy importante de todo proceso de Desarrollo Dirigido por Modelos (MDD de Model-Driven Development), es lo que se conoce como posibilidad de rastreo (de ahora en mas en ingl´es tra- ceability ), 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 t´ermino artefacto, hablamos por ejemplo de un requerimiento de sistema, un componente de software, un caso de prueba, entre otros. El mantenimiento y la definicion de las relaciones y dependencias que existen entre los artefactos no es una tarea facil, constituyendo un desaf´ıo desde principios de 1970.

En el presente documento de tesis se aborda el tema de traceability, presen- tando un analisis de los distintos problemas que au´n se encuentran abiertos a la investigacion y al debate, como as´ı tambi´en un conjunto de soluciones pro- puestas que encontr´e a lo largo de mi investigaci´on. Finalmente, se expone un esquema de traceability con el fin de aportar un avance en el desarrollo de este tema.

Ob jetivos

El objetivo del trabajo de tesis consiste en introducir al lector en el tema de traceability, abordando una serie de problemas que au´n se encuentran abiertos a la investigaci´on y debate en el ´ambito de MDE; para luego, elaborar un esquema de traceability como aporte a la soluci´on de dichos problemas.

Asimismo, disen˜ar e implementar una herramienta que pueda ser integrada a otra de desarrollo MDE, y asista al desarrollador automatizando el proceso de definicion de trazas (de ahora en m´as en ingl´es tracelinks o links ) entre elementos de los modelos origen y destino. Esta solucion proveer´a un mapa

de transformaciones que permitir

determinar la procedencia de cada ´ıtem del

modelo destino, y su correspondiente origen en el modelo fuente.

Cap´ıtulo 1

Traceability

En este primer cap´ıtulo, a modo de introduccion se define el con- cepto de traceability y se exponen, por un lado los beneficios de su aplicacion, y por otro los inconvenientes que supone su implemen- tacion.

´Indice

|  |  |
| --- | --- |
| 1.1. Introduccion | 2 |
| 1.1.1. Beneficios | 3 |
| 1.1.2. Algunos inconvenientes | 4 |

1.1. Introduccion

Segu´n el Glosario Estandar de T´erminos de la Ingenier´ıa de Software del Institute of Electrical and Electronics Engineers (IEEE) [1] la idea de traceability se define como:

El grado o nivel en el cual una relaci´on puede ser establecida entre dos o m´as productos del proceso de desarrollo, especialmente entre productos que tengan una relaci´on de predecesor-sucesor o principal-secundario; por ejemplo el grado en el cual el requerimiento y el disen˜o de un componente de software se corresponden.

La definicion anterior es utilizada principalmente por The Requirements Ma- nagement Community. Nosotros, quienes necesitamos un punto de vista mas cercano al contexto de MDD, usamos el t´ermino traceability para describir

cualquiera de las complejas relaciones l´ogicas 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.

Esta definicion incluye a todos los productos creados durante el proceso de desarrollo, de despliegue o implantacion, y tambi´en los creados a lo largo del mantenimiento. Ademas necesita que la informacion de traceability sea accesible a lo largo de toda la vida del producto de software.

En la ingenier´ıa de software encontramos dos usos o semanticas principales que dependen del contexto de traceability :

traceability en la Ingenier´ıa de Requerimientos : donde se guarda un requerimiento desde su definici´on hasta su implementaci´on. En m´as detalle segu´n [6] se refiere a la habilidad de describir y seguir la vida de los requerimientos en ambas direcciones, hacia delante y hacia atras (forward and backward traceability ). Desde los or´ıgenes, pasando por el desarrollo y la especificacion, hacia su posterior entrega y uso, y a trav´es de todos los per´ıodos de refinamiento e iteracion de cualquiera de estas etapas.

traceability en MDD: donde se almacenan principalmente las relacio- nes existentes entre los artefactos producto de las transformaciones de modelos.

Las relaciones de traceability pueden ser definidas de forma automatica, por ejemplo producto de una transformaci´on de modelos, o de forma manual como el caso de una relacion de implementaci´on entre un requerimiento y un componente de software.

Entre numerosos beneficios de traceability, que se expondran en detalle mas adelante, podemos encontrar que ayuda a identificar las relaciones y dependen- cias que existen entre los artefactos de software. Tambi´en traceability es crucial entre los requerimientos y su representacion en los modelos para asegurar que el conjunto relevante de requerimientos fueron debidamente implementados en el codigo. Pero no solo traceability asegura la identificacion de objetos y elementos relacionados, tambi´en puede facilitar el an´alisis de impactos de cambios durante el desarrollo de software.

Por todo lo anterior, queda demostrado que una buena solucion de traceabi- lity que se encuentre provista de informacion actualizada ser´a un servicio muy valioso tanto para jefes de proyectos, como para desarrolladores de software y/o consultores de mantenimiento.

1.1.1. Beneficios

A continuacion se listan un conjunto de actividades que provienen de dis- tintos dominios de la ingenier´ıa, en las cuales el uso de traceability es muy beneficioso segu´n [2] y [3].

An´alisis de Sistemas Ayuda a entender la complejidad de un siste- ma, navegando el modelo de tracelinks resultante de la ejecucion de las distintas cadenas de transformacion.

An´alisis de Cobertura Por ejemplo en el momento de ejecuci´on de los casos de prueba, el uso de traceability es crucial a la hora de determinar si todos los requerimientos fueron cubiertos, es decir tenidos en cuenta.

An´alisis de Impacto de Cambios Traceability nos ayuda a ver como los cambios en un modelo repercutir´an en los modelos relacionados; tam- bi´en 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.

An´alisis de Hu´erfanos Permite encontrar f´acilmente los elementos hu´erfanos de un modelo, pues seran aquellos artefactos que no se encuen- tren relacionados a ningu´n tracelink.

Comprensi´on del Software y la Ingenier´ıa Inversa Traceability es crucial cuando se necesita identificar todas las entidades relacionadas a una en particular, entender el tipo de relacion existente, identificar las abstracciones, es decir los patrones de disen˜o y/o estilos de arquitectura, etc.

An´alisis de Requerimiento Traceability por ejemplo ayuda a identi- ficar el artefacto particular que demanda una propiedad espec´ıfica; como as´ı tambi´en a encontrar y resolver un conjunto de requerimientos contra- dictorios, entre otros.

Apoyo en la toma de decisiones Para justificar una decisi´on dado que nos facilita el entendimiento de los factores y metas que influyen en la

misma; tambi´en traceability nos ser

muy u´til en el analisis y evaluacion

de distintas propuestas de solucion que se nos puedan presentar.

Configuraci´on del Sistema y Versionado El uso de traceability es beneficioso para identificar: las restricciones entre los componentes, los cambios necesarios para resolver una restricci´on, las diferencias entre dos versiones distintas de un mismo artefacto y el impacto que estas diferencias tendran sobre otros artefactos.

1.1.2. Algunos inconvenientes

Mas all

de las numerosas ventajas que el uso de traceability ha demostrado

hoy en d´ıa, su puesta en practica apenas se ha establecido. Las principales razones de ello, se deben segu´n [3] a:

El alto costo de la creacion de la informacion de traceability.

La necesidad de mantenimiento manual de la informacion de tracea- bility.

La falta de heur´ısticas que determinen qu´e informaci´on de los links

deben ser grabados.

Las discrepancias entre los distintos roles que ejercen los usuarios de la informacion de traceability, por ejemplo entre quienes crean los tracelinks y quienes los interpretan.

La carencia de soporte adecuado de las herramientas.

El uso de diferentes lenguajes, dado que por ejemplo los requerimien- tos se escriben en lenguaje natural mientras que los programas en algu´n lenguaje formal de programacion.

Las diferencias entre los distintos niveles de abstracci´on en los cuales los artefactos describen el sistema de software, por ejemplo el nivel de abstraccion de los artefactos usados durante el ciclo de disen˜o difiere de los niveles usados en las etapas de implementacion.

Mas adelante en el cap´ıtulo 3 se presentaran con mas detalle los desaf´ıos presentes en la implementaci´on y uso de traceability.

Cap´ıtulo 2

Teor´ıa de traceability

En el presente cap´ıtulo se aborda un conjunto de temas impor- tantes de la teor´ıa de traceability, en la primera seccion se explica el concepto de nivel de granularidad de un tracelink. Las siguientes dos secciones se refieren a la sem´antica de traceability, introdu- ciendo primero las dos distintas estrategias de meta-modelos de traceability que existen, y luego, la compleja tarea de la definici´on de tipos de tracelinks. En la secci´on siguiente, se presentan dos formas distintas de generaci´on de tracelinks, y en la u´ltima, las dos principales estrategias de almacenamiento de la informa- cion de traceability que se pueden implementar. Estos temas han sido muy importantes y relevantes a la hora de definir el esquema de traceability propuesto que se detalla en el cap´ıtulo 4.

´Indice

|  |  |
| --- | --- |
| 2.1. Nivel de Granularidad | 7 |
| 2.2. Meta-modelos de traceability | 7 |
| 2.2.1. Meta-modelo de prop´osito general | 7 |
| 2.2.2. Meta-modelo de caso espec´ıfico | 8 |
| 2.3. Tipos de tracelinks | 9 |
| 2.3.1. Una clasificaci´on gen´erica de traceability | 10 |
| 2.4. Generacion de tracelinks | 12 |
| 2.4.1. Generacion impl´ıcita | 13 |
| 2.4.2. Generacion Expl´ıcita | 14 |
| 2.5. Estrategias de almacenamiento | 14 |
| 2.5.1. Almacenamiento intra-modelo de tracealinks | 15 |

2.5.2. Almacenamiento externo de tracelinks 15

2.1. Nivel de Granularidad

El t´ermino granularidad se refiere al nivel o grado de detalle en el cual un tracelink se genera y/o se registra, es una caracter´ıstica ´ıntimamente relacionada con el uso que se le quiera dar a traceability. Por ejemplo cuando se desee trabajar sobre diagramas de clases UML, se podran generar tracelinks a nivel de paquetes, de clases o de m´etodos. La granularidad de un tracelink (de ahora en mas en ingl´es trace granularity ) es definida por la granularidad del artefacto origen y la granularidad del artefacto destino.

Esta propiedad debe ser definida cuidadosamente para efectivamente dar un buen soporte en las tareas de traceability, dado que es un factor que determi-

nar

la complejidad, y por tanto el esfuerzo, en el analsis y en la utilizacion del

conjunto de los tracelinks obtenidos.

2.2. Meta-modelos de traceability

Para llegar a la definicion de cualquier propuesta de traceability se necesita de un modelo en el cual se especifiquen los conceptos, las reglas y las relaciones que existen, por ejemplo entre los artefactos y los tracelinks. E´ ste modelo de traceability va a estar determinado por un meta-modelo, el cual puede llegar a ser clasificado como un meta-modelo de traceability de prop´osito general, o como un meta-modelo de traceability de caso espec´ıfico, a continuacion se explica cada uno.

2.2.1. Meta-modelo de prop´osito general

En este caso, nos encontramos con un meta-modelo gen´erico que permite la captura de tracelinks entre cualquier tipo de elementos de un determinado modelo. Un tracelink se puede conectar con cualquier nu´mero de elementos, de cualquier tipo y de cualquier modelo. Las principales ventajas de este tipo de meta-modelo 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 otorgandoles la capacidad de importar, exportar y gestionar informacion de traceability en un formato comu´n.

Por otro lado, como el meta-modelo de proposito general no capta casos es- pec´ıficos de tracelinks fuertemente tipados, es decir sin semantica y restricciones definidas rigurosamente, se abre la puerta a establecimientos de links ileg´ıtimos. Como por ejemplo, en el caso de tracelinks entre un diagrama de clases y un

modelo de base de datos relacional en donde existir´an v´ınculos entre las cla- ses del primer modelo y las tablas del segundo, un meta-modelo de traceability gen´erico permitir´a el establecimiento de links ileg´ıtimos entre una clase y una columna.

Para permitir un mejor soporte para el caso de los requisitos espec´ıficos, es un m´etodo de uso frecuente la provision de mecanismos de extension que acom- pan˜an el meta-modelo de proposito general. Sin embargo, todav´ıa carecen de la eficacia que ofrecen los meta-modelos de casos espec´ıficos para capturar estos tipos de situaciones que requieran tal nivel de legitimidad entre la informaci´on y su semantica.

2.2.2. Meta-modelo de caso espec´ıfico

En este caso, para cada escenario de traceability se define un meta-modelo espec´ıfico. Este meta-modelo de traceability captura links fuertemente tipados para casos espec´ıficos con una sem´antica bien definida, que pueden o no in- cluir restricciones de correccion. Debido a su naturaleza de tipado fuerte y a las restricciones asociadas, limita a los usuarios y a las herramientas para que s´olo puedan establecer links leg´ıtimos. Por otro lado, la definici´on de un meta-modelo para cada caso espec´ıfico requiere de mucho esfuerzo en su construcci´on, as´ı co- mo el uso de 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 expl´ıcitamente 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 lograr tal meta-modelo, la tecnolog´ıa 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 ejem- plo de tecnolog´ıa que soporta referencias inter-meta-modelo es el framework Eclipse Modeling Framework (EMF).

Mas all

de que un meta-modelo de traceability (que fue definido utilizando

una tecnolog´ıa que permite referencias inter-meta-modelo) puede brindar tipos seguros, encontramos frecuentemente otras restricciones que necesitan especi- ficarse y que dicho meta-modelo no puede capturarlas. Por ejemplo, tomando como referencia el ejemplo anterior, podr´ıamos precisar que cada instancia A de MMa solo se puede vincular a no m´as de una instancia B de MMb. Para es- pecificar tales restricciones, se requiere un lenguaje de especificacion que pueda expresar restricciones que abarquen elementos que pertenezcan a modelos defini-

dos por diferentes meta-modelos. En la actualidad el Lenguaje de Restricciones para Objetos (OCL de Object Constraint Language) carece de esta capacidad, ya que no proporciona las construcciones para expresiones que atraviesen mo- delos (cross-model ). Ejemplos de lenguajes que soportan el establecimiento con tales restricciones incluyen el Epsilon Validation Language (EVL) y el XLinkit toolkit.

La combinacion de un meta-modelo de traceability fuertemente tipado en conjunto con la verificacion de restricciones inter-modelo restringe a los usua- rios y a las herramientas a establecer y mantener solo tracelinks con sentido, los cuales pueden ser autom´aticamente validados para descubrir posibles omisiones e inconsistencias. Estas cuestiones pueden realizarse ya sea durante el estable- cimiento de los tracelinks, o m´as tarde, durante el ciclo de vida de los modelos donde ya los tracelinks han sido establecidos.

2.3. Tipos de tracelinks

Uno de los desaf´ıos presentados en el cap´ıtulo 3 (seccion 3.4) se refiere a la definicion semantica de los tracelinks, tarea que nos permite descubrir los distintos tipos de links con los que nos podemos llegar a encontrar, entender su significado y uso. Es importante remarcar que la determinacion de la semantica de un link se encuentra totalmente guiada por la raz´on del usuario respecto de qu´e quiere realizar o representar con dicho link. No predefinir la semantica de un tracelink correctamente, puede resultar en fallas de razonamientos y/o entendi- mientos. Por ejemplo, un link que muestra la relacion entre un requerimiento textual y un elemento de un diagrama de caso de uso UML tiene una sem´antica muy distinta a la de un tracelink que determina una relacion de refinamiento dentro de un modelo de clases tambi´en UML.

La definicion del conjunto de tipos de tracelinks por lo general es fuerte- mente dependiente del contexto de un proyecto, por lo cual definirlo de forma fija tendr´a como consecuencia una p´erdida de flexibilidad para los usuarios que deseen tipar los tracelinks solamente de acuerdo a las necesidades del proyecto o la compan˜´ıa. Au´n as´ı, conforme un proyecto de desarrollo crece, la adminis- tracion de la informacion de traceability que en consecuencia se va generando se vuelve extremadamente compleja, una jerarqu´ıa de clasificaci´on de dicha infor-

macion resultar

esencial para poder entenderla y administrarla mejor. En [10]

se encuentra una propuesta de clasificacion gen´erica de traceability en conjunto con la descripcion del proceso usado para su obtenci´on al que llamaron Tracea- bility Elicitation and Analysis Process (TEAP) y que a continuaci´on se explica en detalle.

2.3.1. Una clasificacion gen´erica de traceability

Esta propuesta comienza con una clasificaci´on o meta-modelo de traceability inicial, luego de forma iterativa e incremental esta clasificacion se va refinando de acuerdo a las siguientes tareas: obtenci´on, analisis y clasificacion. En la ob- tencion se identifica un tracelink y sus relaciones. En el analisis, se abstraen las principales caracter´ısticas del link obtenido identificando las restricciones, sus relaciones y generalizaciones. Y por u´ltimo, se define la clasificacion a la que pertenece.

El meta-modelo inicial se puede ver en el dibujo 2.1, en el que se encuentran los siguientes conceptos fundamentales: Artefact, Tracelink y Operation. Artefact refiere tanto a artefactos MDE, por ejemplo modelos espec´ıficos de dominio, como a no MDE, por ejemplo hojas de calculo. En Operation se encierran las operaciones tanto manuales como automaticas que determinan qu´e informaci´on de traceability debe almacenarse. Por u´ltimo Tracelink da inicio con la clasifica- cion de tracelink impl´ıcito (Implicit Link) por un lado, y tracelink expl´ıcito (Explicit Link) por el otro, en donde tracelink impl´ıcito refiere a los links que son creados y manipulados por la aplicaci´on de las operaciones MDE, y por su parte tracelink expl´ıcito refiere a los links que se encuentran concretamente ya representados en los modelos.

Clasificaci´on de tracelinks impl´ıcitos

En esta jerarqu´ıa que se muestra en el dibujo 2.2 se encuentran representados el conjunto de operaciones MDE principales: Consulta (Query Link), Transfor- macion modelo a modelo (M2M Link), Transformacion modelo a texto (M2T Link), Composicion (Composition Link), Actualizacion (Update Link), Creacion (Creation Link), Eliminaci´on (Delete Link), entre otras.

Clasificacion de tracelinks expl´ıcitos

En un principio esta clasificaci´on se encuentra dividida en dos grandes gru- pos representados por las siguientes clases basicas: Link modelo-modelo, que representa por ejemplo las relaciones de dependencias UML, y Link modelo- artefacto, conjunto al que pertenece por ejemplo el link entre un modelo y un documento de texto que dicta un requerimiento. En el dibujo 2.3 aparecen como Model-Model Link y Model-Artefact Link respectivamente.

