REFINAMIENTO DEL DIAGRAMA DE CLASES UML A ORACLE®9I EN ATOM^3

UML CLASS DIAGRAM – ORACLE® 9I REFINEMENT IN ATOM^3

CARLOS M. ZAPATA
Grupo de Investigación en Ingeniería de Software. Facultad de Minas.Universidad Nacional de Colombia-sede Medellín

CARLOS A. ÁLVAREZ
Grupo de Investigación en Ingeniería de Software. Facultad de Minas.Universidad Nacional de Colombia-sede Medellín

FERNANDO ARANGO I.
Grupo de Investigación en Ingeniería de Software. Facultad de Minas.Universidad Nacional de Colombia-sede Medellín

Recibido para revisar 12 de Enero de 2006, aceptado 26 de Septiembre de 2006, versión final 15 de Octubre de 2006

RESUMEN: El OMG define el refinamiento como el proceso de transformación desde un modelo independiente de la plataforma de implementación hacia un modelo específico de la plataforma de implementación. Las herramientas CASE convencionales han experimentado problemas con este tipo de transformación, debido a la definición estática de los modelos incluidos en ellas, a las dificultades para la definición de reglas de transformación y al pobre rendimiento mostrado por ellas en la obtención de código. Las herramientas MetaCASE han surgido con nuevas capacidades para mejorar el refinamiento en el contexto de la transformación entre modelos. En este artículo se presenta una implementación en AToM^3 para refinamiento que transforma un diagrama de clases UML independiente de la plataforma de implementación a un diagrama de clases UML dependiente de la plataforma de implementación Oracle® 9i. Además, se muestra el uso de esta clase de refinamiento con un caso de estudio.

PALABRAS CLAVE: Diagrama de Clases, Modelo Relacional, Oracle® 9i, Metamodelamiento, Reglas de Refinamiento.

ABSTRACT: Defined by OMG, Refinement is a transformation process from a platform independent model to a platform specific model. CASE Tools have experienced problems with this kind of transformation, because of the static definition of metamodels included on them, difficulties for defining transformation rules and the poor performance showed by them in code obtaining. MetaCASE Tools have emerged with new capabilities for improving the refinement in the context of model transformation. In this paper, we present an AToM^3-based implementation for refinement between a platform-independent UML Class Diagram and an Oracle® 9i-platform UML Class Model. Furthermore, we show the use of this kind of refinement with a study case.

KEY WORDS: Class Diagram, Relational Model, Oracle® 9i, Metamodeling, Refinement Rules.

1. INTRODUCCIÓN

Entre las diferentes definiciones que se pueden encontrar sobre refinamiento en la literatura, se destaca una de las más recientes emitida por el Object Management Group (OMG, 2006A), que define el refinamiento como la adición de detalles a un modelo, con el fin de transformarlo desde un modelo independiente de la plataforma de implementación en un modelo específico de la plataforma de implementación. Esta forma de refinamiento permite establecer el nexo entre un modelo que describa un problema específico y un modelo de diseño, que posibilite un acercamiento con la implementación de la solución definitiva.

El Unified Modeling Language (UML), otro de los estándares del OMG (2006B), se ha venido generalizando como el paradigma de modelamiento de los problemas en Ingeniería del Software. Su utilización se ha fomentado aún

más con la aparición de herramientas CASE (Computer-Aided Software Engineering), tales como Rational® Rose, Oracle® Designer – Anderson y Wendelken, 1996 – o Poseidon®, que han permitido la elaboración de los diferentes diagramas que conforman a UML y su posterior conversión a piezas de código en diferentes lenguajes como Java, C++ o SQL. La generación de este código se realiza a partir de reglas de refinamiento que están programadas dentro de las herramientas mismas y que, por los general, no son accesibles al modelador.

En este enfoque se presentan tres tipos de problemas, a saber:

- Cuando se introducen cambios en el paradigma de modelamiento que soportan, como por ejemplo los cambios que se produjeron en UML al pasar de la versión 1.5 a la 2.0, se hace necesario actualizar la versión de la herramienta misma, pues no es posible realizar modificaciones al "metamodelo" con el que internamente estas herramientas describen el paradigma que soportan.
- Las decisiones de diseño involucradas al definir las reglas de refinamiento no son accesibles al modelador, dejándole como única opción para acceder a otros tipos de decisiones la realización directa de modificaciones sobre el código generado.
- El código que se genera aún es muy incipiente y prácticamente se limita a la realización de "plantillas" que debe completar el modelador, pues son bastante incompletas y no definen todas las características necesarias para la realización de la aplicación.

La introducción del Metamodelado en las herramientas CASE, aparece como una forma de solución de los dos primeros problemas mencionados, que le posibilita al modelador especificar el paradigma de modelamiento que soporta la herramienta, en términos de su metamodelo, y le suministra herramientas para definir las conversiones a código mediante la definición de reglas editables.
La aparición del paradigma MDA (Model Driven Architecture) del OMG (2006A), para posibilitar la animación de modelos y su conversión a código ejecutable ha impulsado el planteamiento explícito de reglas de refinamiento. UMLX, denominado UML ejecutable, es un ejemplo de esta tendencia (Willink, 2006). En Arango et al. (2006) se define una conversión del diagrama de clases de UML al modelo relacional de Oracle® 9i, mediante el establecimiento de los metamodelos asociados a cada uno de los paradigmas y la definición de unas reglas de refinamiento cuya aplicación permite la transformación de un modelo en otro. Sin embargo, estas experiencias aún no se han llevado a implementación o su implementación aún está en experimentación.

Actualmente existe un conjunto de herramientas basadas en metamodelamiento, denominadas MetaCASE tales como AToM^3 (De Lara y Vangheluwe, 2002), DOME (DOME, 2006), GME (Ledeczi et al., 2001) y MetaEdit+ (MetaEdit+, 2006) que posibilitan la animación de modelos, permitiendo al modelador la libertad de definir las características deseables de los metamodelos en los que se basan dichos modelos. En el grupo de investigación en Ingeniería de Software se han realizado algunos trabajos que utilizan mecanismos de transformación entre modelos para definir diferentes formas de consistencia y refinamiento en AToM^3 (Zapata y Alvarez, 2005, Zapata et al., 2006) y DOME (Cabarcas et al., 2006).

En este artículo se describe una propuesta de implementación del trabajo de Arango et al. (2006) empleando la herramienta AToM^3 y su mecanismo de transformación denominado "Gramática de Grafos". Particularmente, se define una parte del metamodelo de clases UML simplificado y el metamodelo de Oracle® 9i definidos por Arango et al. (2006) y se implementan cuatro reglas de refinamiento de este trabajo para realizar la transformación entre los metamodelos y obtener finalmente código SQL.

