Propuesta para el manejo de restricciones en modelos de clases usando atom3*

Carlos M. Zapata J.**, Carlos A. Alvarez C.***,
Fernando Arango I.****
Grupo de Investigación UN-INFO, Escuela de Sistemas, Facultad de Minas
Universidad Nacional de Colombia

Resumen

La construcción de los modelos para el desarrollo de software se ha realizado tradicionalmente con herramientas CASE. En estas herramientas los formalismos de cada modelo ya se encuentran plenamente definidos, lo que implica que no es posible agregarles nuevas restricciones. Las herramientas de Metamodelado surgieron como una manera de solución a este problema, pues poseen formalismos propios (generalmente gráficos) que permiten la expresión de diferentes modelos, incluyendo sus restricciones. En este artículo se presenta una propuesta para involucrar restricciones en el modelo de clases de UML empleando para ello el ATOM³, una herramienta de metamodelado.

Palabras claves: Metamodelo, restricciones, ATOM³, diagrama de clases de UML.

Abstract

Modeling for software development has been traditionally made by CASE tools. In these tools, formalisms for every model are completely defined, which implies it's not possible adding new restrictions. Meta-modeling tools emerge like a solution for this problem, because they have their own formalisms (generally in graphic environment) allowing different model expressions, including restrictions. In this paper a proposal for involving restrictions in UML class model using ATOM³ (a meta-modeling tool) has been presented.

Key words: Meta-model, restrictions, ATOM³, UML class diagram.

Fecha de recepción: 9 de febrero de 2005
Fecha de aceptación: 8 de junio de 2005

* 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.

** Ingeniero Civil. Magister en Ingeniería de Sistemas y candidato a Doctor en Ingeniería con éfansis en Sistemas, Universidad Nacional de Colombia. Dirección: Cra. 80 N° 65-223, Bloque M8, Facultad de Minas, Universidad Nacional, Medellín (Colombia). *cmzapata@unalmed.edu.co*

*** Ingeniero de Sistemas e Informática, Universidad Nacional de Colombia. *caalvare@unalmed.edu.co*

**** Ingeniero Civil, Universidad Nacional de Colombia. Master en Ciencias y candidato a Doctor, Colorado State University. Ph.D., Universidad Politécnica de Valencia. Dirección: Cra. 80 N° 65-223, Bloque M8, Facultad de Minas, Universidad Nacional, Medellín (Colombia) *farango@unalmed.edu.co*

1. INTRODUCCIÓN

Desde el surgimiento de la Ingeniería de Software como enfoque metodológico para el desarrollo de *software*, la elaboración de las diferentes piezas de *software* se ha fundamentado en una serie de modelos que van paulatinamente refinando los requisitos de los interesados (*stakeholders*), usualmente en lenguaje natural, hasta una especificación del *software* en la plataforma de implementación. El surgimiento del Lenguaje Unificado de Modelamiento, UML [1], definió un conjunto de diagramas que se han convertido en los estándares de facto en el desarrollo de *software*. Para dar apoyo a la elaboración de estos diagramas, han surgido diferentes herramientas CASE (Computer Aided Software Engineering), tales como Rational Rose®, ArgoUML® o Poseidón®. Las herramientas CASE ofrecen, entre otras, facilidades de dibujo para elaborar los diagramas y facilidades de análisis para verificar el grado de corrección de los mismos.

Una limitación importante de las herramientas CASE es que permiten manipular sólo un conjunto limitado de tipos de diagrama y verificar sólo un conjunto limitado de restricciones sobre los diagramas de estos tipos.

Como una respuesta a estas limitaciones han surgido herramientas CASE con facilidades de meta-modelado (habitualmente denominadas META-CASES), tales como Generic Model Environment – GME [2], [3], el DOME [4], el ATOM^3 [5, 6] y el MetaView [7], entre otros. Los META-CASES proveen formalismos para la especificación de los diferentes tipos de diagrama que van a ser elaborados con la herramienta. Con una herramienta META-CASE es posible describir de manera completa un tipo de diagrama cualquiera, de interés para un problema específico. Una vez el tipo de diagrama ha sido descrito, la herramienta se puede usar para la elaboración de instancias del mismo.

Las herramientas META-CASE actuales fueron, sin embargo, concebidas con miras a facilitar el dibujo de los modelos especificados, y poco se ha elaborado con respecto a las facilidades requeridas para la definición de las restricciones que deben verificarse en los diagramas de los tipos definidos.

En este artículo se presenta una propuesta para definir las restricciones asociadas con el diagrama de clases del Modelo UML, planteadas en el estándar del OMG [1], en el marco de las facilidades existentes en el META-CASE ATOM^3 [5, 6]. Más concretamente, se presenta la manera de especificar las restricciones de herencia cíclica y de unicidad de los nombres de los atributos propios o heredados al interior de una clase. El efecto de especificar la restricción será, en el primer caso, el de impedir su violación desde la construcción misma

del diagrama, mientras que en el segundo caso será el de sugerirle al usuario su corrección después de pulsado un botón que induce el chequeo de las restricciones).

Este artículo está organizado de la siguiente manera: en la sección 2 se presenta una breve reseña de las facilidades ofrecidas para la verificación de las restricciones en las especificaciones del *software* (o Esquemas Conceptuales) en las más representativas herramientas CASE sobre UML, Gestores de bases de datos orientados por objetos y META-CASEs del estado del arte; en la sección 3 se introducen las herramientas de Meta-modelado en general y el ATOM^3 en particular; en la sección 4 se describe la propuesta para el manejo de las restricciones en ATOM^3; finalmente, en la sección 5 se presentan algunas conclusiones y trabajos futuros.

