Computación y Sistemas Vol. 7 Núm. 4 pp. 224-238

Un Método para el Diseño de la Base de Datos a partir del Modelo Orientado a Objetos

A New Method to the Data Base Design from the Object Oriented Model

Dra. Anaisa Hernández González
Instituto Superior Politécnico José Antonio Echeverría (ISPJAE)
Directora del Centro de Formación de Profesores de Ciencias Informáticas, Cuba
e-mail: anaisa@ceis.ispjae.edu.cu
Teléfono(537) 2607912, (537) 2606418
Fax (537) 271575

Artículo recibido en septiembre 9,2003, aceptado en mayo 18, 2004

Resumen

La mayoría de las aplicaciones de software que se desarrollan en el mundo requieren del almacenamiento y gestión de grandes volúmenes de información. Con el auge del paradigma de la orientación a objetos, este proceso ha tomado nuevas dimensiones porque la persistencia es de objetos no solo de datos. En este trabajo se sistematizan los pasos para el diseño de la base de datos, a partir del análisis de un problema utilizando este enfoque. Se incluyen recomendaciones que permiten obtener el comportamiento estático y dinámico de los objetos; así como el procedimiento a seguir para interpretar toda la información, representada visual y textualmente, en función de este diseño. Se propone una estructura de una capa persistente de clases que incluye dos subcapas que responden al modelo de persistencia propuesto y que aíslan a las clases del dominio de la forma en que se almacenan y recuperan los objetos.
Palabras Clave: Diseño de la base de datos, Orientación a objetos.

Abstract

Most of the software applications that are developed in the world require of the storage and administration of big volumes of information. With the peak of the object-oriented model, this process has taken new dimensions because of the persistence is in objects, not only in data. In this work are defined the steps for the database design, starting from the object-oriented analysis. The article proposes a structure of a persistent layer of classes that includes two layers that they isolate to the domain classes in the way in that the classes are stored and the objects are recovered.
Keyword: Database design, object-oriented

1. Introducción

Para producir software de calidad es necesario el desarrollo de un proceso disciplinado. Esta calidad no se logra con el esfuerzo puntual en alguna o algunas de las fases o etapas del ciclo de vida, es un trabajo consciente en la aplicación de técnicas ingenieriles, junto al establecimiento de métricas de calidad desde el inicio de su construcción.

La calidad es un tema complejo por la cantidad y diversidad de aristas que involucra, por lo que este trabajo se propone abordar un punto dentro de todos ellos, crítico en la mayoría de las aplicaciones que se desarrollan en la actualidad, *el diseño de la información almacenada*.

El paradigma de la orientación a objetos ha provocado una revolución en los conceptos de la Informática siendo el cambio más importante desde que surgieron los métodos estructurados, impactando en mayor escala en los lenguajes de programación de alto nivel y en las metodologías de análisis y diseño de sistemas informáticos.

En el campo de las bases de datos en la actualidad de comercializan productos que soportan el modelo relacional, gestores de objetos y sistemas de gestión relacionales que extienden sus capacidades hacía el modelo orientado a objetos (MOO).

Esta diversidad hace difícil el diseño de la base de datos (BD) cuando se modela un sistema siguiendo el paradigma de la orientación a objetos. El tema ha comenzado a ser tratado con mayor profundidad en los últimos años, aunque todavía quedan puntos en los que hay mucho que hacer, como se verá más adelante.

El tema ha comenzado a ser tratado con mayor profundidad en los últimos años, aunque todavía quedan puntos en los que hay mucho que hacer, como se verá más adelante.

Desde el punto de vista estructural o estático hay bastante consenso ya que por lo general están definidos los conceptos de clase, atributo, relaciones entre clases, herencia y agregación, pero no siempre se tienen en cuenta todas las restricciones estáticas que validan la inserción inapropiada de elementos. El comportamiento dinámico de los objetos es abordado insuficientemente ya que, aunque se incluyen casi siempre diagramas que reflejan estas características, no se construyen en función de su aporte al diseño de la BD, ni se define cómo utilizar la información que contienen en la etapa de implementación.

