Derivación de Vistas en un Sistema de Bases de Datos Orientadas a Objetos

Amparo López Gaona
ETSI Telecomunicaciones,
Universidad Politécnica de Madrid
e-mail: agaona@dit.upm.es

Resumen

En el tema de derivación de vistas en bases de datos orientadas a objetos se ha escrito acerca de aspectos estructurales o las nuevas funcionalidades de éstas, sin embargo poco se ha escrito de los aspectos del comportamiento que pueden exhibir los objetos de las vistas. En este documento se presenta una solución general al problema de sobrecarga de nombres en las vistas de BDOs, tomando el prototipo llamado DISCO como base de experimentación.

1 Introducción

Las bases de datos orientadas a objetos (BDOs) integran la tecnología de bases de datos tradicional, que gestiona la representación, almacenamiento y acceso de información persistente y compartida, y el paradigma de programación orientada a objetos [ABDD92].

El mecanismo de vistas en bases de datos relacionales ha demostrado ser de gran utilidad, pues permite entre otras cosas restringir la visibilidad de los datos en las aplicaciones, con lo cual es posible reestructurar la base de datos de acuerdo a las necesidades específicas de una aplicación sin afectar a otras aplicaciones, puesto que no alteran el esquema conceptual. Otras ventajas de las vistas son la independencia de datos, el control de acceso y la seguridad. Dotar de un mecanismo para derivación de vistas en bases de datos orientadas a objetos es tarea bastante compleja debido a la riqueza del modelo de datos, sin embargo se puede lograr mayor funcionalidad que su equivalente en el modelo relacional.

Los sistemas comerciales de gestión de BDOs no incluyen aún mecanismos para tratar con vistas. Sin embargo, a nivel investigación sí existen varios modelos [ShSw89, ScLT91, AbBo91, KiKe95, GBCG97, KuRu96] en los cuales, haciendo analogía con el modelo relacional, se considera a las vistas como clases virtuales. Ahora bien, esto no quiere decir que el manejo de vistas en BDO sea una copia fiel de su equivalente en el modelo relacional, ya que debido a la riqueza del modelo de objetos se tienen que tratar conceptos tales como la ubicación de la vista junto al esquema conceptual, manejo de la población, operaciones de actualización permitidas, así como la nueva funcionalidad que proporcionan las vistas. La definición y gestión de tales conceptos es lo que varía ampliamente en los modelos propuestos.

El comportamiento de los objetos es un punto no tratado, hasta ahora, en este campo de estudio. Este trabajo de investigación se plantea como solución al problema de la inexistencia del tratamiento del aspecto dinámico de los objetos en las vistas.

En la siguiente sección se presenta el estado actual en cuanto al desarrollo de modelos para derivación de vistas considerando la problemática que presentan éstas así como la nueva gama de funcionalidades que proporcionan. Después se presenta el modelo de objetos del sistema denominado DISCO sobre el cual he desarrollado el sistema de vistas que se describe. En la última sección se describen las vistas para el sistema DISCO haciendo énfasis en el aspecto dinámico de los objetos de ellas.

2 Antecedentes y estado actual

Considerando que las vistas deben comportarse como clases, lo primero que se tiene que decidir es su lugar en la jerarquía de clases. La vista puede incluirse en la misma jerarquía de clases reales, ya sea de forma manual [KiKS92], o automática -mediante algoritmos de integración- [AbBo91, KuRu96, ScLT91], o bien se puede tener una jerarquía de vistas aparte de la jerarquía de clases [GBCG97, KiKe95, Souz95] con una relación de derivación de vistas entre una clase existente y una vista. Elegir cualquiera de estas posibilidades tiene además de sus ventajas ciertos inconvenientes que hay que considerar en la especificación y gestión de un determinado modelo.

La siguiente cuestión básica es la relacionada con la población de las vistas. Se puede poblar una clase virtual con objetos existentes en otra clase que satisfagan alguna condición dada, incluyendo subclases de una clase general o clases que tengan métodos o atributos particulares. La población de una clase virtual con objetos que existen en la vista pero no en la base de datos, denominados imaginarios [AbBo91], permite al programador adaptar la base de datos a las necesidades del usuario. Al crear un nuevo objeto, el mecanismo de vistas debe crear y asignarle un identificador de objeto.

