Adopción de BPMN para Modelado CIM y su Transformación hacia Vistas de un modelo PIM

Leopoldo Nahuel[1,2], Leandro Rocca[1], Cecilia Ariste[1], Matías Caputti[1], Iván Zugnoni[1], Julieta Ponisio[1], Lautaro Mendez[1], Roxana Giandini[1,2]

[1] Laboratorio de Innovaciones en Sistemas de Información, LINSI.
Universidad Tecnológica Nacional, Facultad Regional La Plata.
Calle 60 esq. 124, La Plata, Buenos Aires, Argentina.

[2] Laboratorio de Investigación y Formación en Informática Avanzada, LIFIA.
Universidad Nacional de La Plata. Facultad de Informática.
Calle 50 y 115, La Plata, Buenos Aires, Argentina.

{lnahuel, lrocca, cariste, mcaputti, izugnoni, jponisio, lmendez, rgiandini}@linsi.edu.ar

Resumen. Bajo un desarrollo tradicional, cualquier cambio producido sobre un requisito, impactaba sobre el código fuente, produciendo inconsistencias y dificultades de mantenimiento. Actualmente, es el desarrollo dirigido por modelos (Model Driven Development, MDD), el paradigma más creativo que pretende resolver los problemas típicos del desarrollo de aplicaciones. En este trabajo se destaca la importancia de involucrar a los especialistas del dominio y usuarios del sistema, desde las primeras etapas a través de la obtención de modelos CIM creados entre informáticos y los stakeholders. Esta participación entre analistas y usuarios, tiene lugar a través de lenguajes de modelado desprovistos de conceptos computacionales que los usuarios puedan manejar rápidamente, como BPMN. Los modelos creados, servirán para la confección de un modelo CIM dentro de un desarrollo MDD, sobre el cual se aplicaran transformaciones para la obtención de ciertas vistas de modelo PIM, como diagramas de actividades y diagramas de clases conceptuales.

Palabras claves: Modelado de Software, BPMN, MDD, Transformaciones de modelos, ATL.

1 Introducción

El desarrollo de la informática en las últimas cuatro décadas ha estado signado por la elevación del nivel de abstracción de las tecnologías de desarrollo. Pasamos de lenguajes de bajo nivel escritos sobre editores de texto básicos, ha lenguajes de alto nivel, cada vez más cercanos al lenguaje humano, y a ambientes de desarrollo integrados muy sofisticados con multitud de asistencias al programador. Este proceso ha sido guiado por la necesidad de lograr aplicaciones cada vez más carentes de errores; fáciles de mantener y actualizar; procesos de desarrollo flexibles a los cambios en los requerimientos; y sobre todo, por la necesidad de que el software se ajuste bien a los requerimientos y cumpla los objetivos por los cuales fue

desarrollado. Esto último, aunque es obvio, es el motivo principal de todos los avances en desarrollo de aplicaciones desde el surgimiento del concepto "crisis del software" en la década de los 60.

En los últimos años hemos sido testigos del surgimiento de nuevas tecnologías que avanzaban en esta misma dirección: herramientas CASE [1] complejas, Frameworks, lenguajes de cuarta generación, etcétera. El paso siguiente fue evidente, y se comenzó a concentrar esfuerzos en la generación automática de código. Actualmente, el desarrollo dirigido por modelos (MDD por sus siglas en inglés) [2] se alza como el paradigma más novedoso para la creación de software. El Object Management Group [3] ha llevado a este paradigma a un estándar llamado Model Driven Architecture (MDA, pos sus siglas en inglés) [4]. La idea central subyacente a esta tecnología es que los modelos guíen todo el proceso de producción de software, y sea a partir de refinaciones sucesivas de estos, que el código se genere de forma automática. Bajo este paradigma, el eje se centra sobre el modelado, y con un motor de transformaciones, estos modelos van transformándose en otros más precisos y robustos, hasta llegar a la instancia en la que el próximo paso sea el código compilable (o interpretable). Los modelos en MDD tienen nombres que identifican el nivel de abstracción que tienen, desde el de más alto nivel llamado CIM (Computer Independent Model), pasando por el PIM (Platform Independent Model), hasta el más concreto de todos antes del código: el PSM (Platform Specific Model). De manera que partiendo del modelo de un CIM, mediante transformaciones automáticas se pueden obtener modelos más refinados y robustos, que contemplen las tecnologías de desarrollo y las plataformas subyacentes, hasta llegar a un modelo PSM que será transformado a código que puede ser ejecutado.

