CIMTool: Una herramienta para la definición de un diagrama de clases UML

María Carmen Leonardi María Virginia Mauco Hernán Leoni
INTIA - Departamento de Computación y Sistemas
Facultad de Ciencias Exactas
Universidad Nacional del Centro de la Pcia. de Buenos Aires
Argentina
{cleonard, vmauco, hleoni}@exa.unicen.edu.ar

Resumen. *Model Driven Architecture* es un *framework* de desarrollo de software cuyo concepto clave es la transformación automática de modelos. Uno de estos modelos, el *Computer Independent Model* (CIM), se usa para definir el modelo del negocio. En este trabajo se presenta CIMTool, una herramienta que implementa un proceso de definición automática del CIM. Este proceso aplica un conjunto de reglas de transformación a modelos de requisitos basados en lenguaje natural derivando un diagrama de clases UML. CIMTool puede integrarse con cualquier herramienta CASE que acepte archivos XML. Así, el CIM derivado puede ser la base para un desarrollo basado en MDA.

Palabras clave: Derivación automática de Modelos de Requisitos, Modelos de Requisitos basados en Lenguaje Natural, *Model Driven Architecture* (MDA), Diagramas de clase UML.

1. Introducción

La *Model Driven Architecture* [1], conocida como MDA, es un *framework* para el desarrollo de software definido por OMG [2] que se basa en la definición de modelos para cada aspecto y nivel de abstracción de un sistema de software. Uno de los aspectos claves de MDA es la transformación automática entre modelos. El primer modelo de un desarrollo MDA es el modelo CIM (*Computer Independent Model*) el cual describe el negocio independientemente del sistema de software que se vaya a implementar. Aunque dentro del contexto de MDA tanto este modelo como su proceso de construcción son los menos estudiados, han aparecido recientemente algunos trabajos relacionados [3]. También pueden mencionarse algunas herramientas que si bien no fueron concebidas en el contexto de MDA, construyen automáticamente modelos UML, en particular [4] deriva diagramas de secuencia a partir de casos de uso y [5] deriva diagramas de clase a partir del modelo organizacional i*.

En trabajos anteriores [6] se ha presentado y formalizado un proceso de derivación que, partiendo de modelos de requisitos basados en lenguaje natural, aplica un conjunto de reglas de transformación para definir un diagrama de clases UML. Este diagrama puede ser considerado como un CIM, ya que representa el negocio y no al sistema de software. Los modelos de requisitos usados [7] son un modelo léxico para representar el lenguaje del negocio y un modelo de escenarios para representar su comportamiento.

En este artículo se presenta CIMTool, una herramienta que implementa este proceso de derivación permitiendo la definición automática de un diagrama de clases UML que puede ser considerado un CIM dentro de un proceso de software basado en MDA. La herramienta genera documentos XML por lo que puede integrarse perfectamente con cualquier CASE de desarrollo orientado a objetos que acepte este formato.

El artículo está organizado de la siguiente manera. La Sección 2 describe brevemente los modelos de requisitos. En la Sección 3 se presenta sintéticamente el proceso de derivación del

diagrama de clases a partir de los modelos de requisitos. En la Sección 4 se detalla la herramienta CIMTool que automatiza el proceso. En la Sección 5 se presentan ejemplos del uso de la herramienta. Finalmente, la Sección 6 presenta algunas conclusiones y trabajos futuros.

2. Modelos de Requisitos en Lenguaje Natural

Los modelos que se presentan brevemente en esta sección son conocidos, usados y aceptados por la comunidad de Ingeniería de Requisitos. Una descripción completa de los mismos puede encontrarse en [7]. Los modelos son:

Léxico Extendido del Lenguaje (LEL): Es una estructura que permite la representación de términos significativos del Universo de Discurso. Está compuesto por un conjunto de símbolos que tienen un nombre y un conjunto de sinónimos, una noción (describe qué es el símbolo) y un impacto (describe cómo repercute el símbolo en el sistema). Los símbolos del LEL definen objetos, sujetos, frases verbales y estados. En la descripción de los símbolos deben seguirse simultáneamente dos reglas: el "principio de circularidad" y el "principio de vocabulario mínimo".

Modelo de Escenarios: Un escenario describe situaciones en el Universo de Discurso. Cada escenario está vinculado con el LEL, y está compuesto por: un título que lo identifica, un objetivo que describe su propósito, un contexto para definir ubicaciones temporales y geográficas, y precondiciones, actores que son entidades involucradas activamente en el escenario (generalmente personas u organizaciones), un conjunto de recursos que identifican entidades pasivas con las cuales los actores trabajan, y un conjunto de episodios donde cada episodio representa una acción realizada por actores usando recursos.

3. El Proceso de Transformación del Modelo de Requisitos al CIM

En esta sección se describe brevemente el proceso automático para obtener el diagrama de clases UML que representa los aspectos estructurales de un CIM. El proceso consiste en un conjunto de pasos que aplican reglas de transformación a los modelos de LEL y escenarios para definir las clases y sus relaciones.