La mayoría de los modelos estudiados [ShSw89, HeZd88, ScLT91] sólo consideran vistas que están pobladas a partir de objetos existentes, proporcionando así diferentes formas de verlos, lo cual es útil para soportar objetos con múltiples interfaces y comportamiento dependiente del contexto. En [KuRu96] se describe un modelo cuyas vistas están pobladas a partir de objetos existentes, pero con la característica que se pueden tomar de diferentes clases, no sólo de una. En [Souz95, GBCG97] las vistas se pueblan de objetos existentes, posiblemente con una nueva interfaz en la cual se pueden añadir u ocultar atributos, además se tiene la posibilidad que las consultas retornen un conjunto de tuplas. Para cada tupla en este conjunto se genera un nuevo objeto que se agrega a la extensión de la vista. Hasta ahora los modelos que incluyen las vistas en una sola jerarquía no incluyen la posibilidad de generar nuevos objetos.

Para que las vistas preserven objetos se necesita que el ejemplar de una clase también lo sea de todas las vistas cuyas condiciones de la consulta se satisfacen por tal objeto. Algunos modelos propuestos simulan la creación de ejemplares pertenecientes a múltiples clases, por ejemplo en [Souz95] lo simulan mediante objetos sustitutos que son ejemplares de clases virtuales con un atributo especial cuyo valor es un puntero a su objeto base. En [KuRu96] la simulación se hace mediante la creación de clases que son la intersección de aquellas a las que debe pertenecer el objeto. Los modelos basados en lenguajes funcionales orientados a objetos, como Chimera [GBCG97] y Cocoon [ScLT91] sí soportan generación de ejemplares pertenecientes a clases múltiples, permitiendo con ello que la implementación del mecanismo de vistas que preservan objetos sea más simple y elegante.

La mayoría de las vistas en el modelo relacional no son actualizables debido a problemas de ambigüedad, este problema no existe en el modelo orientado a objetos debido al concepto de identidad de objetos y métodos de actualización específicos. Para que se puedan hacer actualizaciones a la base de datos a través de las vistas se requiere, igual que en el modelo relacional, una correspondencia biunívoca entre cada ejemplar de la vista y el ejemplar correspondiente de la clase sobre la que se define la vista.

Los cambios de esquema que pueden involucrar cambios en las vistas son cambios en la definición de una clase (real o virtual) sobre la cual se definió la vista o bien cambios en la estructura de la jerarquía. Eliminar atributos, métodos o clases puede invalidar la definición de alguna vista y las de aquellas que hereden algunos de sus atributos, en tal caso se debe eliminar la vista o modificar su definición. Las modificaciones en el esquema resultantes de añadir atributos, métodos o clases sólo afectan en caso de que la vista tome en cuenta todos los atributos de la clase. En [KiKe95] mencionan que para realizar automáticamente los cambios a las vistas afectadas por la actualización en el esquema general se podría mantener una relación de dependencia entre las vistas y las clases, para que el sistema avise de los efectos potenciales al actualizar alguna parte del esquema.

Debido a la riqueza del modelo de objetos, se puede dotar de mayor funcionalidad al mecanismo de vistas en el modelo orientado a objetos que en el relacional. Como funcionalidad mínima de una vista se debe tener la equivalente al modelo relacional, por tanto se debe permitir al programador reestructurar la base de datos, modificar el comportamiento y estructura de los objetos, especificar implícitamente valores de atributos en lugar de almacenarlos e introducir nuevas clases en la jerarquía.

Además del uso tradicional de las vistas como ventana dinámica hacia la base de datos, las vistas pueden usarse para la integración de sistemas de bases de datos distribuidas y heterogéneas, pues la definición de vistas integra clases semánticamente equivalentes pertenecientes a diferentes esquemas [KiKe95]. Para poder integrar varias bases de datos es necesario que la definición de las vistas permita la especificación de más de una consulta, cada una a distintas bases de datos.

Existe un prototipo [DeST95] para lograr interoperabilidad entre sistemas para BDOs y para bases de datos relacionales (BDRs). Conecta O_2 con cualquier sistema relacional y construye una vista de objetos de la BDR, donde se pueden federar diferentes esquemas para formar una vista unificada. La arquitectura del sistema puede usarse para migrar valores de datos de BDR o integrar ambos tipos de bases de datos.