2. ENFOQUES PARA EL MANEJO DE LAS RESTRICCIONES SOBRE ESQUEMAS CONCEPTUALES

2.1. **Herramientas CASE convencionales**

Sommerville [8] define las herramientas CASE como sistemas de *software* que tienen la intención de proveer soporte automático para las actividades del proceso de desarrollo de *software*. Bajo esta definición se pueden agrupar diversas herramientas que van desde ayudantes para la construcción de gráficos, como el Microsoft Visio®, hasta herramientas que permiten la elaboración de modelos y su conversión a piezas más o menos completas de código ejecutable, como es el caso de productos como el ArgoUML® o el Rational Rose®.

La característica fundamental de las herramientas CASE es que sólo se pueden elaborar los diagramas que vienen por defecto definidos internamente en la herramienta de modelamiento, excepto herramientas como el Microsoft Visio®, en la cual se pueden importar plantillas de otros diagramas. En una herramienta típica de diagramas UML, tal como el ArgoUML®, el analista puede elaborar diagramas de casos de uso, clases, transición de estados, colaboración o secuencias, por mencionar sólo algunos. Por tener tipos de diagramas definidos internamente, no le es posible al analista realizar modificaciones sobre estos tipos, limitando de este modo los diferentes elementos que se hayan definido en cada tipo de modelo. Entre estos elementos se incluyen las diferentes restricciones que pueden ser verificadas en los diferentes diagramas que considere la herramienta.

Una de las herramientas que maneja las restricciones es ArgoUML®, la cual contiene un módulo denominado "Críticos", donde incluye los principales problemas de violación de restricciones que se pueden presentar en la elaboración de instancias de cada uno de los tipos de diagrama que se pueden elaborar en ella. Este módulo verifica para cada elemento construido las violaciones a las restricciones que se pueden estar presentando, categorizándolas en altas, medianas o bajas y las presenta por pantalla al usuario para que él, asistido en algunos casos por un ayudante de la herramienta misma, realice las correcciones pertinentes. A modo de ejemplo, en la figura 1 se muestra una interfaz que incluye una instancia del diagrama de clases de UML con una violación a la restricción de herencia circular. En la parte inferior izquierda de la interfaz se presenta la manera como la herramienta sugiere al usuario la realización de las correcciones pertinentes que conduzcan a la eliminación de la violación correspondiente al diagrama.

En productos de este tipo, sin embargo, no es posible agregar nuevas restricciones que no estén incluidas en el formalismo del diagrama. Por ejemplo, en el caso del ArgoUML® no se manejan algunas de las restricciones correspondientes a la versión 2.0 de UML, puesto que en su formalismo se incluyeron las restricciones correspondientes a la versión 1.5 de UML.

2.2. **Gestores de bases de datos orientados por objetos**

Los modernos Gestores de bases de datos orientados por objetos (GBDOO) incorporan facilidades para llevar a cabo, de forma automática, gran parte del proceso de "evolución del Esquema Conceptual", o proceso reiterado de modificar la especificación de la base de datos orientada por objetos (el "Esquema Conceptual") y adaptar, en correspondencia, los datos y los programas asociados [9]. Para garantizar que el Esquema Conceptual se mantiene correcto frente a las modificaciones, dichos gestores procuran satisfacer las "invariantes del esquema", que no son otra cosa que conjuntos de restricciones consideradas como condiciones necesarias y suficientes para garantizar la corrección del esquema [10].

Algunos de los trabajos que se han realizado en el manejo de invariantes para la evolución de los esquemas conceptuales son:

- Proyecto ORION [9]: Posee un GBDOO que contiene un conjunto de invariantes que buscan tener la Base de Datos en estados consistentes. Esas invariantes se refieren al manejo de las clases y la herencia y el dominio. El GBDOO evalúa las invariantes y realiza las correcciones necesarias para que la Base

de Datos pase a un estado consistente, incluyendo la propagación de todos los cambios sobre la Base de Datos. Si bien la Base de Datos queda en un estado consistente, los cambios se realizan de forma no supervisada, lo que podría generar estados consistentes pero poco deseables de los datos.

- NO^2 [11]: Define un conjunto de invariantes que posibilitan el control de los cambios en un punto del Esquema Conceptual y sus efectos sobre el resto del mismo; como en el caso de ORION, las invariantes se refieren a las clases y la herencia y se agrega la integridad referencial de la base de datos. Si bien no poseen un GBDOO implementado, proponen señalar las inconsistencias para que sea el usuario el que realice las correcciones necesarias, antes de aceptar los efectos de las invariantes.

- COCOON [12, 13]: Utiliza un "meta-esquema" en el cual se realizan restricciones sobre los atributos y las relaciones, incluyendo aciclicidad de las relaciones subclase-superclase, unicidad de nombres e integridad referencial. En esta propuesta no se aclara cuál sería el mecanismo de corrección.

- MAGIC [10]: Incluye un GBDOO basado en OASIS y con un metamodelador que posibilita la definición de restricciones y evaluación. La propuesta de este trabajo es admitir cualquier cambio que se plantee en las restricciones, pero sin propagar sus efectos sobre la Base de Datos y señalando las inconsistencias para que sea el usuario quien realice las correcciones pertinentes.

