Hacia una integración de MDA y el Proceso Unificado a través de reglas de transformación QVT

Ariel Arsaute, Marcelo Uva, Fabio Zorzan, Marcela Daniele, Paola Martellotto, Ariel Gonzalez, Mariana Frutos

Universidad Nacional de Río Cuarto, Facultad de Ciencias Exactas, Fco-Qcas y Naturales, Dpto. de Computación, Ruta 36 Km 601 CP X5804BYA, Río Cuarto, Córdoba, Argentina,
{aarsaute, uva, fzorzan, marcela, paola, agonzales, mfrutos}@dc.exa.unrc.edu.ar

Abstract. Model Driver Architecture (MDA) define un proceso de construcción del software basado en la producción y transformación de modelos. MDA se fundamenta en los principios de abstracción, automatización y estandarización. Vinculado con la filosofía MDA, la Object Management Group (OMG) ha definido el estándar Query/View/Transformation (QVT) para la definición y transformación de modelos de software. Por otro lado, el Proceso Unificado (PU), también define un proceso de construcción del software generando distintas vistas o modelos. En este trabajo se sientan las bases para la integración de MDA y el PU. Se propone un conjunto de reglas QVT que establecen una transformación de forma automática entre los modelos producidos en las etapas de Captura de Requisitos y Análisis. El objetivo del trabajo es la definición del conjunto completo de reglas QVT que posibiliten la transición desde la etapa de Captura de Requerimientos a la etapa de Análisis.

Keywords: Model Driver Architecture MDA, Query/View/Transformation QVT, Proceso Unificado PU.

1 Introducción

La dinámica propia de la Ingeniería de Software implica el surgimiento constante de nuevas tecnologías que den soporte al proceso de construcción de sistemas de información. Todo esto implica un arduo trabajo de integración que posibilite la coexistencia de las tecnologías involucradas.

MDA [1] define un proceso de construcción de software basado en la producción y transformación de modelos. MDA se fundamenta en los principios de abstracción, automatización y estandarización. La idea principal subyacente en MDA es abstraer propiedades y características de los sistemas de información en un modelo abstracto independiente de los cambios producidos en las tecnologías.

En sintonía con MDA, la OMG, ha definido el estándar QVT [2]. QVT consta de consultas, vistas, y reglas de transformación. El componente de consultas de QVT recibe como entrada un modelo, y selecciona elementos específicos. Una vista puede

verse como el resultado de una consulta sobre un modelo que proporciona un punto de vista particular. El componente de transformaciones de QVT permite definir transformaciones entre modelos. Dichas transformaciones describen relaciones entre dos metamodelos: fuente y objetivo. Ambos metamodelos deben ser especificados en Meta Object Facility (MOF) [3]. Una vez definida y aplicada la transformación, se obtiene el modelo instancia del metamodelo objetivo, a partir de un modelo fuente.

Por otro lado, el Proceso Unificado [4] es una metodología de desarrollo de software mundialmente conocida y utilizada con éxito, define tareas y responsabilidades para la construcción de un producto de software. La metodología divide el desarrollo en etapas, cada una de las cuales produce diferentes modelos o vistas del sistema. Las primeras etapas del proceso, centran sus esfuerzos en la comprensión del problema, las tecnologías a utilizar, la delimitación del ambiente del proyecto, el análisis de los riesgos, y la definición de la arquitectura del proyecto. Las actividades centrales son aquellas encargadas de modelar el negocio.

El objetivo final de esta línea de investigación, es lograr la automatización parcial o total de las actividades del PU, desde la Captura de Requisitos hasta la Implementación, aplicando reglas de transformación de modelos QVT. En este artículo se presenta el primer paso para el logro de este objetivo. Se definen las reglas QVT para la transformación entre los modelos fuente y objetivo. El modelo fuente es una instancia del metamodelo "Descripción Detallada de Casos de Uso", definida en [5] por los autores de este trabajo, y el modelo objetivo es una instancia del metamodelo de Unified Modeling Language (UML) [6]. La aplicación de las reglas QVT produce un diagrama de clases de análisis (etapa de análisis) a partir de la descripción detallada de los casos de uso(etapa de captura de requisitos).

El artículo esta organizado de la siguiente forma: En la sección 2 se presenta MDA, en la sección 3 se referencia al estándar QVT definido por la OMG. La metodología de desarrollo del Proceso Unificado [4] es presentada en la sección 4. En la sección 5 se presenta la propuesta junto con un caso de estudio, y finalmente, en la sección 6 se expone las conclusiones.

