UStoReqIF: Conectando requisitos ágiles con tradicionales

Souhaila Serbout, Francisco Javier Bermúdez Ruiz, and Begoña Moros Valle

Universidad de Murcia
{serbout.souhaila,fjavier,bmoros}@um.es

Resumen A pesar de la reconocida importancia del uso de la Ingeniería de Requisitos en el éxito de proyectos de desarrollo de software, no está todavía claro el papel que juega dicha ingeniería en el desarrollo de software ágil, donde la técnica más utilizada para la especificación de requisitos son las historias de usuario. Sin embargo es importante documentar requisitos de alto nivel para no perder de vista la imagen general del sistema, siendo necesario conectar historias de usuario con requisitos tradicionales. Dado que los requisitos cuentan con una especificación que define un metamodelo de intercambio de datos denominado ReqIF, en este trabajo se propone tender un puente entre requisitos ágiles, basados en historias de usuario y tradicionales basados en especificaciones de requisitos textuales. La propuesta ha sido implementada haciendo uso de técnicas MDE. Se ha definido un metamodelo para representar historias de usuario, junto con un lenguaje textual y su correspondiente editor. Además se ha implementado una transformación modelo a modelo que permite obtener las especificaciones de requisitos conformes a ReqIF a partir del modelo de historias de usuario.

Keywords: Requirement Engineering · MDE · User Stories · ReqIF

1. Introducción

Las metodologías ágiles se están implantando de forma imperante en los equipos de desarrollo, que esperan con este enfoque, fundamentalmente, acelerar las entregas de software, mejorar la capacidad de gestionar los cambios en las prioridades e incrementar la productividad y la calidad del software. De acuerdo al 12º Informe Anual sobre el Estado de la Agilidad [16], el 97 % de las organizaciones que han participado en la encuesta aseguran que utilizan métodos de desarrollo ágiles y el 52 % declaran que más de la mitad de los equipos de desarrollo de su organización siguen las prácticas ágiles.

En este contexto, a pesar de la importancia que se reconoce que tiene la Ingeniería de Requisitos (RE) para el éxito de los proyectos de desarrollo de software, no está claro el papel que juega la RE en el desarrollo de software ágil [5,8]. En las metodologías ágiles, habitualmente se considera la RE como una actividad burocrática y que resta agilidad al proceso. Así, la técnica más ampliamente utilizada para la especificación de requisitos son las historias de usuario [12],

dejando de lado las especificaciones de requisitos textuales tradicionales. Esto supone que los métodos ágiles dependan de conocimiento tácito y comunicación cara a cara, en lugar de la documentación de requisitos que recoja de forma explícita el conocimiento compartido [8]. Del estudio realizado por Kasauli et al. [10] se desprende que es importante documentar los requisitos de alto nivel para no perder de vista la imagen general del sistema, pues no es fácil reconstruir el comportamiento deseado de un componente a partir de las historias de usuario o los casos de prueba y que, por tanto, éstos no son suficientes para la especificación de la funcionalidad del sistema. Además, estos requisitos deben mantenerse actualizados a medida que se modifican las historias de usuario, de lo contrario, los requisitos del sistema pronto quedan obsoletos y alejados del sistema real. La adopción de un enfoque sistemático para gestionar las actualizaciones de requisitos es uno de los grandes retos de la RE ágil [10].

Por otro lado, las técnicas MDE han demostrado ser efectivas para dar soporte a soluciones de interoperabilidad de datos ([1], [2], [6]) utilizando la definición de metamodelos para el intercambio de datos entre formatos y transformaciones de modelos para convertir los datos.

Dado que los requisitos cuentan con una especificación que define un metamodelo denominado ReqIF, estandarizado por el OMG [14] como modelo de intercambio de requisitos entre herramientas de gestión de requisitos, en este trabajo se propone aplicar las técnicas de MDE para tender un puente entre la RE ágil, basada en historias de usuarios y la RE tradicional, basada en especificaciones de requisitos textuales. El trabajo propuesto debe servir como primer paso para conseguir mantener la consistencia entre los requisitos de alto nivel del sistema y las historias de usuario. Para ello, será necesario definir un metamodelo que permita representar las historias de usuario, junto con un lenguaje textual y su editor para definir historias de usuario. Además se requiere definir una transformación modelo a modelo que permita obtener las especificaciones de requisitos conformes a ReqIF a partir del modelo de las historias de usuario. De esta forma, la especificación de requisitos de las historias de usuario puede importarse en cualquier herramienta de gestión de requisitos. El trabajo aquí descrito presenta una doble contribución: (1)una propuesta e implementación de un metamodelo de historias de usuario y su lenguaje textual (incluyendo soporte de edición) y (2) una implementación de un mapping para generar modelos de ReqIF desde historias de usuario creadas con el editor previamente definido.