Este artículo está organizado de la siguiente manera: en la Sección 2 se definen algunos de los conceptos que involucran el refinamiento en el marco de MDA, en la Sección 3 se muestra la herramienta AToM^3 con su mecanismo de

transformación entre modelos y en la Sección 4 se realiza la propuesta de refinamiento en AToM^3, mostrando un caso de estudio para su ejemplificación. Finalmente, en las Secciones 5 y 6 se muestran las conclusiones y los trabajos futuros respectivamente.

2. EL REFINAMIENTO EN EL MARCO DE MDA

La arquitectura orientada por modelos (Model-Driven Architecture – MDA) es el enfoque definido por el OMG para posibilitar la separación de las especificaciones de la solución a un problema de la implementación de esa solución en una plataforma de implementación definida (e.g. Oracle®, J2EE, Java, etc.). Para ello, define dos tipos de modelos (OMG, 2006A):

- Modelos Independientes de la plataforma de implementación (Platform independent models – PIM), que son modelos UML que describen las características de la solución, pero que aún no poseen elementos que sean propios de una plataforma específica.
- Modelos Específicos de la plataforma de Implementación (Platform Specific models – PSM), que son modelos UML enriquecidos con estereotipos que describen ciertos elementos que son propios de cada una de las plataformas de implementación disponibles para la elaboración de código ejecutable.

El refinamiento se define en términos de MDA como la adición de los detalles necesarios para pasar de un modelo abstracto (por ejemplo un PIM) a un modelo más detallado (que puede ser un PSM) que en general estará más cerca de una plataforma de implementación.

Una opción para la definición de esta transformación se presenta en Arango et al. (2006), donde se realiza un metamodelo simplificado del diagrama de Clases de UML y se elabora un metamodelo también simplificado del modelo relacional de Oracle® 9i, los cuales se pueden ver en las Figuras 1 y 2 respectivamente. Adicionalmente, en este trabajo se realiza una recopilación de las reglas de refinamiento entre ambos modelos, entre las cuales se cuentan las siguientes (algunas de ellas con su expresión en lógica de predicados; un mayor detalle se puede consultar en Arango et al., 2006):

- Regla 1: Toda instancia de la Metaclase UML *Class*, que no participe en ninguna relación de generalización se transforma en una instancia de la Metaclase Oracle *Tabla Persistente* si su atributo *isAbstract* es falso, conservando el mismo nombre.

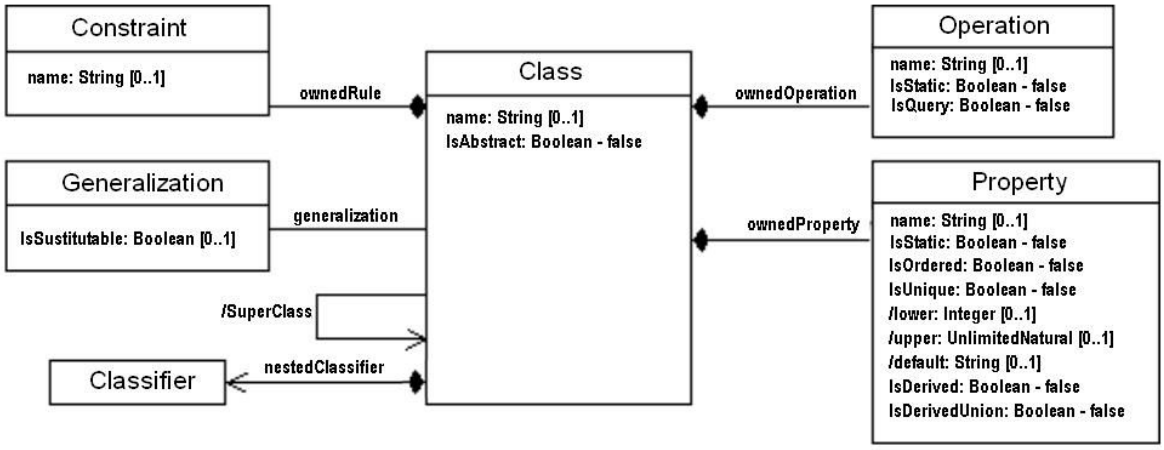

Figura 1. Metamodelo del Diagrama de Clases, tomado de Arango et al. (2006)
Figure 1. Class Diagram Metamodel, taken from Arango et al. (2006)

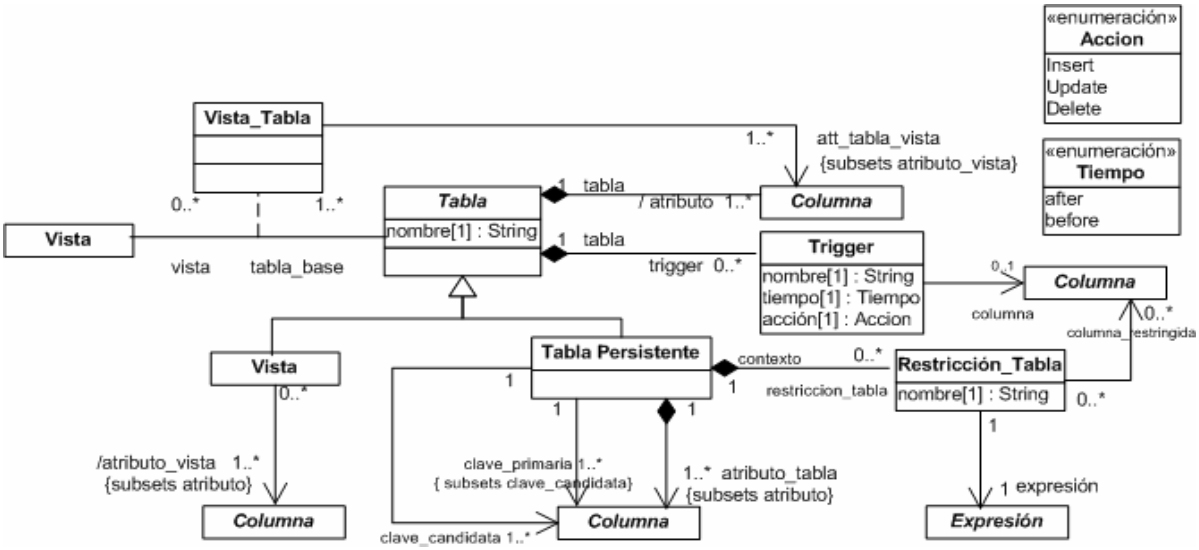

Figura 2. Metamodelo del Modelo Relacional de Oracle® 9i, tomado de Arango et al. (2006)
Figure 2. Oracle® 9i Relational Model Metamodel, taken from Arango et al. (2006)

La expresión en lógica de esta regla es:

$$\begin{aligned}&\text{Regla}:\forall(c\in MetaclaseUML\_Class)\\&[(\neg c.isAbstract)\wedge(c.generalization=\varnothing)\Rightarrow\\&\exists(t\in MetaclaseORACLE\_Tabla\ Persistente\\&[t.nombre=c.name])\end{aligned}$$

- Regla 2: Todas las instancias de la Metaclase UML *Property* que pertenecen a una Clase UML (valores del atributo ownedAttribute), se convierten en instancias de la Metaclase Oracle *Columna*, manteniendo el mismo nombre. Esta Metaclase es abstracta y se especializa en las Metaclases Oracle *Atributo Estructural* y *Atributo de Implementación*. Las Columnas obtenidas pertenecerán a la instancia de la Metaclase Oracle *Tabla Persistente* a la que se mapeó la Clase y constituyen los valores del atributo *atributo_tabla* de dicha Metaclase Oracle.