2 Model Driver Architecture (MDA)

MDA define un proceso de construcción de software basado en la producción y transformación de modelos. Los principios en los cuales se fundamenta MDA son: abstracción, automatización y estandarización. El proceso central de MDA es la transformación de modelos. La idea principal subyacente es utilizar modelos, de manera que las propiedades y características de los sistemas queden contenidas en un modelo abstracto independiente de los cambios producidos en la tecnología. MDA proporciona una serie de guías o patrones expresadas como modelos. MDA establece cuatro niveles de abstracción: Modelo Independiente de Cómputo (Computation Independent Model - CIM), Modelo independiente de la plataforma (Platform Independent Model - PIM), Modelos Específicos de la Plataforma (Platform Specific Model - PSM), y la aplicación final. Los modelos CIM describen el entorno en el que se utilizará el sistema, sin referencia directa a su implementación. Los PIM modelan funcionalidad y

estructura de un sistema de información sin considerar detalles tecnológicos de la plataforma en la cual se implementará el sistema. Los PSM describen los modelos específicos de plataforma donde se ejecutará el sistema. MDA plantea el siguiente proceso de desarrollo: de los requisitos se obtiene un modelo independiente de la plataforma (PIM), luego este modelo es transformado con la ayuda de herramientas en uno o más modelos específicos de la plataforma (PSM), y finalmente cada PSM es transformado en el código. Por lo tanto, MDA incorpora la idea de transformación de modelos (CIM a PIM, PIM a PSM, PSM a código) necesitando herramientas para la automatización de estas actividades. Estas herramientas constituyen uno de los elementos básicos de MDA.

La Fig.1 muestra el proceso y los roles definidos por MDA.

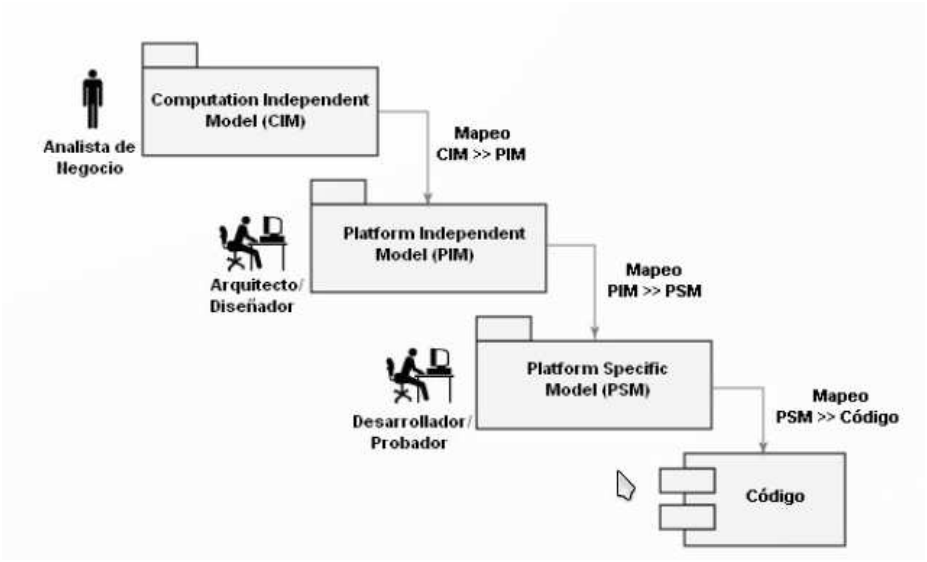

Fig. 1. Procesos y Roles en MDA.

El principio de abstracción utilizado por MDA centra su atención en el dominio del problema más que en la tecnología. Los diferentes modelos tienen por finalidad la definición de una semántica que separe aspectos relevantes del problema de los vinculados a la tecnología. Con respecto a la automatización, MDA favoreció al surgimiento de nuevas herramientas CASE con funcionalidades específicas destinadas al intercambio de modelos, verificación de consistencia, transformación de modelos y manejo de metamodelos, entre otras.

3 Query/View/Transformation (QVT)

La OMG [7] ha definido el estándar QVT [2] para trabajar con modelos de software. QVT consta de consultas, vistas, y transformaciones. El componente de consultas de

QVT toma como entrada un modelo, y selecciona elementos específicos del mismo. Para la resolución de las consultas, se propone el uso de una versión extendida de OCL 2.0 [8]. Una vista es una proyección realizada sobre un modelo, creada mediante una transformación. Una vista puede verse como el resultado de una consulta sobre un modelo, proporcionando un punto de vista particular, restringiéndolo de acuerdo a alguna condición impuesta. En esta sección se presenta el componente de transformaciones de QVT que tiene como objetivo definir transformaciones entre modelos. Estas transformaciones describen relaciones entre un metamodelo fuente F y un metamodelo objetivo O, ambos metamodelos deben ser especificados en MOF [3]. Una vez definida la transformación, es utilizada para obtener un modelo objetivo, una instancia del metamodelo O, a partir de un modelo fuente, que es una instancia del metamodelo F. También la transformación puede ser utilizada para chequear la correspondencia entre dos modelos. La especificación de QVT 1.1 [2] tiene una naturaleza híbrida declarativa/imperativa. En este trabajo interesa el lenguaje Relations que tiene naturaleza declarativa.

