Configurando la Trazabilidad de Requisitos para Proyectos basados en UML*

Patricio Letelier y Víctor Anaya
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, España
{letelier, vanaya}@dsic.upv.es

Resumen

La trazabilidad de requisitos es un reconocido factor de calidad en los procesos de desarrollo de software, sin embargo, su grado de aplicación y consenso en cuanto a prácticas varía considerablemente. Una de las principales razones es la falta de un marco de trabajo que sea preciso, simple y adaptable a las necesidades de un determinado proyecto. Esta carencia repercute en la dificultad para establecer la configuración de trazabilidad para un proyecto, en la cual esencialmente se determinan los tipos de artefactos que tendrán trazabilidad y los tipos de enlaces que se especificarán entre ellos. En este artículo se presenta un marco de trabajo para trazabilidad de requisitos que integra especificaciones textuales y elementos de modelado de UML. Se propone un *profile* UML, obtenido a partir de un metamodelo que refleja las necesidades de información para trazabilidad de requisitos. Este *profile*, gracias a los mecanismos de extensión de UML, proporciona un marco de trabajo fácilmente adaptable. Hemos incluido un pequeño ejemplo para mostrar la aplicación de nuestro enfoque para configurar la trazabilidad de requisitos de un proyecto, usando como proceso de software *Rational Unified Process* (RUP).

Keywords: Ingeniería de requisitos, trazabilidad de requisitos, gestión de requisitos, UML

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 debería estar integrada como subproceso dentro del proceso de desarrollo [1]. Para que este subproceso sea efectivo es imprescindible 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 gestión de requisitos puede ser un proceso muy costoso, con lo cual, debe configurarse 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 actividad.

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.

Respecto de la forma de expresarlos, tradicionalmente los requisitos se han especificado usando sobre todo especificaciones textuales en lenguaje natural. Consecuentemente, las herramientas de apoyo a la

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

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. Otra alternativa es especificar en diagramas (adicionales a los del modelo del sistema) las dependencias entre artefactos, dibujándolas una a una, lo cual es inviable, incluso para sistemas pequeños.

Por otra parte, respecto del modelado del software, UML [11] ha llegado rápidamente a convertirse en la notación más popular para modelado orientado a objeto. 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 artículo es presentar un marco de trabajo para configurar la trazabilidad de requisitos, integrando especificaciones textuales y elementos de modelado de UML. El enfoque propuesto es independiente del proceso de software que se desee utilizar y permite su adaptación a las necesidades de trazabilidad del proyecto.

Este trabajo está organizado en siete secciones. Después de esta introducción, la siguiente sección describe un metamodelo para trazabilidad. La sección tercera explica cómo las especificaciones textuales y los enlaces de trazabilidad se pueden definir en UML consiguiendo un marco homogéneo para toda la información de trazabilidad. A continuación, en la sección cuarta se presenta un proceso para la configuración de la trazabilidad y se define lo que entenderemos por trazabilidad explícita y trazabilidad implícita. En la sección quinta se ilustra la aplicación del enfoque propuesto utilizando el ejemplo de un pequeño proyecto basado en RUP. La sección sexta describe trabajos relacionados en el área de trazabilidad de requisitos y algunas herramientas representativas, específicas para la gestión de requisitos. Finalmente la sección séptima 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 no-funcional, un caso de uso, una clase, un atributo, etc. Esta granularidad para una *TraceableSpecification* se define mediante la agregación con el nombre de rol *partOf*.