  Las expresiones en lógica de predicados para esta regla son las siguientes:

$$\begin{aligned}&\text{Pre}:\exists(t\in MetaclaseORACLE\_Tabla\ Persistente)\\&\wedge\exists(c\in MetaclaseUML\_Class)\\&t=f(c)\end{aligned}$$

$$\begin{aligned}&\text{Regla}:\forall(p\in MetaclaseUML\_Property)[((p\in c.ownedAttribute)\\&\wedge(p.association=\varnothing))\Rightarrow\\&\exists(att\_est\in MetaclaseORACLE\_Atributo\ Estructural\\&[(att\_est\in t.atributo\_tabla)\wedge(att\_est.nombre=p.name)])\end{aligned}$$

$$\begin{aligned}&\text{Pre}:\exists(t\in MetaclaseORACLE\_Tabla\ Persistente)\\&\wedge\ \exists(c\in MetaclaseUML\_Class)\ /\\&t=f(c)\end{aligned}$$

$$\begin{aligned}&\text{Regla}:\forall(r\in MetaclaseUML\_Constraint)\\&[(r\in c.ownedRule)\Rightarrow\\&\exists(rt\in MetaclaseORACLE\_Restriccion\ Tabla\\&[(rt\in t.restriccion\_tabla)\wedge(rt.nombre=r.name))]]\end{aligned}$$

- Regla 3: La generalización se transforma a una tabla que reúne tanto los atributos de la superclase (obligatorios y no obligatorios) como los de las subclases (opcionales). Además se debe tener una columna obligatoria llamada *tipo* para especificar para cada tupla de la tabla a cual clase de la jerarquía se asocia.

  La expresión en lógica de predicados para esta regla es:

----- Creación de la nueva instancia de Tabla Persistente

$$Regla: \exists (c, c1, c2 \in MetaclaseUML\_Class / c1.superclass = c2.superclass = c) \Rightarrow \exists (tabla \in MetaclaseORACLE\_Tabla\ Persistente / (tabla.nombre = c.name) \wedge (tabla.atributo\_tabla = f(c.ownedAttribute) \cup [att = f(c1.ownedAttribute) \cup f(c2.ownedAttribute) / \forall (a \in att / a.obligatorio = falso)]) \wedge \exists (att\_est \in MetaclaseORACLE\_Atributo\ Estructural / (att\_est.nombre = tipo) \wedge (att\_est.obligatorio = verdadero)) \wedge (tabla.atributo\_tabla = tabla.atributo\_tabla \cup att\_est)$$

------ Determinación de los valores posibles para el Atributo Estructural *tipo*

$$\wedge (att\_est.tipo \in MetaclaseORACLE\_Tipo\ Basico) \wedge (att\_est.tipo = Varchar) \wedge \exists (r \in MetaclaseORACLE\_Check / (r.nombre = restriccion + att\_est.name) (r.valores = c1.name \cup c2.name)) \wedge (att\_est.restriccion = r)$$

- Regla 4: Para las asociaciones uno a muchos, la cardinalidad uno se representa como un nueva columna en la tabla del lado de muchos, del mismo tipo de la clave primaria de la tabla del lado de uno, con sus respectivas restricciones de clave foránea y obligatoriedad.

En la siguiente sección se muestra el AToM^3 y su mecanismo de transformación de modelos, como una alternativa para la implementación del refinamiento descrito en esta sección.

3. ATOM^3 Y LA TRANSFORMACIÓN ENTRE MODELOS

Entre las herramientas de metamodelamiento disponibles en la actualidad se encuentra el AToM^3 (A Tool for Multi-Formalism Modeling and Meta-Modeling) (De Lara y Vangheluwe, 2002), escrita en el lenguaje de programación Phyton. En esta herramienta se incluye un formalismo básico similar al modelo entidad relación extendido con el cual se realiza la definición de los diferentes meta-modelos en un entorno gráfico, que posteriormente se convierte a código phyton. Así, se posibilita la definición de cualquier tipo de metamodelo, simplemente definiendo las "entidades" que hacen parte del mismo y sus posibles interconexiones o "relaciones". Cuando se ha definido el metamodelo, es posible la definición de modelos basados en ese formalismo (De Lara y Vangheluwe, 2002); así, se soluciona la limitación de las herramientas CASE, en las cuales sólo se pueden emplear los formalismos definidos en ellas para la elaboración de los diferentes diagramas que representan un problema.

En adición al formalismo que se describió, AToM^3 tiene la posibilidad de realizar conversiones entre modelos empleando para ello la denominada "Gramática de Grafos", un mecanismo también programado en el lenguaje phyton. La Gramática de Grafos posee características similares a las gramáticas basadas en texto puesto que pueden definir una transformación entre dos tipos de modelos pero, a diferencia de ellas, sus reglas de transformación se expresan de manera gráfica y no a modo de texto. La gramática de grafos incluida en AToM^3 define reglas que poseen un lado izquierdo (left-hand side o LHS) donde se definen de manera gráfica las precondiciones necesarias para disparar una regla y un lado derecho (right-hand side o RHS) que contiene el modelo que reemplazará el que equipare el lado izquierdo de la regla. Para las reglas expresadas de esta manera también se deben definir unas condiciones y unas acciones para ejecutar cuando la regla se active. La Gramática de Grafos de AToM^3 posee también un mecanismo que va rescribiendo el modelo a medida que las diferentes reglas se van activando hasta que no haya reglas que se puedan ejecutar (De Lara et al., 2003).

Tanto el mecanismo de definición de los metamodelos como la gramática de grafos se emplean en la siguiente sección para realizar la transformación de refinamiento que es objeto del presente artículo.

