Una Ontología para la Gestión de Versiones de Familias de Producto

M. Soledad Sonzini[1,2], Marcela Vegetti[1], Horacio P. Leone[1]

[1] INGAR, Instituto de Desarrollo y Diseño (3000), Avellaneda 3657, Santa Fe, República Argentina.
[2] Universidad Nacional de La Rioja (5300), Luis M. de la Fuente S/N, La Rioja, República Argentina.
[1]{ssonzini, mvegetti, hleone}@santafe-conicet.gob.ar

Resumen. La gestión del ciclo de vida de producto (PLM - Product Lifecycle Management) requiere un conjunto de soluciones para representar e integrar la información de una familia de producto. Las soluciones deben considerar los cambios en los requerimientos de los usuarios y del dominio, los cuales pueden ocurrir por diversas causas. Cada cambio modifica la información de producto, generando una nueva versión del mismo. Por lo tanto, la propuesta de este trabajo tiene por objetivo introducir un enfoque basado en ontologías para la gestión de versiones de información de familia de productos a lo largo de su ciclo de vida. Esta propuesta plantea conceptos generales para la captura y representación de los cambios, independientemente del modelo de productos utilizado para la representación de estas familias. Asimismo, se presenta la aplicación del modelo propuesto en dos modelos de productos diferentes: la ontología de productos PRONTO (PRoduct ONTOlogy) y el modelo de características (FM – Feature Model).

Palabras Clave: Versión de Producto, Variabilidad, Ontología, Familia de Productos, PRONTO, Product Lifecycle Management, Feature Model.

1 Introducción

La administración del ciclo de vida de producto (PLM) es una actividad eficiente para gestionar toda la información relacionada con un producto desde su inicio, desarrollo, madurez, hasta su final [1]. Es importante gestionar la información de los productos en todas las fases de su vida debido a que un simple cambio en el mismo durante una determinada fase podría propagarse y afectar la consistencia e integridad de la información en el resto de las fases. Los sistemas PLM requieren soluciones robustas para representar de forma consistente los modelos de datos de producto y así poder intercambiarlos con otras organizaciones, stakeholders, procesos, etapas en el ciclo de vida.

En muchos casos es necesario que un producto cambie con el propósito de dar soporte a diferentes segmentos del mercado, adoptar nuevas tecnologías, mejorar una

característica para competir en el mercado o modificar el diseño del producto. Esta capacidad de cambio de un producto recibe el nombre de *variabilidad*, definido como la habilidad de un producto para ser eficientemente extendido, cambiado, personalizado o configurado para su utilización en un dominio particular [2]. Pohl y colab. en [3] sostienen que es fundamental hacer una distinción entre *variabilidad en el tiempo* y *variabilidad en el espacio.* La primera de ellas es definida como "la existencia de diferentes versiones de un artefacto que es válido en diferentes tiempos", este tipo de variabilidad denota la evolución de un artefacto definiendo puntos de variación que ayudan a mantener el control del impacto de pequeños cambios. La variabilidad en el espacio es definida como "la existencia de un artefacto en diferentes formas en un mismo tiempo". Esta dimensión abarca de forma simultánea el uso de un artefacto variable en diferentes formas, es decir diferentes variantes de productos. Varias propuestas se han presentado en diferentes dominios para la representación de la variabilidad en el espacio. En el dominio de las industrias de manufactura varias propuestas recurren al concepto de familia de productos [4, 5, 6, 7, 8] para gestionar más eficientemente la variabilidad. En tanto, en la industria del software es muy utilizado el modelo de características (FM- Feature Model) y el modelo de variabilidad ortogonal (OVM) [3], entre otras.

Otras propuestas, en cambio, se centran en el abordar la problemática de la variabilidad en el tiempo. En el dominio de las ontologías existen algunas propuestas para la gestión de las versiones de una ontología [9, 10, 11]. Estas propuestas se enfocan en el versionado de ontologías en cuanto a la evolución en su estructura, sin considerar las versiones y la representación del cambio en la información que ésta representa. Dentro del área de manufactura, Surdasan y colab. [12] proponen representar la evolución de familias de productos a través de 3 submodelos diferentes para representar la familia, la evolución y los fundamentos de la misma, respectivamente.

A pesar de que existen propuestas para el manejo de las variabilidades en el espacio y en el tiempo, no se ha encontrado ninguna que aborde simultáneamente estas dos problemáticas. Teniendo en cuenta la necesidad de gestionar ambos tipos de variabilidades conjuntamente, este artículo propone un enfoque basado en ontología para la gestión de versiones y la representación del cambio ocurrido en la información de productos durante su ciclo de vida. El modelo propuesto define conceptos generales que pueden ser especializados para incorporar el manejo de versiones en modelos de productos que ya tengan en cuenta la representación de la variabilidad en el espacio, independientemente del modelo de productos utilizado.

