XV Jornadas de Ingeniería del Software y Bases de Datos
JISBD 2006
José Riquelme - Pere Botella (Eds)


# TRANSFORMACIONES DE MODELOS PARA EL DESARROLLO DE BASES DE DATOS XML

**Juan M. Vara, Belén Vela y Esperanza Marcos**

Grupo Kybele
Escuela Superior de Ciencias Experimentales y Tecnología
Universidad Rey Juan Carlos
C/ Tulipán S/N, 28933Móstoles (MADRID)
e-mail: juanmanuel.vara, belen.vela, esperanza.marcos@urjc.es
Web: http://www.kybele.es/

**Palabras clave:** Transformaciones de Modelos, Formalización, Gramáticas de Grafos, Desarrollo Bases de Datos XML, MDA.

**Resumen**. *En este trabajo refinamos y formalizamos las transformaciones de modelos que completan una propuesta para el desarrollo de Bases de Datos XML (BD XML) enmarcada en MIDAS, una metodología dirigida por modelos para el desarrollo de Sistemas de Información Web basada en MDA. En esta propuesta, el Modelo Independiente de la Plataforma (PIM, Platform Independent Model) será el modelo conceptual de datos y a partir de éste se obtiene el Modelo Específico de Plataforma (PSM, Platform Specific Model), que en este caso será el modelo del XML Schema que define el esquema de la BD. Ambos serán modelos UML y por lo tanto, se propone una extensión de UML para el modelado de XML Schemas. Este artículo se centra en la formalización de los mappings entre el PIM y el PSM. Así, definimos las transformaciones de modelos para obtener el esquema de la BD XML (el modelo del XML Schema) a partir del modelo conceptual de datos (PIM). Estas transformaciones se definen inicialmente como reglas expresadas en lenguaje natural, para formalizarlas posteriormente mediante gramáticas de grafos.*

## 1. INTRODUCCIÓN

Desde que el W3C (World Wide Web Consortium) propusiese el lenguaje XML [1], éste se convirtió en el estándar de facto para el intercambio de información entre distintas organizaciones y ha terminado por ser el formato elegido por estas organizaciones, no sólo para el intercambio y transporte de datos, sino también para el almacenamiento de los mismos. Para estas organizaciones son necesarias tecnologías que permitan realizar una gestión eficiente de documentos XML y dado que lo que finalmente se quiere gestionar no son más que datos, parece evidente que la mejor opción es el uso de una Base de Datos.

Tradicionalmente, la información que contenían los documentos XML era procesada y el resultado se almacenaba en una BD convencional, obviando la naturaleza jerárquica de los datos. Pero durante los últimos años las BD XML se han erigido como la forma ideal para almacenar y gestionar documentos XML y numerosos productos comerciales han optado por incorporar soporte para BD XML ([2][3][4][5][6][7][8]). En [9] se puede encontrar un estudio bastante completo de las diferentes propuestas para la gestión de documentos XML y una clasificación de las mismas. Por tanto, tomando como medida la aceptación por parte de la industria de las propuestas que vienen del campo de la investigación, se puede concluir que las BD XML son una tecnología consolidada y de uso extendido.

Sin embargo, disponer de buena tecnología (en forma de buenos productos) no es suficiente para soportar aplicaciones o datos XML demasiado complejos. Son necesarias nuevas metodologías que guíen al desarrollador en la tarea de diseño de la BD XML, de igual forma que existen metodologías que guían al desarrollador en el diseño de las BD relacionales (e incluso objeto-relacionales) tradicionales [10][11]. En este sentido, no existe a día de hoy una metodología completa que goce de cierta aceptación para el desarrollo de BD XML. Dicha metodología debería incorporar el modelo XML y tener en cuenta tanto los problemas más tradicionales, como los más recientes, por ejemplo la migración entre plataformas o el grado de dependencia de una plataforma concreta. Las metodologías tradicionales para el desarrollo de BD han adoptado una aproximación común para abordar estos problemas, y en la misma dirección va la propuesta de este trabajo: proponer un proceso de desarrollo dirigido por modelos. Es decir, utilizar diferentes modelos para los distintos niveles de abstracción y las distintas fases del proceso de desarrollo y definir reglas de transformación entre dichos modelos.

El máximo exponente de esta tendencia en el desarrollo software durante los últimos años ha sido MDA (Model Driven Architecture) [12], la propuesta del OMG (Object Management Group). MDA es un marco de trabajo para el desarrollo software cuya principal característica es la definición de los modelos como elementos de primer orden en el diseño, desarrollo e implementación del software y la definición de las transformaciones entre dichos modelos. En función del nivel de abstracción, MDA considera diferentes tipos de modelos: los requisitos del sistema son recogidos en los Modelos Independientes de Computación (CIM, *Computation Independent Models*); los Modelos Independientes de la Plataforma (PIM, *Platform Independent Models*) representan la funcionalidad del sistema abstrayéndose de la plataforma final y los Modelos Específicos de la Plataforma (PSM, *Platform Specific Models*) combinan las especificaciones contenidas en un PIM, con los detalles de la plataforma elegida. A partir de los distintos PSM se pueden generar automáticamente distintas implementaciones del mismo sistema.

