Diseño (2)

Alejandro Teruel

27 de abril 2013

Contenido: Diagramas de clase y diagramas ERE. La codificación de un diagrama de clase.

¿Qué es diseñar?

Todo diseño comienza con una motivación: resolver un problema, una necesidad, una insatisfacción o, más informalmente, una *brecha.*

La brecha puede estar clara o vaga. En general, cuando miramos más de cerca la brecha, nos damos cuenta que no está tan clara como creíamos inicialmente. De allí que nos puede convenir comenzar un proceso de *análisis* de la brecha que nos permite entender el problema mejor e identificar los *requerimientos* que creemos debe satisfacer una solución aceptable. En algunos casos la mejor forma de aclarar la naturaleza de la brecha y los requerimientos de una solución aceptable consiste en diseñar y construir uno o más modelos o prototipos que permitan afinar nuestras ideas sobre la naturaleza de la brecha y los requerimientos de una solución aceptable.

En todo caso, la solución tiene que ser *factible* y tiene que *construirse satisfactoriamente*, lo que suele significar por lo menos con cierto grado de certeza en su culminación en un plazo razonable y a un costo aceptable.

El proceso que lleva de la brecha a su solución no suele ser lineal.

Un diseño comienza con una brecha que puede analizarse para entenderla mejor y sigue con la generación de ideas que pueden llevar a la construcción de soluciones. La actividad de diseñar consiste en generar esas ideas, en refinarlas mediante el uso de modelos y prototipos y la consulta con el cliente y los afectados potencialmente por el artefacto, en evaluar qué tan prometedor es el camino que muestra y en seleccionar, de todas las opciones la que mejor conduce a la solución más satisfactoria dadas las restricciones de recursos...

> En el campo de Ingeniería Civil, se puede comenzar con una brecha como podría serlo la insatisfacción con la Autopista Caracas-La Guaira. Al analizar la insatisfacción nos damos cuenta que operan varios factores: la falta de capacidad de la vía para el volumen de tráfico que requiere usarla, la fragilidad del viaducto número uno y la necesidad estratégica de contar con una vía alterna que vincule a la capital con el puerto principal del país. Podemos precisar estos factores con miras a poder posteriormente evaluar la bondad de opciones de diseño. Las ideas de diseño pueden incluir la construcción de una moderna vía alterna (¿por dónde?), el acondicionamiento permanente de la carretera vieja como vía estratégica para afrontar contingencias, el reemplazo del viaducto número uno, o la construcción de un nuevo viaducto paralelo al número uno, la construcción de un sistema de transporte público, de pasajeros o de carga, alterno a la autopista (metro, ferrocarril), un sistema de incentivos o desincentivos para el uso de la autopista en horas pico, etc. Algunas de estas ideas tendrán que explorarse en mayor detalle, quizás involucren estudios de suelo, planos, cálculos estructurales, costos y modelos, ante de decidir si son viables y cómo se comparan con las alternativas.
>
> El computador personal, la hoja de cálculo, el modelo relacional de bases de datos, los primeros lenguajes de programación, los buscadores de Internet, *Google* y el software en que se fundamenta la red social *Facebook* nacieron para llenar una brecha percibida por sus creadores y desarrolladores. Como ejercicio le recomendamos volver a ver la película *La Red Social* (2010, en inglés *The Social Network*) para identificar la brecha que, según

el guionista, Aaron Sorkin, atiende *Facebook*, así como pensar en qué se diferenció su concepción de otros desarrollos similares de la época).

El rol de los modelos en el diseño

A medida que crece el volumen y la complejidad de un artefacto, más se requieren modelos que permitan abstraerse de los detalles para enfocarse en los lineamientos o fundamentos de la estructura que permita visualizar y estudiar las bondades, retos y deficiencias de su estructura, su desempeño , su costo o el aspecto del producto final:

> "Un modelo es la abstracción de una cosa real. Tu modelo es una simplificación de un sistema real, una simplificación que permite que el el diseño y la viabilidad del sistema sea entendido, evaluado y críticado más rápidamente que si tuvieras que trabajar con el sistema real."
>
> Russ Miles, Kim Hamilton: *Learning UML 2.0.* O´Reilly, 2006