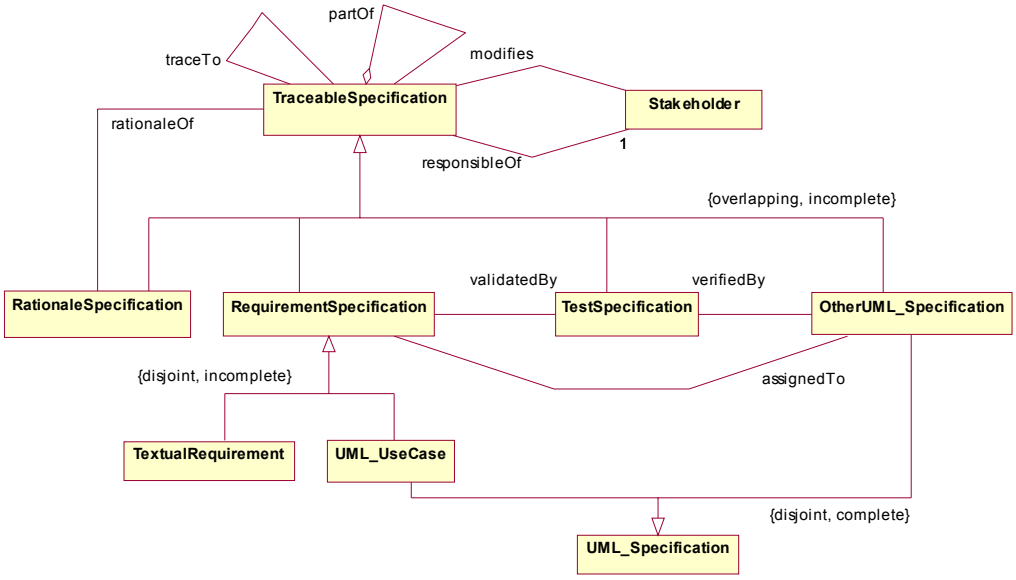

Figura 1: Un metamodelo para trazabilidad de requisitos

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). Las generalizaciones cuyas clases padre son *TraceableSpecification* y *RequirementSpecification* están definidas como "incompletas" para permitir otros tipos de especificaciones que puedan ser de interés para trazabilidad. Por ejemplo, algunos tipos de especificación no textuales ni UML son: vídeos, imágenes, voz, etc. Éstos corresponden usualmente a fundamentos, útiles durante la revisión y evaluación de modelos de análisis y diseño [4], sin embargo, podría tratarse simplemente de otro tipo de medio, por ejemplo, una *RequirementSpecification* registrada en un vídeo.

El tipo de enlace más genérico es la asociación con nombre de rol *traceTo* el cual permite establecer enlaces de trazabilidad entre *TraceableSpecifications*. El resto de los tipos de enlace (*modifies*, *responsibleOf*, *rationaleOf*, *validatedBy*, *verifiedBy* y *assignedTo*) son más específicos. El tipo de enlace *modifies* permite establecer una 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 tipo de enlace *validatedBy* relaciona a las *RequimementsSpecifications* con las correspondientes *TestSpecification* que las validan. Correspondientemente, el tipo de enlace *verifiedBy* determina las *TestSpecifications* que verifican una especificación UML. Por último, el tipo de enlace *assignedTo* determina a qué 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 incluidas en el trabajo de Ramesh y Jarke [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 los tipos de enlace *responsibleOf* y *modi-*

fies para determinar los *Stakeholders* involucrados. Para pre-trazabilidad se dispone del tipo de enlace *traceTo* entre requisitos en diferentes niveles de abstracción y *rationaleOf* para fundamentos asociados a dichas especificaciones de requisitos. La post-trazabilidad es soportada por los tipos de enlace *traceTo*, *validatedBy*, *verifiedBy*, *assignedTo* y *rationaleOf*.

3. El Metamodelo en el contexto de UML

Para que la aplicación del metamodelo sea sencilla y práctica es conveniente integrar todos los tipos de entidad y de enlace 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 una metaclase del metamodelo de UML que se utilizará como clase base para establecer un estereotipo. Para aquellos tipos de entidad y de enlace 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.

Entidades en UML Para los tipos de entidad del metamodelo el elemento de modelado UML que los 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 ha utilizado como clase base para el estereotipo. Para las entidades correspondientes a especificaciones no-UML estándar se ha seleccionado 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 los tipos de entidad 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 estemos dividiendo un sistema en subsistemas, respectivamente.

