Un lenguaje de Transformación específico para Modelos de Proceso del Negocio

Roxana Giandini[1] Gabriela Pérez[1] Claudia Pons[2]

[1] LIFIA, Facultad de Informática, Universidad Nacional de La Plata
[2] Universidad Abierta Interamericana (UAI)
Buenos Aires, Argentina
[giandini,gperez,cpons]@lifia.info.unlp.edu.ar

Resumen. En el desarrollo dirigido por modelos (MDD), los lenguajes de modelado juegan un rol central. Abarcan desde los más genéricos como UML hasta los llamados lenguajes específicos del dominio (DSL). En el dominio de los procesos de negocio, el lenguaje estándar de modelado BPMN se ha tornado popular y tal como sucede en otras áreas, el diseño de un modelo BPMN necesita ser mejorado a través de técnicas de refactorización. En MDD, la refactorización de modelos es vista como una forma particular de transformaciones de modelos. Así como BPMN es un DSL para construir modelos de proceso, seria deseable también contar con un lenguaje de transformaciones específico del dominio (DSTL), en lugar de utilizar un lenguaje genérico para transformaciones como QVT para especificar refactorización de modelos de proceso. En este trabajo proponemos un DSTL para expresar transformaciones de modelos especificados en BPMN que asista al desarrollador proveyendo construcciones específicas para refactorización de modelos de proceso.

Palabras clave: Desarrollo Dirigido por Modelos, BPMN, Transformaciones de Modelos, DSTL

Abstract. In Model-Driven Development (MDD), modeling languages play a central role. They range from the more generic modeling languages like UML to the so-called Domain-Specific Languages (DSL). In the business process domain, the BPMN modeling standard language has become popular. Like in other areas, the design of a BPMN model needs to be improved through Refactoring techniques. In MDD, model refactoring is seen as a particular form of model transformations. As BPMN is a DSL for building process models, it would be desirable to have a Domain-Specific Transformation Language (DSTL), instead of a generic transformation language like QVT to specify process model refactoring. In this paper we propose a DSTL to express transformations of BPMN models in order to assist the developer by providing specific constructions for process model refactoring.

Keywords: Model Driven Development, BPMN, Model Transformation, DSTL

1 Introducción

El desarrollo dirigido por modelos (MDD) [1] [2] [3] ha emergido recientemente como un nuevo y prometedor paradigma en Ingeniería de Software (IS) [4]. MDD promueve el uso de modelos no sólo para documentación y comunicación sino como artefactos de primera clase para generar otros productos durante el proceso de desarrollo, tales como otros modelos y código fuente. Los lenguajes de modelado juegan un rol central en MDD. Abarcan desde los más genéricos como UML [5] hasta

los llamados lenguajes específicos del dominio (DSL) [6] [7], es decir lenguajes cuyos constructores representan conceptos de un dominio de problema específico.
La propuesta de Gestión de Procesos de Negocios (BPM) [8] ha adquirido una atención considerable recientemente tanto por las comunidades de administración de negocios como por la de ciencia de la computación. BPM provee un conjunto de metodologías para el análisis, comprensión y documentación de los procesos de negocios. En este dominio, el lenguaje de modelado estándar del OMG [9], Business Process Modeling Notation (BPMN) [10] [11] se ha tornado popular. Similarmente a lo que sucede en otras áreas de modelado en IS, el diseño de un modelo en BPMN necesita ser adaptado con el fin de mejorar su mantenimiento y calidad. El mecanismo típicamente usado para mejorar la calidad de modelos y código es la refactorización. En MDD las técnicas para refactorización de modelos son vistas como una forma particular de transformaciones de modelos. Una transformación de modelos consiste de un conjunto de reglas que describen cómo un modelo escrito en un lenguaje fuente es mapeado a un modelo escrito en un lenguaje destino. Hay diversas propuestas para especificar, implementar y ejecutar transformaciones de modelos [12]. Algunos ejemplos de lenguajes de propósito general para transformación de modelos, son el estándar QVT [13] y otros inspirados en él como ATL [14] [15] y RubyTL [16]. Estos lenguajes permiten especificar diversas variantes de transformaciones de modelos. Aunque los beneficios de ser genéricos y estándares son claros, es también claro que elevar el nivel de abstracción con lenguajes de transformación específicos del dominio (DSTL) nos brindará las mismas ventajas que al usar cualquier otro DSL. Es decir, si usamos un DSL (como BPMN) para construir los modelos de proceso, en lugar de UML, seria deseable también contar con un lenguaje de transformación especifico para dicho dominio. Este lenguaje proveerá operaciones específicas sobre los elementos del dominio, en vez de reglas generales de transformación como sucede en un lenguaje genérico como QVT.
En este artículo proponemos un DSTL para especificar refactorizaciones de modelos de proceso del negocio. Lo llamamos *Business Process Modeling Transformation Language* (BPMTL). El trabajo se organiza como sigue: la sección 2 introduce conceptos del lenguaje BPMN; la sección 3 describe las operaciones consideradas en BPMTL. En la sección 4 definimos la sintaxis de BPMTL y formalizamos su semántica a través del uso del lenguaje ATL. La sección 5 muestra un ejemplo de aplicación del lenguaje y la 6 presenta trabajos relacionados y conclusiones.