Alrededor de estas nuevas formas de desarrollo de software, se han generado múltiples técnicas y tecnologías que ayudan a completar el paradigma MDD y las transformaciones. Entre estas herramientas se encuentran los lenguajes de especificación de transformaciones, como ATL [5]. El Atlas Transformation Language es un lenguaje para escribir transformaciones. Funciona integrado con el ambiente de desarrollo Eclipse a través de un plugin, y es capaz de realizar transformaciones modelo a modelo. Para que este motor de transformaciones funcione, el modelo fuente y el destino deben tener una definición precisa a través de un lenguaje gráfico de más bajo nivel que los especifique (de manera análoga a como BNF describe lenguajes formales). El metamodelo estándar que describe los modelos gráficos utilizados en MDD es MOF (Meta Object Facillity) [6] Esto concluye una plataforma tecnológica integral capaz de lograr transformaciones modelo a modelo escritas en ATL, dentro de un proceso de desarrollo dirigido por modelos. El proceso señalado puede darse en la etapa de CIM a PIM, como de PIM a PSM. El nivel de abstracción de los modelos indicaran en qué paso del desarrollo nos encontramos, pero las técnicas explicadas en este trabajo pueden extenderse a modelos de todos los niveles de abstracción.

A través de modelos con altos niveles de abstracción, es posible involucrar a personas ajenas a la informática, pero relacionadas en el dominio de implantación del sistema en desarrollo. Esta interacción puede conseguirse a través de modelos BPMN [7], los cuales pueden fácilmente ser entendidos, e incluso, desarrollados por los propios usuarios. Esto es altamente beneficioso, ya que incorpora una herramienta más para la recopilación de requisitos del sistema, al clásico abanico de posibilidades

que tenían los analistas tradicionalmente. El modelado de procesos de negocio, realizado por los propios involucrados e interesados del sistema, provee una fuente confiable y sólida de información que los analistas pueden utilizar para la confección de un CIM, o parte de él. Este modelo CIM, bajo un desarrollo dirigido por modelos, se puede transformar en diversos diagramas UML (con las limitaciones de semántica que BPMN tiene, claro está) que completaran el CIM, y ayudarán a la creación de un modelo PIM, más preciso y útil para las etapas de desarrollo siguiente.

2 Marco Teórico: antecedentes y trabajos previos

Dentro del marco teórico y los objetivos que marcan el contexto de este trabajo, se deben mencionar algunos artículos previos. Los mismos forman la base de conceptos y procedimientos que aquí se detallan. En primer lugar, en [8] se menciona la idea del uso del lenguaje BPMN en un entorno de desarrollo ágil. Continuando el desarrollo de este aporte, en [9] se destaca la construcción de modelos de procesos de negocio (Business Procces Diagram) como parte del desarrollo de un modelo CIM que refleje los requerimientos del dominio del sistema. Por último, en [10] se presenta una alternativa sobre software libre para la construcción de una plataforma de transformación de modelos, presentando algunos ejemplos de transformación tomando como entrada modelos BPMN. A partir de estos desarrollos, el presente artículo avanzará sobre éstas ideas, aunando los esfuerzos conseguidos, y formalizando una propuesta de transformación con inclusión de usuarios (que por lo general no tienen conocimientos de computación) que ayuden al modelado de un CIM que sirva como entrada para lograr parte de un PIM a través de transformaciones automáticas.

3 Un motor para transformaciones Model-To-Model en Eclipse

El entorno para desarrollo de software Eclipse [11], en conjunción con el lenguaje ATL, nos proveen un ambiente adecuado para la confección de un motor de transformaciones capaz de cumplir con las primeras etapas de un proceso MDD.

Un proceso de transformación completo haciendo uso de ATL, se compone de los siguientes elementos: metamodelos fuente y destino; modelo fuente; código ATL. En la figura 1 se puede observar la interacción de estos componentes. La definición de todos los metamodelos está basada sobre MOF. Si bien no se mencionó, aunque ATL es un lenguaje textual, también puede definirse con MOF.