Las propuestas de metodologías por lo general comienzan con una definición de los principales conceptos que caracterizan al enfoque, pero salvo excepciones como [11][20], se basan en definiciones informales. Muchos métodos se han introducido sin formalización [13][14].

Desde el punto de vista metodológico, sobre el diseño de la base de datos la mayoría de las metodologías no incluyen un modelo de persistencia. Esta situación era lógica si tenemos en cuenta la poca fortaleza de los gestores orientados a objetos. Hay una tendencia a convertir clases a tablas, pero teniendo en cuenta, y no de forma completa, la parte estructural. Por lo general olvidan que los objetos en su definición incluyen el comportamiento, y que el modelo al que se transforma (el modelo relacional) centra su atención en los datos [3][11][18][19][24]. Además las metodologías definen por lo general los pasos para construir una aplicación, pero no cómo diseñar la BD; ofrecen poca o ninguna recomendación para construir los modelos y diagramas que incluyen, y resultan insuficientes en la explicación de cómo transformar la información contenida en los diagramas hacía el medio de almacenamiento.

Algunos trabajos muestran una inclinación hacía la definición de una capa de interfaz entre la información almacenada y la aplicación. A esta capa se le conoce como capa persistente y aisla a las clases del dominio de la forma en que se almacenó la información [2][3][11][16][18][25].

A partir de ésta situación, se definió como **problema de investigación**: **"***El desarrollo de un método, denominado DIBAO (DIseño de la BAse de datos a partir de un modelo orientado a Objetos) que se base en un formalismo y que garantice: la transformación de la semántica asociada a un objeto hacía el medio de almacenamiento que se utilice, y conservar en la transformación las principales características del paradigma de la orientación a objetos."*
En este artículo se describe la esencia de este método, explicado con un mayor de detalle en [12].

2. Método DIBAO

El estado del arte en el campo de las bases de datos de objetos (BDO), condiciona las alternativas del diseño de la base de datos (BD). El método para el DIseño de la BAse de datos a partir del modelo orientado a Objetos (DIBAO), está compuesto por dos partes: un modelo de persistencia y una capa persistente de clases (Figura 1). El modelo de persistencia describe los pasos para el diseño de la BD, las herramientas que se deben utilizar en este proceso, cómo convertir las clases al medio de almacenamiento seleccionado y cómo interpretar la información contenida en los diagramas y especificaciones textuales, en función de este diseño. La capa persistente de clases incluye las subcapas de especificación y de interfaz, que se encargan de definir las clases para describir las nuevas especificaciones asociadas a las relaciones entre los objetos, que se describen en el diccionario de clases; y la relación con el medio de almacenamiento, respectivamente.

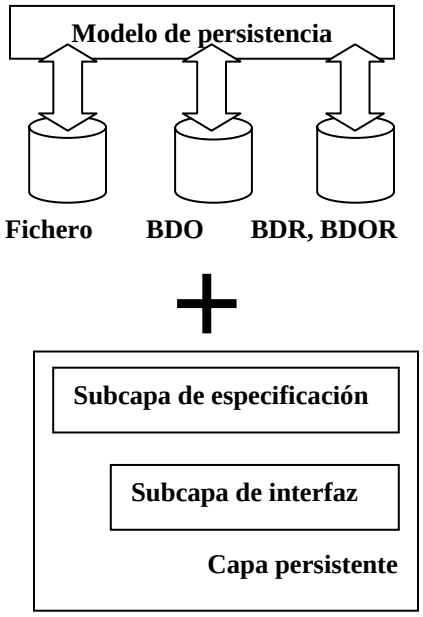

Fig. 1. Estructura del método DIBAO

2. Modelo de Persistencia

Independientemente del medio de persistencia seleccionado para almacenar los objetos persistentes, se deben realizar un grupo de pasos que permiten completar la semántica de los objetos en aspectos que son importantes referentes a su estructura estática y comportamiento dinámico. Los pasos que se proponen son:

1. Definir las clases persistentes. Todas las clases identificadas en el dominio del análisis no son persistentes. La persistencia es la capacidad de un objeto de mantener su valor en el espacio y en el tiempo. Es responsabilidad del diseñador definir cuáles clases son las que deben ser persistentes. En este proceso la autora de recomienda aplicar algunas reglas, ellas son:

1. Cuando una clase que está formada por otras clases es persistente, automáticamente las clases componentes también son persistentes. Lo contrario no se cumple necesariamente.
2. Cuando una clase hija de una jerarquía es persistente, automáticamente son persistentes sus ancestros en el árbol de jerarquía. Lo contrario no se cumple necesariamente.
3. Cuando se define como persistente a una clase que agrupa a objetos de un mismo tipo de clase base (se refiere a las clases listas, colecciones, registros), entonces automáticamente son persistentes todas las clases hijas a partir de la clase base, incluyendo a la clase base.
4. Cuando hay herencia múltiple, esta debe ser resuelta antes si el medio de almacenamiento a utilizar no soporta este concepto.

En fases anteriores dentro del ciclo de vida del desarrollo de un proyecto, puede que se hayan definido clases que coordinan el trabajo de varias clases, por lo que el comportamiento que de ellas interesa es el asociado a la función controladora. Estas clases, por concepto, no son persistentes.

Las clases persistentes por lo general tienen como origen las clases clasificadas como entidad porque ellas modelan la información y el comportamiento asociado de algún fenómeno o concepto, como una persona, un objeto del mundo real o un suceso.

2. Clasificar las clases y los atributos. Las clases en este trabajo se clasifican en dependencia de los valores de los atributos que la integran en:

- Simples: todos los atributos toman valores atómicos.
- Compuestas: es el resultado de la agrupación de varias clases para formar una entidad conceptual con significado en el dominio. Este tipo de asociación puede ser una composición o una agregación y puede tener otros atributos que tomen valores atómicos asociados a la relación.
- Complejas: son clases que tienen uno o varios atributos que referencian a otros objetos de otras clases o de ella misma. La clasificación de una clase como compleja sugiere que la relación de ella con respecto a las clases que contiene es referencial, es decir, un cambio en la primera no afecta a la segunda. Tradicionalmente se reconocen como las relaciones de asociación entre dos o más clases.

Existen varias clasificaciones de los atributos de una clase. En este trabajo se parte de la clasificación propuesta en [22], que identifica a los atributos en alguna de las siguientes categorías: dinámico (cambia su valor en el tiempo), estático (no cambia su valor una vez creado el objeto) o derivado (su valor esta dado por una fórmula de derivación formado por otros atributos).

3. Realizar el diagrama de clases persistentes. La mayoría de las características de la perspectiva estática son capturadas gráficamente en el diagrama de clases (DC) cuya notación toma como base la del diagrama de clases de UML [24], aunque se añaden nuevas características usando estereotipos y notas. Se adiciona, usando estereotipos, la definición de OO-Method [22] de dimensión estática/dinámica (E/D), para las relaciones entre la clase compuesta y cada una de sus clases componentes, porque añade una restricción de integridad no representada en la notación UML

Es un diagrama de clases en el que solo aparecen las clases persistentes, de las que hay que expandir detalles estructurales en cuanto a los atributos especificando tipo de dato, valor inicial, rango de valores, si tienen un valor común para todas las instancias de la clase en el momento de creación y otras restricciones de dominio. Además, hay que buscar patrones comunes que compliquen el diseño físico para darles solución. Estamos haciendo referencia a las asociaciones cíclicas, relaciones n-arias, de m:n y de 1:1 y la herencia múltiple.

Aunque el concepto de llave no existe en el MOO porque allí se trabaja con el concepto de identificador de objeto que no está ligado a los atributos; como la variante de medio de almacenamiento más utilizada es la conversión hacía una BD relacional u objeto/relacional, se debe definir para las clases entidad el conjunto mínimo de atributos que la identifican, es decir, las llaves primarias. Es importante precisar que:

En el extremo de la clase compuesta como máximo la cardinalidad es 1, para la agregación puede **ser mayor.**

- En el extremo de la clase componente la cardinalidad mínima por defecto es 1, la máxima depende de: si no es una agrupación de objetos es 1, si es una agrupación es m y si la clase compuesta tiene más de un atributo de igual tipo (por ejemplo, la clase matrimonio tiene dos atributos de tipo persona, una para cada padrino), la relación es con la clase y con la cardinalidad máxima igual a la cantidad de atributos iguales.
  Estos valores por defecto son lo mínimos que se pueden tomar a partir de las relaciones reflejadas en la definición de las clases.