2 El lenguaje BPMN

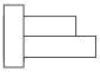
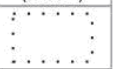
El objetivo primario del lenguaje estándar BPMN fue proveer una notación que sea legible y entendible para todos los usuarios de negocios, desde los analistas que realizan el diseño inicial de los procesos y los responsables de desarrollar la tecnología que ejecutará estos procesos, hasta los gerentes de negocios encargados de administrar y realizar el monitoreo de los procesos. BPMN define un modelo de procesos de negocio basándose en diagramas de flujo. Un modelo de procesos de negocio, es una red de objetos gráficos que representan las actividades (por ejemplo tareas) y los controles de flujo que definen su orden de ejecución [17]. Hasta la aparición de BPMN no existía un estándar específico sobre técnicas de modelado desarrollado para estos fines. BPMN ha sido desarrollado para proveer una notación

estándar a los usuarios, de forma análoga a como UML estandarizó el mundo del modelado en la IS. A continuación, describimos los elementos básicos de BPMN.

2.1 Elementos del lenguaje

Los elementos utilizados para construir los modelos BPMN fueron elegidos para ser distinguibles unos de otros y utilizar las figuras que son familiares a la mayoría de los diseñadores. Por ejemplo, las actividades se representan mediante rectángulos y las decisiones mediante rombos.

Tabla 1. Elementos de BPMN

Categoria	Elemento	Descripción	Gráfica
Objetos de Flujo	Evento	Es algo que sucede durante el curso del proceso de negocio. Afectan al flujo del proceso. Normalmente tienen una causa (disparador) o un impacto (resultado). Dependiendo de cuando afectan al flujo serán eventos iniciales, intermedios o finales.	
	Actividad	Es un termino genérico para el trabajo que realiza una compañía. Puede ser atómica (tarea) o compuesta (sub-proceso). Para indicar la no atomicidad se coloca un signo + en la esquina del símbolo de actividad.	
	Gateway	Se utiliza para controla la convergencia o divergencia de flujos. Representa una decisión para mezclar o unir caminos.	
Objetos Conectores	Secuencia	Se utiliza para mostrar el orden o secuencia en que las actividades se realizan en un proceso	
	Mensaje	Se utiliza para mostrar el flujo de mensajes entre dos participantes separados.	
	Asociación	Se utiliza para mostrar entradas y salidas de actividades.	
Swimlanes	Pool	Representa un participante en un proceso. Actúa como contenedor grafico para particionar un conjunto de actividades.	
	Lane	Es una sub-particion dentro de un pool y puede extenderse a todo lo largo o ancho del pool. Se utilizan para organizar y categorizar actividades.	
Artefactos	Objeto de Datos	Mecanismo para mostrar como los datos son requeridos y producidos por las actividades. Se conectan a las actividades por asociaciones.	 Nombre (estado)
	Grupo	Se utiliza para documentación o para propósitos de análisis, pero no afecta al Flujo de Secuencias	
	Anotación	Mecanismo para que quien esta modelando provea información adicional para el lector del diagrama.	

El enfoque adoptado fue la organización de la notación en categorías específicas. Estas categorías permiten al lector del diagrama de procesos de negocio reconocer

fácilmente los elementos básicos y comprender el diagrama. Las cuatro categorías básicas de elementos son: *flow objects* (objetos de flujo), *connecting objects* (objetos conectores), *swimlanes* (andariveles) y artefactos. Dentro de estas categorías de elementos se pueden incluir variaciones adicionales o información para soportar requerimientos complejos sin agregar demasiada complejidad al diagrama. La Tabla 1 muestra los elementos que forman cada categoría, su descripción y notación gráfica.

3 Refactorización de modelos de proceso

En [20], los autores adaptan refactorizaciones convencionales en la IS a las necesidades de los modelos de proceso y las completan con otras específicas para BPM. La aplicación de estas operaciones transforma un modelo de proceso P en un nuevo modelo de proceso P´. Varias de las técnicas descriptas no sólo se aplican a actividades (tareas o subprocesos), sino también a artefactos (grupos) con una sola entrada y una sola salida. El término *Fragment* es utilizado para unificar estas posibilidades. Consideramos aquí algunas de las refactorizaciones básicas propuestas en [20], más comúnmente usadas:

Rename Activity: La aplicación de *Rename Activity* genera que el nombre de una actividad x sea cambiado a y. Es comparable a *Rename Method* [21].