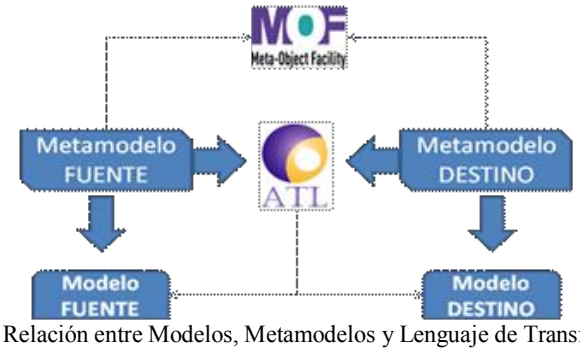

Figura 1: Relación entre Modelos, Metamodelos y Lenguaje de Transformación.

3.1 El Metamodelo Fuente y el Metamodelo Destino

La creación de MOF como estándar regulado por la OMG, trajo consigo una formalidad que se echaba en falta en el modelado de sistemas. Los lenguajes gráficos aparecieron, pero al revés de los lenguajes textuales, éstos no tenían una base formal desde dónde ser definidos. A partir de MOF esto cambio, y los lenguajes de modelado de software, pasaron a contar con un metalenguaje que los definía de manera precisa. Con MOF se han definido una gran diversidad de lenguajes gráficos, desde UML hasta BPMN, pasando por otros no tan relacionados al software, como las redes de Petri.

Un metamodelo es un modelo a partir del cual puede ser instanciado todo un lenguaje de modelado. Todos los lenguajes gráficos aquí mencionados tienen su metamodelo escrito con MOF. A través del estándar XMI [12], los metamodelos MOF pueden ser transcriptos en forma textual. Esto es aprovechado por ATL y el motor de transformaciones. De manera general, cada elemento del metamodelo de un lenguaje, va a tener una etiqueta XMI que lo define, conjuntamente con algunos atributos específicos.

Ejemplo 1: etiquetado para el elemento "class" de UML:

```
<eStructuralFeatures xsi:type="ecore:EReference"
name="class" ordered="false"
eType="#//Class" changeable="false" volatile="true"
transient="true">

 <eAnnotations
source="http://www.eclipse.org/emf/2002/GenModel">

<details key="documentation" value="References the Class
that owns the Property."/>

 </eAnnotations>

      <eAnnotations source="subsets"
references="#//NamedElement/namespace
#//Feature/featuringClassifier"/

 </eStructuralFeatures>
```

Para los fines de este artículo, se seguirá un ejemplo sencillo de transformación desde un diagrama BPMN, hacia tres tipos de diagramas UML: diagrama de actividades, de Casos de Uso, y de clases. La elección de BPMN se debe a que es un lenguaje desprovisto de conceptos computacionales y técnicos, lo cual lo hace especialmente útil para la confección de modelos CIM en cuya elaboración se encuentren involucrados personas que formen parte del dominio del sistema, y que no sean informáticos.

Los metamodelos, tanto fuente como destino, deben estar escritos de manera textual con XMI. A partir de los mismos, la transformación ATL es ejecutada.

3.2 El lenguaje ATL y una transformación Model-To-Model

El motor de transformaciones puede interpretar el código ATL y pasarlo a un bytecode específico que una máquina virtual es capaz de ejecutar. De esta manera, el

toolkit que se integra a Eclipse como un plugin, cuenta con tres capas: el lenguaje ATL, el compilador (que compila a bytecode), y la máquina virtual para ejecución del bytecode. El proceso de interpretación y ejecución es completamente transparente para el desarrollador. Solo basta con escribir las transformaciones y contar con los metamodelos involucrados.

El lenguaje ATL cuenta con dos modos de programación: imperativo y declarativo. El elemento principal para describir una transformación es llamado regla. Las reglas sirven para indicar cuál elemento del modelo fuente, se convertirá a otro elemento del modelo destino. Esta relación es unívoca, de manera que para cada elemento (o conjunto de ellos) del modelo origen, existe una, y solo una relación con otro elemento del modelo destino. Existen tres formas para declarar una regla: las "matched rules" que corresponden a la forma declarativa de programación; y las "lazy rules" y las "called rules" que corresponden a la forma imperativa. En este trabajo usaremos la forma declarativa, escribiendo la transformación con reglas matched.