Model-Model Link se encuentra a su vez subdividido en Link estatico (Static Link) y Link din´amico (Dynamic Link). En el primero se representan relacio- nes estructurales que no cambian con el tiempo, en cambio en el segundo se

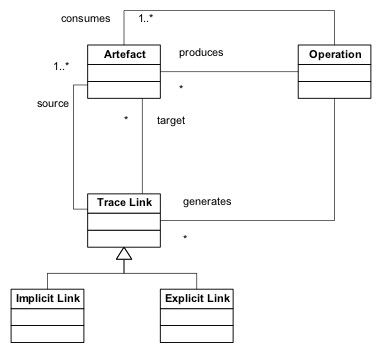


Figura 2.1: Clasificaci´on de traceability inicial

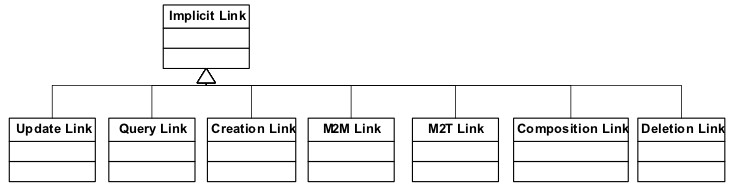


Figura 2.2: Jerarqu´ıa de links impl´ıcitos

representa informacion de los modelos que si puede llegar a variar.

Los links estaticos pueden ser, o Links consistentes (Consistent-With) don- de dos modelos deben mantenerse de acuerdo o consistentes entre s´ı, o Links dependientes (Dependency) donde la estructura y/o la semantica de un modelo depende de otro. Por su parte, los Links dependientes pueden ser: Link de subti- po (Is-A), Link de referencia, link de subconjunto, de importacion y exportacion, de uso, de refinamiento (Refines), etc´etera.

Entre los links din´amicos se incluyen los Links de llamadas (Calls) donde un modelo hace uso de m´etodos provistos por otro y los Links de notificacion (Notifies) donde es necesario almacenar informaci´on que puede ser manejada au- tomaticamente. Tambi´en encontramos aqu´ı las relaciones en tiempo de disen˜o como los Links de generaci´on o construccion (Generates), que indican qu´e in- formacion de un modelo es usada para producir o deducir otro, y los Links de sincronizaci´on (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 en esta clasificacion s´olo se resumen un subconjunto de ejemplos, entre ellos: la relaci´on satisface (Satisfies), que indica qu´e propiedad o requerimiento de un artefacto es satisfecha por un modelo; los Links de asignacion (Allocated-To), usados cuando la informaci´on de un artefacto no modelo es asignada a un modelo espec´ıfico que la representa; la relaci´on de realizacion (Performs), que indica qu´e tarea descripta por un artefacto es llevada a cabo por el modelo; las relaciones explica y respalda (Explains y Supports) que dictan qu´e modelo se encuentra explicado y respaldado respectivamente por un artefacto no modelo.

2.4. Generacion de tracelinks

Uno de los principales temas de traceability refiere a los dos enfoques dis- tintos de generaci´on de tracelinks que se pueden presentar o usar durante una transformacion de modelos, estos son la generacion impl´ıcita y la generaci´on expl´ıcita.

La diferencia entre estas dos posibilidades radica en que, por un lado en la generacion impl´ıcita las transformaciones proveen un soporte integrado de traceability, mientras que por el otro, en la generacion expl´ıcita es responsabi- lidad del desarrollador de la transformacion codificar expl´ıcitamente las reglas de traceability que generar´an esta informacion como un modelo mas de salida.

A continuacion se detallan las ventajas y desventajas de cada enfoque segu´n

[3]:

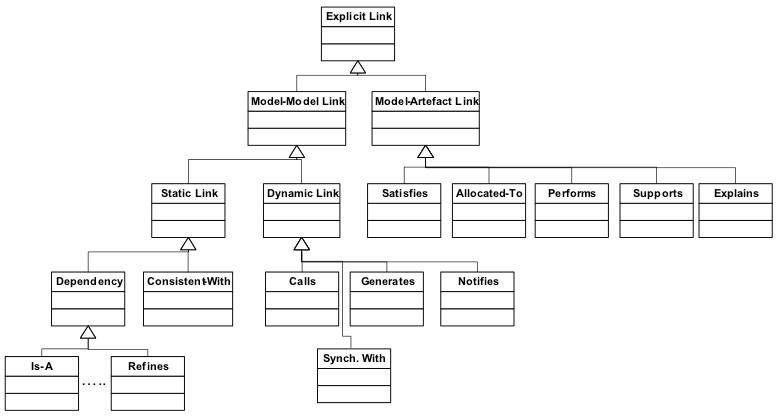


Figura 2.3: Jerarqu´ıa de links expl´ıcitos

2.4.1. Generacion impl´ıcita

Venta jas

X La u´nica y gran ventaja de la generacion impl´ıcita es que no es necesa- rio ningu´n esfuerzo adicional para obtener los tracelinks que relacionan los modelos de entrada y salida, dado que son generados en paralelo y automaticamente con el modelo resultante de la ejecucion de la transfor- macion.

Desventa jas

× El meta-modelo de traceability tiene que ser necesariamente fijo, pero dada la infinidad de enfoques de transformaciones que dan lugar a diferentes definiciones de meta-modelos, lograr un est´andar entre estos diferentes enfoques es muy complejo.

× Dada la poca flexibilidad que se obtiene al hacer uso de un meta-modelo fijo, se presentan los siguientes inconvenientes a tener en cuenta:

- cuando se generan tracelinks para todos los elementos del modelo referenciado, obtener una cantidad considerable de ellos puede vol- verse incomprensible y en consecuencia inu´til. Tambi´en, en el caso de transformaciones de modelos grandes y complejos, podemos en- contrarnos con problemas de rendimiento.

- la configuracion del nivel de trace granularity var´ıa de un escenario de traceability a otro.

- un cliente, por ejemplo aduciendo a motivos de seguridad, puede requerir que para cierta informaci´on de un modelo no se le generen tracelinks.

2.4.2. Generacion Expl´ıcita

Venta jas

X Es posible el tratamiento de traceability como un modelo regular com- plementario al resultado de la transformacion, que se puede obtener por la incorporacion de reglas de transformaci´on adicionales. Por lo cual, la eleccion del meta-modelo es responsabilidad completa del programador de la transformaci´on y no depende del motor de transformaciones.

X Dada la flexibilidad del meta-modelo, el nivel de trace granularity de los

tracelinks es f´acil de adaptar al dominio de la transformacion.

Desventa jas

× Se requiere de un esfuerzo adicional para definir las reglas de transfor- macion espec´ıficas para traceability, que en consecuencia contaminan la implementacion.

× Como la definici´on de las reglas de traceability es responsabilidad del pro- gramador, es propensa a errores y puede llegar a demandar mucho tiempo. Mas aun si pensamos que esta tarea se tiene que repetir para cada trans- formacion que se requiera.

2.5. Estrategias de almacenamiento

Segu´n [7] hay dos tipos principales de estrategias a seguir para almacenar y administrar la informaci´on de traceability, almacenamiento intra-modelo y almacenamiento externo. En la primera, la informacion de traceability se encuentra embebida en los modelos a los que ella hace referencia. En cambio en la segunda, dicha informacion se encuentra almacenada de forma separada de los modelos:

2.5.1. Almacenamiento intra-modelo de tracealinks

Como ya se dijo, bajo esta estrategia la informacion de traceability es alma- cenada dentro de los modelos a los que hace referencia, esto puede ser mediante elementos pertenecientes al modelo o mediante atributos (como etiquetas o pro- piedades).

Es una estrategia sencilla y amigable, pero puede ser muy problematica por varias razones. Si los tracelinks son dirigidos y almacenados solamente en el mo- delo origen, ´estos 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 informacion de traceability es almacenada en ambos modelos, entonces nos encontramos con el desaf´ıo de que dicha informacion se mantenga consistente por cada cambio que se suceda.

A todo lo anterior, se suma el problema de la poluci´on que se genera en el modelo con la informaci´on de traceability, dado que la misma es ajena a su contexto. Dicha polucion puede incluso lograr que el modelo se vuelva imposible de comprender y mantener.

Por otro lado, en un entorno MDE es comu´n que los modelos tengan sus propias representaciones y semanticas, lo cual puede volver mas compleja la tarea de diferenciar la informaci´on de traceability de los objetos que representan el modelo del dominio.

Como resultado de los inconvenientes anteriores, el analisis automatizado de la informacion de traceability se hace muy dif´ıcil. Las principales metodolog´ıas que hacen uso de esta estrategia utilizan construcciones de lenguaje espec´ıficas, por ejemplo determinados tipos de enlaces de traceability estan representados

en los diagramas UML mediante el uso de estereotipos como ✭refines ✮.

2.5.2. Almacenamiento externo de tracelinks

En esta estrategia la informacion de traceability se encuentra almacenada de forma separada a los modelos a los que hace referencia, en un modelo aparte. Esta propuesta tiene dos ventajas claras, la primera es que los modelos origen y destino se mantienen totalmente limpios, con lo cual la polucion del almace- namiento intra-modelo no sucede. Y la segunda, dado que el modelo en donde se almacenan los tracelinks se encuentra definido por un meta-modelo con una semantica clara, logra que el proceso de analisis de la informacion sea mucho mas facil que en la otra estrategia.

Un requisito previo para el almacenamiento externo de los tracelinks, es que los diferentes elementos del modelo tengan identificadores u´nicos, de modo que

los links que los relacionan se puedan resolver inequ´ıvocamente. Un ejemplo es el mecanismo propuesto por MetaObject Facility (MOF ) y EMF en la forma de un identificador xmi.id.

Cap´ıtulo 3

Problemas y Desaf´ıos

En este cap´ıtulo se describe, por un lado, los principales pro- blemas que se presentan en el a´mbito de traceability (organizados por tema segu´n el criterio presentado en [5]), y por otro, los gran- des desaf´ıos que surgen a partir de ellos. Desaf´ıos que au´n hoy, se encuentran abiertos a la investigaci´on y al debate.

´Indice

|  |  |  |
| --- | --- | --- |
| 3.1. | Conocimiento de traceability | 18  18 |
| 3.2. | Capacitacion y certificacion |
| 3.3. | Soporte y evolucion | 19 |
| 3.4. | Sem´antica de los tracelinks | 20 |
| 3.5. | Escalabilidad | 20 |
| 3.6. | Factores humanos | 21 |
| 3.7. | An´alisis de costo-beneficio | 22 |
| 3.8. | M´etodos y herramientas | 22 |
| 3.9. | Procesos | 23 |
| 3.10. | Conformidad | 24 |
| 3.11. | Mediciones y Benchmarks | 24 |
| 3.12. | Transferencia de tecnolog´ıa | 25 |

3.1. Conocimiento de traceability

Problemas

× Existe poco consenso respecto a cu´ales son las mejores t´ecnicas y m´etodos para la aplicaci´on de traceability, escasas anotaciones y documentacion sobre las mejores practicas, sumado a una falta de recursos que provean una buena base de conocimiento.

× Las definiciones sem´anticas no coinciden y las terminolog´ıas son dispares, todo ´esto crea barreras de comunicacion.

Desaf´ıos

X Crear una base de conocimiento en la que se vuelquen las mejores pr´acticas de traceability, una terminolog´ıa estandar y suficiente informaci´on adicio- nal, como por ejemplo casos de estudio.

3.2. Capacitacion y certificaci´on

Problemas

× Se halla muy poca gente competente en la tarea de definicion de tracelinks, paralelamente es escasa la disponibilidad de programas educativos que ensen˜en dicha tarea.

× Existen pocos programas de certificacion, de los cuales pocos incluyen componentes de traceability.

× No hay definido un conjunto de estrategias estandar de traceability.

Desaf´ıos

X Identificar las a´reas de conocimiento y las estrategias principales asociadas a traceability.

X Desarrollar buenos componentes educativos para la puesta en practica de

traceability.

X Desarrollar materiales pedag´ogicos efectivos para educar poniendo ´enfasis en la importancia y evaluacion de los costos-beneficios que conllevan la implementacion de traceability.

3.3. Soporte y evolucion

Problemas

× La informacion precisa, coherente, completa y actualizada sobre tracea- bility es fundamental para diversos a´mbitos y aplicaciones. Sin embargo, las t´ecnicas actuales de captura de tracelinks au´n son realizadas de forma manual y por lo tanto son propensas a errores.

× Para que los tracelinks sean

u´tiles, ´estos deben reflejar la dependencia

actual entre los artefactos. Dado que el costo y esfuerzo para mantenerlos

durante la evolucion del sistema es inmenso, a menudo los tracelinks pasan a encontrarse en un estado err´oneo o incorrecto.

× Las herramientas actuales de administraci´on de requerimientos incluyen caracter´ısticas como suspect tracelinks para ayudar a los analistas a admi- nistrar y entender la evolucion de los tracelinks, pero en la mayor´ıa de los proyectos complejos, el nu´mero de enlaces marcados como suspect trace-

links se vuelven r´apidamente excesivos, minimiz´andose dr´asticamente la utilidad de tal caracter´ıstica.

× Los tracelinks tienen que evolucionar de forma sincronica con los artefactos a los que se encuentran relacionados, sin embargo los sistemas actuales de gestion de cambios y la semantica de los enlaces no son lo suficientemente sofisticados como para acompan˜ar tal evoluci´on.

Desaf´ıos

X Desarrollar t´ecnicas de captura de tracelinks automaticos para artefactos descriptos de forma textual, que sean tan precisos como el proceso manual y, a la vez, mucho m´as efectivos en tiempo y costo.

X Desarrollar la funcionalidad de captura de tracelinks de forma integrada a los Entornos de Desarrollo Integrados o Integrated Development Envi- ronment (IDE).

X Desarrollar sistemas de administraci´on de cambios que efectivamente acom- pan˜en la evolucion de los tracelinks sobre los mu´ltiples tipos de artefactos que existen.

X Desarrollar t´ecnicas que maximicen la reutilizaci´on de los tracelinks cuan- do un codigo existente se vuelva a utilizar en un nuevo producto.

3.4. Sem´antica de los tracelinks

Problemas

× Para efectivamente utilizar y entender las relaciones por debajo de tra- ceability, es necesario definir la semantica de los tracelinks. Sin embargo, definir una formalidad para representar esta semantica no es una tarea facil y puede llegar a quedar acotada a un dominio espec´ıfico, cosa que no

es conveniente.

× Es muy importante para mantener la consistencia en traceability, conocer y establecer el nivel de trace granularity. El problema es que no existe au´n, un modelo claro de costo-beneficio que determine un´ıvocamente cual es el nivel correcto.

Desaf´ıos

X Definir meta-modelos para representar la informaci´on sem´antica de los tracelinks y proveer ejemplos de instancias sobre distintos dominios es- pec´ıficos.

X Desarrollar t´ecnicas y procesos para determinar el correcto nivel de trace granularity de un proyecto.

3.5. Escalabilidad

Problemas

× Las t´ecnicas corrientes de traceability no escalan adecuadamente en pro- yectos largos.

× Las herramientas de visualizaci´on son esenciales para dar ayuda en la com- presion y el uso de la gran cantidad de informacion de los tracelinks. Sin embargo, las t´ecnicas actuales no escalan bien y no son efectivas al presen- tar informacion compleja, dado que carecen de caracter´ısticas sofisticadas

de filtrado, navegacion, consultas, etc.

× Muchos conjuntos de datos industriales son compuestos por largos e ines- tructurados documentos que son dif´ıciles de enlazar mediante tracelinks.

Desaf´ıos

X Obtener conjuntos de datos de escala industrial desde varios dominios y usarlos para investigar la escalabilidad de las t´ecnicas disponibles actual- mente y, si es necesario, crear nuevas aproximaciones que escalen mas eficientemente.

X Desarrollar mecanismos visuales efectivos que permitan la navegaci´on y consulta de un gran nu´mero de tracelinks y los artefactos asociados.

X Desarrollar t´ecnicas escalables para marcar los tracelinks independiente- mente de que el conjunto de datos sea heterog´eneo, de taman˜o considerable y/o se encuentre d´ebilmente estructurado.

3.6. Factores humanos

Problemas

× Los m´etodos autom´aticos de traceability por lo general trabajan produ- ciendo tracelinks candidatos. Sin embargo, el proceso es inu´til si el analista no es capaz de evaluarlos correctamente y lograr diferenciar los buenos de los malos, o si no puede confiar en la completitud y precisi´on de los resul-

tados.

× Idealmente la captura de tracelinks deber´ıa ser invisible durante todo el proceso de desarrollo. Sin embargo, la generacion y el uso de los tracelinks es continuamente interrumpido por interacciones humanas dado que, en los ambientes de desarrollo actuales, au´n no es posible automatizar todo

el proceso.

× Los tracelinks por lo general enlazan artefactos semanticamente diferentes, a su vez estos artefactos son creados por diferentes personas y frecuente- mente escritos en diferentes documentos. Como resultado, los usuarios de un lado de los tracelinks no entienden bien los artefactos que pertenecen

al otro lado de la relacion.

Desaf´ıos

X Basandose en el estudio del uso de las herramientas actuales de traceability, crear nuevas que reu´nan las necesidades practicas que vayan surgiendo.

X Entender las vulnerabilidades a fallas humanas y sus impactos del proceso de traceability, y desarrollar t´ecnicas que ayuden a los analistas a prevenir tales errores y/o minimizar el impacto de los mismos cuando ocurran.

X Desarrollar t´ecnicas que ayuden a las personas a superar las barreras semanticas que se pueden presentar en el proceso de desarrollo completo.

3.7. An´alisis de costo-beneficio

Problemas

× En un escenario de traceability completo, los tracelinks son creados entre artefactos que se encuentran en un nivel bajo de abstracci´on, lo que puede ser deseable para prop´ositos de comprension, sin embargo este nivel tan bajo no es por lo general pr´actico ni efectivo en t´erminos de costo.

× Se carece de un modelo de costo-beneficio que ayude en el analisis que sea necesario realizarse sobre un proyecto cualquiera que implemente y/o use traceability, que gu´ıe, por ejemplo, en el tratamiento y selecci´on de un conjunto de tracelinks potenciales que puedan surgir de un proyecto.

