Integrando Especificaciones Textuales y Elementos de modelado UML en un Marco de Trabajo para Trazabilidad de Requisitos*

Patricio Letelier y Víctor Anaya
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
{letelier, vanaya}@dsic.upv.es

Palabras Clave: Ingeniería de requisitos, trazabilidad de requisitos, UML

Abstract. La trazabilidad de requisitos permite asegurar la continua concordancia entre los requisitos de los *stakeholders* y los artefactos producidos durante el proceso de desarrollo de software. Aunque el importante papel de la trazabilidad de requisitos es ampliamente reconocido, el grado de aplicación y consenso en cuanto a prácticas varía considerablemente entre equipos de desarrollo de software. Hasta ahora no ha existido una propuesta que reuna los resultados de las comunidades de ingeniería de requisitos con los de modelado y construcción del software. UML emerge como una oportunidad para establecer un marco común para trazabilidad de requisitos. En este trabajo presentamos un metamodelo de referencia para trazabilidad de requisitos, basado en UML y que integra tanto especificaciones textuales como elementos de modelado UML, consiguiendo así una representación homogénea para todos los artefactos producidos durante el proceso de desarrollo y para las dependencias de trazabilidad entre ellos. Aprovechando los mecanismos de extension de UML, hemos conseguido de manera natural que nuestra propuesta sea adaptable y extensible según las características particulares de un proyecto. Finalmente, hemos incluido un pequeño ejemplo para mostrar la aplicación de nuestro enfoque a un proyecto, usando como proceso de software *Rational Unified Process* (RUP).

1 Introducción

Los requisitos del software son susceptibles de cambios, no sólo después de la entrega del producto sino que también durante el proceso de desarrollo iterativo. La gestión de requisitos es el proceso que administra los cambios en los requisitos del software. La gestión de requisitos es un Área Clave de Proceso (*Key Process Area*) necesaria para alcanzar el nivel 2 (nivel repetible) del CMM. La gestión de requisitos debería estar integrada como subproceso dentro del proceso de desarrollo [1]. Para que este subproceso sea efectivo es imprescindible

* Este trabajo ha sido financiado por el proyecto DOLMEN-SIGLO de la Comisión Interministerial de Ciencia y Tecnología, TIC2000-1673-C06-01.

que las relaciones entre los requisitos y otras especificaciones elaboradas durante el proyecto de desarrollo de software estén adecuadamente definidas. La trazabilidad de requisitos se define como la habilidad para describir y seguir la vida de un requisito en ambos sentidos, hacia sus orígenes o hacia su implementación, a través de todas las especificaciones generadas durante el proceso de desarrollo de software. La trazabilidad de requisitos ha sido definida en la literatura como un factor de calidad, una característica que un sistema debería incluir como si se tratase de un requisito no-funcional [6]. En el Estándar IEEE 830-1998 "Prácticas recomendadas para Especificación de Requisitos de Software", trazabilidad es una de las ocho características que debe tener una buena especificación de requisitos.

La gestión de requisitos puede ser un proceso muy costoso, con lo cual, debe planificarse el nivel de detalle que se desea, dependiendo del proyecto. Es primordial que la recolección de información de trazabilidad y su uso sea acorde con las necesidades específicas del proyecto para así conseguir un resultado positivo respecto del costo-beneficio de esta tarea.

En la actualidad la efectividad de las prácticas en trazabilidad de requisitos varía considerablemente entre diferentes equipos de desarrollo. Algunos problemas que explican esta situación son [9][10]: la carencia de guías detalladas respecto de los tipos de información que debe ser capturada para trazabilidad, del contexto en que dicha información debe ser usada y la falta de consenso referente a la semántica de los enlaces de trazabilidad entre especificaciones.