- En el caso de las clases complejas, son válidas las recomendaciones anteriores en cuanto a la cardinalidad, excepto en el extremo de la compleja que la cardinalidad mínima debe ser <0,1> porque no es una relación de todo-parte.

En la figura 2 se muestra un ejemplo sencillo de un Diagrama de clases persistente para el concepto libro, en el que se refleja que un libro es un concepto que se forma de agrupar su título, ISBN, Autores y Ediciones (La clase libro se comporta como clase compuesta con relaciones de Agregación y Composición); que es publicado por una única Editora a la que se dan derechos exclusivos (La clase libro se comporta como una clase compleja porque existe una asociación con la clase Editora).

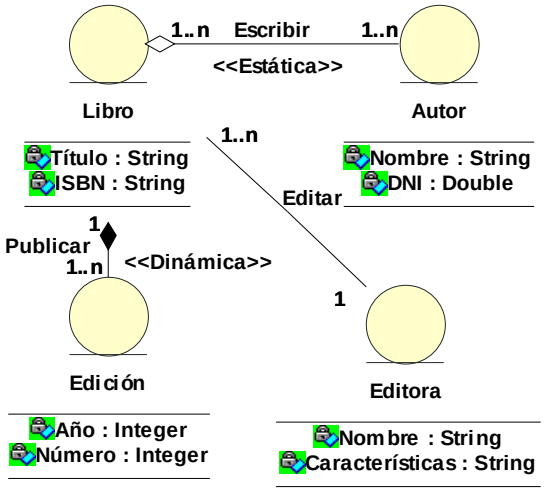

Fig. 2. Ejemplo de diagrama de clases persistente

4. Realizar el diagrama de estado. Para mostrar la dinámica del comportamiento de un sistema, se ha hecho uso durante años de los diagramas de transición de estado (DE) [7]. Este método recomienda utilizarlos en función del diseño de la BD, construyéndolos para aquellas clases que posean atributos dinámicos pues en función de estos es que están los posibles estados por los que transita un objeto.

La descomposición en varios niveles del DE, utilizando los estados agregados, es un tópico poco definido en cuánto a cuáles criterios utilizar para agrupar. En [8] se recomiendan los estudios de George Miller que indican la cantidad de 7±2 elementos por nivel y que lo general esté en los niveles más altos y lo particular en los más bajos. El método DIBAO propone dos criterios para agrupar por niveles. Un criterio está relacionado con los atributos, de manera que si existen varios eventos que afectan a un mismo atributo ubicando al objeto en estados diferentes, con todos estos estados podría definirse uno agregado. Otro criterio sería agrupar a estados que están fuertemente acoplados.

Los diferentes niveles de diagramas deben estar balanceados en cuanto a las transiciones de entrada y salida cumpliéndose las siguientes reglas [10]:

- Para el caso de las transiciones de entrada es necesario que cada uno de los eventos en OR, de la lista de eventos de cada una de las transiciones entrada al estado agregado, esté formando parte de la lista de eventos de al menos una de las transiciones que parten del estado inicial hacía los subestados.
- Para el caso de las transiciones de salida debe cumplirse: cada subestado tendrá una transición al subestado final y cada una de estas, contendrá entre su lista de eventos las listas de eventos de todas las transiciones de salida del estado agregado relacionadas entre sí por el operador lógico OR.

En la especificación de los DE es importante clasificar los atributos dinámicos de acuerdo a la forma en que cambian su valor en el tiempo, por esto este trabajo toma como base la propuesta descrita en [22], que clasifica a los atributos dinámicos en:

a. Cardinales: el efecto en el atributo es el incremento/decremento en 1 o una cantidad dada.
b. Característicos de un estado: el valor que toma el atributo es independiente del valor anterior.
c. Perteneciente a una situación: el atributo toma valor en un dominio limitado. El nuevo valor depende del interior, es decir, estando en un estado dado solo se pueden tomar determinados valores.

