Aplicando herramientas MDE en la definición de un lenguaje especifico de dominio para la gestión de modelos

Gabriela Pérez [1], **Jerónimo Irazábal** [1,2], **Claudia Pons** [1,2] **y Roxana Giandini** [1]
[1]LIFIA, Facultad de Informática, Universidad Nacional de La Plata
[2]CONICET, Consejo Nacional de Investigaciones Científicas y Técnicas
Buenos Aires, Argentina
{gperez , jirazabal, cpons, giandini}@lifia.info.unlp.edu.ar

Resumen. En la ingeniería de desarrollo de software dirigida por modelos (MDE), los lenguajes de modelado juegan un rol central. Abarcan desde los más genéricos como UML, hasta los más particulares, llamados específicos del dominio (DMSL). Estos lenguajes serán utilizados para crear modelos y debe acompañarlos a lo largo de su ciclo de vida y su evolución.

En este trabajo proponemos un lenguaje específico de dominio para la gestión de modelos, que facilite la tarea del usuario, desarrollado con técnicas y herramientas usadas en el paradigma MDE.

Palabras clave: Desarrollo Dirigido por Modelos, DSL, lenguaje especifico de dominio. Lenguaje especifico de dominio para la gestión de modelos.

1. Introducción

El modelado es importante para poder tratar la complejidad de los sistemas durante los procesos de desarrollo y durante su posterior mantenimiento. La ingeniería dirigida por modelos [1-3], [5,22] propone un proceso de desarrollo de software en el cual los elementos principales son los modelos. Estos modelos permiten a los ingenieros capturar de forma precisa aspectos de un sistema desde una perspectiva dada y en un nivel apropiado de abstracción y a partir de estos, automatizar el desarrollo de un sistema.

Los modelos se pueden expresar utilizando diferentes lenguajes: lenguajes de modelado de propósito general (GPMLs), como UML, o lenguajes específicos de dominio de modelado (DSMLs) [4], tales como por ejemplo el Business Process Modeling Notation (BPMN), para modelado de procesos de negocio. Los DSMLs son lenguajes de alto nivel, diseñados para tareas particulares. Permiten especificar una solución usando directamente conceptos del dominio del problema. Como los conceptos del lenguaje son utilizados dentro de la organización, se reduce considerablemente el tiempo de aprendizaje del mismo. Asimismo los expertos del dominio pueden comprender, validar, modificar y a menudo desarrollar programas en el DSL. Tienen una sintaxis simple (es decir, pocas construcciones enfocadas en un dominio en particular), pero su semántica es mucho más compleja (ya que toda la semántica del dominio particular está embebida en el lenguaje). Al estar restringidos a un dominio particu-

lar, son más eficientes en la generación de código, permitiendo notables mejoras en la productividad, la interoperabilidad, mantenibilidad y calidad de los productos generados. Según las experiencias, si se utilizan DSMLs se puede simplificar el desarrollo de sistemas de software complejos, proporcionando abstracciones específicas de dominio permitiendo especificar los modelos de manera precisa, y a la vez simple y concisa.

Los DSMLs incrementan el poder expresivo pero a la vez incrementan la complejidad: para el usuario final del lenguaje, construir un modelo en un nuevo DSML, y poder realizar los cambios necesarios para mantenerlo significa crear e interconectar elementos de bajo nivel propios del lenguaje, lo cual es una tarea propensa a errores. Estas dificultades podrían eliminarse si se proveyera al usuario de un lenguaje que permita interactuar con los modelos específicos de dominio de una forma más amigable. El usuario no será el responsable de generar elementos y relacionarlos consistentemente, sino que podrá delegar esta responsabilidad al lenguaje para gestión de modelos utilizando las operaciones ofrecidas por éste.

Contar con un lenguaje específico para la gestión de modelos, o DSMML [28, 32](por sus siglas en ingles Domain Specific Model Management Language), separado del lenguaje específico de dominio permitirá:

- Que el lenguaje de gestión y el lenguaje de modelado evolucionen independientemente. Se podrían mejorar algunos aspectos del lenguaje de modelado específico de dominio sin alterar la interfaz del lenguaje de gestión. O incluso cambiarlo radicalmente. De forma análoga, se podrían modificar, o extender las operaciones del lenguaje de gestión sin afectar al lenguaje de modelado específico de dominio.
- Contar con distintos lenguajes de gestión para un mismo lenguaje específico de modelado. Se podrían desarrollar distintos lenguajes con operaciones distintas que reflejen las responsabilidades de cada usuario. Por ejemplo, tener un lenguaje sólo de consulta y otro que permita operaciones más críticas, de edición o borrado.
- En el caso de usar un lenguaje específico de dominio estándar, acercará, aun más, ese estándar a los usuarios: definir un lenguaje de gestión de modelos sobre un lenguaje estándar, como BPMN, posibilitará al usuario una interacción más amigable el lenguaje.

La figura 1 muestra cómo la definición de un lenguaje específico para la gestión de modelos puede ayudar a los usuarios con el uso del lenguaje. En la parte superior puede verse un usuario que todavía tiene algunas dudas respecto al uso del lenguaje específico de dominio (DSML), mientras que en la parte inferior, pueden verse dos usuarios que disponen de dos lenguajes específicos para la gestión de modelos (DSMMLs), no necesariamente iguales, inmediatamente pueden comenzar a utilizar el lenguaje.