4. REFINAMIENTO DEL DIAGRAMA DE CLASES DE UML AL MODELO RELACIONAL DE ORACLE® 9I EN EL MARCO DE ATOM^3

4.1 Definición de los metamodelos

El formalismo que emplea AToM^3 para la definición de los metamodelos es el modelo entidad relación extendido, en el cual las cajas

son las entidades y los rombos las relaciones. Se puede asociar una forma gráfica a cada elemento, que permite utilizar el metamodelo para generar las instancias correspondientes. En las Figuras 3 y 4 se pueden apreciar los elementos definidos para los metamodelos correspondientes a las Figuras 1 y 2. Para el metamodelo de clases (véase Figura 3) las entidades principales son *Class, Operation, Property* y *Constraint*, equivalentes a las clases respectivas en el metamodelo de la Figura 1, y las relaciones principales son *aggregation* entre la entidad *class* y las demás entidades, la relación *superclass* entre las entidades del tipo *Class* y la relación *Association* entre las entidades *property*. Para el metamodelo de Oracle® correspondiente a la Figura 2 (véase Figura 4), las entidades son *Table* y *Column* y las relaciones *OMAggregation* y *OMForeignKey*; la entidad *SQLCode* y la relación *OMConnection* se requieren únicamente para efectos de almacenar el código SQL que se genera después de la conversión.

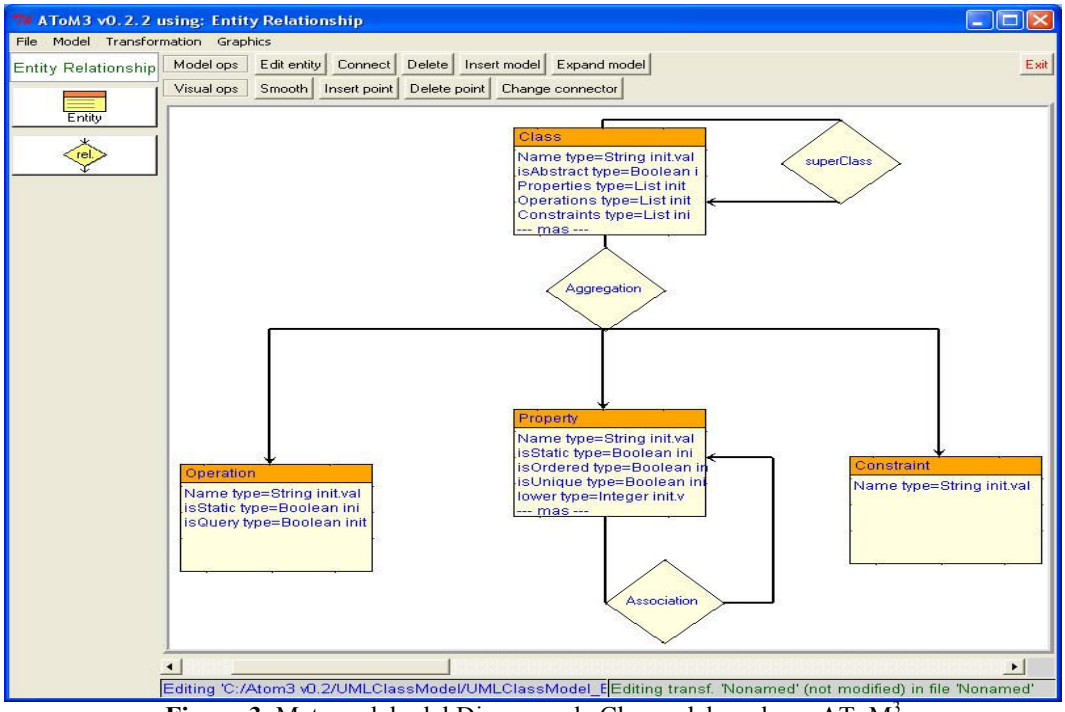

Figura 3. Metamodelo del Diagrama de Clases elaborado en AToM^3
Figure 3. Class Diagram Metamodel made in AToM^3

4.2 Reglas de Refinamiento:

La Gramática de Grafos es el mecanismo empleado por el AToM^3 para la escritura de las reglas de transformación entre los diferentes metamodelos, que para este caso en particular serán las reglas de refinamiento de la Sección 2 entre los dos metamodelos definidos en la Sección 4.1. En la Figura 5 se muestra la ventana correspondiente a la definición de las reglas, incluyendo las siguientes:

- convertClass1: Realiza una combinación de las reglas de refinamiento 1 y 2.
- convertClass2: Realiza una combinación de las reglas de refinamiento 2 y 3.
- convertAssociation: Realiza la regla de refinamiento 4.
- deleteClass: Borra las clases una vez se han transformado en tablas.
- deleteProperty: Borra las propiedades cuando se han transformado en columnas.
- deleteOperation: Borra las operaciones, las cuales no son transformadas con ninguna de las reglas definidas.
- deleteConstraint: Borra las restricciones, que tampoco se transforman con ninguna de las reglas definidas.

- deleteSuperClass: Borra las relaciones de herencia cuando ya se ha realizado su transformación.
- deleteAggregation: Borra las agregaciones cuando ya se ha realizado su transformación.
- deleteAssociation: Borra las asociaciones entre propiedades cuando ya se han transformado en claves foráneas.