La organización del trabajo es la siguiente: en la sección 2 se presentan los conceptos principales del metamodelo ReqIF, que serán necesarios para definir la transformación desde el modelo de historias de usuario; en la sección 3 se discuten los metamodelos encontrados en la literatura para modelar historias de usuario; en la sección 4 se describe la solución que se propone en este trabajo para la definición y edición de historias de usuario y su transformación a modelos de requisitos estándar; por último, en la sección 5, se exponen las conclusiones y los trabajos futuros.

2. ReqIF

ReqIF fue creado con el objetivo de intercambiar especificaciones entre herramientas de gestión de requisitos. El primer borrador del formato recibió el nombre de RIF en 2004 y cuando fue estandarizado por el OMG en 2011, su nombre cambió al de ReqIF. La principal ventaja de ReqIF es su independencia de la plataforma, la cual posibilita el intercambio global de requisitos. Desde el OMG se proporcionan versiones de ReqIF implementadas en XMLSchema y CMOF. El objetivo de las implementaciones es proporcionar una plataforma común que cubra todos los conceptos básicos de ReqIF para evitar esfuerzos innecesarios en la implementación del estándar. Actualmente existe una implementación Open Source de referencia: Requirements Modeling Framework (*RMF*) [9]. Se trata de la implementación Open Source de ReqIF más activa. Es una solución basada en EMF (Eclipse Modelling Framework) la cual proporciona una implementación del metamodelo de ReqIF en formato Ecore y una interfaz de usuario para la captura de requisitos denominada *ProR*. Dicha interfaz soporta la gestión de requisitos trabajando directamente con los datos del estándar ReqIF en su versión 1.0 sobre el metamodelo que implementa el propio proyecto RMF.

2.1. Metamodelo ReqIF

En la Figura 1 se muestra el metamodelo ReqIF, el cual contiene la clase `ReqIF` como raíz del metamodelo. Dicha clase contiene tres clases principales: `ReqIFHeader`, `ReqIFContent`, `ReqIFToolExtension`.

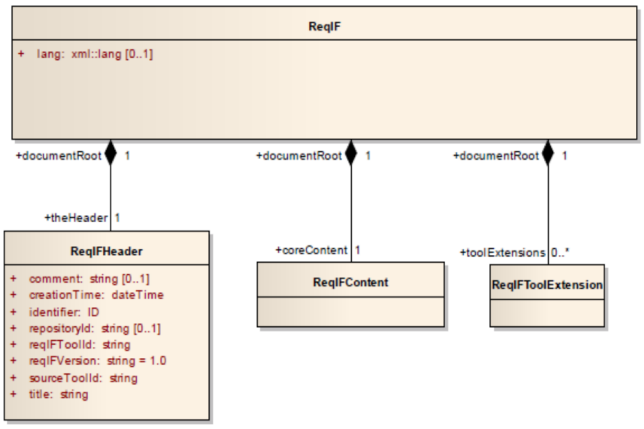

Figura 1. Elementos del metamodelo ReqIF

`ReqIFHeader` permite declarar los atributos de cabecera propios del metamodelo. El elemento `ReqIFToolExtension` permite que herramientas de terceros

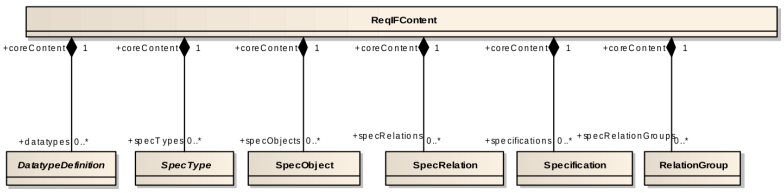

Figura 2. Contenido de la clase ReqIFContent

puedan introducir su información. Por último, el elemento `ReqIFContent` es el que contiene toda la información de la especificación de requisitos.

En la Figura 2 se muestra con mayor detalle el contenido de `ReqIFContent`. De manera resumida, `DatatypeDefinition` permite especificar nuevos tipos de datos. Los requisitos se representan mediante instancias de `Specification`. Los elementos `SpecObject` representan objetos de baja granularidad que constituyen la información del requisito. Las relaciones entre los requisitos se definen mediante instancias de `SpecRelation` mientras que los elementos `RelationGroup` permiten agrupar relaciones de requisitos. Todos los elementos anteriores tienen asociado un tipo que los define. Dicho tipo es subclase de `SpecType`, como se puede apreciar en la Figura 3.

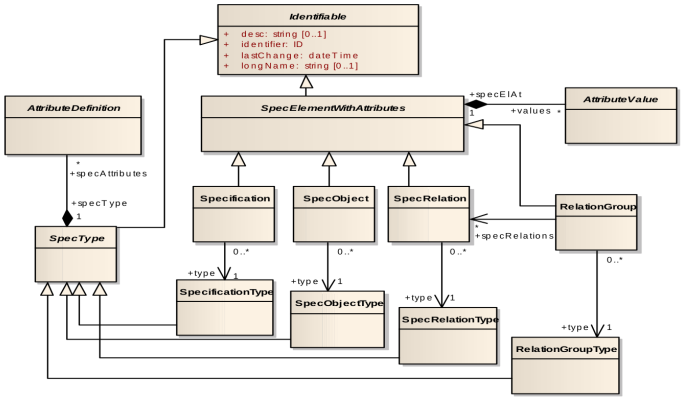

Figura 3. Contenido de la clase SpecType

Al igual que sucede con otros lenguajes extensibles (e.g. SPEM [13]), existen dos niveles de abstracción de elementos en el propio metamodelo: elementos para declarar tipos de objetos y elementos que representan las instancias de los tipos previamente declarados. En la sección 4 se describe cómo se usan estas clases en nuestra solución de interconexión de requisitos ágiles y tradicionales.