El problema a enfrentar entonces, es implementar este lenguaje específico para la gestión de los modelos. Actualmente existen frameworks muy poderosos para la creación de lenguajes específicos de dominio, como pueden verse en [9-11]

En el presente trabajo se describe una propuesta para definir lenguajes específicos para la gestión de modelos y se analiza una forma novedosa de definir su semántica. Nuestra propuesta consiste en usar herramientas MDE para la implementación de

estos lenguajes, lo que mejora su modularidad y reuso. La organización del artículo es la siguiente: en la sección 2 se presentan las principales características de nuestra propuesta. La sección 3 presenta un enfoque general para la definición de un lenguaje de gestión de modelos sobre un dominio existente, como por ejemplo el dominio de procesos de negocio. La sección 4 muestra un ejemplo en dicho lenguaje. La sección 5 compara nuestro enfoque con otros trabajos de investigación relacionados. Y finalmente presentamos conclusiones y líneas de trabajo futuro.

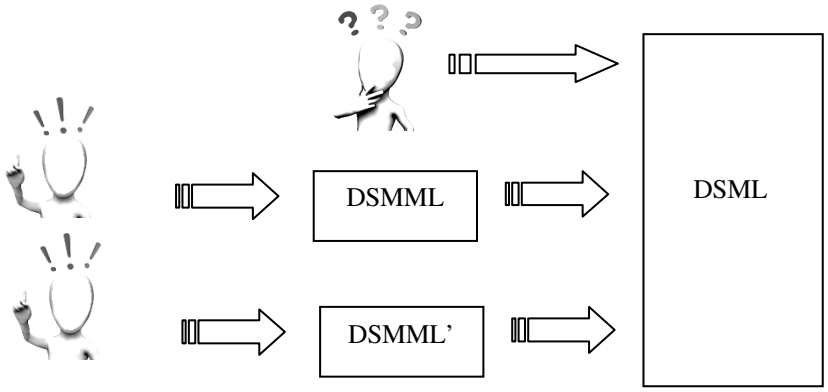

Figura 1: El DSMML facilita al usuario el uso del DSML.

2. DSMML: esquema de su implementación

Todo lenguaje consiste de dos elementos principales: una sintaxis y la semántica. La sintaxis se refiere a la notación del lenguaje, sin tener en cuenta el significado de cada uno de los símbolos. Describe su vocabulario, es decir, los conceptos provistos por el lenguaje y como estos conceptos se pueden combinar para crear modelos.

Por otro lado, la semántica, asigna un significado sin ambigüedad a cada elemento sintáctico en el lenguaje. En las ciencias de la computación, cualquier lenguaje debe consistir en un conjunto de reglas describiendo que expresiones son sintácticamente bien formadas, junto con reglas formales describiendo su semántica.

En la teoría de lenguajes de programación, la semántica es el campo que concierne al estudio riguroso matemático del significado de los lenguajes. La semántica formal de un lenguaje esta dada por la estructura matemática que describe todas las posibles computaciones expresadas por el lenguaje. Existen algunas propuestas para definir la semántica formal, entre ellas la semántica dada por la traducción a otro lenguaje. La ventaja de esta propuesta es que si existe una maquinaria que ejecute el lenguaje destino, es posible obtener una semántica ejecutable para el lenguaje por medio de la traducción. La semántica formal permite una clara comprensión del significado del lenguaje, y la posibilidad de verificar ciertas propiedades de los programas, como correctitud, terminación, performance, equivalencia entre programas, etc.

Técnicamente, una definición semántica para un lenguaje consiste de dos partes: un dominio y una función de mapeo, denotado por μ, que hace corresponder los elementos de la sintaxis al dominio de la semántica.

En particular, nuestra propuesta consiste en utilizar un lenguaje de transformaciones de propósito general, como dominio semántico. En este caso utilizamos ATL [6-7].

Como se muestra en la figura 2, la función μ está definida mediante un lenguaje de transformación de modelo a texto (en este caso utilizamos MOFScript [16]). Esta función toma un programa escrito en nuestro DSMML como entrada, y generará un programa escrito en ATL como salida.

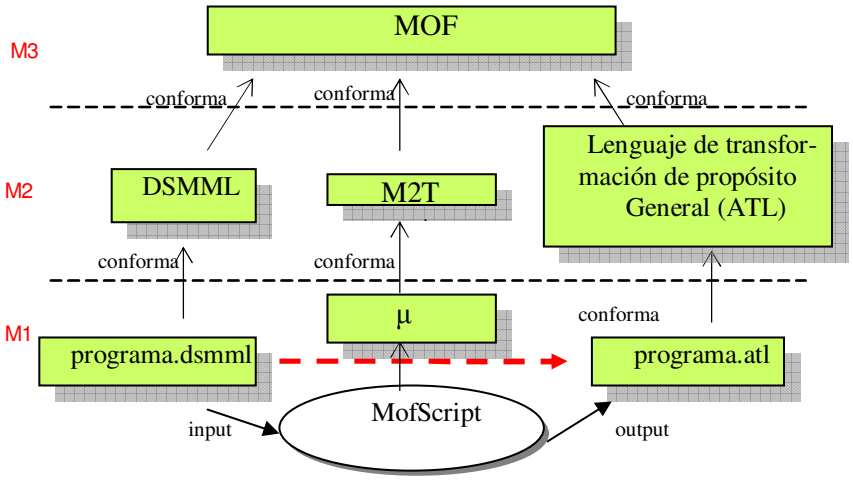

Figura 2: Esquema de transformación general para la definición de la función semántica μ

La ventaja de esta técnica es que el lenguaje de transformación es conocido, tiene una semántica bien definida y además proporciona un entorno de ejecución aceptable. Por lo tanto, la semántica del nuevo lenguaje también será ejecutable.