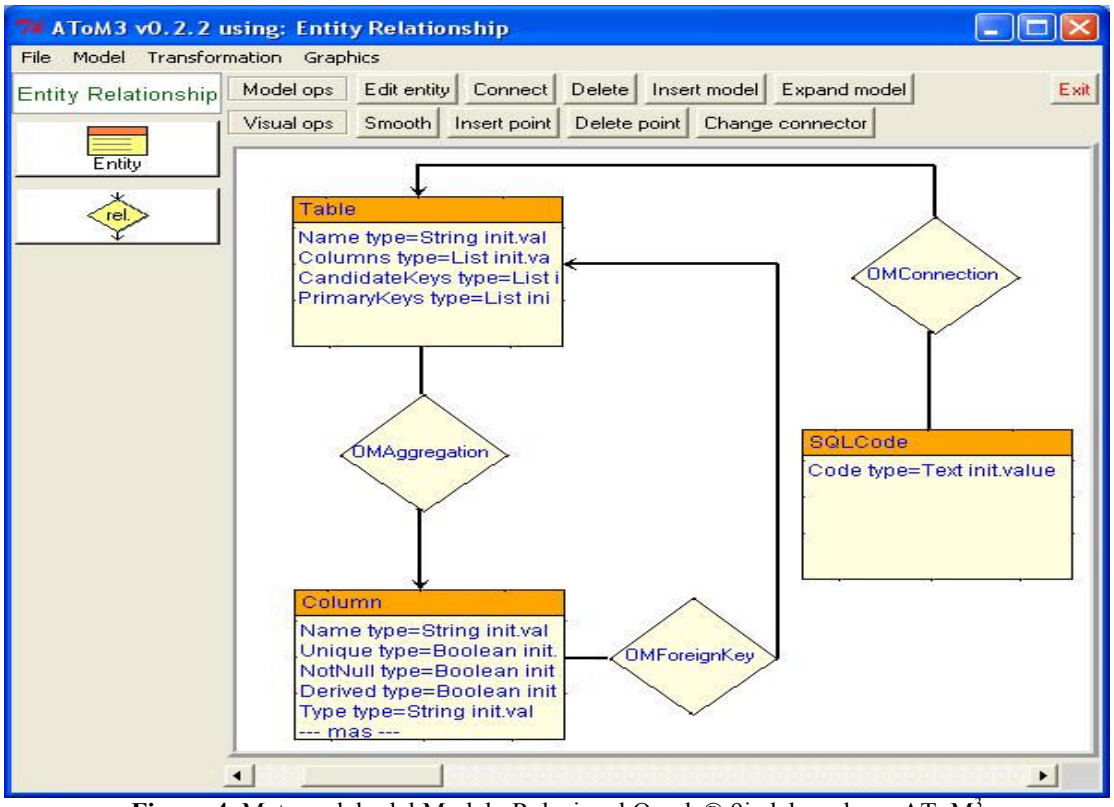

Figura 4. Metamodelo del Modelo Relacional Oracle® 9i elaborado en AToM[3]
Figure 4. Oracle® 9i Relational Model Metamodel made in AToM[3]

A modo de ejemplificación, se mostrará la forma en que se programó la regla convertClass1. Las demás reglas tienen esquemas similares que no se mostrarán por razones de falta de espacio. En la Figura 6 se muestran el LHS y el RHS de la regla que determina qué elementos tomar y en qué se deberán transformar. En este caso particular, se toman todas las clases y se transforman en clases copiadas, para poder extraer las características necesarias para terminar la conversión.

Como el mecanismo de la definición del LHS y el RHS de la regla no es suficiente para la implementación completa de la regla, la conversión final se realiza con un conjunto de acciones en lenguaje phyton que se ejecutan cuando se activa la regla; el código es el siguiente (los comentarios comienzan con el símbolo #; las demás líneas son comandos en lenguaje phyton):

#Se realizan el filtro sobre las asociaciones únicamente.

```
def filtrar(x): return x.__class__.__name__ ==
'Association'
```

#Se invocan las funciones que servirán para la creación de los elementos del metamodelo Oracle

```
from OracleModel_MM import
createNewTable
```

```
from OracleModel_MM import
createNewColumn
from OracleModel_MM import
createNewOMAggregation
from Tools import getObject
from Tools import toString
from Tools import getAttributes
from Tools import getInheritedAttributes
```

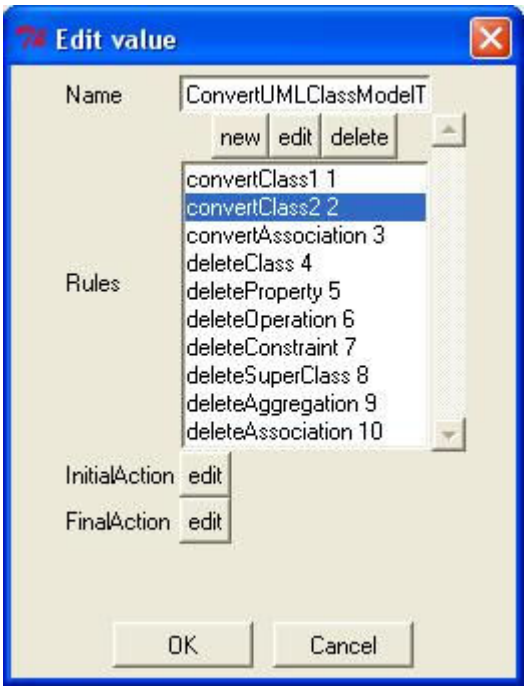

Figura 5. Ventana de definición de reglas de refinamiento elaborada en AToM^3

Figure 5. Refinement Rules definition window made in AToM^3

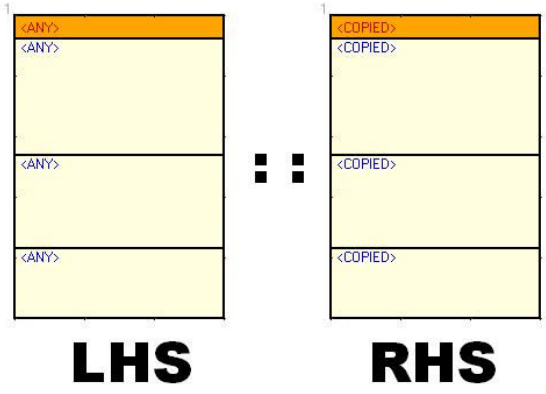

Figura 6. LHS y RHS de la regla convertClass1.

Figure 6. LHS and RHS from convertClass1 rule.

#Se crea la tabla

```
clase = self.getMatched( graphID,
self.LHS.nodeWithLabel(1))
table = createNewTable(atom3i,
clase.graphObject_.x, clase.graphObject_.y, 1)
table.Name = clase.Name
table.graphObject_.ModifyAttribute("Name",
table.Name.toString())
```

#Se crean las columnas

```
keys = []
columns = []
for col in getAttributes(clase, "Aggregation",
"Property"):
    name = col.toString()
    columns.append(ATOM3String(name))
    obj = getObject(clase, 'Class',
clase.Name.toString(), 'Aggregation',
'Property', name)
    if(obj):
        rel = obj[1]
        obj = obj[2]
        column = createNewColumn(atom3i,
obj.graphObject_.x, obj.graphObject_.y, 1)
        column.Name = obj.Name
        column.Unique.setValue(0)
        column.NotNull.setValue(0)
        column.Derived.setValue(0)
        if(obj.isUnique.getValue()[1]):
column.Unique.setValue(1)
        if(obj.lower.getValue()):
column.NotNull.setValue(1)
        if(obj.isDerived.getValue()[1]):
column.Derived.setValue(1)
        if(obj.upper.getValue() > 1):
column.Type.setValue('Collection')
        else:  column.Type.setValue('Basic')

column.graphObject_.ModifyAttribute("Name
", column.Name.toString(25))

column.graphObject_.ModifyAttribute("Uniqu
e", column.Unique.toString(25))

column.graphObject_.ModifyAttribute("NotN
ull", column.NotNull.toString(25))

column.graphObject_.ModifyAttribute("Deriv
ed", column.Derived.toString(25))

column.graphObject_.ModifyAttribute("Type",
column.Type.toString(25))
```

```
        isAssociation = len(filter(filtrar,
  obj.out_connections_ + obj.in_connections_))
  > 0
        if(not isAssociation and (not
  column.Derived.getValue()[1]) and
  column.NotNull.getValue()[1] and
  column.Unique.getValue()[1]):
              keys.append(ATOM3String(name))
        x = rel.graphObject_.x
        y = rel.graphObject_.y
        flujo =
  createNewOMAggregation(atom3i, x, y, 1)
        atom3i.drawConnections((table, flujo),
  (flujo, column))
  table.Columns.setValue(columns)
  table.CandidateKeys.setValue(keys)
  if(len(keys) == 1):
  table.PrimaryKeys.setValue(keys)
  table.graphObject_.ModifyAttribute("Columns
  ", table.Columns.toString(25, 6))
  table.graphObject_.ModifyAttribute("Candidat
  eKeys", table.CandidateKeys.toString(25, 4))
  table.graphObject_.ModifyAttribute("Primary
  Keys", table.PrimaryKeys.toString(25, 3))
```

