CONCEPTO DE RECUBRIMIENTO COMO SOPORTE AL VERSIONADO TOTAL DE ESQUEMAS EN BASES DE DATOS ORIENTADAS A OBJETO*

Jesús Manuel Maudes Raedo
Área de Lenguajes y Sistemas Informáticos
Dpto. Ingeniería Electromecánica y Civil
Escuela Universitaria Politécnica, Universidad de Burgos
Avenida Gral. Vigón s/n, 09006 Burgos
e-mail jmaudes@cid.cid.ubu.es
Tfno. +34 47 25 89 89, +34 47 25 89 70
Fax +34 47 25 89 56

José Manuel Marqués Corral
Dpto. Informática
Edificio de Tecnologías de la Información y Telecomunicaciones
Campus Miguel Delibes, Universidad de Valladolid
Tfno. +34 83 42 30 00. Ext. 5638
e-mail jmmc@infor.uva.es

Resumen

La evolución de los requisitos de los sistemas de información que utilizan bases de datos arrastra consigo la evolución de los esquemas de dichas bases de datos, dando lugar a distintas versiones de los esquemas. El versionado de esquemas permite acceder y manipular instancias de la base de datos con independencia de la versión de esquema en la que fueran creadas y de la versión de esquema desde la cual se estén accediendo/manipulando a dichas instancias por parte de usuarios y aplicaciones.

Nuestra aproximación se circunscribe a bases de datos objetuales, aportando una solución basada fundamentalmente en dos aspectos. El primero, el situar a las instancias en un esquema que aglutine toda la riqueza semántica presente en el conjunto de todas las versiones de esquema, (esquema recubrimiento). El segundo sería encontrar reglas ECA, (Evento Condición Acción), que permitan convertir el acceso o manipulación de las instancias desde una versión cualquiera del esquema, en la operación correspondiente sobre su localización real, o lo que es lo mismo, sobre el esquema recubrimiento. Ello permite establecer una metodología de definición de dichas reglas ECA, la cual puede transcribirse en términos de un entorno automatizado que de soporte al versionado de esquemas.

1. Introducción

Los sistemas de información son susceptibles de ser sometidos a constantes cambios, bien por una mala recogida de los requisitos de los mismos, bien una evolución de tales requisitos, o bien un replanteamiento de su diseño conducente a una optimización. Cuando los sistemas de información se basan en un esquema base de datos, habitualmente dicho esquema es compartido por otras aplicaciones, por lo que una reorganización del esquema puede satisfacer los nuevos requisitos de una aplicación que evoluciona dejando al descubierto la consistencia del resto de aplicaciones que dependen de dicho esquema.

No todo gestor de bases de datos permite tener el mismo grado de libertad en las actuaciones de adaptación que se realizan sobre un esquema; así la evolución permite simplemente modificar un esquema de base de datos poblado, mientras que el versionado además permite

* Este trabajo está financiado parcialmente por el proyecto CICYT TIC97-0593-C05-05.

acceder a la información vieja a través de esquemas recientes, y a la información nueva a través de esquemas antiguos. Si se limita este último tipo de acceso de tal forma que no se pueden hacer modificaciones en los datos de otra versión del esquema que no sea la actual, se dice que el versionado es parcial, en caso contrario se dice que es total [21].

Una forma primaria de conseguir el versionado total sería a través de la denominada *independencia lógica de los datos* que ya Codd [9] enunciaba para el modelo relacional. En el modelo de referencia ANSI/X3/SPARC [2] dicha independencia entre aplicaciones y esquema conceptual se consigue a través de vistas del esquema conceptual que constituyen los esquemas externos. Por ello parece lógico pensar que toda versión de esquema puede asimilarse como un esquema externo derivado de un esquema conceptual único que va evolucionando por adición de nuevos elementos.

El modelo relacional, por su simpleza, presentaría un conjunto limitado de operaciones de evolución de esquema y por tanto tal aproximación basada en vistas parece suficiente en muchos casos. Sin embargo el modelo orientado a objeto contempla aspectos que aumentan la complejidad de las implicaciones que acarrea el versionado de esquemas, (redefiniciones de métodos, reorganizaciones de jerarquías de herencia, tipos complejos de datos etc...). Si a estos problemas se añade la posibilidad de definir, modificar y eliminar restricciones sobre el esquema de base de datos como operaciones de evolución de esquema, se obtiene la auténtica dimensión del problema de versionado de esquemas en bases de datos orientadas a objeto, en el que una aproximación basada en vistas parece insuficiente.

El objetivo fundamental del presente trabajo es resolver el problema de compatibilidad entre versiones de esquema y aplicaciones desarrollando una aproximación que soporte versionado total de esquemas, la cual será elaborada desde la perspectiva del control de versiones y de la evolución de esquemas utilizando reglas ECA.

El control de versiones toma sus orígenes en las bases de datos de diseños CAD y se encarga de la problemática del tratamiento de diseños compuestos de elementos de distintas versiones [12]. Según Katz, una versión es una instantánea de un objeto de diseño con pleno significado. Es por ello que los objetos de diseño son tomados de un repositorio donde están almacenados, operación de *check out*, sobre ellos se realizan operaciones conducentes a una nueva versión de los mismos, y finalmente se devuelve la nueva versión al repositorio, no sin antes testear la mencionada plenitud semántica, en una operación denominada de *check in*.

En el presente trabajo un esquema de base de datos es considerado un objeto de diseño. La definición del esquema, que reside en el diccionario de la base de datos, está sometida a diversos refinos en su ciclo de vida sufriendo evoluciones representables por una secuencia de operaciones. Esa plenitud semántica a la que anteriormente se hacía referencia, vendrá dada en este caso por el cumplimiento de una serie de invariantes que garantizan su consistencia, como se muestra en [4, 7, 27] para el caso de ORION, *GemStone* y O_2 respectivamente.