Substitute Fragment: Usando *Substitute Fragment* los diseñadores de procesos pueden reemplazar un fragmento por otro. Esto puede resultar útil en ciertos casos, por ejemplo porque el nuevo fragmento sea más simple o realice la misma tarea en forma diferente; se compara con la refactorización *Substitute Algorithm* [21].

Extract Fragment: La aplicación de *Extract Fragment* nos permite extraer un fragmento generando un nuevo subproceso, con el fin de eliminar redundancias en el modelo, promover reuso o bien para reducir el tamaño del modelo. Es similar a *Extract Method* [21].

Replace Fragment: Aplicando *Replace Fragment by Reference*, un fragmento puede reemplazarse por una actividad compleja (subproceso) cuyo contenido coincide con el fragmento, haciendo referencia a ella. Esta refactorización es generalmente usada en combinación con *Extract Fragment.*

4 BPMTL: Un DSTL para refactorización de Modelos de Proceso del Negocio

Definimos en esta sección un lenguaje de transformaciones específico para el dominio de los Modelos de Proceso del Negocio con el fin de expresar refactorizaciones de modelos definidos en BPMN. Llamamos a nuestro lenguaje *Business Process Model Transformation Language* (BPMTL). Cabe aclarar que el alcance de BPMTL se limita a la definición de refactorizaciones aplicables a modelos BPMN. La verificación de la preservación semántica de estos modelos queda fuera del alcance de esta propuesta.

4.1 Metamodelo de BPMN

Debido a que el lenguaje de transformación propuesto permitirá transformar modelos expresados en BPMN, debemos tener definida formalmente la sintaxis de este lenguaje a través de un metamodelo, instancia del estándar MOF [18]. De acuerdo a las categorías de elementos vistas en la sección 2, hemos definido un metamodelo simplificado de BPMN inspirado en el que propone Eclipse [29]. La Figura 1 muestra este metamodelo. Presentamos también algunas de las reglas de buena formación que deben cumplir sus metaclases, especificadas en OCL [19]:

```
--los elementos Connection no pueden relacionar otros elementos
connection
context Connection
inv: not self.target.oclIsKindOf(Connection) and not
self.source.oclIsKindOf(Connection)

--los elementos Activity en un Pool deben tener distinto nombre.
context Pool
inv:  self.processObjects->  select(e|e.oclIsKindOf(Activity))->
forAll(e1, e2| e1.name =e2.name implies e1=e2)
```

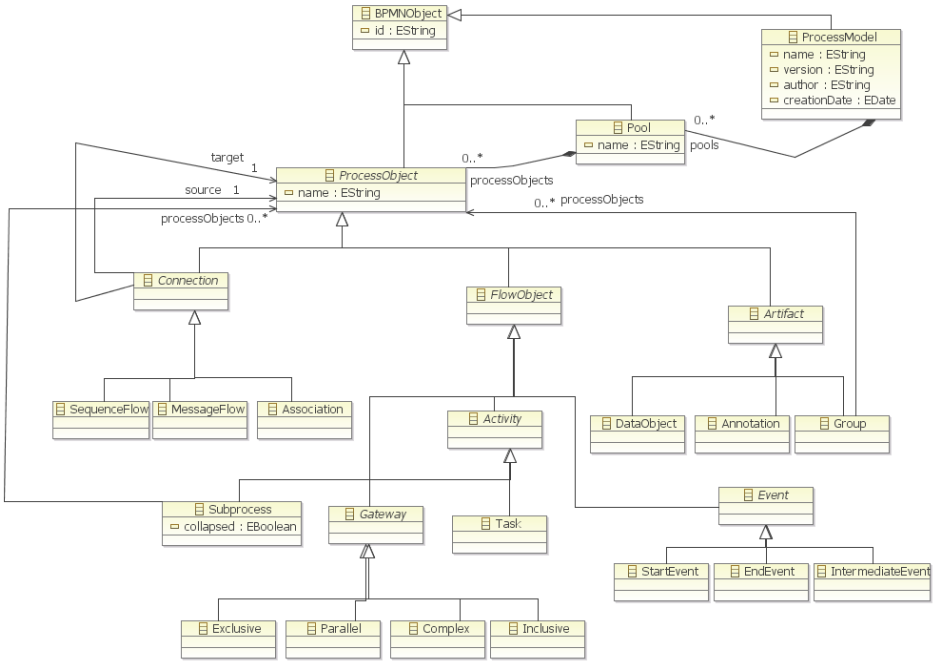


Figura 1. Metamodelo BPMN simplificado

4.2 El lenguaje de transformación BPMTL

Como mencionamos anteriormente, BPMTL nos permite denotar refactorizaciones en el dominio de modelos de proceso, a través de transformaciones. Incluiremos inicialmente en la definición del lenguaje, las cuatro refactorizaciones básicas descriptas en la sección 3. La sintaxis abstracta de BPMTL se define de la siguiente manera:

<**BPMNrefactoring**> ::= ***BPMNrefactoring*** **{** ***inputModelPath*** <string>
refactorings {<**BasicRefactoring**> } }