Tradicionalmente la especificación de requisitos se ha realizado usando sobre todo especificaciones textuales en lenguaje natural. Consecuentemente, las herramientas de apoyo a la gestión de requisitos se han enfocado a la manipulación de trozos de texto. Estos requisitos expresados textualmente se enlazan formando un grafo de trazabilidad el cual se usa para gestionar los requisitos y su trazabilidad. En este enfoque, las especificaciones generadas en las otras actividades del desarrollo de software pueden también ser añadidas al grafo de trazabilidad representándolas como texto (normalmente el nombre de la especificación, por ejemplo el nombre de una clase, atributo u operación). Las especificaciones de pruebas son principalmente textuales, con lo cual también pueden ser tratadas de manera similar. Aunque varios proveedores de herramientas CASE anuncian que sus productos ofrecen una buena integración entre los módulos para requisitos, desarrollo y pruebas, normalmente la solución implementada se basa en mecanismos de importación/exportación entre dichos módulos. Esta estrategia es poco natural de cara a ser promulgada como parte del proceso de software y debe enfrentar los problemas obvios de sincronización. Otra alternativa posible sería modelar en diagramas las dependencias entre artefactos, dibujándolas una a una, lo cual es inviable, incluso para sistemas pequeños.

Por otra parte, UML [11] ha llegado rápidamente a convertirse en la notación más popular para modelado orientado a objetos. Gracias a la definición de su metamodelo y a los mecanismos de extensión incluidos, UML ofrece una excelente oportunidad para establecer un marco común para la representación de especificaciones de requisitos, de desarrollo y de pruebas.

El objetivo de este trabajo es establecer un marco integrado para la trazabilidad de requisitos que reconcilie especificaciones textuales y elementos de modelado UML, permitiendo establecer trazabilidad a través de cualquier tipo de artefacto. El enfoque propuesto es independiente del proceso de software que se desee utilizar y permite su adaptación a las necesidades de trazabilidad de un proyecto.

El trabajo está organizado en seis secciones. Después de esta introducción, la siguiente sección describe un metamodelo para trazabilidad. La sección tercera explica cómo las especificaciones textuales y lo enlaces de trazabilidad se pueden definir en UML consiguiendo un marco homogéneo para toda la información de trazabilidad. En la sección cuarta se ilustra la aplicación del enfoque propuesto utilizando como ejemplo un proyecto pequeño con RUP. La sección quinta describe algunos trabajos relevantes en el área de trazabilidad de requisitos y algunas herramientas representativas, específicas para la gestión de requisitos. Finalmente la sección sexta presenta las conclusiones.

2 Un Metamodelo para Trazabilidad de Requisitos

Antes de presentar nuestro metamodelo de trazabilidad resumiremos las necesidades de información para la gestión de requisitos. A continuación se indican los tipos de información asociada a la trazabilidad de requisitos y sus posibles usos (adaptado desde [2]):

1. Los enlaces de trazabilidad entre diferentes tipos de especificaciones permiten: validar que la funcionalidad del sistema reúne las expectativas del cliente y que no se ha implementado funcionalidad superflua, y realizar análisis de impacto de cambios en los requisitos.
2. Las estructuras de contribución [3], es decir, los enlaces entre participantes en el proyecto (*stakeholders*) y las especificaciones permiten: mejorar la comunicación y cooperación entre los participantes del proyecto, y asegurar que la contribución de cada individuo es considerada y registrada.
3. Los fundamentos asociados a las especificaciones, incluyendo alternativas, decisiones, suposiciones, etc. contribuyen a: mejorar la comprensión y aceptación del sistema por parte de los *stakeholders*, y a mejorar la gestión de los cambios evitando reconsiderar cuestiones ya antes descartadas, pues las soluciones y sus fundamentos, así como las alternativas descartadas son accesibles.

El metamodelo que proponemos se muestra en la Figura 1 mediante un diagrama de clases. Las clases representan los tipos de entidades y las asociaciones los tipos de enlaces de trazabilidad. Se utilizan nombres de roles (*rolnames*) para distinguir los distintos tipos de enlaces de trazabilidad.