Una vez se han definido todas las reglas de refinamiento, se asigna la transformación correspondiente a un botón del metamodelo, de forma tal que se pueda ejecutar instantáneamente a partir del modelo que se define como instancia del metamodelo. El código en lenguaje phyton correspondiente a la ejecución de la transformación es el siguiente:

#Se invoca la conversión

```
from ConvertUMLClassModelToOracleModel
import
ConvertUMLClassModelToOracleModel
```

#Se invoca la reescritura

```
from GraphRewritingSys import
GraphRewritingSys
self.grs = GraphRewritingSys(self,
[ConvertUMLClassModelToOracleModel(self
)], self.ASGroot)
self.grs.evaluate(0, 0, self.grs.SEQ_RANDOM)
```

Con ello se garantiza que se reescriban los elementos del metamodelo de Clases en términos del metamodelo de Oracle.

4.3 Caso de Estudio

Para ejemplificar el manejo de las reglas de refinamiento en AToM^3 se escogió un diagrama de clases para una empresa en la cual existen clientes que pueden ser personas naturales o jurídicas. Los clientes tienen nombre, tope de pedidos y valor de despachos; las personas naturales se identifican con su cédula, en tanto que las personas jurídicas se identifican con un nit.

Los clientes en general pueden realizar pedidos, de los cuales se conoce su número y su fecha. En la Figura 7 se puede apreciar el modelo en AToM^3, que presenta ligeras diferencias con la notación estándar de UML, pues se pueden apreciar las agregaciones de propiedades ligadas con cada una de las clases presentes, y se incluye una caja adicional en las clases para las restricciones.

Además, en cada una de las clases se hacen explícitas las propiedades que pertenecen a una clase y las propiedades que hereda (lo cual se denota con el nombre de la clase padre, el símbolo $ y el nombre de la propiedad, por ejemplo cliente$nombre). Finalmente, se hicieron explícitas las asociaciones entre propiedades y no entre clases, con el fin de identificar mejor las claves foráneas en el modelo relacional de Oracle.

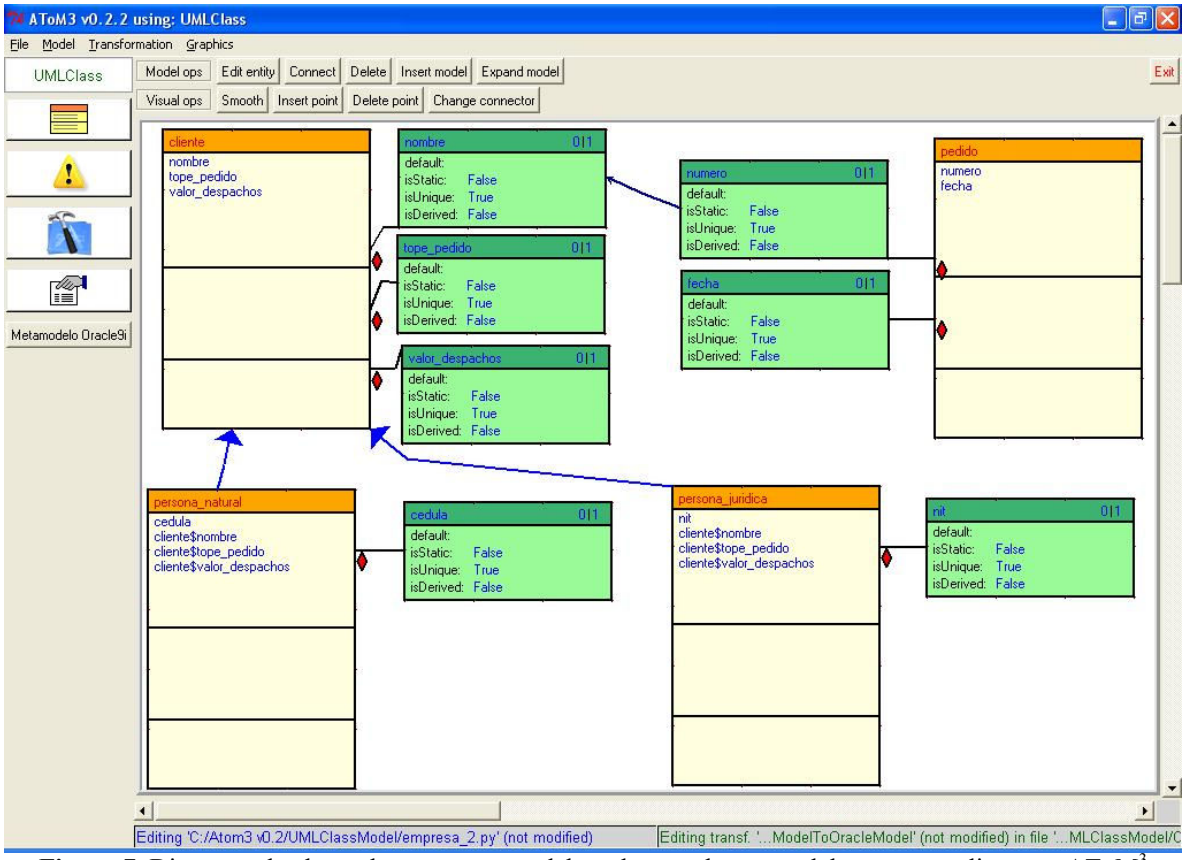

Figura 7. Diagrama de clases de una empresa elaborado con el metamodelo correspondiente en AToM^3
Figure 7. Enterprise Class Diagram made with the corresponding metamodel in AToM^3

Una vez se oprime el botón "Metamodelo Oracle 9i" se invoca el proceso de transformación y se obtiene el nuevo modelo que se muestra en la Figura 8, con sólo dos tablas persistentes (cliente y pedido); se aprecia cómo aparece la columna "Type", que incluye los tipos persona_natural y persona_jurídica, y también cómo se involucran en esa tabla las columnas correspondientes al cliente y a los diferentes tipos de clientes. Además, se incluye la relación entre número del pedido y nombre del cliente.