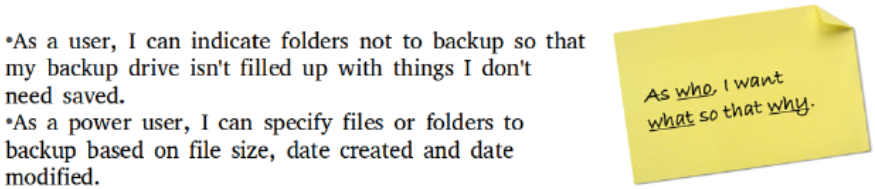

Figura 4. Ejemplo de una Historia de Usuario

3. Historias de usuario

Los métodos ágiles de requisitos consideran las historias de usuario como un artefacto clave para la captura de requisitos. Por lo general, se escriben en un formato informal [3]: "AS <Who> I WANT <What> [SO THAT <Why>]" (ver ejemplo en Figura 4). Se denotan en lenguaje natural elementos simples que las hacen operativas, fáciles de entender y por tanto eficientes. Son diferentes fuentes las que han abordado el uso de historias de usuario desde un enfoque práctico.

A falta de una especificación formal reglada y estandarizada, se ha considerado [3] como el trabajo de referencia sobre historias de usuario y su aplicabilidad práctica. En él su autor presenta el concepto de historias de usuario y las aplica en un proceso ágil de requisitos con el objetivo de ahorrar tiempo, reducir tareas y producir software de calidad. El libro explica cuán adecuadas son las historias de usuario para crear un software que satisfaga las necesidades de los usuarios debido a que proporcionan descripciones simples, claras y breves de la funcionalidad que resulta valiosa para los usuarios finales. Se muestra además cómo organizar historias de usuarios, priorizarlas y utilizarlas, incluyendo el uso de roles de usuario.

Aunque las historias de usuario deben ser simples y fáciles de entender, en [17] los autores indican que escribirlas mediante plantillas en forma de conceptos básicos relacionados con las dimensiones *Who*, *What* y *Why* en una frase en lenguaje natural puede ser problemático. Presentan, por tanto, un estudio de varias plantillas textuales encontradas en diversas fuentes de la literatura con el objetivo de alcanzar una unificación en la sintaxis de los conceptos usados para cada dimensión, un acuerdo en su semántica, así como elementos metodológicos que permiten aumentar la escalabilidad inherente de proyectos que están basados en el uso de historias de usuario. El trabajo presenta una propuesta de modelo unificado para historias de usuario, cuyo breve resumen se muestra en las Figuras 5 y 6.

Debido a que el trabajo pretende ofrecer un modelo unificado atendiendo a múltiples fuentes en la literatura, el modelo refleja ciertas incongruencias con respecto a los conceptos básicos descritos y aplicados en el libro de referencia [3]. La propuesta describe a dos niveles de detalle las historias de usuario, aunque incluye ciertas inexactitudes. Como se aprecia en la Figura 5, un elemento *User_Story* incluye 2 o 3 elementos `Descriptive_Concept` que representan las 3

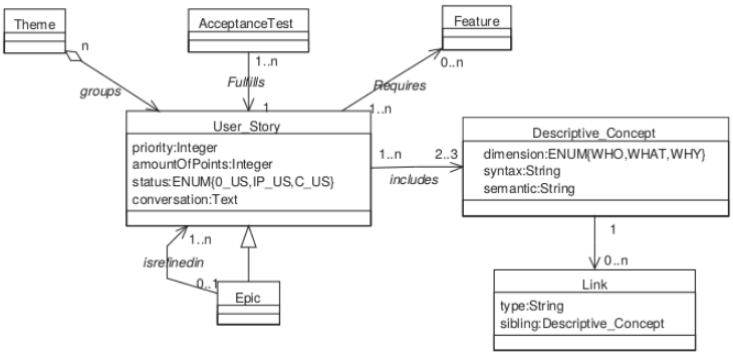

Figura 5. Historias de Usuario a macro nivel (obtenido de [17])

posibles dimensiones (*Who*, *What*, *Why*). Sin embargo, el metamodelo no limita cuáles de estas dimensiones pueden ser usadas en la descripción de la historia de usuario. Además define `Epic` como un tipo de historia de usuario que es a su vez agregado de historias de usuario. Sin embargo, y según la descripción de [3], un elemento `Epic` solo puede contener historias de usuario (algo contrario a lo expresado en el modelo unificado propuesto, donde `Epic` puede estar compuesto de *User_Story* y más `Epic`, por ser éste un subtipo).