3.1 Lenguaje Relations

El lenguaje Relations es una especificación declarativa de las relaciones entre metamodelos MOF. Una transformación especifica un conjunto de relaciones que deben cumplir los elementos de los modelos involucrados. La ejecución de una transformación en una dirección particular se realiza seleccionando el metamodelo objetivo de la transformación. Una relación especifica la relación entre elementos de los modelos candidatos. La relación involucra dos o más dominios, y dos restricciones denominadas cláusulas *guard* (o cláusula *when*) y cláusula *where*. La cláusula *guard* de la relación especifica la condición bajo la cual la relación debe ser cumplida. La cláusula *where* define la condición que deben cumplir los elementos intervinientes en la relación. Una relación puede ser declarada en modo sólo chequeo (check only) o forzado (enforced). Si un domino es marcado como sólo chequeo, cuando se ejecute la transformación sólo será chequeado para ver si existe una correspondencia válida con otro dominio perteneciente al modelo objetivo; en cambio, si el modelo está marcado como forzado, cuando una relación no se satisface, los elementos del modelo objetivo serán creados, borrados o eliminados en el modelo para satisfacer la relación. En la actualidad existen herramientas bastantes maduras que implementan el lenguaje Relations, una de estas es MediniQVT[9].

3.2 MediniQVT

Para que la integración entre herramientas de software tengan sentido, es necesario que existan acuerdos comunes o interfaces entre artefactos y procesos. Estas interfaces pueden ser representadas utilizando el Eclipse Modeling Framework (EMF) [10]. EMF es un marco de modelado para la herramienta Eclipse [11]. EMF toma un modelo definido en UML, un esquema Extensible Markup Language (XML) [12] o una interfaz Java, y genera las correspondientes clases de implementación. Uno de los papeles más importantes del EMF es vincular los conceptos de modelado directamen-

te con su implementación. Esto proporciona a Eclipse los beneficios de modelado con bajo costo de entrada para desarrolladores centrados en código. EMF pretende unificar la representación de las estructuras de datos definidas en la aplicación, por lo que es irrelevante si esas estructuras están definidas en UML como esquemas XML o como interfaces Java. Los modelos descritos en EMF se representan en un modelo interno llamado Ecore. EMF es la realización de IBM de MOF [3]. MOF es un estándar definido por la OMG para describir repositorios de metadatos. MOF es similar a Ecore en su capacidad de especificar clases con sus características de comportamiento y estructurales, herencia, paquetes y reflexión. Difieren en que MOF tiene complejidades adicionales para el ciclo de vida, estructuras de datos, relaciones de paquetes y tipos complejos de asociaciones. Utilizar QVT/Relation tiene la ventaja que en la actualidad hay herramientas maduras que implementan el lenguaje, como por ejemplo la utilizada en este trabajo denominada MediniQVT [9]. Esta herramienta implementa la especificación QVT/Relations de la OMG en un poderoso motor QVT. Está diseñada para transformaciones de modelos permitiendo un rápido desarrollo, mantenimiento y particularización de reglas de transformación de procesos específicos. La herramienta está integrada a Eclipse y utiliza EMF para la representación de modelos. Además, posee un editor con asistente de código y un depurador de relaciones.

4 Proceso Unificado: De la Captura de Requerimientos al Análisis

El Proceso Unificado [4] es una metodología de desarrollo de software mundialmente conocida y utilizada con éxito. El PU define tareas y responsabilidades para construir un producto de software. Es una guía para todos los participantes del proyecto: clientes, usuarios, desarrolladores, directivos, etc. El PU especifica los artefactos que deben producirse en cada una de las etapas del desarrollo y ofrece criterios para el control y medición, además de reducir riesgos y aportar al proyecto predecibilidad. El PU utiliza UML como medio de expresión de los diferentes modelos generados en cada etapa. UML es un lenguaje estándar de modelado que permite visualizar, especificar, construir y documentar los artefactos de un producto de software.

El PU divide el desarrollo del producto de software en fases o etapas. Cada una de éstas produce diferentes modelos o vistas del sistema. Las primeras etapas centran sus esfuerzos en la comprensión del problema, las tecnologías a utilizar, la delimitación del ambiente del proyecto, el análisis de los riesgos, y la definición de la arquitectura del proyecto. La etapa de captura de requerimientos establece la creación de una serie de artefactos con el objeto de comprender la estructura y la dinámica de la organización. Asegurar a los clientes, usuarios, y desarrolladores una visión común de la organización. Los artefactos producidos en esta etapa son: El Modelo de Dominio, Glosario de Términos, las Reglas del Negocio y el modelo de casos de uso del negocio y de sistema. En las sucesivas etapas, se construyen artefactos que modelan el refinamiento de los casos de uso (CU) o funcionalidades. Cada CU se requiere de una descripción detallada donde se indica: una precondición, una poscondición, un flujo de eventos

principal y otro flujo de eventos alternativo. De la corrección y completitud de las descripciones detalladas de cada CU se fundamenta el éxito de todo proyecto.

