Una implementación de PRONTO en OWL y SWRL

Padula, Nicolas
INGAR – Instituto de Desarrollo y Diseño (CONICET-UTN)
Universidad Tecnológica Nacional – Facultad Regional Santa Fe
nicolasvpadula@gmail.com

Abstract
La creciente demanda de variedad por parte del mercado obliga a las empresas a replantear sus esquemas productivos para mantener su competitividad. Un diseño y modelado adecuado de familias de productos en función de su estructura, en conjunto con la integración semántica de los sistemas que gestionan información de estructura de productos pueden proporcionar grandes beneficios a las organizaciones productivas. Este trabajo plantea una implementación de PRONTO (PRoduct ONTOlogy), un modelo que permite la representación de familias de producto con estructuras complejas, utilizando OWL y SWRL.

Palabras Clave
Familias de productos, BOM Genérico, Ontologías, Web Semántica, PRoduct ONTOlogy

Introducción

La necesidad de satisfacer una demanda creciente por productos cada vez más específicos por parte del mercado ha llevado a las organizaciones a reevaluar sus estrategias de negocio con el fin de mantener su competitividad. Las organizaciones apuntan a lograr una diversificación de su oferta que satisfaga las necesidades de los distintos segmentos del mercado sin perder eficiencia en la producción [1,2].

Sin embargo, ambos objetivos entran en conflicto entre sí. Una mayor diversidad de productos conlleva de forma inherente un aumento en la complejidad de producción, y por lo tanto, un aumento del costo. A su vez, una mayor estandarización de los productos resultaría en una incapacidad de satisfacer las demandas más específicas. Una solución eficaz a este conflicto es el diseño de familias de productos (Product Family Design) [3,4]. Una familia de productos se define como un conjunto de productos que comparten características y componentes en común, con variaciones únicas entre sí, para satisfacer necesidades distintas [5]. El conjunto común es conocido como la plataforma del producto [6]. La plataforma es el núcleo común de las familias de producto, son conjuntos de componentes, tecnologías, subsistemas, y procesos que forman una estructura común de la cual se pueden derivar conjuntos de productos [7].

Una plataforma correctamente diseñada permite la manufactura eficiente de los componentes comunes de una familia [8], aumenta la flexibilidad y reduce la complejidad y puede incrementar la reusabilidad y adaptabilidad del esquema productivo [6]. El diseño de plataformas se enfoca en maximizar las partes comunes (*commonality*), mediante la identificación de los aspectos a ser compartidos entre los productos de una familia, y las distintas configuraciones de estos aspectos entre los individuos.

Un diseño adecuado de una familia de productos debe reconciliar aspectos opuestos: Los aspectos externos, que provienen de las necesidades de los clientes y el mercado, y generalmente apuntan a una mayor variedad de funciones y productos; y por otro lado los aspectos internos a la organización, referidos a la factibilidad técnica y económica de la producción, la complejidad logística y de manufactura, y la gestión misma de la organización, todos aspectos que se ven beneficiados ante la estandarización e integración de procesos y componentes.

Debido a la necesidad de encontrar un equilibrio óptimo entre partes comunes y diversidad, y a la cantidad de factores y variables a considerar, el diseño de familias de producto es un problema esencialmente complejo.

Como se expuso anteriormente, existe un gran abanico de criterios con los que se pueden definir familias de productos. Una correcta definición de familias de productos en función de la estructura de los mismos es de especial importancia para la optimización de los procesos de manufactura, sobre todo en ambientes industriales que implementen una cadena de suministro integrada, algo usual en el entorno competitivo actual [9]

La información de la estructura de productos suele almacenarse de manera dispersa en sistemas que, al integrarse las cadenas de suministro, resultan ser heterogéneos entre sí. [10]. Para subsanar el problema de interoperabilidad de estos sistemas heterogéneos, se plantea la necesidad de una integración a nivel semántico [11]. Desde hace unos años las ontologías son vistas como una solución a este problema de integración semántica [12]. Los lenguajes utilizados para implementar las ontologías propuestas en este trabajo son OWL (Ontology Web Language) y SWRL (Semantic Web Rule Language), ambos estándares definidos por el World Wide Web Consortium para el ámbito de la Web Semántica.