Las acciones se corresponden con los eventos definidos en el DE. En las clasificaciones a y b, el efecto de la acción está descrito dentro de las acciones a ejecutar en el estado. Para los atributos clasificados como c, la especificación describe restricciones de integridad dinámica de la clase que se obtiene cuando se sigue la traza de las transiciones de estado que están se recomienda seguir los siguientes criterios:

- Clasificar los atributos dinámicos en alguna de estas tres categorías.
- Si el atributo es clasificado como a, identificar los eventos que lo afectan teniendo en cuenta cuáles aumenta su valor, cuáles lo decrementan y cuáles lo reinicializan, agrupar todos los eventos que tengan el mismo efecto y el mismo tipo de acción (incrementadora, decrementadora y reinicializadora), y definir un estado que refleje esta situación. Si hay más de un evento en está unión, se relacionan usando el operador OR.
- Si el atributo se clasifica como b, identificar los eventos y el efecto que provocan, agrupar usando el operador OR los eventos que provocan una misma forma de obtener el nuevo valor, y definir un estado para ellos.
- Si el atributo es clasificado como c, identificar todos los eventos, el nuevo valor que provocan y el valor del atributo para el cual ese evento tiene sentido. Se definen tantos estados, como posibles valores existan.

Tomando como referencia las clases y relaciones que aparecen en la figura 2, solo podríamos considerar como dinámica la relación que se establece entre el Libro y la Edición porque con el transcurso del tiempo se publican nuevas ediciones. Fíjese que el evento que genera la transición es la publicación de una nueva edición del libro, que para el sistema genera la necesidad de registrar las características de esa nueva edición, manteniendo a es objeto Libro en el mismo estado. En este caso el Diagrama de Estado resultante se representa en la figura 3.

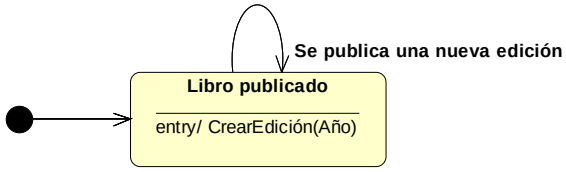

Fig. 3. Ejemplo de diagrama de transición de estado

5. Obtener las restricciones estáticas y las fórmulas dinámicas. El DC incluye la definición de restricciones de integridad estática a través de las cardinalidades de las relaciones, la dimensión E/D y el tipo de datos asociado a cada atributo.

Para el resto de estas restricciones, el método DIBAO sugiere que se especifiquen textualmente completando para cada atributo de cada clase: si toma valor único, si puede ser nulo, rango (para el caso de numérico y enumerativo), valor por defecto y fórmula de derivación (para los atributos derivados).

Estas restricciones a columnas se corresponden con diferentes categorías que se referencian en la literatura. Existen otras condiciones que deben satisfacerse y que no es posible categorizarlas, por lo que se pueden definir como restricciones. Para poder determinar la necesidad de estas restricciones al dominio, es necesario especificar entonces, para todos aquellos atributos que lo requieran, reglas que indiquen qué valores pueden tomar, o lo que es igual, qué condiciones deben cumplir los atributos. El formato general que se propone en este trabajo para expresar estas restricciones es: <atributo> = <condición>. En la condición hay que indicar las tablas involucradas y se puede usar NOT, EXIST, funciones de agregación, AND, OR, IN, BETWEEN.

Las restricciones se validan cada vez que se intente cambiar el valor de los atributos a los que se asocia cada una. Su implementación dependen de los predicados que brinde el lenguaje de definición de datos del gestor seleccionado.

Estas restricciones estáticas se pueden definir para cualquier atributo independientemente de cómo se clasifiquen. Del DE se pueden obtener varias de las fórmulas dinámicas. En función del diseño de la BD, a continuación se describe como se propone derivarlas del diagrama, aunque hay algunas características dinámicas que como se verá requieren de nuevas formas de especificación.

Precondiciones.