En las disciplinas más veteranas como Ingenería Civil, Ingeniería Mecánica y Arquitectura, juega un rol crucial en el diseño el *plano*, elemento que sirve para la discusión de ciertas propiedades de diseño, del artefacto concebido en el diseño y como referencia indispensable para el proceso de construcción de la autopista, viaducto, motor, edificación o artefacto correspondiente. El plano se complementa con otros elementos como modelos a escala o modelos computacionales necesarios o útiles para aclarar las propiedades futuras del artefacto. La complejidad de los arcos de la Catedral de la Sagrada Familia proyectada por el destacado arquitecto Gaudí rebasaba las posibilidades de cómputo de la época: Gaudí ingeniosamente resolvió la complejidad con un modelo en que invirtó el elemento estructural -construyó una red de cuerdas de las que colgó pesos, las formas que tomaron las cuerdas mostraban una versión de cabeza de las formas que debían tener los arcos.

UML

En este curso aprovecharemos algunos elementos de una notación, *Unified Modeling Language*, que nos permite elaborar diagramas para modelar software, así como otros profesionales usan planos para modelar otros artefactos ingenieriles.

Los profesionales pueden usar un plano apenas bosquejado para comunicar las ideas principales de un diseño o pueden realizar planos muy meticulosos que documentan y especifican el detalle de un diseño. Similarmente, Martin Fowler, un destacado proponente de métodos ágiles, indica que podemos usar los diagramas UML con diversos grados de formalidad: podemos garrapatear un diagrama de clases en una servilleta para explicarle a un colega una concepción inicial o un abordaje a un diseño, o podemos usar una herramienta automatizada para documentar formalmente un software complejo de requerimientos exigentes y vitales.

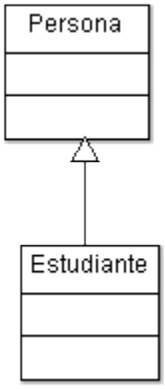
El diagrama de clases

El *diagrama de clases* es un modelo de la estructura estática conceptual del dominio de una aplicación o de un software orientado por objetos -en el primer caso el diagrama de clases ayuda a *analizar* los conceptos claves de ese dominio y sus interrelaciones, en el segundo caso ayuda a *diseñar* el software y a documentar parte de ese diseño. Esta dualidad *concepto del dominio - abstracción de software* es, por un lado una fortaleza (se acorta la distancia semántica aparente entre el *concepto* y su *simulacro digital*) y por otra una fuente permanente de posible confusión semántica -sobre todo cuando el

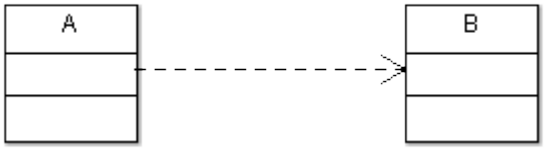
concepto y su simulacro discrepan en la calidad e intensidad de sus rol como agentes en sus respectivos mundos: para mencionar un notorio ejemplo de las viejas interfaces gráficas de Apple Macintosh, las *papeleras* de la vida real no expulsan discos.

Concretamente un diagrama de clases muestra los nombres de clases, (opcionalmente) sus atributos y/o sus métodos (más usualmente los públicos) y una indicación de los tipos de *acoplamiento* que se puede presentar entre esas clases.

Explicitamos varios tipos de acoplamientos. Un tipo de acoplamiento, característico en el software orientado por objetos es el inducido por la herencia entre clases:



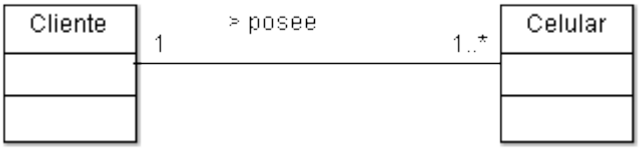
El acoplamiento más débil entre clases se denomina *dependencia* entre clases y es cuando un objeto de la clase A puede llegar a invocar a un método de un objeto de la clase B:



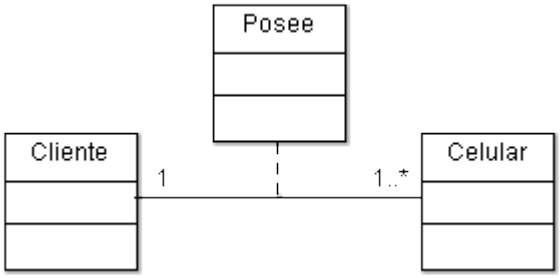
Este caso representa, por ejemplo, perfectamente el uso de clases del paquete java.math

Más interesante es la *asociación* entre clases. Una clase está asociada con otra si nos interesa explicitar asociaciones entre objetos de las dos clases:

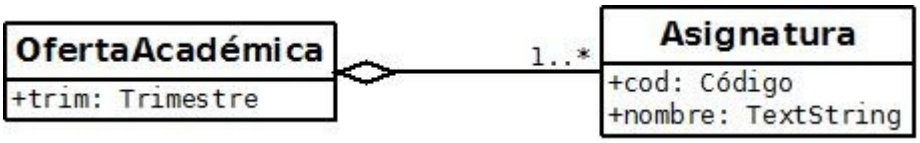
Al igual que en un diagrama ER, podemos especificar la multiplicidad de la asociación:



En este caso un cliente *posee* uno o más celulares, y cada celular tiene un único dueño. A diferencia del diagrama ER, la asociación puede a su vez ser una clase:



Tenemos dos tipos aún más fuertes de acoplamiento, *agregación y composición* correspondientes a relaciones tipo *tiene-un* (en Inglés, *has_A*). La diferencia estriba que el *agregante* no es dueño de los *agregados*, mientras que la *composición* es dueña de los *componentes*; varios *agregantes* pueden compartir un *agregado*, por lo que éste puede permanecer si se destruye un *agregante* -en cambio la *composición* es dueña de los *componentes* y si desaparece la *composición* también desaparecen los *componentes*. Incluir a un objeto como un atributo de una clase es establecer un acoplamiento de composición.



La OfertaAcadémica de un trimestre contiene una o más asignaturas. Una composición se representa en forma similar pero con un rombo relleno. Note que en este caso, aparte del nombre de las clases, hemos incluido algunos atributos.

El nivel de acoplamiento entre clases también depende de la los atributos y métodos públicos y protegidos de la clase.

El lector atento habrá notado la similitud entre el diagrama de clases y un diagrama ER o ERE. Las diferencias fundamentales son:

1. Un diagrama ER o ERE es un modelo de *datos* correspondientes a entidades y sus relaciones; un diagrama de clases es un modelo de clases, entes que encapsulan *datos* y *acciones*. Dicho de otra manera, las clases encapsulan atributos ***y*** métodos: las entidades y relaciones sólo encapsulan atributos, siendo los únicos "métodos" aplicables los que conforman SQL.
2. Las asociaciones en el diagrama de clases puede, a su vez, ser clases;
3. Un atributo de una clase puede, a su vez, ser un objeto de una clase (forma implícita de composición entre clases);
4. En el diagrama de clases se pueden representar relaciones de dependencia que no son representables en diagramas ER o ERE.
5. En el diagrama de clases se pueden representar relaciones de herencia que no son representables en diagramas ER, aunque sí lo sean en ERE.
6. En ER, hay tablas implícitas (tabla por entidad o relación, donde cada fila es el equivalente de un "objeto" -pero no pueden haber filas repetidas) -en POO el contenedor de los objetos de una clase (o asociación entre clases) tiene que crearse explícitamente. Los objetos viven "independientemente" en POO, dentro de "tablas" en RDBMS...

Una decisión de diseño: ¿qué se lleva a una base de datos?

El mundo de las bases de datos se desarrolló para facilitar el registro, estructuración, selección, combinación y consulta, modificación y eventual borrado de datos *persistentes,* es decir capaces de perdurar más allá de la ejecución o la vida de los programas que los aprovechan. El mundo del software orientado por objetos no nació con la preocupación de la persistencia sino nació del desarrollo de una metáfora *animística*, todo objeto contendría entonces, encapsulados, datos y esquemas de comportamientos: los objetos son como perros entrenados, son distinguibles, tienen características físicas propias y responden de manera predeterminada a las órdenes que se les han dado y que saben cómo ejecutar. Detrás de los primeros lenguajes orientados por objetos están los principios de simulación (e.g. *Simula 67*) y la concepción derivada de la arquitectura *pipes and filters* de Unix donde los programas *(filters)* se comunicaban entre si gracias al acoplamiento del archivo de salida de uno al archivo de entrada de otro *(pipes),* concepción que derivó en crear componentes que serían como pequeños programas (y como tal encapsulan variables y procedimientos) que pueden, *hablarse (talk)* entre si en su pequeño *(small)* mundo (e.g. *Smalltalk).* Dicho de otro modo, la orientación a objetos nació de una preocupación por estructurar mejor los programas estáticamente organizados en cada vez más rutinas y las bases relacionales de datos nacieron de la preocupación de estructurar mejor los datos, conservando su integridad:

> "OO primarily focuses on ensuring that the structure of the program is reasonable (maintainable, understandable, extensible, reusable, safe), whereas relational systems focus on what kind of behaviour the resulting run-time system has (efficiency, adaptability, fault-tolerance, liveness, logical integrity, etc.)"