Existen varios aspectos a considerar a la hora de definir familias de producto, tales como la identificación de variantes [13], la exclusión de variantes inválidas mediante restricciones de obligatoriedad o incompatibilidad [13, 14] y si la conformación de un producto se da en base a la composición o descomposición de otros.

Un enfoque para la gestión de variantes es el de *BOM Genérico* (GBOM) [15]. El mismo consiste en definir estructuras de productos genéricos, con componentes genéricos, los cuales se reemplazarán por productos y componentes concretos al momento de especificar una variante puntual. Cada componente genérico representa el conjunto de productos concretos que pertenece a la familia denotada por el mismo. Dicho componente genérico puede ser reemplazado por cualquier producto concreto, y cada reemplazo equivale a una variante distinta de la familia representada por el producto genérico.

PRONTO (PRoduct ONTOlogy) [16] es un modelo que contempla todas estas características.

La idea central de PRONTO parte del concepto de BOM Genérico, y se basa en dos jerarquías: La AH (*Abstraction Hierarchy* o Jerarquía de Abstracción), la cual está asociada al nivel de especificidad en la definición de un objeto del modelo. Consta de tres niveles, de más abstracto a más específico: Familia, Conjunto de Variantes y Producto. El nivel Familia representa un conjunto de productos que tienen algo en común. El nivel Conjunto de Variantes representa un subconjunto de una familia dada, que responde a una determinada modificación o selección dentro de dicha familia. El nivel Producto representa un único producto físico, perteneciente a un conjunto de variantes y una familia en particular.

En paralelo a ésta jerarquía, existe otra llamada SH (*Structural Hierarchy* o Jerarquía Estructural). La SH define relaciones estructurales de composición (o descomposición, según sea el caso) que en un objeto requiere para ser manufacturado.

Todos los niveles de la AH participan de relaciones de la SH, y la especificación de objetos en los distintos niveles de la AH se hace a través de modificaciones o restricciones de las relaciones de la SH.

En otras palabras: para definir una variante concreta (nivel Producto) se aplican modificaciones a su estructura genérica (niveles Familia y Conjunto de Variantes).

Actualmente, las empresas suelen almacenar la información de sus productos tratando a cada variante como un producto distinto. Adoptar un modelo de familias no solo implicaría beneficios técnicos (menor duplicación de datos y consecuentemente una menor dificultad de mantenimiento), sino que también resultaría en la obtención de conocimiento de valor estratégico para la toma de decisiones. Debido a esto, resulta

de interés poder inferir familias de productos a partir de un conjunto de productos existente. OWL y SWRL, en conjunto con los sistemas razonadores, proveen facilidades para la inferencia de nuevo conocimiento.
Este trabajo propone una implementación utilizando OWL y SWRL. El desarrollo de los distintos elementos del modelo se hará en la siguiente sección, conforme se presenten las nuevas implementaciones de los mismos.
Las actividades que han dado origen a este trabajo se enmarcan en un programa de becas de iniciación a la investigación, dentro de un proyecto homologado por el programa de incentivos.

Elementos del Trabajo y metodología

El primer elemento a definir es el concepto de Familia. Una familia es el nivel más abstracto de la AH, y representa un conjunto de productos con características en común. Cada familia tendrá al menos un conjunto de variantes (*VariantSet*) asociado, estos conjuntos de variantes representan subconjuntos de productos que pertenecen a dicha familia, ésta asociación se hace a través de la relación *hasMembers*. Las familias se clasifican en familias simples (*SFamily*), que representan conjuntos de productos primarios, o compuestas (*CFamily*), que representan conjuntos de productos que se obtienen a través de la manufactura de productos primarios. En la Figura 1 puede observarse la definición de la clase *Family*, que refleja lo expresado previamente.