<BasicRefactoring> ::=

renameActivity {***name*** <string> ***newName*** <string> } |

substituteFragment **{** ***oldFragmentName*** <string> ***newFragmentName*** <string>

inputModelPath <string>} |

extractFragment **{** ***fragmentName*** <string> } |

replaceFragment {***oldFragmenName*** <string> ***subprocessName*** <string>} |

<**BasicRefactoring**> ; <**BasicRefactoring**>

<**string**> ::= a | b | c | … | <string> <string>

Donde el ***inputModelPath*** inicial hace referencia al archivo que contiene al modelo BPMN sobre el que se aplicarán las refactorizaciones. La aplicación de ***substituteFragment*** necesita también hacer referencia a otro archivo (*inputModelPath*) conteniendo el modelo BPMN con el nuevo fragmento que reemplaza al viejo. Debido a que usaremos transformación de modelos para implementar este DSTL, necesitamos tener definida su sintaxis abstracta a través de un metamodelo. La Figura 2 muestra el metamodelo de BPMTL.

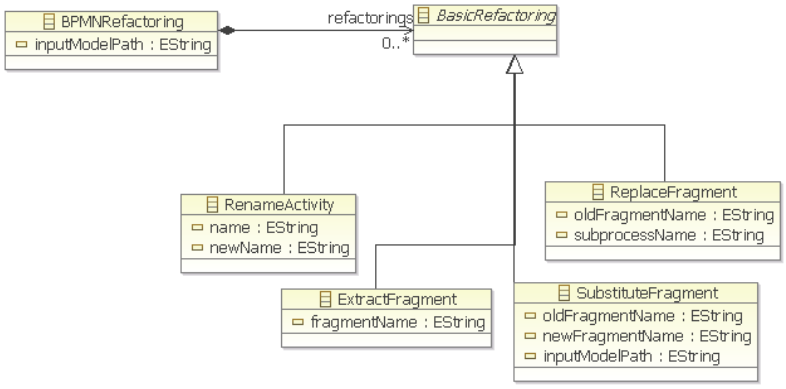


Figura 2. Metamodelo de BPMLT

Luego de tener definida la sintaxis de BPMTL, debemos definir su semántica. En la sección 3, al describir las refactorizaciones, hemos especificado su semántica usando

lenguaje natural. Estas definiciones transmiten en forma intuitiva el significado de cada constructor sintáctico, sin embargo se requiere más formalidad para garantizar la correcta implementación del DSTL.
Para formalizar la semántica de un lenguaje, es necesario contar con una función μ que al aplicarla a un elemento de dicho lenguaje, permita obtener el elemento correspondiente en el dominio semántico.
En particular, para establecer la semántica de BPMTL, utilizamos como dominio semántico un lenguaje para transformaciones ya existente como ATL. De esta manera contamos con la ventaja de que este lenguaje ya tiene su semántica bien definida y provee una maquinaria en funcionamiento para ejecutarlo.
Respecto a la función semántica μ, para nuestro contexto, está definida mediante una transformación escrita en MOFScript [28], un lenguaje de transformaciones modelo a texto. Esta transformación, dada una especificación en BPMTL, permite obtener el código ATL correspondiente (ver Figura 3).