Ejemplo 2: forma típica de declaración de reglas en forma declarativa:

```
-- @path bpmn=/test2/bpmn1.ecore
-- @path UML=/test2/MAPS_UML.ecore
module trans;
create OUT: UML from IN: bpmn;
-- Reglas de transformación

rule BPMNDiagram2UMLDiagram {
from
s: bpmn!ProcessModel
to
t: UML!UMLDiagram (
name <- s.name,
activity <- s.pools)}

rule Task2Activity {
from
s: bpmn!Task
to
t: UML!Activity (
name <- s.name)}
```

Las líneas que comienzan con "**--@path"**, muestran cómo indicarle al motor de ATL dónde encontrar los metamodelos (archivos con extensión .ecore) y darles un alias para usar en el código.

Ejemplo 3: una etiqueta XMI para el elemento "Activity" del metamodelo de UML se escribe de la siguiente forma:

```
//Metamodelo BPMN

<eClassifiers   xsi:type="ecore:EClass"   name="Activity"
abstract="true"             eSuperTypes="#//FlowObject"/>
<eClassifiers     xsi:type="ecore:EClass"     name="Task"
eSuperTypes="#//Activity"/>
……

//Metamodelo UML
```

```
<eClassifiers xsi:type="ecore:EClass" name="Activity"
eSuperTypes="#//Behavior"/>
```

4 Uso de BPMN para la construcción de un modelo CIM

En MDD, el primer modelo que se confecciona es el modelo independiente de la computación, o CIM. Este modelo está integrado por diversos elementos y diagramas, que están (o deben estarlo) alejados de cualquier tecnología, idea, o noción de informática. En la literatura sobre MDD, se comienza a hablar sobre transformaciones automáticas a partir del modelo PIM. Este modelo, incorpora mayor precisión y está compuesto por elementos relacionados con la computación, aunque en conceptos generales, y todavía se mantiene aislado de la plataforma específica de implementación. Es a partir de este modelo, que MDD comienza a transformar para lograr obtener el código en una última etapa. Resulta conveniente pensar en los beneficios de una transformación automática, desde antes de un modelo PIM, es decir, desde el primer modelo: el CIM. Para este fin, se dispone a crear diagramas de procesos de negocio con un lenguaje gráfico de alto nivel de abstracción: el BPMN. A partir de estos diagramas, seremos capaces de lograr tres tipos de transformaciones para llegar a tres diferentes modelos destinos: un diagrama de casos de uso; un diagrama de actividades; y un diagrama conceptual de clases.

4.1 Modelado con BPMN como técnica adicional para los analistas de negocio

Se han mencionado ya los beneficios que se obtendrían haciendo participes de la etapa de modelado a los usuarios y especialistas del dominio. Esto incluso puede verse como una herramienta más en la adquisición de requerimientos en la primera etapa de desarrollo. Las técnicas tradicionales, como entrevistas, lectura de documentación, etc., suelen estar acompañadas de ciertas incertezas, y ambigüedades. La incorporación de esta técnica podría aportar más valor al proceso inicial del desarrollo.

Las personas que están relacionadas al dominio del sistema, o aquellos interesados en su implantación, generalmente serán ajenas a la disciplina informática. Sería muy difícil por lo tanto, pedirles que hagan diagramas haciendo uso de un lenguaje como UML. Se necesita entonces, un lenguaje que sea fácil de entender, y de un algo lenguaje de abstracción que transparente todos aquellos conceptos con los cuales las personas no estén involucrados. El lenguaje de Notación para Modelado de Procesos de Negocio (BPMN por sus siglas en inglés), adquiere relevancia en este apartado. Su intuitiva sintaxis y su simplicidad, hacen que sea relativamente sencilla su utilización. De esta manera, se puede pedir a los stakeholders que modelen procesos con BPMN, los cuales serán la entrada de nuestras transformaciones. En el proceso de modelado de estos procesos de negocio no hay involucradas personas intermedias. El volcado de información sobre los mismos es directo, y por ende, se minimiza la posibilidad de malos entendidas o interpretaciones erróneas.