Finalmente, al oprimir el botón "Generar SQLCode" se adicionan dos nuevas entidades al modelo que permiten visualizar el código SQL correspondiente a la creación de cada una de las tablas. Si bien el código phyton, correspondiente a la creación de este botón y su resultado, escapa al alcance de este artículo,se deja la Figura 9 como muestra de lo que podría hacerse para la generación del código SQL a partir del modelo Oracle expresado en la Figura 8.

5. CONCLUSIONES

- Las herramientas metaCASE pueden realizar las mismas acciones de las herramientas CASE convencionales, pero añadiendo la flexibilidad de definición de metamodelos y reglas de transformación entre modelos. Con estos mecanismos se pueden realizar labores complejas como el refinamiento entre modelos independientes de la plataforma de implementación y modelos específicos de la plataforma de implementación.

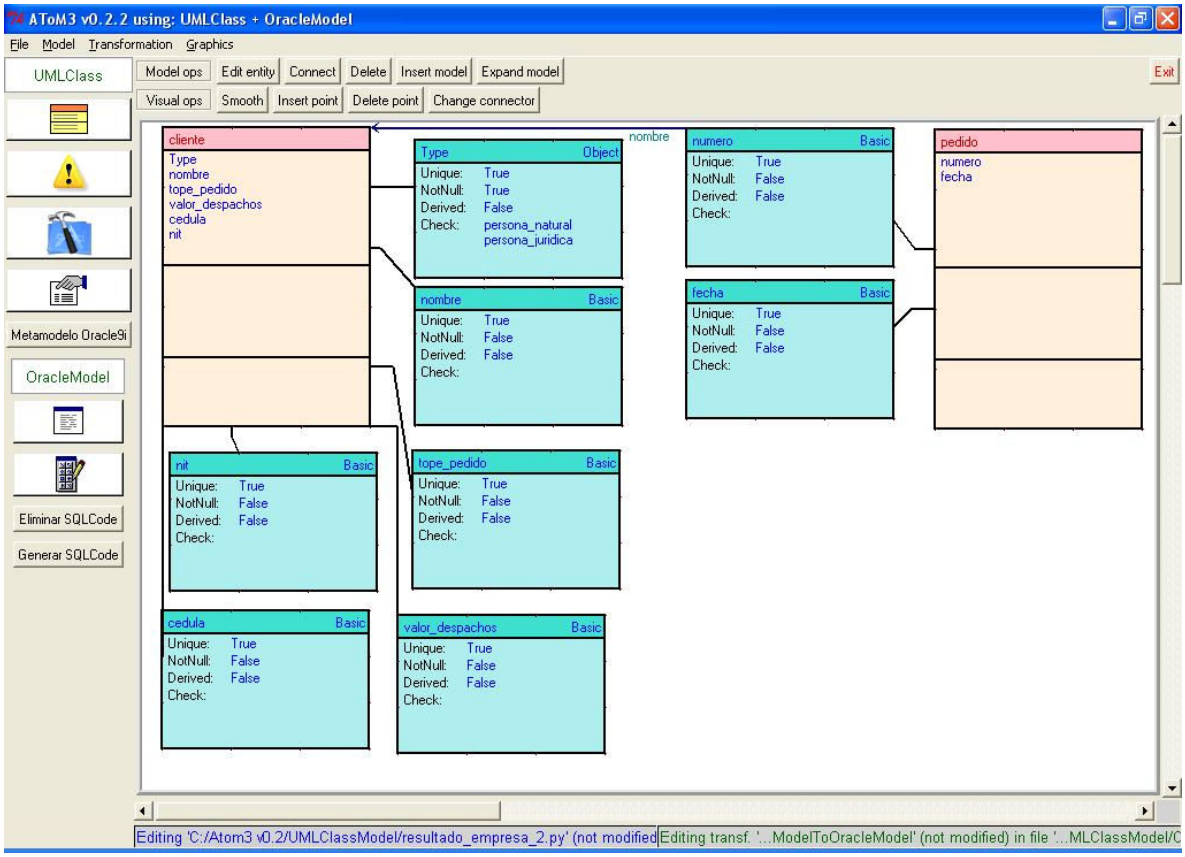

Figura 8. Modelo Relacional de una empresa obtenido después de presionar el botón "Metamodelo Oracle® 9i"
Figure 8. Enterprise Relational Model obtained by pressing "Metamodelo Oracle® 9i" button

- En este artículo se ha realizado una propuesta para la aplicación de cuatro reglas de refinamiento del diagrama de clases de UML al modelo relacional de Oracle® 9i, utilizando para ello la herramienta metaCASE AToM^3 y su mecanismo de transformación entre modelos denominado "Gramática de Grafos". Igualmente, se ejemplificó la transformación con un caso de estudio y se mostró que es posible la generación de código SQL a partir del modelo relacional de Oracle® 9i empleando el entorno de AToM^3 como si fuera una herramienta CASE convencional.
- Si bien es posible realizar la implementación de las reglas de conversión, un análisis de la expresión de las reglas en lógica de predicados versus su implementación en AToM^3 revela la complejidad de la Gramática de Grafos para realizar tal conversión, pues no es suficiente con la definición del LHS y el RHS y se debe recurrir a la programación de acciones en phyton.
- El AToM^3 es una herramienta metaCASE con orientación a los diagramas y no a los modelos con muchos diagramas. El código fuente de una aplicación se construye con el aporte de muchos diagramas que conforman el modelo del problema. Este tipo de situaciones no es posible tratarlo con el mecanismo de refinamiento explorado para el AToM^3.

6. TRABAJOS FUTUROS

- En este artículo se trabajaron únicamente cuatro de las reglas de refinamiento entre los metamodelos anotados. Sin embargo, en Arango et al. (2006) se listan muchas