En este trabajo completamos la propuesta para el desarrollo de BD XML en marcada en MIDAS [11], una metodología dirigida por modelos para el desarrollo de Sistemas de Información Web (SIW). Se puede contemplar MIDAS como una aproximación metodológica compuesta de pequeñas metodologías que proporcionan soluciones a problemas más concretos y localizados, como por ejemplo el desarrollo del Hipertexto o de la Base de Datos del SIW. Estas soluciones se combinan para dar lugar el resultado final: el SIW a desarrollar.

Como ya se ha mencionado, este trabajo se centra en el método para el desarrollo de BD XML.

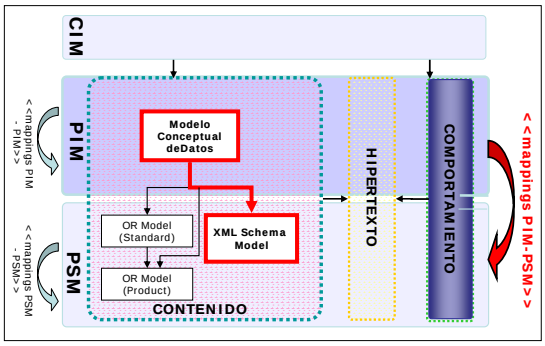


Figura 1. Desarrollo de BD XML en MIDAS

La Figura 1 resume el método y proporciona una visión del contexto en el que se ubica. El método propuesto para el desarrollo de este tipo de BD se presentó en [13] y, dado que todos los modelos contemplados en MIDAS se expresan con notación UML, se propuso y se publicó una extensión para el modelado de XML Schemas en UML [14]. Aquí completamos la propuesta formalizando las transformaciones entre los modelos PIM y PSM que forman parte del proceso de desarrollo presentado y que fueron introducidas en [13]. Para ello, siguiendo una aproximación similar a la que ya se ha utilizado para definir las transformaciones para el desarrollo de BDOR (BD Objeto-Relacionales) presentada en [15], se expresan estas transformaciones como reglas en lenguaje natural, para después formalizarlas utilizando gramáticas de grafos, un tipo especial de transformaciones basadas en reglas representadas mediante diagramas [17]. Por lo tanto y, dado que ya habíamos definido las transformaciones en forma de reglas, resulta un paso natural el uso de grafos para formalizar dichos *mappings*.
Con la definición y formalización de las transformaciones entre modelos concluimos la definición de la metodología y obtenemos finalmente una metodología completa para el desarrollo de BD XML basada en MDA.

## 2. MODELADO DE BASES DE DATOS XML

El método para el desarrollo de BD XML parte del modelo conceptual de datos representado mediante un diagrama de clases UML. Este modelo se abstrae completamente de la plataforma final en la que se desplegará la BD y por tanto se considera el modelo a nivel PIM. Este PIM se podría utilizar como modelo de partida en el desarrollo de cualquier otro tipo de BD, y de hecho en la propuesta de MIDAS se contempla también como punto de partida en el desarrollo de la BDOR [15] y se utiliza para el desarrollo del hipertexto [16]. El siguiente paso es la generación del modelo de datos para una plataforma concreta, es decir se piensa ya en la plataforma en la que se desplegará finalmente la BD y se habla por tanto de modelo PSM. Éste modelo no es más que la representación del esquema de la BD, que en el caso de una BD XML, se recoge en un XML Schema que define la estructura de los documentos XML que se almacenarán en la BD. MIDAS propone la utilización de estándares a lo largo de

todo el proceso de desarrollo, y más concretamente el uso de UML como notación para cualquier actividad de modelado, pero UML no incluye soporte para el modelado de XML Schemas. Una posible solución sería modificar el metamodelo de UML, pero esta solución es demasiado drástica. UML, sin embargo, ha sido diseñado para que pueda extenderse de una forma controlada y proporciona para ello sus propios mecanismos de extensión para cubrir la necesidad de flexibilidad. Dichos mecanismos permiten crear nuevos bloques de construcción por medio de estereotipos, valores etiquetados y restricciones recogidos todos ellos en lo que se denomina perfil UML. Así, como parte de la propuesta para el desarrollo de BD XML, se ha definido un perfil UML para el modelado de XML Schemas. Dicho perfil permite representar gráficamente los componentes específicos del estándar XML Schema conservando el orden y grado de anidamiento dado.
Tanto el perfil propuesto como el metamodelo que resulta de aplicar el perfil se han presentado anteriormente en [13] y [14], así pues se remite al lector a dichos trabajos para una mejor comprensión del modelado de XML Schemas en UML.