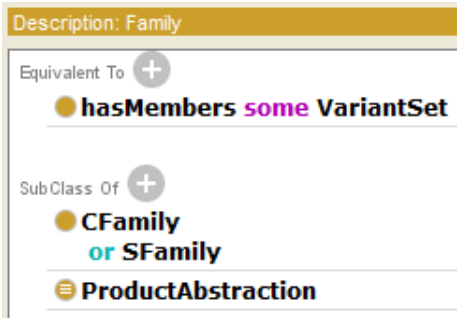

Figura 1. Definición de la clase *Family*.

Análogamente a las familias, los conjuntos de variantes pueden ser simples (*SVariantSet*) o compuestos (*CVariantSet*), dependiendo si representan productos primarios o manufacturados respectivamente. Considerando esto, debe especificarse la restricción de que un conjunto de variantes compuesto solo pueda estar asociado a una familia compuesta, y la restricción equivalente para conjuntos de variantes y familias simples.
Las familias compuestas estarán asociadas a al menos una estructura (*Structure*), la cual especifica un conjunto de relaciones entre la familia padre (manufacturada) y aquellas que son necesarias para la producción de la misma. La Figura 2 muestra axiomas para la definición de familias compuestas.

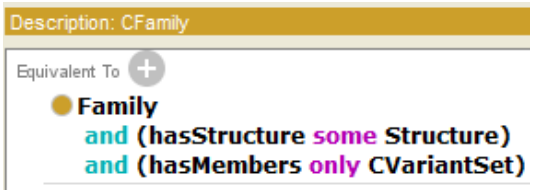

Figura 2. Definición de la clase *CFamily*.

Las estructuras pueden ser de composición (CStructure) o descomposición (*DStructure*) según representen relaciones de ensamblaje o desagregación para obtener el producto manufacturado. En concordancia con esto, una estructura de composición tendrá un conjunto de relaciones de composición (*CRelation*) mientras que una estructura de desagregación estará asociada con relaciones de descomposición (*DRelation*). PRONTO contempla la posibilidad de que una familia tenga más de una estructura (que pueden representar distintas rutas o metodologías de producción), para reflejar esto se hace uso de la relación *alternativeStructure*, que relaciona una estructura con su alternativa. El valor de ésta relación se puede inferir a través de la regla SWRL que se presenta en la Figura 3. En la Figura 4 se ven los axiomas para la definición de la clase D*Structure*. La relación derivatives permite asociarla a una

instancia de *DRelation*, la cual contendrá la información correspondiente a esa relación. Una definición análoga se realiza para *CStructure* y *CRelation*.

```
Family(?F), Structure(?S1), Structure(?S2),
structureOf(?S1, ?F), structureOf(?S2, ?F), DifferentFrom
(?S1, ?S2) -> alternativeStructure(?S1, ?S2)
```

Figura 3. Regla para la inferencia de la relación *alternativeStructure*.

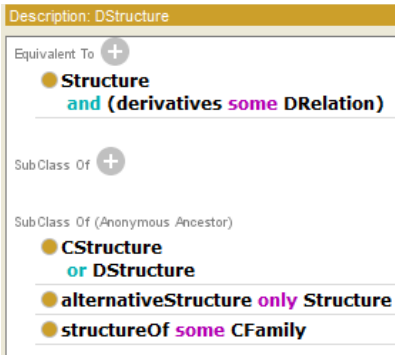

Figura 4. Definición de la clase *DStructure*.

Cada instancia de relación, contendrá la información sobre:

- Cantidad necesaria de un determinado producto para manufacturar una unidad, si es una relación de composición. O cantidad de unidades que se obtienen de la descomposición de un producto, si es una relación de desagregación.
- Proporción que un componente dado representa respecto a la unidad del producto que compone, si es una relación de composición. O rendimiento que el producto representa en la descomposición de una materia prima.
- Restricciones de máximo y mínimo para los datos previamente mencionados.
- Tipo de relación.