En las secciones siguientes se presenta un ejemplo de DSMML que incluye la definición de la sintaxis y la semántica según el método propuesto.

3. Un lenguaje para gestión de Modelos de Proceso de Negocio.

Se llama gestión o administración de procesos de negocio (Business Process Management o BPM según sus siglas en inglés) a la metodología basada en los procesos de negocio. Su objetivo es mejorar el desempeño de la organización a través de la gestión de sus procesos. Para esto, esos procesos se deben diseñar, modelar, organizar, documentar y optimizar de manera continua. La propuesta de BPM [23] ha gana-

do considerable atención, tanto desde las comunidades de gestión empresarial como desde las comunidades de informática. En este ámbito, el lenguaje de modelado estándar definido por la OMG [24-25] es el lenguaje de notación para procesos de negocios (Bussiness Process Modeling Notation o BPMN según sus siglas en ingles) se ha convertido en un lenguaje muy popular. Ha sido desarrollado para proporcionar una notación estándar a los usuarios de negocios, de forma similar a como UML provee conceptos estandarizados para el modelado de software.

3.1. Motivación para la creación de un lenguaje de gestión para procesos de negocio

Como se mencionó en la sección anterior, el lenguaje BPMN fue desarrollado para proporcionar una notación estándar de conceptos para el dominio BPM. Estos conceptos, tales como Proceso, Actividad o Tarea ya eran utilizados por los usuarios de negocios, por lo que se reduce considerablemente el tiempo de aprendizaje del lenguaje.

De la misma manera que sucede en otros campos del modelado de software, un modelo BPMN requiere modificaciones durante su ciclo de vida. Estos usuarios, a pesar de conocer los conceptos del dominio, no conocen en profundidad detalles de implementación de este nuevo lenguaje. Estos detalles van desde donde guardar las instancias de los conceptos hasta que consecuencias tiene el borrado de un elemento. Por lo tanto, para llevar a cabo esas modificaciones, hay que tener un conocimiento detallado de su metamodelo, así como de las relaciones entre los elementos. Realizar estos cambios a mano, directamente sobre el modelo, pone en riesgo la integridad del mismo, ya es una forma propensa a la introducción de errores. Para que esto no ocurra, seria deseable contar con un lenguaje para gestión que permita ocultar los detalles de implementación de las operaciones. Este lenguaje deberá proveer operaciones específicas para la gestión de modelos BPMN. Actualmente existe una gran variedad de herramientas para editar visualmente diagramas BPMN. Estas herramientas están mayormente enfocadas en la creación, edición y borrado de los elementos del modelo. Sin embargo, seria útil contar con operaciones mas complejas que faciliten y acompañen la evolución de estos modelos. Frecuentemente, los procesos de negocios suelen ser reemplazados por otros ya descriptos. Con un lenguaje apropiado de gestión, estas tareas se simplifican debido a que los detalles de las modificaciones necesarias para llevar a cabo un cambio quedan ocultos para el usuario.

3.2. BPMML: un DSMML para gestionar modelos de proceso de negocios.

En esta sección se muestra la definición de un DSMML para realizar modificaciones en los modelos de procesos de negocio. Este lenguaje se llama Lenguaje para la Gestión de Modelos de Procesos de Negocio (Business Process Modeling Management o BPMML según sus siglas en ingles). En el proceso de diseño de BPMML hemos considerado las operaciones de gestión que se aplican con más frecuencia en los modelos de procesos de negocio. En particular, mostramos refactorizaciones que son comúnmente utilizadas [26-27], tales como las siguientes.

- La operación sustituir_Fragmento (SubstituteFragment) permite al diseñador del proceso reemplazar un fragmento por otro, cambiando las relaciones entre los elementos y el primer fragmento por relaciones con el segundo fragmento.
- La operación Extract_Fragment que permite extraer un fragmento generando un nuevo proceso, con el fin de eliminar la redundancia. Si existieran relaciones entre otros elementos y el fragmento, se establecerán unas relaciones análogas entre los mismos elementos y el proceso generado.
- La operación Replace_Fragment_by_Reference que sustituye a una actividad compleja por una referencia.
- Además de estas operaciones, se han definido las operaciones adicionales que son útiles en el dominio de la modelización de procesos comerciales, tales como la importación de un proceso, intercambiar de posición a dos procesos (permitiendo el último que se produzca antes que el primero), romper la conexión entre dos procesos, establecer una conexión entre dos procesos, entre otras.