Desaf´ıos

X Definir y desarrollar t´ecnicas de generacion y mantenimiento de la infor- macion de traceability eficientes.

X Definir un modelo de costos que sea pr´actico y aplicable en la generaci´on y el mantenimiento de los tracelinks, que tome en consideracion factores tales como el taman˜o del proyecto, el tiempo, el esfuerzo y la calidad.

X Definir un modelo de beneficios del uso de tracelinks, que tome en consi- deracion la cr´ıtica y la volatilidad, e incorpore el valor logrado gracias al uso de traceability.

3.8. M´etodos y herramientas

Problemas

× Los m´etodos de recuperacion de tracelinks que relacionan artefactos mul- timedia no son lo suficientemente sofisticados y/o soportados. M´as au´n,

se ha realizado poco por incorporar tales t´ecnicas multimedia en las he- rramientas de traceability.

× Que traceability sea automatico es esencial, sin embargo, su puesta en practica se hace dif´ıcil dada la falta de consistencia entre los artefactos relacionados y la imprecision de los modelos.

× El uso de traceability implica todas las siguientes actividades: construccion y/o generacion, evaluaci´on, mantenimiento y uso de los tracelinks. Sin embargo, no existe au´n una sola herramienta que pueda cubrir todas estas tareas por completo.

Desaf´ıos

X Desarrollar m´etodos efectivos que enlacen artefactos multimedia.

X Construir m´etodos y herramientas con altos niveles de automatizacion que soporten el ciclo de vida entero: la construcci´on, la evaluacion, el mantenimiento y el uso de los tracelinks.

X Desarrollar m´etodos de traceability que den soporte a requerimientos no funcionales.

3.9. Procesos

Problemas

× Traceability no se incluye frecuentemente como una parte integral del ciclo de vida del desarrollo.

× Traceability automatico provee una alternativa eficiente en comparacion a la metodolog´ıa manual, sin embargo la practica ha mostrado que algunos conjuntos de datos son dif´ıciles de procesar en la obtencion de tracelinks,

´esto se debe a las inconsistencias en terminolog´ıa y est´andares, la carencia de estructuras, los formatos heterog´eneos, etc.

Desaf´ıos

X Construir modelos de proceso que definan el ciclo de vida completo de

traceability.

X Desarrollar t´ecnicas que permitan evaluar la posibilidad/capacidad que tiene un determinado conjunto de datos de soportar los m´etodos autom´ati- cos de traceability.

3.10. Conformidad

Problemas

× El uso de estandares aseguran procesos consistentes y completos, aun- que abundan los est´andares en el ´ambito de traceability, no est´a claro si los investigadores y/o profesionales estan completamente consientes de la existencia de los mismos.

× En la comunidad de traceability se encuentran eruditos sobre t´ecnicas y procesos del tema, pero tienen poca influencia sobre los contenidos de tra- ceability relacionados con los procesos estandar de ingenier´ıa de software.

× No est

claro c´omo se puede demostrar el cumplimento de los estandares

y regulaciones.

Desaf´ıos

X Establecer un mecanismo de comunicacion para hacer que la comunidad de expertos de traceability dictamine los est´andares relacionados con la tecnolog´ıa.

X Sumarse a la comunidad que define normativas y estandares con el fin de influir y/o desarrollar los est´andares de traceability.

X Como comunidad, desarrollar y promover escenarios validos para probar que las herramientas, las t´ecnicas y las metodolog´ıas de traceability cum- plen con los estandares.

3.11. Mediciones y Benchmarks

Problemas

× Los estudios emp´ıricos se necesitan para demostrar la eficacia de los m´eto- dos de traceability y as´ı, facilitar el trabajo colaborativo y evolutivo entre los investigadores y profesionales. Sin embargo, no se dispone de disen˜os

comunes de experimentacion, metodolog´ıas, ni cotas de referencia (de aho- ra en adelante en ingl´es benchmarks ) para poder realizar dichos estudios.

× Los benchmarks, m´etodos y m´etricas propuestas actuales no han sido va- lidadas a trav´es de estudios o pruebas emp´ıricas.

× No existen o no se han realizado buenas pruebas y benchmarks para tra- ceability y, las que existen, no son compatibles.

× No existen benchmarks que ayuden en la comparacion de m´etodos y t´ecni- cas ya definidas y desarrolladas de traceability.

× La detecci´on de errores en los tracelinks determina la eficacia del producto y del proceso, sin embargo los modelos de detecci´on de errores actuales son primitivos e inv´alidos.

Desaf´ıos

X Definir procesos estandares para la realizacion de estudios emp´ıricos en el ambiente de investigacion de traceability.

X Crear benchmarks para evaluar los m´etodos y las t´ecnicas de traceability.

X Definir medidas que ayuden en la evaluacion de calidad de tracelinks, tanto de uno como de un conjunto de ellos.

X Desarrollar t´ecnicas de evaluacion de m´etodos y procesos de traceability.

3.12. Transferencia de tecnolog´ıa

Problemas

× Uno de los objetivo de la investigacion de traceability es lograr transferir soluciones eficaces a la industria. Sin embargo, en la realidad en la in- dustria son reacios a probar t´ecnicas nuevas donde la eficacia au´n no fue demostrada.

× La carencia de dialogo entre los investigadores y los profesionales limita, por un lado a los investigadores a acceder a un conjunto de datos reales para testear nuevas t´ecnicas, y por otro, inhibe la retroalimentacion de la industria hacia los investigadores.

× Los prototipos de traceability son generalmente disen˜ados para mostrar pruebas de conceptos en el campo de la investigacion. Sin embargo, estos

prototipos no son lo suficientemente rigurosos para el campo de pruebas de la industria.

Desaf´ıos

X Crear una infraestructura y un conjunto de m´etodos relacionados con el fin de organizar el proceso de transferencia de tecnolog´ıa.

X Identificar los casos de estudio exitosos y darlos a conocer con el fin de demostrar la eficacia de rentabilidad y t´ecnica que ofrecen las metodolog´ıas de traceability en el a´mbito industrial.

X Identificar a los usuarios de traceability y definir sus necesidades en t´ermi- nos de calidad, ciclo de vida, comunicaci´on, etc.

X Incorporar las herramientas de traceability que se encuentren a la van- guardia en los IDEs m´as comunes (como Eclipse) y en las herramientas de administracion de requerimientos industriales.

Cap´ıtulo 4

Manos a la obra

En este cap´ıtulo se introduce, desarrolla y fundamenta el es- quema de traceability propuesto en este trabajo de tesis. Del mis- mo modo se expone el prototipo de la herramienta construida que acompan˜a y ejemplifica las ideas del esquema presentado. Dicha he- rramienta, desarrollada como un plug-in del Eclipse, asiste en la generacion de tracelinks sobre transformaciones de modelos existen- tes, y luego, presenta un mapa en el que se muestra la manera en que se relacionan las transformaciones, los links y los artefactos or´ıgenes y resultados de la ejecuci´on de las transformaciones.

´Indice

4.1. Requerimientos 28

4.1.1. Introduccion 28

4.2. El esquema de traceability propuesto 28

4.2.1. Lo que no ofrece 30

4.3. El prototipo 30

4.3.1. La herramienta 30

4.3.2. Arquitectura 31

4.3.3. Transformaci´on QvtoTrace To Trace 33

4.3.4. Editor grafico y tabular TraceEditor 33

4.1. Requerimientos

4.1.1. Introduccion

La idea fundamental es lograr un meta-modelo bien simple en el que se capturen solamente los conceptos mas relevantes de traceability y se almacene la informacion m´ınima necesaria. Gracias a todo lo anterior, y solo as´ı, logra- mos 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 proposito general (2.2.1) que apunte principalmente a dar solucion a los siguientes desaf´ıos:

Lograr un esquema que represente el conjunto de tracelinks independien- temente de como fueron generados, expl´ıcita o impl´ıcitamente (2.4), su trace granularity (2.1) y/o semantica.

Mantener la comunicacion lo mas simple posible, tanto sea la que se rea- liza entre los distintos usuarios con sus respectivos roles, como la de los distintos tipos de herramientas que necesiten aplicar traceability.

Cuando se desee agregar la funcionalidad de traceability mediante el uso de este esquema, la adaptaci´on de la herramienta, cualquiera sea, tiene que ser una tarea sencilla.

Au´n dada la infinidad de artefactos que se pueden presentar, el esquema siempre tiene que ofrecer informaci´on semantica de los mismos, con el fin de facilitar la validaci´on de los links leg´ıtimos.

Lograr identificar un´ıvocamente cada artefacto sobre el que se desee rea- lizar traceability.

4.2. El esquema de traceability propuesto

Con el fin de cumplir con las consignas fundamentales listadas en 4.1.1, se

defini

una propuesta de esquema de traceability que se encuentra representado

en el diagrama de clases del dibujo 4.1.

Como se puede observar, el meta-modelo se puede dividir logicamente en dos partes, la principal en la que se encuentran las abstracciones fundamentales de los elementos m´ınimos necesarios en cualquier herramienta que desee ofrecer traceability, y una secundaria en la que se engloba la configuracion del esquema. Esta configuracion permite ofrecer un meta-modelo flexible a cualquier escena- rio/dominio en el que se desee hacer uso de traceability.

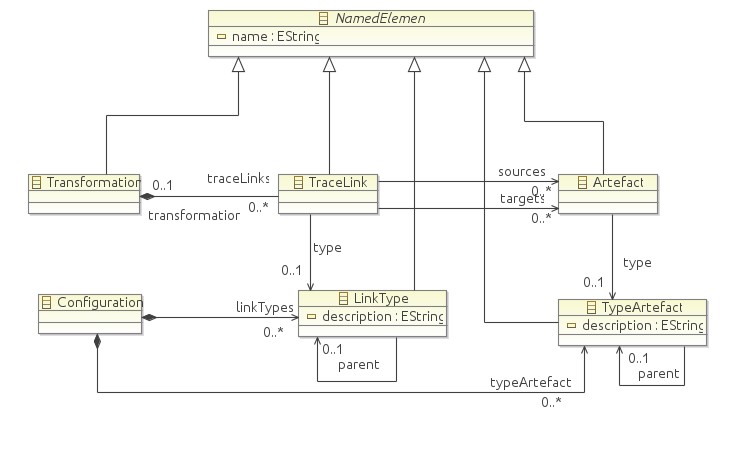


Figura 4.1: Esquema de traceability propuesto

Clases fundamentales

Entre los elementos fundamentales se encuentra la clase TraceLink, la cual

representa un tracelink. Esta clase est

compuesta por un nombre que sirve como

identificador (name), un tipo con el cual se determina su semantica (type), una referencia de la transformacion a la que pertenece (transformation) la cual es optativa porque un tracelink puede o no ser producto de una transformacion, y por u´ltimo dos conjuntos de artefactos: el de los or´ıgenes/fuentes (sources) y el de los objetivos/resultados (targets).

Los artefactos se encuentran representados por la clase Artefact, cuya iden- tificacion tambi´en pasa por su nombre (name), su descripcion tambi´en por un diccionario de clave-valor, y por u´ltimo su semantica tambi´en puede ser definida por un tipo definido en la configuracion.

Por u´ltimo entre los elementos fundamentales, tenemos la clase Transforma- tion que abstrae de las transformaciones un nombre para su identificaci´on, una descripcion v´ıa un diccionario y la coleccion traceLinks que son producto de su ejecucion.

Configuracion del esquema

Esta parte refleja la configuracion necesaria para lograr un esquema adap- table a la infinidad de escenarios en los que puede aplicarse el mismo. Dicha configuracion consiste en la definici´on de dos jerarqu´ıas, una en la que se define la semantica de los artefactos que se van a tracear, y la otra para determinar la semantica de los tracelinks que pueden llegar a crearse. Ambas jerarqu´ıas estan determinadas mediante las clases LinkType y TypeArtefact respectivamen- te, ´estas comparten la siguiente estructura: un nombre (name) que las identifica, un texto que posibilita describirlas (description) y una relaci´on (parent) que aso- cia una clasificacion con su padre, gracias a dicha relacion se logra la estructura jerarquica.

Un ejemplo de una configuracion de tipos de tracelinks puede ser la jerarqu´ıa propuesta en la seccion 2.3.1, que se puede ver siguiendo los diagramas 2.1, 2.2 y 2.3.

4.2.1. Lo que no ofrece

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

funcionalidad de versionado : que permita navegar por los tracelinks y refleje las modificaciones en el tiempo que se fueron realizando sobre los artefactos relacionados, a lo largo de la ejecuci´on de todas las tareas que forman parte de la ingenier´ıa de un software.

m´etodos de detecci´on o informaci´on de errores : que determinen o informen cuando un tracelink es inv´alido.

4.3. El prototipo

En la presente seccion se explica y presenta el prototipo desarrollado que materializa la idea detras del esquema de traceability propuesto en la seccion

4.2.

4.3.1. La herramienta

La idea es lograr una herramienta que, dada la definicion de una transfor- macion acompan˜ada con el conjunto de modelos de entrada/origen y de resulta- do/destino, retorne y muestre de una forma amigable el mapa de tracelinks que

se generaron impl´ıcitamente producto de su ejecuci´on. Tambi´en, que permita la edicion de dicho mapa posibilitando:

crear, editar y eliminar tracelinks

editar artefactos de los modelos origen y destino editar transformaciones

Uso

A continuaci´on se detalla c´omo es la secuencia del uso de la herramienta para lograr lo explicado anteriormente, la imagen 4.2 muestra el resumen de la idea.

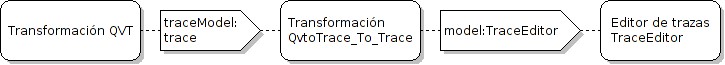


Figura 4.2: Secuencia del prototipo

Para lograr el mapa de tracelinks, dada una definici´on de una transformacion cualquiera Query/View/Transformation (QVT), primero se debe ejecutar dicha transformacion con su modelo origen, con el fin de obtener el modelo resultado. Es importante aclarar que en conjunto con el resultado de la transformacion, QVT genera impl´ıcitamente informaci´on de tracelinks en un modelo definido por el meta-modelo que se muestra en la imagen 4.3.

Luego, se transforma el modelo de tracelinks de QVT al modelo definido por la herramienta (visto en la imagen 4.1), para lograr ´esto, el prototipo incluye una definici´on de una transformaci´on llamada QvtoTrace To Trace.

De esta manera, ya podemos acceder al mapa de tracelinks de la transforma- cion que quer´ıamos mediante la herramienta TraceEditor, navegar sus relaciones y realizar las ediciones que se deseen.

4.3.2. Arquitectura

El prototipo se encuentra definido por dos m´odulos: QvtoTrace To Trace que convierte la informaci´on de trazas QVT en un modelo definido por el meta- modelo que usa el otro m´odulo, y TraceEditor que es el editor grafico y tabular de la informacion de traceability.

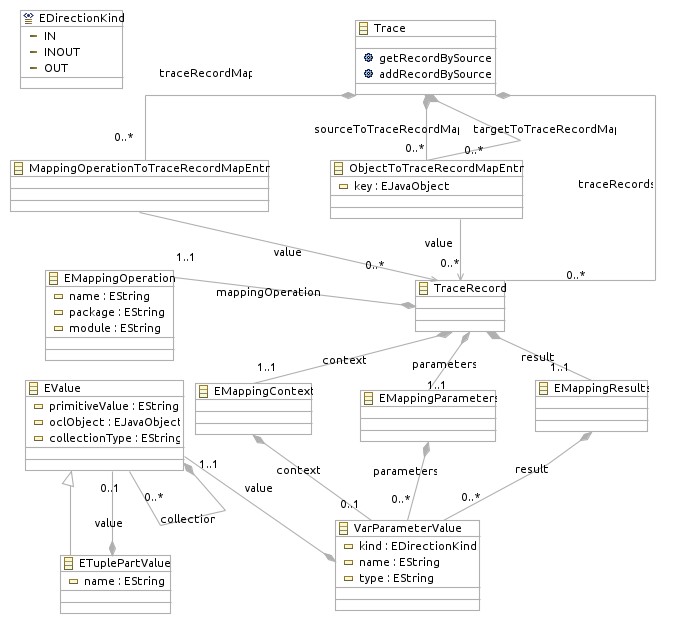


Figura 4.3: Modelo de tracelinks de QVT

El modelo del mapa de tracelinks (o meta-modelo) que usa la herramienta se encuentra definido segu´n el esquema de traceability propuesto y presentado en

4.2. Por otro lado, el prototipo de la herramienta propuesta trabaja solo sobre transformaciones QVT, en el anexo 6.5.3 se presentan algunas ideas de como implementar algo similar para transformaciones Atlas Transformation Language (ATL).

4.3.3. Transformaci´on QvtoTrace To Trace

Como ya se mencion´o, esta tranformacion toma como entrada un modelo de tracelinks definido por el meta-modelo de la imagen 4.3 y retorna un modelo basado en el esquema propuesto de traceability (4.1) llamado TraceEditor como la herramienta que lo usa y cuya definicion se puede ver en la imagen 4.4.

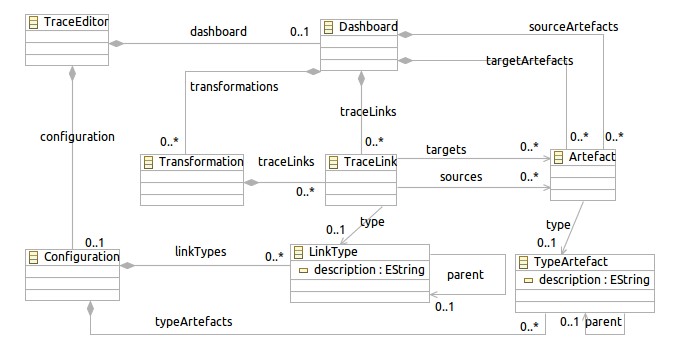


Figura 4.4: Meta-modelo TraceEditor

La transformaci´on se desarroll

con el lenguaje QVT (en la seccion 5.5 se

detalla esta tecnolog´ıa) y su definicion completa se encuentra en el Anexo 6.5.3. Esta transformacion se encuentra documentada en detalle mediante comentarios incorporados en el codigo.