El resto del presente trabajo se estructura de la siguiente forma: En la Sección 2, se critican los mecanismos hasta ahora existentes para implementar el versionado de esquemas. En la Sección 3 se discute el modelo de datos que se va a tomar como referencia. La Sección 4 construye la teoría de los recubrimientos estructurales partiendo de los recubrimientos de dominios y extendiéndose hasta los recubrimientos de esquemas. En esta misma sección se establecerá una definición genérica de las reglas ECA necesarias para el mantenimiento de versiones de esquema simultaneas. Finalmente en la Sección 5 se plantean las conclusiones y trabajos futuros.

2. Estado del Arte

El versionado total se hace necesario frente a la evolución del esquema y la posterior readaptación de las aplicaciones existentes al nuevo esquema, porque el costo en tiempo que conlleva esta última solución es elevado cuando no impracticable. Una posible solución en

esta línea se puede ver en [23], consistente en generar herramientas enfocadas a encontrar los puntos de las aplicaciones en los que éstas pueden verse afectadas por las evoluciones de esquemas. Esta solución es muy eficiente para determinadas operaciones de evolución de esquema, como pudiera ser el renombrar un atributo, pero para otras como pueda ser un cambio de dominio o una reorganización de una jerarquía de herencia, no ofrece una solución definitiva ni plenamente automatizable constituyendo un elemento complementario a nuestra aproximación para determinadas operaciones de evolución.

Realmente el problema del versionado de esquemas es puesto de manifiesto en los mismos términos que en el presente artículo en [22], donde aparece una taxonomía inicial de estos problemas, proponiéndose una solución que recurre a manejadores para mantener la consistencia de las instancias persistentes respecto de las aplicaciones que las utilizan. El problema de este mecanismo es que de alguna manera rompe el encapsulamiento, al estar codificados los manejadores en las propias clases, haciendo que quien las diseña no disfrute de una transparencia de los problemas subyacentes del versionado, con lo que la implementación final acaba estando condicionada por los mismos.

Otra forma de soporte del versionado surge de simular la evolución de esquema a través de vistas [5, 8, 6, 14, 20] o por cualquier otro tipo de mecanismo orientado a la proyección y derivación de determinadas partes del esquema, como pudieran ser los objetos multiestructurales [26]. Estos tipos de mecanismos usualmente están orientados a simular el comportamiento de una evolución del esquema para luego probarla antes de realmente llevarla a cabo, y no para dar un soporte definitivo al versionado. Las vistas se construyen a partir de clases, de tal forma que obtienen o derivan información ya existente, por lo que no pueden ser utilizadas por ejemplo, para simular la adición de una clase nueva sobre un esquema, salvo en el caso de las *object generating views* en *Chimera* y *Multiview*, [5] y [15] respectivamente, que soportan vistas con identificadores de objeto independientes. Aún así, en estos últimos casos no se da soporte a aspectos tales como las actualizaciones de valores de atributos que han sufrido un cambio de tipo/dominio.

En lo que a nuestro conocimiento respecta, la única posibilidad de redefinición del dominio de un atributo en vistas está ligada a la aplicación de la regla de la covarianza, (ver el modelo de *Chimera* [11]), lo cual parece insuficiente. Obsérvese por ejemplo, como un mismo atributo *peso* en una versión española definida sobre una vista en unidades del sistema métrico, no sería actualizable sobre un esquema conceptual base que lo hubiera definido en términos de libras. Sin embargo establecer una correspondencia biyectiva entre ambos dominios resulta trivial, por lo que no debiera de haber ningún problema técnico para actualizar este atributo expresado en kilogramos en la vista, en términos de libras en el esquema conceptual.

En este sentido existen implementaciones de sistemas gestores de bases de datos, (*Postgres* [24]), incluyen disparadores con modo de disparo "*instead*", es decir disparadores que pueden reconducir la acción de actualizar el valor sobre la vista en actualizar el valor sobre la tabla o clase en que realmente esté la vista basada. Este tipo de aproximación se acerca bastante a [17]. En esta última propuesta se utilizan reglas ECA como funciones de mapeo que permiten a las aplicaciones manipular y acceder a las instancias de una versión de esquema como si fueran instancias de la versión de esquema sobre la que trabaja la aplicación que las maneja.

Sin embargo, aunque estas aproximaciones proveen de los mecanismos para poder implementar la evolución de esquemas no se ofrece ninguna metodología o herramienta que a partir de las operaciones de evolución de esquema necesarias para generar una nueva versión, determine cuáles son los disparadores que habría que crear. Es en este punto fundamentalmente donde cobra sentido nuestro trabajo.

3. El modelo de datos

En principio, y pese a la necesidad de la existencia de disparadores (ausentes en el modelo de datos de ODMG 2.0, [18]), éste modelo es tomado como modelo objeto de referencia. Por tanto las operaciones de evolución de esquema, habrán de ser efectuadas sobre un esquema compatible ODMG 2.0 y conducirán a la definición de un nuevo esquema ODMG 2.0. En lo sucesivo se entenderá por esquema, un esquema de bases de datos en el modelo elegido.

Se intuye que las conclusiones que se obtendrán de trabajar con este modelo en principio serán asimilables a sistemas objeto-relacionales tipo SQL-3. Si bien es cierto que trabajar con sistemas objeto-relacionales aportaría por un lado el estar más cerca de las tendencias industriales, así como una especificación para la implementación de *triggers*, sin embargo:

- La variedad de eventos y modos de disparo de los triggers SQL-3 es insuficiente para lo que este trabajo requiere.
- La variedad de estructuras de SQL-3 es menor que la del modelo ODMG, (baste ver las colecciones posibles en ODMG 2.0 y en SQL-3).
- Finalmente el modelo SQL-3, parece todavía en muchos aspectos en construcción, mientras que ODMG aunque sigue evolucionando al menos a nuestro entender, proporciona un modelo más compacto y estable.