El proceso está organizado en tres etapas que se ejecutan de manera secuencial:

- **Identificación de clases**: las dos reglas definidas para esta etapa, identifican las clases a partir de los símbolos del LEL clasificados como sujetos y objetos.

  TRC1: Regla de Transformación Sujeto-Clase

  Cada sujeto del LEL se transforma en una clase UML. Los atributos de la clase se definen de la siguiente manera: para cada entrada en la noción del símbolo LEL analizado que no referencia a otro símbolo LEL, la regla identifica cada sustantivo y lo define como un atributo de la clase.

  TRC2: Regla de Transformación Objeto-Clase

  Cada objeto del LEL se transforma en una clase UML. Los atributos de la clase se definen de la siguiente manera: para cada entrada en la noción del símbolo LEL analizado que no referencia a otro símbolo LEL, la regla identifica cada sustantivo y lo define como un atributo de la clase. Los métodos de la clase se definen agregando el prefijo GET a cada nombre de atributo para definir un método de acceso y agregando el prefijo SET a cada nombre de atributo para definir un método de modificación.

- **Identificación de métodos**: las dos reglas propuestas para esta etapa definen los métodos (con sus correspondientes parámetros) para las clases provenientes de símbolos del LEL clasificados como sujetos.

  TRM1: Regla de Transformación ImpactoDeSujeto-Método

Cada entrada del impacto de un símbolo del LEL clasificado como sujeto que fue modelado como una clase aplicando TRC1, se modela como un método de esta clase.

TRM2: Regla de Transformación InformaciónSujeto-ParámetrosMétodo

Según el proceso de construcción de escenarios [7], cada entrada del impacto de un símbolo del LEL clasificado como sujeto, se transforma en un escenario. TRM2 modela a los actores y a los recursos de un escenario como parámetros del método definido por TRM1 a partir de la correspondiente entrada del impacto que generó dicho escenario. El actor que referencia al símbolo del LEL que generó el escenario es excluido.

- **Identificación de relaciones**: el diagrama de clases se completa con la definición de relaciones de herencia, agregación y asociación mediante la aplicación de la regla de transformación TRR que analiza la noción de símbolos del LEL que se transformaron en clases.

TRR: Regla de Transformación RelacionesDeLEL-RelacionesDeClase

Esta regla se aplica tanto a símbolos del LEL clasificados como sujetos u objetos que han sido definidos como clases por TRC1 o TRC2. La regla analiza cada entrada de la noción de un símbolo del LEL L1, previamente modelado como clase, con el objetivo de detectar otros símbolos del LEL definidos también como clases por TRC1 o TRC2. Para cada símbolo del LEL detectado, llamado L2, se define una relación de asociación entre las correspondientes clases. El tipo de la relación se determina sobre la base de un patrón lingüístico sugerido por [9] que determina una clasificación de verbos.

Relación de herencia: L1 y L2 tienen la misma clasificación (objeto o sujeto). L1 aparece en la noción de L2. Las entradas de las nociones de L1 y L2 involucradas contienen, de manera complementaria, dos clase de verbos [9]: *bottom-up* (es un, es un tipo de, es una clase de) o *top-down* (es, puede ser, puede ser clasificado como, clasifica como).

Relación de agregación: en la noción del símbolo considerado como Contenedor deben aparecer verbos del tipo *component-composition*: consistir, contener, tener, poseer, incluir, formar, componer, dividir (estos tres últimos en voz pasiva). Asimismo en la noción del símbolo correspondiente a la clase Componente deberán aparecer verbos del tipo *content-composition*: forma parte, pertenece, es un componente, esta incluido en, entre otros. Debido a que no es posible distinguir automáticamente entre una relación de agregación o composición, la regla define a la relación como una agregación.

Relación de Asociación: cualquier relación entre símbolos del LEL que no represente a ninguna de las dos relaciones previas, se considera como una relación de asociación. El verbo que aparece en la noción, clasificado por [9] como *general*, se toma como el nombre de la asociación.

Estas reglas de transformación asumen decisiones fijas de diseño, por lo que el diagrama de clases UML resultante de su aplicación debe ser revisado y mejorado por los ingenieros de software. Una descripción completa del proceso que incluye una formalización en OCL [8] de las reglas de transformación se encuentra en [6].

4. Herramienta CIMTool

CIMTool permite la ejecución automática del proceso presentado en la Sección 3 generando un CIM que será el primer modelo para un desarrollo basado en MDA. La Figura 1 describe la herramienta y su contexto. CIMTool toma como entrada dos archivos: uno con los modelos de LEL y Escenarios y otro con un diccionario del lenguaje utilizado en la descripción de los modelos de requisitos. La salida son dos archivos: el archivo con la especificación XMI del diagrama de clases y un archivo de log con información del resultado del proceso. El archivo XMI respeta el formatoXMIv1.2 [10] lo que permite integrar el resultado de esta herramienta con otras

herramientas CASE. Por ejemplo podemos citar las herramientas Poseidón Comunity Edition v2.6 [11] y Enterprise Architect v4.0 [12], que fueron utilizadas para evaluar los resultados obtenidos en las ejecuciones de la herramienta.