4.3.4. Editor gr´afico y tabular TraceEditor

Este editor grafico y tabular se ejecuta como un plugin de Eclipse, permite presentar todo el mapa de tracelinks que se genera producto de una transfor-

macion que se quiera analizar, as´ı como realizar algunas modificaciones sobre dicho modelo. Permite identificar facilmente, el listado de los artefactos or´ıgenes de una transformacion, el de los destinos o resultados, el conjunto de tracelinks agrupados por la transformaci´on que los genero, y los artefactos or´ıgenes y des- tinos que enlaza cada tracelink. De un artefacto y de un tracelink muestra el nombre que lo identifica y su tipo, y de una transformacion so´lo el nombre.

Est

conformado por los siguientes componentes:

Un editor grafico

Un editor tabular

La definici´on de una perspectiva

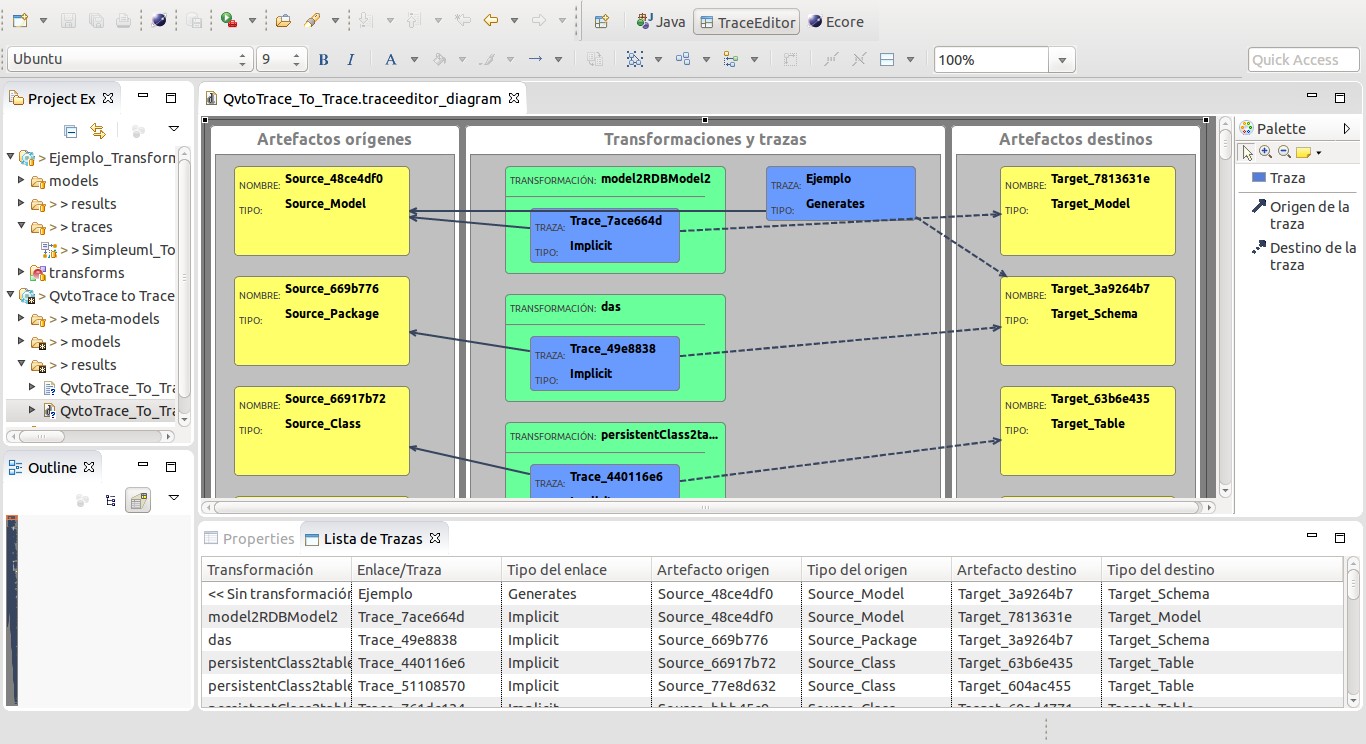


Figura 4.5: Editor grafico y tabular TraceEditor

El editor gr´afico

Este editor que se puede observar en detalle en la imagen 4.6, se compone de un plano principal y una paleta de herramientas. El plano se presenta dividido en 3 zonas en las que se distribuyen los distintos componentes del modelo de tracelinks generado:

Artefactos or´ıgenes , Transformaciones y trazas Artefactos destinos

Las transformaciones se muestran como cajas que contienen los tracelinks que se generaron producto de ellas, y las trazas muestran sus relaciones con artefactos or´ıgenes con una l´ınea continua y con los artefactos destinos con una l´ınea de trazos.

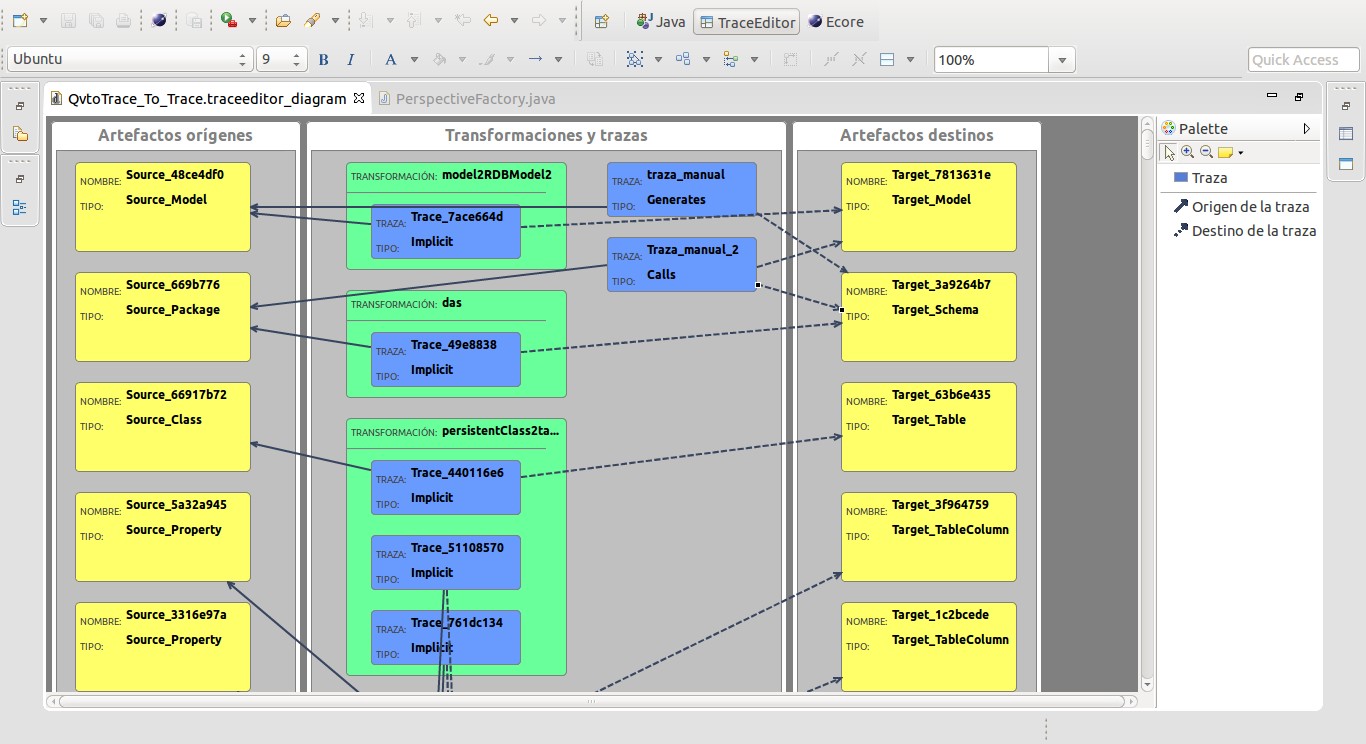


Figura 4.6: El editor gr´afico

El editor permite:

Crear nuevas trazas personalizadas

Modificar los identificadores de las transformaciones

Modificar los identificadores de los tracelinks

Definir nuevas relaciones entre los tracelinks y los artefactos, y modificar las existentes

Eliminar u ocultar cualquier objeto (transformaci´on, artefacto, relaci´on) con el fin de mejorar un analisis, dado que por lo general los modelos generados por la ejecucion de una transformaci´on son muy extensos

El editor fue implementado mediante el framework Graphical Modeling Fra- mework (GMF), el cual ayuda en la construccion de editores gr´aficos para un modelo de datos definido. Una descripcion con mas detalle de esta tecnolog´ıa se encuentra en la seccion 5.3.

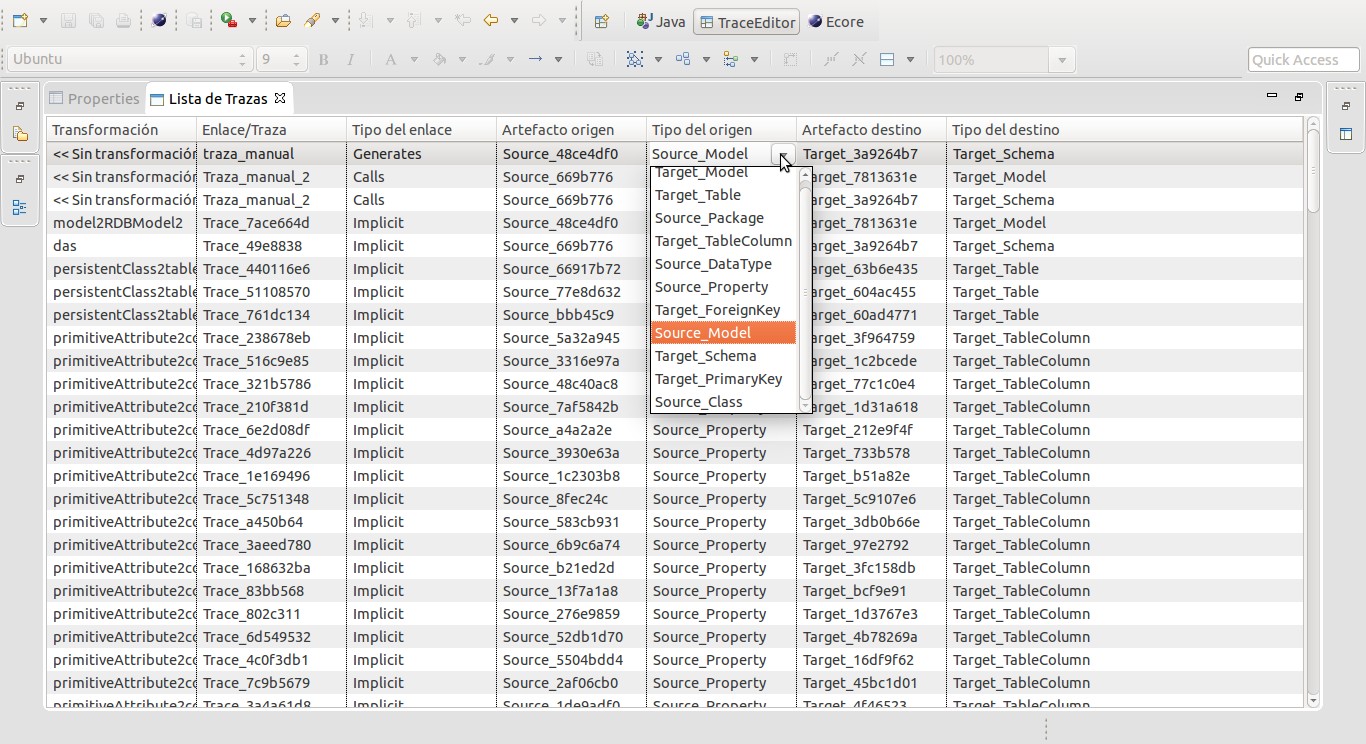


Figura 4.7: El editor tabular

El editor tabular

Este editor muestra el mismo modelo de tracelinks que se presenta en el edi- tor grafico que explicamos anteriormente, pero en una lista ordenada tabulado de acuerdo a los siguiente campos:

Transformaci´on Enlace/Traza Tipo de enlace Artefacto origen

Tipo del artefacto origen

Artefacto destino

Tipo del artefacto destino

Una visualizaci´on del editor se puede observar en la imagen 4.7. Con el editor tabular se pueden realizar las siguientes acciones:

Renombrar una transformacion Renombrar un tracelink Renombrar un artefacto

Definir o cambiar el tipo de un tracelink

Definir o cambiar el tipo de un artefacto

Este editor en conjunto con la perspectiva que se presenta m´as adelante, fueron desarrollados con las herramientas provistas por el Plug-in Development Environment (PDE ) de Eclipse.

La perspectiva

Como se describe en la subseccion de la plataforma Eclipse 5.1.2, una pers- pectiva es una agrupacion de vistas y editores que en su conjunto ayudan en una actividad completa. En este caso, la perspectiva propuesta para nuestro editor TraceEditor fue ofrecer una vista en la que se presentan los dos editores (el grafico y el tabular) en forma simultanea dada la complejidad y el volumen de informacion que puede llegar a presentar un modelo de tracelinks cualquiera.

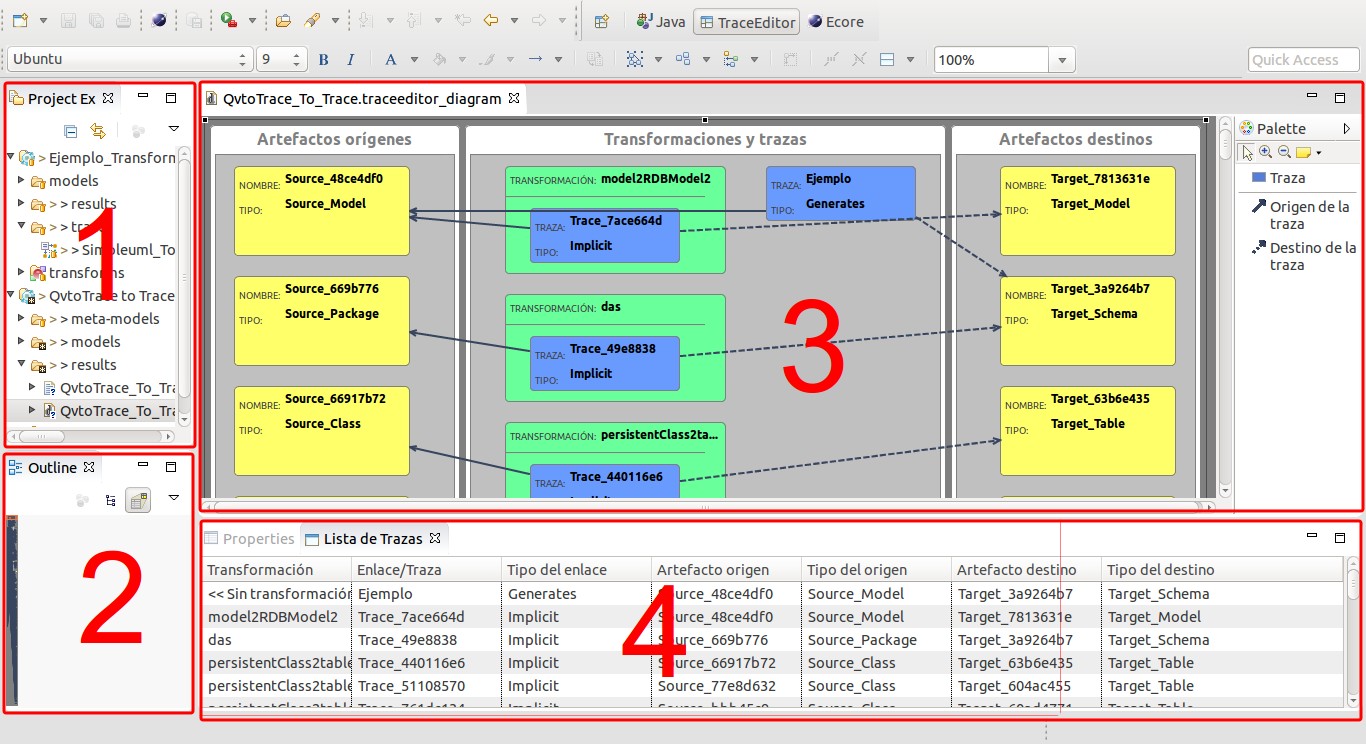


Figura 4.8: Perspectiva del editor

Para una explicacion con mas detalle, se presenta en la imagen 4.8 el Work- bench Eclipse dividido en cuatro zonas que definen la perspectiva desarrollada:

Zona 1 En este sector se encuentra el Project Explorer o Explorador de Proyec- tos, que es una vista de Eclipse que presenta una visual jerarquica de los recursos (proyectos, carpetas y archivos) del Workspace cargados en el Workbench.

Zona 2 Aqu´ı se presenta la vista Outline de Eclipse, la cual se encarga de mostrar un resumen de la estructura de los elementos del editor activo/seleccio-

nado. En el caso que el activo sea el editor gr´afico, esta vista mostrar completo del modelo de tracelinks que se est´e editando y permitira´ con un recuadro de selecci´on que ofrece.

el grafico navegarlo

Zona 3 Esta zona es el a´rea de edicion o editores, es el espacio del Workbench Eclipse asignada para que se muestren todos los editores Eclipse de cualquier tipo. Nuestro editor gr´afico se abrir´a aqu´ı cuando seleccionemos en el Project Explorer un recurso de tipo diagrama TraceEditor.

Zona 4 En esta a´rea se muestran, en primer plano el editor tabular titulado Lista de Trazas, presentado anteriormente en esta misma subseccion 4.3.4, y en segundo plano, la vista de propiedades de Eclipse (Properties) que nos permite ver todas las propiedades de cualquier elemento seleccionado en el editor que se encuentre activo.

Cap´ıtulo 5

Descripci´on de tecnolog´ıas

En este cap´ıtulo se realiza una breve introduccion sobre cada una de las tecnolog´ıas que fueron utilizadas para la implementacion del prototipo que hace uso del esquema propuesto de traceability.