4.2 Limitaciones semánticas de BPMN para transformaciones automáticas

Una transformación entre modelos, solo tiene lugar gracias a las características semánticas de los lenguajes involucrados. Se debe mapear entre elementos semánticos similares entre un modelo fuente, y otro modelo destino. Mientras más significados, conceptos y relaciones compongan un modelo fuente, las posibilidades de

transformación se incrementaran. El BPMN es un lenguaje acotado en comparación con UML. Esto es debido a la diversidad y complejidad de un lenguaje de modelado como UML, el cual es de propósito general. Los diversos diagramas que componen su notación, hacen que éste sea capaz de modelar componentes del sistema, estructurales y de comportamiento. Para el caso de BPMN esto no es así, dado que fue creado con la finalidad de modelar procesos de negocio. Los elementos que componen éste lenguaje, y los diagramas creados a partir de los mismos, están relacionados con procesos y actividades dentro del sistema, lo cual estaría emparejado con la parte de modelado de comportamiento dentro de los modelos de UML. Es por esto que no es factible obtener la información necesaria, a partir de un diagrama BPMN, para lograr obtener una gran diversidad de otros diagramas a partir de una transformación. Menos aún si la intención es transformar a diagramas que modelen componentes de la arquitectura del sistema.

La idea de generar parte de un CIM con diagramas BPMN presenta limitaciones obvias por lo ya comentado. Pero sí es posible obtener un conjunto de diagramas que pueden ayudar al analista a tener un panorama más completo del sistema, con un mínimo esfuerzo a partir de la aplicación de transformaciones automáticas dentro de un desarrollo MDD. Más adelante se tratará una posibilidad de transformación hacia un modelo destino que contengo información estructural del sistema con un diagrama de clases conceptuales de muy alto nivel de abstracción, pero no por esto deja de ser de gran utilidad.

5 Transformaciones M2M entre BPMN y UML con ATL

Las transformaciones modelo a modelo, o model-to-model, son una parte fundamental dentro del desarrollo dirigido por modelos. Si bien se ha naturalizado que en un desarrollo MDD, el motor de transformaciones tiene participación a partir de un modelo PIM (generado manualmente) en este trabajo se plantea la posibilidad de comenzar con transformaciones entre modelos en una etapa anterior: desde el modelo CIM. El primer modelo, y por tanto más abstracto e independiente de la computación, es visto desde un desarrollo clásico dirigido por modelos, como un conjunto de elementos, de confección manual y cuyos constructores son solamente los analistas e involucrados con el desarrollo. El CIM, tradicionalmente, es la culminación de un proceso de relevamiento. Aquí se intenta ampliar esta concepción, adjudicando dos nuevas alternativas y conceptos en su construcción: por un lado se plantea la posibilidad de lograr transformaciones a partir del modelo CIM que culminen en un modelo CIM más completo, y que con algún esfuerzo pueda también resultar en parte de un modelo PIM; por otro lado, el uso de BPMN por parte de los stakeholders puede incluir beneficios en el proceso de construcción del software en las primeras etapas, al incluir en su desarrollo a aquellas personas interesadas en la implantación del sistema. Concretamente, las transformaciones descritas en este trabajo apuntan a lograr la automatización del modelado partiendo desde un modelo independiente de la computación. Los nuevos modelos obtenidos, serán parte de una nueva versión del CIM, más completa y diversa. Además, se prevé que con algo de trabajo adicional por parte de analistas, algunos de estos modelos sean parte de un PIM. Tal es el caso de los diagramas de clase obtenidos de una transformación planteada en una sección posterior, que pueden completarse con poco esfuerzo, y que podrían formar parte de un modelo menos abstracto que un CIM.

En las siguientes secciones se plantean tres transformaciones posibles partiendo desde un diagrama BPMN: diagrama de actividades, diagrama de casos de uso y un diagrama de clases conceptuales.