Las vistas permiten a los usuarios experimentar con cambios de esquemas sin afectar a otros usuarios. En los modelos de [GBCG97, Rund96] es posible simular la creación de nuevos atributos a las clases debido a que permiten tener atributos no derivados de las clases originales. Por su parte en [KiKe95] se tienen menos posibilidades de cambios en las vistas ya que sólo se pueden tener atributos derivados y no es posible tener nuevos objetos en una vista.

3 El sistema DISCO

El sistema DISCO (por las siglas *Distributed Information System with Cooperative Objects)* es un prototipo de un sistema de bases de datos orientadas a objetos distribuido y cooperativo desarrollado en el Laboratorio de Bases de Datos y Conocimiento del DIT de la UPM [CoBV95].

La arquitectura de DISCO está estructurada en capas. En el nivel más bajo está el núcleo, que soporta las características esenciales del modelo (clases, enlaces, métodos, identificación de objetos, etc.). En un nivel más alto están las herramientas del sistema, con un lenguaje asociado y una interfaz gráfica. Para el núcleo de DISCO todos los elementos que se manejan -clases, enlaces, métodos- son objetos, con su correspondiente identificador de objeto generado por el sistema.

DISCO tiene un modelo de objetos inspirado en los modelos de datos semánticos y en el paradigma de la orientación a objetos, está muy próximo al estándar ODMG3, además de incorporar notables extensiones al referido estándar.

Un esquema es un grafo dirigido acíclico, por tanto los únicos conceptos que contempla el modelo de objetos son nodos y enlaces, que son tipos implementados como clases persistentes contenedoras de todos los ejemplares de ese tipo. Las entidades del mundo real se modelan mediante nodos, denominados estructurados, que agrupan los ejemplares en clases de equivalencia que comparten la misma representación. Los objetos tienen un *oid*, enlaces que modelan su estado y métodos asociados que representan su comportamiento. Los nodos terminales del grafo, denominados atómicos, se clasifican en primitivos y abstractos. Los nodos primitivos son los tipos de datos básicos que contienen los valores. Su equivalente en ODMG-93 son las literales. Los nodos abstractos tienen una estructura encapsulada y oculta, así que su comportamiento sólo puede conocerse mediante la interfaz. El usuario final no necesita distinguir entre nodos atómicos y estructurados, ya que toda la información se le ofrece a un mismo nivel de representación. A nivel extensional los ejemplares se gestionan por el sistema de manera homogénea.

Un nodo es un mundo autónomo, siendo los enlaces sus proveedores de semántica. Un enlace va de un nodo fuente a uno destino y tiene nombre. La unión de un objeto (nodo estructurado) con otro o con una literal (nodo atómico) mediante un enlace de agregación o propiedad tiene su equivalente con los atributos de ODMG-93. La unión de dos objetos mediante un enlace de generalización establece la relación tradicional *is-a* de cualquier modelo de objetos.

Los métodos representan el comportamiento de los objetos sobre los que están definidos. Un método tiene un identificador, un nombre, un nodo propietario y un subgrafo asociado; una vez que se codifica un método, el usuario asigna nodo propietario y decide el lugar en que se producirán los efectos consecuencia de la ejecución el método.

En este prototipo no se había trabajado en el desarrollo de un modelo para derivación de vistas. A continuación se describen las características del modelo de vistas con que he extendido el sistema DISCO.

4 Vistas en DISCO

El tratamiento de la parte dinámica de los objetos (es decir, el tratamiento de los métodos) se ha ignorado o asumido como caso particular de los atributos. Es frecuente encontrar modelos en los que se especifica que es posible eliminar métodos o atributos como si fueran la misma cosa, sin embargo esto no es cierto debido a la necesidad de resolver ciertos problemas; por ejemplo al eliminar un método, aún un atributo, es necesario eliminar todos los otros métodos que hagan referencia a él directa o indirectamente. Por lo anterior, se ha planteado como objetivo solucionar algunos aspectos relacionados con la inexistencia del tratamiento del aspecto dinámico de los objetos en las vistas. En particular, el trabajo se ha centrado en determinar la influencia de la sobrecarga de nombres en las vistas, en cómo asegurar que las vistas formen esquemas consistentes y en determinar cómo encontrar, dada la signatura de un método, un cuerpo único para él.