Los primeros tres ítems son almacenados mediante la clase *RestrictedValue*, junto con la unidad de medida correspondiente. Esto, junto con el tipo de relación (*RelationType*) es definido en la clase general *Relation*, como se ve en la Figura 5.

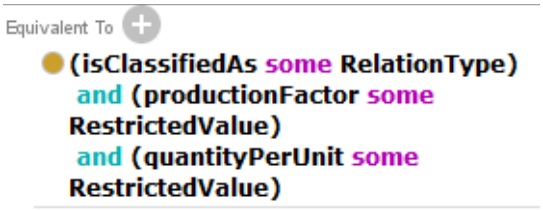

Figura 5. Definición de la clase *Relation*.

Las relaciones pueden clasificarse en tres tipos distintos:

- Selectivas (*SelectiveRel*). El componente o derivado se selecciona a partir de un conjunto, pero sólo un elemento del conjunto puede estar presente en la estructura.
- Opcionales (*OptionalRel*). El componente o derivado puede o no estar presente en la estructura.
- Obligatorias (*MandatoryRel*). El componente o derivado debe estar presente en la estructura.

En las especializaciones de relación se expresa además, con qué familia componente o derivado, según sea *CRelation* o *DRelation*, se está haciendo la asociación. En la Figura 6 se observa lo expresado para el caso de *DRelation*.

Figura 6. Definición de la clase *DRelation*.

Hasta este punto se han definido las relaciones estructurales básicas del nivel Familia de la AH. En base a esto pueden definirse reglas que permitan inferir conocimiento relevante sobre éstas familias. La relación *isComposedBy* para *CStructure* o *isDerivedInto* para *DStructure* representan las relaciones de la SH entre familias. Estas relaciones son definidas como transitivas para, mediante el uso de un razonador, inferir los árboles multinivel necesarios para manufacturar un producto dado. Esta relación se instancia mediante la regla presentada en la Figura 7, para el caso de relaciones de composición.

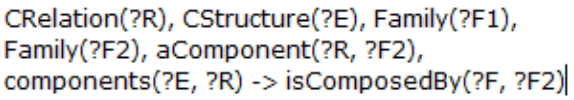

Figura 7. Regla para la inferencia de *isComposedBy*.

Esta regla puede interpretarse de la siguiente manera: si existe una relación en una estructura que asocie a F2 como componente de F, entonces F está compuesta por F2. Una regla análoga fue definida para las relaciones de desagregación. También resulta útil poder inferir relaciones estructurales independientemente de si son por medio de estructuras de composición o descomposición. Para esto se definió la relación *isManufacturedWith*, a partir de las reglas presentes en la Figura 8. Esta relación también se definió como transitiva para poder inferir las estructuras en todos sus niveles.

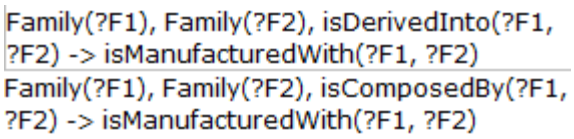

Figura 8. Reglas para la inferencia de *isManufacturedWith*.

El paso siguiente es formalizar los conceptos relacionados con el nivel de abstracción Conjunto de Variantes.
Como se mencionó antes, un *SVariantSet* solo podrá estar relacionado con una *SFamily* y la misma restricción aplicará para *CVariantSet* y *CFamily*. Además, cada *VariantSet* solo podrá ser miembro de una única familia, esto se formaliza mediante la definición de la propiedad *memberOf* como funcional.
En el caso de los conjuntos de variantes compuestos, los mismos estarán necesariamente asociados a una estructura dada, de la cual derivan. Esta estructura también es única para cada conjunto de variantes. Esta asociación se lleva a cabo mediante la relación *has*.
Un conjunto de variantes compuestas puede o no modificar la estructura de la cual deriva, de hacerlo, se debe especificar qué cambios se aplican a la estructura de la familia para arribar a la estructura de un conjunto de variantes en particular. Estas modificaciones se especifican a través de la clase *ChangeSet*. Cada instancia *ChangeSet* permite especificar que estructura se está modificando (*modifies*) y qué cambio se está aplicando (*appliedChange*).
Los distintos tipos de cambio se agrupan bajo la clase *Change*, la misma referencia que relación en particular de la estructura se está modificando.
Hay tres tipos de cambios que pueden efectuarse sobre una relación:

- Cambios de cantidad de la relación (*QuantityChange*), ya sea la cantidad por unidad (*QuantityPerUnitCh*) o el porcentaje del producto sobre la estructura (*ProductionFactorCh*).
- Selección de una de las relaciones pertenecientes a un conjunto de relaciones de tipo selectiva (*FamilySpecification*).
- Eliminación de un componente o derivado (*FamilyRemoval*).

En los últimos dos casos, debe comprobarse que el cambio a realizar sea consistente con el tipo de relación que se está modificando. Es decir, no se puede eliminar una relación que no es de tipo opcional, ni se puede especificar una relación en particular que no sea de tipo selectiva. Esta restricción de integridad se realiza mediante las reglas presentadas en la Figura 9.

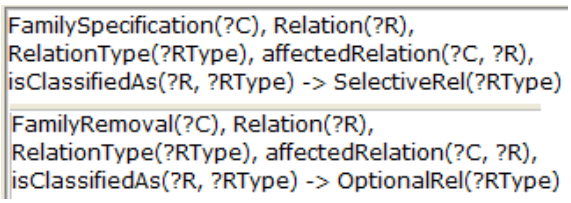

Figura 9. Reglas para verificar la consistencia de los cambios.

Para los cambios relacionados a las cantidades, debe verificarse que las nuevas cantidades estén dentro de los límites permitidos por la estructura. La regla de la Figura 10 permite especificar esta restricción de integridad para los cambios

de tipo *QuantityPerUnitCh*. La definición para cambios *ProductionFactorCh* es análoga.

```
QuantityPerunitCh(?C), Relation(?R), RestrictedValue(?RV),
Value(?V), affectedRelation(?C, ?R), newQuantityPerUnit(?C,
?V), quantityPerUnit(?R, ?RV), LowerBound(?RV, ?N2),
Number(?V, ?N), UpperBound(?RV, ?N3) ->
greaterThanOrEqual(?N, ?N2), lessThanOrEqual(?N, ?N3)
```

Figura 10. Regla para verificar que la nueva cantidad esté dentro de los límites.

Otro concepto de relevancia en este nivel de abstracción es el de preselección (*EnforcedVariantSet*). Un *CVariantSet* puede imponer a través de una preselección, una restricción que indica los conjuntos de variantes particulares que deberán ser usados en la especificación del *CVariantSet* en cuestión. Una instancia de *EnforcedVariantSet* incluirá la información de qué familia se está afectando al hacer la preselección, y que conjunto de variantes en particular se está especificando.
Finalmente, de igual manera que en el nivel Familia, pueden establecerse reglas que permitan inferir los árboles de estructura del nivel Conjunto de Variantes. Para esto se utilizó la relación *isManufacturedWith* definida previamente. Un *CVariantSet* CV será manufacturado a partir de un conjunto CV2 si se da una de dos condiciones: La primera, que exista una preselección (*EnforcedVariantSet*) que así lo indique, es decir, que CV preseleccione directamente a CV2. La segunda, que CV2 sea miembro de alguna familia que forme parte de la estructura de la familia a la cual CV pertenece, y no exista ninguna preselección sobre dicha familia. Estas dos condiciones son reflejadas en las reglas de la Figura 11.

```
CVariantSet(?VS1), EnforcedVariantSet(?EVS),
VariantSet(?VS2), enforcedBy(?VS1, ?EVS),
enforcedSet(?EVS, ?VS2) -> isManufacturedWith(?VS1, ?VS2)

CVariantSet(?VS1), Family(?F1), Family(?F2),
VariantSet(?VS2), (isAffectedBy max 0
EnforcedVariantSet)(?F2), isManufacturedWith(?F1,
?F2), memberOf(?VS1, ?F1), memberOf(?VS2, ?F2)
-> isManufacturedWith(?VS1, ?VS2)
```

Figura 11. Reglas para la inferencia de *isManufacturedWith* entre *VariantSets*.