## 3. DEL MODELO CONCEPTUAL AL ESQUEMA DE LA BD

Como ya se ha mencionado, MIDAS sigue una aproximación dirigida por modelos para el desarrollo de SIW y por tanto, considera los modelos como actores de primer orden a lo largo de todo el proceso de desarrollo. En este contexto se puede pensar en el proceso de desarrollo como el conjunto de tareas a completar para generar los distintos modelos incluidos en el proceso. En el caso concreto que se aborda en este trabajo, la transformación resulta relativamente sencilla: se toma como entrada el modelo conceptual y se obtiene como salida del proceso el modelo que representa el esquema de la BD XML, esto es, el XML Schema que dicta la estructura de la BD XML que almacenará los documentos XML. A continuación se definen formalmente estas transformaciones, primero definiéndolas como reglas expresadas en lenguaje natural para posteriormente formalizarlas utilizando una aproximación basada en gramáticas de grafos.

### 3.1. Reglas de Transformación entre PIM y PSM expresadas en lenguaje natural

De acuerdo con [12], "la descripción de los *mappings* puede realizarse en lenguaje natural, un algoritmo en un *action language* o un modelo en un lenguaje de transformación". En nuestro caso, y como una primera aproximación a las transformaciones de modelos para el desarrollo de BD XML, se ha optado por describir las reglas de transformación en lenguaje natural, para después expresarlas por medio de gramáticas de grafos. Dichas reglas se recogen en la tabla 1.

| **Reglas de Transformación de PIM a PSM<br>Modelo Conceptual a Modelo XML Schema** |
|---|
| 1. Cada **clase** del modelo conceptual corresponderá a un **elemento** en el XML Schema con el mismo nombre de la clase. Además, para especificar el tipo del elemento, se incluirá un nuevo ***complexType*** *inline*, o si se opta por definirlo con nombre se utilizará la expresión *nombreclase_type* para nombrarlo. El elemento y el ***complexType*** se relacionarán por medio de una asociación ***uses*** |

| | |
|---|---|
| 2. | Los **atributos** de una clase se recogerán en el XML Schema como **subelementos** del ***complexType*** que sirve para definir el tipo del elemento que representa a la clase |
| 2.1. | En el caso de atributos **obligatorios** el valor de la propiedad ***minOccurs*** del **subelemento** será **1** (éste es el valor por defecto) |
| 2.2. | En el caso de atributos **opcionales** el valor de la propiedad ***minOccurs*** del **subelemento** será **0** |
| 2.3. | En el caso de atributos **multivaluados** el valor de la propiedad ***maxOccurs*** del **subelemento** será ***unbounded*** |
| 2.4. | En el caso de atributos **compuestos** el **subelemento** será de tipo ***complexType*** **anónimo**. Dicho ***complexType*** utilizará el ***compositor sequence*** para incluir un subelemento por cada componente del atributo compuesto |
| 2.5. | En el caso de atributos **enumerados** el **subelemento** será de un tipo ***simpleType*** anónimo. Este ***simpleType*** se relacionará con una clase ***enumeration*** donde se especificaran los posibles valores del atributo |
| 3. | Una **asociación** entre 2 clases se recogerá incluyendo un **subelemento** en uno de los ***complexType*** correspondiente a las clases que participan en la asociación, ya que se recogerán siempre como asociaciones **unidireccionales**. El subelemento recibirá el mismo nombre que la asociación que representa. Dicho subelemento, de tipo **REF**, referenciará al elemento que corresponde a la otra clase que participa en la asociación. El valor del atributo ***minOccurs*** de dicho subelemento dependerá de la multiplicidad mínima de la asociación (**0** ó **1**) y el valor del atributo ***maxOccurs*** será **1** |
| 3.1. | Si la multiplicidad es **1:N** el subelemento se incluirá forzosamente en el ***complexType*** que corresponde a la clase de multiplicidad **N**. |
| 3.2. | Si la multiplicidad es **N:M** el valor del atributo ***maxOccurs*** será ***unbounded*** |
| 4. | Las relaciones de **agregación** se recogerán incluyendo un **subelemento** en el ***complexType*** correspondiente a la clase que actúa como **TODO** en la relación. Para nombrarlo se utilizará el nombre de la relación y, en su defecto, la cadena ***is_aggregated_of***. El subelemento será de tipo **REF** y referenciará al elemento correspondiente a la clase que actúa como **PARTE**. El valor del atributo ***minOccurs*** de dicho subelemento dependerá de la multiplicidad mínima de la asociación (**0** ó **1**) |
| 4.1. | Si la **multiplicidad máxima** de la relación es **N**, el valor de la propiedad ***maxOccurs*** del subelemento será ***unbounded*** |
| 5. | Las relaciones de **composición** se recogerán incluyendo un **subelemento** en el ***complexType*** correspondiente a la clase que actúa como **TODO**. Para nombrarlo se utilizará el nombre de la relación y, en su defecto, la cadena ***is_composed_of***. Este subelemento será de tipo ***complexType*** anónimo y utilizará el *compositor* **all** para incluir un conjunto de elementos del tipo del complexType correspondiente a la/s clase/s que actúan como **PARTE** en la **composición** |
| 6. | En las relaciones de **generalización** el ***complexType*** utilizado para definir el tipo del elemento que representa a la clase **hija** será una extensión del ***complexType*** correspondiente a la clase **padre**. |

Tabla 1. Reglas de Transformación PIM – PSM (Modelo Conceptual – Modelo XML Schema)