La organización del trabajo es la siguiente: la sección 2 introduce brevemente los preguntas que un modelo de representación de versiones debería ser capaz de contestar. En la sección 3 se describe la propuesta y su aplicación a los modelos de representación de productos PRONTO y FM. En la sección 4 se presenta la implementación de la propuesta en el lenguaje Ontology Web Language (OWL). Finalmente, en la sección 5 se presentan las conclusiones y trabajos futuros.

2 Requerimientos para la representación de los cambios

Un cambio en un lugar puede tener efectos no deseados en otra parte, es decir la consecuencia de un cambio implica un nuevo cambio creando un efecto dominó y la información de productos podría convertirse en un conjunto de datos incompletos o inconsistentes. Para ello se requiere de un proceso eficiente para administrar las versiones, asegurando que todos los cambios se gestionen correctamente en su propagación y que no se produzcan cambios innecesarios afectando la consistencia de los modelos de datos de una familia de producto. Sjober en [13] considera que para obtener soluciones sofisticadas y para predecir las consecuencias de los cambios es necesario tener en cuenta un conjunto de preguntas para contribuir con la administración del cambio con respecto a su captura, representación y fundamentación. Este conjunto de preguntas puede formularse de la siguiente forma:

- ¿Qué componentes fueron afectados?
- ¿Cuándo esto ocurrió?
- ¿Por qué ocurrió el cambio?
- ¿Cómo éste cambió?
- ¿Cómo se registró el cambio?

Una propuesta para la administración de versiones debería considerar estas preguntas para analizar el impacto del cambio, registrarlo correctamente y detectar o prevenir las inconsistencias consecuentes.

A partir de un análisis de las diferentes propuestas para la gestión de cambios y versiones que se encontraron, algunas de las cuales se citan en la sección 1, se ha identificado que ninguna de ellas permite responder a todas las preguntas planteadas arriba. En la siguiente sección, se presenta el modelo de representación de versiones propuesto, el cual permite responder a dichas preguntas.

3 Ontología para la gestión de versiones de producto

En esta sección se describe el modelo de datos de la propuesta de una ontología para la gestión de versiones de información de producto. Luego, se describe la aplicación de ésta en dos modelos de representación de familia de productos. Estos modelos son la ontología de productos PRONTO [4] y el FM.

3.1 Modelo de datos de la ontología para la administración de versiones

Este artículo propone un modelo conceptual que permite capturar y representar los cambios en la información de una familia de producto durante su ciclo de vida. Estos cambios pueden tener lugar a causa de diversos factores: de diseño, tecnológicos, decisiones comerciales, problemas en la provisión de materias primas y componentes

El modelo de datos propuesto, introducido en la Figura 1, representa por medio de la entidad *ProductConcept* a la familia de productos cuyas versiones se van a gestionar. Este concepto, deberá ser especializado dependiendo del modelo utilizado

para representar la variabilidad en el espacio, esta especialización se explica en la sección 3.1 y 3.2.