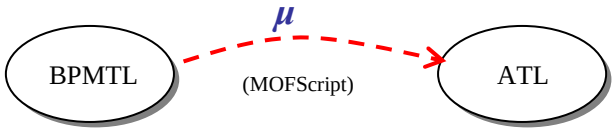


Figura 3. Semántica de BPMLT en términos de ATL

4.3 Implementación de BPMTL

Como ya mencionamos, la definición de la función semántica se realizó utilizando una transformación escrita en MOFScript, por la cual se obtiene el código ATL correspondiente a las refactorizaciones requeridas. Además, mediante una tarea *Ant* [30], se genera un archivo para poder ejecutar en secuencia el código ATL obtenido en el paso anterior. El siguiente código muestra la transformación en MOFScript que genera un archivo de código ATL por cada refactorización y finalmente un archivo *Ant* para ejecutar secuencialmente dichos archivos:

```
texttransformation ExampleTransformation (in
bpmtl:"http://refactoring/1.0") {
  var fileList : List;
  bpmtl.BPMNRefactoring::main () {

    self.refactorings->forEach(br:bpmtl.BasicRefactoring) {
        br.createATLFile();
   }
   self.createAntTask(self.inputModelPath, fileList);
  }

bpmtl.RenameActivity::createATLFile(){
   var fileName : String = "/refactorings/RenameActivity" + self.name +
"TO" + self.newName + ".atl";
   self.generateFile(fileName);
}
bpmtl.SubstituteFragment::createATLFile(){
   var fileName : String = "/refactorings/SubstituteFragment" +
self.oldFragmentName + "TO" + self.newFragmentName + ".atl";
   self.generateFile(fileName);
```

```
}
bpmtl.ReplaceFragment::createATLFile(){
   var fileName : String = "/refactorings/ReplaceFragment" +
self.oldFragmentName + "TO" + self.subprocessName + ".atl";
   self.generateFile(fileName);
}
bpmtl.ExtractFragment::createATLFile(){
   var fileName : String = "/refactorings/ExtractFragment" +
self.fragmentName + ".atl";
   self.generateFile(fileName);
}

bpmtl.BasicRefactoring::generateFile (fileName: String) {
   file (fileName);
   fileList.add(fileName);
   self.printCode();
}

bpmtl.RenameActivity::printCode(){
 println("-- @nsURI BPMN=http://BPMN/1.0");
 println("module RenameActivity;");
 println("create OUT : BPMN refining IN : BPMN;");
 println("helper def: activityToRename: BPMN!Activity =");
 println("BPMN!Task.allInstancesFrom('IN')-> select(a | a.name = '" +
self.name + "')");
 println("-> union (BPMN!Subprocess.allInstancesFrom('IN')-> select(a |
a.name = '"+ self.name +"'))");
 println("->first();\n");
 println("helper def: notExistsActivityNamed: Boolean =”);
......
println(" rule Task2Task {");
......

bpmtl.BPMNRefactoring::createAntTask (modelPath : String, listNames:
List){
 println(" <?xml version=\"1.0\"?>");
 println("<project default=\"transform\" basedir=\".\">");
        println("<target name=\"transform\">");
     println("    <atl.loadModel name= \"BPMN\"  + " metamodel=\"MOF\"
nsURI=\"http://bpmn/1.0\" />");
     println("    <atl.loadModel name=\"inputModel\" metamodel=\"BPMN\"
path=\"./model/inputModel.xmi\" />");
......
```

A continuación se muestra el archivo con el código ATL generado, en este caso para RenameActivity donde se especificó como *name* el string 'Compra' y como *newName* el string 'CompraMinorista'. Puede verse que usamos el mecanismo *refinement*, el cual permite escribir código solamente para la parte del modelo de entrada que es modificado por la transformación, mientras que el resto del modelo pasará al modelo de salida sin ninguna modificación.

```
-- @nsURI BPMN=http://BPMN/1.0

module RenameActivity;
create OUT : BPMN refining IN : BPMN;

helper def: activityToRename: BPMN!Activity =
```

```
        BPMN!Task.allInstancesFrom('IN')-> select(a | a.name = 'Compra')
        -> union (BPMN!Subprocess.allInstancesFrom('IN')-> select(a |
a.name = 'Compra'))
        ->first();

helper def: notExistsActivityNamed: Boolean =
        BPMN!Task.allInstancesFrom('IN')-> select(a | a.name =
'CompraMinorista')
        -> union (BPMN!Subprocess.allInstancesFrom('IN')-> select(a |
a.name = 'CompraMinorista'))
        ->first().oclIsUndefined();

rule Task2Task {
        from task: BPMN!Task in IN (task = thisModule.activityToRename
and thisModule.notExistsActivityNamed )

        to taskOut: BPMN!Task (
                name <- 'CompraMinorista',
                pool <- task.pool
        )
}

rule Subprocess2Subprocess {
        from sp: BPMN!Subprocess in IN (sp = thisModule.activityToRename
and thisModule.notExistsActivityNamed)

        to spOut: BPMN!Subprocess (
                name <- 'CompraMinorista',
                pool <- sp.pool
        )
}
```

La implementación completa de BPMTL puede obtenerse accediendo al sitio: http://www.lifia.info.unlp.edu.ar/eclipse/BPMTL/

5 Un Ejemplo

En esta sección mostramos la aplicabilidad del lenguaje BPMTL a través de un ejemplo descripto textualmente como sigue:

```
BPMNRefactoring {
inputModelPath "c:\eclipse\workspace\inputModel.xmi"
refactorings {
     SubstituteFragment { oldFragmentName "Fragment1"
          newFragmentName "Fragment2"  inputModelPath
          "c:\eclipse\workspace\auxModel.xmi" };

     SubstituteFragment { oldFragmentName "Fragment1"
          newFragmentName "Fragment2"  inputModelPath
          "c:\eclipse\workspace\auxModel.xmi" };

     ExtractFragment  { fragmentName "Fragment2" };

     ReplaceFragment { oldFragmentName "Fragment2"
                          subprocessName  "Fragment2" }
  }
}
```

La Figura 4 muestra un modelo de entrada muy simple al que se le aplicarán las refactorizaciones incluidas en la descripción textual de arriba. La figura 5 muestra una parte del mismo modelo pero ahora como instancia del metamodelo BPMN (ver Figura 1). El modelo consiste de un proceso formado por dos secuencias excluyentes.