En términos generales interesan dos tipos de entidades *TraceableSpecification* y *Stakeholders*. Los *Stakeholders* son responsables de crear y modificar especificaciones. Una *TraceableSpecification* es una especificación de software con un determinado nivel de granuralidad, es decir, puede ser un documento, un modelo, un diagrama, un apartado de un documento, un texto especificando un requisito

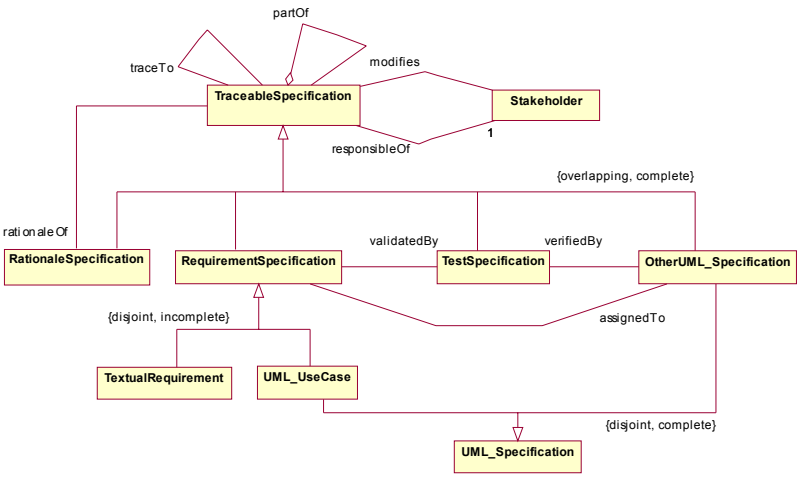

Fig. 1. Un metamodelo para trazabilidad de requisitos

no-funcional, un caso de uso, una clase, un atributo, etc.). Esta granularidad para una *TraceableSpecification* se define mediante la asociación con el nombre de rol *partOf*.

El tipo de entidad *TraceableSpecification* es una generalización de *RationaleSpecification*, *RequirementSpecification*, *TestSpecification*, y *OtherUML Specification*. Una *TraceableSpecification* puede clasificarse en más de uno de estos subtipos, por ejemplo, cuando se trata de un documento que incluye distintos tipos de especificaciones (mediante *partOf*). Una *RequirementSpecification* es un requisito o grupo de requisitos. Los requisitos, según cómo se expresan, puede clasificarse en *TextualRequirements* (requisitos expresados mediante un texto) o *UML_ UseCase* (elemento de modelado usado en UML para representar un requisito funcional). Una *RationaleSpecification* establece, por ejemplo: fundamentos, alternativas o suposiciones asociadas a una *TraceableSpecification*. Finalmente, una *TestSpecification* define una prueba, ya sea para validar un requisito o para verificar un elemento de modelado UML (por ejemplo: para verificar un fichero con código fuente que es la implementación de una clase o un componente). La generalización cuya clase padre es *TraceableSpecification* está definida como "completa" para enmarcar todos los tipos de especificación de interés para trazabilidad en uno (o más) de los subtipos establecidos. Por ejemplo, algunos tipos de especificación no textuales ni UML son: vídeos, imágenes, voz, etc. Dichas especificaciones no textuales ni UML usualmente constituyen fundamentos, útiles durante la revisión y evaluación de modelos de análisis y diseño [4], pero podrían ser otro tipo de especificación entre los establecidos (por ejemplo, una *RequirementSpecification* registrada en un vídeo)

El tipo de enlace más genérico es la asociación con nombre de rol[1] *traceTo* el cual establece relaciones de trazabilidad entre *TraceableSpecifications*. El resto de los enlaces (*modifies*, *responsibleOf*, *rationaleOf*, *validatedBy*, *verifiedBy* y *assignedTo*) constituyen enlaces de trazabilidad más específicos. El enlace *modifies* establece la relación entre los *Stakeholders* y las *TraceableSpecifications* que éstos modifican. Del mismo modo, *responsibleOf* determina el *Stakeholder* que es responsable de la definición y mantenimiento de una *TraceableSpecification*. El enlace *validatedBy* relaciona a las *RequimementsSpecifications* con las correspondientes *TestSpecification* que las validan. Correspondientemente, el enlace *verifiedBy* determina las *TestSpecifications* que verifican una especificación UML. Por último, el enlace *assignedTo* determina a cuáles elementos de modelado UML se les asigna la realización de un determinado requisito, por ejemplo, qué clases realizan un Caso de Uso.