El siguiente código muestra la definición de la sintaxis concreta escrita en EMF-Text[33] del lenguaje para gestión de modelos BPMML.

```
SYNTAXDEF bpmml
FOR <http://bpmml/1.0> <bpmml.genmodel>
START BPMML
RULES {
       BPMML ::= "BPMML" "open" inputModelPath['"','"']
              "{" ( manipulations : Manipulation)* "}";

       ExtractGroup ::= "Extract" group[] ";";
       RenameActivity ::= "Rename" name[] "to" newName[]";";
       ReplaceSubProcess::= "Replace" oldSubProcess[] "by"
newSubProcess[]";";
       SubstituteSubProcess ::= "Substitute" oldSubProcess[]
"by" newSubProcess[] "located in" modelPath[]";";
       ImportPool::= "Import pool" pool[] "located in"
modelPath[]";";
       ImportSubProcess::= "Import subprocess" subProcess[] "to"
targetPool[] "located in" modelPath[]";";
       DeletePool::= "Delete pool" pool[]";";
       DeleteElementsFromPool::= "Delete elements from"
pool[]";";
       SwapElements::="Swap" source[]"with" target[]";";
       CreateActivityBetween::= "Create Activity" named[]
"between" source[] "and" target[]";";
       SplitFlow::= "Split Flow add" element[] "between"
source[] "and" target[]";";
       AllActivitiesFirstUpper::= "Format name to all
activities" ";";
  }
```

3.3. Implementación del lenguaje BPMML

En esta sección se presenta la implementación de nuestro lenguaje DSMML. En la figura 3 se muestra el enfoque por traducción para la definición de la semántica del lenguaje de dominio específico de gestión de modelos BPMN.

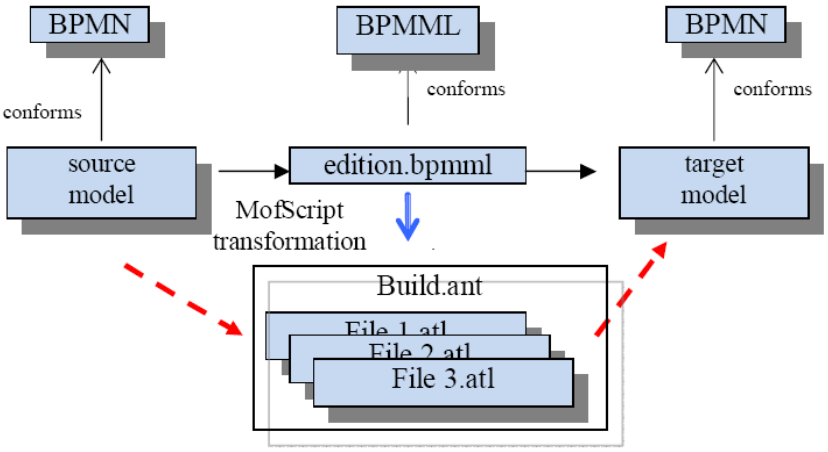

Figura 3: Esquema de transformación para la definición de la función semántica μ realizado con ATL

Como ya se mencionó, las sentencias escritas en el lenguaje de gestión de dominio específico BPMML se traducen a sentencias de un lenguaje de transformación de propósito general, como ATL. Las reglas de traducción están escritas en MOFScript, un lenguaje de transformación de modelo a texto. El conjunto de archivos ATL son la interpretación semántica de nuestras sentencias en BPMML.

Debido a restricciones de ATL, es necesario generar por cada instrucción del programa BPMML un archivo ATL separado. Estos archivos necesitarán entonces ser coordinados para que se puedan ejecutar adecuadamente. Para esto utilizamos la herramienta Apache Ant ya que permite sistematizar la ejecución de los archivos ATL con los correspondientes modelos. En la figura 3 puede verse los archivos generados por el programa MOFScript, que incluyen la secuencia de archivos ATL y el script build.ant que los organizará para su ejecución.

El siguiente código muestra la transformación MOFScript, separada en dos módulos. Por un lado, el modulo principal, que tomará cada operación del programa de gestión y las traducirá a su respectivo código ATL.

```
import ("BPMML_Library.m2t");
 texttransformation BPMML_Semantic (in pmml:"http://bpmml/1.0"){
  bpmml.BPMML::main () {
   self.operations->forEach(operation:bpmml.Operation)
      operation.createATLFile(operation.
                              getFilename(number));
  }
 self.createAntTask(self.inputModelPath, fileList, antLaunchs);
}
```

El siguiente código muestra el segundo modulo, que traduce cada operación a código ATL.

```
texttransformation SemanticaBPMNTL_Library (in
bpmml:"http://bpmml/1.0")
{
bpmml.RenameActivity::printCode(){
println("-- @nsURI BPMN=http://stp.eclipse.org/bpmn");
println("module RenameActivity;");
println("create OUT : BPMN refining IN : BPMN; \n");
….

bpmml.SplitFlow::printCode(){}
bpmml.SubstituteSubProcess::printCode(){…}
…
```