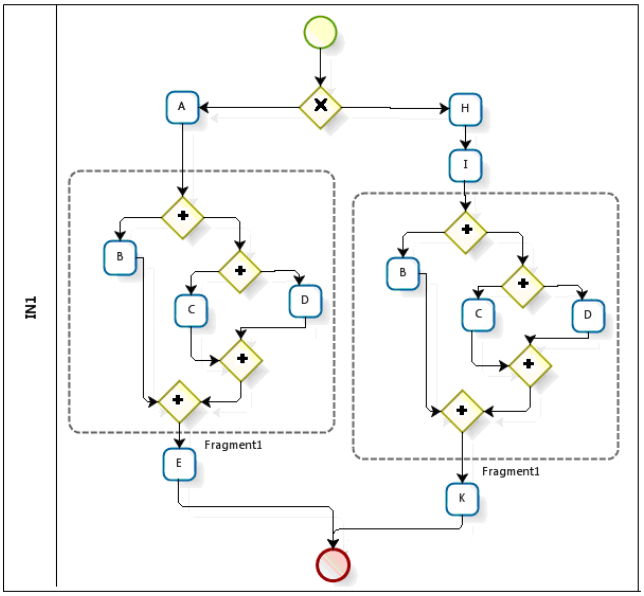


Figura 4. Modelo de entrada para aplicar los Refactorings.

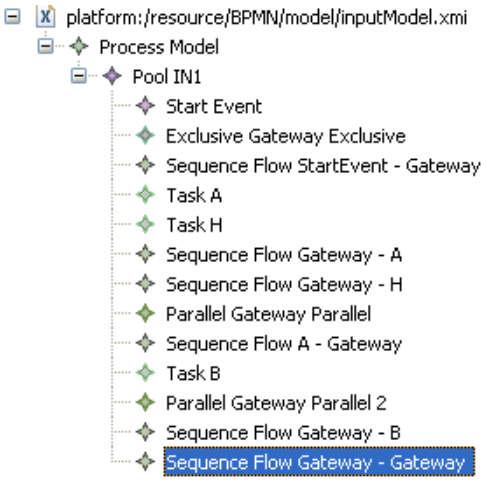


Figura 5. Modelo de entrada como instancia del metamodelo BPMN.

En ambas secuencias aparece repetido el elemento Fragment1. Como se explicó en la sección 3, en ciertos casos resulta útil reemplazar un elemento por otro. Por lo tanto es necesario contar con el elemento reemplazante, en este caso llamado Fragment2 (ver Figura 6, parte izquierda.). La salida correspondiente a aplicar dos veces *SubstituteFragment* puede verse en el modelo de la Figura 6 (parte derecha).

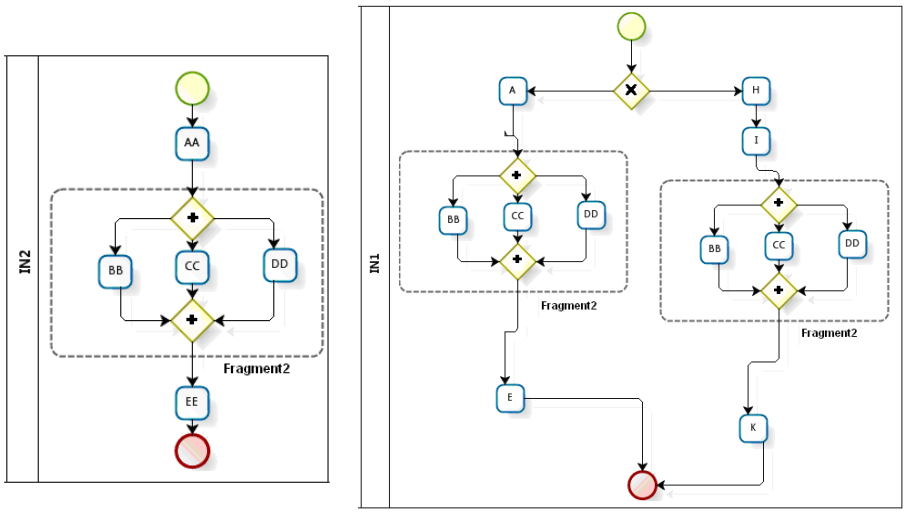


Figura 6. Modelo auxiliar con el grupo reemplazante (izq.). Modelo de salida luego de aplicar SubstituteFragment 2 veces (der.).