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 de la relación *partOf*, la cual se representa mediante la agregación o composición entre especificaciones usando como clase base la metaclase *Association*. 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 proceso entre dos elementos que representan el mismo concepto sin especificar reglas de derivación entre ellos [11]. Exceptuando el tipo de relación *partOf*, todos los tipos de enlace de nuestro metamodelo de trazabilidad serán especializaciones del estereotipo predefinido «trace».

De acuerdo a lo comentado en los apartados previos, en la Figura 2 y Figura 3 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. Configuración de la trazabilidad

En trazabilidad de requisitos podemos identificar dos actividades: (a) configurar la trazabilidad respecto de las necesidades del proyecto y (b) especificar 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* para trazabilidad (definido por el metamodelo de trazabilidad). El *profile* de trazabilidad

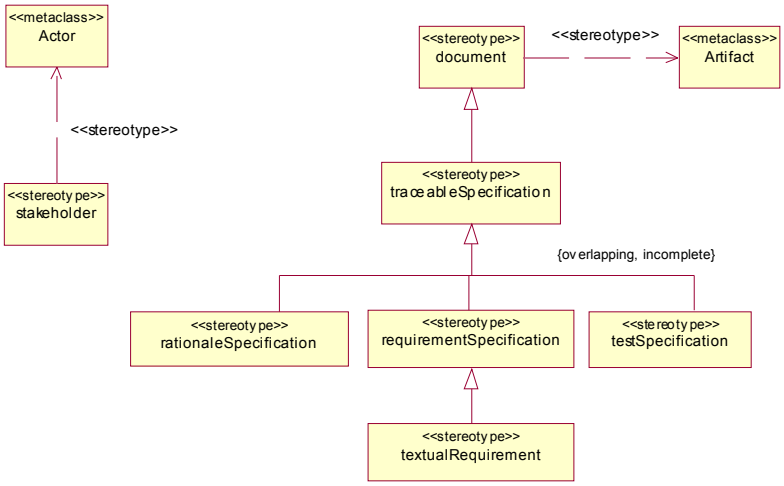

Figura 2: Estereotipos para *stakeholder* y para especificaciones textuales básicas

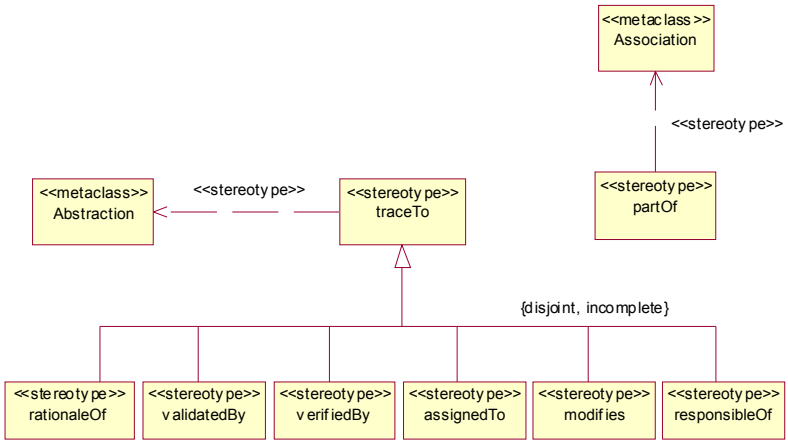

Figura 3: Estereotipos para tipos de enlaces de trazabilidad

actuará como marco para establecer los tipos de artefacto[1] entre los que se desea registrar trazabilidad y los tipos de enlaces de trazabilidad entre ellos. Consecuentemente, los enlaces de trazabilidad establecidos durante el modelado del sistema se verifican con respecto de la configuración de trazabilidad (por ejemplo, para un proyecto específico ciertos tipos de enlaces son válidos sólo entre determinados tipos de artefactos).

La configuración de trazabilidad para un proyecto incluye las siguientes tareas:

1. Seleccionar los tipos de artefactos para los cuales se realizará un seguimiento de trazabilidad. Se trata de un subconjunto del total de tipos de artefactos que se utilizarán en el proyecto. A cada tipo de artefacto se le asocia un estereotipo especializando uno de los estereotipos del *profile* de trazabilidad (excepto para aquellos tipos de artefacto que se corresponden directamente con un elemento de modelado UML). Al hacer esto, se está extendiendo el *profile* de trazabilidad en lo que respecta a tipos de artefacto (esto también se prodría hacer para los *stakeholders*, creando en este caso estereotipos especializados de *stakeholders*).