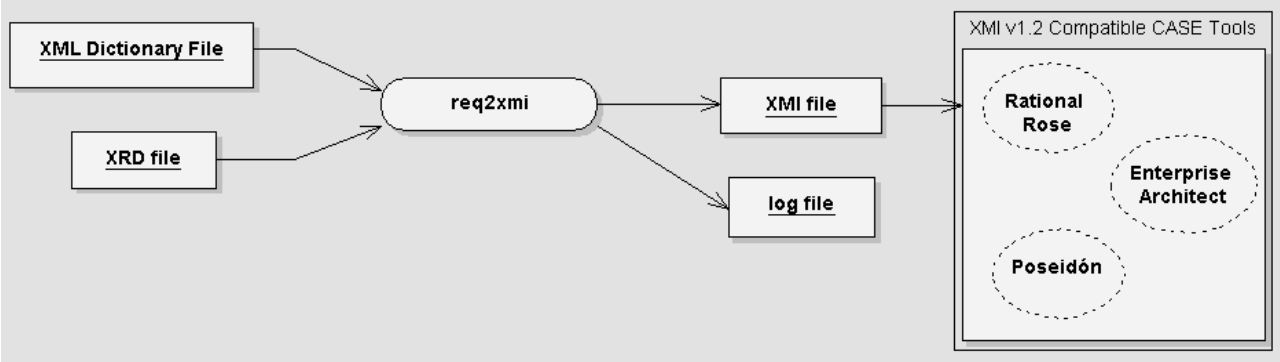

Figura 1: Contexto CIMTool

4.1 Arquitectura de la herramienta

CIMTool fue desarrollada con la herramienta ´Oracle JDeveloper 10g´ en el lenguaje Java, utilizando el ´Java 2 Runtime Environment Standard Edition v1.4.2´. La figura 2 muestra la arquitectura de la herramienta la cual está organizada en los siguientes paquetes:

- **uml** : Implementación del meta-modelo UML v1.5 [13]
- **nl2oo** :
  - **prg** : Es el paquete encargado de la interacción de los demás paquetes para llevar a cabo la transformación.
  - **dic** : Este paquete permite la clasificación de una palabra determinada.
  - **req** : Este paquete contiene un conjunto de clases java que definen los modelos de LEL y Escenarios según [6].
  - **rules** : Este paquete implementa las reglas de transformación.

A continuación, se describen brevemente los dos paquetes principales de la herramienta: ´req´ y ´rules´.