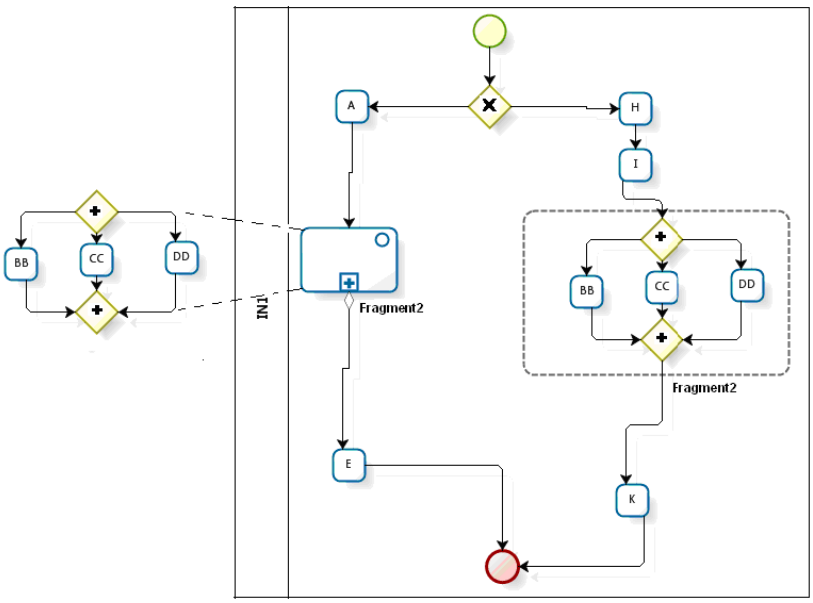


Figura 7. Modelo de salida luego de aplicar ExtractFragment

La siguiente refactorización que aplicamos al ejemplo es *ExtractFragment* sobre el grupo Fragment2. Como resultado se crea un nuevo subproceso con el contenido del grupo, como se muestra en la Figura 7. El contenido del subproceso puede verse al expandirlo.

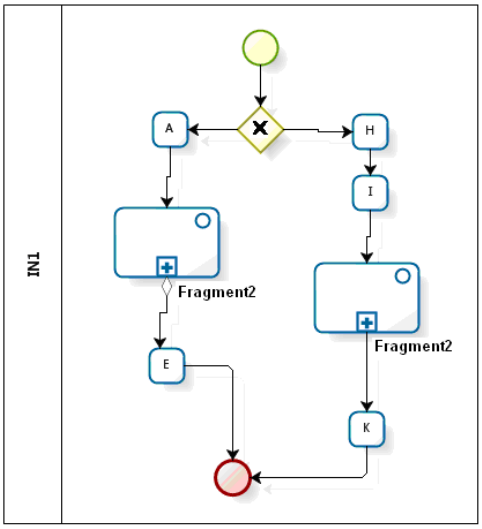


Figura 8. Modelo de salida luego de aplicar ReplaceFragment

Finalmente, la Figura 8 muestra el modelo de salida luego de aplicar *ReplaceFragment* sobre el grupo Fragment2 y el subproceso creado en el paso anterior.

6. Conclusiones y Trabajos Relacionados

En este artículo hemos propuesto un DSTL para refactorización de modelos de proceso del negocio, llamado *Business Process Modeling Transformation Language* (BPMTL). Este lenguaje provee reglas específicas sobre los elementos del dominio, en vez de reglas generales de transformación como sucede al tratar con lenguajes genéricos como QVT o ATL. Esto permite a los expertos en BPMN trabajar más cómodamente con constructores que representan conceptos conocidos. Consecuentemente, es esperable que estos expertos puedan escribir transformaciones (en este caso refactorizaciones) entendibles y reusables en menos tiempo y sin necesidad de conocer detalles técnicos tales como el metamodelo de BPMN. Además, definimos la semántica de BPMTL usando ATL, un lenguaje para transformaciones de propósito general. Al implementar BPMTL utilizando un lenguaje para transformaciones ya existente como ATL contamos con la ventaja de que este lenguaje ya tiene su semántica bien definida y provee una maquinaria en funcionamiento para ejecutarlo. Hemos incluido también el desarrollo de un simple ejemplo de uso de nuestro lenguaje, mostrando la aplicación de las operaciones.

Respecto a los trabajos relacionados, existen pocas propuestas de DSTLs para otros dominios. En [24] el dominio es el modelado orientado a aspectos; en [25] se considera también este domino y el de ontologías de herramientas de modelado. En ambas propuestas, los DSTLs fueron implementados en el lenguaje C++. En [23] los autores proponen un lenguaje imperativo para expresar transformaciones de BPEL a UML. Estas tres propuestas mencionadas no aprovechan los beneficios de implementar el DSTL utilizando un lenguaje de transformación ya existente como planteamos en este trabajo.
Por otro lado, en [22] se presenta un DSTL para el dominio de líneas de producto de software que ayuda a los expertos a administrar familias de modelos y en [27] se propone un DSTL para transformar modelos relacionales. Estos dos trabajos en forma similar a la nuestra, pero respecto a otros dominios de aplicación, utilizan lenguajes de transformación ya conocidos para implementar el nuevo lenguaje (ATC [26] y ATL respectivamente).
Como dijimos, el alcance de BPMTL se limita a la definición de operaciones aplicables a modelos BPMN. La verificación de la preservación semántica de estos modelos queda fuera de su alcance. Por esta razón, actualmente estamos enfocados en integrar BPMTL a un editor BPMN que sea capaz de verificar la preservación de invariantes del modelo previamente a la aplicación de las refactorizaciones. Otra posible línea de trabajo futuro es la extensión de BPMTL. Incluiremos operaciones más complejas para variantes de proceso y evolución de modelos de proceso.