Para proveer de un mecanismo para la derivación de vistas es necesario, además de un modelo de objetos, un lenguaje para definir las vistas. En la literatura aparecen dos modalidades: como extensión del lenguaje de definición de objetos o como lenguaje independiente. La tendencia de crear un lenguaje para definición de vistas tiene un representante en Multiview que presenta un lenguaje formado por operadores algebraicos [KuRu96]. Como ejemplo de extensión al lenguaje para la definición de objetos se encuentran las hechas al lenguaje funcional y orientado a objetos denominado Chimera y presentadas en [GBCG97]. Para DISCO he creado una extensión al lenguaje GDL (*Graph Definition Language*), este lenguaje es una interfaz al usuario del sistema que ofrece la posibilidad de definir los conceptos del modelo de objetos.

Para este modelo defino una vista como un subesquema del esquema conceptual posiblemente reestructurado, en términos de DISCO una vista es un subgrafo del grafo conceptual. Una vez definido lo que es una vista se plantean las siguientes cuestiones: cómo crearla, cómo relacionarla con el esquema conceptual y cómo ejecutar métodos en ella. Se presenta la respuesta a estas interrogantes en el marco de DISCO.

Las vistas no se incluyen en el esquema conceptual debido a que resulta poco claro tener en una única jerarquía tanto clases como vistas, esto es resultado de la necesidad de incluir clases intermedias carentes de significado para el usuario. Además, no siempre es posible determinar el lugar exacto para la vista en la jerarquía de clases.

En DISCO se permite al creador de la vista seleccionar las clases del esquema conceptual (u otra vista)[1] que desea incluir en ésta con el tipo o extensión, de acuerdo a las necesidades de la aplicación para la cual está pensada ésta. El tipo representa la información intensional dada por la descripción de la estructura y comportamiento de los objetos, es decir, por el conjunto de nodos, enlaces de propiedad, enlaces de generalización y métodos. La extensión está formada por objetos concretos, es decir por el conjunto de valores de los nodos y enlaces.

La extensión de una clase en la vista restringe la extensión de la clase en el esquema conceptual al seleccionar objetos de acuerdo a algún criterio proporcionado. El tipo de la clase en la vista puede tener menos atributos o métodos que en la clase original, o bien pueden aparecer con nombre diferente, o, más interesante aún, puede tener métodos diferentes de los originales.

Al realizar cualquier operación para la creación de la vista, automáticamente se determinan cuales son los métodos que tendrán validez dentro de la vista. Por ejemplo, al ocultar algún atributo del esquema conceptual en la vista debe quedar oculto también cualquier método que tenga acceso (directa o indirectamente) a tal atributo, para evitar que la vista no sea cerrada por necesitar elementos que estén fuera de ella [GBCG97]. Además que sería un esquema incongruente porque en apariencia el usuario no ve el atributo pero puede seguir usando métodos que accedan a él.

En DISCO es posible que los objetos de la vista tengan comportamiento diferente de los objetos del esquema original, mediante la creación de métodos virtuales, con lo cual se dota de mayor poder expresivo a los objetos de la vista y se facilita la creación de aplicaciones desarrolladas sobre una vista.

Existen dos maneras de crear métodos virtuales: una es modificando el código de métodos existentes en la vista y la otra es creando métodos exclusivos para la vista. Es importante aclarar que estas modificaciones de comportamiento se hacen en la vista y que el esquema original no sufre variación alguna, ni se requiere una actualización de la extensión de la clase original.

Al crear un método virtual puede suceder que en la clase donde se cree o en alguno de sus ancestros: a) exista una signatura igual a la proporcionada, b) exista al menos un método con igual nombre proporcionado en la signatura pero con todo lo demás diferente, o c) no exista método con la signatura descrita.

En caso de que exista la signatura completa en la vista, se trata de modificar el cuerpo de un método existente sobreescribiéndolo. Por ejemplo, si se tuviera en una vista un método que imprimiera el nombre de los objetos de tipo persona empezando por el nombre y seguido de los apellidos y se deseara hacerlo empezando por los apellidos,

[1] Toda vista puede verse en última instancia definida en términos de clases del esquema conceptual.

podría sobreescribirse el método original para lograr el efecto deseado. La sobreescritura de un método puede consistir en especificar que el nuevo cuerpo esté vacío con lo cual se está ocultando un método.