Al carecer el modelo ODMG de alguna forma de implementación de reglas ECA, el modelo de datos con el que se va a trabajar ha de ser completado con esta característica adicional. Una regla ECA es una regla compuesta de evento, condición y acción, como su acrónimo indica, que permite la modelización de disparadores y restricciones. En una base de datos, donde estuvieran definidas estas reglas se ejecutaría la acción que definen, sobre la instancia que dispara el evento, siempre que se verifique la condición. La ejecución de la acción lleva asociada un modo o momento de disparo: *before* (antes de producirse el evento), *after* (después de producirse el evento), *delegation*[1] (*en lugar de* producirse el evento), y *exception* (cuando la ejecución la acción asociada al evento levanta una excepción) [17].

Las reglas ECA se activarían con el paso de mensajes expresados en términos del lenguaje de manejo de datos, al intentar realizar operaciones de recuperación, modificación, inserción, borrado, ejecución de un método de una instancia. Estos mensajes generan un evento, que si coincide con alguno definido en las reglas provoca la comprobación de la condición asociada a dicha regla, la cual, caso de satisfacerse dispara la ejecución de una acción procedural en un instante expresado a través del modo de disparo.

Los disparadores que requieren las reglas que se van a utilizar en el versionado de esquemas tienen modo de disparo *delegation*, (que permite sustituir la ejecución del evento por la ejecución de la acción procedural especificada en la regla), y su modo de acoplamiento transaccional será inmediato, esto es, la acción procedural se desarrollará en el seno de la propia transacción que la ha generado, poniéndose en funcionamiento en lugar de (*delegation*) la ejecución del evento que la ha desencadenado.

4. Los Recubrimientos Estructurales

Algunos sistemas de bases de datos [4,7,27] establecen una migración de poblaciones de los objetos en el esquema viejo hacia la nueva definición de esquema. Obsérvese que por tanto dichos sistemas se limitan a dar un soporte a la evolución y no al versionado de esquemas.

El origen último de este hecho es que el nuevo esquema resultante no tiene porqué ser más rico semánticamente que el que había originariamente. (Por ejemplo, supóngase la eliminación de un atributo o de una clase). Por tanto para dar soporte al versionado sería necesario que la riqueza semántica del esquema nuevo fuera al menos la de su esquema antecesor y que a la vez contemplara aquellos elementos nuevos que han podido ser añadidos

[1] En algunos sistemas comerciales el modo de disparo *delegation* se conocen como *instead*.

al nuevo esquema fruto de su evolución, (esto es nuevas clases, nuevos atributos, nuevos métodos, nuevas interrelaciones). Es por ello que surge la necesidad de definir un esquema que contuviera la riqueza semántica de todas las versiones de esquemas anteriores, el cual se denominará esquema recubrimiento.

Los recubrimientos se centran en aspectos meramente estructurales, dejando a un lado los aspectos de comportamiento implementados a través de los métodos. Esto es debido a que los métodos pueden ser vistos como pequeñas aplicaciones diseñadas para el esquema en que están definidos, por tanto si la aproximación de versionado garantiza que los valores de las instancias se pueden manipular transparentemente desde cualquier versión de esquema y en concreto a la que pertenecen los métodos, éstos han de seguir funcionando cualquiera que sea la versión de esquema a la que realmente pertenezcan las instancias a la que dichos métodos estén accediendo o manipulando. (Véase [17] para más detalle).

4.1. Recubrimientos entre Dominios

Un recubrimiento es un formalismo que permite describir como la extensión de un conjunto puede ser representada en términos de la extensión de otro conjunto. Por tanto este formalismo parece especialmente adecuado para representar las operaciones de cambio de dominio, punto en el que el resto de aproximaciones ya presentadas han demostrado ser extremadamente débiles. Además, este formalismo se podrá aplicar al resto de operaciones de evolución de esquema.

Definición 1:

Se dice que un conjunto A es recubierto por otro B sii existe una función inyectiva de A en B.

En el contexto de este trabajo se asumirá que un dominio es un conjunto del cual una variable o atributo puede tomar sus valores, por tanto se podrá aplicar el concepto de recubrimiento a los mismos. De esta definición de dominio y del modelo de datos adoptado, (ODMG 2.0 [18]), se deduce que al poder definirse atributos como referencias a objetos pertenecientes a extensiones de alguna clase, las extensiones de las clases, (conjunto de instancias de las mismas), pueden ser vistas como dominios. Sin embargo dichos dominios al configurar el estado de la base de datos son variables en el tiempo. Por tanto se distingue entre dominios invariables (conjuntos de elementos pertenecientes a alguno de los tipos predefinidos del modelo de datos, o pertenecientes a tipos enumerados, o bien tuplas embebidas en otros objetos o colecciones de estructuras embebidas en objetos), y dominios variables, (conjuntos de instancias pertenecientes a la extensión de una clase, las cuales pueden ser referenciadas desde otras instancias, esto es: referencias a objetos y colecciones de referencias a objetos).

Al efecto de caracterizar este dinamismo de las extensiones se asume la existencia de un conjunto finito $T = \{1,...,n\}$, el cual representará de forma cronológica las transacciones que suceden en la base de datos. En toda la base de datos se considerará una única transacción i-ésima que transcurrió en una versión de esquema dado justo después de la transacción *i-1* y antes de la *i+1*, las cuales sucedieron en la misma o distinta versión de esquema.