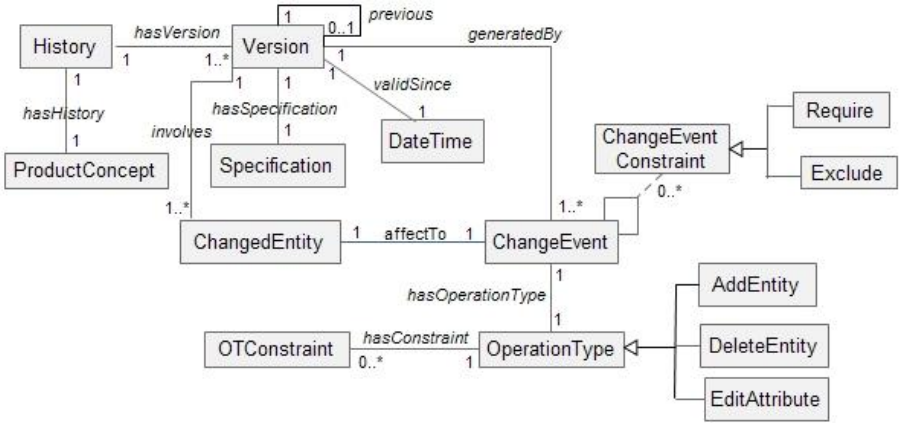

Fig. 1. Modelo Conceptual para la gestión de versiones de productos

La ocurrencia de al menos un cambio (*ChangeEvent)* genera una nueva versión (*Version*) de la información de producto, afectando a un elemento de cambio (*ChangedEntity*) por medio de una relación denotada como *AffectTo*. Para capturar un conjunto de versiones asociados a *ProductConcept*, se introduce la entidad *History*, lo cual permitirá representar todas las versiones que surgen durante el ciclo de vida de una familia de productos. Cada *Version* tiene asociado un tiempo y una fecha (*DateTime*) para indicar el instante en que la versión es válida (ver la relación *validSince* en la Figura 1). Ese instante *DateTime* indica además, el final de una versión anterior. Excepto la versión inicial, cada versión se asociada con su versión predecesora por medio de la relación *previous.*

La entidad *Specification* permite capturar la información acerca de la versión generada, tal como las causas del cambio ocurrido y un identificador de la persona responsable del registro de la versión. Cada versión tiene asociado a través de la relación *involves*, el conjunto de elementos involucrados en el cambio representados por medio de la entidad *ChangedEntity*.

Un *ChangeEvent* es relacionado a un tipo de operación (*OperationType*) por medio de la relación *hasOperationType*. Cada evento de cambio (*ChangeEvent*) puede afectar a una entidad de cambio (*ChangedEntity)*, de tres formas generales y diferentes: agregar una entidad (*AddEntity*), eliminar una entidad (*DeleteEntity*) o modificar atributos de una entidad (*EditAttribute*).

Dependiendo del dominio de aplicación, no todos los elementos pueden ser afectados por todos los tipos de operaciones. Por este motivo, el modelo define la entidad *OTConstraint* para restringir que un tipo de operación *AddEntity*, *DeleteEntity* o *EditAttribute* pueda afectar o no a un *ChangedEntity* específico. Por otro lado, la propuesta define la entidad *ChangeEventConstraint* para describir una relación de dependencia entre dos *ChangeEvent.* Un evento de cambio específico puede requerir (*Require*) o excluir (*Exclude*) la consideración de otro evento de cambio.

ChangeEventConstraint es muy diferente a la entidad *OTConstraint* ya que la primera es aplicada a la ocurrencia de un cambio y la segunda restringe la aplicación de un tipo de operación a una entidad específica. La definición de este conjunto de restricciones es necesaria para mantener la consistencia del modelo conceptual y asegurar la correcta interpretación de los mismos.

En base a lo expuesto, podemos realizar un análisis de la propuesta con respecto al conjunto de preguntas expuestas por Sjober en [13]. Para responder la primera de ellas: "¿Qué componentes fueron afectados?" se definió la entidad *ChangedEntity.* Del mismo modo, para responder "¿Cómo éste cambió? se definieron la entidad *ChangeEvent,* para identificar qué cambio afectó al elemento *ChangedEntity,* y la entidad *OperationType* para especificar el tipo de operación aplicado al evento de cambio. Las entidades *DateTime* y *Specification* permiten responder a las preguntas: "¿Cuándo esto ocurrió?" y "¿Por qué ocurrió el cambio?", respectivamente. Finalmente, para responder "¿Cómo se registró el cambio?", la ontología posee una entidad *History* para capturar todas las versiones de *ProductConcept* durante su ciclo de vida, y una entidad *Version* para registrar toda la información acerca del cambio ocurrido.

3.2 Aplicación de la propuesta al modelo PRoduct ONTOlogy

PRONTO permite la representación de datos de productos en diferentes niveles de abstracción y en diferentes dominios de la industria [4]. Seleccionamos esta ontología por su característica de representación de variantes en la información de producto de forma consistente. Para ello, define una jerarquía estructural (SH- Structural Hierarchy) para representar de forma eficiente la información concerniente a los productos y componentes que participan en la manufactura de productos (Ver Figura 3) y una jerarquía de abstracción (AH- Abstraction Hierarchy) que permite la representación de información no estructural de productos en diferentes niveles de abstracción y la representación de procesos de agregación y desagregación de información entre estos niveles.