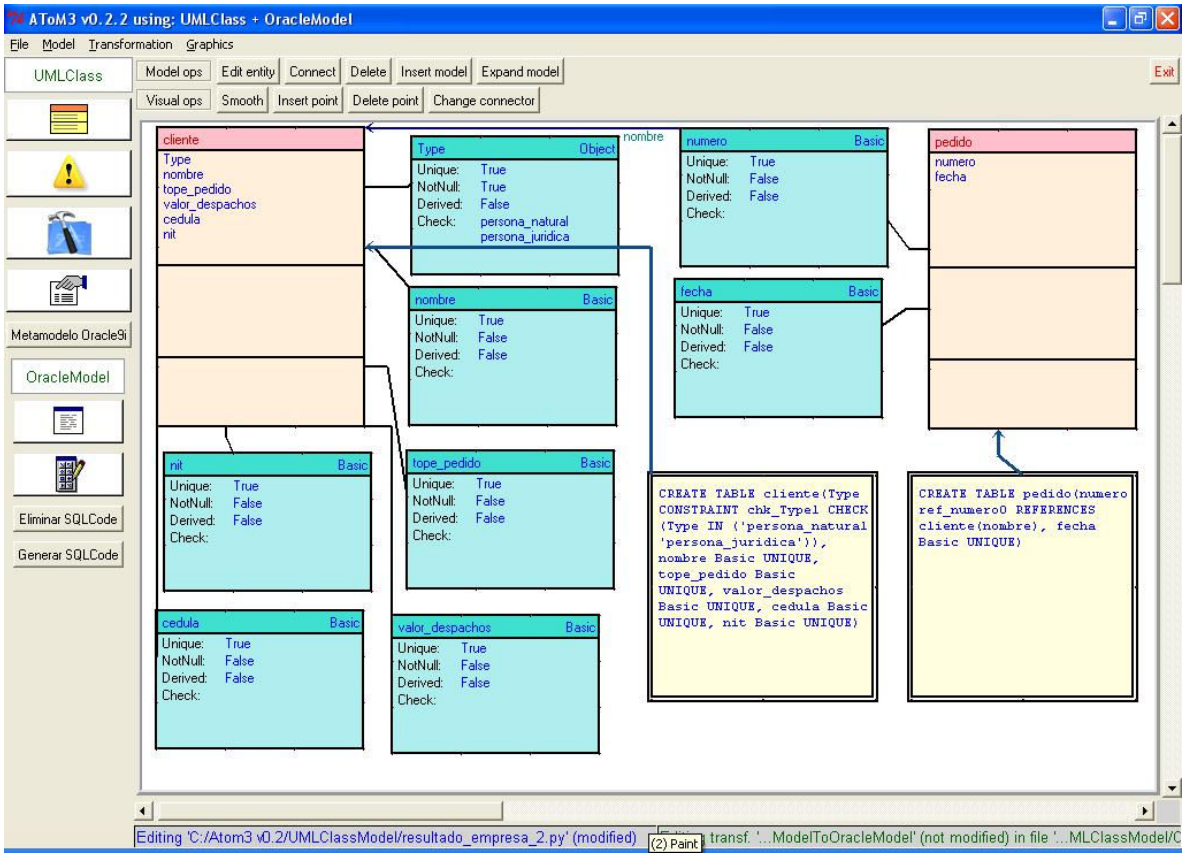

Figura 9. Código SQL obtenido después de presionar el botón "Generar SQLCode"
Figure 9. SQL Code obtained by pressing "Generar SQLCode" button

más reglas que son susceptibles de ser programadas con las capacidades anotadas para la herramienta metaCASE AToM^3.

- La propuesta se realizó tomando como base el diagrama de clases y llegando al metamodelo de Oracle® 9i. Sin embargo, con la definición de diferentes tipos de metamodelos de plataformas de implementación, tales como Java o J2EE se podrían lograr traducciones a diferentes plataformas. El refinamiento ejemplificado y definido se podría realizar también partiendo de otros tipos de diagramas, como el diagrama de secuencias, por ejemplo, que está ya muy cercano a la plataforma de implementación definida para la solución de un problema específico. Incluso, definiendo los metamodelos en el formalismo disponible en AToM^3 para ello, sería posible tomar como punto de partida otros diagramas diferentes a UML, como el diagrama de procesos, por ejemplo.
- Sería interesante también explorar la posibilidad de obtener un modelo más completo de la aplicación a partir de un conjunto de diagramas y no de uno sólo como se está realizando en este caso. Por ejemplo, se podría partir de diagramas de clases, secuencias y transición de estados para obtener un modelo específico de la plataforma de implementación, que estaría muy cercano a código ejecutable en un lenguaje definido. Esta característica motiva el trabajo en nuevas herramientas metaCASE que tengan orientación al modelo más que orientación al diagrama.

AGRADECIMIENTOS

Este artículo se realizó en el marco del proyecto de Investigación "Extensiones en herramientas CASE con énfasis en formalismos y reutilización" financiado por COLCIENCIAS, la

Universidad Nacional de Colombia y la Universidad EAFIT.

REFERENCIAS

[1]. OMG. *Model-Driven Architecture MDA*. Object Management Group. Available: http://www.omg.org/MDA/. [Citado 10 de Enero de 2006A].

[2]. OMG. *OMG Unified Modeling Language Specification.* Object Management Group. Available: http://www.omg.org/UML/. [Citado 10 de Enero de 2006B].

[3]. Anderson, C., y Wendelken, D. *The Oracle® Designer/2000 Handbook*. Addison-Wesley. 1996

[4]. Arango, F., Gómez, M. C. y Zapata, C. M. *Transformación del Modelo de Clases UML a Oracle® 9i bajo la directiva MDA: Un caso de Estudio.* Por aparecer, Revista DYNA, 2006.

[5]. Willink, E. *The UMLX Language Definition.* Available: http://www.eclipse.org/gmthome/doc/umlx/umlx.pdf. [Citado 10 de Enero de 2006]

[6]. De Lara, J., y Vangheluwe, H. *AToM3: A tool for Multi-Formalism and Meta-Modeling*. Proceedings of the Fifth Ingernational Conference on Fundamental Approaches to Software Engineering, Grenoble, 2002. Pp. 174-188.

[7]. DOME. *What is Dome.* Available: http://www.htc.honeywell.com/dome/description.htm. [Citado 10 de Enero de 2006].

[8]. Ledeczi A., Maroti M., Bakay A., Karsai G., Garrett J., Thomason IV C., Nordstrom G., Sprinkle J. y Volgyesi P. *The Generic Modeling Environment*. Proceedings of the Workshop on Intelligent Signal Processing, Budapest. 2001..

[9]. MetaEdit+. MetaCase Consulting. Available: http://www.metacase.com. [Citado 8 de Diciembre de 2004].

[10]. Zapata, C. M. y Álvarez, C. *Conversión de Diagramas de Procesos en Diagramas de Casos de Uso usando AToM3.* Revista DYNA, No. 146, 2005. Pp. 103-113.

[11]. Zapata, C. M., Álvarez, C. y Arango, F. *Una Propuesta para el Manejo de Restricciones en Modelos de Clases usando AToM3.* Por aparecer, Revista Ingeniería y Desarrollo, 2006.

[12]. Cabarcas, D., Arango, F. y Zapata, C. M. *Establecimiento y Verificación de la Consistencia en DOME: un caso de estudio.* Por aparecer, Revista DYNA, 2006.

[13]. De Lara, J., Vangheluwe, H y Alfonseca, M. *Using Meta-Modelling and Graph-Grammars to Create Modelling Environments*. Electronic Notes in Theoretical Computer Science, Vol. 72 No. 3. 2003.