### 3.2. Formalización de las reglas con gramática de grafos

Como ya se ha mencionado, el método que se propone para el desarrollo de BD sigue la propuesta de MDA. De acuerdo con los principios de MDA, las transformaciones entre modelos incluidas en cualquier proceso de desarrollo deben ser automatizadas, al menos en cierta medida, pero además, en nuestra opinión y como paso previo a la automatización, dichas transformaciones deben ser formalizadas con el objetivo de reducir la posibilidad de introducir errores en los modelos que se van generando a lo largo del proceso de desarrollo. Este objetivo cobra mayor importancia si cabe en el marco de MDA: dado que

los modelos son ahora el hilo conductor del proceso, cualquier error incluido en un modelo repercutirá directamente en la calidad y corrección del código generado. Conviene mencionar que la corrección de un modelo o, en este caso, de una transformación, no se refiere sólo a características estáticas, lo que en el caso de un modelo equivaldría a cumplir con el metamodelo correspondiente y en el caso de una transformación, a cumplir con las reglas del lenguaje utilizado para expresarla. La corrección de un modelo o una transformación se refiere también a la componente dinámica típicamente relacionada con el dominio del modelo o la transformación. Para asegurar la corrección de un modelo o una transformación bajo este punto de vista se recurre tradicionalmente a técnicas de formalización. Así, en el caso que nos ocupa se ha optado por utilizar una aproximación basada en grafos [19] para formalizar las transformaciones, y adicionalmente dar un paso más en la dirección de la automatización de dichas transformaciones, dado que proyectos como el *Attributte Graph Grammar System* (AGG) [18] proporcionan la funcionalidad necesaria para automatizar transformaciones entre modelos expresadas con grafos. Las transformaciones basadas en grafos son una aproximación declarativa, visual y lo más importante, formal, a las transformaciones de modelos. En esencia son un tipo de transformaciones basadas en reglas, en las que las reglas se representan con diagramas y el hecho de que los modelos en general se puedan representar como grafos aumenta su atractivo desde el punto de vista del desarrollo software dirigido por modelos [20]. Evidentemente, este tipo de transformaciones resultan especialmente útiles e intuitivas cuando se trata de definir transformaciones entre modelos definidos con lenguajes visuales, cuyo más claro exponente es UML.

La expresión de transformaciones de modelos como gramáticas de grafos se basa en la definición de un conjunto de reglas que siguen la estructura *LHS:= RHS (Left Hand Side:= Right Hand Side)*. En la parte izquierda (*LHS*) se define un (sub)grafo patrón y en la parte derecha (*RHS*) se define un (sub)grafo de sustitución. Cada vez que se encuentra una correspondencia con el patrón en el modelo origen, se sustituye en el modelo destino por el grafo de sustitución. En este trabajo se sigue la aproximación que ya se ha utilizado en [15] y [16] para definir reglas de transformación basadas en grafos. A continuación se presentan estas reglas para el desarrollo de BD XML.

**Transformación de clases**

En la Figura 2 mostramos las reglas para transformar las clases incluidas en el modelo conceptual, esto es las clases del PIM, a elementos del XML Schema que representará el esquema de la BD, es decir, clases en el PSM, para el caso general (a) y para el caso en que la clase sea abstracta (b).

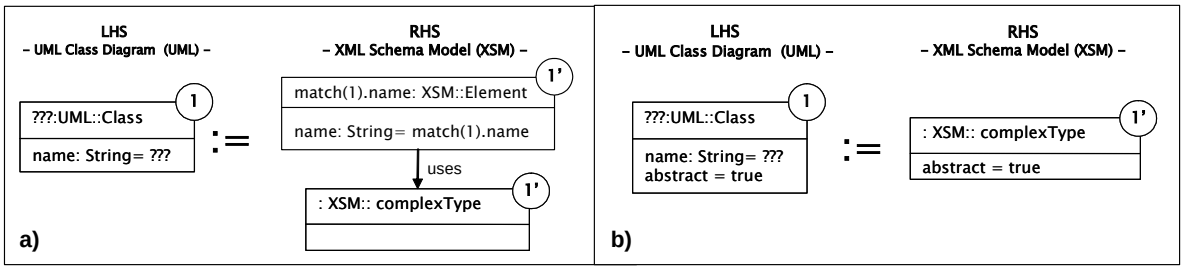


Figura 2. Transformación de clases a nivel PIM a elementos XML o clases a nivel PSM

Por cada clase UML incluida en el PIM (①) se añade en el PSM una clase *Element* que representa un elemento XML con el mismo nombre que la clase del modelo conceptual (①') y una clase *complexType* que sirve para definir el tipo del nuevo elemento (①'). Cuando la clase origen sea abstracta (figura b), tan sólo hay que especificar que el *complexType* correspondiente es también un tipo abstracto y en este caso no se incluye la clase *element* ya que el tipo no es directamente instanciable.

La regla general para la **transformación de los atributos de una clase** se recoge en la Figura 3. Por motivos de espacio no se incluyen las reglas para los casos especiales.