Definición 2

Sea D un dominio, y D_i el conjunto de valores asociados a D tras la transacción i Se define D^* como $D^* = \bigcup_{i \in T} D_i$.

Definición 3

Se dice que D es un *dominio invariable* sii $\forall i, j, i \neq j \;\; D_i = D_j = D^*$. En caso contrario se dice que D es un *dominio variable*.

Como se señaló anteriormente, el interés de este trabajo se centra en aspectos meramente estructurales; por ello a continuación se define el recubrimiento estructural entre dominios.

Definición 4

Sean D y D' dos dominios cualesquiera, se dice que D *recubre estructuralmente a* D', $D' \underset{Structurally}{\subseteq} D$, sii existe una función inyectiva, $f : D'^{*} \rightarrow D^{*}$. Esta función se conocerá como *función de mapeo directa.*

La correspondiente función $f^{-1} : Img_f(D'^{*}) \rightarrow D'^{*}$, se denominará *función de mapeo inversa.*

Si un dominio D recubre a otro D', los valores en D' podrán ser guardados en la base de datos como valores de D. La función inyectiva, por tanto, provee la traducción de los valores en el dominio recubierto en valores del dominio recubrimiento; por tanto las actualizaciones de valores en D' pueden convertirse en actualizaciones sobre D a través de la función de mapeo directo especificada como una acción procedural en una regla ECA que se activa con el propio evento de actualización.

Es más, la información del dominio D' expresada en términos de D siempre puede ser recuperada a través de la función de mapeo inversa, cuya existencia se ha garantizado al recubrir D a D'. Por tanto de manera similar, una regla ECA activada con el evento de acceso a un valor de D' deberá codificar en su acción procedural tal función inversa.

Por ejemplo, el dominio de los enteros es recubierto por el de los reales, ya que una función de conversión definida de enteros a reales, es inyectiva. Lo que permite representar a los enteros como reales. Asimismo una función de redondeo o de truncamiento de reales[2] podría servir como función de mapeo inversa.

Definición 5

Sean D y D' dos dominios cualesquiera, se dice que D es un *recubrimiento estructural* equivalente de D', $D' \underset{Structurally}{\Leftrightarrow} D$, sii existe una función biyectiva, $f : D'^{*} \rightarrow D^{*}$.

Luego si dos dominios D y D' son recubrimientos estructurales uno de otro, ocurrirá que D recubre estructuralmente a D' y D' recubre estructuralmente a D.

La equivalencia de recubrimiento estructural entre dominios es un caso ideal en el que no existen pérdidas de precisión en ambas funciones inversas de mapeo. Así, en el caso de los enteros y los reales la función de truncamiento o redondeo de reales produce una pérdida de precisión para aquellos valores de un atributo que habiendo sido escritos desde una versión de esquema que los consideraba reales, están siendo leídos por otra versión de esquema que los trata como enteros. Dicha pérdida de precisión puede ser aceptable o no por la semántica del problema que se esté modelando. En el caso de que no fuera aceptable, una aproximación basada en versionado no sería aplicable, y por tanto habría que recurrir a una aproximación de evolución de esquema con el consiguiente costo de adaptación de las aplicaciones existentes. Sin embargo, en bastantes casos pudiera ser que la pérdida de precisión fuera aceptable al menos temporalmente, (típicamente durante el tiempo que el que el equipo de desarrollo necesite para llevar a cabo dichas adaptaciones de las aplicaciones existentes). Obsérvese que la razón última de esta pérdida de información es que los reales y los enteros no se recubren estructuralmente de forma mutua, esto es no son equivalentes.

Por el contrario cuando existe equivalencia de recubrimiento estructural entre dos dominios, al existir una función biyectiva entre ambos no hay esta pérdida de precisión. Este tipo de equivalencia, lejos de ser utópica, se manifiesta en múltiples ejemplos reales; (el ejemplo de libras y kilogramos expuesto anteriormente).

4.2. Recubrimientos entre Versiones de Esquemas

La definición formal del recubrimiento de versiones de esquema necesita de la introducción de una serie de conceptos previos basados en [1].

[2] No se entra a especificar cual, porque en general dependerá de los requisitos del problema.

Abiteboul [1] define un esquema de base de datos orientada a objeto como un par formado por el *esquema propiamente dicho* y la *instancia del esquema*. El *esquema propiamente dicho* describiría a las clases y sus interrelaciones, mientras que la *instancia del esquema*, sin embargo contiene las extensiones de las clases (conjunto de instancias pertenecientes a las mismas) y la codificación de la implementación de los métodos.

Para Abiteboul cada clase se define a partir de los interfaces de los métodos y de un *tipo estructural* asociado, (una tupla o una colección). Así la clase coche, podría tener un conjunto de métodos, (*fecha de la próxima revisión, revisar coche*), y una tupla constituyendo su tipo estructural, [*matrícula: tipo cadena, color: tipo enumerado color, modelo: referencia a un objeto de la clase "modelos de coche"*]. Por ello, Abiteboul define una función de *representación*, $R(C)$, que dado un identificador o nombre de una clase, C, devuelve su tipo estructural asociado. Esta función es parte integrante de la definición del *esquema propiamente dicho*. El resto de elementos constitutivos del *esquema propiamente dicho* serían el conjunto de identificadores de clase, el conjunto de nombres de variables para acceder a los objetos persistentes, el conjunto de los interfaces de los métodos (definidos de forma consistente con la jerarquía de herencia) y un orden parcial definiendo la jerarquía de herencia entre las clases que ha de ser congruente con la jerarquía de herencia definida entre los tipos estructurales asociados a las mismas.