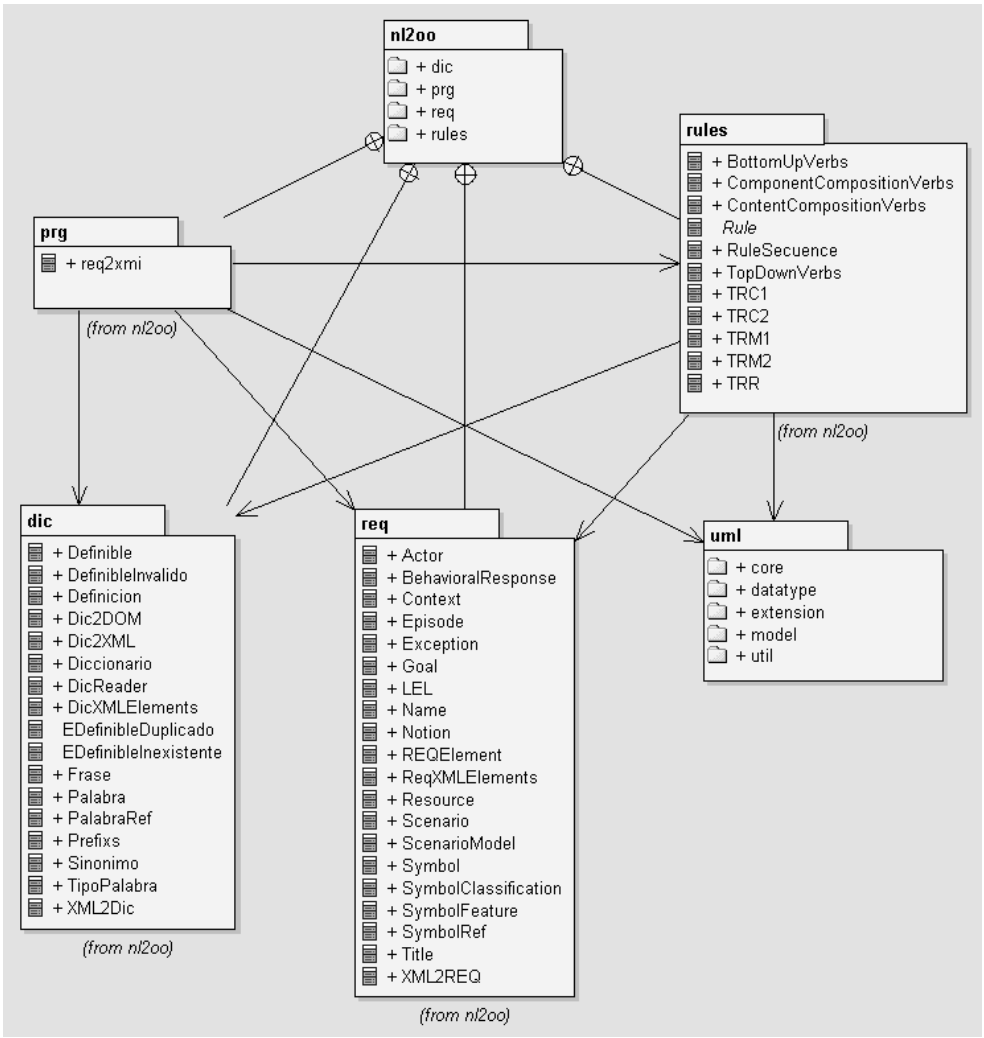

Figura 2: Arquitectura de CIMTool

4.1.1 Paquete ´req´: Modelo de Requisitos

El modelo de requisitos representado en los modelos de LEL y Escenarios se implementó en el paquete java denominado ´req´. La figura 3 muestras las clases java que componen este paquete. Para la representación del modelo de requisitos se creó una especialización del lenguaje XML a la que se denominó XRD (XML Requirement Declaration). XRD integra el LEL y el Modelo de Escenarios. XRD incluye además una serie de características requeridas por CIMTool, como por ejemplo referencias XML explícitas entre elementos del lenguaje, verbos destacados que darán lugar a nombres y tipos de relaciones en el modelo UML resultante.

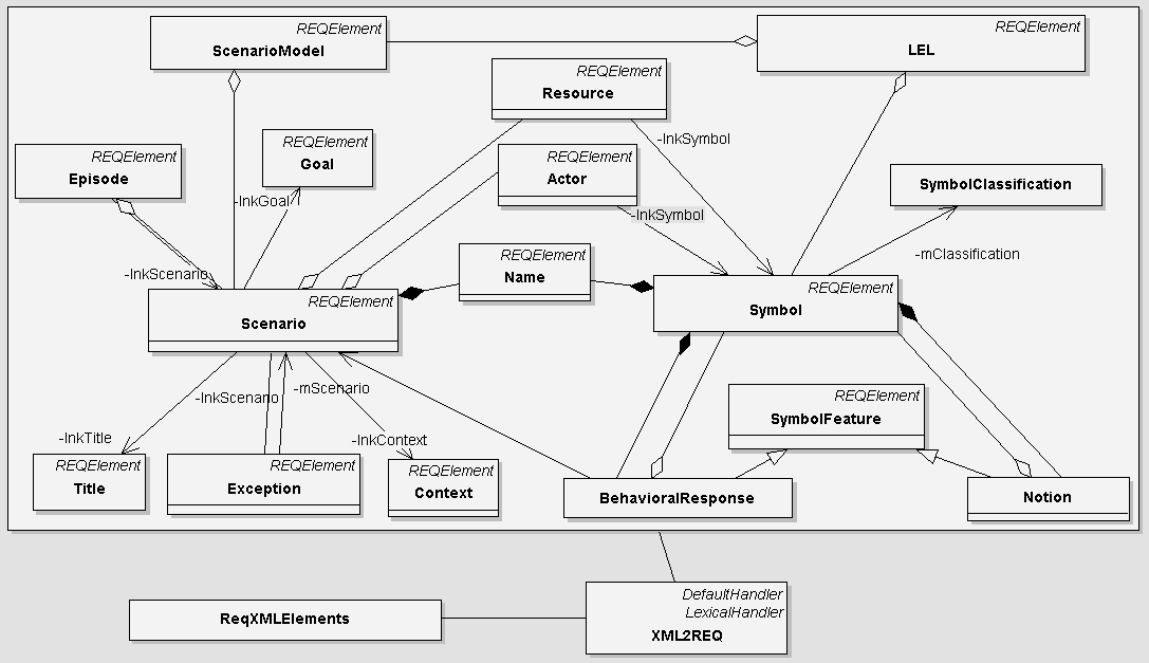

Figura 3: Diagrama de clases UML del LEL y del Modelo de Escenarios

4.1.2 Paquete ´rules´: Reglas de Transformación

El paquete ´rules´ presentado en la Figura 4 implementa las reglas de transformación. Cada regla se implementó en una clase java, que hereda de la clase abstracta Rule, la cual brinda la interfaz para todas las reglas con el método *constructor* que tiene como parámetros objetos Diccionario, LEL y UML::Model, y el método *apply*, el cual es el punto de entrada para la ejecución o aplicación de la regla. Como clase auxiliar se definió la clase RuleSecuence, que es un ´helper´ que implementa una regla que permite agrupar otras, y cuyo método *apply* no hace más que llamar al método *apply* de las reglas que contiene en el orden en que fueron agregadas. Como se explicó en la Sección 3, es importante el orden de aplicación de las reglas ya que una regla puede utilizar elementos UML creados por la aplicación de reglas anteriores. Por ejemplo, la regla TRR es la encargada de encontrar relaciones entre las clases UML que las reglas aplicadas anteriormente encontraron. Otras clases auxiliares definidas en este paquete son las clases TopDowVerbs, BottomUpVerbs, ContentCompositionVerbs y ComponentCompositionVerbs que son utilizadas por la clase TRR para identificar los diferentes tipos de verbos y clasificar las relaciones encontradas.