Referencias

1. Stahl, T., Völter, M. Model-Driven Software Development. John Wiley & Sons, Ltd. (2006)
2. C. Pons, R. Giandini, G. Pérez. “Desarrollo de Software Dirigido por Modelos. Conceptos teóricos y su aplicación práctica”. 1er. Edición. EDULP & McGraw-Hill Educación. (2010). ISBN: 978-950-34-0630-4
3. Kleppe, A. and Warmer J., and Bast, W. MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc., USA. (2003)
4. D.C. Schmidt. Model-Driven Engineering. *IEEE Computer*, 39(2):41-47, 2006.
5. Unified Modeling Language (UML), version 2.2 OMG, http://www.omg.org
6. Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. ACM Computing Surveys, 37(4):316–344, 2005.
7. Richard C. Gronback. Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley Professional. ISBN: 0-321-53407-7, 2009.
8. Weske Mathias, “Business Process Management: Concepts, Languages, Architectures”. Springer, Pag 3-67. ISBN 978-3-540-73521-2. 2008
9. Object Management Group (OMG), http://www.omg.org
10. Business Process Modeling Notation (BPMN) Version 1.2 OMG, http://www.omg.org/spec/BPMN/1.2
11. Martin Owen, Jog Raj. “BPMN and Business Process Management Introduction to the New Business Process Modeling Standard”. Popkin Software. 2003
12. Czarnecki, Helsen. Feature-based survey of model transformation approaches. IBM System Journal, V.45, N3, 2006.

13. MOF QVT Adopted Specification 2.0. OMG Adopted Specification. November 2005. http://www.omg.org
14. ATLAS team: ATLAS MegaModel Management (AM3) Home page, http://www.eclipse.org/gmt/am3/. (2006)
15. Jouault, F., Kurtev, I.: Transforming Models with ATL. In: Satellite Events at the MoDELS 2005 Conference. Volume 3844 of Lecture Notes in Computer Science, Springer-Verlag (2006) 128–138
16. Sánchez Cuadrado, J., García Molina, J. and Menarguez Tortosa, M. : RubyTL: A Practical, Extensible Transformation Language. In proceedings of ECMDA 4066. Springer. (2006)
17. Stephen A.White, "Introduction to BPMN". IBM Corporation. http://www.bpmn.org, 2004
18. OMG/MOF Meta Object Facility (MOF) 2.0. OMG Adopted Specification. October 2003. http://www.omg.org
19. OMG. The Object Constraint Language (OCL) Specification–Version 2.0, for UML 2.0, http://www.omg.org, May 2006.
20. B. Weber and M. Reichert, Refactoring Process Models in Large Process Repositories. Bellahs`ene and L´eonard (Eds.): CAiSE 2008, LNCS 5074, pp. 124–139, 2008. Springer-Verlag Berlin Heidelberg 2008
21. Fowler, M.: Refactoring-Improving the Design of Existing Code. Addison-Wesley, Reading (2000)
22. O. Avila-García, A. Estévez García, E. Sánchez Rebull. Using Software Product Lines to Manage Model Families in Model-Driven Engineering. In SAC '07, March 11-15, 2007, Seoul, Korea. ACM 1-59593-480-4/07/0003
23. T. Reiter, E. Kapsammer, W. Retschitzegger,W. Schwinger,M. Stumptner. A generator framework for domain-specific model transformation languages. In ICEIS: The Eight Int. Conf. On Enterprise Information Systems, May 2006.
24. J. Gray, T. Bapty, S. Neema, D. C. Schmidt, A. Gokhale, and B. Natarajan. An approach for supporting aspect-oriented domain modeling. In GPCE2003: Proceedings of The 2nd Int. Conf. on Generative Programming and Component Engineering, volume 2830 of LNCS, pages 151-168.Springer-Verlag, Sep 2003.
25. J. Gray and G. Karsai. An examination of DSLs for concisely representing model traversals and transformations. Proceedings of HICSS 2003, Jan 2003.
26. A. Estévez, J. Padrón, V. Sánchez, and J. L. Roda. ATC: A low-level model transformation language. In Proceedings of the 2nd Int.Workshop MDEIS, 2006.
27. J. Irazabal, C. Pons, C. Neil. Model transformation as a mechanism for the implementation of domain specific transformation languages.E-Journal Sadio. To be published in 2010.
28. Jon Oldevik. MOFScript User Guide. Version 0.6 (MOFScript v 1.1.11), 2006.
29. Tutorial BPMN. http://wiki.eclipse.org/GMF_Tutorial_BPMN
30. Apache Ant Project. Ant 1.8.1. http://ant.apache.org/, May 2010.