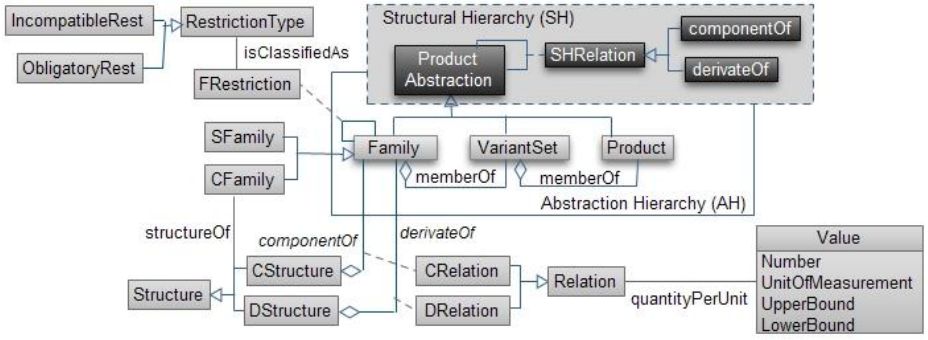

Fig. 3. Principales Conceptos de PRONTO

La AH consta de 3 niveles: nivel de Familia (*Family*), nivel de Conjunto de Variantes (*VariantSet*) y nivel de Producto (*Product*). Estos tres niveles se relacionan entre sí mediante una asociación *memberOf*, indicando que las entidades comprendidas un nivel inferior, son miembros de una instancia de abstracción de un nivel superior.

La SH considera dos tipos de relaciones de estructuras, que se especializan como *componentOf*, para aquellas estructuras que relacionan al producto con sus partes componentes, y *derivateOf*, para aquellas estructuras que enlazan al producto con sus derivados constituyentes. Esto quiere decir que una entidad puede tener otras entidades como componentes o derivados siempre en el mismo nivel de abstracción, lo cual permite representar las diferentes BOM de productos que son manufacturados por medio del ensamblado de partes componentes o la desagregación de materias primas no atómicas. La estructura de una familia (*Structure*) es especializado en familia simple (*SFamily*) para indicar aquellas familias que no tienen estructuras, y en familia compuesta (*CFamily*) para denotar las familias que tienen una o más estructuras. A su vez, PRONTO considera dos tipos de relaciones: relaciones compuestas (*CRelation*) para identificar los componentes de la estructura de composición (*CStructure*) y relaciones de descomposición (*DRelation*) para identificar los derivados de la estructura de descomposición (*DStructure*). Cada una de esas relaciones se relaciona con un concepto *Value*, cual representa la cantidad de partes requeridas para producir una unidad (*quantityPerUnit*). Además, PRONTO introduce un conjunto de restricciones asociadas a la validación de productos, para asegurar la consistencia con los requerimientos de los clientes.

La aplicación de la propuesta al modelo de PRONTO es introducida en la Figura 4. Para poder representar la gestión versiones en los tres niveles de la AH, se especializa la entidad *ProductConcept* en *Family*, *VariantSet* y *Product*. De esta forma es posible capturar la historia, en términos de versiones, en los tres niveles propuestos por PRONTO. En cada uno de los niveles de la AH los cambios afectan a elementos distintos. Así, por ejemplo, en el nivel más alto las modificaciones están dadas en la estructura de los productos que forman la familia. En el nivel intermedio pueden ser cambiados la selección de componentes, así como las restricciones entre conjuntos de variantes, y en el nivel más bajo puede verse afectado la especificación de los productos concretos. Para representar las diferentes entidades que pueden ser cambiadas, así como el evento de cambio correspondiente, se especializa los conceptos *ChangedEntity* y *ChangeEvent* del modelo propuesto. Dada las limitaciones de espacio en la Figura 4 se muestra solamente las entidades de cambio y los eventos de cambio definidos para el nivel de Familia. De esta forma, la entidad *ChangedEntity* se especializa en *FRestricion*, *CRelation* y *Value*, que son afectadas respectivamente por *FRestrictionCE*, *CRelationCE* y *ValueCE*, siendo éstas últimas instancias de *ChangeEvent.* Cada evento de cambio tiene asociado un tipo de operación. Para comprender la aplicación de la propuesta se analiza la misma en el nivel de instancia (Instances Model en la Figura 4), considerando como base un ejemplo muy sencillo consistente en un Auto con dos opciones de transmisión, Manual y Automática.