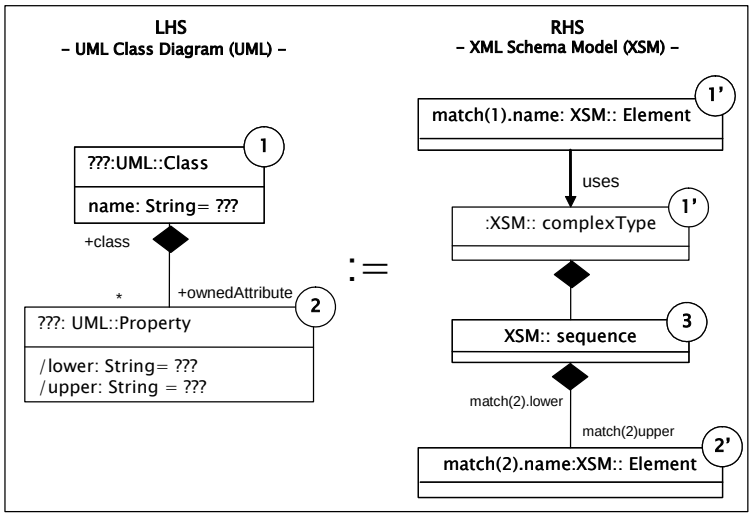


Figura 3. Regla general para la transformación de atributos

Cada atributo o propiedad de una clase del modelo conceptual (②) se recoge en el modelo del XML Schema incluyendo una clase *Element* (②') y una clase *sequence* (③) que representan un subelemento del *complexType* (①') que sirve para definir el tipo del elemento XML que mapea la clase del modelo conceptual. Además, los valores de los atributos *lower* y *upper* que definen la multiplicidad de la propiedad UML sirven para definir la multiplicidad de la relación de composición entre la clase *sequence* y la nueva clase *Element*, o lo que es lo mismo, el valor de los atributos *minOccurs* y *maxOccurs* de la nueva clase *Element.*

## Transformación de Asociaciones

Como se recoge en la tabla 1, las asociaciones UML que encontramos en el nivel PIM son transformadas a relaciones unidireccionales en el nivel PSM. En la transformación de asociaciones del PIM al PSM (del modelo conceptual al esquema de la BD XML), la multiplicidad de la asociación es la clave. Recuérdese que la multiplicidad de una asociación viene determinada por la multiplicidad máxima con que cada una de las clases interviene en la asociación. En términos del metamodelo de UML, este valor es en realidad el valor del atributo *upper* de la clase *property* que representa cada uno de los extremos de la asociación. Así, en función de estos valores, se proponen diferentes reglas de transformación para los distintos valores que puede tomar la multiplicidad de la asociación. Por motivos de espacio y dado que no hay grandes diferencias entre las reglas correspondientes, se ha optado por condensar las 3 reglas en la Figura 4.

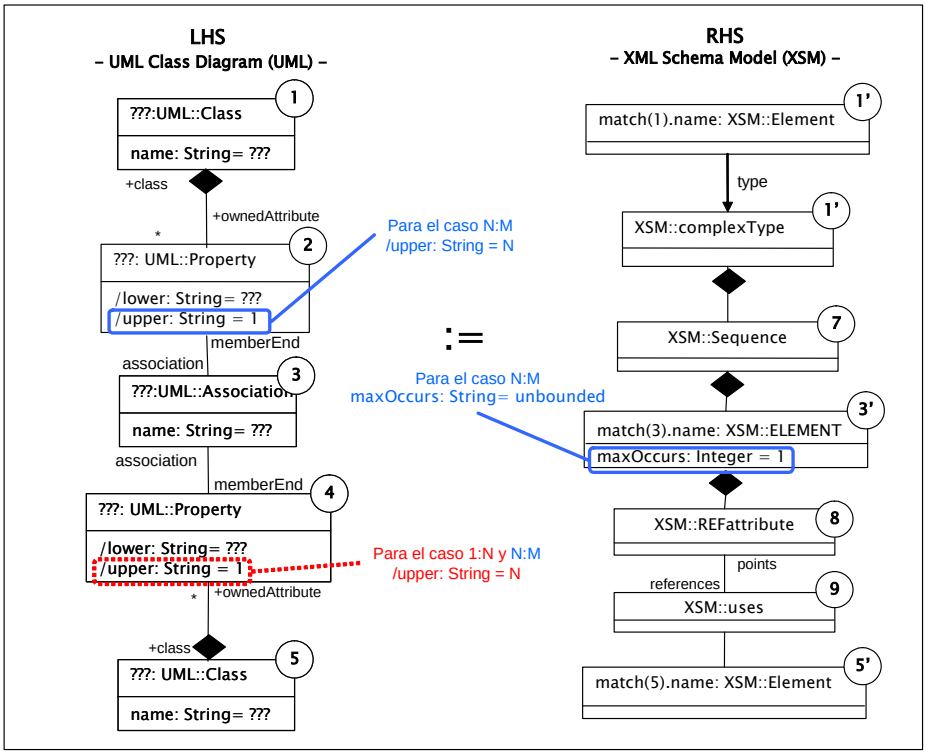


Figura 4. Transformación de asociaciones

En el primer caso, las **asociaciones 1:1**, el valor del atributo *upper* de las clases *property* (① y ④) es 1. Para representar esta asociación, se incluye una clase *sequence* (⑦) asociada a la clase *complexType* que representa el tipo del elemento XML que mapea una de las clases participantes en la asociación (①→①'). Esta clase *sequence* estará compuesta a su vez de una clase *Element* (③') que representa la asociación del modelo conceptual y que por tanto recibe el mismo nombre que dicha asociación (`match(3).name`). Además, dado que la multiplicidad es 1:1, el valor del atributo