Del DE se pueden obtener directamente las precondiciones que deben cumplirse para que ocurra un cambio de estado. Estas precondiciones quedan reflejadas, en la descripción de las transiciones, como la unión del evento que ocurre y las condiciones que deben evaluarse para provocar el cambio. Las restricciones que tienen los estados se deben cumplir para que el objeto se mantenga en ese estado, por lo tanto lo contrario de una restricción tiene que ser una condición de alguna de las transiciones de salida que se deriven de ese estado.

En el DE solo se representa de un estado su nombre, siendo difícil al observarlo poder determinar cuáles atributos han sido afectados. En la descripción de las actividades, que se ejecutan cuando se está en un estado, están definidos los pasos que se efectúan para dar un nuevo valor a el o los atributos dinámicos que se afectan y también se pueden incluir las acciones que se están haciendo mientras el objeto está en ese estado. En ésta descripción se puede identificar el o los atributos que se modifican pues, directamente o como parte de un método de la clase, se utiliza una asignación en la que en la parte izquierda está un atributo de la clase, y en la derecha el nuevo valor.

Las fórmulas de evaluación describen los posibles valores que puede tomar un atributo dinámico, a partir de un estado dado, y cómo se determina ese nuevo valor. Su especificación depende de la clasificación de los atributos dinámicos. Una vez procesados todos los atributos dinámicos de todas las clases, se tienen todas las fórmulas de evaluación.

Disparadores.

Una forma de mantener la integridad de los datos es definir disparadores [9]. Un disparador es una sentencia que el sistema ejecuta automáticamente como efecto secundario de una modificación de la BD [17]. Siguiendo las reglas propuestas en [6], solo se debe usar disparadores para garantizar la integridad y la lógica de control y no como herramienta para la validación de los datos porque implican operaciones internas costosas. En tiempo y además reducen la portabilidad de la BD ya que cada gestor tiene su forma particular de implementarlos.

Es lógico que nos sea difícil obtener disparadores del DE pues no siempre un cambio de estado tiene como antecedente o evento que lo provoque, un valor específico de uno o varios atributos. El DE no es la fuente natural. Un mecanismo de disparo entra a funcionar cuando en una tabla se adicionan o eliminan elementos o cuando un atributo dado toma un valor determinado.

El método DIBAO propone que la especificación de los mecanismos de disparo debe seguir el siguiente formato: <causa que lo provoca><tabla>: condición]/<acción>{ANTES/DESPUES}. Donde:

Causa: INSERT-adición DELETE-borrado UPDATE-modificación.
Tabla: Nombre de la tabla o Nombre de la tabla, atributo y valor
Acción: Acciones que se ejecutan como efecto del disparo.
Antes/Después: Momento en que se producen las acciones con relación al cambio que provoca el disparo.

Algunas transiciones tienen asociadas acciones. Hay ocasiones en que estas acciones podrían implementarse como las acciones definidas en el disparador. En estos casos la causa sería la modificación (UPDATE) de el o los atributos que se afectan cuando se pasa al nuevo estado. La propiedad antes/después tomaría el valor de antes. Esto es válido cuando ocurre algo de lo siguiente:

- Solo hay un estado y una transición que llega a ese estado, para el o los atributos afectados.
- Solo hay un estado relacionado con el cambio de el o los atributos y todas las transiciones que llegan a él tienen asociadas las mismas acciones.
- Hay varios estados relacionados con el cambio de el o los atributos y todas las transiciones que llegan a ellos tienen asociadas las mismas acciones.

Derivación.

Las fórmulas de derivación de los atributos derivados, se definen textualmente utilizando los operadores matemáticos necesarios, valores constantes (por ejemplo, si es el doble de valor de un atributo sería dos * <valor del atributo>) y atributos (pueden ser de la clase a la que pertenece el atributo derivado o de otra clase, en cuyo caso hay que especificar de cuál <nombre de la clase>.<nombre del atributo>).

Especificación de métodos.

Como se ha planteado anteriormente se propone utilizar en la especificación de acciones, actividades y métodos de las clases, el lenguaje de especificación definido en UML. Además se pueden utilizar otras, construcciones como las decisiones, selección, repetición o precondiciones/postcondiciones. En la descripción se pueden usar las funciones de

agregación: contar, sum (sumar), avg (promedio), máx (máximo) y mín (mínimo), indicando sobre qué elementos se hacen efectivas.