[1] A partir de aquí utilizaremos el témino "artefacto" en un sentido más amplio al dado en UML, tal como lo hacen la mayoría de los procesos de software (por ejemplo RUP). Así, consideraremos como artefactos a todos los documentos, ficheros y otros elementos físicos generados y/o usados durante el proyecto, pero además, llamaremos también artefactos a cualquier elementos de modelado UML.

2. Definir las relaciones de agregación entre tipos de artefactos. Esta tarea puede que no sea necesaria si dicha información está predefinida como parte de la descripción de cada tipo de artefacto en el metamodelo.

3. Establecer los tipos de enlaces de trazabilidad de interés para el proyecto. Éstos han de establecerse entre pares de tipos de artefactos seleccionados en la tarea 1. En este caso también puede ser necesario extender el *profile* de trazabilidad definiendo tipos de enlaces de trazabilidad más específicos.

4. Definir criterios para la derivación de enlaces de trazabilidad implícita y establecer qué enlaces de trazabilidad, de los indicados en la tarea 3, se desean generar por alguno de los criterios definidos. La definición de los conceptos de trazabilidad implícita y explícita se presentan en el apartado siguiente.

4.1. Trazabilidad explícita y trazabilidad implícita

Durante la configuración de trazabilidad sólo se especifican los tipos de enlace de trazabilidad que son relevantes para el proyecto (tal como lo define la tarea 3 antes mencionada). Sin embargo, aún con esta restricción el esfuerzo asociado a la introducción de los enlaces de trazabilidad puede ser considerable, con lo cual es deseable que se provean mecanismos que permitan derivar automáticamente parte de los enlaces de trazabilidad. Denominaremos trazabilidad explícita a aquellos enlaces de trazabilidad que son introducidos manualmente. Consecuentemente, llamaremos trazabilidad implícita a aquellos enlaces de trazabilidad que se derivan automáticamente de acuerdo con ciertos criterios.

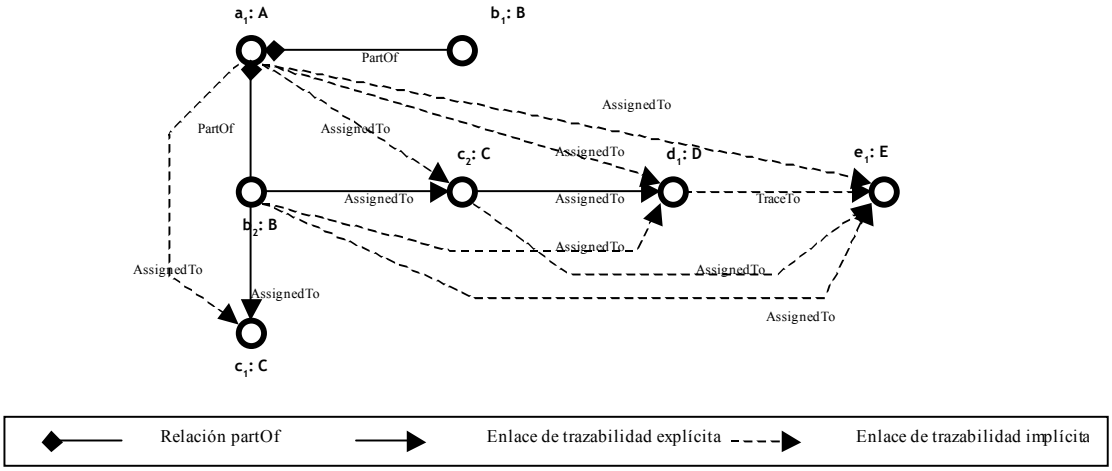

Figura 4: Ejemplo de grafo de trazabilidad

Antes de presentar algunos criterios para derivar trazabilidad implícita presentaremos un ejemplo. Supongamos que se tiene la siguiente configuración de trazabilidad (resultado de las tareas 1, 2 y 3):

Tipos de artefactos: A, B, C, D y E.