Tanto ORION como MAGIC poseen implementaciones que permiten elaborar invariantes y verificar su cumplimiento en una base de datos, en tanto que NO^2 y COCOON son planteamientos conceptuales y teóricos. Además, todos los proyectos se plantean en el dominio de los Modelos de Datos y no se trabajan para otros tipos de modelos, como los funcionales.

2.3. **Restricciones en herramientas de Metamodelado**

En la actualidad, el grupo de investigación UN-INFO viene trabajando en la definición de restricciones en algunas herramientas de metamodelado, tales como ATOM^3 [5 y 6], que se describe en este artículo, y DOME [4]. Se está iniciando la exploración de otras herramientas como GME y MetaView. Las ventajas de este enfoque se discutirán de manera más precisa en la sección 4.

Adicionalmente, en el marco del proyecto de investigación que fundamenta este trabajo, se viene desarrollando una herramienta META-CASE orientada a dar solución a los problemas tanto de las herramientas CASE como de los GBDOO

mencionados, en el sentido de que se orienta al Modelo y no a los diagramas (entendiendo el modelo como un conjunto de diagramas que describen el problema). Esta herramienta se ha denominado provisionalmente UN-MetaCASE.

En la siguiente sección se introduce el uso de herramientas de Metamodelado y el AT o M^3 en particular.

3. LAS HERRAMIENTAS DE METAMODELADO Y EL AToM^3

De Lara, Vangheluwe y Alfonseca [14] definen el Metamodelado como el "proceso de modelamiento de formalismos", el cual se suele realizar para determinar si una instancia de un modelo particular es consistente con su especificación en forma de metamodelo [15]. Las herramientas de Meta-modelado han surgido como alternativas a las herramientas CASE convencionales, con el fin de permitir a sus usuarios la creación de nuevos formalismos para diagramas que no se encuentren disponibles en la herramienta o para la complementación de las diferentes restricciones y reglas de consistencia internas de los diagramas que se pueden elaborar en la herramienta.

Entre las diferentes herramientas de Metamodelado disponibles se encuentra el ATOM^3 (A Tool for Multi-Formalism Modeling and Meta-Modeling), que fue desarrollado en el lenguaje de programación Python [5]. ATOM^3 posee un formalismo inicial para la especificación de modelos que se basa en el modelo entidad relación extendido con restricciones; ese formalismo tiene una interfaz gráfica que facilita la construcción de las especificaciones del diagrama de manera similar a como se elaboran las instancias de un diagrama en una herramienta CASE convencional. Esta especificación se puede cargar posteriormente para su uso en la materialización de instancias que representen el dominio de un problema particular.

Como ventajas principales de las herramientas de Metamodelado sobre las herramientas CASE convencionales, se pueden mencionar las siguientes:

- Se puede iniciar la construcción de formalismos de modelos desde cero, lo cual implica la posibilidad de correcciones (adición o borrado de elementos, por ejemplo) en modelos que cambian constantemente de versión o en otros que presentan modificaciones sutiles frecuentemente.

- Facilidad gráfica de expresión del formalismo de los diferentes modelos, lo que reduce la complejidad en la comprensión de los mismos.

- Posibilidad de creación de instancias de los modelos y de verificación de las restricciones planteadas en el meta-modelo sobre esas instancias en particular.

ATOM^3 soporta un formalismo adicional que le permite expresar ciertas restricciones; ese formalismo se conoce como Gramática de Grafos, la cual tiene similitudes con las reglas de inferencia de los lenguajes declarativos en que pueden ser usadas para definir un conjunto de acciones que se va a realizar a partir del cumplimiento de unas precondiciones. La gramática de grafos puede combinar la expresión gráfica con la textual, en forma de pre y postcondiciones que pueden ser establecidas en el lenguaje Python. Las gramáticas de grafos se definen como un conjunto de reglas que poseen un lado izquierdo (*left-hand side* o LHS) que contiene las precondiciones que deben ser cumplidas para activar una determinada regla y un lado derecho (*right-hand side* o RHS) que contiene el grafo que reemplazará el que equipare el lado izquierdo de la regla. [14].

En la siguiente sección se presenta la propuesta para el manejo de las restricciones usando los diferentes mecanismos que provee el ATOM^3 como la Gramática de Grafos y las restricciones en los diferentes eventos asociados con los elementos de metamodelado.

4. PROPUESTA PARA EL MANEJO DE LAS RESTRICCIONES EN AToM^3

4.1. **Definición del Metamodelo de Clases en el formalismo de AToM^3**

Para efectos de esta propuesta se elaboró en el formalismo de ATOM^3 una versión reducida del diagrama de clases de UML, pues el objetivo es la ejemplificación del manejo de restricciones y no el manejo de la complejidad que puede tener el diagrama de clases en su versión completa.

En la figura 2 se muestra el metamodelo construido. Los rectángulos son entidades y los rombos relaciones. Como elementos básicos se definen:

- *Class*: Es una entidad que corresponde a una clase del diagrama de clases y que incluye los siguientes atributos: un nombre (*name*) de tipo string, una lista de atributos (*attibutes*), una lista de operaciones (*methods*), una lista de atributos heredados (*InheritanceAttributes*) y una lista de operaciones heredadas (*Inheritancemethods*). En esta entidad se incluyen también dos restricciones: HerenciaCiclica y CargarAtributosHeredados, de las cuales se hablará en el numeral siguiente.

- *Inheritance*: Es una relación correspondiente a la herencia de un diagrama de clases. No tiene atributos ni operaciones.