El siguiente ejemplo muestra el código ATL que se generó a partir de la operación Rename Activity, donde se renombra una actividad llamada 'Start' con el nombre 'Experience an unexpected behavior'. Se puede ver que se utiliza el mecanismo de refinamiento, lo que nos permite escribir código sólo los elementos que serán afectados por la transformación, mientras que el resto del modelo se mantiene sin cambios.

```
  -- @nsURI BPMN=http://stp.eclipse.org/bpmn
  module RenameActivity;
  create OUT : BPMN refining IN : BPMN;

   helper def: activityToRename: BPMN!Activity =
     BPMN!Activity.allInstancesFrom('IN')-> select(a | a.name =
'Start') ->first();

   helper def: notExistsActivityNamed: Boolean =
         BPMN!Activity.allInstancesFrom('IN')->    select(a    |
      a.name = 'Experience an unexpected behavior')
      ->first().oclIsUndefined();

   rule Activity2Activity {
         from activity: BPMN!Activity in IN (activity =
               thisModule.activityToRename     and     thisModu-
            le.notExistsActivityNamed)
      to activityOut: BPMN!Activity (
            name <- 'Experience an unexpected behavior',
            graph <- activity.graph
      )
   }
  }
```

4. Un ejemplo de uso

En esta sección se muestra la aplicabilidad del lenguaje de gestión de modelos BPMML mediante un ejemplo de uso. La figure 4 muestra el proceso para reportar errores o bugs con Bugzilla. Bugzilla es una herramienta web que permite reportar bugs, asignar la resolución de bugs a un desarrollador determinado, mantener la información del progreso en la corrección, etc. Este ejemplo fue presentado en Eclipse-Con 2008.

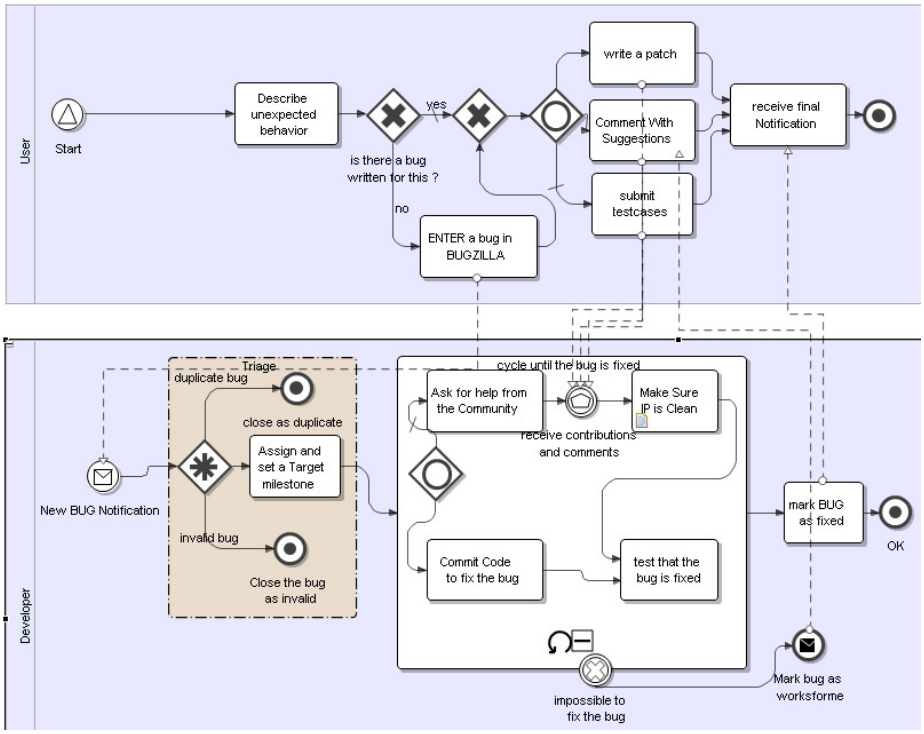

Figure 4. Modelo BPMN para reportar errores en Bugzilla.

Trabajando sobre este bosquejo, descubrimos que pueden aplicarse ciertas mejoras, como por ejemplo:

- Cambiar el nombre del evento de entrada por otro nombre mas significativo, por ejemplo “Experience an unexpected behavior”.
- Se evidencia la necesidad de agregar al proceso una actividad de Investigación, con el fin de asegurarnos que esta tarea se haga apropiadamente. Como este es un proceso común, ya desarrollado por otros procesos, podemos importarlo e intercalarlo entre el evento de entrada y la tarea de descripción del comportamiento no esperado.
- Además, podríamos uniformar los nombres de las actividades, de modo que todos estén escritos en minúscula y comiencen con la primera letra mayúscula.

Estos cambios están especificados mediante el siguiente código escrito en BPMML

```
BPMML open "bugzilla.bpmn" {
 Rename "Start" to "Experience an unexpected behavior";
 Import subprocess "Investigate" to "User" located in
"otherProject.bpmn";
 Split Flow add "Investigate" between "Experience an
unexpected behavior" and "Describe unexpected behavior";
 Format name to all activities;
}
```

La figura 5 muestra el modelo de salida, luego de haberle aplicado las modificaciones requeridas. Para esto, se ejecutaron los archivos ATL coordinados mediante el archivo ant creado para tal fin. El modelo de salida puede visualizarse en un editor gráfico para bpmn.