Relaciones de agregación: $A \diamond\!\!-\!\!- B$

Tipos de enlaces de interés:

$A \overset{assignedTo}{\rightarrow} C$, $A \overset{assignedTo}{\rightarrow} E$, $B \overset{assignedTo}{\rightarrow} C$, $C \overset{assignedTo}{\rightarrow} D$ y $D \overset{traceTo}{\rightarrow} E$

Supongamos además que se tienen los siguientes artefactos (con sus respectivos tipos): $a_1 : A$, $b_1 : B$, $b_2 : B$, $c_1 : C$, $c_2 : C$, $d_1 : D$ y $e_1 : E$. La Figura 4 muestra un grafo de trazabilidad para nuestro ejemplo, en el cual se ha introducido la siguiente trazabilidad explícita: relaciones de agregación $a_1 \diamond\!\!-\!\!- b_1$ y $a_1 \diamond\!\!-\!\!- b_2$, y los enlaces de trazabilidad $b_2 \overset{assignedTo}{\rightarrow} c_1$, $b_2 \overset{assignedTo}{\rightarrow} c_2$ y $c_2 \overset{assignedTo}{\rightarrow} d_1$. Veremos a continuación cómo puede derivarse trazabilidad implícita de manera que se obtengan automáticamente otros enlaces de trazabilidad, tal como se muestra en la Figura 4.

4.2. Algunos criterios para derivar trazabilidad implícita

En este apartado se explican tres criterios básicos para derivar trazabilidad implícita. Estos criterios son quizás los más intuitivos y directos pero no descartan la posibilidad de definir otros más elaborados.

a. Convención de nombre

Sean X e Y tipos de artefactos y sean x e y artefactos tal que $x \in X$ e $y \in Y$. Si $X \rightarrow Y$ es un tipo de enlace de interés, y satisfaciéndose un criterio dado de convención de nombre sobre dicho par de tipos de artefactos, entonces se deriva $x \rightarrow y$.

Supongamos que en el ejemplo de la Figura 4 se ha establecido el criterio de igualdad de nombre para el enlace de interés $D \overset{traceTo}{\rightarrow} E$. Además, si asumimos que los nombre de los artefactos d_1 y e_1 son iguales, entonces se deriva implícitamente el enlace entre el artefacto $d_1 \overset{traceTo}{\rightarrow} e_1$, tal como se muestra en la Figura 4.

b. Transitividad

Sean X, Y y Z tipos de artefacto y sean x , y, z artefactos tal que $x \in X$, $y \in Y$ e $z \in Z$. Si $X \rightarrow Y$, $Y \rightarrow Z$ y $X \rightarrow Z$ son tipos de enlaces de interés, de acuerdo con el criterio de transitividad se cumple que $(x \rightarrow y) \wedge (y \rightarrow z) \implies (x \rightarrow z)$.

En el grafo de la Figura 4 se muestran los enlaces implícitos por transitividad que se podrían derivar: $b_2 \overset{assignedTo}{\rightarrow} d_1$, $b_2 \overset{assignedTo}{\rightarrow} e_1$, $c_2 \overset{assignedTo}{\rightarrow} e_1$, $a_1 \overset{assignedTo}{\rightarrow} d_1$ y $a_1 \overset{assignedTo}{\rightarrow} e_1$. Sin embargo, tal como se realizó la configuración de nuestro ejemplo, sólo el último de ellos corresponde a un tipo de enlace de interés, con lo cual este se tomaría en cuenta y los otros se ignorarían.

c. Enlaces derivados por relaciones de agregación

Sean v y w artefactos, utilizaremos las siguientes definiciones:

$$components(w) = \{v \mid (w \diamond\!\!- v)\} \cup components(v)$$
$$aggregates(v) = \{w \mid (w \diamond\!\!- v)\} \cup aggregates(w)$$
$$allTraceTo(w) = \{v \mid (w \rightarrow v)\}$$