El metamodelo de la Figura 1 cubre las cuatro perspectivas de información de trazabilidad [10]: requisitos, fundamentos (*rationale*), asignación de requisitos a elementos de modelado o implementación, y finalmente, pruebas. Por otra parte, nuestro metamodelo incorpora los aspectos de pre-trazabilidad y post-trazabilidad [6][13]. La pre-trazabilidad permite ir desde los orígenes de los requisitos hasta su especificación explícita en una especificación de requisitos del software (SRS: *Software Requirement Specification*) o viceversa. La post-trazabilidad permite ir desde la SRS a las posteriores especificaciones de software y las pruebas que las validan/verifican, y viceversa. En ambos tipos de trazabilidad nuestro metamodelo provee enlaces *responsibleOf* y *modifies* para determinar los *Stakeholders* involucrados. Para pre-trazabilidad se dispone del enlace *traceTo* entre requisitos en diferentes niveles de abstracción y *rationaleOf* asociados a dichas especificaciones de requisitos. La post-trazabilidad es soportada por los enlaces *traceTo*, *validatedBy*, *verifiedBy*, *assignedTo* y *rationaleOf*.

3 El Metamodelo en el marco de UML

Para que la aplicación del metamodelo sea sencilla y práctica es conveniente integrar todos los tipos de entidades y enlaces en un contexto común. Considerando que: (a) las especificaciones UML están definidas con mayor precisión y aceptación que los otros tipos de especificaciones incluidas en el metamodelo, (b) que UML provee mecanismos de extensión (*stereotypes*, *tagged values* y *constraints*) para incorporar nuevos tipos de especificaciones y (c) que las especificaciones UML tienen un amplio soporte en las herramientas CASE, resulta evidente que sería conveniente integrar todos los tipos de especificaciones del metamodelo dentro del contexto de UML. Así, para cada tipo de entidad y tipo de enlace presentes en el metamodelo de trazabilidad se establecerá una correspondencia con un elemento de modelado en UML. En cada caso, se seleccionará

[1] Para facilitar la lectura de los tipos de enlace se ha asumido una dirección reflejada en el nombre de rol. Sin embargo, todos los enlaces pueden ser recorridos en ambos sentidos, tal como normalmente se hace con el enlace genérico "trace" cuya lectura podría ser "trace to" o "trace from", según la dirección que se utilice.

una metaclase del metamodelo de UML que se utilizará como clase base para establecer un estereotipo. Para aquellas entidades y relaciones del metamodelo que coincidan semánticamente con una metaclase de UML se utilizará directamente dicha metaclase para representarlos, sin definir un nuevo estereotipo. El resultado de este análisis es un *profile* UML asociado a nuestro metamodelo de trazabilidad. A continuación se detalla cómo se realiza dicha integración.

3.1 Entidades en UML

Para las entidades del metamodelo el elemento de modelado UML que las represente debería permitir asociaciones para poder establecer relaciones de agregación entre especificaciones. De acuerdo con esto, la elección debería estar entre los elementos de modelado UML que son especialización de *Classifier* . Para la entidad *Stakeholder* la elección es directa; el elemento de modelado *Actor* se utilizó como clase base para el estereotipo. Para las entidades correspondientes a especificaciones no-UML estándar se seleccionó el *Classifier* llamado *Artifact* (añadido en la versión 1.4 de UML). Nótese que *Artifact* ya tiene definidos algunos estereotipos, entre ellos «document» que es el que utilizaremos para representar documentos y secciones de documentos. Para las entidades que se corresponden directamente con elementos de modelado de UML (*UML_ UseCase* y *OtherUML_ Specification*) se utilizará el correspondiente elemento de modelado UML para representarlas. Por otra parte, para la agrupación y organización de artefactos UML y *Stakeholders* utilizaremos el elemento de modelado UML provisto para este fin, denominado *Package* y opcionalmente añadiremos los estereotipos predefinidos «model» o «subsystem» , según estemos definiendo un modelo de un sistema/subsistema o estaremos dividiendo un sistema en subsistemas, respectivamente. La Figura 2 muestra el contexto UML para los elementos de modelado *Actor* y *Artifact*.