6. Convertir las clases al medio de almacenamiento.

La propuesta de diseño de base de datos que se incluye se divide en tres partes, que dependen del sistema que se utilice para implementar la solución al problema planteado. Estas tres soluciones son:

- Si se tiene un SGBDO, hay que garantizar que las definiciones obtenidas se correspondan con el modelo ODMG.
- Si lo que se tiene es un LPOO que permite relacionarse con una base de datos relacional o un SGBDOR, la solución es llevar las clases definidas a tablas.
- Si lo que se tiene es un LPOO que no permite relacionarse con base de datos relacional, pero que permite salvar registros, entonces se deben utilizar las clases que brinda para almacenar los atributos de las clases persistentes en ficheros estructurados. Puede que el lenguaje permita la relación con BDR, pero se escoge como forma de almacenamiento la organización en ficheros.

El factor decisivo en la selección depende de las aplicaciones y los datos que maneja.

Implementación de la persistencia para el almacenamiento en ficheros.

Cuando se utiliza esta forma de almacenamiento se debe garantizar para las clases persistentes que se añadan métodos para salvar y recuperar información., se defina el código de estos métodos, de acuerdo al tipo de los atributos; y se defina el código para registrar las clases. Si se desea que sea persistente el comportamiento de los objetos, se puede almacenar su implementación y la definición de la clase en cualquier medio (por ejemplo, tratando la información como texto), pero es responsabilidad del diseñador definir todo lo necesario para recuperar y salvar esta información. La posibilidad de almacenar la estructura completa de una clase está sustentada en el concepto de metaclase. Esto puede hacerse siempre y cuando el lenguaje de programación lo permita.

Modelo de objeto de ODMG.

El modelo de objetos es la base del estándar ODMG [5]. En el método DIBAO las definiciones de clases, con sus comportamientos estático y dinámico, incluyen totalmente los conceptos de este modelo y van más allá. Esto último sugiere que con el gestor y lenguaje de programación con que se trabaje tendrán que implementarse algunas características (por ejemplo, las fórmulas de evaluación, las restricciones asociadas a las relaciones entre clases).

Conversión de clases a tablas.

Las llaves en el MR están asociadas a atributos de una tupla de una relación que caracteriza unívocamente a esa tupla. En ocasiones es difícil determinar un conjunto mínimo y suficiente de atributos para conformar un identificador. Por otra parte, no hay nada que impida a una aplicación modificar los valores de una columna identificada como llave, aunque hay SGBD que apoyan a esta restricción de integridad [15]. Por esto se recomienda asociar a las clases simples y complejas un atributo que constituya una llave de autoincremento (por ejemplo, SQL Server permite este tipo de atributos) , sin significado semántico en la aplicación. Esto no excluye la posibilidad de definir las llaves tradicionales, en cuyo caso en el DC hay que identificar los atributos llaves indicando si es <<llave primaria>> o <<llave extranjera>>. La llave de las clases compuestas, por defecto, es la suma de las llaves de las clases que la integran. Las reglas de conversión de clases a tablas que se proponen son:

1. Cada clase con dominios simples se convierte en una tabla.
2. En el caso de la jerarquía de clase, se pueden aplicar tres posibilidades. Estas posibilidades son:

    a. Definir una única tabla con los atributos del padre y los atributos de cada hija.
    b. Definir una tabla para cada clase hija hoja que incluya los atributos de la clase padre.
    c. Definir una tabla para cada clase hija y para cada clase padre. Para recuperar la información de una clase hay que consultar la clase hija y sus ancestros.

Este método recomienda la última variante porque está más preparada para la evolución del esquema ya que en las variantes a y b las tablas creadas se modifican si ocurre algún cambio al no implementar el concepto de herencia. Además las tablas tendrían atributos que tienen sentido solo para algunos hijos, lo que implica permitir el valor nulo y por lo tanto los programadores tendrían que definir los procedimientos para validar estas restricciones de dominio. Las propuestas a y b son superiores en la rapidez en la recuperación de la información.