´Indice

5.1. Eclipse 41

5.2.

|  |  |  |
| --- | --- | --- |
| 5.1.1. | El proyecto | 41 |
| 5.1.2. | La plataforma Eclipse | 42 |
| 5.1.3. | Resumen y mas informaci´on | 44 |
| Eclipse Modeling Framework 44 | | |
| 5.2.1. | El framework EMF | 44 |
| 5.2.2. | El (Meta) modelo Ecore | 46 |
| 5.2.3. | Beneficios y mas informaci´on | 47 |
| Graphical Modeling Framework 47 | | |
| 5.3.1. | El framework GMF | 47 |
| 5.3.2. | Arquitectura | 47 |
| 5.3.3. | Modelos y flujo de trabajo | 48 |
| 5.3.4. | Mas informaci´on | 50 |
| Atlas Transformation Language 50 | | |
| 5.4.1. | ¿Qu´e es ATL? | 50 |
| 5.4.2. | Conceptos de ATL | 50 |
| 5.4.3. | El lenguaje ATL | 51 |
| 5.4.4. | Mas informaci´on | 51 |

5.3.

5.4.

5.5. QVT 52

5.5.1. Introduccion a QVT 52

5.5.2. Lenguajes QVT 52

5.5.3. Mas informaci´on 53

5.1. Eclipse

En la presente secci´on se introduce al lector en el entorno Eclipse, proyecto base sobre el cual el prototipo va a funcionar y/o ejecutarse. Se describe el proyecto y se detalla brevemente la plataforma con sus principales componentes y/o funcionalidades.

5.1.1. El proyecto

Segu´n se presenta en [19, 17] Eclipse es un proyecto de desarrollo de software de c´odigo abierto, cuyo prop´osito es proveer una plataforma de herramientas altamente integradas para la construccion, implementacion y administracion de software a lo largo de todo su ciclo de vida.

El proyecto nu´cleo es un framework gen´erico para la integraci´on de herra- mientas en conjunto con un entorno de desarrollo Java que ya se incluye para usarlo. Otros proyectos extienden el framework nu´cleo para soportar distintos tipos de herramientas y ambientes de desarrollo espec´ıficos. Estos proyectos en Eclipse estan implementados en Java y pueden ser ejecutados en muchos sistemas operativos.

La comunidad Eclipse tiene mas de 200 proyectos, los cuales pueden organi- zarse conceptualmente dentro de las siguientes 7 categor´ıas:

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 comer- ciales, 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 construcci´on de IDEs, el mismo ha sido descripto como “un ambiente para cualquier cosa y nada en par- ticular”. La plataforma propone simplemente la estructura basica del IDE, luego mediante la definici´on de herramientas espec´ıficas que ampl´ıan y se conectan al framework terminan especificando un IDE particular de forma colectiva.

Arquitectura de plugins

En Eclipse la unidad basica de funcionamiento, o sencillamente un com- ponente, es llamado plug-in. Tanto la plataforma misma de Eclipse como las herramientas que la extienden est´an compuestas por ´estos plug-ins. Una herra- mienta sencilla puede consistir en un simple plug-in, pero las m´as complejas por lo general estan divididas en varios de ´estos.

Un plug-in incluye todo lo necesario para la ejecucion del mismo, ´esto puede ser: c´odigo Java, imagenes, textos, etc. Tambi´en 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 dis- ponibles, sin embargo ´estos son solo activados cuando es necesaria su ejecucion con el fin de no ralentizar el arranque.

Workspace

Las herramientas integradas en Eclipse trabajan con archivos y carpetas ordinarias, pero tambi´en disponen de una Interfaz de programacion de aplica- ciones 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 forma en que Eclipse representa un archivo y/o una carpeta, a los que provee de capacidades adicionales como ser detectores de cambios (change listeners ), marcadores como por ejemplo las listas por hacer (to-do list ) y/o mensajes de errores, y un registro de historia de cambios.

Pro ject 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 cuan- do 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 est´a compuesta de vistas, editores y perspectivas.

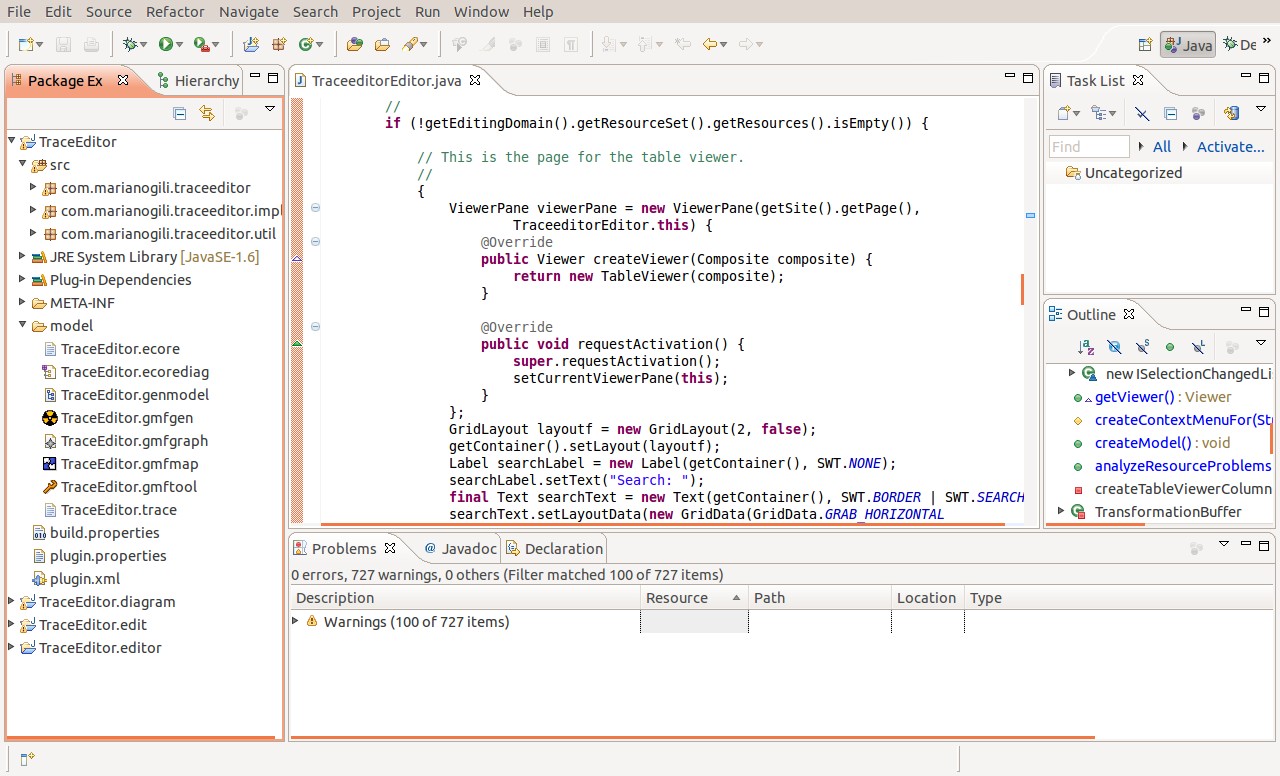


Figura 5.1: Elcipse Workbench

Editores

objectos.

E´ stos permiten al usuario abrir, editar y guardar distintos tipos de

Vistas Proveen informacion de algu´n objeto sobre el que el usuario se en- cuentra trabajando en el Workbench. Por ejemplo, una vista puede asistir a un

usuario editor dando informacion sobre el documento que se est

editando.

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

5.1.3. Resumen y m´as informaci´on

En conclusion, la plataforma Eclipse proporciona un nu´cleo de elementos basicos y un conjunto de APIs gen´ericas, como ser el workspace y el workbench, y varios puntos de extension a trav´es del cual se pueden integrar nuevas fun- cionalidades. A trav´es de estos puntos de extension, las herramientas escritas como plug-ins independientes pueden extender la plataforma Eclipse.

Para obtener mas informacion sobre el proyecto Eclipse ver [17, [www.eclipse. org],](http://www.eclipse.org) y para informacion t´ecnica consultar [18, Eclipse Platform Technical Over- view].

5.2. Eclipse Modeling Framework

A continuacion se describe una de las tecnolog´ıas del proyecto Eclipse usada para la implementaci´on de una parte del prototipo, el framework EMF [15].

5.2.1. El framework EMF

El EMF es un framework de modelado y generacion de c´odigo para herra- mientas de construccion y otras aplicaciones basadas en un modelo de datos estructurados. A partir de una especificaci´on de un modelo descrito en XML Metadata Interchange (XMI), EMF provee herramientas y un entorno de ejecu- cion para producir un conjunto de clases Java para el modelo y un conjunto de clases adaptadoras que permiten la visualizaci´on y la edicion de dicho modelo.

EMF se conforma de tres partes fundamentales:

EMF EMF.Edit EMF.Codegen

EMF

El framework EMF base incluye un meta-modelo (Ecore) para la descrip- cion de modelos y un entorno de ejecucion para dichos modelos que incluye las funcionalidades de notificaci´on de cambios, persistencia por defecto v´ıa seriali- zacion XMI y una librer´ıa muy eficiente para la manipulaci´on de objectos EMF gen´erica.

EMF.Edit

El framework EMF.Edit incluye clases gen´ericas reusables para la construc- cion de editores de modelos EMF. Este ofrece:

Clases proveedoras de contenido y etiquetas, acceso a propiedades or´ıgenes y otras clases convenientes que permiten a los modelos EMF ser visuali- zados mediante entornos graficos estandares de escritorio (v´ıa JFace) y/o hojas de propiedades.

Un framework de comandos que incluye un conjunto de clases que im- plementan dichos comandos para la construccion de editores con soporte

´ıntegro de deshacer y rehacer (undo/redo).

EMF.Codegen

La funcionalidad de generaci´on de c´odigo EMF es capaz de producir todo lo necesario para la construccion de un editor de un modelo EMF completo. E´ sta incluye una Interfaz de Usuario Grafica o Graphical User Interface (GUI) desde la cual las opciones de construccion pueden ser especificadas y los gene- radores invocados. Dicha funcionalidad de generacion aprovecha el componente de Eclipse Java Development Tool (JDT ).

Tres niveles de generacion de codigo son soportados: Model

Adapters

Editor

Model Proporciona la implementacion de interfaces y clases Java para to- das las clases del modelo, ademas la implementacion de una clase f´abrica y el paquete.

Adapters Genera la implementaci´on de las clases, llamadas ItemProviders, que se adaptan a las clases del modelo para su edici´on y visualizaci´on.

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 personalizacion.

5.2.2. El (Meta) modelo Ecore

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

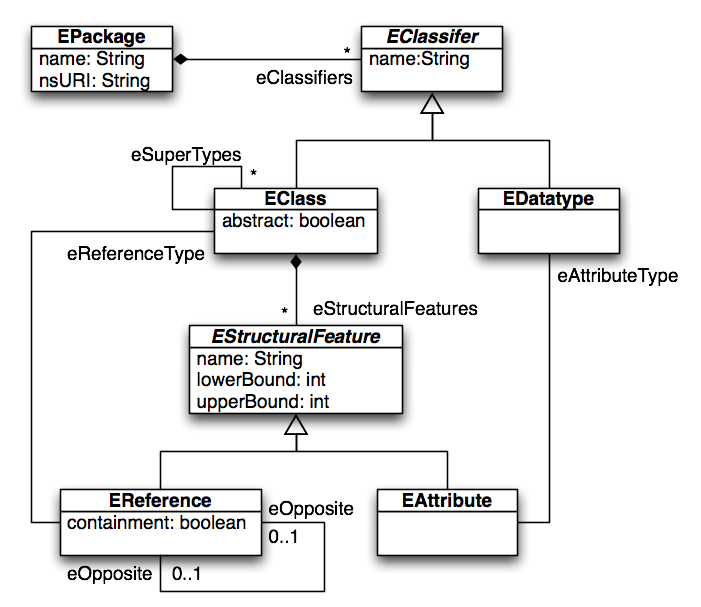


Figura 5.2: Modelo Ecore simplificado

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 relacion entre clases. Tiene un nombre, una marca que indica si es una composicion, 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 m´as informaci´on

Ademas de incrementar la productividad gracias a la generaci´on automatica de codigo, el uso del framework EMF provee otros beneficios: la notificaci´on de cambios, una funcionalidad de persistencia (como la serializacion XMI entre otras), y una API gen´erica reflexiva muy eficiente que sirve para la manipulaci´on de objetos EMF. Sin embargo, uno de los beneficios m´as importantes de este framework es que EMF provee las bases para la interoperabilidad con otras herramientas y aplicaciones que se basan en ´el.

Mas informacion sobre el framework EMF se puede obtener en [15] y [19].

5.3. Graphical Modeling Framework

En la presente secci´on se analiza el framework GMF, el mismo fue usado para el desarrollo de la parte grafica del editor de tracelinks del prototipo.

5.3.1. El framework GMF

GMF es un framework para la construcci´on de editores gr´aficos de mode- lado para la plataforma Eclipse, por ejemplo editores UML, Ecore, de procesos

de negocios, diagramas de flujo, etc. Este framework est

conformado por un

componente de generaci´on (GMF Tooling) y un entorno de ejecuci´on (GMF Run- time) que ayudan en el desarrollo de editores graficos que se basan en EMF y Graphical Editing Framework (GEF ).

GMF Tooling incluye por un lado editores para crear y/o modificar los mo- delos que describen los aspectos de notacion, semantica y utilidad de un editor grafico, y por otro, un generador que da como resultado la implementacion del editor definido.

GMF Runtime Los plug-ins generados con GFM-Tooling dependen de este componente que es el entorno de ejecuci´on 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 mues- tran las dependencias existentes entre el editor grafico generado, el entorno GMF

Runtime, EMF, Graphical Editing Framework (GEF ) y la plataforma Eclipse. Como se puede ver, el editor grafico GMF depende del componente GMF Runti- me, y a su vez hace uso directo de EMF, Graphical Editing Framework (GEF ) y la plataforma Eclipse.

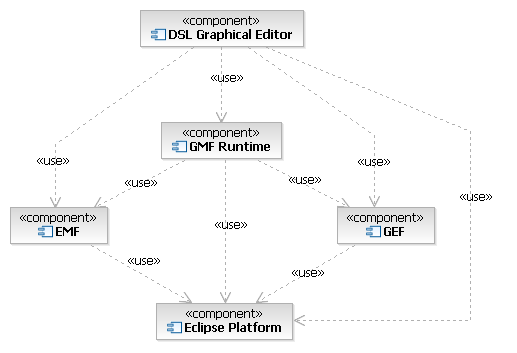


Figura 5.3: Arquitectura GMF

5.3.3. Modelos y flujo de traba jo

En el diagrama 5.4 se encuentran los principales componentes y modelos usados durante el desarrollo de un editor gr´afico mediante el framework GMF. Entre ellos se encuentra el concepto de modelo de definici´on gr´afica, el cual contiene toda la informacion relacionada con los elementos graficos que for- maran parte del editor (figuras, nodos, enlaces, etc), pero que no tiene ninguna conexion directa o no es dependiente de ninguno de los componentes del modelo

de dominio para el cual ofrecer

la representacion y/o edicion. Tambi´en tenemos

el modelo de definici´on de herramientas que es opcional y usado para el disen˜o de la paleta, el menu´ y las barras de herramientas.

Tanto la definicion grafica como la de las herramientas pueden funcionar para modelos variados de dominios, ´este 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 asignacion

o mapeo, que se encuentra separado y vincula las definiciones graficas y las herramientas con los correspondientes modelos de dominios seleccionados.

Una vez que los mapeos o asignaciones son definidos, GMF dispone de un modelo de generacion el cual posibilita la definici´on de los detalles para la implementacion de la siguiente y u´ltima fase, la generacion del editor.

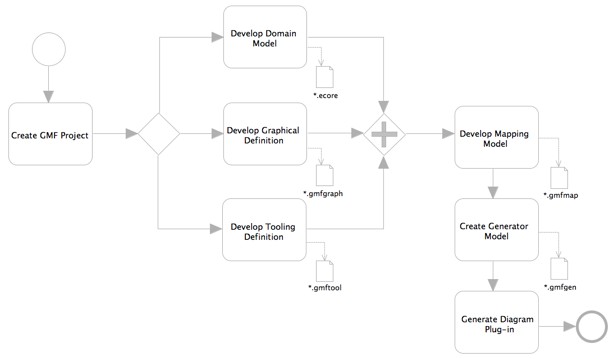


Figura 5.4: Flujo de trabajo de GMF

Flujo de traba jo

1. Creaci´on del modelo del dominio, en este modelo se define la informacion no gr´afica que gestiona el editor.

2. Creaci´on del modelo de definicion grafica, en el que se definen los elementos graficos que se mostrar´an/presentar´an en el editor.

3. Creaci´on del modelo de definicion de herramientas, para definir la paleta, el menu´ y las barrar de herramientas.

4. Creaci´on del modelo de asignacion grafica, que es el que define la corres- pondencia entre los elementos del modelo del dominio con los elementos graficos del modelo de definicion grafica.

5. Generaci´on del editor grafico.

6. Mejora del editor grafico por medio de la edicion del c´odigo del plug-in generado.

5.3.4. M´as informaci´on

Mas informacion sobre el framework GMF se puede obtener en [20], [21] y

[22].

5.4. Atlas Transformation Language

A continuacion se introduce a la tecnolog´ıa ATL, que no solo refiere a un lenguaje de transformaci´on de modelos como su nombre puede confundir, sino que ademas trata de un conjunto de herramientas de desarrollo construidas para ser ejecutadas sobre la Plataforma Eclipse (introducida en 5.1.2).

5.4.1. ¿Qu´e 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 h´ıbrido de la programaci´on declarativa e imperativa, dado que aunque el estilo declarativo es el m´as conveniente para la definicion de las transformaciones, ATL tambi´en provee la posibilidad de construcciones imperativas con el fin de facilitar la especificacion 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 estandar con el fin de facilitar el desarrollo de las transformaciones mediante este lenguaje como el resaltado de sintaxis, el auto-completado de codigo, un depurador, entre otras.

5.4.2. Conceptos de ATL

Un modelo fuente se transforma en un modelo destino gracias a la definici´on de una transformacion escrita en ATL, la cual a su vez es un modelo en s´ı mismo. Los modelos fuente, destino y la definicion de la transformaci´on, conforman sus respectivos meta-modelos y, a su vez, todos los meta-modelos se ajustan a MetaObject Facility (MOF ) o Ecore. Esta relacion se puede observar en la imagen 5.5.