Si el nombre del método es lo único de la signatura que existe en la vista, se trata de sobrecarga de métodos. La sobrecarga en la vista implica verificación estática de su signatura, que debe ser compatible con las signaturas de métodos existentes con el mismo nombre y relacionadas a él por herencia de clases. Una signatura incompatible implica el rechazo de la actualización en la vista. Siguiendo con el ejemplo de la impresión de nombres si interesara que en la vista aparecieran ambos métodos las signaturas no podrían tener el mismo tipo de parámetros en el mismo orden.

En la definición de una clase es frecuente que los métodos hagan referencia a otros métodos definidos en la misma clase, por tanto si se redefine (con sobrecarga o sobreescritura) un método se verán afectados todos los métodos que hagan referencia a éste en la clase que se redefina y en las subclases de ésta. En otras palabras, en este caso se modifica el alcance de todos los métodos definidos en las superclases de la clase donde se realizó la redefinición y que tienen el mismo nombre.

La tercera posibilidad para la creación de métodos virtuales es que la signatura del nuevo método no exista en la vista, en cuyo caso se trata de un nuevo método válido en el nodo en que se defina y en sus descendientes, pero sólo dentro de la vista. Con estos métodos virtuales es posible crear nuevos 'atributos' en la vista a partir de atributos en la clase original. Por ejemplo, si en un esquema se tuviera una clase con atributos a y b podría crearse, en la vista, un atributo nuevo obtenido por la aplicación de operaciones aritméticas a los atributos mencionados.

La adición de un método puede llevar a inconsistencias de comportamiento que deben verificarse mirando el código del método. En particular puede haber inconsistencias si el código del nuevo método contiene referencias a métodos o clases que no existen en la vista. En este caso se rechaza el nuevo método.

Debido a lo anterior, se ve la necesidad de tener un algoritmo para resolución de métodos, en el cual además de tomar en cuenta las definiciones explícitas e implícitas en la vista y en el esquema conceptual, se considere la sobrecarga de nombres. Para ello he definido el siguiente algoritmo para encontrar el único cuerpo de un método a partir de su nombre. Este algoritmo aunque se presenta para DISCO es tan general que puede aplicarse a otros sistemas de vistas.

La *clase actual* es la clase de dónde se envía el mensaje en la vista.
El *esquema actual* es la vista.

Repetir:
fracaso es falso.
Repetir:
[Aplicar el algoritmo de búsqueda para la definición del método en la *clase actual*.]
encontrado es falso.
Repetir mientras exista método con nombre igual al buscado o *encontrado* es verdadero.
¿Ambos métodos tienen igual cantidad de parámetros?
Si: ¿El tipo de cada parámetro actual coincide con el tipo del parámetro formal correspondiente?
encontrado es verdadero.
[Termina la repetición.]

[¿Búsqueda exitosa?]
¿*encontrado* es verdadero?
Si: Definición encontrada.
No: ¿Existe un nivel superior en la jerarquía de clases?
Si: [Subir de nivel en la jerarquía.]
clase actual es la clase de nivel superior.
No: [Fracaso en esta jerarquía.]
fracaso es verdadero.
Hasta que haya encontrado la definición del método o *fracaso*.
¿*fracaso* es verdadero?
Si: ¿El *esquema actual* es el esquema conceptual?
Si: [El método no está definido.]
No: [Ir al esquema de dónde se derivó la vista.]
clase actual es la clase de dónde se derivó la vista.
fracaso es falso.
Hasta que se haya encontrado la definición del método o *fracaso*.

5 Conclusiones

En este documento se presentó un modelo para derivación de vistas con gran capacidad expresiva en cuanto al comportamiento de los objetos de las mismas. En este modelo se tienen:

- Vistas derivadas del esquema conceptual pero no integradas en él debido a la imposibilidad de hacerlo automáticamente.
- Vistas pobladas con objetos existentes tomando clases del esquema original, posiblemente con otro nombre, y seleccionando objetos, atributos y métodos del esquema original o añadiendo comportamiento vía sobrecarga o sobreescritura de métodos existentes o bien creando nuevos métodos.