- *Notes*: Es una entidad que corresponde a las notas para la expresión de restricciones y mensajes en el diagrama de clases. Esta entidad tiene un listado de los atributos repetidos (*attributes*, de tipo lista) y un mensaje (*message*) de tipo string. Esta entidad se utiliza como mecanismo para indicar al usuario los atributos que se repiten en una clase.

- *Explanation*: Es una relación que corresponde a la asociación de la nota a una clase.

4.2. La herencia cíclica expresada como una postcondición en la creación de conexiones

Dentro de la entidad "Class" se añadió como restricción "HerenciaCiclica", que es un procedimiento expresado en lenguaje Python elaborado para realizar el control de la herencia cíclica, la cual establece que "Una clase no puede heredar directa ni indirectamente de sí misma" [1]. El de AToM^3 permite establecer el momento de verificación de la restricción, definiéndola como pre o postcondición, y definiendo un evento asociado (edit, save, create, connect, delete, etc.). En este caso, el código Python generado para la restricción se eligió como una postcondición que se ejecuta cuando se va a intentar la conexión de una clase con otra (evento "connect"). El código correspondiente se muestra a continuación; se han añadido algunos comentarios resaltados para facilidad de comprensión del mismo:

```
#Definición de una función para obtener sólo relaciones del tipo herencia (inheritance):
def filtrar(x): return x.__class__.__name__=="Inheritance"
#con la siguiente pieza de código se devuelven todos los 'Class' del modelo
for obj1 in self.parent.ASGroot.listNodes['Class']:
obj2 = obj1
clases = [obj2]
pos = len(clases)
while len(filter(filtrar, obj2.out_connections_)) > 0:
#Con las siguientes instrucciones se devuelven todos los 'Inheritance' del 'Class' seleccionado:
for obj3 in filter(filtrar, obj2.out_connections_):
if(pos != len(clases)):
clases = clases[:pos]
#Finalmente, con las siguientes instrucciones se devuelven todos los 'Class' conecta-
```

```
dos a los 'Inheritance' anteriores
for obj4 in obj3.out_connections_:                    if(obj4 in clases):
ciclo = "["
for i in (clases[clases.index(obj4):] + [obj4]):
ciclo = ciclo + i.Name.toString() +", "
ciclo = ciclo[:len(ciclo)-2] +"]"
return("Herencia Ciclica no permitida: " + ciclo, self)
else: clases.append(obj4)
obj2 = obj4
pos = len(clases)
return None
```

En la figura 3 se muestra el efecto de la aplicación de esta pieza de código durante la creación de las diferentes clases de un modelo. Allí se muestra lo que ocurre en el ATOM^3 cuando se invoca la creación del mismo modelo de la figura 1. Al tratar de realizar la conexión con la tercera clase se genera un mensaje invocado por la pieza de código anterior.

4.3. **El manejo de los atributos repetidos para una clase empleando la Gramática de Grafos**

En los diagramas de clases de UML, los nombres de los atributos al interior de una clase deben ser únicos. Para efectos de demostrar la aplicabilidad de los formalismos de ATOM^3 en la verificación de restricciones para un diagrama particular, se hará extensiva la restricción planteada para los atributos heredados de una clase, lo cual implica que una clase no puede tener nombres repetidos en los atributos ya sean estos propios o heredados.

En esta propuesta se utilizan las restricciones de creación de elementos y la gramática de grafos del ATOM^3 para implementar la restricción de unicidad de los atributos. Inicialmente se agregó la restricción "CargarAtributosHeredados" a la entidad "Class". Esta restricción verifica que el nombre de cada uno de los atributos sea válido, es decir que se conforme como una combinación de letras, números y *underscore* ("_") y que comience con una letra o *underscore*; adicionalmente, actualiza todos los atributos heredados de las clases. Cada nombre de atributo heredado se compone del nombre de la clase seguido por el símbolo "$" y por el nombre del atributo. En este caso, el código Python generado para la restricción se eligió como una postcondición que se ejecuta cuando se va a intentar la conexión o desconexión de una clase con otra (eventos "connect" y "disconnect" respectivamente) o la edición de una clase (evento "edit").

El código Python de esa función es el siguiente:

```
#Definición de una función para obtener sólo relaciones del tipo herencia (inheritance):
def filtrar(x): return x.__class__.__name__=="Inheritance"
#Procedimiento que se encarga de asignar los atributos heredados a cada elemento 'Class' del modelo
def AtributosHeredados():
    for obj1 in self.parent.ASGroot.listNodes['Class']:
        atributosHeredados = []
        clases = ObtenerClases(obj1)
        for obj5 in clases:
            for atr in obj5.Attributes.getValue():
                atributosHeredados.append(ATOM3String(obj5.Name.toString() + "$" + atr.toString()))
        if len(atributosHeredados) > 0:  obj1.InheritanceAttributes.setValue(atributosHeredados)
    return None
#Función recursiva para la obtención de las superclases de cada clase del modelo
def ObtenerClases(obj2):
    clases = []
    if len(filter(filtrar, obj2.in_connections_)) > 0:
        for obj3 in filter(filtrar, obj2.in_connections_):
            for obj4 in obj3.in_connections_:
                clases.append(obj4)
            clases2 = []
            if obj4: clases2 = ObtenerClases(obj4)
            if len(clases2) > 0: clases = clases[:] + clases2
    return clases
#Procedimiento para verificar si el nombre de un atributo es válido, es decir aquel que comienza con una letra e incluye letras, números y/o "_"
for obj1 in self.Attributes.getValue():
    vname = obj1.toString()
    if (not vname) or (vname == ""):
        return "El Nombre del Atributo debe ser especificado", self
    if string.count(vname, " ") > 0:
        return "Nombre de Atributo invalido: No se permiten espacios en blanco", self
    if (vname[0] >= '0') and (vname[0] <= '9'):
        return "Nombre de Atributo invalido: El primer caracter debe ser una letra o '_'", self
    if vname[0] != '_' and (vname[0]<'A' or vname[0]>'z'):
        return "Nombre de Atributo invalido: El primer caracter debe ser una letra o '_'", self
    for c in range(len(vname)-1):
```

```
        if vname[c+1] < 'A' or vname[c+1] > 'z':
            if vname[c+1] < '0' or vname[c+1] > '9':
                if vname[c+1] != '_':
                    return "Nombre de Atributo invalido: El caracter '"+vname[c+1]+"' no
esta permitido", self
    return AtributosHeredados()
```