Estos cinco elementos, (identificadores de clase, nombres de variables persistentes, interfaces de métodos, función de representación y jerarquía de herencia), pueden ser vistos como una 5-tupla. Por ello utilizando una notación de punto (*dot notation*) para un esquema S, $S.Class$ y $S.R(C)$ denotarán respectivamente el conjunto de identificadores de clase en S, y la representación o tipo estructural asociado a una clase C del esquema S. El resto de elementos no son utilizados en este artículo, el cual se centra en aspectos meramente estructurales.

Toda instancia u objeto de la base de datos es un par $(oid, value)$, donde oid es un identificador único de objeto provisto por el sistema, y $value$ es un valor. Siguiendo con la notación de punto $o.oid$ representaría el identificador del objeto o, y $o.value$ su valor.

El conjunto de instancias de una clase conforman su extensión; mientras que el conjunto de instancias de la clase que no son instancias de ninguna subclase de la misma constituyen la *extensión directa de la clase C*, que se denotará por $[\![C]\!]$. Claramente las extensiones, directas o no, son dominios variables. Por tanto utilizando la notación ya definida $[\![C]\!]_i$ representa la extensión directa de C tras la transacción i-esima, mientras que $[\![C]\!]^*$ representa el conjunto de todas las posibles extensiones de C.

Se define $dom[R(C)]$ como el dominio asociado al tipo estructural de la clase C. Es por ello que si $o \in [\![C]\!]^*$, entonces $o.value \in dom[R(C)]$

Sea OID^* el conjunto de todos los posibles identificadores de objeto en la base de datos. Se define la función *instancia directa de*, denotada por DIO_S [3], asociada al esquema S, como $DIO_S : OID^* \times T \rightarrow S.Class$. Por tanto dada una instancia o identificada por $o.oid$, y la transacción i-ésima del sistema, $DIO_S(o.oid, i) = C$, donde $o \in [\![C]\!]_i$. (Dicho de otra forma, DIO_S devuelve el identificador de clase del que un objeto es instancia directa en ese momento).

Se asume en la presente aproximación que una instancia o en una versión de esquema S, tiene el mismo *oid* que si fuera vista como una instancia de otra versión de esquema S'. Esta técnica

[3] *Direct Instance Of*. Similar a la ***función de asignamiento*** en [1], pero enfocada solo a extensiones directas de las clases.

ya es presentada con el nombre de *object preserving view*, en las vistas para bases de datos orientadas a objeto[11] y permitirá encontrar una forma sencilla de representar las instancias de una versión de esquema como instancias de otra versión de esquema.

4.2.1. Recubrimientos de Esquemas

Definición 6

Sea S un esquema y T el conjunto de transacciones, se define la *extensión de S tras la transacción* $i \in T$, y se denota por $[[S]]_i$ como la unión de todas las extensiones directas de las clases del esquema tras la transacción i. $[[S]]_i = \bigcup_{C_j \in S.Class} [[C_j]]_i$.

Definición 7

Se define la *extensión potencial de un esquema S*, y se denota por S^*, a la unión de todas sus extensiones posibles. $[[S]]^* \equiv \bigcup_{i \in T} [[S]]_i$.

Definición 8

Dados dos esquemas S y S', se dice que S *recubre estructuralmente a* S', $S' \underset{Structurally}{\subseteq} S$

$\forall o \in [[S']]^*, \forall i \in T,\ dom[[S'.R()]]IO_{S'}(o).oid, i \underset{Structurally}{\subseteq} dom[[S.R()]]IO_S(o).oid, i$.

Definición 9

Dados dos esquemas S y S' cualesquiera, y el conjunto de transacciones T se dice que F_i, definida como $F_i : [[S']]_i \to [[S]]_i$, es una *función de mapeo directo entre esquemas* tras un $i \in T$, sii dicha función es inyectiva.

Teorema:

Dados dos esquemas S y S' tal que $S' \underset{Structurally}{\subseteq} S$, y el conjunto de transacciones T, entonces para cada transacción $i \in T$ existirá al menos una función de mapeo directo entre esquemas.

Prueba:

Debido a la preservación de *oid*s entre versiones de esquemas:

$$\forall i \in T, \forall o \in [[S]]_i, F_i(o) = (F_i(oid), value) = (oid, f(value))$$

donde *f* es la función de mapeo directa entre las extensiones de las clases de las que el objeto *o* es instancia directa:

$$f : dom[[S'.R()]]IO_{S'}(o).oid, i \to dom[[S.R()]]IO_S(o).oid, i$$

Dicha función *f* existe y es inyectiva debido a que $S' \underset{Structurally}{\subseteq} S$. Por tanto $F_i(o)$ existe y es inyectiva también.

Este teorema permite reducir el problema de representar instancias de un esquema en otro al problema de encontrar las funciones de mapeo directo entre los dominios de los valores.

Definición 10

Dos esquemas S y S', son *estructuralmente equivalentes*, lo cual se denota como $S' \underset{Structurally}{\Leftrightarrow} S$, sii $S' \underset{Structurally}{\subseteq} S$ $S \underset{Structurally}{\subseteq} S$.

Definición 11

Un esquema S es un *recubrimiento estructural común* de un conjunto de esquemas $\mathbf{S} = \{S_1, ..., S_n\}$, $\mathbf{S} \overset{Common}{\underset{Structurally}{\subseteq}} S$, sii $\forall S_i \in \mathbf{S}, S_i \underset{Structurally}{\subseteq} S$.