Sea X, Y y Z tipos de artefactos y sean x , y y z artefactos tal que $x \in X$, $y \in Y$ y $z \in Z$. Si $X \diamond\!\!- Y$ es una relación de agregación y $Y \rightarrow Z$ y $X \rightarrow Z$ son tipos de enlaces de interés, entonces se cumple que:

$$(x \diamond\!\!- y) \wedge (y \rightarrow z) \implies (y \rightarrow r) \wedge (y \rightarrow s) \wedge (x \rightarrow t)$$

$\forall\ r \in components(z)$, $\forall\ s \in aggregates(z)$ y $\forall\ t \in allTraceTo(y)$

En el grafo de la Figura 4 se obtienen los siguientes enlaces derivados según las relaciones de agregación establecidas: $a_1 \overset{assignedTo}{\rightarrow} c_1$, $a_1 \overset{assignedTo}{\rightarrow} c_2$, $a_1 \overset{assignedTo}{\rightarrow} d_1$ y $a_1 \overset{assignedTo}{\rightarrow} e_1$. Tal como antes, debido a que no se ha definido el tipo de enlace de interés $A \overset{assignedTo}{\rightarrow} D$, entonces el enlace implícito $a_1 \overset{assignedTo}{\rightarrow} d_1$ sería ignorado.

No sólo la especificación de enlaces de trazabilidad explícitos conlleva la derivación de enlaces implícitos, sino que a su vez la derivación de uno de éstos puede implicar la derivación de otros enlaces implícitos. Por ejemplo en el grafo de la Figura 4, el enlace derivado por transitividad $a_1 \overset{assignedTo}{\rightarrow} d_1$ se obtiene en base al enlace implícito por relación de agregación $a_1 \overset{assignedTo}{\rightarrow} c_2$. También puede ocurrir que el mismo enlace implícito pueda derivarse por más de un criterio, por ejemplo, el enlace implícito $a_1 \overset{assignedTo}{\rightarrow} e_1$ se obtiene por transitividad y por relación de agregación.

Es importante destacar que tanto por transitividad como por relaciones de agregación podría derivarse una multitud de enlaces implícitos. Sin embargo, probablemente muchos de ellos no sean relevantes. Por lo tanto, sólo serán considerados aquellos que correspondan a tipos de enlaces de interés, el resto serán ignorados.

Por otra parte, el tipo de enlace de trazabilidad que se asignará a un enlace derivado será el que se haya definido en el respectivo tipo de enlace de interés, definido en la configuración de la trazabilidad.

5. Configurando la trazabilidad para 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]. En RUP se proporcionan plantillas Word y HTML para los artefactos que son documentos, y para el modelado del software se utiliza UML.

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

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

Tipo de Artefacto RUP	Estereotipo desde el cual se especializa
Vision	«traceableSpecification»
Software Feature	«textualRequirement»
Supplementary Specification	«traceableSpecification»
Non-Functional Requirement	«textualRequirement»
Assumption	«rationaleSpecification»
Use Case Specification	«traceableSpecification»
Flow of Events	«traceableSpecification»
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.

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

Vision ◇— Software Feature
Vision ◇— Assumption
Supplementary Specification ◇— Non-Functional Requirement
Use Case Specification ◇— Flow of Events
Use Case Model ◇— Use Case
Analysis & Design Model ◇— Class
Implementation Model ◇— Component
Data Model ◇— Table

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

Stakeholder $\xrightarrow{\text{«responsibleOf»}}$ RUP Artifact
Stakeholder $\xrightarrow{\text{«modifies»}}$ RUP Artifact
Software Feature $\xrightarrow{\text{«traceTo»}}$ Use Case
Assumption $\xrightarrow{\text{«supports»}}$ Software Feature
Use Case $\xrightarrow{\text{«traceTo»}}$ Use Case Specification
Use Case $\xrightarrow{\text{«validatedBy»}}$ Test Case
Flow of Events $\xrightarrow{\text{«traceTo»}}$ Class
Class $\xrightarrow{\text{«traceTo»}}$ Component
Class $\xrightarrow{\text{«traceTo»}}$ Table

Class $\overset{\text{«verifiedBy»}}{\longrightarrow}$ Test Case
Component $\overset{\text{«verifiedBy»}}{\longrightarrow}$ Test Case

Hemos utilizado *RUP Artifact* para referirnos a cualquiera de los tipos de artefacto RUP seleccionados en la tarea 1. El estereotipo «supports» se ha introducido como un nuevo estereotipo, siendo éste una clase hija del estereotipo «rationaleOf».