El último nivel de abstracción a definir es el nivel Producto. Cabe destacar que los integrantes de este nivel son los únicos elementos de PRONTO que tienen existencia física.
De igual modo que en los niveles anteriores, los productos se dividen en simples (*SProduct*) o compuestos (*CProduct*).
Se puede afirmar que un *CProduct* estará completamente definido cuando se hallen identificados todos los productos que intervienen en la estructura de la cual el mismo deriva. Esta identificación se realiza a través de la relación *selects*. A través de esta relación, un *CProduct* selecciona una instancia de la clase *SelectedProduct*, la misma incluye información sobre qué producto se está seleccionando, y dentro de que *VariantSet* se encuentra. La Figura 12 muestra la definición de esta clase.

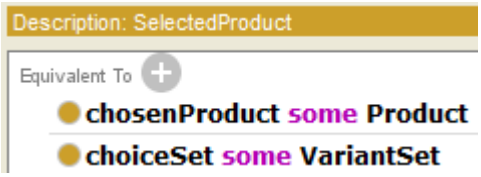

Figura 12. Definición de la clase *SelectedProduct*.

El valor que tome la relación *choiceSet* puede surgir, o bien de una preselección (*EnforcedVariantSet*) existente, o del total de conjuntos de variantes que integran una familia. La regla para inferir este valor se presenta en la Figura 13.

```
CVariantSet(?V1), Product(?P1), Product(?P2),
SelectedProduct(?S), VariantSet(?V2),
VariantSetMembership(?P1, ?V1),
VariantSetMembership(?P2, ?V2),
chosenProduct(?S, ?P2), isManufacturedWith(?V1,
?V2), selects(?P1, ?S) -> choiceSet(?S, ?V2)
```

Figura 13. Regla para la inferencia de *choiceSet*.

El último concepto a definir es el de restricción (*Restriction)*. Las restricciones son transversales a toda la AH, y facilitan la exclusión de estructuras inválidas a lo largo del modelo. Son definidas en los tres niveles: *FRestriction, VSRestriction* y *PRestriction*, para *Family*, *VariantSet* y *Product*, respectivamente. Cada restricción tiene un tipo, el cual puede ser de

obligatoriedad (*MandatoryRest*) o de incompatibilidad (*IncompatibleRest*).
La restricción de incompatibilidad impone la restricción de que dos elementos no pueden estar presente en la misma estructura, mientras que la restricción de obligatoriedad establece que dos elementos deben estar presentes en la misma estructura. La Figura 14 presenta dos reglas que permiten verificar esto para el nivel de conjunto de variantes. Las definiciones para los otros niveles de la AH se realizan de forma análoga.

```
CVariantSet(?V1), IncompatibleRest(?T), VSRestriction(?R),
VariantSet(?V), VariantSet(?V2), hasType(?R, ?T),
imposedRestriction(?V1, ?R), isManufacturedWith(?V1, ?V),
restricts(?R, ?V2) -> DifferentFrom (?V, ?V2)
```

```
CVariantSet(?V1), MandatoryRest(?T), VSRestriction(?R),
VariantSet(?V2), hasType(?R, ?T), imposedRestriction(?V1,
?R), restricts(?R, ?V2) -> isManufacturedWith(?V1, ?V2)
```

Figura 14. Restricciones de incompatibilidad y obligatoriedad.

Resultados

A través de los distintos axiomas, reglas y especificaciones a lo largo del trabajo, se ha arribado a una ontología que representa el modelo conceptual establecido por PRONTO. La Figura 15 presenta un grafo con las principales clases y relaciones definidas en esta ontología.