Una transformacion ATL es unidireccional, o sea que trabaja sobre un mode- lo fuente de solo lectura y produce un modelo destino de solo escritura. Durante

la ejecucion de una transformaci´on, el modelo fuente puede ser navegado pero no cambiado, en cambio el modelo destino no puede ser navegado.

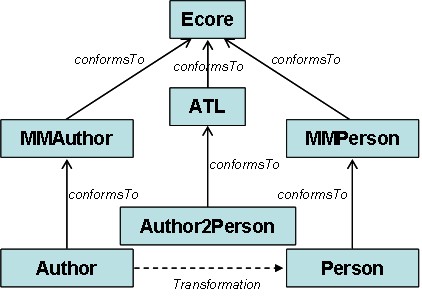


Figura 5.5: Modelos, transformaciones y sus meta-modelos

5.4.3. El lengua je ATL

En este lenguaje de transformaciones modelo a modelo o Model-to-Model Transformation (MMT), las operaciones de transformaci´on son especificadas mediante m´odulos ATL. Cada modulo ATL permite a un desarrollador especifi- car la forma de producir un conjunto de modelos resultados desde un conjunto de modelos fuentes/or´ıgenes.

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

Por u´ltimo ATL ofrece la posibilidad de desarrollar librer´ıas independientes que pueden ser importadas a lo largo de diferentes tipos de unidades ATLs. E´ sto nos da una forma conveniente de factorizar el codigo que va a ser usado por muchas unidades ATL.

5.4.4. M´as informaci´on

Mas informacion sobre la tecnolog´ıa ATL se puede encontrar en la seccion de documentacion de la p´agina del proyecto [23, [www.eclipse.org/atl/].](http://www.eclipse.org/atl/)

5.5. QVT

Finalizando este cap´ıtulo dedicado a las tecnolog´ıas que forman parte del trabajo de tesis, se introduce al lenguaje estandar para transformaciones espe- cificado por la Object Management Group (OMG ), el lenguaje QVT.

5.5.1. Introduccion a QVT

QVT, como ya se mencion

antes, es el est´andar de Object Management

Group (OMG ) para la definicion de transformaciones. La especificaci´on de QVT es h´ıbrida, como en el caso de ATL, relacional (o declarativa) y operacional (o imperativa). Comprende tres diferentes lenguajes MMT: dos lenguajes declara- tivos llamados Relations y Core, y un tercer lenguaje, de naturaleza imperativa, llamado Operational Mappings. Puede observarse la relacion de los meta-modelos de cada uno de los lenguajes en la figura 5.6.

La naturaleza h´ıbrida de este estandar fue introducida para abarcar distintos tipos de usuarios con diferentes necesidades, requisitos y h´abitos.

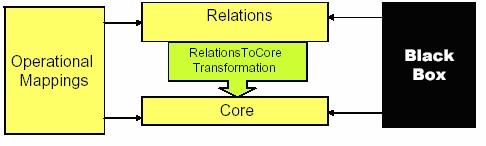


Figura 5.6: Relaci´on entre los meta-modelos QVT

5.5.2. Lengua jes QVT

QVT declarativo

Relations Es un lenguaje amigable para el usuario, que soporta pattern mat- ching complejo y la creaci´on de templates para objetos. Tiene creacion de tra- celinks impl´ıcita. Tambi´en incluye la propagaci´on de cambios, ya que provee un mecanismo para identificar elementos del modelo destino.

Core Es un lenguaje pequen˜o con un soporte de pattern matching acotado. Tiene traceability expl´ıcito y las tracelinks pueden crearse y borrarse como cual- quier otro objeto. Es igual de poderoso semanticamente que el lenguaje Relations pero trabaja a un nivel mas bajo de abstraccion. Esta propuesta absolutamen- te minimal lleva a que el lenguaje Core sea usado como un “assembler” de los lenguajes de transformacion.

QVT imperativo

Operational Mappings Este lenguaje se especific

como una forma estandar

para proveer implementaciones imperativas de transformaciones unidirecciona- les. Al igual que el lenguaje Relations, dispone de creaci´on de tracelinks impl´ıcita sobre el mismo modelo de tracelinks.

Implementaciones Black-box Estas implementaciones de caja negra, per- miten escribir transformaciones en otro lenguaje distinto a QVT. Una imple- mentacion de este tipo no tiene relacion explicita con el lenguaje Relations. Aunque una caja negra podr´ıa implementar una Relation, la misma debe ser responsable de mantener las tracelinks entre los elementos relacionados.

5.5.3. M´as informaci´on

Para obtener mas informacion introductoria consultar [25]. Para mas detalles t´ecnicos y ejemplos se puede leer el documento de especificacion MetaObject Facility (MOF ) Query/View/Transformation [24].

Cap´ıtulo 6

Traba jos relacionados

En este cap´ıtulo se realiza una breve introducci´on de un con- junto de trabajos que tienen relaci´on con el tema que se desarrolla en la presente tesis, trabajos que encontr´e durante el proceso de aprendizaje e investigaci´on.

´Indice

6.1. Un motor de traceability de transformaci´on de mo-

delos en la Ingenier´ıa de Software 55

6.2. Un Framework de Traceability dirigido por mode-

los para el desarrollo de Software Product Line (SPL) 57

6.2.1. Meta-modelo de traceability 57

6.2.2. Arquitectura 58

6.3. Integracion de herramientas Case 59

6.4. Framework gen´erico de extraccion de datos de tra- ceability 60

6.5. Traceability local y global 62

6.5.1. Meta-modelo de Traceability Local 62

6.5.2. Meta-modelo de Traceability Global 63

6.5.3. ¿Como trabaja el framework? 64

6.1. Un motor de traceability de transforma- cion de modelos en la Ingenier´ıa de Soft- ware

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 generacion de las tra- celinks del c´odigo que pertenece a la transformacion. El mismo trabaja sobre transformaciones escritas en Java usando la API EMF [15]. A continuaci´on se listan sus principales caracter´ısticas:

El codigo de generacion de tracelinks no es intrusivo en el c´odigo de la transformacion;

La generacion de tracelinks tiene que ser activada expl´ıcitamente por el disen˜ador de la transformaci´on;

Los modelos de las tracelinks se encuentran aislados de los modelos origen y destino que forman parte de la transformacion;

Los modelos de las tracelinks pueden ser usados a diferentes niveles de granularidad.

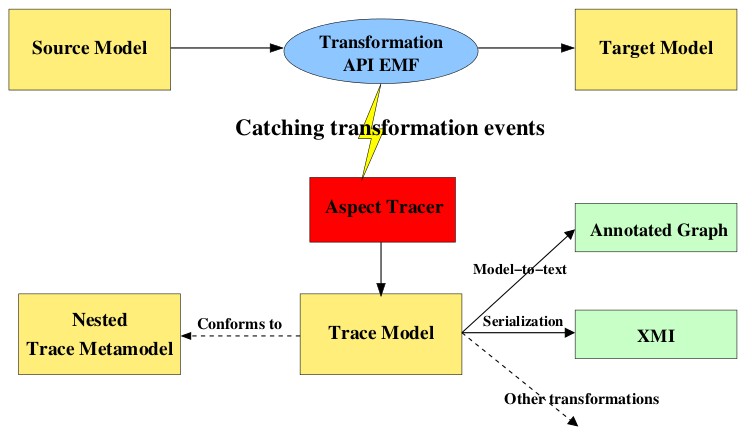


Figura 6.1: Arquitectura de la herramienta ETraceTool

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

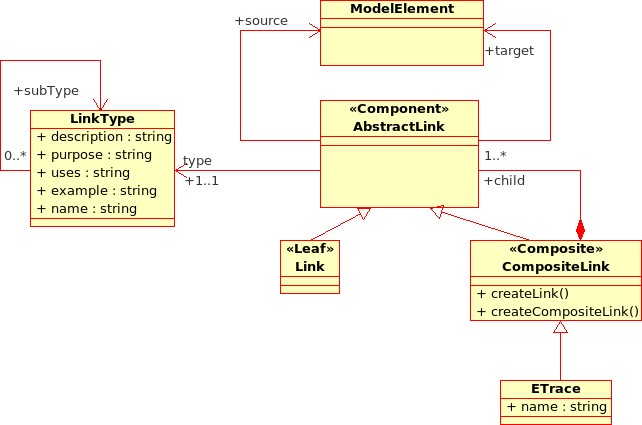


Figura 6.2: Meta-modelo de tracelinks anidado

El fundamento del disen˜o del meta-modelo de tracelinks anidado propues- to es para el caso en el que se presente una operacion de transformacion que llama o hace uso de otra transformacion. En este caso el enlace compuesto per- mite separar las operaciones de bajo nivel (creacion, eliminacion, etc) de las operaciones de alto nivel (como una operacion de refactorizacion).

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

El framework presentado en [16] provee una plataforma abierta y flexible para crear enlaces de tracelinks entre distintos artefactos del desarrollo de una L´ınea de Producto de Software o Software Product Line (SPL). Pero, dado que el disen˜o del framework es gen´erico, ´este tambi´en puede aplicarse o usarse en otras a´reas del desarrollo SPL. El mismo ha sido disen˜ado e implementado basandose en el uso de t´ecnicas dirigidas por modelos. El meta-modelo de traceability descripto en el dibujo 6.3 permite definir distintos tipos de enlaces de tracelinks entre los artefactos.

Las principales funcionalidades ofrecidas por el framework son las siguientes:

1. Creaci´on y mantenimiento de los enlaces de tracelinks de artefactos exis- tentes (modelos UML, codigo fuente, etc);

2. Almacenamiento de los enlaces de tracelinks mediante el uso de un repo- sitorio;

3. Bu´squeda de enlaces de tracelinks espec´ıficos usando consultas de trace- links predefinidas o personalizadas;

4. Visualizaci´on flexible de los resultados de las consultas de tracelinks por medio de diferentes tipos de vistas, como vista de a´rbol, grafo, tabla, etc.

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 m´etodo de dicha clase. El artefacto es identificado mediante la propiedad resourceId.

Un TraceLink es la abstraccion de una transicion de un artefacto a otro. Cada TraceableArtefact tiene asignada una instancia de TraceableArte-

factType, ´estos se pueden encontrar agrupados de forma jer´arquica.

Analogo a los tipos de los artefactos los TraceLinks tambi´en tienen un tipo (TraceLinkType), teniendo en cuenta que la semantica de una relaci´on entre dos artefactos puede variar.

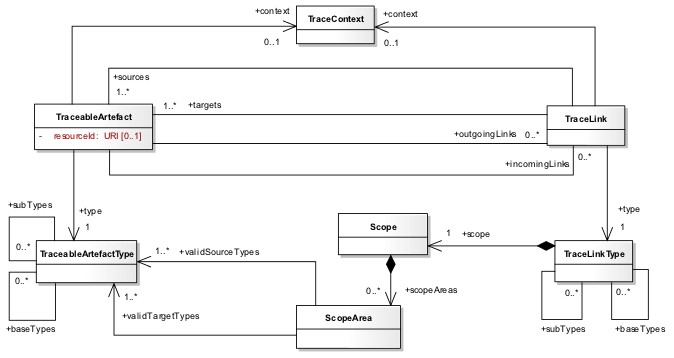


Figura 6.3: Meta-modelo de traceability

Informacion adicional de artefactos y enlaces puede ser modelada median- te un contexto, el cual se encuentra representado por la clase TraceCon- text.

Las restricciones que determinan el conjunto de artefactos validos, sobre los cuales los tipos de enlaces tambi´en son v´alidos 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 princi- palmente en t´ermino de cuatro m´odulos. Cada uno de los cuales implementa una de las funcionalidades m´as importantes del framework. Dichos modulos se detallan a continuacion:

1. Trace Register: este m´odulo provee mecanismos para crear y mantener

(actualizar, eliminar y buscar) los enlaces de tracelinks;

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 tracelinks espec´ıficos que se encuentran previamente almacena- dos;

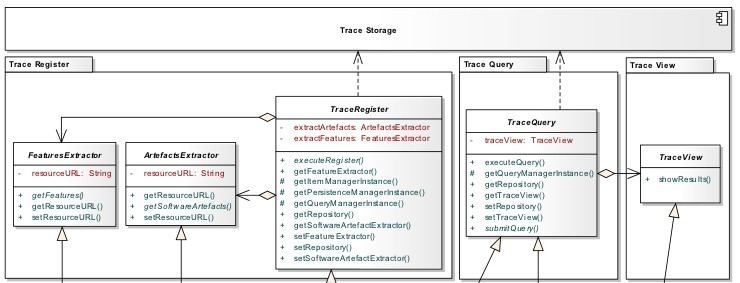


Figura 6.4: Arquitectura del Framework de Traceability

4. Trace View: usado espec´ıficamente para la representaci´on visual de los enlaces resultados de una consulta realizada.

6.3. Integracion 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 genera una variedad muy amplia de artefactos de software (documentos de textos, hojas de c´alculo, resultado de pruebas, modelos, graficos, etc). Estos artefactos, aunque en esencia se encuentran relacionados logicamente, 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 logicas nombradas se pierden, o mejor dicho no existen o pasan desapercibidas en la practica. En otras palabras, se nos presenta el problema de la imposibilidad de traceability que encontramos en la mayor´ıa de las herramientas Computer Aided Software Engineering (CASE) actuales.

Como soluci´on a este problema, se propone un ambiente de integracion para las herramientas CASE al que llamaron TiE - Tool Integration Environment, el cual basa su integraci´on en la creaci´on de enlaces de tracelinks entre los artefactos de las distintas herramientas.

6.4. Framework gen´erico de extraccion de da- tos de traceability

En [13] se propone un framework gen´erico de traceability que toma como entrada una transformacion de modelo y le aumenta arbitrariamente su funcio- nalidad con un mecanismo de traceability. En el dibujo 6.5 se muestra un pano- rama de alto nivel de la arquitectura propuesta. E´ sta se basa en una interfase gen´erica que provee un punto de conexi´on para cualquier motor de transfor- macion de modelos, mediante una API que se ofrece al ingeniero que conecta su motor de transformaci´on con el motor de traceability (oAW connector, QVT connector). Como resultado el motor de transformaci´on incluye la funcionalidad de traceability. El modelo de datos que usa el framework es el lenguaje espec´ıfico de dominio para traceability que se denomina Trace-DSL.

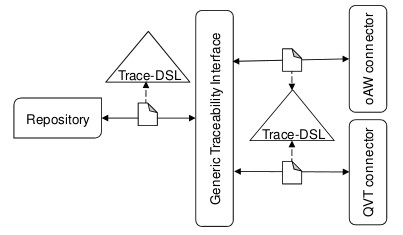


Figura 6.5: Resumen de la arquitectura del Framework Gen´erico de Traceability

Trace-DSL que se detalla en el dibujo 6.6, tiene como elemento ra´ız Tra- ceModel. Artefact representa cualquier producto traceable generado durante el proceso de desarrollo, esto puede ser un requerimiento, una clase o un compo- nente, como por ejemplo el m´etodo de una clase. Todo artefacto es identificado un´ıvocamente mediante un identificador universal (UUID). TraceLink es una abstraccion de una transici´on de un artefacto a otro dirigida por la relacion desde-hacia entre artefactos origen y destino. TraceLink puede ser de una de las siguientes cuatro instancias: CreateTraceLink, QueryTraceLink, UpdateTrace- Link y DeleteTraceLink.

Para la asignacion 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. Ademas de lograr una solucion simple al tipado de artefactos y enlaces, se obtiene un mecanis-

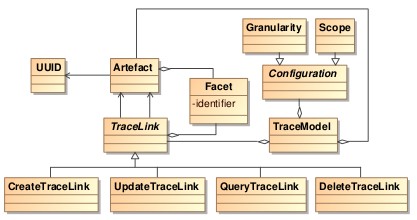


Figura 6.6: Lenguaje espec´ıfico de dominio para traceability

mo facilmente extensible y configurable al contexto donde se necesite aplicar

traceability.

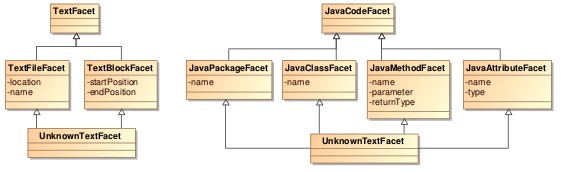


Figura 6.7: Faceta para traceability de codigo fuente

La configuraci´on 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 configuracion de la granularidad consiste en la especificacion de qu´e tipos (definidos por las facetas) de artefactos y enlaces seran trazados para un esce- nario de traceability particular. En cambio la configuracion del alcance implica restringir los datos de traceability a una combinaci´on espec´ıfica de valores. En otras palabras la primera solo chequea la existencia de facetas, mientras que la segunda adicionalmente examina las propiedades especificas de las mismas. 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] se emplea la idea de separacion del proceso de traceability en los siguientes niveles: traceability en lo pequen˜o y traceability en lo grande refiri´endose a los mismos como traceability local y traceability global respectivamente.

6.5.1. Meta-modelo de Traceability Local

Este meta-modelo toma las tracelinks de la entrada y la salida de una u´nica

transformacion. El meta-modelo est

basado en el meta-modelo de tracelinks

presentado en [9] y se muestra en el dibujo 6.8.

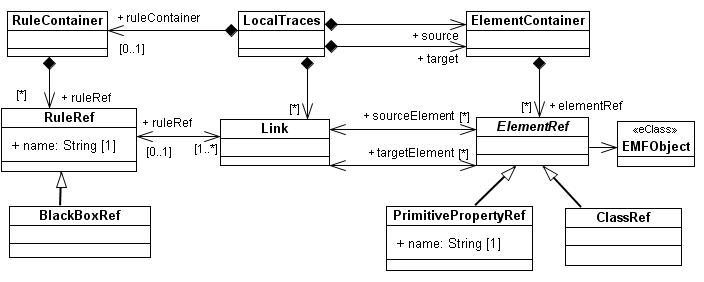


Figura 6.8: Meta-modelo de tracelinks Local

El meta-modelo de tracelinks local contiene dos conceptos principales, Link y ElementRef, los cuales expresan segu´n su definici´on que uno o m´as elementos or´ıgenes son enlazados a uno o mas 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 informaci´on sobre las reglas de transformaci´on aplicadas as´ı 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 ejecuci´on varios enlaces, se define la relacion 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 depuraci´on de las transformaciones, y la segunda si nos encontramos en la situaci´on en la que ciertas partes del sistema no puede ser pu´blica su implementacion.