3.2 Enlaces en UML

Los tipos de enlaces están indicados como asociaciones en nuestro metamodelo de trazabilidad y serán representados como elementos de modelado UML del tipo *Abstraction*, excepto para el caso del enlace *partOf*, el cual se repesentará mediante la agregación o composición entre especificaciones usando como clase base la metaclase *Association*. UML incluye varia dependencias predefinidas mediante estereotipos, éstas pueden verse en la Figura 2 en el comentario conectado con el subtipo de dependencia *Abstraction*. Aunque los distintos tipos de enlaces son modelados por diferentes asociaciones en el metamodelo de trazabilidad, éstas no son independientes, de hecho, el tipo de enlace *traceTo* es la generalización de todos los otros tipos de enlace. Así, *traceTo* se hará coincidir con el estereotipo «trace», ya definido de UML. Una dependencia «trace» indica una relación histórica o de process entre dos elementos que representan el mismo concepto sin especificar reglas de derivación entre ellos [11]. Exceptuando el tipo de enlace *partOf*, todos los otros enlaces de nuestro metamodelo de trazabilidad serán especializaciones del estereotipo predefinido «trace».

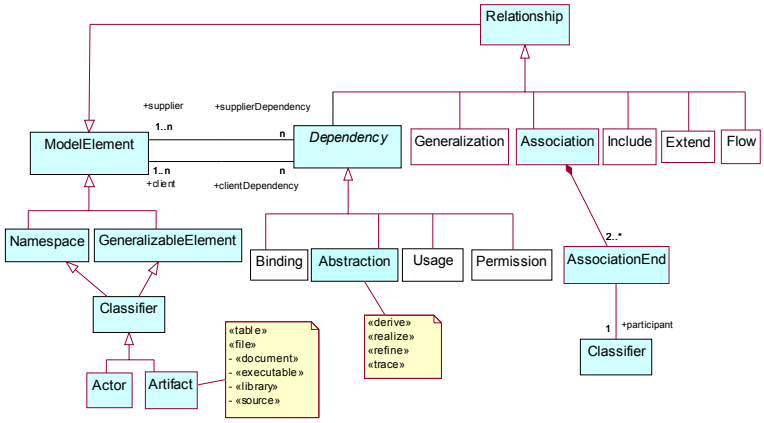

Fig. 2. Contexto UML para la integración del metamodelo de trazabilidad

3.3 Profile UML para Trazabilidad

De acuerdo a lo comentado en los aparados previos, en la Figura 3 y Figura 4 se muestra la representación UML de los tipos de entidades y los tipos de enlaces del metamodelo de trazabilidad. Esta representación constituye un *profile* UML para trazabilidad de requisitos.

4 Aplicando el Metamodelo con un Proyecto RUP

El metamodelo propuesto es independiente del proceso de desarrollo. Sin embargo, para ilustrar su aplicación hemos elegido RUP como proceso, principalmente por disponer en él de un gran detalle respecto de los artefactos. RUP es un producto de Rational Software basado en el Proceso Unificado de Desarrollo de Software [5].

Independiente del proceso de desarrollo que se utilice, respecto de trazabilidad de requisitos podemos identificar dos actividades: (a) configurar la trazabilidad a las necesidades del proyecto y (b) definir y explotar la información de trazabilidad durante el desarrollo y mantenimiento del software. Nos centraremos en la actividad de configuración haciendo uso de nuestro *profile* de trazabilidad.

A continuación se ilustrarán las tareas necesarias para configurar la trazabilidad de requisitos en un proyecto pequeño basado en RUP.

Tarea 1 Seleccionar el conjunto de artefactos a los cuales se les aplicará trazabilidad. Los artefactos de RUP que utilizaremos en nuestro ejemplo se muestran en la siguiente tabla:

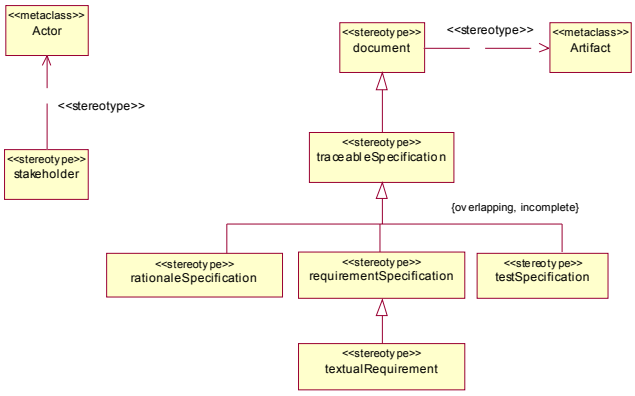

Fig. 3. Estereotipos para stakeholder y para especificaciones textuales básicas

Artefacto RUP	Estereotipo
Vision	«traceableSpecification»
Software Feature	«textualRequirement»
Supplementary Specification	«traceableSpecification»
Non-Functional Requirement	«textualRequirement»
Assumption	«rationaleSpecification»
Use Case Specification	«traceableSpecification»
Flow of Events	«flowOfEvents»
Use Case Model	«model»
Use Case	
Analysis & Design Model	«model»
Class	
Implementation Model	«model»
Component	
Data Model	«model»
Test Case	«testSpecification»

Cuando en la tabla no aparece el estereotipo es porque se utiliza exactamente el mismo elemento de modelado definido en UML. En el caso del estereotipo «model», tal como lo mencionamos anteriormente, se trata de un paquete UML con dicho estereotipo. El estereotipo «flowOfEvents» se introduce como nuevo estereotipo siendo clase hija del estereotipo «traceableSpecification».

Tarea 2 Definir las relaciones de agregación entre artefactos. Para los artefactos de nuestro ejemplo son las siguientes:

Vision ◇— Software Feature
Vision ◇— Assumption
Supplementary Specification ◇— Non-Functional Requirement

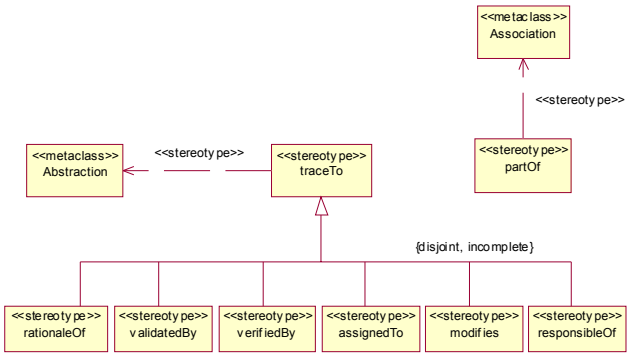

Fig. 4. Estereotipos para tipos de enlaces de trazabilidad

Use Case Specification ◇— Flow of Events
Use Case Model ◇— Use Case
Analysis & Design Model ◇— Class
Implementation Model ◇— Component
Data Model ◇— Table

Tarea 3 Establecer las dependencias de trazabilidad que son de interés para el proyecto. En nuestro ejemplo supondremos que son de interés las siguientes:

Stakeholder «responsibleOf» RUP Artifact
Stakeholder «modifies» RUP Artifact
Software Feature «traceTo» Use Case
Assumption «supports» Software Feature
Use Case «traceTo» Use Case Specification
Use Case «validatedBy» Test Case
Flow of Events «traceTo» Class
Class «traceTo» Component
Class «traceTo» Table
Class «verifiedBy» Test Case
Component «verifiedBy» Test Case

Hemos utilizado *RUP Artifact* para referirnos a cualquiera de los artefactos RUP seleccionados en la Tarea 1. El estereotipo «supports» se introduce como nuevo estereotipo siendo clase hija del estereotipo «rationaleOf».

Tarea 4 Definir criterios para deducir trazabilidad implícita [14]. En nuestro ejemplo utilizaremos coincidencia exacta de nombres entre artefactos para determinar dependencias de trazabilidad implícita. Esto se aplicará a los artefactos participantes en las siguientes dependencias:

Use Case «traceTo» Use Case Specification

Use Case «validatedBy» Test Case
Class «traceTo» Table
Class «verifiedBy» Test Case
Component «verifiedBy» Test Case

Además, se establecerán dependencias de trazabilidad implícitas entre artefactos compuestos cuando existan dependencias de trazabilidad entre sus componentes. En caso de existir más de un tipo de dependencia entre componentes, se establecerá «traceTo» entre los artefactos compuestos.