Por ende clases del diagrama de clases en las que destacan los atributos y cuya utilidad puede sobrevivir al contexto de un programa, son claramente candidatas para ser implementadas como tablas en un manejador relacional de bases de datos, mientras que clases en las que destacan los métodos y se esconden los atributos o éstos se convierten en secretos, son claramente afines a tipos abstractos de datos y candidatas a componentes de programas.

Un caso sencillo de clases a tablas relacionales

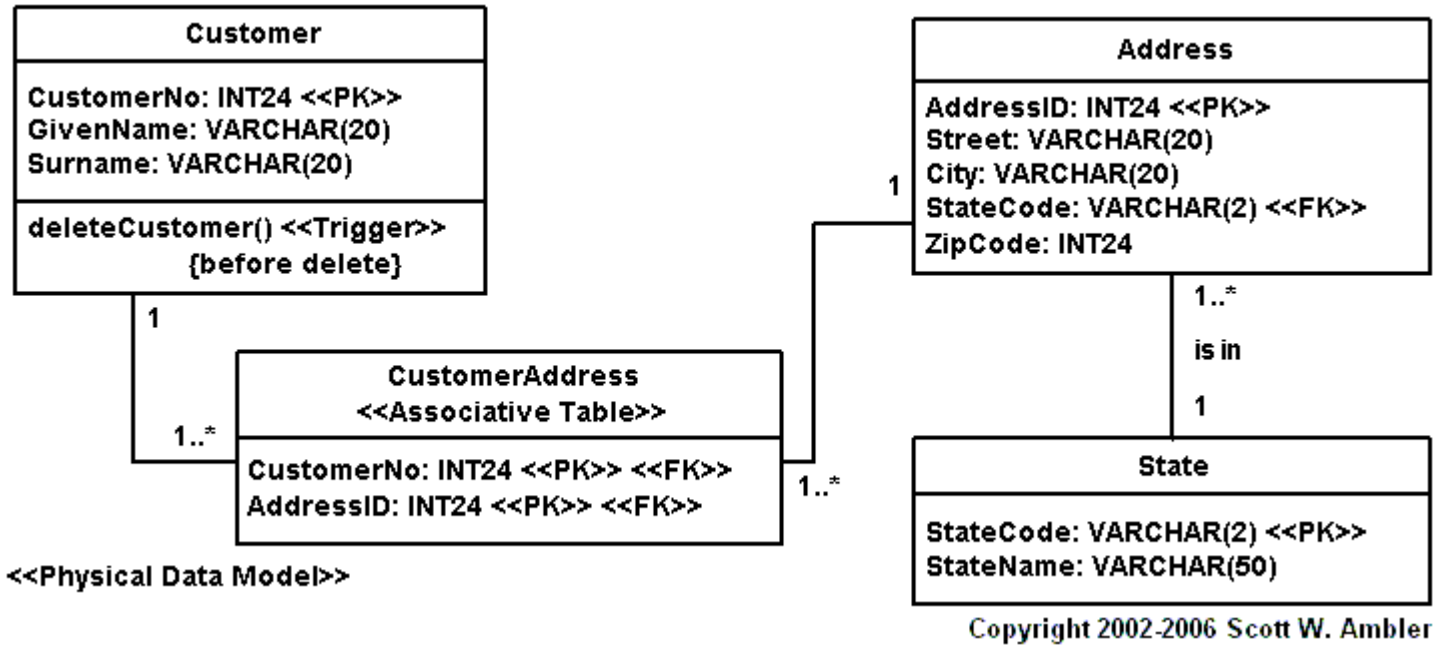
Si se detecta que:

- Los objetos de una clase contienen datos en sus atributos que claramente deben persistir en el tiempo;
- Pueden haber muchos de esos objetos;
- Los métodos son pocos y pueden expresarse como operaciones CRUD (*Create, Read, Update, Delete*) o en términos de operaciones tipo SQL;
- Se puede identificar o agregar un atributo clave entre los atributos de la clase que permita identificar de manera única cada objeto de la clase

entonces esa clase es fuerte candidata a implementarse como tabla relacional. Si además esa clase está vinculada a otras con las mismas características mediante asociaciones, esa porción del diagrama de clases es candidata a ser vista como un modelo de datos. En un primer paso se convierte la clase en una tabla y los atributos públicos y protegidos en columnas.