Definición 12

Sean S y $\mathbf{S}$, un esquema y un conjunto de esquemas respectivamente verificando $\mathbf{S} \overset{Common}{\underset{Structurally}{\subseteq}} S$. Se dice que S es un *recubrimiento mínimo estructural, MCSC* [4], de $\mathbf{S}$, sii $\forall S'$, $\mathbf{S} \overset{Common}{\underset{Structurally}{\subseteq}} S' \Rightarrow S \underset{Structurally}{\subseteq} S'$.

Dado que pudieran existir varios *MCSC* estructuralmente equivalentes, el *MCSC* de un conjunto de esquemas no es en general único, por ello $S \in MCSC(\mathbf{S})$ denotará con propiedad que S pertenece al conjunto de posibles recubrimientos mínimos de $\mathbf{S}$, o lo que es lo mismo, que S es un posible *MCSC* del conjunto de esquemas $\mathbf{S}$. Nótese que si existiera $S_j \in \mathbf{S}$ verificando $\mathbf{S} \overset{Common}{\underset{Structurally}{\subseteq}} S_j$, entonces $S_j \in MCSC(\mathbf{S})$.

4.2.2. Arquitectura de versionado total basada en recubrimientos

Las versiones de esquemas que se van generando a lo largo del tiempo constituyen un conjunto de esquemas de los cuales, en función de la taxonomía de operaciones de evolución de esquema elegida, admitirán en todos o en buena parte de los casos un recubrimiento común mínimo. Por razones de espacio, en este artículo no se ha entrado a valorar cual sería la taxonomía completa de operaciones de evolución de esquema que se va a soportar. No obstante, asumiremos que dicha taxonomía tan solo amparará operaciones de evolución conducentes a la existencia en todo momento de un recubrimiento mínimo de todas las versiones de esquema. En principio, en dicho esquema recubrimiento residirán las instancias correspondientes a todas las versiones de esquema que se hayan ido generando, denominándose *esquema recubrimiento activo*, y denotándose por $MCSC_{Activo}$. El esquema que fue $MCSC_{Activo}$ antes del actual se denotará por $MCSC_{Anterior}$.

Inicialmente el sistema solo contendrá una única versión de esquema S_0. Es claro que $S_0 \in MCSC(\{S_0\})$, o lo que es lo mismo $MCSC_{Activo} = S_0$. A medida que el tiempo transcurre el sistema va generando nuevas versiones de esquema $S_1,...,S_n$. De tal forma que existirá un conjunto $\mathbf{S} = \{S_0, S_1,...,S_n\}$ de versiones de esquema. Supóngase que se genera una nueva versión S' aplicando una secuencia de operaciones de evolución de esquema sobre alguna de las versiones existentes. Entonces:

$$MCSC_{Activo} \in MCSC(\{S'\} \bigcup \mathbf{S}) \underset{Structurally}{\Leftrightarrow} MCSC(\{MCSC_{Anterior}, S'\})$$

Luego el $MCSC_{Activo}$ basta que sea un recubrimiento del $MCSC_{Anterior}$ y de la nueva versión de esquema creada.

Cualquiera que fuera la taxonomía de operaciones de evolución de esquema elegida, parece claro que existirían dos tipos de operaciones:

- Aquellas que bien reducen la semántica del esquema, o bien evolucionan hacia un esquema equivalente, (p.e. eliminar un atributo, cambiarlo de nombre).
- Por otro lado las que incrementan la semántica del esquema (p.e. añadir atributos o clases, aumentar la precisión de un dominio).

[4] Minimum Common Structural Cover.

Las primeras hacen que el nuevo $MCSC_{Activo}$ se mantenga igual que el $MCSC_{Anterior}$. Esto es, el $MCSC_{Anterior}$ vale como nuevo $MCSC_{Activo}$. Las segundas requerirán un enriquecimiento semántico parejo en el nuevo $MCSC_{Activo}$ y por tanto su evolución, (esto es añadirle el atributo o clase o aumentar la precisión del dominio).

Por tanto a partir del presente trabajo y habiendo definido una taxonomía de evolución de esquema apropiada, se debería de definir asimismo:

- La evolución de esquema paralela que hay que hacer sobre el $MCSC_{Activo}$ para que además de recubrir las versiones de esquema anteriores a la nueva versión, recubra dicha nueva versión.
- Los disparadores que traducen las operaciones de acceso y manipulación de instancias sobre una versión de esquema en operaciones de acceso y manipulación de instancias en el $MCSC_{Activo}$.
- La migración de las poblaciones de objetos instaladas en el $MCSC_{Anterior}$ al $MCSC_{Activo}$.

Estos tres elementos completarían la anunciada metodología, y serían susceptibles de ser implementados en unos disparadores sensibles a las operaciones de evolución de esquema, que como resultado de las mismas llevaran a cabo en la acción procedural estas tres tareas. Estos disparadores sensibles a las operaciones de evolución se denominarán *meta-disparadores de versionado de esquema*.

Los meta–disparadores se activarían tras la definición del conjunto de operaciones que definirían una nueva versión de esquema, es decir tras ser validado el *check-in* del mismo. Por tanto parece lógico implementarlos con un modo de acoplamiento transaccional diferido, al objeto de que no sean sensibles a operaciones de deshacer efectuadas por el administrador al hacer *check-out/check-in* de un esquema para crear una nueva versión.

Los disparadores para acceso y manipulación de instancias a través de una versión de esquema S que generarían los meta–disparadores, serían reglas ECA activadas con cualquiera de las operaciones de acceso y manipulación, en las que la condición asociada siempre se verifique, con modo de acoplamiento inmediato, tipo/momento de ejecución *delegation* y acción procedural según la Tabla 1.