Tarea 4 En nuestro ejemplo utilizaremos transitividad, relaciones de agregación y coincidencia exacta de nombres entre tipos de artefacto para determinar enlaces de trazabilidad implícita. Para este último caso, se aplicará el criterio de igualdad de nombre sólo para los siguiente tipos de enlaces de interés:

Use Case $\overset{\text{«traceTo»}}{\longrightarrow}$ Use Case Specification
Use Case $\overset{\text{«validatedBy»}}{\longrightarrow}$ Test Case
Class $\overset{\text{«traceTo»}}{\longrightarrow}$ Table
Class $\overset{\text{«verifiedBy»}}{\longrightarrow}$ Test Case
Component $\overset{\text{«verifiedBy»}}{\longrightarrow}$ Test Case

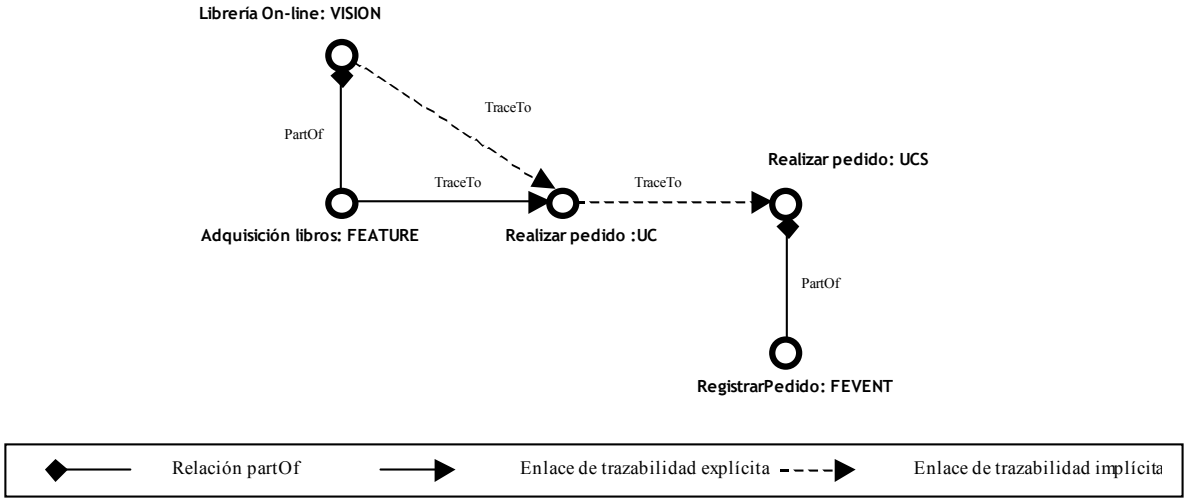

Figura 5: Ejemplo de grafo de trazabilidad de una librería on-line

Finalmente, según la configuración de la trazabilidad realizada, en un momento dado podríamos tener el grafo de trazabilidad que se muestra en la Figura 5. En dicha figura *VISION* corresponde al tipo de artefacto *Vision*, *FEATURE* se corresponde al tipo *Software Feature*, *UC* al tipo *Use Case*, *UCS* al tipo *Use Case Specification*, y *FEVENT* al tipo *FlowOfEvents*. El enlace $LibreríaOnLine : VISION \overset{traceTo}{\rightarrow} RealizarPedido : UC$ se deriva por la relación de agregación y el enlace $RealizarPedido : UC \overset{traceTo}{\rightarrow} RealizarPedido : UCS$ se deriva por convención de nombres.