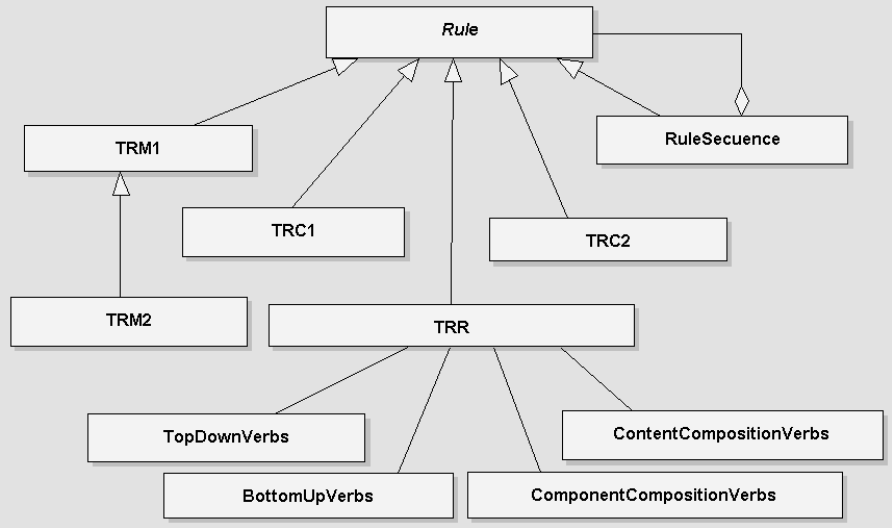

Figura 4: Implementación de las reglas de transformación

4.2 Ejecución

CIMTool es actualmente una herramienta de línea de comando que, como se muestra en la Figura 1, toma como entrada un archivo XRD con la definición de los modelos de LEL y escenarios de un caso de estudio concreto y otro archivo con el diccionario en el idioma correspondiente, y produce a partir de la aplicación de las reglas un archivo de salida XMI con la especificación del diagrama de clases UML resultante. La figura 5 muestra un diagrama de secuencia con la ejecución de la herramienta, en la que destacamos los siguientes pasos:

En primera instancia CIMTool utiliza el método estático *load* de la clase XML2Dic para cargar el diccionario; luego utiliza, análogamente, el método estático *load* de la clase XML2REQ para cargar el modelo de requisitos y luego crea un modelo UML inicialmente vacío. Posteriormente, la implementación de las reglas utilizará los objetos Diccionario y LEL para crear los elementos que formarán parte del modelo UML.

El paso siguiente crea una instancia de la clase RuleSecuence, a la que se le agregan posteriormente, en orden, instancias de las clases que implementan las reglas a aplicar. En este caso se agrega una regla TRC1, luego una TRC2, una TRM1, en cuarto lugar una TRM2 y por último una TRR.

Luego, se llama al método *apply* de la regla que es instancia de la clase RuleSecuence, y este método invoca a los métodos *apply* de las reglas que han sido agregadas anteriormente a la secuencia de reglas.

El llamado al método *apply* de cada regla crea elementos del metamodelo UML y los inserta en la instancia del modelo UML creado al inicio de la ejecución, completándolo a medida que se aplican las reglas.

Como último paso de la ejecución se utiliza la clase UML2XMI para crear el archivo XMI resultante y grabar en él el contenido del modelo UML derivado.