UML permite etiquetar el diagrama o sus componentes con <<estereotipos>> para indicar que tienen

un significado especial:



En la figura anterior observe como se usan estereotipos asociados a un modelo físico de datos asociado a la tecnología RDBMS para:

- Indicar que el diagrama corresponde a un <<*modelo físico de datos*>>;
- Una <<*tabla de asociación*>>;
- Los atributos identificados como parte de la clave primaria (<<*PK*>>) de una fila
- Atributos que son clave extráneas (<<*FK*>>).
- Un <<*trigger*>>

De clases a POO

> "The trickiest issue for mapping a UML class model to OO databases is the handling of associations. An association relates objects of the participating classes and is inherently bidirectional. For example, the WorksFor association can be traversed to find the collection of employees for a company, as well as the employer for an employee.
>
> By their very nature, associations transcend classes and cannot be private to a class. Associations are important because they break class encapsulation."
>
> Michael Blaha en entrevista de Robert Zicari, 2011

El desarrollador puede decidir qué partes de su diagrama se programarán conservando la orientación a objetos. En este caso, surge la pregunta: ¿cómo se codifican las asociaciones?

En el caso de asociaciones dirigidas, es decir donde está previsto que la navegación se hace en una sola dirección de la asociación, es natural pensar en incluir en la clase fuente de partida de la asociación, un apuntador a un objeto de la clase destino de la asociación (si la aridad es *1 a muchos*) o una lista de apuntadores (si la aridad es *muchos a muchos*). En caso de bidireccionalidad, agregamos otro apuntador

o lista de apuntadores en los objetos de la segunda clase. En caso de aridad "*m a n*" también puede crearse una tabla o alguna otra estructura *contenedora* de pares de apuntadores.

De clases a entidades y relaciones: ¿qué hacemos con los métodos?

Si las demás características de la clase la hacen una fuerte candidata a ser implementada como tabla en una base relacional de datos, conviene estudiar esos métodos para ver si pueden ser manejados como *stored procedures*.

Otra opción es estudiar la posibilidad de extraer la información persistente necesaria de la base de datos, unos objetos en el mundo del programa orientado por objeto con la estructura necesaria para implementar más eficientemente el método, actualizando posteriormente la base de datos si fuera necesario. Esta traducción de un mundo a otro debe ser acometido con cuidado pues puede:

- Crear problemas de eficiencia, en la traducción, o en la ejecución de la parte correspondiente a uno o ambos mundos;
- Dejar sembrado un fuerte potencial para una futura inconsistencia por desactualización -por ejemplo si cambia el modelo datos y se modifican las tablas correspondientes en la base de datos, ¿cómo y cuándo se entera de ello el responsable del programa orientado por objetos y en qué momento se podrá llevar a cabo la modificación ("mantenimiento adaptativo") del proceso de traducción y las porciones del programa afectados por esos cambios?

¿Cómo afecta la herencia la traducción de clases a tablas?

El modelo relacional no soporta la relación de herencia. Hay tres formas de manejar la herencia que puede aparecer en un diagrama de clases al llevarlo al mundo de la base de datos relacional:

- Cada clase se lleva a una tabla distinta. Así, si la clase *Estudiante* hereda de la clase *Persona* creamos una tabla para *Estudiante* y otra para *Persona.* Los atributos de cada clase pasan a ser columnas en la tabla correspondiente. Necesitamos que todas las tablas tengan una clave común para poder llevar a cabo los *joins* necesario. Note que para conocer todos los datos asociados con un estudiante necesitamos recorrer las entradas correspondientes a su clave en la tabla Persona y las correspondientes a su clave en la tabla Estudiante.
- Crear una tabla sólo para las clases que son hojas del árbol de herencia. Así si las clases *Estudiante* y *Profesor* heredan de la clase *Persona,* crearíamos una tabla para *Profesor* y otra para *Estudiante* -no crearíamos una tabla para *Persona*. Los atributos de las superclases se agregan como columnas en las columnas de las tablas correspondientes a las subclases hojas. Así, en el ejemplo, todos los atributos de la clase *Persona* pasan a ser columnas tanto de *Estudiante* como de *Profesor.*
- Creamos una tabla para toda la jerarquía. Así en el caso que las clases *Profesor* y *Estudiante* hereden de *Persona*, crearíamos una única tabla *Persona+*que contiene como columnas todas las derivadas de los atributos de *Persona,* todas las derivadas de los atributos de *Profesor* y todas las derivadas de los atributos de *Estudiante.* En algunos casos podemos agregar una columna adicional para discriminar si la fila representa un *Estudiante* o un *Profesor.* Note que en este caso, la tabla *Persona+* puede llegar a requerir de muchas entradas nulas.