Por último, como se puede ver en la Figura 6 donde se define una visión a menor nivel del contenido de una historia de usuario, el número de elementos `Task` que pueden ser secuenciados mediante la referencia `'so that'` es ilimitado. Esto también contradice a lo indicado en el libro de referencia [3].

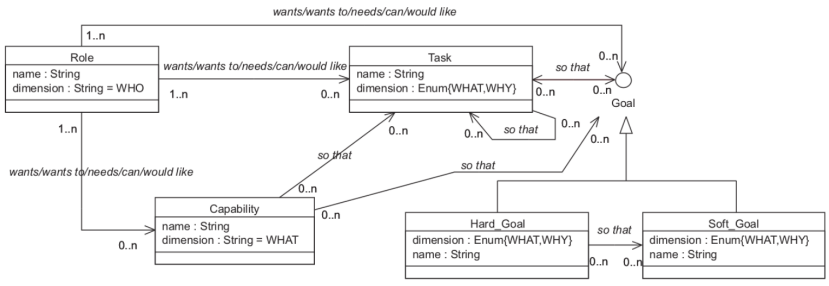

Figura 6. Historias de usuario a micro nivel (obtenido de [17])

El modelo unificado resultante refleja por tanto un amplio número de conceptos, y en ocasiones un uso confuso y no correcto de las bases que subyacen a las historias de usuario.

Por último, en [11], los autores definen un modelo conceptual de historias de usuario. El metamodelo es un esbozo de cómo con lenguaje natural se puede

describir una historia de usuario, por lo que se definen en términos de sujeto, verbo, adjetivo y no de dimensiones *Who*, *What* y *Why*. Dicha representación se aleja de la terminología propia de historias de usuario, resultando un metamodelo menos apropiado.

4. UStoReqif: Un Editor de Historias de Usuario con soporte de ReqIF

El trabajo aquí propuesto se basa en el uso tanto de las técnicas MDE como de los modelos de ingeniería de requisitos. Aborda el enfoque de exportar información textual de requisitos definidos mediante métodos ágiles a modelos de requisitos usados en metodologías de requisitos tradicionales.

Por tanto, el objetivo es modelar las historias de usuario empleadas en metodologías de requisitos ágiles y transformarlos en requisitos definidos globalmente en formato ReqIF, usados en otras metodologías tradicionales. Para lograr dicho objetivo se requerirán abordar tres pasos.

1. Estudiar el metamodelos ReqIF y seleccionar una implementación.
2. Proporcionar una representación adecuada para implementar modelos de historias de usuario. Además sería deseable proporcionar un lenguaje textual para describir los modelos de historias de usuario (agregando una sintaxis concreta al metamodelo de historias de usuario). Dicho lenguaje debe disponer de un editor propio.
3. Implementar una solución de interoperabilidad a nivel de modelos basada en una arquitectura generativa compuesta de transformaciones texto-a-modelo, y modelo-a-modelo. La solución debe exportar los requisitos definidos mediante historias de usuario al formato estándar ReqIF para que puedan ser usados en cualquier entorno de gestión de requisitos tradicional con soporte de ReqIF (e.g IBM Doors [1] y PTC Integrity [2]).

4.1. Metamodelo de Historias de Usuario

Tal y como se ha presentado previamente, en [17] encontramos una propuesta de un metamodelo que no respeta la simplicidad propia de las Historias de Usuario y que utiliza algunos conceptos no conformes a [3].

La propuesta aquí planteada respeta los conceptos básicos relacionados con historias de usuario. Dado que se usan fundamentalmente en metodologías ágiles, el metamodelo propuesto incluye el concepto de Product Backlog usado en la metodología ágil Scrum [15], como elemento aglutinador y gestor de historias de usuario y epics.

A continuación se enumeran las consideraciones adoptadas para la definición del metamodelo de historias de usuario, tratando de ser fieles al libro de referencia [3].

[1] https://www.ibm.com/es-es/marketplace/requirements-management
[2] https://www.ptc.com

- Según la plantilla y definición propuesta en [3], una historia de usuario es un elemento que puede ser largo o corto.
- Por tanto, desde una punto de vista de alto nivel, definimos historia de usuario y epic como elementos que se pueden estimar y priorizar, pudiendo ser eliminados o añadidos a la colección de elementos representada mediante *Product Backlog*.
- Según la definición de *Epic*, se trata de una historia de usuario larga compuesta por un gran número de historias de usuario (pero no por más *Epic*).
- Además de eso, se puede recopilar un conjunto de historias de usuario mediante epic representando un tipo de funcionalidad similar a través de un componente llamado *Theme*.
- Desde un punto de vista de bajo nivel, una historia de usuario es un concepto que se compone de dos conceptos o dimensiones obligatorios (*Who*, *What*) y otro opcional (*Why*).
- Por último, una historia de usuario también debe cumplir con los Criterios de Aceptación (*Acceptance*) definidos y puede estar vinculado a recursos externos como gráficos, figuras, imágenes, videos.. (*Link*).