*maxOccurs* de esta clase *Element* debe ser 1. Finalmente, el elemento XML representado por esta clase incluye un atributo de tipo REF (⑧) que apunta a la clase *Element* correspondiente a la otra clase que participa en la asociación (⑤→⑤').
Como se ha optado por definir todas las asociaciones como relaciones unidireccionales, la única diferencia entre la regla de la Figura 4 para asociaciones 1:1 y la regla de transformación para **asociaciones 1:N** estriba en el valor del atributo *upper* que, en este caso deberá ser N para una de las clases *property*, tal y como se señala en la figura con línea discontinua. En este caso, bastará con asegurarse de que el elemento XML que representa la asociación (③') es un subelemento del elemento correspondiente a la clase que participa en la asociación con multiplicidad N (①→①).
Finalmente en el caso de **asociaciones N:M**, la regla de transformación no difiere mucho tal y como se muestra en la Figura 4. En el patrón del lado izquierdo de la regla, el valor de las propiedad *upper* de las dos clases *property* (② y ④) deberá ser N y en el patrón del lado derecho, el valor del atributo *maxOccurs* de la clase *Element* (③') que representa la asociación será *unbounded*.

**Transformación de asociaciones de agregación y composición**

Dado que se ha optado por representar todas las asociaciones como relaciones unidireccionales, se deduce que no existe diferencia alguna entre la representación de una asociación N:M y la de una agregación en el modelo PSM (el XML Schema). Esta circunstancia se deriva de la diferencia entre las capacidades semánticas de los dos estándares: UML y XML Schema. Evidentemente, el primero es mucho más rico semánticamente, y de ahí que exista una clara diferenciación, no sólo a nivel sintáctico, sino también semántico entre una asociación N:M y una agregación en UML, que desaparece cuando se utiliza XML Schema. Este hecho afecta directamente a la definición de las reglas de transformación que se aborda en este trabajo. Así, la única diferencia entre la regla de transformación para una asociación N:M y la regla de transformación para una agregación estriba en el patrón derecho de la regla, mientras que el patrón izquierdo es idéntico en ambos casos.
En la Figura 5 (a) se resalta la diferencia señalada: siempre que el atributo *aggregation* de la clase *property* tome el valor *shared*, se identifica la asociación como asociación de agregación. El valor que tomen el resto de atributos es indiferente y por tanto no se explicitan. En cambio, en el caso de la asociación N:M, donde se da por sentado que el valor del atributo *aggregation* no es *shared*, el valor de los atributos *lower* y *upper* sí influye directamente en la definición del patrón izquierdo de la regla y por tanto se deben explicitar. En cambio, sí es posible utilizar las construcciones proporcionadas por el estándar XML Schema para plasmar la semántica de una asociación de **composición**. Para ello, en la regla de transformación que se muestra en la Figura 5 (b) la clase *Element* que representa la clase que actúa como parte en el modelo origen (el modelo conceptual) (⑤→⑤') está asociada por una relación de composición a la clase *complexType* que define el tipo del elemento XML que mapea la clase que actúa como todo en el modelo conceptual (①→①')