Caso de Uso: **Insertar Cliente**
Precondición: existe un cliente para ser ingresado al sistema.
Poscondición: se registra el nuevo cliente en el sistema o el cliente ya estaba cargado.

Flujo de Eventos Principal

Secretaria	Sistema
1. Ingresa el DNI del cliente.	
	2. Chequea existencia.
3. Ingresa nro socio, nombre, teléfono y localidad	
	4. Include (buscar localidad)
	5. Carga al nuevo cliente.

Flujo de Eventos Alternativo

2.1. El cliente ya existe, el sistema informa tal situación y termina el caso de uso.
4.1. La localidad ingresada no existe, debe ingresar una localidad existente.

Fig. 2. Descripción detallada del Caso de Uso “Insertar Cliente”.

Una vez culminada la captura se continúa con la etapa de Análisis. Para ello el PU establece un mapeo entre la descripción detallada de cada Caso de Uso y las denominadas Clases de análisis. Cada descripción de CU se transforma en una o más clases de análisis. Cada clase es estereotipada como una clase Límite, de Entidad o de Control, se identifican sus atributos, relaciones y responsabilidades. En definitiva, el modelo generado es un diagrama de clases de UML. En la Fig. 2 se muestra la descripción detallada del caso de uso “Insertar Cliente” y en la Fig. 3 se muestra el diagrama de clases de análisis correspondiente.

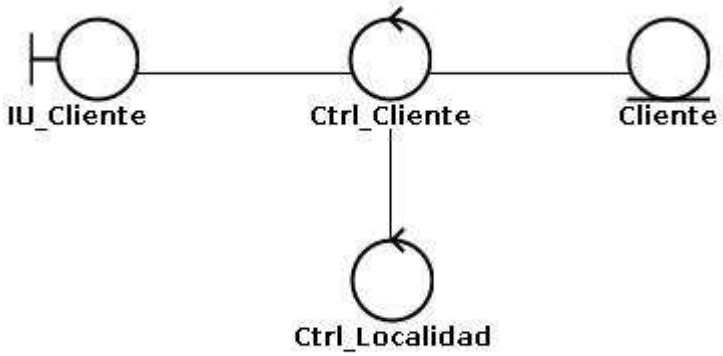

Fig. 3. Diagrama de clases de Análisis - Caso de Uso “Insertar Cliente”.

5 Automatización del mapeo entre la descripción detallada de casos de uso y clases de análisis

El objetivo final de esta línea de investigación es lograr la automatización parcial o total de las actividades del PU, es decir, desde la "Captura de Requisitos" hasta la "Implementación", aplicando reglas de transformación de modelos QVT. En este trabajo se presenta la primera etapa que consiste en la transformación del modelo de descripción de Casos de Uso al modelo de Análisis de éstos. Como se mencionó anteriormente, una transformación en QVT requiere al menos dos modelos, uno fuente y otro objetivo. En este caso, el modelo fuente es una instancia del metamodelo "Descripción Detallada de Casos de Uso", propuesto por los autores de este trabajo en [5] y mostrado en la Fig. 4. La aplicación de la transformación QVT produce un modelo de análisis que cosiste en un diagrama UML de clases de análisis.

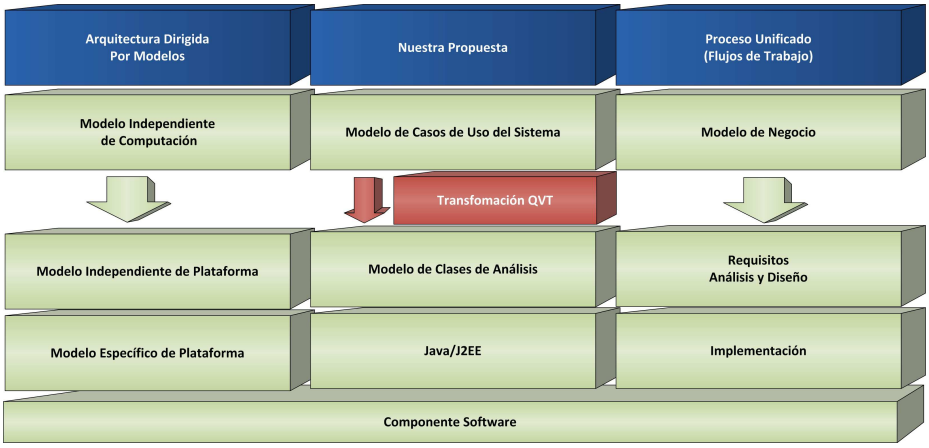

Fig. 4. Propuesta de Transformación.