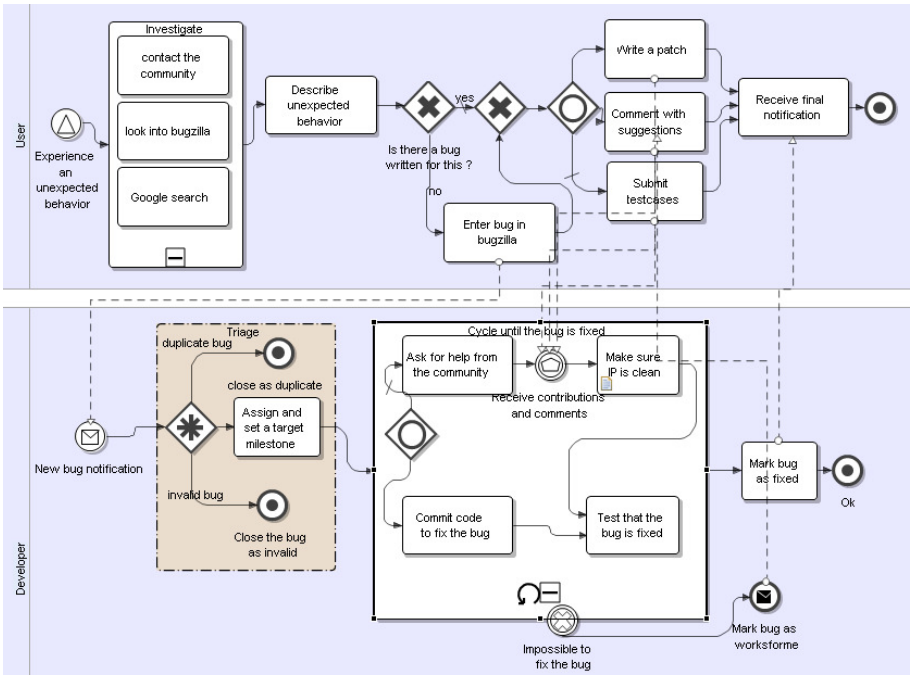

Figura 5. Esquema de implementación del DSMML usando el enfoque por traducción.

En este ejemplo se hace evidente la ventaja de contar con un lenguaje específico de gestión de modelos BPMN, ya que facilita el manejo de los mismos. Las modificaciones mencionadas anteriormente se podrían haber realizado mediante un editor gráfico, pero sería una tarea más laboriosa, y propensa a errores.

5. Trabajos relacionados

Los trabajos relacionados con esta propuesta que se han analizado están vinculados con la creación y uso de lenguajes de modelado específicos de dominio.

Por un lado, Kolovos [30-31] propone EuGENia, un lenguaje de anotaciones que pretende bajar la barrera de aprendizaje de GMF y de esta manera hacerlo mas accesible a los usuarios. GMF es un proyecto que permite generar editores gráficos en Eclipse. GMF precisa de un metamodelo y en base a éste requiere definir tres modelos, uno para la definición gráfica, GMFGraph, en el cual se definen formas, conectores, etiquetas, etc. El segundo para la definición de herramientas, GMFTool, en el cual se especifica qué ítems de creación de elementos estarán visibles en la paleta del editor. Y por el último un modelo que permite relacionar los elementos del metamodelo, del modelo gráfico y de herramientas. Estos tres modelos, mas el metamodelo, serán la base para generar un editor gráfico totalmente funcional. EuGENia propone una alternativa a la construcción de estos tres modelos requeridos por GMF: hacer anotaciones directamente sobre el metamodelo y a partir de ellas crear esos modelos automáticamente. Esta propuesta de alguna manera es parecida a la nuestra ya que el nuevo lenguaje intenta hacer el uso de un lenguaje existente, en este caso GMF, más amigable al usuario. A diferencia de nuestra propuesta, este nuevo lenguaje no se define para gestionar los modelos sino para generar los modelos necesarios para construir un editor gráfico automáticamente con GMF. En esta propuesta se plantea crear uno o varios lenguajes que serán usados a lo largo de todo el ciclo de vida de los modelos a los que se esta gestionando.

Por otro lado, Kermeta [29] es un lenguaje de metamodelado que permite describir estructura y comportamiento de los modelos. Esta diseñado para ser totalmente compatible con el lenguaje EMOF y provee un lenguaje de acciones para especificar el comportamiento de los modelos. Como diferencia nuestra propuesta plantea una separación entre el metamodelo y la especificación de su comportamiento, otorgándoles de esta manera mayor flexibilidad, para por un lado permitir definir y utilizar varios lenguajes de gestión y por otro lado dándole libertad para que ambos lenguajes puedan evolucionar independientemente.

Nuestro enfoque puede ser visto como una técnica para la abstracción y la modularización en la cual cada operación de alto nivel, escrita en el DSMML, esta asociada con una o varias operaciones de bajo nivel, escritas en un lenguaje de propósito general. Los usuarios no necesitan conocer los detalles de las operaciones de bajo nivel. En este sentido, los trabajos que proponen técnicas para construir transformaciones complejas están relacionados con nuestra propuesta. En esta categoría, podemos mencionar la técnica de composición descripta por A. Kleppe in [17], la propuesta de Model Bus [18], el Framework de modelado para transformaciones compuestas definido por Jon Oldevik [19] y la técnica de superposición de módulos [20], entre otros. En contraste con estos trabajos, nuestra propuesta genera una especificación de la transformación compuesta de una manera más simple, sin agregar ninguna maquinaria de composición explícita.