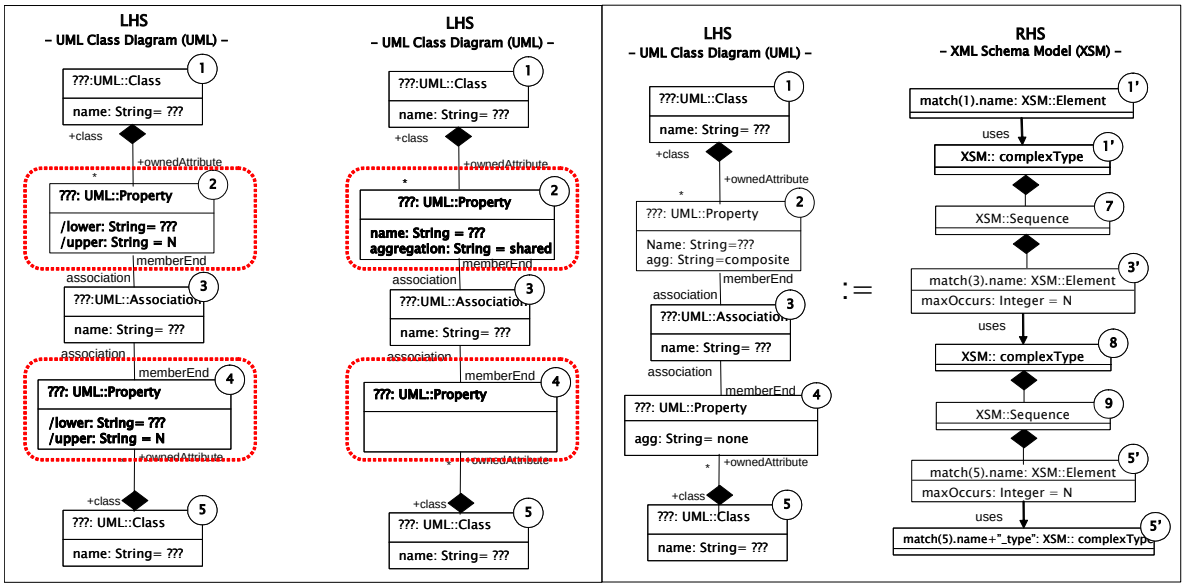


Figura 5. (a) Patrón derecho de la regla para transformación de asociaciones N:M y agregaciones
(b) Regla de transformación de relaciones de composición

## Transformación de Generalizaciones.-

Finalmente, la última de las reglas de transformación que se han definido corresponde a la transformación de las generalizaciones incluidas en el modelo conceptual.

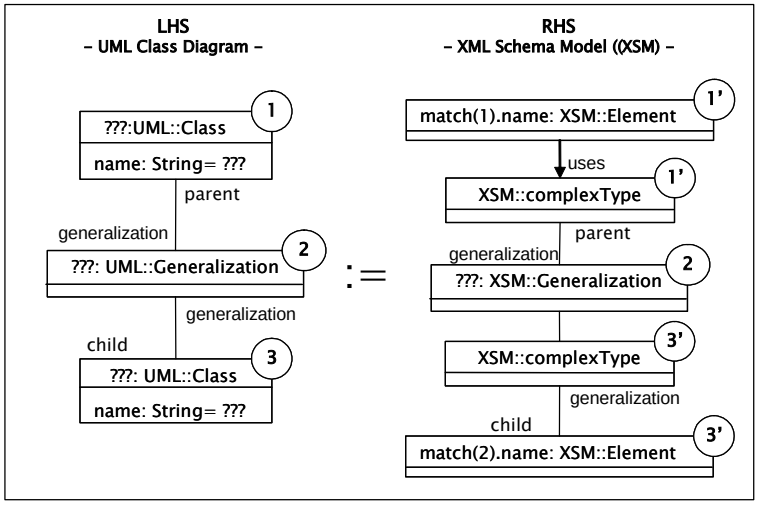


Figura 6. Regla para la transformación de generalizaciones

En este caso, tal y como se indica en la tabla 1 y recoge la regla de transformación de la Figura 6, basta con reflejar la generalización entre las clases del modelo conceptual (①-②-③) como una relación de generalización entre las clases *complexType* que sirven para definir el tipo de los elementos XML en que se transforman dichas clases ((①'-②-③').

## 4. CONCLUSIONES

Este artículo presenta el refinamiento y la formalización de las reglas de transformación del método para el desarrollo de BD XML de MIDAS. A partir del modelo conceptual, y aplicando sobre él las reglas de transformación propuestas se obtiene el modelo del XML Schema que define el esquema de la BD XML. Estas transformaciones son un ejemplo de transformaciones verticales en las que los modelos implicados poseen diferentes niveles de abstracción. Así, mediante la aplicación de las transformaciones propuestas se pasa del nivel PIM al nivel PSM, eligiendo como plataforma concreta el espacio tecnológico definido por el lenguaje XML, que proporciona un sistema para la representación de modelos y un conjunto de soluciones técnicas para manejarlos [21].

Las transformaciones de modelos propuestas, junto con el perfil UML para el modelado de XML Schemas ya presentado en [13] y [14] proporcionan un proceso de desarrollo completo para BD XML. Pero además, en conjunción con el trabajo presentado en [15] proporcionan el soporte necesario para modelar y desarrollar el aspecto del contenido de un SIW según la metodología MIDAS. Ambas propuestas combinan perfectamente y ofrecen un claro ejemplo de las ventajas de MDA: a partir de un modelo conceptual de alto nivel de abstracción que, además es parte fundamental en el desarrollo del resto del SIW, se obtiene el esquema de la BD que actúa como repositorio de información del SIW para dos tecnologías diferentes, y por tanto para dos plataformas distintas. Igualmente, este proceso de desarrollo permitiría migrar de una plataforma a otra con relativa facilidad.

Actualmente se trabaja en la integración del proceso de desarrollo presentado en la herramienta M2DAT (MIDAS MDA Tool), que integra todas las técnicas propuestas en MIDAS para la generación semiautomática de SIW. La arquitectura y primeras funcionalidades ya han sido presentadas en trabajos anteriores [22]. Concretamente ya se ha completado el desarrollo de un subsistema capaz de generar automáticamente el XML Schema a partir de su modelo UML extendido. Igualmente, la automatización de las transformaciones usando tecnologías existentes para transformaciones de grafos y lenguajes de reescritura de términos está siendo abordada.