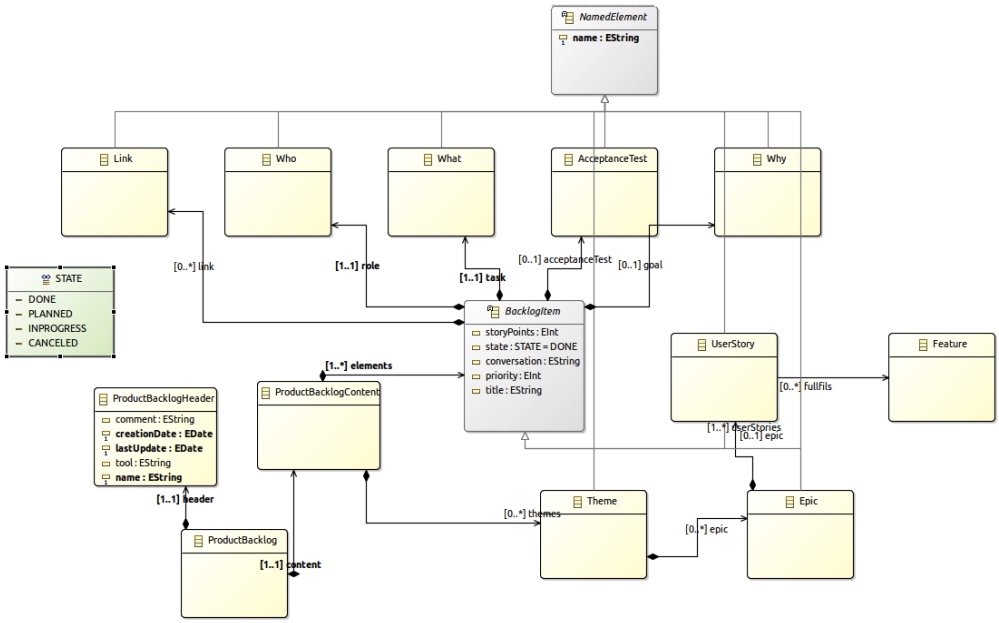

Figura 7. Metamodelo de Historias de Usuario propuesto

En la Figura 7 se puede ver un resumen del metamodelo propuesto para la descripción de historias de usuario tras considerar principios básicos como la simplicidad y la facilidad de comprensión. `ProductBacklog` es el elemento raíz

del metamodelo, conteniendo atributos de cabecera en `ProductBacklogHeader` y un conjunto de elementos `BacklogItem` y `Theme` como contenido de `ProductBacklogContent`. El elemento `Theme` representa a una colección de `Epic` similares en funcionalidad. La clase principal viene representada por la clase abstracta `BacklogItem`, superclase de los elementos `UserStory` y `Epic`. Esta clase permite mediante atributos recoger información del item sobre: estimar el esfuerzo (`storyPoints`), priorizar dentro del `ProductBacklog` (`priority`), indicar el estado actual (`state`), además de incluir un título y texto de la conversación que representa cada item. Por otro lado, `UserStory` es un elemento que puede verificar ciertas características `Features`, mientras que `Epic` puede contener múltiples `UserStory`, tal y como se especifica en la referencia de historias de usuario. Por último, `BacklogItem` contiene obligatoriamente un elemento `Who` y `What`, y opcionalmente un elemento `Why`. Además, puede contener varios recursos externos (`Link`)) y puede contener una prueba de aceptación (`AcceptanceTest`).

4.2. Un lenguaje específico de dominio para Historias de Usuario

```
PRODUCT BACKLOG :
    HEADER {name : M_COHN_NEWS_EXAMPLE
            creationDate : 2018-01-01
            lastUpdate : 2018-01-05
    }
    CONTENT {
        elements {
            USERSTORY : US1 = AS A  site_visitor I WANT TO "read current news on the home page"
                                        SO THAT "I stay current on agile news";
            USERSTORY (INPROGRESS) : US2 = AS A site_visitor I WANT TO " access old news that is no longer on the home page"
                                                SO THAT "I can access things I remember
                                                        from the past or that others mention to me";
            USERSTORY (DONE) : US3 = AS A site_visitor I WANT TO "email news items to the editor"
                                                SO THAT "they can be considered for publication";
            USERSTORY (PLANNED) : US4 = AS A site_editor
                                    I WANT TO "set the following dates on a news item:
                                            Start Publishing Date, Old News Date, Stop Publishing Date.
                                            These dates refer to the date an item becomes visible on the site
                                            (perhaps next Monday), the date it stops appearing on the home page,
                                            and the date it is removed from the site (which may be never)."
                                    SO THAT "articles are published on and through appropriate dates";
            USERSTORY (CANCELED) : US5 = AS A site_member I WANT TO "subscribe to an RSS feed of news (and events?)";

            EPIC (PLANNED) : EP1 = AS A site_editor I WANT TO "show items displayed on the front page based on priority"
                [USERSTORY (INPROGRESS) : US1_1 = AS A site_editor I WANT TO "assign priority numbers to news items"]
                [USERSTORY (PLANNED) : US1_2 = AS A site_editor I WANT TO "indicate which articles I want featured
                                                                    most prominently on the site"];

        }
    }
```

Figura 8. Ejemplo de la sintaxis del DSL propuesto

En este trabajo se propone además una sintaxis concreta y una herramienta de edición que se ajusta al metamodelo de las historias de usuario definidas en la sección anterior. Se ha elegido una gramática cercana a la terminología utilizada por el metamodelo.