5 Trabajos Relacionados

Ramesh y Jarke en [10] ofrecen una amplia visión de la información necesaria para trazabilidad en proyectos industriales de desarrollo de software. Identifican dos segmentos de usuarios de trazabilidad y a partir de esto proponen dos metamodelos de trazabilidad (siendo uno la simplificación del otro). El metamodelo más amplio tiene 31 tipos de entidades (metaclases en el metamodelo de trazabilidad) y alrededor de 50 tipos de enlaces entre entidades. Hay que destacar que todas las especificaciones de análisis y diseño se representan sólo con un tipo de entidad, denominado "*system_ subsystem_ component*", es decir, no se presenta mayor granularidad ni conexión con alguna notación de modelado específica. Además de la complejidad asociada a la diversidad de elementos, no se provee una definición precisa ni para las entidades ni para los enlaces lo cual hace muy difícil su aplicación. Por otra parte, el metamodelo propuesto no ofrece mecanismos para adaptarlo a necesidades específicas de trazabilidad en un proyecto, y tal como lo proponen los autores dicha adaptación se realizaría rudimentariamente cortando o añadiendo partes del metamodelo.

Toranzo y Castro en [15] proponen un metamodelo para trazabilidad definido mediante múltiples vistas, cada una asociada a un determinado tipo de usuario de la información de trazabilidad (*Project Manager*, *Requirement Engineer* o *Software Engineer*). Sin embargo, tampoco en este trabajo se proveen mecanismos de extensión o adaptación del metamodelo. Además, el nivel de granularidad de los artefactos es muy grueso, trabajando en el ámbito de documentos y diagramas.

Spence y Probasco en [14] presentan varias estrategias de trazabilidad aplicables cuando se trabaja con un proceso dirigido por los Casos de Uso. Cada estrategia es descrita con un metamodelo para trazabilidad, estableciendo artefactos y dependencias de trazabilidad entre ellos. Todas las estrategias propuestas sólo ilustran dependencias de trazabilidad entre artefactos de requisitos, la trazabilidad hacia otros tipos de especificaciones se deja implícita al nivel de granuralidad que por defecto tiene un proceso dirigido por los Casos de Uso. Similarmente, Leite et al. en [7][8] proponen un marco de trabajo para la captura y organización de requisitos expresados en lenguaje natural. Se establecen

enlaces de trazabilidad entre los requisitos registrados pero tampoco se incluye trazabilidad hacia otros tipos de artefactos.

Por otra parte, las herramientas para gestión de requisitos[2] ofrecen un tratamiento satisfactorio para las especificaciones textuales pero presentan inconvenientes cuando se desea integrar dichas especificaciones con otras de análisis, diseño o implementación. Respecto de la integración con otros entornos (por ejemplo herramientas de modelado), normalmente esto se provee mediante importación/exportación de datos con diferentes formatos. La herramienta TOOR (*Traceability of Object-Oriented Requirements*), presentada por Pinheiro y Goguen en [12], está basada FOOPS, un lenguaje formal orientado a objeto. El usuario de la herramienta debe tener un entrenamiento previo en especificaciones FOOPS. Curiosamente, no se registra una continuación de TOOR posterior a dicha publicación, sin embargo, el enfoque formal utilizado y la funcionalidad descrita mantienen vigencia. En Rational RequisitePro (`www.rational.com`) se pueden enlazar especificaciones textuales no-UML con especificaciones UML dentro del repositorio de Rational Rose. Sin embargo, esto es posible sólo para Casos de Uso, es decir, el resto de los elementos de modelado dentro de los modelos en Rational Rose no son accesibles directamente desde RequisitePro. Mediante un lenguaje disponible para desarrollar extensiones podría mejorarse este aspecto. En RequisitePro sólo existe un tipo de enlace, que corresponde la dependencia genérica *trace*. Otra estrategia de integración entre una herramienta específica para gestión de requisitos y una herramientas CASE es la seguida por Telelogic DOORS (`www.telelogic.com`). Esta herramienta permite la conexión con diversas herramienta CASE para importar los elementos de los modelos en el repositorio de la CASE hacia el entorno de gestión de requisitos. En este caso está el inconveniente de tener que cambiar de entorno según se quiera gestionar requisitos o modelar y construir el software. Las herramientas comentadas presentan inconvenientes durante la configuración de la trazabilidad del proyecto. Por un lado, no están orientadas a un proceso de software ni a una notación determinada y por otro, aunque permiten definir tipos de requisitos, no se ofrece un marco de trabajo para dicha configuración. Así, toda las definiciones de la información de trazabilidad y su interpretación queda en manos de los usuarios de la herramienta.