Para activar la verificación correspondiente a la restricción de unicidad de nombre de los atributos en una clase se utilizó la Gramática de Grafos de la siguiente manera:

Se definió una regla llamada buscarAtributosRepetidos, que tiene como LHS y RHS las componentes mostradas en la figura 4. En el LHS se establece que cualquier elemento del tipo "Class" se seleccionará para realizar la verificación y en el RHS se establece la realización de una copia de cada elemento seleccionado por el LHS, que se colocará en el modelo de destino (recuérdese que la Gramática de Grafos es una gramática de transformación, lo que implica que se toma un diagrama origen y se transforma en un diagrama destino, que en este caso es el mismo diagrama con la restricción ya verificada).

Dentro de la regla definida se incluyeron también una condición, que define cuáles elementos seleccionados por el LHS sufrirán la transformación definida por el RHS, y una acción, que construye las entidades "Notes" correspondientes a cada entidad "Class" que viola la restricción de unicidad de los atributos propios o heredados. En cada entidad "Notes" generada se incluirán en el mensaje los atributos repetidos para cada entidad "Class" asociada. Tanto la condición como la acción definidas para esta regla emplean las siguientes funciones:

```
#Convierte una lista de tipo ATOM3 String al tipo String de Python:
def toString(obj1):
    obj2 = []
    for obj3 in obj1:
        obj2.append(obj3.toString())
    return obj2
#Elimina el nombre de la Clase de los atributos heredados:
def filtrar(obj1):
    obj2 = []
    for obj3 in obj1:
        pos = obj3.toString().find("$") + 1
        x = ATOM3String(obj3.toString()[pos:])
        obj2.append(x)
```

```
    return obj2
#Obtiene los atributos repetidos para una clase particular:
def getAtributosRepetidos(obj1):
    obj2 = filtrar(obj1.InheritanceAttributes.getValue()) + obj1.Attributes.getValue()
    obj3 = [obj2[0]]
    obj4 = []
    for obj5 in obj2[1:]:
        if obj5.toString() in toString(obj3):
            obj4.append(obj5)
        else:
            obj3.append(obj5)
    return obj4
```

El código Python correspondiente a la condición es el siguiente:

```
#Define una función para relaciones del tipo "Explanation":
def filtrar2(x):
   return x.__class__.__name__=="Explanation"
#Obtención del elemento seleccionado por el LHS:
n1 = self.getMatched( graphID, self.LHS.nodeWithLabel(1))
#Constata que no se vuelva a verificar un elemento ya verificado:
if len(filter(filtrar2, n1.in_connections_)) > 0: return 0
obj1 = getAtributosRepetidos(n1)
if len(obj1) > 0: return 1
return 0
```

Finalmente, el código Python correspondiente a la acción es el siguiente:

```
#Importa las funciones necesarias para la creación de entidades "Notes" y relaciones "Explanation":
from ClassModel_MM import createNewNote, createNewExplanation
#Obtención del elemento seleccionado por el LHS:
n1 = self.getMatched( graphID, self.LHS.nodeWithLabel(1))
#Obtención de los elementos repetidos para la clase seleccionada:
obj1 = getAtributosRepetidos(n1)
if len(obj1) > 0:
#Creación de la entidad "Notes" asociada con esta clase:
    n2 = createNewNote(atom3i, n1.graphObject_.x-180, n1.graphObject_.y, 1)
    n2.Message = ATOM3String("Existen " + str(len(obj1)) + " Atributos repetidos:")
    n2.Attributes.setValue(obj1)
    n2.graphObject_.ModifyAttribute("Message", n2.Message.toString())
    n2.graphObject_.ModifyAttribute("Attributes", n2.Attributes.toString(30, 5))
    x = n2.graphObject_.x + 170
    y = n2.graphObject_.y + 170
```

```
    #Creación de la relación «Explanation»:
        flujo = createNewExplanation(atom3i, x, y, 1)
    #Conexión de la clase seleccionada con la entidad "Notes" creada por medio de la
relación "Explanation" creada:
        atom3i.drawConnections((n2, flujo), (flujo, n1))
    return None
```

En la figura 5 se muestra un ejemplo al cual se le aplica la transformación descrita en esta sección y en la figura 6 el efecto de la aplicación de esta regla en gramática de grafos, después de presionar el botón "Buscar atributos repetidos". El botón "Eliminar notas" sirve para restaurar el diagrama verificado en el diagrama anterior a la verificación.