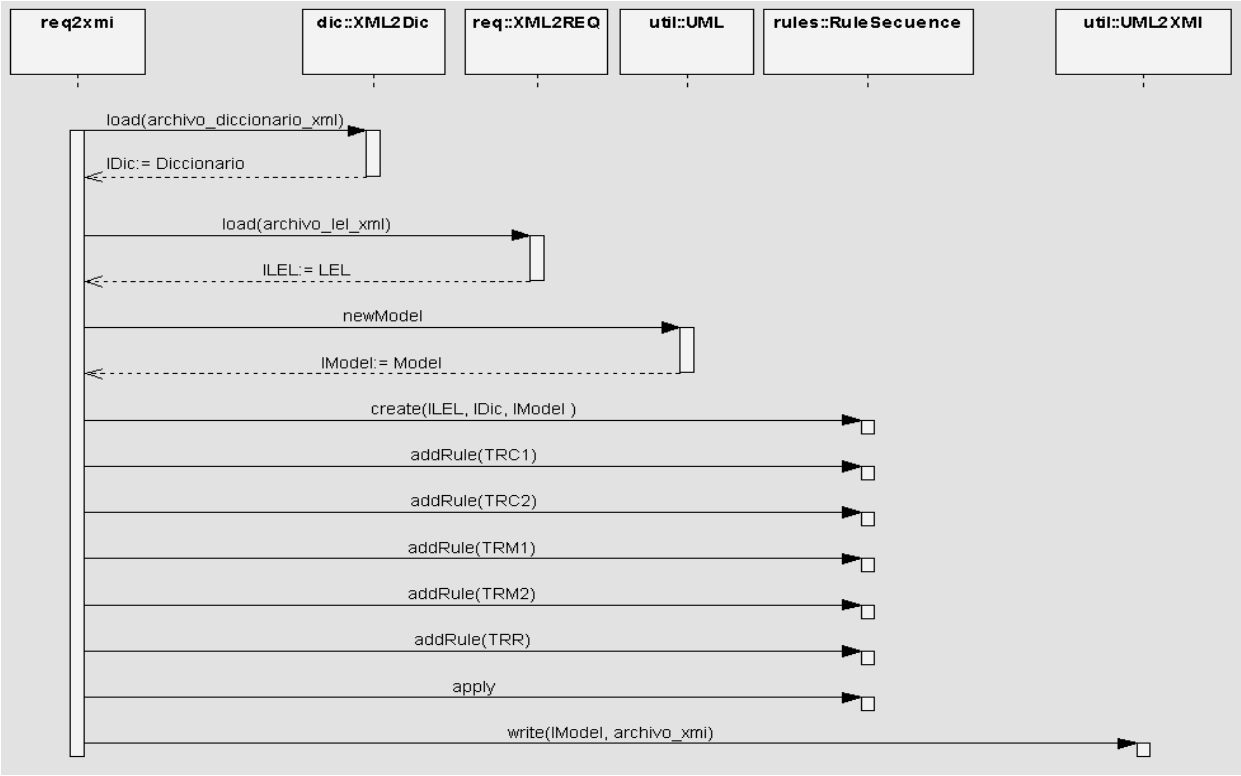

Figura 5: Secuencia de ejecución

Durante la ejecución se genera un archivo de *log* con información cuantitativa (tal como detalle de las clases, atributos, métodos y relaciones derivados) y problemas detectados al aplicar las reglas. Los problemas que frecuentemente pueden encontrarse son dos:

- Se detecta un símbolo del LEL que referencia a otro y éste no referencia al primero. En estos casos se agrega una relación con el verbo que figura en la referencia, y se indica con un *warning* que dice "relación unilateral".
- Se detectan dos símbolos que se referencian mutuamente pero los verbos involucrados no son complementarios, en este caso se deja una de las relaciones y se elimina la otra, indicando con un ´warning´ que dice "Verbos no complementarios" y se indica la relación excluida. Este punto podría ser configurable, preguntando al usuario cuál de las dos relaciones se agrega, o si se agregan las dos.

Este archivo puede ser usado por el ingeniero de software para mejorar el diagrama de clases resultante.

5. APLICACIÓN DE CIMTOOL A UN CASO DE ESTUDIO

En esta sección se muestran ejemplos de la ejecución de CIMTool en un Sistema de Producción Lechera [14]. Por cuestiones de espacio se muestra la aplicación de algunas de las reglas. En cada una de las partes del ejemplo veremos un extracto del archivo de entrada XRD con parte de la definición de los símbolos del LEL involucrados y parte de la especificación de los archivos de salida en XMI

A continuación se muestra un ejemplo de la **Regla TRC1**. A partir de un símbolo del LEL correspondiente a un sujeto detallado parcialmente en un archivo XRD, la herramienta encuentra la clase "*dairy_farmer*" con tres atributos: "*name*", "*salary*" y "*employees*", como se ve en el archivo de salida XMI.

XRD File:

```
<symbol classification="subject" id="dairy_farmer">
    <name>DAIRY FARMER</name>
    <notion><text>Person in charge of the activities in a dairy farm.</text>
      <symbol_ref symbol_id="dairy_farm" verb="in charge"/> </notion>
    <notion><text>He has a name.</text></notion>
    <notion> <text>He has a salary.</text>   </notion>
    <notion> <text>He may have one or more employees.</text> </notion>
    …
</symbol>
```

XMI File:

```
  <UML:Class      xmi.id="104c035cfe6e77dbbe7703ca62d"
          name="dairy_farmer" isSpecification="false"
          isRoot="false" isLeaf="false" isAbstract="false" isActive="false">
    <UML:Classifier.feature>
          <UML:Attribute          xmi.id="104c035cff62b5bb44488f3849b"
                                  name="name" isSpecification="false" visibility="public"/>
          <UML:Attribute          xmi.id="104c035cff6be34408d59fd9040"
                                  name="salary" isSpecification="false" visibility="public"/>
            <UML:Attribute        xmi.id="104c035cff63e5c0b056e192960"
                                  name="employees" isSpecification="false" visibility="public"/>
      …
    </UML:Classifier.feature>
    …
    </UML:Class>
```

Log: --- Aplicando TRC1 ---
Log: Clases encontrada: dairy_farmer
Log: Clases encontradas :1
Log: Atributos encontrados :3

En el siguiente ejemplo se muestra como la **Regla TRM1** encuentra y define en un archivo de salida XMI el método "*registers_heat*" para la clase "*dairy_farmer*", previamente identificada con la **Regla TRC1**. Para esto se basa en los impactos del término correspondiente cuya descripción se encuentra en el archivo XRD dado como entrada.