Insert *o*	**Delete *o***
Insert $F(o)$	*Delete* $F(o)$
Retrieve *o*	**Update $o.a_i$ with *value***
Retrieve $F^{-1}(o)$	*Update* $F(o).a_{B(i)}$ [5] *with* $f(value)$

Tabla 1

Donde F se ha de interpretar como una función de mapeo directo entre el esquema S y el recubrimiento activo, $F : [\![S]\!]^* \rightarrow [\![MCSC_{Activo}]\!]^*$, y f es una función de mapeo directo entre los dominios asociados a las representaciones de las clases de las que la instancia o es instancia directa en el esquema S y en el $MCSC_{Activo}$ respectivamente. Esto es:

$$f : dom[S.R(DIO_S(o.oid, i))] \rightarrow dom[MCSC_{Activo}.R(DIO_{MCSC\ Activo}(o.oid, i))]$$

En cuanto a gestionar la migración de poblaciones entre un $MCSC$ viejo y otro recién creado, caben dos posibilidades:

[5] En dos versiones de esquema S y S', es posible que fruto de una operaciones de eliminación y adición de atributos de una misma clase así como de las operaciones de reorganización de jerarquías de herencia, el atributo i-ésimo de la clase $C \in S.Class$, pase a ser el atributo $B(i)$-ésimo de la clase $C' \in S'.Class$. [$B(i)$ *bounded to i*, ligado a i]

- Migración centralizada. Es decir, los cambios en los esquemas requieren su propagación a las instancias de forma inmediata [19, 16]. Esto requiere un bloqueo de grano grueso (de una parte considerable de la base de datos) durante el proceso de migración, por lo que puede dar lugar a un desempeño anormal que lo haga indeseable.
- Migración dispersa o *lazy* [4, 3, 13, 25]. Este tipo de migración no necesita de un bloqueo de grano grueso, sino que las instancias van migrando de esquema a medida que se va accediendo a las mismas. Como desventaja, las operaciones sobre las instancias crecen en complejidad y tiempo de respuesta.

La adopción de una estrategia centralizada implica que la arquitectura final del sistema tenga un único *MCSC* (el $MCSC_{Activo}$), mientras que la estrategia dispersa conlleva la cohabitación de distintos *MCSC*s en el sistema, cada uno de los cuales en algún momento hizo el papel de $MCSC_{Activo}$. Estos *MCSCs* que coexisten simultáneamente tendrán sus propias instancias, de tal manera que los disparadores de la tabla anterior deberían de ser redefinidos de forma que un acceso o manipulación de una instancia cualquiera situada en uno de estos *MCSC*s fuera acompañada de una migración previa hacia el esquema recubrimiento. (Ver [17]).

5. Conclusiones y líneas futuras

En este trabajo se ha analizado la problemática del versionado de esquemas enfocada al mantenimiento de la consistencia de las aplicaciones basadas en un esquema de base de datos que evoluciona dando lugar a distintas versiones. Asimismo, se ha justificado la insuficiencia de las soluciones existentes. De entre estas aproximaciones, la basada en reglas ECA, se encuentra falta a de una metodología que permita obtener las reglas ECA correspondientes a una evolución de esquema dada. Por ello, se ha definido una formalización basada en recubrimientos que permitirá llegar a esta metodología una vez definido el conjunto de operaciones de evolución de esquema que se vaya a considerar. Este formalismo permite adoptar políticas de migración centralizada o dispersa indistintamente según cómo se definan los disparadores de acceso y manipulación de instancias, dando lugar a distintas arquitecturas según el caso.

Además, se ha presentado el concepto de meta-disparador, como una regla ECA sensible a un tipo especial de eventos cuales son las operaciones de evolución de esquemas. La consecución de la metodología combinada con los meta–disparadores, permitirá el desarrollo de herramientas o entornos de versionado de esquemas automatizados en algún grado.

El presente artículo expone alguno de los resultados de un trabajo de tesis en desarrollo cuyo objetivo es caracterizar formalmente los mecanismos activos necesarios para dar soporte al versionado total de esquemas. En este sentido, los próximos trabajos que en esta línea se piensan abordar incluyen la adopción de una taxonomía de operaciones de evolución de esquema. Este conjunto de operaciones probablemente ha de superar en complejidad a los clásicos, [4, 7, 27], ya que se ha de contemplar la conciliación de versiones [10] orientada a esquemas, (esto es el poder definir versiones nuevas de esquema tomando y transformando elementos correspondientes a más de una sola versión de esquema ancestro). Tras esta tarea, el encontrar los meta–disparadores asociados parece lo inmediato. Además, otros puntos de interés habrán de ser investigados, entre ellos:

- Encontrar qué operaciones de evolución no admiten versionado total o parcial; o si solo lo admiten en algunos casos, identificar tales casos. A priori, parece evidente que la mayor parte de dichos casos aparecerán al considerar la adición, cambio y eliminación de restricciones del esquema como nuevas operaciones de evolución de esquema.
- Establecer las repercusiones que tendrían sobre el esquema recubrimiento, la eliminación de versiones de esquema obsoletas y establecer los procesos de recolección de basura pertinentes.

- Buscar qué incidencias tiene la aproximación basada en recubrimientos sobre otras líneas de investigación, como son la actualización de vistas en bases de datos orientadas a objetos y la integración de bases de datos semánticamente heterogéneas.

6. Referencias

[1] S. Abiteboul, R. Hull, V. Vianu, *Foundation on Databases*. Adison Wesley, 1995.

[2] Database Architecture Framework Task Group (DAFTG) of the ANSI/X3/SPARC Database Study Group; *"Reference Model for DBMS Standarization"*, SIGMOD RECORD, Vol. 15, No. 1, March 1986.

[3] J. Banerjee, H. Chou, H. Kim, H. Korth, *Schema Evolution in Object Oriented Persistent Databases.* En Proc. 6th Advanced Databases Symposium, Tokyo 1986. pp. 23-31.