5.1 Transformación de diagrama de procesos de negocio (BPMN) a diagrama de actividades (UML)

La obtención de diagramas de actividades como conclusión de un proceso de transformación a partir de diagramas BPMN, resulta un proceso bastante simple. El mapeo es bastante directo y por ello es muy simple ver las relaciones entre los metamodelos y la transformación escrita en ATL. En el ejemplo de código 2 se puede ver la forma típica de declaración de reglas en forma declarativa. En el ejemplo 3, se ejemplificó la forma de etiquetado de XMI para el elemento "Activity" de metamodelo de UML. También, en el ejemplo de código 3, se pueden observar las partes del código XMI de los metamodelos BPMN y UML (fuente y destino de la transformación) que hacen referencia al elemento Actividad del diagrama BPMN y al elemento Actividad del diagrama UML. En el ejemplo 2 está el código de la regla que relaciona ambos conceptos y deriva uno a través del otro ejecutando la transformación. Observe que, en realidad, el elemento que empareja la regla declarada es "Task". En el metamodelo, "Task" es una instancia concreta que hereda de "Activity", la cuál es una clase abstracta. Esto es posible dado que en MOF son válidos los conceptos de objetos como clases y herencia.

5.2 Transformación de diagrama de Procesos de Negocio (BPMN) a diagrama de Casos de Uso (UML)

Las características semánticas de los modelos de procesos de negocio, hacen que sea posible obtener otro tipo de diagrama de comportamiento UML: diagramas de caso de uso. Se seguirá la metodología propuesta en [13] en la cual se plantea el siguiente mapeo entre elementos de los metamodelos:

1. **BpmnDiagram → Model**: tanto el diagrama fuente, como el destino, se encuentran circunscriptos en contenedores conceptuales que encuadran a los mismos bajo un nombre que los identifican. Esta relación mapea el nombre de los diagramas.

2. **SubProcess → UseCase**: cada subproceso del diagrama fuente, representa una ejecución particular del sistema, sobre el contexto que indica su carril. Por lo tanto, cada subproceso puede tomarse como una unidad funcional, y por lo tanto es posible mapear hacia un caso de uso UML.

3. **Activity (Task) → UseCase**: las actividades tienen un significado semántico similar a los subprocesos en BPMN, y es por esto que el mapeo es igual. Cada actividad, se convierte en un caso de uso.

4. **Activity (Gateway) → UseCase**: el elemento "Gateway" de la notación de BPMN, es en definitiva, una actividad en la cual se realiza un proceso de decisión para realizar una bifurcación en el proceso, para definir continuar por alguno de los diferentes caminos posibles. Este proceso, al igual que los subprocesos y las actividades comunes, se empareja a un caso de uso.

5. **Lane → Actor**: un "Lane" o carril, pone en contexto la ejecución de las actividades que se dan sobre él, y por ende otorga una categorización de las tareas. Esta organización puede expresarse en un rol el cuál es representado en un actor del diagrama de casos de uso a través de una transformación.

6. **SecuenceFlow → UseCase**: las flechas que indican un flujo de secuencia en BPMN denotan un orden de ejecución entre actividades. Este avance significa la finalización de una actividad, y el comienzo de otra. Entre ambas actividades, habrá un procedimiento que permita este cambio entre una y otra, dentro del curso de ejecución modelado. Por tanto, puede verse a esta ejecución de finalización y comienzo de actividad, como una unidad funcional del sistema, modelado en un caso de uso.

5.3 Transformación de diagrama de Procesos de Negocio (BPMN) a diagrama de Clases Conceptuales (UML)

Un diagrama de clases es parte de los componentes de modelado estructural de UML. Sus componentes modelan la arquitectura del sistema y está relacionado a elementos estáticos del sistema, alejado de los procesos de negocio modelados a partir de BPMN. Sin embargo, es posible obtener un primer acercamiento a un diagrama de clases, de un alto nivel de abstracción, a partir de modelos BPMN. Como ya se ha mencionado, todo modelo fuente tiene sus limitaciones en una transformación, y por tanto no es posible lograr cualquier tipo de modelo destino. Aquí se plantea un mapeo simple entre elementos de ambos lenguajes. El modelo de clases que se obtendrá constará de clases sueltes con algunos métodos. Los atributos y las relaciones entre clases podrán ser completados posteriormente mediando un proceso manual de los analistas con un esfuerzo considerablemente menor al que resultaría de realizar todo el proceso de forma manual.

Para conseguir un modelo estructural compuesto de clases sueltas con algunos métodos, a partir de un diagrama de un proceso escrito en BPMN, se plantea el siguiente emparejamiento entre elementos de los metamodelos que los componen:

1. **Carriles y Clases**: los carriles o "lane", son elementos que separan los procesos dentro de un diagrama BPMN según su rol o categoría. Un carril es el contexto donde se ejecutan ciertas actividades de un proceso, y por ende puede ser tomado como una Clase de UML que tendrá ciertos métodos relacionados a las actividades que se ejecutan dentro del carril.