ElementRef tiene una referencia al objeto real de los modelos origen y des- tinos. Como estos modelos estan implementados mediante EMF, la referencia EMFObject es un EObject del meta-modelo Ecore. La clase LocalTraces repre- senta la raiz del modelo de tracelinks 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 or´ıgenes y destinos permite reducir los costos de bu´squedas de elementos de entrada o salida.

6.5.2. Meta-modelo de Traceability Global

Este meta-modelo enlaza tracelinks locales de acuerdo a la cadena que define la transformaci´on. Un modelo de tracelinks global es el punto de entrada prin- cipal en el cual todas los modelos de tracelinks locales se encuentran, y describe qu´e modelo origen/objetivo de una transformacion es el modelo objetivo/ori- gen de la siguiente/previa transformaci´on. En el dibujo 6.9 se puede observar el meta-modelo.

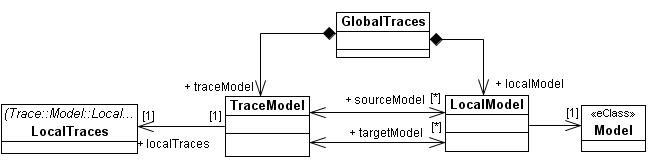


Figura 6.9: Meta-modelo de tracelinks Global

En este meta-modelo se engloban todas las tracelinks locales, los modelos de una cadena de transformaci´on y la forma en que ´estos se encuentran enlazados mediante TraceModel y LocalModel. Los modelos pueden ser compartidos entre distintas transformaciones, es decir uno puede ser producto de una transforma- cion y tambi´en ser consumido por otra transformacion.

Introducir este nivel global de tracelinks permite la navegacion entre los modelos transformados y sus modelos de tracelinks locales, dando una mejor separacion de lo que es en verdad la compleja informacion de traceability, lo que permite adem´as una mejor flexibilidad para la creacion de las tracelinks y la explotacion de las mismas. No utilizar esta idea de tracelinks globales tiene como consecuencia disponer de todas las tracelinks en un u´nico modelo de toda una cadena de transformacion, lo cual desencadenar´ıa en el colapso para la creacion y la consulta de dicho modelo.

6.5.3. ¿C´omo traba ja el framework?

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

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

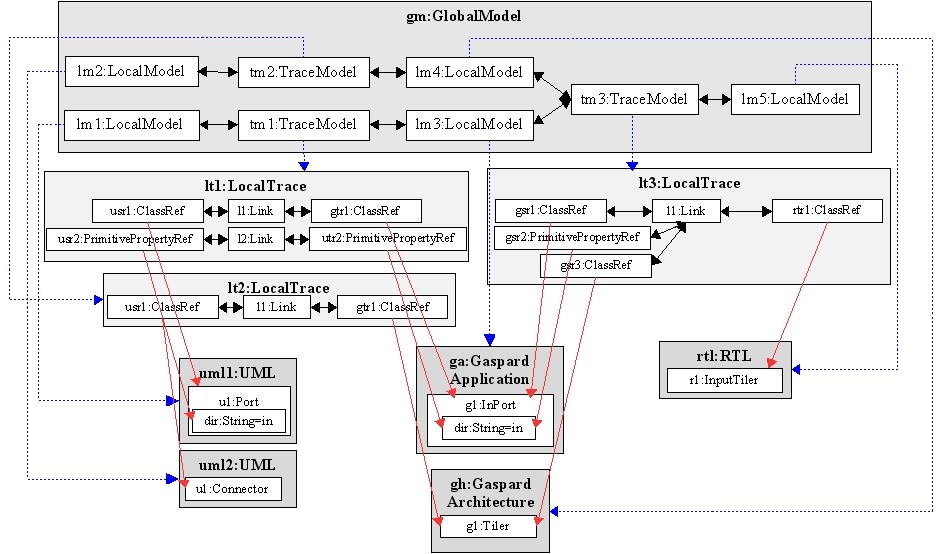


Figura 6.10: Ejemplo de un modelo de tracelinks local y global

Conclusio´n

Anexos

Traceability en ATL

La informaci´on de traceability forma parte del lenguaje ATL, ´esta ayuda en la interaccion de cada regla de transformacion cuando necesita tomar la salida de alguna otra regla por medio del mecanismo interno de traceability impl´ıcito. Sin embargo, actualmente ATL ofrece un acceso muy limitado a dicha informacion de traceability solo por medio del m´etodo resolveTemp(). Adem´as, ´esta informa- cion es eliminada ni bien la ejecucion de la transformacion termina. Dada las carencias enumeradas anteriormente, para lograr obtener alguna informacion de traceability en esta tecnolog´ıa es necesario implementar dicha funcionalidad.

Existen varias propuestas de implementacion, una de ellas, la que se presenta en [9] en donde se propone considerar como modelos tanto a los programas de transformacion como a la informacion de traceability, entonces mediante la de- finicion de una nueva transformaci´on modificar los programas para que generen la informaci´on de traceability como una salida mas al modelo resultado origi-

nal. Con este enfoque, la generaci´on de codigo de traceability est

claramente

separada de la logica de la transformaci´on y la misma se puede agregar despu´es de que un programa haya sido escrito. Por lo cual, es posible an˜adir el soporte para nuevos formatos o modificar la granularidad o rango de las tracelinks sin interferir con la logica del programa.

Otras propuestas hacen uso del mecanismo de traceability impl´ıcito de ATL, copiando toda la informaci´on de tracelinks durante la ejecucion de la trans- formacion, o directamente persisti´endola y dej´andola accesible en un archivo aparte, pero para todas estas implementaciones se requiere de modificaciones del motor de ejecucion de ATL.

QvtoTrace To Trace

A continuaci´on se expone la definicion completa de la transformaci´on QVT

QvtoTrace To Trace implementada en el prototipo que se desarroll sente Tesis:

en la pre-

/ ∗

∗ La d e f i n i c i o´ n d e l a p r e s e n t e t r a n s f o r m a c i ´o n

∗ Q v t o T r a c e T o T r a c e f o r m a p a r t e d e l p r o t o t i p o

∗ d e s a r r o l l a d o p a r a l a t e s i s d e l alumno Mariano G a b r i e l

∗ G i l i .

∗

∗ E s t a t r a n s f o r m a c i ´o n toma como e n t r a d a l a s t r a z a s g e n e r a d a s

∗ p o r l a t e c n o l o g ´ı a QVT q u e c o n f o r m a n e l meta−m o d e l o Trace ,

∗ y g e n e r a un m o d e l o p a r a e l p r o t o t i p o d e e d i c i ´o n d e t r a z a s

∗ q u e c o n f o r m a e l meta−m o d e l o p r o p u e s t o T r a c e E d i t o r .

∗/

/ ∗

∗ D e f i n i c i ´o n d e l o s meta−m o d e l o s .

∗/

modeltype T r a c e E d i t o r u s e s ’ h t t p : / / t r a c e e d i t o r / 1 . 0 ’ ;

modeltype Q v t o T r a c e u s e s

’ h t t p : / / /www . e c l i p s e . o r g /m2m/ q v t / o p e r a t i o n a l / t r a c e . e c o r e ’ ;

/ ∗

∗ D e c l a r a c i ´o n d e l a t r a n s f o r m a c i ´o n .

∗/

tran sformation Q v t o T r a c e T o T r a c e ( i n q v t o : Q v t o T r a c e ,

out T r a c e E d i t o r ) ;

/ ∗

∗ P u n t o d e e n t r a d a d o n d e s e i n i c i a l a t r a n s f o r m a c i ´o n p o r l o s

∗ o b j e c t o s r a i z d e t i p o T r a c e d e l m o d e l o d e e n t r a d a m e d i a n t e

∗ l a i n v o c a c i o´ n d e l mapeo t r a c e 2 T r a c e E d i t o r ( ) .

∗/

main ( ) {

q v t o . r o o t O b j e c t s ( ) [ Q v t o T r a c e : : T r a c e ]−>map

t r a c e 2 T r a c e E d i t o r ( ) ;

}

// / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /

// / / / / / / / / / / / / / / / / / / / M a p p i n g s / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /

// / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /

/ ∗

∗ t r a c e 2 T r a c e E d i t o r ( ) toma una i n s t a n c i a d e T r a c e y r e t o r n a

∗ una d e T r a c e E d i t o r . Genera l a c o n f i g u r a c i ´o n

∗ ( c o n f i g u r a c i o n ) m e d i a n t e e l mapeo t r a c e 2 C o n f i g u r a t i o n ( ) y

∗ e l d a s h b o a r d m e d i a n t e t r a c e 2 D a s h b o a r d ( ) .

∗/

mapping Q v t o T r a c e : : T r a c e : : t r a c e 2 T r a c e E d i t o r ( ) : T r a c e E d i t o r : : T r a c e E d i t o r {

c o n f i g u r a t i o n := s e l f . map t r a c e 2 C o n f i g u r a t i o n ( ) ;

d a s h b o a r d := s e l f . map t r a c e 2 D a s h b o a r d ( ) ;

}

/ ∗

∗ t r a c e 2 C o n f i g u r a t i o n ( ) g e n e r a una c o n f i g u r a c i ´o n d e l e d i t o r

∗ T r a c e E d i t o r r e p r e s e n t a d a p o r e l m o d e l o d e

∗ T r a c e E d i t o r : : C o n f i g u r a t i o n en l a q u e s e d e f i n e n l o s t i p o s

∗ d e l a s t r a z a s y s e o b t i e n e n l o s t i p o s d e a r t e f a c t o s q u e s e

∗ e n c u e n t r a n en l a s t r a z a s d e l m o d e l o Q v t o T r a c e : : T r a c e .

∗/

mapping Q v t o T r a c e : : T r a c e : : t r a c e 2 C o n f i g u r a t i o n ( ) : T r a c e E d i t o r : : C o n f i g u r a t i o n {

l i n k T y p e s := s e l f . l i n k T y p e s ( ) ;

t y p e A r t e f a c t s := s e l f . typesByName ( )−>c o l l e c t ( n |

o b j e c t T r a c e E d i t o r : : T y p e A r t e f a c t {

name := n ;

//TODO: s o l u c i o n a r p r o t o t i p o d e e d i t o r e s p a r a q u e

// m u e s t r e n l a d e s c r i p c i ´o n s i l a t i e n e e

// i n i c i a l i z a r l a c o n a l g u n a i n f o r m a c i ´o n .

// d e s c r i p t i o n := ” ” ;

} ) ;

}

/ ∗

∗ t r a c e 2 D a s h b o a r d ( ) d e l a s t r a z a s d e l m o d e l o d e e n t r a d a

∗ Q v t o T r a c e : : T r a c e o b t i e n e e l l i s t a d o d e t r a n s f o r m a c i o n e s

∗ ( t r a n s f o r m a t i o n s ) y l o s l i s t a d o s d e l o s a r t e f a c t o s o r i g e n

∗ y d e s t i n o ( s o u r c e A r t e f a c t s y t a r g e t A r t e f a c t s

∗ r e s p e c t i v a m e n t e ) m e d i a n t e l a i n s p e c c i ´o n d e l a s i n s t a n c i a s

∗ T r a c e R e c o r d d e d i c h o m o d e l o . L a s t r a z a s q u e r e p r e s e n t a n

∗ l o s e n l a c e s e x p l ´ı c i t o s ( t r a c e l i n k s ) s e i n i c i a l i z a n como un

∗ c o n j u n t o v a c ´ı o .

∗/

mapping Q v t o T r a c e : : T r a c e : : t r a c e 2 D a s h b o a r d ( ) : T r a c e E d i t o r : : D a s h b o a r d {

t r a n s f o r m a t i o n s := s e l f . t r a n s f o r m a t i o n s B y N a m e ( )−>c o l l e c t (

t |

o b j e c t T r a c e E d i t o r : : T r a n s f o r m a t i o n {

name := t ;

t r a c e L i n k s := s e l f . t r a c e R e c o r d s

[ m a p p i n g O p e r a t i o n . name=t ]−> map

t r a c e R e c o r d 2 T r a c e L i n k ( ) ;

} ) ;

// d a d o q u e l a s t r a z a s d e QVT s o n t o d a s p r o d u c t o d e

// t r a n s f o r m a c i o n e s , l a c o l e c c i ´o n t r a c e L i n k s s e

// i n i c i a l i z a v a c ´ı a .

t r a c e L i n k s := OrderedSet { } ;

s o u r c e A r t e f a c t s := s e l f . t r a c e R e c o r d s . c o n t e x t . c o n t e x t

[ name=” s e l f ” ] . map

v a r P a r a m e t e r V a l u e 2 A r t e f a c t ( ” S o u r c e ” ) ;

t a r g e t A r t e f a c t s := s e l f . t r a c e R e c o r d s . r e s u l t . r e s u l t . map

v a r P a r a m e t e r V a l u e 2 A r t e f a c t ( ” T a r g e t ” ) ;

}

/ ∗

∗ t r a c e R e c o r d 2 T r a c e L i n k ( ) c o n v i e r t e una t r a z a T r a c e R e c o r d en

∗ s u c o r r e s p o n d i e n t e d e t i p o T r a c e L i n k , e l nombre ( name ) e s

∗ l a r e p r e s e n t a c i ´o n en s t r i n g d e l a t r a z a o r i g e n , e l t i p o d e

∗ l a t r a z a e s ’ i m p l ´ı c i t a ’ , l a t r a n s f o r m a c i ´o n

∗ ( t r a n s f o r m a t i o n ) e s d e l r e s u l t a d o e l a r t e f a c t o q u e l o

∗ c o n t i e n e , l o s a r t e f a c t o s o r i g e n y d e s t i n o ( s o u r c e s y

∗ t a r g e t s ) s o n e l c o n t e x t o ( c o n t e x t ) y l o s r e s u l t a d o s

∗ ( r e s u l t ) r e s p e c t i v a m e n t e .

∗/

mapping Q v t o T r a c e : : T r a c e R e c o r d : : t r a c e R e c o r d 2 T r a c e L i n k ( ) : T r a c e E d i t o r : : T r a c e L i n k {

name := s e l f . r e p r ( ) . t o T r a c e L i n k N a m e ( ) ;

t y p e := s e l f . v a r P a r a m e t e r V a l u e L i n k T y p e ( ’ I m p l i c i t ’ ) ;

t r a n s f o r m a t i o n := r e s u l t . c o n t a i n e r ( ) . o c l A s T y p e ( T r a c e E d i t o r : : T r a n s f o r m a t i o n ) ;

s o u r c e s := s e l f . c o n t e x t . c o n t e x t [ name=” s e l f ” ] . map

v a r P a r a m e t e r V a l u e 2 A r t e f a c t ( ” S o u r c e ” ) ;

t a r g e t s := s e l f . r e s u l t . r e s u l t −>map

v a r P a r a m e t e r V a l u e 2 A r t e f a c t ( ” T a r g e t ” )−> f l a t t e n ( ) ;

}

/ ∗

∗ v a r P a r a m e t e r V a l u e 2 A r t e f a c t ( ) t r a n s f o r m a l o s p a r ´a m e t r o s

∗ V a r P a r a m e t e r V a l u e d e l a s t r a z a s d e l m o d e l o d e e n t r a d a en

∗ a r t e f a c t o s d e t i p o A r t e f a c t , d e r i v a l a t r a n s f o r m a c i o´ n en

∗ v a r P a r a m e t e r V a l u e S i m p l e 2 A r t e f a c t ( ) y

∗ v a r P a r a m e t e r V a l u e C o l l e c t i o n 2 A r t e f a c t ( ) d e p e n d i e n d o d e s i

∗ e l p a r ´a m e t r o r e p r e s e n t a un v a l o r s i m p l e o una c o l e c c i ´o n

∗ r e s p e c t i v a m e n t e .

∗/

mapping Q v t o T r a c e : : V a r P a r a m e t e r V a l u e : :

v a r P a r a m e t e r V a l u e 2 A r t e f a c t ( i n p r e f i x : S t r i n g )

: OrderedSet ( T r a c e E d i t o r : : A r t e f a c t )

d i s j u n c t s Q v t o T r a c e : : V a r P a r a m e t e r V a l u e : : v a r P a r a m e t e r V a l u e S i m p l e 2 A r t e f a c t , Q v t o T r a c e : : V a r P a r a m e t e r V a l u e : :

v a r P a r a m e t e r V a l u e C o l l e c t i o n 2 A r t e f a c t {}

mapping Q v t o T r a c e : : V a r P a r a m e t e r V a l u e : :

v a r P a r a m e t e r V a l u e S i m p l e 2 A r t e f a c t ( i n p r e f i x : S t r i n g )

: OrderedSet ( T r a c e E d i t o r : : A r t e f a c t )

when { s e l f . v a l u e . c o l l e c t i o n T y p e = n u l l } {

i n i t {

r e s u l t := o b j e c t T r a c e E d i t o r : : A r t e f a c t {

name := s e l f . v a l u e . r e p r ( ) . t o A r t e f a c t N a m e (

p r e f i x ) ;

t y p e := s e l f . v a r P a r a m e t e r V a l u e T y p e A r t e f a c t (

s e l f . t y p e . p r e f i x ( p r e f i x ) ) ;

}−>a s O r d e r e d S e t ( ) ;

}

}

mapping Q v t o T r a c e : : V a r P a r a m e t e r V a l u e : :

v a r P a r a m e t e r V a l u e C o l l e c t i o n 2 A r t e f a c t ( i n p r e f i x : S t r i n g )

: OrderedSet ( T r a c e E d i t o r : : A r t e f a c t )

when { s e l f . v a l u e . c o l l e c t i o n T y p e <> n u l l } {

i n i t {

var typeName := s e l f . t y p e . s u b s t r i n g A f t e r ( ” ( ” )

. s u b s t r i n g B e f o r e ( ” ) ” ) ;

r e s u l t := s e l f . v a l u e . c o l l e c t i o n −>c o l l e c t ( e |

o b j e c t T r a c e E d i t o r : : A r t e f a c t {

name := e . r e p r ( ) . t o A r t e f a c t N a m e ( p r e f i x ) ;

t y p e := s e l f

. v a r P a r a m e t e r V a l u e T y p e A r t e f a c t (

typeName . p r e f i x ( p r e f i x ) ) ;

} )−>a s O r d e r e d S e t ( ) ;

}

}

// / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /

// / / / / / / / / / / / / / / / / / / / / / C o n s t r u c t o r s / / / / / / / / / / / / / / / / / / / / / / / /

// / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /

c o n s t r u c t o r T r a c e E d i t o r : : L i nk T y p e : : Li n k T y pe ( aName : S t r i n g , a P a r e n t : T r a c e E d i t o r : : Li nk T y p e ) {

name := aName ;

p a r e n t := a P a r e n t ;

}

// / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /

// / / / / / / / / / / / / / / / / / / / / / / / H e l p e r s / / / / / / / / / / / / / / / / / / / / / / / / / / /

// / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /

/ ∗

∗ l i n k T y p e s ( ) c r e a un c o n j u n t o d e t i p o s d e e n l a c e s p o r

∗ d e f e c t o .