En el nivel de Familia, en PRONTO, el concepto de la familia de productos Auto se representa mediante la entidad *Car* (Figura 4) que a su vez posee un historial de

versiones *CarHistory*. Este historial se compone de numerosas versiones tales como *V1, V2, V3…Vn.* La versión *V2* involucra dos elementos afectados: una relación de composición de transmisión manual y otra automática, y son representadas como *TransmisionManualCRelation* y *TransmisionAutomaticCRelation,* respectivamente. Esta versión es válida desde 1*2-03-2014* y posee su especificación por medio de *SpecificationV2*. Además, la versión es generada por dos eventos de cambio definidos como *TManualCE* y *TAutomaticCE* y afectan a *TransmisionManualCRelation* y *TransmisionAutomaticCRelation*, *respectivamente*, por medio de dos tipo de operaciones: *DeleteTManual* y *AddTAutomatic.* Finalmente, el evento de cambio *TManualCE* tiene asociado una restricción (*CECRequireTAutomatic)* para indicar que al eliminar la transmisión manual es necesario un evento de cambio para agregar una transmisión automática. Esta restricción controla la propagación del cambio y contribuye con la consistencia de la información de la familia Auto.

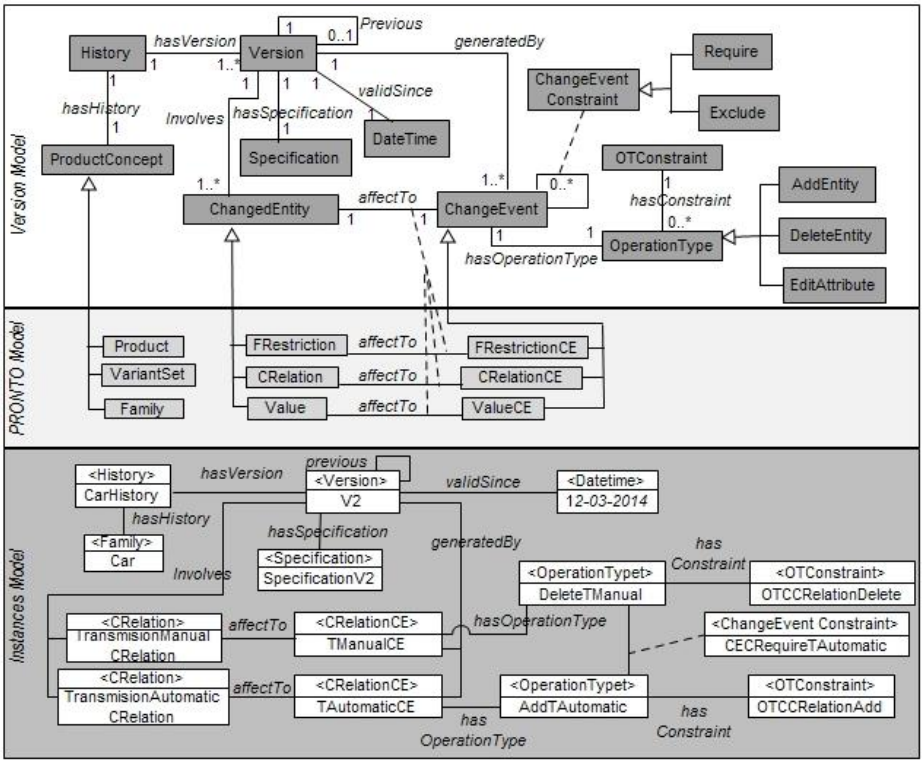

Fig. 4. Aplicación de la propuesta en PRONTO

3.3 Aplicación de la propuesta en el Modelo de Características (FM)

El modelo de características es un método eficiente y efectivo para identificar y organizar los elementos variables y elementos comunes entre productos o líneas de producto [14]. Una característica es un aspecto prominente o distintivo de un sistema,

que es visible a todos los stakeholders. Kang y colab. en [15] sostienen que las características son fundamentales para la comprensión entre clientes y desarrolladores, quienes en su comunicación se expresan naturalmente e intuitivamente en términos de "características de un producto" que deben llevar a cabo. En [14], Kang y colab. proveen un framework para identificar las características de un sistema software bajo una categorización de las mismas en: Características de Capacidades, Características del entorno del sistemas, Características de la tecnología del dominio y Características de la tecnología de implementación.

El FM posee una relación estructural denominada "consiste de (*consist Of*)", para representar una estructura lógica de características. Esta estructura relaciona características padres con características hijas o subcaracterísticas en una estructura de árbol. Los tipos de relaciones de un FM pueden categorizarse como sigue:

- *Obligatoria*: para indicar que una característica hija es requerida.
- *Opcional*: para indicar que una características puede ser requerida o no.
- *Selección* (OR): indica que al menos una característica hija debe ser seleccionada.
- *Alternativa* (XOR): indica que una de las características hija debe seleccionarse.

El FM posee dos tipos de restricciones: una restricción para indicar que una característica requiere la selección de otra y una restricción de exclusión para indicar que dos características no pueden ser parte de una misma característica padre.