2. **Task y Métodos de clase**: siguiendo la idea mencionada en un mapeo lane-clase, las actividades que se dan dentro de un carril, serán los métodos de la clase conseguida a partir de ese carril.

6 Conclusiones y Trabajo Futuro

La posibilidad de comenzar un desarrollo dirigido por modelos, mediante la aplicación de transformaciones automáticas a partir del primer modelo, se halla como una idea atractiva dentro de MDD. La bibliografía tradicional sobre el tema, plantea al modelo CIM como un producto del trabajo artesanal del analista, y compuesto por una heterogeneidad de componentes que pueden llegar a ser muy disímiles entre sí. La idea propuesta aporta homogeneidad y criterio al desarrollo de un modelo

independiente de la computación, con el agregado del diseño de tres tipos de transformaciones posibles a partir de diagramas BPMN. El producto obtenido, compone un CIM mucho más rico en contenido y semántica, lo cual contribuye a un mejor modelo del sistema sobre el cual se trabaja. A su vez, se presentó la idea subyacente al desarrollo de estas transformaciones, de lograr hacer uso de la experiencia de los stakeholders. Estas personas, que por lo general no tienen contenidos computacionales ni formación en informática, sí conservan una alta experiencia y un elevado conocimiento sobre el dominio del sistema sobre el cual se trabaja. La participación activa de estos usuarios, a través del modelado con BPMN, enriquece el proceso de obtención de requerimientos, y ayuda a los analistas a obtener un modelo CIM más claro.

Como líneas de trabajo futuro, se pretende lograr una unificación de las transformaciones propuestas, en una metodología de diseño de modelo CIM, capaz de orientar las actividades y aportar solidez al desarrollo. A su vez, se puede proponer otro tipo de transformación a partir de diagramas de procesos de negocio: obtención de diagramas de máquina de estados.

Referencias Bibliográficas

1. Sommerville, I. "Ingeniería del Software". 7° Edición. Pearson Education S.A., pp. 11, 79-83 (2005). ISBN-10: 8478290745.
2. Pons, C., Giandini, R. y Pérez, G. "Desarrollo de Software Dirigido por Modelos: conceptos teóricos y su aplicación práctica". 1°Edición. EDULP & McGraw-Hill Education, Argentina, pp. 28-38 (2010). ISBN-13: 9789503406304.
3. Object Management Group, OMG (2015). www.omg.org
4. OMG Contact: Watson, A. "Model Driven Architecture Guide". Version 2.0, (2014). http://www.omg.org/cgi-bin/doc?ormsc/14-06-01.
5. ATL/User Guide Introduction (2015) http://wiki.eclipse.org/ATL/User_Guide
6. Meta Object Facility. Core Specification. Version 2.4.2, (2014). http://www.omg.org/spec/MOF/2.4.2/
7. BPMN Specification. Versión 2.0.2, (2013). http://www.omg.org/spec/BPMN/2.0.2/
8. Giandini, R., Nahuel, L., Mendez, L., Mangano, M. "Los procesos ágiles en la producción de productos software en ambiente MDD". COMTEL (2012).
9. Mendez, L., Di Girolamo, R., Vargas, L., Pérsico, M., Santos, N. "Lenguajes notacionales y técnicas de ingeniería de software basada en modelos aplicado a entornos bpm". Jornadas de Estudiantes Investigadores, JEI (2012).
10. Perelli, J., Mendez, L., Pérsico, M., Martínez Astudillo, I., Blasi, N. "Desarrollo de prototipo case para transformación de modelos en contexto mdd aplicado a modelos BPMN". Jornadas de Estudiantes Investigadores, JEI (2012).
11. Eclipse IDE, https://eclipse.org/
12. XML Metadata Interchange, especification. Versión 2.4.2 (2014). http://www.omg.org/spec/XMI/2.4.2/
13. Giandini, R., Nahuel, L., Roca, L., Caputi, M., Zugnoni, I. "Implementando Transformación de Modelos utilizando MOSKitt Tool en adhesión al Paradigma MDD". Congreso Nacional de Ingeniería en Informática/Sistemas de Información, CoNaIISI (2014).