[4] J. Banerjee, H. Chou, H. Kim, H. Korth, *Semantics and Implementation of Schema Evolution in Object Oriented Databases.* ACM SIGMOD Conf. SIGMOD Record Vol 16 No. 3, 1987 pp. 311-322.

[5] E. Bertino, *A view mechamnism for object oriented databases*. 3rd Intl. Conf. on Extending Database Technology (EDTB). Viena, Austria, 1992.

[6] P. Breche, F. Ferrandina, Kuklok, *Simulation of Schema Changes using Views*, N. Revell, A. Tjoa eds., Proc. 6th Intl. Conf. and Workshop on Database and Expert Systems Applications, No 978. Lecture Notes in Computer Science, pp 247-258, 1995.

[7] R. Breitl, D. Maier, A. Otis, J. Penney, B. Schuchardt, J. Stein, E. Williams, M. Williams, *The GemStone data management system*. Object Oriented Concepts, Databases & Applications; W. Kim, F. Lochovsky eds. ACM Press, 1989 pp 283-308.

[8] K. Byeon, D. McLeod, *Towards the Unification of Views and Versions for Object Databases*. Object Technologies for Advanced Software; 1st JSSST Intl. Symposium Kanazawa, Japan, Nov 1993. Lecture Notes in Computer Science 742. Springer Verlag. pp 220-235.

[9] E. Codd, "Is Your DBMS Really Relational?" y "Does Your DBMS Run By the Rules?", en Computerworld, 15 y 21 de Octubre, 1985.

[10] D. Ecklund, E., Ecklund, R. Eifrig, F. Tonge, *DVSS: A distributed version storage server in CAD applications*. 13rd VLDB. Brighton, England, 1987, pp. 443-454.

[11] G. Guerrini, E. Bertino, B. Catania, J. García-Molina. *A Formal Model of Views for Object Oriented Database Systems*. Theory and Practice in Object Oriented Systems, Vol. 3, nº 3, pp 157-183. 1997.

[12] H. Katz, *Towards a Unified Framework for Version Modeling in Engineering Databases*. ACM Computing Surveys. Vol. 22, Nº4, Dec. 1990.

[13] W. Kim, H. Chou, *Versions of Schema for object oriented databases.* F. Bancilhon, D. DeWitt eds. Proc. 14th VLDB, Los Angeles, CA. 1988. Morgan-Kaufmann. pp 148-159.

[14] W. Kim, W. Kelley, *On View Support in Object Oriented Database Systems.* Modern Database Systems: The Object Model, Interoperability, and Beyond; W. Kim. eds. ACM Press. New York. 1995. pp 108-129.

[15] H. Kuno, E. Rudensteiner, *The Multiview OODB System: Design and Implementation*. Theory and Practice Of Object Oriented Systems, eds. John Wiley & Sons, Inc., Vol. 2(3), 1996.

[16] S. Lerner, A. Hebermann, *Beyond schema evolution to database reorganization*. SIGPLAN Not. Vol 25 No 10, (1990). pp 67-76.

[17] J. Maudes, J. Marqués, C. Hernández, F. García, *Versionado Total de Esquemas a Través de Reglas ECA.* Actas II Jornadas de Investigación y Docencia en Bases de Datos. A. Miguel, J. Cervera E. Marcos M. Piattini eds. Univ. Carlos III, Getafe, Madrid. Julio 1997. pp 129-138.

[18] *The Object Database Standard: ODMG 2.0*. R.G.G. Cattell, D.K. Barry, eds. Morgan Kaufmann. San Francisco, California. 1997.
[19] D. Penney, J. Stein, *Class modification in the GemStone object-oriented DBMS*. SIGPLAN Not. (Proc. OOPSLA'87) Vol 22 No 12, 1987. pp 111-117.
[20] Y. Ra, E. Rudensteiner, *A Transparent Schema Evolution System Based on Object-Oriented View Technology*, Proc. 11th IEEE Intl. Conf. On Data Engeneering, pp 165-172, 1995.
[21] J.F. Roddick. *A survey of schema versioning issues for database systems.* Information & Software Technology 1995, Volume 37 Number 7.
[22] A. Skarra, S. Zdonik, *Type Evolution in an Object-Oriented Database.* Research Directions in Object Oriented Programming. B. Shriver ed. MIT Press, Cambrigde, MA, 1987, pp 393-416.
[23] D. Sjøberg, *Quantifying Schema Evolution.* En Information & Software Technology, Vol. 35, No., 1, pp. 35 – 44, January 1993.
[24] M. Stonebraker, A. Jhingran, J. Goh, S. Potamianos, *On rules, procedures, caching and views in database systems.* En Proc. Of the ACM SIGMOD Intl. Conf. On Management of Data, pp 281-290, Atlantic City, New Jersey, May 1990.
[25] L. Tan, T. Katayama. *Meta operations for type management in object oriented databases – a lazy mechanism for schema evolution.* Proc 1st DOOD, W. Kim, J. Nicolas, S. Nishio eds. Kyoto, Japan, North Holland 1989, pp 241-258.
[26] K. Tsuda, K. Yamamoto, M. Hirakawa, M. Tanaka, T. Ichikawa, *MORE: An Object - Oriented Data Model with a Facility for Changing Object Structures*. IEEE Transactions on Knowledge & Data Engeneering., Vol. 3, Nº 4, Dec. 1991. pp 444-460.
[27] R. Zicari, *A Framework for Schema Updates in an Object-Oriented Database System.* 7th IEEE Intl. Conf. on Data Engineering, April 8-12, 1991, Kobe, Japan.