6 Conclusiones

En este trabajo hemos presentado un marco de trabajo para trazabilidad que integra especificaciones textuales (para requisitos, fundamentos y pruebas) con especificaciones UML estándares, usando el propio contexto de UML. La importancia de contar con un marco de trabajo como el propuesto es doble, por un lado nos evita el comenzar la configuración de la trazabilidad desde cero en cada nuevo proyecto, y por otro, constituye una valiosa ayuda para tener una interpretación común cuando se trabaja en equipo. Desde el punto de vista de la información

[2] Para obtener un análisis detallado recomendamos visitar el sitio web del INCOSE (Internacional Council On System Engineering) www.incose.org/tools/tooltax.html

necesaria para trazabilidad de requisitos, en el metamodelo propuesto se ofrece un esquema esencial de entidades y enlaces (artefactos y dependencias) que puede ser adaptado a un proyecto específico usando los mecanismos de extensibilidad provistos por UML. Gracias a estar definido en UML, el metamodelo debería poder ser implantado en cualquier herramienta CASE que dé soporte a UML. El metamodelo para trazabilidad ha sido definido como un *profile* UML con lo cual puede ser incorporado y adaptado en una herramienta CASE. Actualmente estamos trabajando en el desarrollo de un módulo para implementar nuestro metamodelo extendiendo Rational Rose y siguiendo el proceso RUP.

References

1. J.-P. Corriveau. Traceability Process for Large OO Projects. IEEE Computer, pp. 63-68, September 1996.
2. R. Dömges and K. Pohl. Adapting Traceability Environments to Project-Specific Needs. Communications of ACM, Vol. 41, No 21, December 1998.
3. O. Gotel and A. Finkelstein. Extended Requirements Traceability: Results of an Industrial Case Study. I'n Proceedings of 3rd International Symposium on Requirements Engineering (RE97), IEEE Computer Society Press, pp. 169-178, 1997.
4. P. Haumer, M. Jarke, K. Pohl and K. Weidenhaupt. Improving reviews of conceptual models by extended traceability to captured system usage. Interacting with Computers, Elsevier Science, 13 (1) pp. 77-95, 2000. `ftp://sunsite.informatik.rwth-aachen.de/pub/CREWS/CREWS-99-16.ps.gz`
5. I. Jacobson, G. Booch and J. Rumbaugh. The Unified Software Development Process. Addison-Wesley, 1999.
6. M. Jarke. Requirements Tracing. Communications of the ACM, Vol. 41, No. 12, pp. 32-36, December 1998.
7. J.C. Leite and A.Oliveira. A Client Oriented Requirements Baseline. In Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE'95), pp. 108-115, IEEE Computer Society Press, 1995
8. J.C. Leite, G. Rossi, F. Balaguer, V. Maiorana, G. Kaplan, G. Hadad, A. Oliveros. Enhancing a Requirements Baseline with Scenarios. Requirements Engineering Vol. 2, No. 4, pp. 184-198, 1997.
9. B. Ramesh. Factors influencing requirements traceability practice. Communication of the ACM, Vol. 41, No. 12, pp. 37-44, December 1998.
10. B. Ramesh and M. Jarke. Toward Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering, Vol. 27, No. 1, pp.58-93, January 2001.
11. OMG Unified Modeling Language Specification. UML 1.4 with Action Semantics, Final Adopted Specification, January 2002. `www.omg.org`
12. F. Pinheiro and J. Goguen. An Object-Oriented Tool for Tracing Requirements. IEEE Software, pp. 52-64, March 1996.
13. K. Pohl. PRO-ART: Enabling Requirements Pre-Traceability. In Proceedings of the 2th International Conference on Requirements Engineering (ICRE'96), pp. 76-44, 1996.
14. I. Spence and L. Probasco. Traceability Studies for Managing Requirements with Use Cases. Rational Software White Paper, 1998. `www.rational.com/products/whitepapers/022701.jsp`
15. M. Toranzo and J. Castro. A Comprehensive Traceability Model to Support the Design of Interactive Systems. ECOOP Workshops 1999, pp. 283-284, Lecture Notes in Computer Science 1743, Springer, 1999.