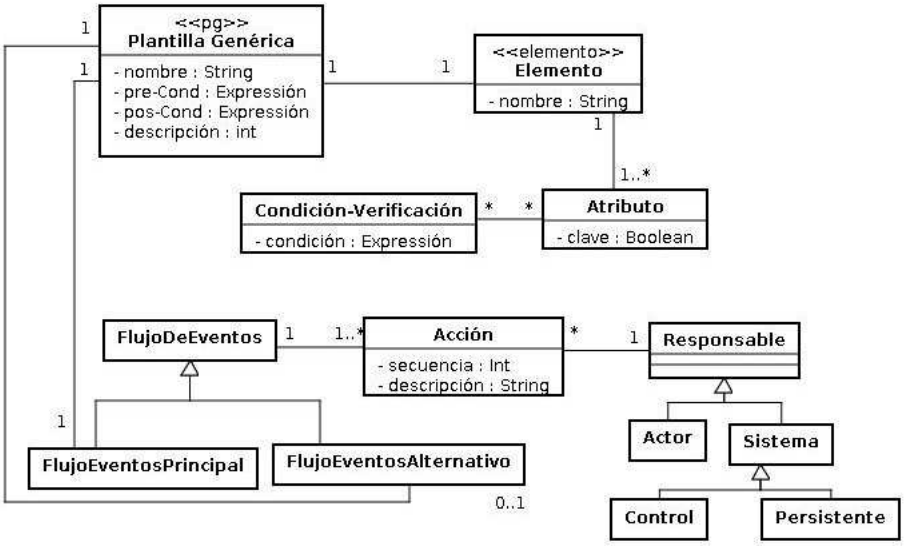

Fig. 5. Metamodelo para la descripción detallada de Casos de Uso.

La herramienta CASE utilizada para la definición de las reglas QVT es MediniQVT [9]. Esta herramienta implementa la especificación QVT/Relations de la OMG en un poderoso motor QVT. MediniQVT está diseñada para transformaciones de modelos permitiendo un rápido desarrollo, mantenimiento y particularización de reglas de transformación de procesos específicos. La herramienta está integrada a Eclipse y utiliza EMF para la representación de modelos. Previo a la definición de las reglas se realizó una traducción de los metamodelos presentados en la Fig. 5 y Fig. 6 al lenguaje Ecore, presentado en la sección 3.

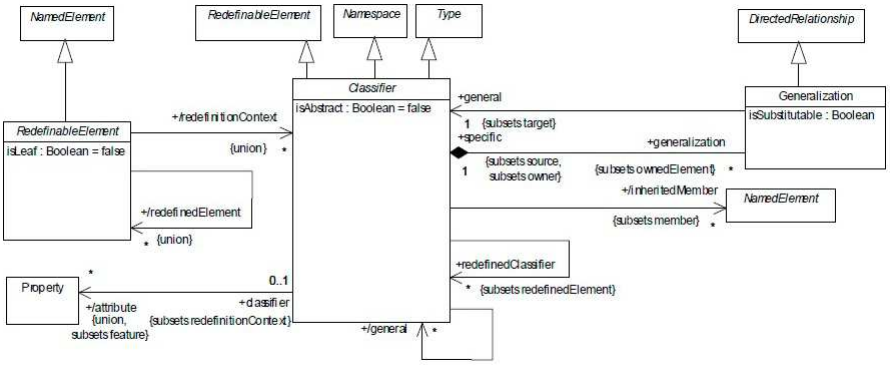

Fig. 6. Metamodelo UML.

5.1 Reglas QVT propuestas: Descripción de CU a Clases de Análisis

La transformación entre el metamodelo de Plantilla Genérica, que corresponde a la descripción detallada de Casos de Uso, y el Metamodelo UML se define en Relations de la siguiente manera:

```
transformation plantillaToAnalisis(mpg:MetamodeloPlantillaGenerica,uml1:uml)
```

Esta transformación toma un modelo de descripción de Casos de Uso (mpg), que es una instancia del metamodelo de Plantilla Genérica y un modelo UML (uml1) que es una instancia del metamodelo UML.