El lenguaje se definió utilizando la herramienta para creación de lenguajes específicos de dominio (DSLs) textuales *Emftext* [7]. La Figura 8 muestra un ejemplo de historias de usuario (extraído del libro de ejemplos de M.Cohn [3])

[3] https://learn.mountaingoatsoftware.com/200-user-stories-example/

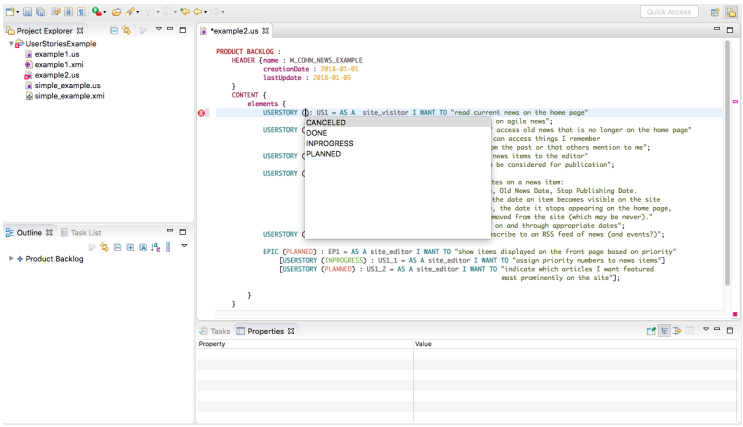

Figura 9. Editor de Historias de Usuario (autocompletado)

haciendo uso del lenguaje implementado. Emftext permite diseñar de forma sencilla una herramienta de edición completa con funciones de autocompletado y coloreado de sintaxis. La Figura 9 muestra el editor generado con Emftext en acción. El modelo de Historias de Usuario generado por el editor y que se corresponde con el texto introducido en el ejemplo se muestra en la Figura 10. El proyecto implementando el DSL puede descargarse aquí [4].

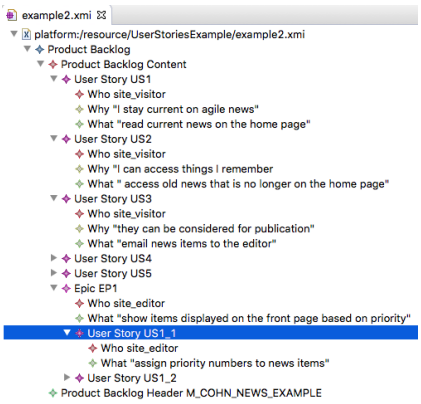

Figura 10. Modelo de Historias de Usuario obtenido con el editor

[4] https://github.com/jisbd19/USEditor

4.3. Transformación de modelo de historias de usuario a modelo de ReqIF

Se ha implementado una transformación modelo-a-modelo mediante el lenguaje *RubyTL* [4]. El primer paso de la transformación consiste en crear las definiciones de atributos y tipos que serán requeridos en ReqIF para representar las historias de usuario usando los conceptos del metamodelo de ReqIF. Será necesario crear en el modelo ReqIF el tipo de datos del enumerado de estados de historias de usuario, así como el tipo *String* que contendrá los valores de cadena de los atributos. Posteriormente se crean directamente en el modelo los atributos necesarios para los tipos de objeto que representarán las historias de usuario y epics (`AttributeDefinitionEnumeration` y `AttributeDefinitionString`). Una vez definidos los tipos de datos y los atributos que definirán los tipos que representan historias de usuario y epics, se procede a crear dichos tipos: (1) para *UserStory* se crea el tipo `SpecObjectType` y para *Epic* se crea `SpecificationType` (ver Figura 11).

```
WHO_ATT_US = REQIF::AttributeDefinitionString.new(
      :desc => 'WHO',
      :identifier => 'WHO',
      :editable => false,
      :type => STRING
   )

WHAT_ATT_US = REQIF::AttributeDefinitionString.new(
      :desc => 'WHAT',
      :identifier => 'WHAT',
      :editable => false,
      :type => STRING
)

WHY_ATT_US = REQIF::AttributeDefinitionString.new(
      :desc => 'WHY',
      :identifier => 'WHY',
      :editable => false,
      :type => STRING
)

EPIC_TYPE = REQIF::SpecificationType.new(
            :identifier => 'Epic',
            :desc => 'Epic',
            :specAttributes => [STATE_ATT_EP, WHO_ATT_EP, WHAT_ATT_EP, WHY_ATT_EP ]
            )

USERSTORY_TYPE = REQIF::SpecObjectType.new(
            :identifier => 'UserStory',
            :desc => 'UserStory',
            :specAttributes => [STATE_ATT_US, WHO_ATT_US, WHAT_ATT_US, WHY_ATT_US ]
            )
```