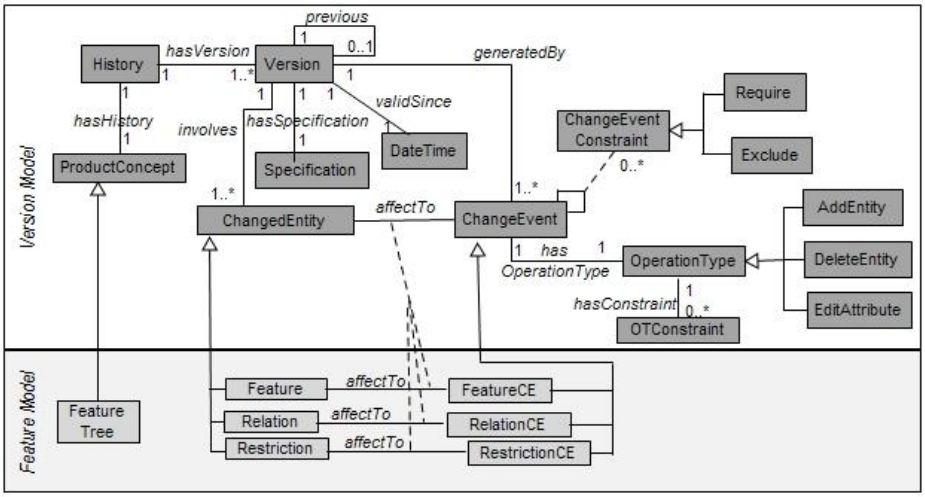

Fig. 5. Aplicación de la propuesta en FM.

En la Figura 5 se representa la aplicación de la propuesta al modelo de características. En esta aplicación se especializa *ProductConcept* en la entidad *FeatureTree*, el cual representa un árbol de características de una familia de productos. En este caso las entidades afectadas por los cambios son las características (*Feature*), las relaciones (*Relation*) y las restricciones (*Restriction*), por lo cual se propone las clases Feature,

Relation y Restriction como especializaciones de *ChangedEntity*. Estos últimos, se representan mediante la especialización del concepto *ChangeEvent* en las clases *FeatureCE, RelationCE y RestrictionCE* que afectan respectivamente a las entidades mencionadas.

4 Implementación en OWL

Mediante la utilización de la herramienta Protégé 3.5 y un razonador Pellet 1.5.2 se implementa el modelo conceptual (Figura 1) de la sección 3.1 en OWL (Ontology Web Languaje). Inicialmente se definió un namespace con el prefijo *asc*, para contener todos los identificadores únicos de los elementos de la ontología. Una vez definido el espacio de nombres, se identificaron todos los conceptos y se clasificaron para organizarlos en una estructura jerárquica. Es decir, la clase *OperationType* tiene como subclases a *AddEntity, Delete Entity* y *EditAttribute*. Del mismo modo para la clase *ChangedEventConstraint.* Siguiendo esta clasificación, se definieron los tipos de datos, las propiedades, el dominio y el rango. Cada propiedad tiene un dominio, el cual consiste de una lista de clases de individuos que pueden ser ubicados a la izquierda de la propiedad, y un rango, el cual consiste de una lista de clases de individuos que se ubican a la derecha de la propiedad. El dominio y rango son usados por el razonador para inferir nuevo conocimiento.

Luego se especificaron las expresiones de cada clase y las condiciones de validez o afirmaciones. Estas condiciones aseguran la consistencia del modelo conceptual, por ejemplo: una afirmación indica que un "*ChangeEvent* debe estar asociado al menos con un *OperationType*" o "una clase *ProductConcept* debe estar asociado a una clase *History*". Estas afirmaciones son definidas como condiciones necesarias.

Table 1. Conjunto Resumido de Reglas SWRL

	Reglas	**Definición**
1	ProductConcept(?*x*) ∧ History(?*y*) ∧ hasHistory(?*x*, ?*y*) → isHistoryOf(?*y*, ?*x*)	*x* es un ProductConcept asociado a una history *y* e *y* es history de ProductConcept *x*.
2	History(?*x*) ∧ Version(?*y*) ∧ hasVersion(?*x* ?*y*) → isVersionOf(?*y*, ?*x*)	*x* es una history e *y* es una versión. *x* tiene una versión *y*. *y* es versión de *x*.
3	EventChange (?*x*) ∧ EditAttribue(?*y*) ∧ hasOperation(?*x*,?*y*) ∧ ChangedEntityEdited(?*t*,?z) → ChangedEntity(?z)	Un evento de cambio *x* que tiene un tipo de operación EditAttribute *y*. Afecta a un ChangedEntity z.
4	ProductConcept(?f) ∧ History(?h) ∧ Version(?v) ∧ EventChange(?ch) ∧ GeneratedBy(?v,?ch) ∧ Add(?o) ∧ hasTypeOperation (?ch,?o) ∧ ChangedEntity (?e) → addEntityToProductConcept(?e,?f)	Esta regla infiere que una entidad de cambio *e* es agregada al ProductConcept *f* en una versión *v*.