CONCLUSIONES Y TRABAJOS FUTUROS

- En este artículo se muestra con un caso de estudio cómo el ATOM^3, a pesar de no haber sido diseñado para el manejo de las restricciones en los tipos de diagrama que especifica, tiene facilidades que permiten llevar a cabo esta tarea. Tales facilidades permiten superar las limitaciones de las herramientas CASE convencionales y de GBDOO descritos para la Evolución de Esquemas Conceptuales, ya que puede involucrar restricciones que no necesariamente están definidas para la sintaxis particular de un diagrama, y pese a que se realizó la ejemplificación con el diagrama de clases, es posible realizar este manejo para cualquier tipo de diagrama, no sólo para Modelos de Datos.

- Uno de los mecanismos de ATOM^3 que facilita la incorporación de restricciones en los diferentes modelos es la Gramática de Grafos, una manera gráfica de expresar y verificar reglas de consistencia. Sin embargo, ese mecanismo no fue suficiente para expresar las restricciones, haciendo necesaria la incorporación de postcondiciones en el lenguaje de programación Python, claro que sin abandonar el entorno del ATOM^3. Debido a lo anterior, se requiere experiencia en el lenguaje Python para la incorporación de las restricciones en el metamodelo.

- La conjugación de la Gramática de Grafos con el uso de las postcondiciones en Python constituye la mayor fortaleza del ATOM^3 para la tarea específica de incorporar restricciones en los metamodelos de los diferentes diagramas. Ello, aunado al hecho de que se puede describir en ATOM^3 el metamodelo cualquier tipo de diagrama, le entrega a esta herramienta la capacidad de representar casi cualquier restricción en cualquier tipo de diagrama.

- Se debe hacer la salvedad de que ATOM^{3}es una herramienta de metamodelado orientada a los diagramas y no al modelo de un problema como tal (entendiendo el modelo como un conjunto de diagramas que describen el problema). Ello implica que se pueden especificar las restricciones para un tipo de diagrama pero no para un modelo; tales restricciones son realmente reglas de consistencia intermodelos. Esas reglas se podrían expresar en Gramática de Grafos y postcondiciones en Python, siempre y cuando se pueda construir un metamodelo que incluya, en un solo diagrama, todos los diferentes diagramas que conforman el modelo.

Como futuros trabajos que se podría realizar a partir de este artículo se plantean:

- La complementación del metamodelo del diagrama de clases de UML expresado en ATOM^3 y la definición en el formalismo de ATOM^3 de las diferentes restricciones que hacen parte de la versión 2.0 de UML.

- De forma análoga a la propuesta realizada en este artículo se pueden abordar otros diagramas UML o incluso otros diagramas que no pertenecen a UML pero que son fundamentales para el modelamiento de los problemas inherentes a una pieza de *softwar*e, como el diagrama de objetivos o el diagrama causa - efecto.

- Se plantea igualmente la construcción del metamodelo en ATOM^3 de un modelo que incluya varios diagramas de UML simultáneamente, sobre el cual se puedan aplicar reglas de consistencia intermodelos.

- Es importante también la exploración de los mecanismos de incorporación de restricciones en otras herramientas META-CASE, tales como DOME, GME o MetaView.

- Un trabajo futuro en el cual está comprometido el grupo UN-INFO es la culminación del desarrollo del UN-Meta CASE, con las características anotadas, que posibilitará la elaboración cualquier tipo de modelo y el chequeo de sus restricciones.

Referencias

[1] OMG. OMG Unified Modeling Language Specification. Object Management Group. Available: http://www.omg.org/UML/. [Citado 8 de diciembre de 2004].
[2] LEDECZI, A., MAROTI, M., BAKAY, A., KARSAI, G., GARRETT, J., THOMASON, IV C., NORDSTROM,

G., Sprinkle, J. & Volgyesi, P. (2001). *The Generic Modeling Environment*. Proceedings of the Workshop on Intelligent Signal Processing Budapest.

[3] Sprinkle, J. & Karsai, G. (2004). A Domain-Specific Visual Language For Domain Model Evolution. *Journal of Visual Languages and Computing*, vol. 15, Nº 2.

[4] Dome. What is Dome. Available: http://www.htc.honeywell.com/dome/description.htm. [Citado 8 de diciembre de 2004].

[5 De Lara, J. & Vangheluwe, H. (2002). AToM^3: A tool for Multi-Formalism and Meta-Modelling. *Proceedings of the Fifth Ingernational Conference on Fundamental Approaches to Software Engineering* (p. 174-188). Grenoble.

[6] Atom^3. Msdl – Atom^3. Página Web de la herramienta atom^3. Available: http://atom3.cs.mcgill.ca/. [Citado 8 de diciembre de 2004].

[7] Metaview Project. Página Web de la herramienta metaview. Avalaible: http://www.cs.ualberta.ca/~softeng/Metaview/project.html. [Citado 8 de diciembre de 2004].

[8] Sommerville, I. (2001). *Software Engineering*. Massachussetts, Addison-Wesley.

[9] Banerjee, J., Kim, W., Kim, H.J. & Korth, H. F. (1987). Semantics and Implementation of Schema Evolution in Object-Oriented Databases. *Proceedings of the 1987 acm sigmod (Special Interest Group on Management of Data) International Conference on Management of data* (p. 311-322). San Francisco, California.

[10] Arango, F. (2003). Gestión de las Inconsistencias en la evolución e interoperación de los esquemas conceptuales OO en el marco formal de oasis. Tesis doctoral, Universidad Politécnica de Valencia.