## AGRADECIMIENTOS

Este trabajo se ha llevado a cabo en el marco del proyecto GOLD, financiado por el Ministerio de Educación y Ciencia (TIN2005-00010/).

## REFERENCIAS

[1] W3C. *XML Extensible Markup Language (XML)* 1.0 (Third Edition). W3C Recommendation. Bray, T., Paoli, J, Sperberg-McQueen, C. M., Maler, E. and Yergeau F. Retrieved from: http://www.w3.org/TR/2004/REC-xml-20040204/.

[2] Barbosa, D., Barta, A., Mendelzon, A., Mihaila, G., Rizzolo, F. and Rodriguez-Gianolli, P. *ToX - The Toronto XML Engine*, International Workshop on Information Integration on the Web, Rio de Janeiro, 2001.

[3] Chaudhri, A.B., Rashid, A. and Zicari, R. (Eds.). *XML Data Management. Native XML*

*and XML-Enabled Database Systems*. Addison Wesley, 2003.

[4] eXcelon Corporation. *Managing DXE. System Documentation Release 3.5.* eXcelon Corporation. Burlington. Retrieved from: www.excelon.corp.com, 2003.

[5] IBM Corportation. *IBM DB2 Universal Database -XML Extender Administration and Programming, Product Documentation Version 7*. IBM Corporation, 2000.

[6] Microsoft Corporation. *Microsoft SQL Server - SQLXML 2.0*, System Documentation. Microsoft Corporation, 2000.

[7] Oracle Corporation. *Oracle XML DB. Technical White Paper*. Retrieved from: www.otn.com, January, 2003.

[8] Software AG. *Tamino X-Query. System Documentation Version 3.1.1*. Software AG, Darmstadt, Germany. Retrieved from: www.softwareag.com, 2001

[9] Westermann, U. and Klas W. *An Analysis of XML Database Solutions for the Management of MPEG-7 Media Descriptions*. ACM Computing Surveys, Vol. 35 (4), pp. 331-373, December, 2003.

[10] Atzeni, P., Ceri, S., Paraboschi, S. and Torlone R. *Database Systems. Concepts, Languages and Architectures*. McGraw-Hill, 1999.

[11] Marcos, E. Vela, B., Cáceres, P. and Cavero, J.M. *MIDAS/DB: a Methodological Framework for Web Database Design.* DASWIS 2001. Yokohama (Japan), November, 2001. LNCS 2465, Springer-Verlag, pp. 227-238, September, 2002.

[12] OMG. *MDA Guide Version 1.0.* Document number omg/2003-05-01. Ed.: Miller, J. y Mukerji, J.http://www.omg.com/mda, 2003.

[13] Vela B., Acuña C. and Marcos E., *A Model Driven Approach for XML Database Development*, 23rd. International Conference on Conceptual Modelling (ER2004). LNCS 3288. Springer Verlag, pp. 780-794. 2004.

[14] Vela, B. and Marcos E. *Extending UML to represent XML Schemas.* The 15th Conference On Advanced Information Systems Engineering. CAISE'03 FORUM. Klagenfurt/Velden (Austria). 16-20 June 2003. Ed: J. Eder, T. Welzer. Short Paper Proceedings, 2003.

[15] Vara, J. M., Vela, B., Cavero, J. M. y Marcos, E. *Transformación de Modelos para el desarrollo de Bases de Datos Objeto-Relacionales*. XI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2006). Sitges (Barcelona). 3-6 Octubre 2006. (Aceptado para su presentación).

[16] Cáceres, P., De Castro, V., Vara, J.M., Marcos, E. *Model Transformations for Hypertext Modeling on Web Information Systems*. The 21st Annual ACM Symposium on Applied Computing (SAC 2006) – Track on Model Transformation. 23-27 de abril de 2006, Dijon (Francia).

[17] Czarnecki, K, and Helsen, S. *Classification of Model Transformation Approaches*. In: Proceedings of the OOPSLA'03 Workshop on the Generative Techniques in the Context Of Model-Driven Architecture, Anaheim, California, USA. 2003.

[18] Buttner, F. y Gogolla, M. *Realizing UML Metamodel Transformations with AGG*, Proceedings of GT-VMT. Electronic Notes in Theoretical Computer Science. Vol. 109, diciembre 2004, pp. 31-42. 2004.

[19] Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. 1999. *Handbook of Graph Grammars and Computing by Graph Transformation.Vol(1)*. World Scientific

[20] Selic, B. *The pragmatics of Model-Driven development*, IEEE Software, Volume 20, Issue 5, Sept.-Oct. 2003 Pp(s):19 – 25, 2003.

[21] Bézivin, J. *On the unification power of models*, Software and Systems Modeling, Volume 4, Issue 2, May 2005, Pages 171 – 188.

[22] Vara, J.M., De Castro, V. y Marcos, E. *WSDL automatic generation from UML models in a MDA framework* In *International Journal of Web Services Practices*. Volume 1 – Issue 1 & 2. Noviembre 2005, pp.1 – 12. 2005.