Evidentemente cada una de estas opciones tiene sus ventajas y sus desventajas, las cuáles se exacerban

con la profundidad de la jerarquía de la herencia, así como la frecuencia de encontrar objetos que no son de la clase raíz ni de las clases hojas de la jerarquía.

De una base relacional de datos a un programa orientado por objetos

En ocasiones al desarrollador le toca adaptarse al legado de una base relacional de datos. En este caso puede optar por extraer datos de la base de datos, traduciendo tablas a clases y ocasionalmente enriqueciendo tales clases con métodos pertinentes para la aplicación a desarrollar. En la literatura las variantes de esta idea se les conoce como *patrones de diseño* como *Table Data Gateway, Row Data Gateway* y *Active Record.*

Precondiciones, postcondiciones e invariantes de clase

UML también preve la posibilidad de expresar precondiciones y postcondiciones sobre los métodos de una clase e invariantes de clase. Para ello permite agregar anotaciones en el *Object Constraint Language (OCL)* entre *{llaves}* al lado de los métodos (típicamente {pre: ....}, {post: ...}) en las clases o en los comentarios que pueden adornar los diagramas de clase [Miles y Hamilton, 2006].

Note que, en caso de decidir implementar las clases como tablas relaciones, las invariantes de clase puede transformarse en *restricciones de integridad.* En realidad no hay razón para que posibles restriccciones de integridad que sobrepasen el contexto de una clase no puedan introducirse como comentarios en un diagrama de clases.

Lecturas adicionales

- Russ Miles, Kim Hamilton: *Learning UML 2.0.* O´Reilly, 2006. Los capítulos 4 (*Modeling a System's Logical Structure: Introducing Classes and Class Diagrams*) y 5(*Modeling a System's Logical Structure: Advanced Class Diagrams*) introducen la notación e ideas principales asociadas a los diagramas de clase.
- Bernd Bruegge, Allen H. Dutoit: *Object-Oriented Software Engineering: Using UML, Patterns and Java.* 3° edición, Prentice-Hall 2010. Las secciones 10.4 y 10.5 contienen una excelente introducción al tema de cómo llevar los elementos de un diagrama de clases a estructuras Java o a un esquema relacional.
- Michael Blaha, William Premelani: *Object-Oriented Modeling and Design.* Prentice-Hall, 1998 (hay una segunda edición publicada en el 2004 y escrita por Michael Blaha y James Rumbaugh, que no he tenido oportunidad de revisar). Entra en detalle en los tópicos asociadoa a desarrollar programas orientados a objetos para aplicaciones con bases relacionales de datos.

También pueden resultar de interés unas referencias un poco más avanzadas sobre el tema de lo que se ha llamado *impedancia* entre el paradigma de objetos y el paradigma relacional:

- Ted Neward: *The Vietnam of Computer Science*. The Blog Ride, 26 de junio 2006. http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx Consultado el 26/04/2013

- Scott Ambler. *The Object-Relational Impedance Mismatch*. **AgileData.org: Techniques for Disciplined Agile Database Development,** 2012 http://www.agiledata.org/essays/impedanceMismatch.html Consultado 26/04/2013
- Scott Ambler: *Mapping Objects to Relational Databases: OR Mapping in Detail* **AgileData.org: Techniques for Disciplined Agile Database Development,** 2012 http://www.agiledata.org/essays/mappingObjects.html. Consultado 26/04/2013
- Robert V. Zicari: *Agile Data Modeling and Databases.* ODBMS Industry Watch, 25 enero 2011. http://www.odbms.org/blog/2011/01/agile-data-modeling-and-databases/ Consultado 26/04/2013 Contiene el texto de una muy breve entrevista a Michael Blaha donde éste resalta la importancia y dificultad de la implementación de las asociaciones entre clases, además de una secuencia de videos donde muestra cómo hacer elaborar un diagrama inicial de clases como parte de las actividades de análisis.