En la Fig. 7 se muestra la relación PackageCUsToPackage, definida en Relations, que forma parte de la transformación. A continuación se presenta una breve explicación de esta relación.

```
transformation plantillaToAnalisis(mpg:MetamodeloPlantillaGenerica,uml1:uml) {

    key uml ::PackageableElement {name};

    top relation PackageCUsToPackage {
      nombrePaquete : String;
      nombrePlantillaGenerica : String;
      nombreElemento : String;

      checkonly domain mpg paquete_pg :  MetamodeloPlantillaGenerica::PaquetePG {
         packagedElement = pg  : MetamodeloPlantillaGenerica::PlantillaGenerica {
                     name=nombrePlantillaGenerica,
                     elemento = e : MetamodeloPlantillaGenerica::Elemento {
                           name = nombreElemento}},
          name = nombrePaquete};

       enforce domain uml1 p : uml::Package {
          packagedElement = claseInterfaz : uml::Stereotype {
              name='GUI_'+nombrePlantillaGenerica,
              icon = ii :uml::Image {format = 'Interfaz'},
              ownedAttribute = atributoClaseControl : uml::Property {
                 name= 'property_interfaz_control_class_'+claseInterfaz.name,
                 association = asociacionInterfazControl : uml::Association {
                    name='AssociationInterfazControl_'+nombrePlantillaGenerica}}},
          packagedElement = asociacionInterfazControl :uml::Association {},
          packagedElement = claseControl  : uml::Stereotype {
             name='Control_'+nombrePlantillaGenerica,
             icon = ic :uml::Image {format = 'Control'},
             ownedAttribute = atributoClaseInterfaz : uml::Property {
                name= 'property_contro_interfaz_class_'+claseControl.name,
                association = asociacionInterfazControl
                                                                  },
             ownedAttribute = atributoClaseEntidad : uml::Property {
                name= 'property_control_entidad_class_'+claseControl.name,
                association = asociacionControlEntidad : uml::Association {
                   name='AssociationControlEntidad_'+nombrePlantillaGenerica}}},
          packagedElement = asociacionControlEntidad :uml::Association {},
          packagedElement = claseEntidad  : uml::Stereotype {
             name= nombreElemento,
             icon = ie :uml::Image {format = 'Entidad'},
             ownedAttribute = atributoClaseControl2 : uml::Property {
                name= 'property_entidad_control_class_'+claseEntidad.name,
                association = asociacionControlEntidad}},
          name = nombrePaquete
        };
       }
}
```

Fig. 7. Código fuente de la Transformación en Lenguaje Relation.

La relación PackageCUsToPackage define la correspondencia entre los paquetes de definición de CUs y paquetes UML. Un paquete UML contiene los diagramas de Clases de Análisis que corresponden la descripción de cada CU.

En la especificación de la relación se define la correspondencia de un paquete UML (modelo de clases de análisis) por cada paquete de descripción de CUs., en dónde cada paquete UML tendrá el diagrama de clases análisis correspondiente a la descripción del CU incluyendo estereotipos y asociaciones entre las Clases.

En la transformación se establece un mapeo entre la descripción de cada Caso de Uso y las denominadas Clases de análisis. Cada descripción de CU que corresponde a un Elemento de dato en particular, se transforma en una o más clases de análisis. Cada clase de análisis es estereotipada como una clase Límite, de Entidad o de Control, identificando e incluyendo también las relaciones existentes entre las mismas. Estas relaciones asocian la clase Interfaz con la clase Control y la clase Control con la clase Entidad . El modelo generado es un diagrama de clases de UML. En el caso de existir más de un CU que utilice un mismo elemento de datos, la transformación deberá generar una única clase Entidad y sus respectivas clases Control e Interfaz incluyendo las relaciones correspondiente entre estas.

Una vez culminada la captura se continúa con la etapa de Análisis. Para ello el PU establece un mapeo entre la descripción detallada de cada Caso de so y las denominadas Clases de análisis. Cada descripción de CU se transforma en una o más clases de análisis. Cada clase es estereotipada como una clase Límite, de Entidad o de Control, se identifican sus atributos, relaciones y responsabilidades. En definitiva, el modelo generado es un diagrama de clases de UML. En la Fig. 2 se muestra la descripción detallada del caso de uso “Insertar Cliente” y en la Fig. 3 se muestra el diagrama de clases de análisis correspondiente.

5.2 Caso de estudio

En esta sección se presenta un ejemplo de aplicación de la transformación QVT definidas en el apartado anterior.

Para poder ver la transformación gráficamente, en la Fig. 8 se muestra cómo se transforma el modelo de CU al modelo de análisis en el PU.

Para llevar a cabo esta transformación se utilizó la herramienta MediniQVT. Teniendo en cuenta que MediniQVT utiliza EMF para representar los modelos/metamodelos involucrados en las transformaciones, se especificó en EMF el metamodelo Plantillas Genéricas, mostrado en la Fig. 9. Cabe aclarar que el metamodelo UML en formato EMF, que es el metamodelo objetivo en la definición de la transformación, viene provisto por la herramienta. Para poder aplicar la transformación, que es lo más importante del caso de estudio, fue necesario especificar el modelo fuente. En la Fig. 10. se muestra éste modelo fuente, el cual es una instancia del metamodelo Plantilla genérica, y representa la descripción de los Casos de Uso Gestionar Libro, Autor y Autor Secundario de un pequeño sistema bibliotecario. Luego de especificar los moodelos/metamodelos, se aplicó la transformación al modelo fuente y se obtuvo el modelo objetivo, como se muestra en la Fig. 11. Este modelo corresponde a una especificación UML de diagrama de clases de análisis.