Si analizamos lenguajes que abstraen a partir de otros lenguajes abstractos podemos mencionar el lenguaje MetaBorg [21], una propuesta basada en transformación para la definición de un DSL textual embebido basado en el Framework Stratego. Similarmente a lo que ocurre en nuestro trabajo, la propuesta MetaBorg define nuevos conceptos (comparables con nuestra noción de lenguaje abstracto) mapeando estos conceptos a expansiones del lenguaje huésped (comparable a nuestra noción de lenguaje concreto.

Por último, podemos mencionar al framework AMMA [12], que permite definir la sintaxis concreta, abstracta y la semántica de los DSLs. En [13-15] se muestran escenarios donde se utilizó el framework AMMA para definir la semántica de un DSL en términos de otro lenguaje, o en términos de máquinas de estado abstractas. Nuestra propuesta es similar, pero presentamos como alternativa novedosa el definir la semántica, por medio de una transformación modelo a texto, que hace corresponder los elementos de la sintaxis al dominio de los lenguajes de transformaciones de propósito general.

6. Conclusiones

En este trabajo hemos explicado el concepto de lenguaje específico de dominio para la gestión de modelos, es decir, lenguajes de gestión para modelos enfocados en un dominio específico. En contraste con los lenguajes de gestión de modelos de propósito general, como EOL [8] y ATL, la sintaxis y la semántica del lenguaje de gestión específico se relacionan directamente con un dominio, por lo que se facilita la escritura y comprensión del lenguaje.

Contar con un lenguaje específico de dominio para la gestión de modelos presenta las siguientes ventajas:

- Permitirá a los usuarios interactuar con los modelos específicos de dominio de una forma más amigable. Los cambios en los modelos son menos propensos a errores, ya que usuario no será el responsable de generar elementos y relacionarlos consistentemente, sino que podrá delegar esta responsabilidad al lenguaje para gestión de modelos utilizando las operaciones ofrecidas por éste. Los expertos del dominio se sentirán más cómodos usando un lenguaje de gestión con las construcciones que reflejan conceptos bien conocidos.
- Quedan separados los roles de diseñador del lenguaje DSMML y programador de dicho lenguaje. Los programadores no necesitan conocer las particularidades de los lenguajes de transformación de propósito general, ya que esta información esta encapsulada en las operaciones ofrecidas por el DSMML.
- Permitirá que el lenguaje de gestión y el lenguaje de modelado evolucionen independientemente. Se pueden tener distintos lenguajes de gestión para un mismo lenguaje específico de modelado. Por ejemplo, tener un lenguaje solo de consulta y otro que permita operaciones más críticas.
- Permite acercar, aun más, un estándar a los usuarios: definir un lenguaje de gestión de modelos sobre un lenguaje estándar, como BPMN, permite acercar al usuario con ese estándar, haciendo más amena su interacción con el lenguaje.

Además, proponemos que la semántica del lenguaje DSMML esté definida usando un lenguaje de transformación de propósito general. Presentamos una propuesta donde una instancia de DSMML no es compilada en un código fuente, sino que es transformada a un lenguaje de gestión de modelos de propósito general. En los casos de estudio hemos utilizado los lenguajes ATL y EOL. Este hecho provee varias ventajas: la semántica del lenguaje está formalmente descripta; es ejecutable La semántica es comprensible debido a que esta escrita en un lenguaje bien conocido, puede ser fácilmente modificada agregando nuevas reglas de transformación, o cambiando radicalmente el lenguaje destino. Aunque esta transformación puede ser considerada como un compilador, las habilidades necesarias para crearla son menores que si creáramos un compilador de código fuente.

Como casos de uso se definieron dos DSMML para distintos dominios, y se escribieron las sintaxis de esos dos lenguajes utilizando lenguajes de propósito general distintos: en un caso ATL y en el otro EOL. La experiencia fue un éxito y se hicieron evidentes las ventajas de definir los lenguajes DSMML para permitir transformaciones dentro del mismo lenguaje, es decir, transformaciones que tomen un modelo y generen otro que conforma al mismo metamodelo.

La presente propuesta separa los roles de desarrollador de un lenguaje específico para la gestión de modelos, del rol del programador de ese lenguaje. Vimos que el diseñador necesita tener conocimientos del metamodelo del DSML al cual se le quieren gestionar los modelos, y de un lenguaje de transformación general para poder escribir la semántica del lenguaje de gestión. Luego el programador podrá escribir programas sin la necesitad de conocer en detalle ninguno de esos dos lenguajes.

Yendo mas allá se podría pensar que el desarrollador del lenguaje es experto solo en el dominio al cual le quiere gestionar los modelos y no de los lenguajes de transformación de propósito general. Ese conocimiento es el primordial y debería ser suficiente para poder definir operaciones complejas. Estas operaciones pueden establecerse a partir de operaciones básicas del metamodelo del DSML, como por ejemplo, la creación de instancias de las metaclases, el acceso, asignación de elementos, el agregado de elementos a las colecciones. Nuestro trabajo se enfoca ahora en cómo el desarrollador del lenguaje puede contar con estas operaciones básicas, y utilizarlas, haciendo transparente para él la traducción de estas operaciones a un lenguaje de transformación existente.