[11] Scherrer, S., Geppert, A. & Dittrich, K. (1993). Schema Evolution in NO_2. Institut für Informatik Universität Zürich, *Technical Report* N° 93.12.

[12] Tresch, M. (1991). A Framework for Schema Evolution by Meta Object Manipulation. *Proceedings of the 3rd International Workshop on Foundations of Models and Languages for Data and Objects*. Aigen, Austria.

[13] Tresch, M. & Scholl, M. (1992). Meta Object Management and its Application to Database Evolution. *Proceedings of the 11th International Conference on the Entity Relationship Approach*. Karlsruhe, Alemania.

[14] De Lara, J., Vangheluwe, H. & Alfonseca, M. (2003). *Using Meta-Modelling and Graph-Grammars to Create Modelling Environments*. Electronic Notes in Theoretical Computer Science, vol. 72, N° 3.

[15] Sourrouille, J. & Caplat, G. (2003). A pragmatic View about Consistency Checking of uml Models. *Proceedings of the Workshop on Consistency Problems in uml-Based Software Development* (p. 43-50). San Francisco.

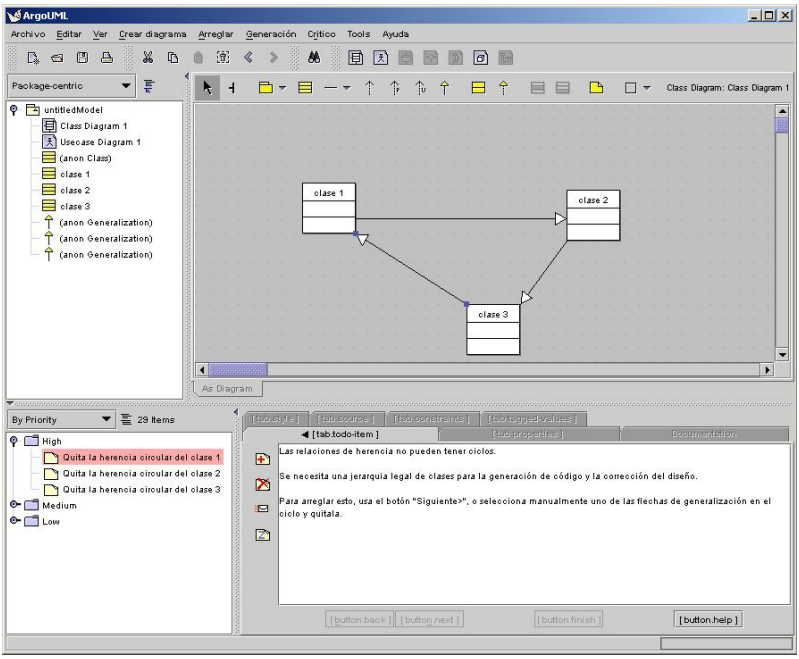

Figura 1. Imagen de un modelo en ArgoUML® con el manejo de la herencia circular para una instancia del Diagrama de Clases

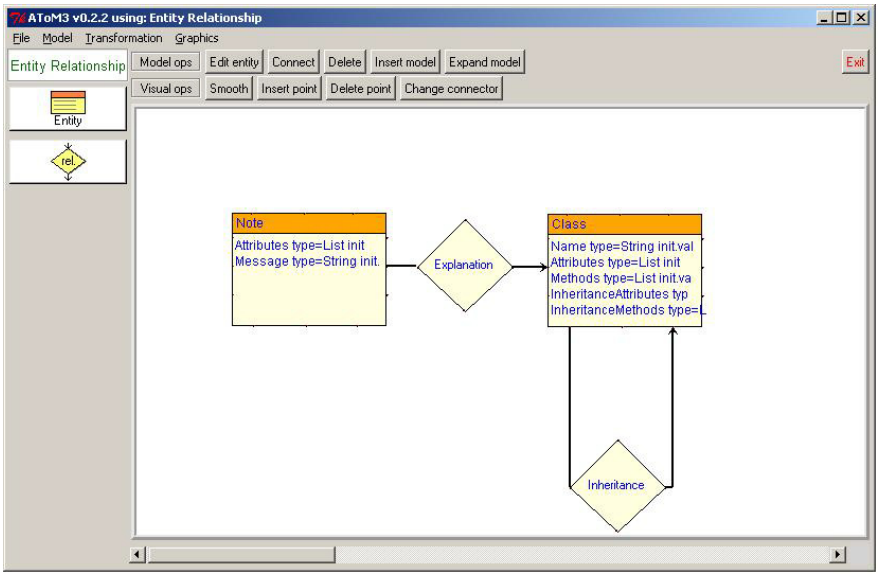

Figura 2. Metamodelo del diagrama de clases en AToM^3 como una versión reducida de la especificación en UML