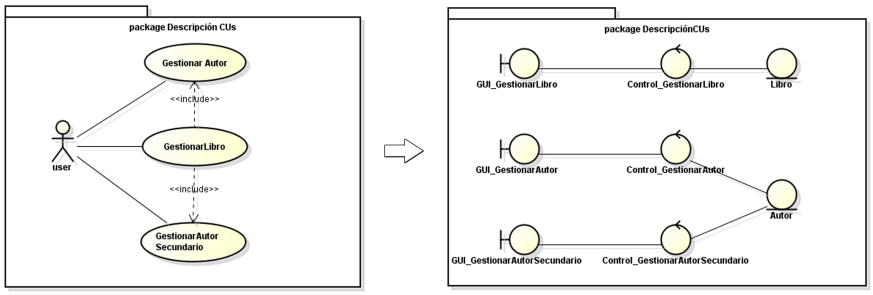

Fig. 8. Transformación del modelo de CU al modelo de análisis en el PU.

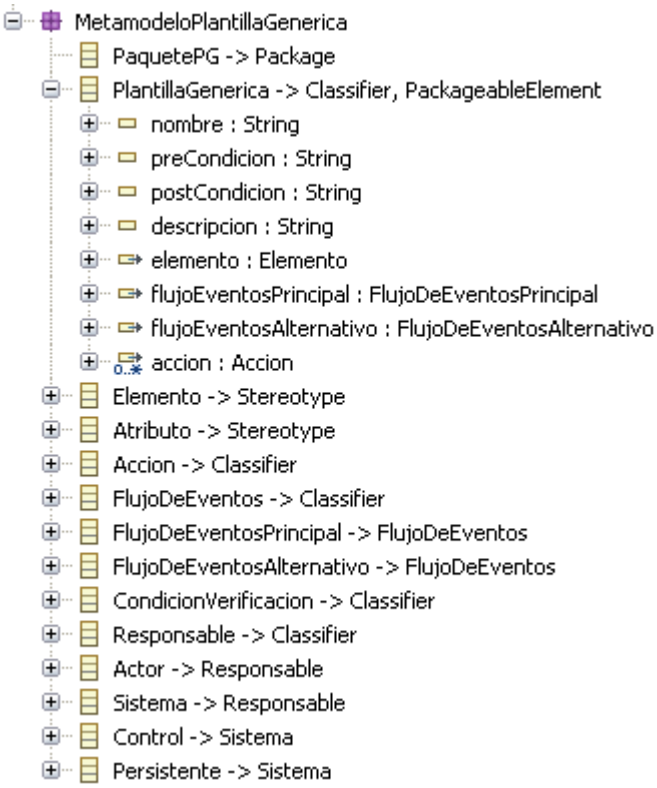

Fig. 9. Metamodelo Plantilla Genérica, vista generada por el Ecore Model Editor.

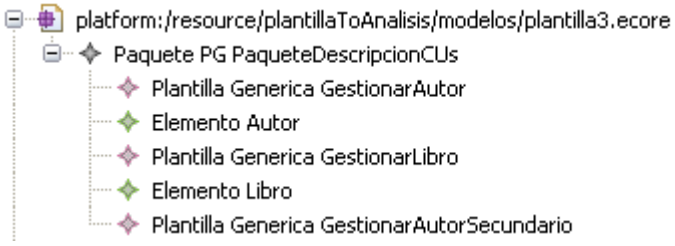

Fig. 10. Modelo Fuente (Plantilla Genérica CUs), vista generada por el Ecore Model Editor.