Una de las principales aportaciones de este trabajo es aplicar los conceptos de la orientación a objetos en cuanto a métodos (sobrecarga, sobreescritura) para que los objetos en las vistas puedan presentar, además del comportamiento definido en el esquema conceptual, comportamiento definido explícitamente para ellas.

Otra aportación es la especificación de un mecanismo para resolver sobrecarga de métodos tomando en cuenta que en la vista éstos pueden ser definidos, heredados o derivados, es decir considerando las jerarquía de herencia de métodos del esquema original y de la vista.

Referencias

[AbBo91] Abiteboul, S. and Bonner, A. *Objects and Views.* ACM SIGMOD RECORD. Denver Colorado, Vol. 20, Num. 2 pp. 238–247, June 1991.

[ABDD92] Atkinson, M., Bancilhon, F., DeWitt, D., Dittrick, K., Maier, D. and Zdonik, S. *The Object-Oriented Database System Manifesto.* In Bancilhon, F., Delobel, C. and Kanellakis, P., editors, Building an Object Oriented Database System. The Story of O_2, chapter 1, pp. 3–18. Morgan Kauffman, 1992.

[CoBV95] Costilla, C., Bas, M. and Villamor, J. *The Object Data Model and its Graphical User Interface for an Object Database System.* Proc. of BITWIT'95 Data Management Systems. IEEE Computer Science Press,, 1995.

[DeST95] Delobel, C., Souza dos Santos, C. and Tallot, D. *Object Views of Relations.* Technical report, O_2 Technology, February 1995.

[GBCG97] Guerrini, G., Bertino, E., Catania, B. and García Molina, J. *A Formal Model of Views for Object-Oriented Database Systems.* Theory and Practice of Object Systems, Vol. 3, Num. 4 pp. 157–184, 1997.

[HeZd88] Heiler, S. and Zdonik, S. *Views, Data Abstraction, and Inheritance in the FUGUE Data Model.* In Dittrich, K. R., editor, Lecture Notes in Computer Science, number 334 in Proc. Second Int'l Workshop on Object-Oriented Database Systems, pp. 225–241, 1988.

[KiKe95] Kim, W. and Kelley, W. *On View Support in Object Oriented Database Systems.* In Kim, W., editor, Modern Database Systems, chapter 6, pp. 108–129. ACM Press, 1995.

[KiKS92] Kifer, M., Kim, W. and Sagiv, Y. *Querying Object-Oriented Databases.* ACM SIGMOD Conference on Management of Data, San Diego, CA, pp. 393–402, 1992.

[KuRu96] Kuno, H. A. and Rundensteiner, E. *The MultiView OODB View System: Design and Implementation.* In Ossher, H. and Harrison, W., editors, Journal of Theory and Practice of Object Systems (TAPOS). John Wiley New York, 1996.

[RaKR94] Ra, Y., Kuno, H. and Rundensteiner, E. *A Flexible Object-Oriented Database Model and Implementation for Capacity-Augmenting Views.* Technical Report CSE-TR-215-94, Electrical Engineering and Computer Science Det, Computer Science and Engineering Division, University of Michigan, Ann Arbor, May 1994.

[Rund96] Rundensteiner, E. *Object-Oriented View Technology: Challenges and Promises.* International Symposium on Cooperative Database Systems for Advanced Applications, December 1996.

[ScLT91] Scholl, M., Laasch, C. and Tresch, M. *Updatable Views in Object-Oriented Databases.* In Kifer, M., editor, Lecture Notes in Computer Science, number 566 in Proc. Second Int'l Conf. on Deductive and Object-Oriented Databases, pp. 189–207, 1991.

[ShSw89] Shilling, J. J. and Sweeney, P. F. *Three Steps to Views: Extending the Object-Oriented Paradigm.* Proceedings OOPSLA'89, pp. 353–361, 1989.

[Souz95] Souza dos Santos, C. *Design and Implementation of Object-Oriented Views.* In Revell, N. and Tjoa, A. M., editors, Lecture Notes in Computer Science, number 978 in Proc. Sixth Int'l Conference and Workshop on Database and Expert Systems Applications, pp. 91–102, 1995.