Discusión

Durante la definición de este modelo se ha hecho uso de las capacidades expresivas que resultan de la combinación de OWL y SWRL respecto de O-Telos, por ejemplo, definir las propiedades que representan composición o dependencia entre dos elementos del modelo como transitivas permite delegar el encadenamiento de los distintos niveles de la SH al razonador. Sin embargo, existen aspectos que pueden ser susceptibles a una reingeniería, para aprovechar mejor las ventajas de las tecnologías semánticas. Un ejemplo de esto pueden ser las definiciones de *Restriction* y *RestrictionType* o *Relation* y *RelationType*. En ambos casos, se define una clase y su tipo por separado, y se asocian a través de una relación. Esto podría implementarse de manera más sencilla si se establecieran los tipos de *Restriction* o *Relation* como subclase de las mismas. Esto disminuiría el número de relaciones y simplificaría la definición de reglas.
Las restricciones de integridad definidas en la implementación original de PRONTO fueron reimplementadas en OWL y SWRL, sin embargo, ninguno de estos lenguajes fue diseñado para la definición de restricciones de integridad. Esto da lugar a expresiones poco intuitivas. Algunas reglas de este estilo son las que hacen referencia a cambios aplicados (Figura 9 y 10) en las cuales se debió definir una condición como consecuente, a causa de la sintaxis de SWRL. Otro ejemplo puede verse en la Figura 14, donde, al no existir la posibilidad de negar una relación, se debió definir esta

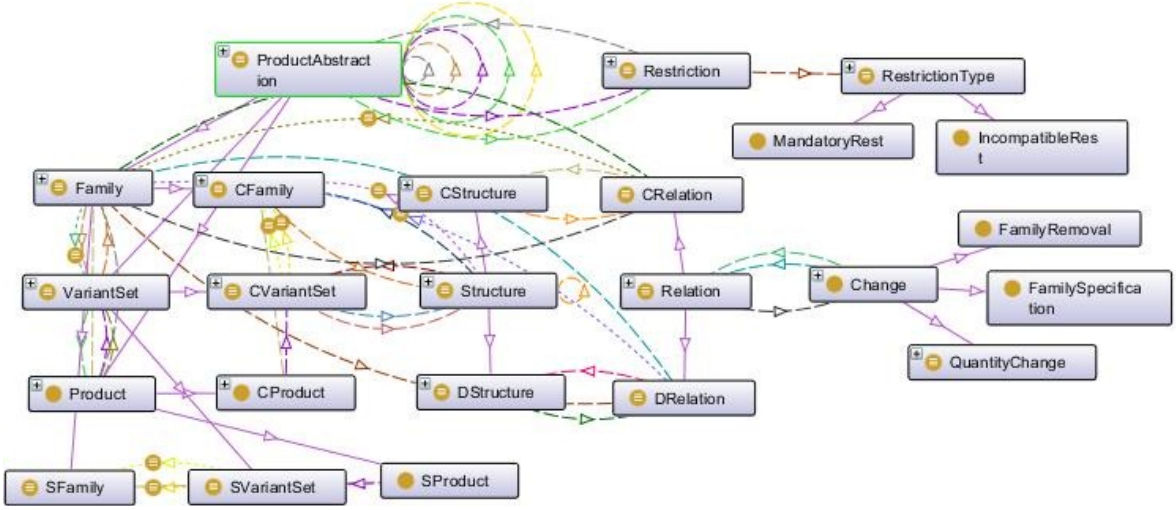

FIgura 15. Vista parcial del modelo obtenido

relación en el antecedente y usar un *DifferentFrom* en el consecuente.
Finalmente, cabe resaltar que el modelo obtenido permite representar familias y estructuras de mayor complejidad, y con una mejor capacidad semántica que en previos intentos de definir familias de productos a través de ontologías [17].