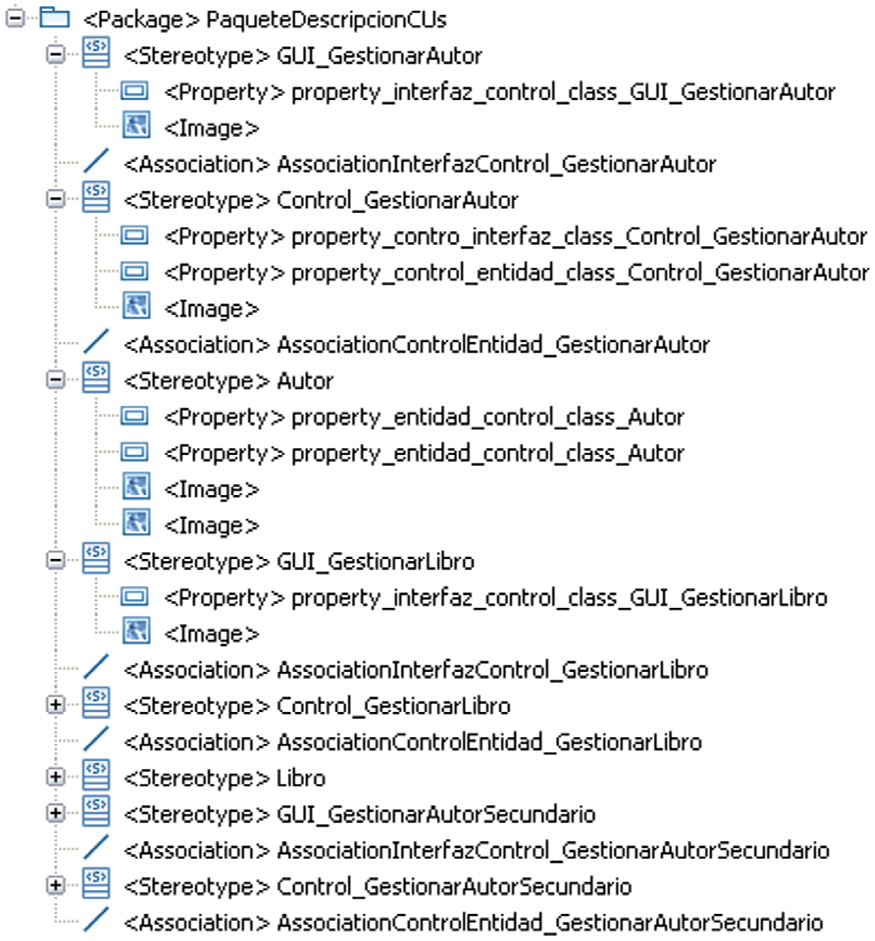

Fig. 11. Modelo Objetivo, vista generada por el UML Model Editor.

6 Conclusiones

Este trabajo tiene como objetivo hacer una contribución a la mejora de los procesos de desarrollo de software. La elaboración y culminación exitosa de este caso de estudio permitió la utilización de una especificación de captura de requerimiento para dar inicio a la construcción del software de manera automática en el marco de MDA utilizando reglas de transformación QVT.

El beneficio de esta transformación se refleja también en el dinamismo de los cambios en los requisitos durante toda la vida del software desarrollado, es decir, cualquier cambio en la especificación de la captura de requerimientos podrá ser propagado automáticamente a los distintos artefactos producidos en el proceso de desarrollo del software, esto debido a que se definió una transformación que puede ser ejecutada con una herramienta permitiendo así adaptar rápidamente los cambios de la captura de requerimiento a la especificación del análisis y diseño.

En esta línea de Investigación, hasta el momento, se ha avanzado principalmente en la transformación que convierte la especificación de Captura de Requerimientos a una especificación UML correspondiente a un artefacto de análisis, pero teniendo como objetivo realizar todas las transformaciones necesarias hasta llegar al modelo final correspondiente al componente de software y de esta manera cubrir con todas las etapas del PU.

Por último, este trabajo intenta promover el uso eficiente de modelos de sistemas en el proceso de desarrollo de software (desarrollo de software dirigido por modelos) que es uno de los principales objetivos de MDA, en el marco de la Ingeniería de Software dirigida por modelos, representando para nosotros, los desarrolladores, una nueva manera de organizar y administrar arquitecturas empresariales, basada en la utilización de herramientas de automatización de etapas en el ciclo de desarrollo del software. De esta forma, permitir definir los modelos y facilitar trasformaciones paulatinas entre diferentes modelos.

References

1. Miller, J., Mukerji, J., MDA Guide Version 1.0.1 Document number omg/2003-06-01, Disponible en: http://www.omg.com/mda, 2003.
2. Object Management Group, Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, OMG Document Number: formal/2011-01-01, Standard Document URL: http://www.omg.org/spec/QVT/1.1/PS/, último acceso Noviembre 2011
3. Object Management Group Meta Object Facility (MOF) Core Specfication OMG Available Specication. Versión 2.0. formal/06-01-01, http://www.omg.org/docs/formal/06-01-01.pdf, ultimo acceso Octubre 2010.
4. Jacobson, I. El Proceso Unificado de Desarrollo de software. Addison-Wesley, EE.UU., 2000.
5. Daniele, M. y et al.ot.. Evolución de Plantillas Genéricas para la descripción de Casos de Uso a Plantillas para el Análisis y Diseño. PIIMEG 2005. UNRC.
6. Booch G., Rumbaugh J., Jacobson I., The Unified Modeling Language. Addison Wesley, Second Edition. 2005.

7. Object Management Group, http://www.omg.org , último acceso agosto 2011.
8. Object Management Group Object Constraint Language Version 2.0. OMG Document Number: formal/06-05-01, Standard Document URL:http://www.omg.org/cgi-bin/doc?omg/03-06-01, último acceso mayo 2011.
9. ikv++: medini QVT. http://www.ikv.de/, ultimo acceso Agosto 2011.
10. Eclipse Modeling Framework, URL: http://www.eclipse.org/modeling/emf/, ultimo acceso Abril 2009.
11. Eclipse The Eclipse Foundation open source community URL: http://www.eclipse.org/
12. Object Management Group, Value type Language Mapping (XML) OMG Formally Released Versions Of XML, Standard Document URL:http://www.omg.org/spec/XML/1.1//, último acceso junio 2011.