Con el fin de definir el comportamiento y la semántica de las relaciones, se utilizó el lenguaje SWRL (Semantic Web Rule Language) para definir un conjunto de reglas que permitan inferir nuevo conocimiento por medio de la deducción. En la Tabla 1 se representa un conjunto de reglas limitado para inferir conocimiento, por ejemplo para inferir las relaciones inversas se definieron las reglas 1 y 2. La regla 3 permite inferir un elemento que fue modificado por la ocurrencia de un cambio y la regla 4 permite inferir que un elemento es incorporado a la información que represente un *ProductConcept*.

Una regla SWRL tiene dos partes, el antecedente y el consecuente. En este sentido, si todos los conceptos atómicos en el antecedente de una regla son verdaderos, entonces la consecuencia debe ser verdadera también.

Table 2. Aplicación de SPARQL para responder el conjunto de preguntas

Pregunta	*Consulta SPARQL*	*Resultados*
¿Qué componentes fueron afectados?	SELECT ?V2 ?Components WHERE { ?V2 asc:involves ?Components }	asc:V2 - asc:TransmisionManualCRelation asc:V2 – asc:TransmisionAutomatic-CRelation
¿Cuándo esto ocurrió?	SELECT ?V2 ?date WHERE { ?V2 asc:ValidSince ?date }	asc:V2 – asc:12_03_14
¿Por qué ocurrió el cambio?	SELECT ?Specification ?Description ?V WHERE { ?Specification asc:Description ?Description . ?Specification asc:isSpecificationOf ?V . ?Specification asc:isSpecificationOf asc:V2}	asc:SpecificationV2 – asc:TransmisionManualCRelation is deleted – asc:V2 asc:SpecificationV2 – asc:TransmisionAutomaticCRelation is added – asc:V2
¿Cómo éste cambió?	SELECT ?V ?CEvent ?OpType ?EntityAffected ?Date WHERE { ?V asc:GeneratedBy ?CEvent . ?V asc:Involves ?EntityAffected . ?CEvent asc:hasOperationType ?OpType . ?V asc:ValidSince ?Date FILTER (?V =asc:V2)}	asc:V2 – asc: TManualCE – asc:DeleteTManual – asc:TransmisionManualCRelation – asc:12_03_14 asc:V2 – asc: TAutomaticCE – asc:AddTAutomatic – asc:TransmisionAutomaticCRelation – asc: 12_03_14
¿Cómo se registró el cambio?	SELECT ?ProductConcept ?History ?V ?Date WHERE{ ?ProductConcept asc:hasHistory ?History . ?History asc:hasVersion ?V . ?V asc:ValidSince ?Date }ORDER BY ASC (?V)	asc:Car – asc:CarHistory – asc:V1 – asc:DateTimeV1 asc:Car – asc:CarHistory – asc:V2 – asc:12_03_14 asc:Car – asc:CarHistory – asc:V2 – asc:12_03_14

Una vez que la información acerca del cambio es capturada y formalizada, es posible formular consultas para obtener y manipular datos almacenados en un formato de tripleta por medio de la utilización de un lenguaje de consultas SPARQL (Protocol and Query Language) [16]. Este lenguaje permitió responder el conjunto de preguntas mencionado en secciones anteriores.

En la Tabla 2 podemos ver la aplicación de estas consultas al modelo de instancias explicado en la sección 3.2. Este conjunto de preguntas provee un conocimiento apropiado para una gestión de cambio de información de una familia de producto durante su ciclo de vida. Estas simples demostraciones registran los cambios ocurridos en las versiones analizadas, sin embargo pueden ser extendidas a otros productos, versiones o cambios.

5 Conclusiones y Trabajos Futuros

Durante el desarrollo del presente trabajo se analizaron diferentes contribuciones para la gestión de la variabilidad de los productos. Este análisis permitió identificar que muchas contribuciones gestionan la variabilidad en el espacio, otro grupo de trabajos se enfocan en la variabilidad en el tiempo. Sin embargo, no se han encontrado propuestas que gestionen ambos tipos de variabilidad.