Figura 11. Ejemplo de declaración de tipos en ReqIF para las Historias de Usuario

Una vez que se definen los tipos, se procede a definir el mapping entre las clases del metamodelo de historias de usuario (a la izquierda) y el metamodelo ReqIF (a la derecha):

- ProductBacklog → ReqIF
- ProductBacklogHeader → ReqIFHeader
- ProductBacklogContent → ReqIFContent

El elemento raíz y su contenido son similares en ambos metamodelos, por lo que el mapping resulta casi directo.

- Epic → Specification
- UserStory → SpecObject
- Who,Why,What → SpecType

```
rule 'usepicTOrspecifications' do
  from US::Epic
  to REQIF::Specification
  #filter { |bli| puts 'COND1'; bli.kind_of?(US::Epic); puts 'COND2' }
  mapping do | usepic, rspec |
    rspec.identifier = usepic.name
    rspec.desc = usepic.name
    rspec.longName = usepic.name
    rspec.type = EPIC_TYPE
    rspec.values = REQIF::AttributeValueEnumeration.new(
                        :definition => STATE_ATT_EP,
                        :values => STATE.specifiedValues.find{ |eValue| eValue.identifier = usepic.state.name})
    rspec.values = usepic.role
    rspec.values = usepic.goal
    rspec.values = usepic.task
    rspec.children = usepic.userStories
  end
end
```

Figura 12. Regla de transformación de Epic a Specification

`UserStory` y `Epic` se transforman de modo similar (ver Figura 12) a elementos de tipo `Specification` y `SpecObject` respectivamente. La mayor diferencia reside en que los elementos `Specification` disponen de un referencia multivaluada `children` de tipo `SpecHierarchy` que permite referenciar a un `SpecObject`. Esta relación permite representar que un `Epic` contiene varios `UserStory`. Es necesario en este punto usar el mecanismo de fases de RubyTL para, (1) en una primera fase añadir al elemento `ReqIFContent` los `SpecObject` de las historias de usuario contenidas en el `Epic` y (2) en una segunda fase referenciar en el elemento `Specification` del Epic las historias de usuario añadidas previamente en la primera fase (nótese que todos los `SpecObject` deben estar añadidos al elemento `ReqIFContent`, no importa si son historias de usuario contenidas o no en un epic).

- Theme → RelationGroup
- AcceptanceTest,Feature,Link → SpecType

Por último, se emplean `RelationGroup` para representar `Theme`. Esto es porque nos permite agrupar `Epic` que son representadas mediante `Specification`. La transformación de `AcceptanceTest,Feature,Link` se hace usando elementos `Spectype`. El proyecto implementando la transformación se encuentra disponible aquí [5]. En la Figura 13 se puede ver el modelo ReqIF generado para el ejemplo anterior.

5. Conclusiones y vías futuras

El trabajo descrito en este documento tiene como objetivo utilizar la solidez de un estándar para obtener una manera confiable de intercambiar requisitos

[5] https://github.com/jisbd19/UStoReqIF

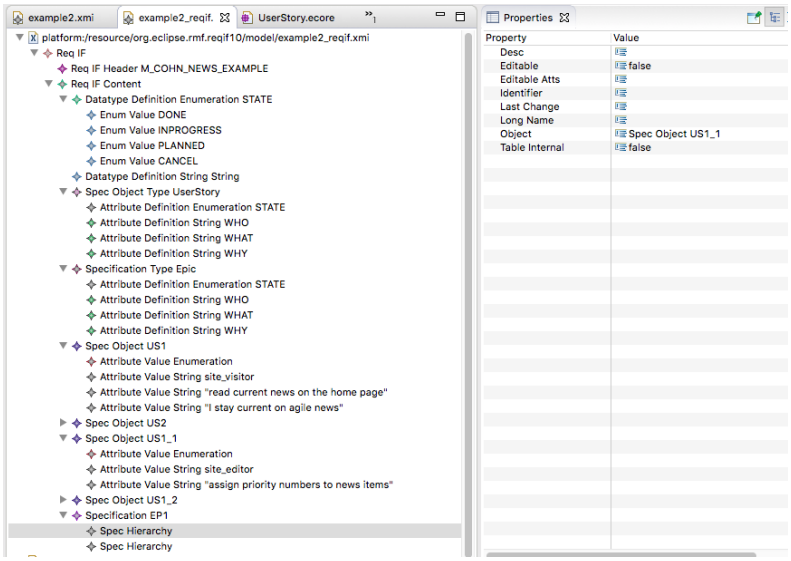

Figura 13. Modelo ReqIF resultado de la transformación

entre entornos heterogéneos. En este caso, nos interesó proporcionar enlace entre los requisitos ágiles en forma de historias de usuario y un enfoque más tradicional soportado mediante el formato ReqIF. Para lograr este objetivo, se ha definido un metamodelo de historias de usuario junto con un lenguaje textual (sintaxis concreta y herramienta de edición) que lo soporte. Finalmente, se ha diseñado e implementado la transformación modelo-a-modelo entre modelos de historias de usuario y modelos de ReqIF.