∗/

h e l p e r Q v t o T r a c e : : T r a c e : : l i n k T y p e s ( ) : OrderedSet ( T r a c e E d i t o r : : L i nk T y p e ) {

var E x p l i c i t := new T r a c e E d i t o r : : L i nk T y p e ( ’ E x p l i c i t ’ ,

n u l l ) ;

var I m p l i c i t := new T r a c e E d i t o r : : L i nk T y p e ( ’ I m p l i c i t ’ ,

n u l l ) ;

var ModelModel := new T r a c e E d i t o r : : L i nk T y p e (

’ Model−Model ’ , E x p l i c i t ) ;

var M o d e l A r t e f a c t := new T r a c e E d i t o r : : L i nk T y p e (

’ Model−A r t e f a c t ’ , E x p l i c i t ) ;

var Query := new T r a c e E d i t o r : : L i nk T y p e ( ’ Query ’ ,

I m p l i c i t ) ;

var M2M := new T r a c e E d i t o r : : L i nk T y p e ( ’M2M ’ , I m p l i c i t ) ; var M2T := new T r a c e E d i t o r : : L i nk T y p e ( ’M2T ’ , I m p l i c i t ) ; var C o m p o s i t i o n := new T r a c e E d i t o r : : L i nk T y p e (

’ C o m p o s i t i o n ’ , I m p l i c i t ) ;

var Update := new T r a c e E d i t o r : : L i nk T y p e ( ’ Update ’ , I m p l i c i t ) ;

var C r e a t i o n := new T r a c e E d i t o r : : L i nk T y p e ( ’ C r e a t i o n ’ , I m p l i c i t ) ;

var D e l e t e := new T r a c e E d i t o r : : L i nk T y p e ( ’ D e l e t e ’ , I m p l i c i t ) ;

var S t a t i c := new T r a c e E d i t o r : : L i nk T y p e ( ’ S t a t i c ’ , ModelModel ) ;

var D i n a m i c := new T r a c e E d i t o r : : L i nk T y p e ( ’ D i n a m i c ’ , ModelModel ) ;

var S a t i s f i e s := new T r a c e E d i t o r : : L i nk T y p e ( ’ S a t i s f i e s ’ , M o d e l A r t e f a c t ) ;

var A l l o c a t e d T o := new T r a c e E d i t o r : : L i nk T y p e (

’ A l l o c a t e d −To ’ , M o d e l A r t e f a c t ) ;

var P e r f o r m s := new T r a c e E d i t o r : : L i nk T y p e ( ’ P e r f o r m s ’ ,

M o d e l A r t e f a c t ) ;

var S u p p o r t s := new T r a c e E d i t o r : : L i nk T y p e ( ’ S u p p o r t s ’ , M o d e l A r t e f a c t ) ;

var E x p l a i n s := new T r a c e E d i t o r : : L i nk T y p e ( ’ E x p l a i n s ’ ,

M o d e l A r t e f a c t ) ;

var D e p e n d e n c y := new T r a c e E d i t o r : : L i nk T y p e (

’ D e p e n d e n c y ’ , S t a t i c ) ;

var C o n s i s t e n t W i t h := new T r a c e E d i t o r : : L i nk T y p e (

’ C o n s i s t e n t W i t h ’ , S t a t i c ) ;

var C a l l s := new T r a c e E d i t o r : : L i nk T y p e ( ’ C a l l s ’ , D i n a m i c ) ;

var G e n e r a t e s := new T r a c e E d i t o r : : L i nk T y p e ( ’ G e n e r a t e s ’ , D i n a m i c ) ;

var N o t i f i e s := new T r a c e E d i t o r : : L i nk T y p e ( ’ N o t i f i e s ’ , D i n a m i c ) ;

return OrderedSet {

I m p l i c i t , E x p l i c i t , ModelModel , M o d e l A r t e f a c t , Query ,

M2M, M2T, C o m p o s i t i o n , Update , C r e a t i o n , D e l e t e ,

S t a t i c , Dinamic , S a t i s f i e s , A l l o c a t e d T o , P e r f o r m s , S u p p o r t s , E x p l a i n s , Dependency , C o n s i s t e n t W i t h , C a l l s , G e n e r a t e s , N o t i f i e s

} ;

}

/ ∗

∗ p r e f i x ( ) a g r e g a e l s t r i n g p a r ´a m e t r o p r e como p r e f i j o .

∗/

h e l p e r S t r i n g : : p r e f i x ( i n p r e : S t r i n g ) : S t r i n g {

return p r e + ” ” + s e l f ;

}

/ ∗

∗ t o A r t e f a c t N a m e ( ) d e f i n e e l f o r m a t o d e l nombre d e l o s

∗ a r t e f a c t o s c o n e l s t r i n g p a r ´a m e t r o p r e como p r e f i j o y

∗ l u e g o tomando s ´o l o l a p a r t e d e l nu´ mero i d e n t i f i c a d o r d e l o

∗ r e t o r n a d o p o r l a f u n c i ´o n r e p r ( ) .

∗/

h e l p e r S t r i n g : : t o A r t e f a c t N a m e ( i n p r e : S t r i n g ) : S t r i n g {

var temp name = s e l f . s u b s t r i n g A f t e r (

” o r g . e c l i p s e . m2m . i n t e r n a l . q v t . oml . t r a c e . i m p l . EValueImpl@

” )

. s u b s t r i n g B e f o r e ( ” ( ” ) ;

return p r e + ” ” + temp name ;

}

/ ∗

∗ t o T r a c e L i n k N a m e ( ) d e f i n e e l f o r m a t o d e l nombre d e l a s

∗ t r a z a s tomando s ´o l o l a p a r t e d e l nu´ mero i d e n t i f i c a d o r d e

∗ l o r e t o r n a d o p o r l a f u n c i ´o n r e p r ( ) y como p r e f i j o ” T r a c e ”

∗/

h e l p e r S t r i n g : : t o T r a c e L i n k N a m e ( ) : S t r i n g {

var temp name = s e l f . s u b s t r i n g A f t e r (

” o r g . e c l i p s e . m2m . i n t e r n a l . q v t . oml . t r a c e . i m p l .

T r a c e R e c o r d I m p l @ ” ) ;

return ” T r a c e ” + temp name ;

}

// / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /

// / / / / / / / / / / / / / / / / / / / / / / / Q u e r y s / / / / / / / / / / / / / / / / / / / / / / / / / / / /

// / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /

/ ∗

∗ v a r P a r a m e t e r V a l u e T y p e A r t e f a c t ( ) r e t o r n a d e l m o d e l o y a

∗ c o n v e r t i d o q u e r e p r e s e n t a l a c o n f i g u r a c i ´on , e l t i p o d e l

∗ a r t e f a c t o q u e c o n f o r m a a T y p e A r t e f a c t i d e n t i f i c a d o p o r e l

∗ p a r a´ m e t r o typeName .

∗/

query Q v t o T r a c e : : V a r P a r a m e t e r V a l u e : :

v a r P a r a m e t e r V a l u e T y p e A r t e f a c t ( i n typeName : S t r i n g )

: T r a c e E d i t o r : : T y p e A r t e f a c t {

return s e l f . c o n t a i n e r ( ) . c o n t a i n e r ( ) . c o n t a i n e r ( )

. o c l A s T y p e ( Q v t o T r a c e : : T r a c e ) . r e s o l v e o n e I n ( Q v t o T r a c e : : T r a c e : : t r a c e 2 C o n f i g u r a t i o n , T r a c e E d i t o r : : C o n f i g u r a t i o n

) . t y p e A r t e f a c t s ! [ name = typeName ] ;

}

/ ∗

∗ v a r P a r a m e t e r V a l u e L i n k T y p e ( ) r e t o r n a d e l m o d e l o y a

∗ c o n v e r t i d o q u e r e p r e s e n t a l a c o n f i g u r a c i ´on , e l t i p o d e

∗ t r a z a q u e c o n f o r m a a L i n k T y p e i d e n t i f i c a d o p o r e l

∗ p a r a´ m e t r o typeName .

∗/

query Q v t o T r a c e : : T r a c e R e c o r d : : v a r P a r a m e t e r V a l u e L i n k T y p e (

i n typeName : S t r i n g )

: T r a c e E d i t o r : : L i n k Ty pe {

return s e l f . c o n t a i n e r ( ) . o c l A s T y p e ( Q v t o T r a c e : : T r a c e )

. r e s o l v e o n e I n (

Q v t o T r a c e : : T r a c e : : t r a c e 2 C o n f i g u r a t i o n , T r a c e E d i t o r : : C o n f i g u r a t i o n

) . l i n k T y p e s ! [ name = typeName ] ;

}

/ ∗

∗ typesByName ( ) r e t o r n a e l c o n j u n t o d e n o m b r e s d e l o s

∗ d i s t i n t o s t i p o s d e a r t e f a c t o s q u e s e e n c u e n t r a n en e l

∗ c o n j u n t o d e t r a z a s d e l m o d e l o d e e n t r a d a T r a c e . E s t e

∗ c o n j u n t o e s e l t i p o d e l o s o r ´ı g e n e s , o b t e n i d o s d e l

∗ c o n t e x t o d e l m o d e l o Trace , u n i ´o n l o s t i p o s d e l o s

∗ r e s u l t a d o s , o b t e n i d o s d e l o s r e s u l t a d o s .

∗/

query Q v t o T r a c e : : T r a c e : : typesByName ( ) : OrderedSet ( S t r i n g ) {

return s e l f . t r a c e R e c o r d s . c o n t e x t . c o n t e x t [ name=” s e l f ” ]

. t y p e . p r e f i x ( ” S o u r c e ” )−>a s S e t ( )−>u n i o n (

s e l f . t r a c e R e c o r d s . r e s u l t . r e s u l t [ name=” r e s u l t ”

and v a l u e . c o l l e c t i o n T y p e = n u l l ] . t y p e . p r e f i x (

” T a r g e t ” )−>a s S e t ( )−>u n i o n (

s e l f . t r a c e R e c o r d s . r e s u l t . r e s u l t [

name=” r e s u l t ” and

v a l u e . c o l l e c t i o n T y p e <> n u l l ]

. t y p e . s u b s t r i n g A f t e r ( ” ( ” ) . s u b s t r i n g B e f o r e ( ” ) ” )

. p r e f i x ( ” T a r g e t ” )−>a s S e t ( ) )

)−>a s O r d e r e d S e t ( ) ;

}

/ ∗

∗ t r a n s f o r m a t i o n s B y N a m e ( ) r e t o r n a e l c o n j u n t o d e n o m b r e s d e

∗ l a s t r a n s f o r m a c i o n e s d e l c o n j u n t o d e t r a z a s q u e s e

∗ e n c u e n t r a n en e l m o d e l o d e e n t r a d a T r a c e .

∗/

query Q v t o T r a c e : : T r a c e : : t r a n s f o r m a t i o n s B y N a m e ( ) :

OrderedSet ( S t r i n g ) {

return s e l f . t r a c e R e c o r d s . m a p p i n g O p e r a t i o n . name−>

a s O r d e r e d S e t ( ) ;

}

Glosario

GEF - Graphical Editing Framework Es un framework Eclipse para el desa- rrollo de editores graficos y vistas del Eclipse Workbench UI. 45, 46

JDT - Java Development Tool Es un proyecto que proporciona herramien- tas tipo plug-ins de Eclipse que asisten en el desarrollo de aplicaciones Java. Incluye la creacion de proyectos Java, una perspectiva para el Work- bench Eclipse, as´ı como tambi´en un conjunto de vistas, editores, asisten- tes, constructores y herramientas de refactorizacion y fusionado de c´odigo. Estas herramientas transforman a Eclipse en un IDE. 43

MDA - Arquitectura Dirigida por Modelos en ingl´es Model-Driven Ar- chitecture (MDA), es una vision o propuesta particular definida por el OMG de MDD que se basa en el uso de est´andares de OMG. vi

MDD - Desarrollo Dirigido por Modelos en ingl´es Model-Driven Develop- ment (MDD), es un paradigma de desarrollo de software que utiliza mo- delos como artefactos principales del proceso de desarrollo. Por lo general, la implementacion se genera o deriva autom´aticamente a partir de los modelos. vi, 2

MDE - Ingenier´ıa de Software Dirigida por Modelos en ingl´es Model-Driven

Engineering (MDE), es un paradigma de ingenier´ıa de software en el cual los modelos juegan el rol principal en todas las actividades del ciclo de vida de la misma. vi, vii, 9, 10, 14, 48

MOF - MetaOb ject Facility es un lenguaje estandar del OMG para MDE, en particular es un meta-meta lenguaje que permite definir meta-modelos en la capa M2, como por ejemplo el meta-modelo Lenguaje Unificado de Modelado (UML de Unified Modeling Language) que describe al lenguaje Lenguaje Unificado de Modelado (UML de Unified Modeling Language). Como su nombre lo indica, MOF se basa en el paradigma Orientado a Objetos. 14, 49, 51

OCL - Ob ject Constraint Language es un lenguaje declarativo para des- cribir reglas que se aplican a meta-modelos MOF y a los modelos Len- guaje Unificado de Modelado (UML de Unified Modeling Language). Fue desarrollado por IBM y en la actualidad es parte del est´andar Lenguaje Unificado de Modelado (UML de Unified Modeling Language). OCL es un lenguaje de texto que permite definir restricciones y consultas sobre expresiones de objetos de cualquier modelo o meta-modelo MOF. OCL es componente clave del estandar OMG para la transformacion de los modelos QVT. Otros lenguajes de transformaci´on de modelos como ATL tambi´en est´an construidos utilizando OCL. 8

OMG - Ob ject Management Group Consorcio internacional de la indus- trial informatica sin fines de lucro y de membres´ıa abierta responsable de la definicion de varios estandares de modelado como Lenguaje Unificado de Modelado (UML de Unified Modeling Language), MetaObject Facility (MOF ) y MDA. vi, 50

PDE - Plug-in Development Environment es un ambiente de construc- cion de Plugins Eclipse confeccionado por un conjunto de herramientas para crear, desarrollar, probar, depurar, construir y desplegar plug-ins, fragmentos, caracter´ısticas, sitios de actualizacion y otros productos Eclip- se. 34

UML - Unified Modeling Language es un lenguaje grafico para visualizar, especificar, construir y documentar un sistema de software. Es el lenguaje de modelado de sistemas de software mas conocido y utilizado en la ac- tualidad y es tambi´en el estandar oficial, respaldado por el OMG. 6, 8, 10,

14, 45, 55, 57

Siglas

ALM Application Lifecycle Management. 39

API Interfaz de programacion de aplicaciones o Application Programming In- terface. 40, 42, 45, 53, 57

ATL Atlas Transformation Language. 31, 48–50, 65

CASE Computer Aided Software Engineering. 57

EMF Eclipse Modeling Framework. 7, 14, 42, 43, 45, 46, 53, 60

EPL Eclipse Public License. 39

EVL Epsilon Validation Language. 8

GMF Graphical Modeling Framework. 33, 45–48

GUI Interfaz de Usuario Grafica o Graphical User Interface. 43

IDE Entornos de Desarrollo Integrados o Integrated Development Environ- ment. 17, 24, 25, 40, 48

IEEE Institute of Electrical and Electronics Engineers. 2

MMT Model-to-Model Transformation. 49, 50

QVT Query/View/Transformation. 29, 31, 50, 51, 57, 66

RCP Rich Client Platform. 39

RIA Rich Internet Applications. 39

SOA Service Oriented Architecture. 39

SPL L´ınea de Producto de Software o Software Product Line. 54, 55

SWT Standard Widget Toolkit. 41

TEAP Traceability Elicitation and Analysis Process. 8

XMI XML Metadata Interchange. 42, 45, 54

Bibliograf´ıa

[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 Fra- mework in Model-Driven Software Engineering. In: ECMDA Traceability Workshop (ECMDA-TW) 2009 Proceedings.

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

[//www.coest.org/index.php/traceability/glossary.](http://www.coest.org/index.php/traceability/glossary)

[5] Center of Excellence for Traceability - Problem Statements and Grand Cha- llenges. 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 Tra- ceability Problem”, International Conference on Requirements Enginee- ring, ICRE’94, Los Alamitos, California, Abril, 1994, pp 94-101.

[7] N. Drivalos, R. F. Paige, K. J. Fernandes, D. S. Kolovos: Towards Ri- gorously 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 Traceabi- lity 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 Tra- ceability 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/](http://www.eclipse.org/modeling/emf/)

[modeling/emf/.](http://www.eclipse.org/modeling/emf/)

[16] A. Sousa, U. Kulesza, A. Rummler, N. Anquetil, R. Mitschke, A. Moreira, V. Amaral, J. Arau´jo: 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.](http://www.eclipse.org)

[18] Eclipse Platform Technical Overview. Object Technology Interna- tional, Inc., February 2003, [http://www.eclipse.org/whitepapers/ eclipse-overview.pdf.](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/.](http://www.eclipse.org/modeling/gmp/) [21] Frederic Plante, IBM: Introducing the GMF Runtime, January

2006 [http://www.eclipse.org/articles/Article-Introducing-GMF/](http://www.eclipse.org/articles/Article-Introducing-GMF/article.html)

[article.html](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](http://wiki.eclipse.org/Graphical_Modeling_Framework/Tutorial/Part_1)

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

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

[25] C. Pons, R. Giandini y G. P´erez: Desarrollo de Software Dirigido por Mo- delos. Conceptos teoricos y su aplicaci´on pr´actica. Facultad de Informatica, Universidad Nacional de La Plata. Octubre de 2008.

[26] Pierre F. Tiako: Designing Software-Intensive Systems, Methods and Prin- ciples. Langston University, USA. 2008.