Este trabajo presenta una propuesta que permite representar la gestión de versiones que surgen debido a la variabilidad en el tiempo de los productos. Esta propuesta puede ser aplicada en diferentes dominios, en modelos de representación de productos que ya manejen la variabilidad en el espacio. Como ejemplo de esta capacidad, el trabajo presenta la aplicación de la propuesta a dos modelos de representación de familia de productos: PRONTO y el modelo de características

Con el fin de verificar que la propuesta responde a las preguntas planteadas como requerimiento al inicio del trabajo, se implementó el modelo conceptual propuesto utilizando el lenguaje OWL. Esta implementación permitió la definición de un conjunto de reglas en lenguaje SPARQL para obtener las respuestas a las preguntas mencionadas. Este trabajo muestra la validación de la propuesta mediante la respuesta a preguntas de competencia en un caso concreto. Este tipo de validación puede ser complementado con métodos de evaluación basados en métricas para el análisis estructural y estimación de complejidad. Sin embargo, los resultados de estas métricas no son objetivos, es decir si se somete una ontología a un análisis estructural y los resultados demuestran que ésta posee una jerarquía de gran profundidad, este resultado no necesariamente implica que se trata de una ontología de mayor o menor calidad.

Como trabajos futuros, se pretende extender la propuesta para gestionar las versiones teniendo en cuenta la propagación de los cambios en diferentes niveles de la estructura de una familia de productos y aplicarlo en otros casos de estudios para complementar la validación de esta propuesta. Posteriormente se desea profundizar el análisis para determinar nuevas restricciones que permitan asegurar la consistencia de los modelos.

6 Agradecimientos

Este trabajo ha sido financiado en forma conjunta por CONICET, la ANPCyT (PICT 2315), la UTN (PID 25-O156) y la Universidad Nacional de La Rioja. Se agradece el apoyo brindado por estas instituciones.

Referencias

1. Matsokis A. Kiritsis D. An ontology- based approach for products lifecycle management. Computer in Industrial 61 787-797. (2010).
2. Asikainen T., Mannisto T., Soininen T. Kumbang: A domain ontology for modeling variability in software product families. Advances engineering informatics. (2006)
3. Pohl K., Bockle G., Van Der Linden F.: Software Product Line Engineering. Foundations, principles, and Techniques. ISBN-10 3-540-24372-0 Springer Berlin Heidelberg New York. (2006)
4. Vegetti, M., Leone, H., Henning, G.: PRONTO: An ontology for comprehensive and consistent representation of product information. Engineering Applications of Artificial Intelligence 24 (8), pp. 1305-1327. (2011)
5. Scheer, A.W.: Business Process Engineering. *Springer-Verlag*, Berlin- Heidelberg. (1998)
6. Van Veen, E.A. and Wortmann, J.C.: New developments in generative BOM processing systems. Production Planning & Control, 3, 327-335. (1992)
7. Olsen, K.A., Sætre, P. Thorstenson, A:. A Procedure- Oriented Generic Bill of Materials. Computers Ind. Engng., 32, 29-45. (1997)
8. Hegge, H.: Intelligent Product Descriptions for bussiness applications. (1995)
9. Shaban-Nejad A., Haarslev V.: Bio-medical Ontologies Maintenance and Change Management. Departament of Computer Science and Software Engineering. (2009)
10. Kirsten T., Hartung M., Grob A., Rahm E.: Efficient Management of Biomedical Ontology Version. OTM 2009 Workshop, LNCS 5872. (2009)
11. Völkel M., Groza T.: SemVersion: RDF-based Ontology versioning system. In: Proc of the IADIS Intl. Conference WWW/internet ICWI. (2006)
12. Sudarsan R., Fenves S.J., Sriram R.D., Wang F.: A product information modeling framework for product lifecycle management. Computer-aided Design 37 1399-1411 (2005)
13. Sjober D.: Managing Change in Information Systems: Technological Challenges. Department of Informatics, University of Oslo. N-0316 Oslo, Norway. (1995)
14. Kang K.C. Lee K., Lee J.: Feature Oriented Product line Software Engineering: Principles and guidelines. Domain-Oriented Systems Development: Practices and Perspectives. Cap 2. ISBN 0203711874. (2003)
15. Kang K., Lee J., Donohoe P.: Feature-Oriented Product Line Engineering. IEEE software. (2002)
16. SPARQL Query Language for RDF. SPARQL Tutorial created by W3C SPARQL Working Group. http://www.w3.org/TR/rdf-sparql-query.