XRD File:

```
<symbol classification="subject" id="dairy_farmer">
                    <name>DAIRY FARMER</name>
                    …
      <behavioral_response scenario_id="register_heat">
          <text>He registers heat.</text>
          <symbol_ref symbol_id="heat_is_registered"/>
      </behavioral_response>
      …
</symbol>
```

XMI File:

```
  <UML:Class      xmi.id="104c035cfe6e77dbbe7703ca62d"
          name="dairy_farmer"
    …
    <UML:Classifier.feature>
        <UML:Operation   xmi.id="104c035d034b9312dbe45f9c586"
                                  name="registers_heat" isSpecification="false"
                                  visibility="public" isQuery="false" concurrency="sequential"
```

```
                                        isRoot="false" isLeaf="false" isAbstract="false">
          </UML:Operation>
      </UML:Classifier.feature>
      …
  </ UML:Class >

  Log: --- Aplicando TRM1 ---
     Log: Métodos encontrados        :1
```

El siguiente ejemplo muestra la aplicación de la **Regla TRR** que a partir de un conjunto de símbolos del LEL que se modelaron como clases por la **Regla TRC2** detecta una relación de herencia mostrada en la Figura 6. En este ejemplo se utilizó el Enterprise Architect para visualizar el diagrama resultante.

XRD File:

```
 <symbol classification="object" id="cow">
        <name>COW</name>
                    …
        <notion>
           <text>It may be a calf, a heifer, or a dairy cow.</text>
           <symbol_ref symbol_id="calf" verb="may be"/>
           <symbol_ref symbol_id="heifer" verb="may be"/>
           <symbol_ref symbol_id="dairy_cow" verb="may be"/>
        </notion>
        …
</symbol>
…
<symbol classification="object" id="calf">
        <name>CALF</name>
        <notion>
             <text>It is a cow of less than 12 months age.</text>
             <symbol_ref symbol_id="cow" verb="is a"/>
        </notion>
        …
</symbol>
<symbol classification="object" id="heifer">
        <name>HEIFER</name>
        <notion>
             <text>It is a fermale cow of 12 months age or more which has not yet had a calf.</text>
             <symbol_ref symbol_id="cow" verb="is a"/>
               …
        </notion>
        …
</symbol>
<symbol classification="object" id="dairy_cow">
        <name>DAIRY COW</name>
        <notion>
             <text>It is a female cow which has had at least one calf.</text>
             <symbol_ref symbol_id="cow" verb="is a"/>
             …
        </notion>
        …
</symbol>
```

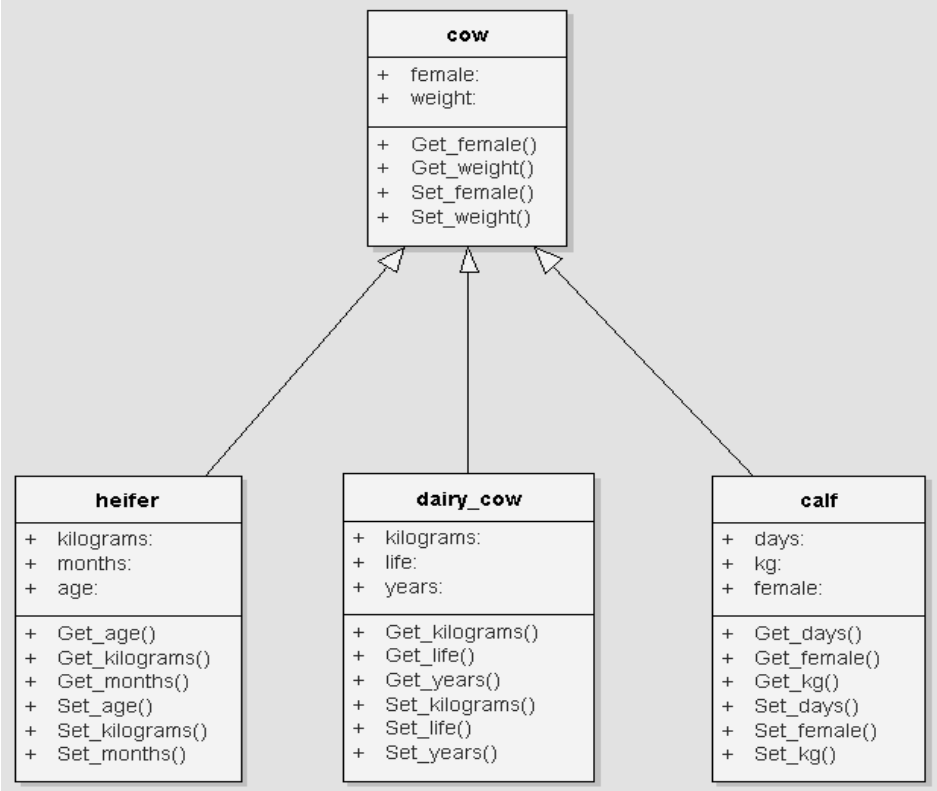

Figura 6: Aplicación de la regla TRR

El último ejemplo muestra un caso del primer tipo de *warning* que el sistema detecta. El archivo XRD describe un símbolo del LEL denominado *Cow* y otro denominado *Concentrated food*. En una entrada de la noción del segundo símbolo existe una referencia al primero, pero *Cow* no contiene referencias a *Concentrated food*. Esto provoca un fallo a las precondiciones de las reglas, pero al haber encontrado una referencia se la considera como una relación del tipo "unilateral". El usuario verá el reporte de lo sucedido, que indica la detección de la asociación y el *warning* que indica el problema.