References

[1] Stahl, T. and Völter, M. Model-Driven Software Development. John Wiley & Sons, Ltd. (2006).

[2] Claudia Pons, Roxana Giandini, Gabriela Pérez. “Model Driven Software Development. Concepts and practical application”. Editorial: EDUNLP and McGraw-Hill Education. (2010).

[3] Kleppe, Anneke G. and Warmer Jos, and Bast, Wim. MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. (2003)

[4] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain specific languages. ACM Computing Surveys, 37(4):316–344, 2005.

[5] MOF QVT Adopted Specification 2.0. OMG Adopted Specification. November 2005. http://www.omg.org

[6] ATLAS team: ATLAS MegaModel Management (AM3) Home page, http://www.eclipse.org/gmt/am3/. (2006)

[7] 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

[8] Kolovos, DS, Paige, RF, Polack, FAC: The Epsilon Object Language (EOL). In: Rensink, A., Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066, pp. 128–142. Springer, Heidelberg (2006).

[9] GME: The Generic Modeling Environment, Reference site, http://www.isis.vanderbilt.edu/Projects/gme. (2006).

[10] Richard C. Gronback. Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley Professional. ISBN: 0-321-53407-7, 2009.

[11] Steve Cook, Gareth Jones, Stuart Kent, Alan Cameron Wills. Domain-Specific Development with Visual Studio DSL Tools. Addison-Wesley Professional. ISBN 0321398203, 2007.

[12] Bézivin, J., Jouault, F., Kurtev, I., Valduriez, P.: Model-based DSL Frameworks. (2006) OOPSLA Companion 2006:602-616[5].

[13] Frédéric Jouault, Jean Bézivin, Charles Consel, Ivan Kurtev, Fabien Latry: Building DSLs with AMMA/ATL, a Case Study on SPL and CPL Telephony Languages. Proceedings of the 1st ECOOP Workshop on Domain-Specific Program Development (DSPD), July 3rd, Nantes, France (2006).

[14] Barbero, M., Bézivin, J., Jouault, F. Building a DSL for Interactive TV Applications with AMMA. In Proceedings of the TOOLS Europe 2007 Workshop on Model-Driven Development Tool Implementers Forum. Zurich, Switzerland (June 2007).

[15] Di Ruscio, D., Jouault, F., Kurtev, I., Bézivin, J., Pierantonio, A.: Extending AMMA for Supporting Dynamic Semantics Specifications of DSLs. http://hal.ccsd.cnrs.fr/docs/00/06/61/21/PDF/rr0602.pdf (Downloaded March 2009).

[16] Jon Oldevik. MOFScript User Guide. Version 0.6 (MOFScript v 1.1.11), 2006.

[17] Kleppe, Anneke. MCC: A Model Transformation Environment. A. Rensink and J. Warmer (Eds.): ECMDA-FA 2006, LNCS 4066, pp. 173 – 187, Spain, June 2006. (2006)

[18] Blanc,X., Gervais, M., Lamari, M. and Sriplakich, P.. Towards an integrated transformation environment (ITE) for model driven development (MDD). In Proceedings of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI'2004), USA, July 2004. (2004)

[19] Oldevik, J. Transformation Composition Modeling Framework. DAIS 2005. Lecture Notes in Computer Science 3543, pp. 108-114. (2005)

[20] Wagelaar, Dennis. Composition Techniques for Rule-based Model Transformation Languages. Procs. of ICMT2008 – Int. Conference on Model Transformation. Zurich, Switzerland. July 2008. (2008)

[21] Bravenboer, M., Visser, E.: Concrete syntax for objects: Domain-specific language embedding and assimilation without restrictions. In: Proc. 19th Annual ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'04), ACM Press (2004) 365-383

[22] OMG/MOF Meta Object Facility (MOF) 2.0. OMG Adopted Specification. October 2003. http://www.omg.org

[23] Weske Mathias, "Business Process Management: Concepts, Languages, Architectures". Springer, Pag 3-67. ISBN 978-3-540-73521-2. 2008

[24] Object Management Group (OMG), http://www.omg.org

[25] Business Process Modeling Notation (BPMN) Version 1.2 OMG, http://www.omg.org/spec/BPMN/1.2

[26] 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

[27] Fowler, M.: Refactoring-Improving the Design of Existing Code. Addison-Wesley, Reading (2000)

[28] Claudia Pons, Jerónimo Irazábal, Roxana Giandini and Gabriela Pérez. On the semantics of domain specific transformation languages: implementation issues. Chapter 13 of the Book "Software Engineering: Methods, Modelling, and Teaching", prefaced by Ivar Jacobson. (2011).

[29] Kermeta web site - http://www.kermeta.org

[30] Eugenia web site - http://www.eclipse.org/epsilon/doc/eugenia/

[31] Dimitrios S. Kolovos, Louis M. Rose, Saad Bin Abid, Richard F. Paige, Fiona A.C. Polack, Goetz Botterweck. Taming EMF and GMF Using Model Transformation, accepted and to appear in Proc. International Conference on Model Driven Engineering Languages and Systems (MoDELS) Oslo, Norway, October 2010

[32] DSMML web site. http://www.lifia.info.unlp.edu.ar/eclipse/DSMML/

[33] EMFText web site - www.emftext.org/