Aunque se han implementado una decena de ejemplos de historias de usuario con nuestro lenguaje (todos ellos extraídas del libro de ejemplos de M.Cohn [6]), el trabajo actual presenta como principal vía futura proporcionar una validación consistente a través de la importación de un mayor número de historias de usuario (se ha planeado usar al menos los 200 ejemplos de M.Cohn). Mediante un proceso de validación automática y manual se verificará que: (1) toda la información representada en modelos de historias de usuario se encuentra recogida en los modelos ReqIF y (2) que la representación de historias de usuario en modelos de ReqIF es semánticamente correcta. Para este último paso será necesario importar los modelos generados en herramientas de gestión de requisitos que trabajen con ReqIF (*DOORS*, por ejemplo) e involucrar a expertos en requisitos para validar la representación de los datos. Otra vía futura importante consiste en mantener sincronizados los modelos de historias de usuario y ReqIF en las dos direcciones. Es decir, extender la actual implementación para proporcionar una transformación adicional de ReqIF a historias de usuario.

[6] https://www.mountaingoatsoftware.com/agile/user-stories

Agradecimientos

Esta investigación forma parte del proyecto GINSENG-UMU (TIN2015-70259-C2-2-R) financiado por el Ministerio de Economía y Competitividad y el Fondo Europeo de Desarrollo Regional (FEDER).

Referencias

1. Bermudez, F.J., García, J.J., Díaz, O.: Data integration between objectiver and dbmain: A case study of a model-driven interoperability bridge pp. 477–488 (2016). https://doi.org/10.5220/0005653504770488
2. Bruneliere, H., et al.: Towards model driven tool interoperability: Bridging eclipse and microsoft modeling tools. In: 6th ECMFA 2010, Paris, FRA, June 15-18, 2010. pp. 32–47 (2010). https://doi.org/10.1007/978-3-642-13595-8
3. Cohn, M.: User Stories Applied: For Agile Software Development. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA (2004)
4. Cuadrado, J.S., et al.: Rubytl: A practical, extensible transformation language. In: ECMDA-FA. Lecture Notes in Computer Science, vol. 4066, pp. 158–172. Springer (2006). https://doi.org/10.1007/11787044_13
5. Curcio, K., Navarro, T., Malucelli, A., Reinehr, S.S.: Requirements engineering: A systematic mapping study in agile software development. Journal of Systems and Software **139**, 32–50 (2018)
6. Fabro, M.D.D., et al.: Model-driven tool interoperability: An application in bug tracking. In: On the Move to Meaningful Internet Systems: CoopIS, DOA, GADA, and ODBASE, Montpellier, FRA, Oct 29 - Nov 3, 2006. Proc. Part I. pp. 863–881
7. Heidenreich, F., Johannes, J., Karol, S., Seifert, M., Wende, C.: Model-Based Language Engineering with EMFText, pp. 322–345. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35992-7
8. Inayat, I., Salim, S.S., Marczak, S., Daneva, M., Shamshirband, S.: A systematic literature review on agile requirements engineering practices and challenges. Computers in Human Behavior **51**, 915–929 (2015)
9. Jastram, M., Graf, A.: Requirements modeling framework. eclipse magazine. (2011). https://doi.org/6.11. 2011.
10. Kasauli, R., Liebel, G., Knauss, E., Gopakumar, S., Kanagwa, B.: Requirements engineering challenges in large-scale agile system development. 2017 IEEE 25th International Requirements Engineering Conference (RE) pp. 352–361 (2017)
11. Lucassen, G., Dalpiaz, F., van der Werf, J.M.E.M., Brinkkemper, S.: Improving agile requirements: the quality user story framework and tool. Requirements Engineering **21**(3), 383–403 (Sep 2016). https://doi.org/10.1007/s00766-016-0250-x
12. Medeiros, J., et al.: Requirements engineering in agile projects: A systematic mapping based in evidences of industry. In: CIbSE (2015)
13. OMG: Software Process Engineering Metamodell SPEM 2.0. Tech. rep. (2006)
14. OMG: Reqif requirements interchange. https://www.omg.org/spec/ReqIF/ (2016)
15. Schwaber, K.: Scrum development process. In: Business Object Design and Implementation. pp. 117–134. Springer London, London (1997)
16. VersionOne: 12th annual state of agile report (2018), https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report, [Online; accessed 7-April-2019]
17. Wautelet, Y., et al.: Unifying and extending user story models. In: CAiSE. pp. 211–225. Springer International Publishing, Cham (2014)