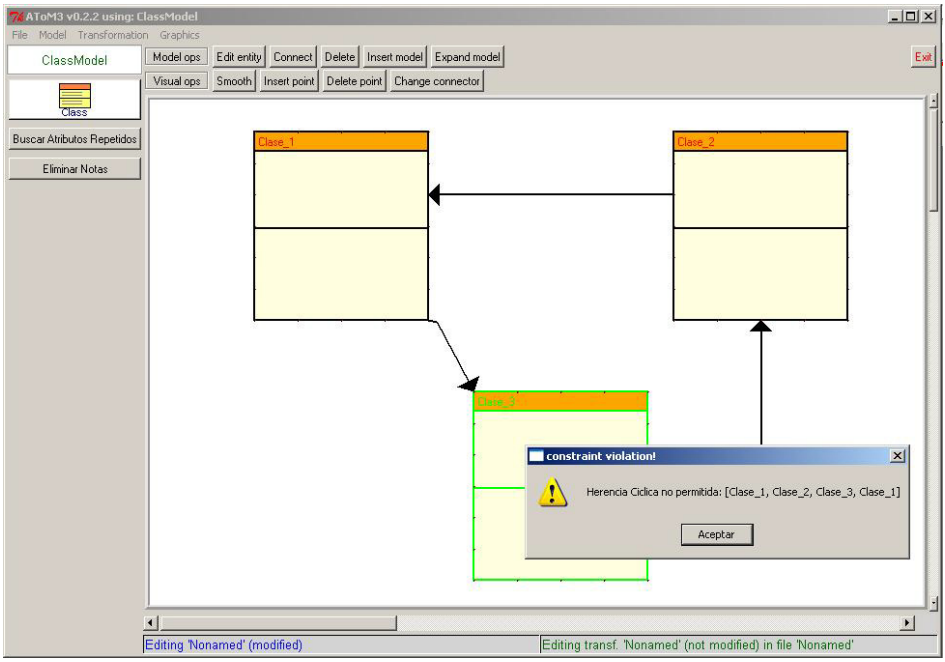

Figura 3. Efecto de la aplicación de la restricción de herencia cíclica para el caso particular de un modelo de clases programado en ATOM^3

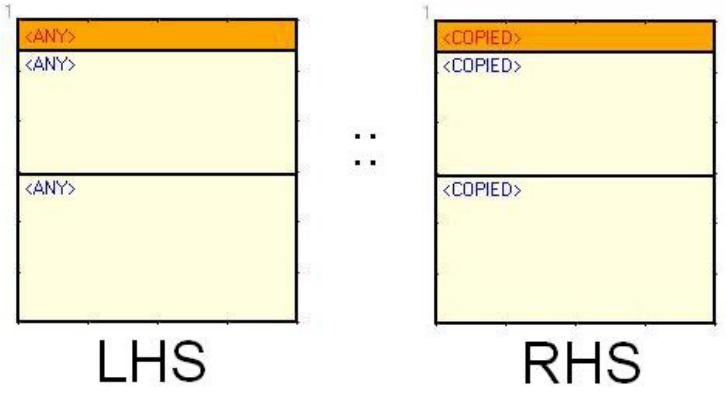

Figura 4. Partes LHS y RHS de la regla buscarAtributosRepetidos

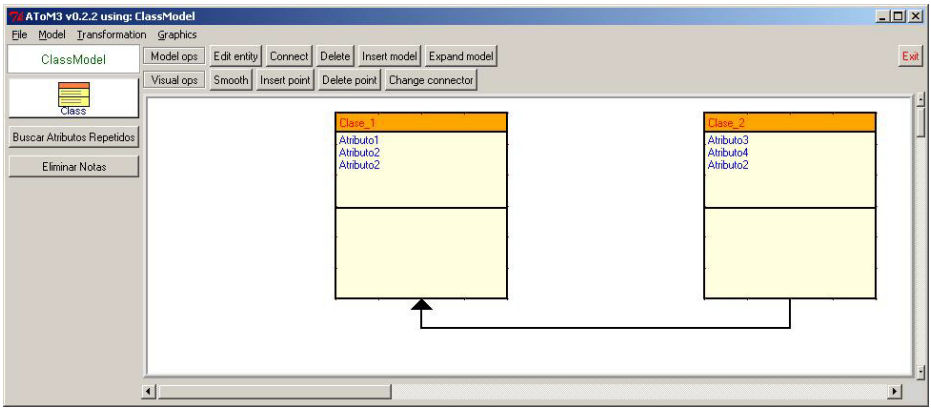

Figura 5. Ejemplo de un diagrama de clases para la verificación de la restricción de unicidad de los atributos propios o heredados

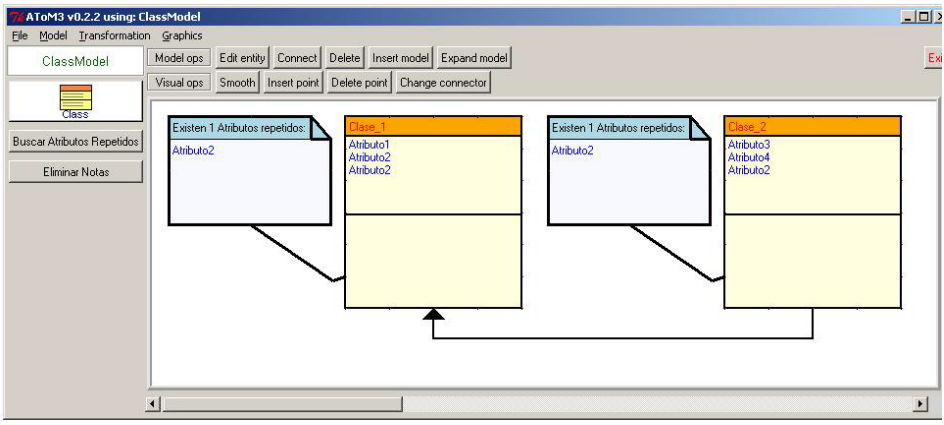

Figura 6. Efecto de la aplicación de la regla de verificación de la restricción sobre el modelo de la figura 5