XRD File:

```
<symbol classification="object" id="cow">
  <name>COW</name>
  <notion>     <text>It is a large animal kept in a farm to produce milk or meat.</text>
    <symbol_ref symbol_id="milk" verb="produce"/>   </notion>
  ....
</symbol>

<symbol classification="object" id="concentrated_food">
  <name>CONCENTRATED FOOD</name>
  <notion>
    <text>It is a mixture of grains (corn,barley,wheat) or balanced food given to cows as food.</text>
    <symbol_ref symbol_id="cow" verb="given to"/>
    <symbol_ref symbol_id="balanced_food"/>
  </notion>
  ....
</symbol>
```

Log: Asociación: concentrated_food given to cow
Warning: Relación unilateral: cow no referencia a concentrated_food

6. Conclusiones y Trabajos Futuros

En este trabajo se presenta CIMTool, una herramienta que permite la derivación automática de modelos de requisitos basados en lenguaje natural hacia modelos UML. Más concretamente, la herramienta analiza modelos de LEL y escenarios y aplica una secuencia de reglas para derivar un diagrama de clases UML. Esta herramienta está basada en la estrategia definida en [6], y puede integrarse en un desarrollo de software basado en MDA, definiendo de manera automática un modelo CIM orientado objetos. Al automatizar las reglas de derivación, se deben tomar decisiones fijas sobre determinados aspectos de diseño, por lo que el diagrama de clases resultante debe ser revisado por los ingenieros de software, quienes sobre la base de su experiencia y al archivo de *log* generado por CIMTool harán las modificaciones necesarias al modelo.

Como la herramienta genera documentos XML puede integrarse perfectamente con cualquier CASE de desarrollo orientado a objetos que acepte este formato. Dentro de los futuros trabajos se pretende integrar a CIMTool como *plugin* dentro de algunas de estas herramientas, en primera instancia para Poseidón. También se planea incorporar herramientas intermedias, que realicen actividades adicionales; por ejemplo, la herramienta Enterprise Architect extiende el formato XMI con elementos para describir la disposición de los elementos UML gráficamente. Sería posible, entonces, crear una herramienta que procese el resultado de CIMTool agregando información de la disposición de los elementos en un diagrama.

Referencias

[1] Kleppe, A., Warmer, J., Bast, W. "MDA Explained: The Model Driven Architecture™: Practice and Promise". Addison Wesley, 2003.

[2] OMG: Object Management Group. http://www.omg.org/

[3] "Business Processes and the OMG: an overview". http://www.omg.org/bp-corner/bp-files/The_OMG_BP_Corner_INTRO_Paper_3-2-04.pdf.

[4] Li, L. "Translating Use Cases to Sequence Diagrams". Proc. of the Fifteenth IEEE International Conference on Automated Software Engineering, 2000. pp 293-296.

[5] Pereira Pedroza, F., Alencar, F.M.R., Castro J.F.B, Silva F.R.C., Santander V.F.A. "Ferramentas para Suporte do Mapeamento da modelagem i* para a UML:eXtended GOOD-XGOOD e GOOSE". Proc. of the VII Workshop on Requirements Engineering WER 2004. Tandil, Argentina. December 2004. pp 164-175.

[6] Leonardi, M.C., Mauco, M.V. "Integrating Natural Language Oriented Requirements Models into MDA". Proc. of the VII Workshop on Requirements Engineering WER 2004. Tandil, Argentina. December 2004. pp 65-76.

[7] Leite, J.C.S, Hadad, G., Doorn, J., Kaplan, G. "A Scenario Construction Process". Requirements Engineering Journal, 5(1), Springer Verlag, 2000. pp. 38-61.

[8] Warmer, J., Kleppe, A. "The Object Constraint Language: Getting Your Models Ready for MDA". Second Edition, Addison Wesley, 2003.

[9] Juristo, N., Moreno, A., López, M., "How to Use Linguistic Instruments for Object-Oriented Analysis". IEEE Software, 17(3). May/June 2000. pp. 80-89.

[10] XMI: OMG XML Metadata Interchange. http://www.xmi.org.

[11] Poseidón, Gentleware. http://www.gentleware.com.

[12] Enterprise Architect, Sparx Systems. http://www.sparxsystems.com.au/ea.htm.

[13] Unified Modeling Language Specification. V.1.5. March2003. http://www.omg.org.

[14] Mauco, M.V. "A Technique for an Initial Specification in RSL". Master thesis, Facultad de Informática, Universidad Nacional de La Plata. La Plata, Argentina. July 2004.