Conclusión

Con este trabajo se ha alcanzado una implementación en OWL y SWRL de PRONTO, obteniendo así un modelo de familias de producto que permite la representación de estructuras complejas.
La continuación de este trabajo consistirá en la reingeniería del modelo conforme a lo establecido en la sección anterior, junto con la posterior evaluación del mismo. Además, resulta de interés la posibilidad de extender el modelo para incluir la funcionalidad de, partiendo de un conjunto de productos concretos, poder inferir sus estructuras genéricas de manera automática. Esto es un aspecto importante puesto que las empresas ya cuentan con su catálogo de productos almacenado en sistemas que gestionan de manera tradicional la información de estructura de productos (sin gestión de variantes). Por lo tanto, la inclusión de esta característica supondría una mayor facilidad para implementar un sistema basado en este modelo en ambientes industriales reales, disminuyendo tiempo y costos de capacitación e implantación.

Agradecimientos
Se agradece el apoyo brindado por la Universidad Tecnológica Nacional.

Referencias

[1] Jiao,J., Simpson, T., Siddique, Z., "Product family design and platform-based product development: a state-of-the-art review", *Journal of Intelligent Manufacturing*, Kluwer Academic Publishers-Plenum Publishers, 2007, pp. 5-29.

[2] Berry, S., Pakes, A., "The Pure Characteristics Demand Model", *International Economic Review*, 2007, pp. 1193-1225.

[3] Marion, T. J., Thevenot, H. J., Simpson, T. W., "A cost-based methodology for evaluating product platform commonality sourcing decisions with two examples", *International Journal of Production Research*, 2008, pp. 99-130.

[4] Park, J., Simpson, T. W., "Toward an activity-based costing system for product families and product platforms in the early stages of development", *International Journal of Production Research*, 2008, pp. 5285-5308.

[5] Meyer, M., Lehnerd, A. P., *The power of product platform – building value and cost leadship*, Free Press, New York, 1997.

[6] Simpson, T., Siddique, Z., Jiao, J., *Product Platform and Product Family Design*, Springer, US, 2006. Pp. 1-15.

[7] Li, L., Huang, G. Q., Newman, S. T., "Interweaving genetic programming and genetic algorithm for structural and parametric optimization in adaptive platform product customization", *Robotics and Computer-Integrated Manufacturing*, 2007, pp. 650-658.

[8] Fellini, R., Kokkolaras, M., Papalambros, P. Y., "Quantitative platform selection in optimal design of product families", *Journal of Engineering Design*, 2006, pp. 429-446.

[9] Panetto, H., Dassisti, M., Tursi, A., "ONTO-PDM: Product-driven ONTOlogy for Product Data Management interoperability within manufacturing process environment", *Advanced Engineering Informatics*, Elsevier, 2007, pp. 334-348.

[10] Vdovjak R., Houben, G. J., Stuchenschmidt, H., Aerts, A., *Semantic Web and Peer-to-Peer Decentralized Management and Exchange of Knowledge and Information*, Springer, 2006, pp. 41-58.

[11] Horrocks, I., Fengsel, D., Broekstra, J., Decker, S., Erdmann, M., Goble, C., van Harmelen, F., Klein, M., Staab, S., Studer, R., Motta, E., *The Ontology Inference Layer OIL*, 2001.

[12] Berners-Lee, T., Hendler, J., Lassila, O., "The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities", *Scientific American*, 2001.

[13] Hegge, H., *Intelligent Product Family Descriptions for Business Applications*, Technische Universiteit Eindhoven, 1995.

[14] Olsen, K. A., Saetre, P., Thorstenson, A., "A Procedure-Oriented Generic Bill of Materials", *Computers and Industrial Engineering*, 1997, pp. 29-45.

[15] Van Veen, E. A., Wortmann, J. C., "Generative Bill of Materials Processing Systems", *Production Planning & Control*, 1992, pp. 314-326.

[16] Vegetti, M., "Un modelo integrado para la representación de productos con estructuras complejas", 2007.

[17] Padula, N., "Definición de un modelo de familias de productos para un catálogo de notebooks utilizando ontologías", *9° Congreso Nacional de Estudiantes de Ingeniería en Sistemas de Información*,2015