6. 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 tipos de entidad. 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 los tipos de entidad ni para los tipos de enlace lo cual hace difícil su aplicación. Por otra parte, el metamodelo propuesto no ofrece mecanismos para configurarlo 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 para configurar la trazabilidad extendiendo o adaptando el 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 Casos de Uso (como es el caso de RUP). Cada estrategia es descrita con un metamodelo para trazabilidad, estableciéndose tipos de artefacto y de enlace de trazabilidad entre ellos. Todas las estrategias propuestas sólo consideran enlaces 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 Casos de Uso. Además, el único tipo de enlace que se utiliza es *traceTo*. 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 no 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, con herramientas de modelado), normalmente esta 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 se corresponde con 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 respecto de 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 quedan en manos de los usuarios de la herramienta.

7. Conclusiones

La trazabilidad de requisitos es la clave para lograr un proceso de gestión de requisitos exitoso. Sin embargo, no existe consenso respecto de las estrategias adecuadas para lograr una trazabilidad efectiva de requisitos. Estos y otros problemas hacen que en la práctica la trazabilidad de requisitos en proyectos de software presente diferentes grados de uso y aceptación. Consecuentemente, el soporte para trazabilidad en herramientas CASE no es el más adecuado. Históricamente, la investigación en gestión de requisitos ha ido en paralelo con la investigación en enfoques para desarrollo de software creando dos comunidades de investigación diferentes: la primera enfocada a mejorar el tratamiento de especificaciones textuales de requisitos sin entrar en detalles respecto de los elementos de modelado particulares de alguna propuesta metodológica, la otra enfocada a mejorar los métodos de análisis y diseño con técnicas estructuradas u orientadas a objeto, pero sin prestar demasiada atención a la gestión de requisitos ni a las especificaciones textuales. En la actualidad UML representa una buena oportunidad para aprovechar los resultados de estas dos comunidades y trasladarlos a un marco común para la especificación de trazabilidad y de su explotación.

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

En este trabajo hemos presentado un metamodelo de trazabilidad que integra especificaciones textuales (para requisitos, fundamentos y pruebas) con especificaciones UML estándares, usando el propio contexto de UML. Así, desde el punto de vista de la información necesaria para trazabilidad de requisitos, nuestro metamodelo ofrece un marco esencial para tipos de entidad y de enlace de trazabilidad, que puede ser adaptado a un proyecto específico usando los mecanismos de extensibilidad provistos por UML. El metamodelo para trazabilidad ha sido definido como un *profile* UML. Gracias a estar definido en UML, el metamodelo podría ser integrado en cualquier herramienta CASE que dé soporte a UML.

Por otra parte, y basándonos en el *profile* establecido, hemos propuesto un proceso para configuración de la trazabilidad de requisitos. Nuestra propuesta de metamodelo y de proceso de configuración son independientes del proceso de desarrollo utilizado. Sin embargo, hemos presentado un ejemplo utilizando RUP como proceso de desarrollo. Actualmente estamos trabajando en el desarrollo de un módulo para implementar nuestro metamodelo y proveer facilidades de configuración, utilizando las capacidades de extensión de Rational Rose y basándonos en el proceso RUP.

Un aspecto importante en la configuración de la trazabilidad del proyecto es el establecimiento de atributos de trazabilidad (y sus posibles valores) para cada uno de los artefactos seleccionados. Intencionalmente este tema no ha sido abordado porque consideramos que dicha tarea no presenta especial dificultad. Puede suponerse que la herramienta de gestión de requisitos ofrecerá un conjunto de atributos predefinidos para seleccionar y asociar a cada artefacto, y que también permitirá definir nuevos atributos. Por ejemplo, en RUP para una *software feature* algunos de los atributos que se proponen son: estado (propuesta, aprobada o incorporada), beneficio (crítica, importante o útil), esfuerzo estimado, riesgo y estabilidad (para estos últimos atributos se suelen usar valores tales como: alto, medio o bajo).

Referencias

[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-Verlag, 1999.