3. Para clases complejas la conversión depende de las cardinalidades máximas en cada extremo. Las cardinalidades mínimas introducen restricciones de integridad estática. Las reglas que se aplican en estos casos son:

m:n	Se convierte en una tabla que tiene como la llave las llaves de las clases que conforman la relación. Otra variante es transformarla en dos relaciones bidireccionales en cuyo caso se añade un atributo de referencia a cada tabla. Esta variante hace más complejas las búsquedas y actualizaciones.
1:m ó m:1	Se puede convertir en una nueva tabla o puede ser añadida una llave extranjera al extremo m, que se corresponde con la llave de la clase del extremo 1.
1:1	Se puede convertir en una nueva relación o se adiciona un nuevo campo a alguna de las tablas que es la llave de la otra.
c:m ó m:c	En el caso de que en uno de los extremos exista en una cantidad mayor que 1 de elementos con los que se relaciona la clase del extremo m, se puede definir una nueva tabla que tenga como atributos llave, la llave del extremo m y tantos atributos del tipo de la llave del extremo constante como indique le valor de la constante. También se pueden adicionar a la clase del extremo m, tantos campos del tipo de la llave del extremo constante como indique el valor de la constante.
c:c1	Se puede definir una nueva tabla que tendrá c+c1 llaves, en la que habrá c campos del tipo de la llave del primer extremo, y c1 campos del tipo de la llave del extremo. También se puede adicionar a la clase de uno de los extremos, tantos campos del tipo de la llave del otro extremo como indique el valor de la constante.

En las relaciones de 1:1 y 1:m, si no hay ciclos, se puede tener en una sola tabla los objetos que participan en la relación, pero esto viola la segunda forma normal porque introduce redundancia.

A pesar de que no es necesario crear nuevas tablas que reflejen la relación en casi ningún caso, es recomendable hacerlo porque el esquema puede evolucionar y no sería en estos casos necesario hacer cambios a la estructura de la BD, solo a las restricciones de integridad estática asociadas a la cardinalidad. Además hay menos complejidad en las búsquedas y en la actualización de la información, y se conserva la propiedad de encapsulamiento, clave en el enfoque orientado a objetos, ya que un objeto no debe conocer cosas de otro objeto si no es necesario.

4. Para las clases compuestas Se define una tabla para cada clase y para el caso de las relaciones de 1:1 y de 1:m, se pone en el extremo de la parte un atributo que es la llave del todo como llave extranjera. Si la relación es una agregación, se etiqueta como Non_Identifying y en el caso de la composición, como Identifying. Para el caso de las relaciones de m:n entre el todo y las partes, se tranforma igual que las asociaciones de m:n, independientemente de si es una agregación o composición y siempre se etiquetan como Identifying.

   Las relaciones, usando Rational Rose, se estereotipan en: Non_Identifying que representa una relación entre dos tablas independientes e Identifying que representan una relación entre dos tablas dependientes, donde la tabla hija no puede existir sin la tabla padre.

   En el caso de que la implementación sea hacía un SGBDOR, que permita tener campos que referencien a otro objeto, no es necesario que se cambie el tipo de éste campo para el tipo de la llave de la clase a la que pertenece el objeto referenciado. La implementación de las características asociadas al comportamiento estático dependen de las potencialidades del gestor que se utilice. Por ejemplo, **SQL Server** incluye el predicado **check** que permite especificar restricciones de dominio de los atributos [9].

En el caso del comportamiento dinámico se pueden implementar como procedimientos almacenados las validaciones asociadas a las fórmulas de evaluación, las acciones que se ejecutan como parte de los mecanismos de disparo y las actividades asociadas a los estados.

En la figura 4 se muestra el modelo de datos que resulta de transformar el DC de la figura 2 a partir del supuesto que se usará como gestor de BD a Oracle 9i. En este caso se generó una tabla por cada una de las tablas y la herramienta CASE utilizada (Rational Rose), al encontrar una relación de m:n, generó una nueva tabla tal como se indica en la regla 3. Las rectricciones que se incluyen en cada tabla son las que genera por defecto asociadas a las llaves.

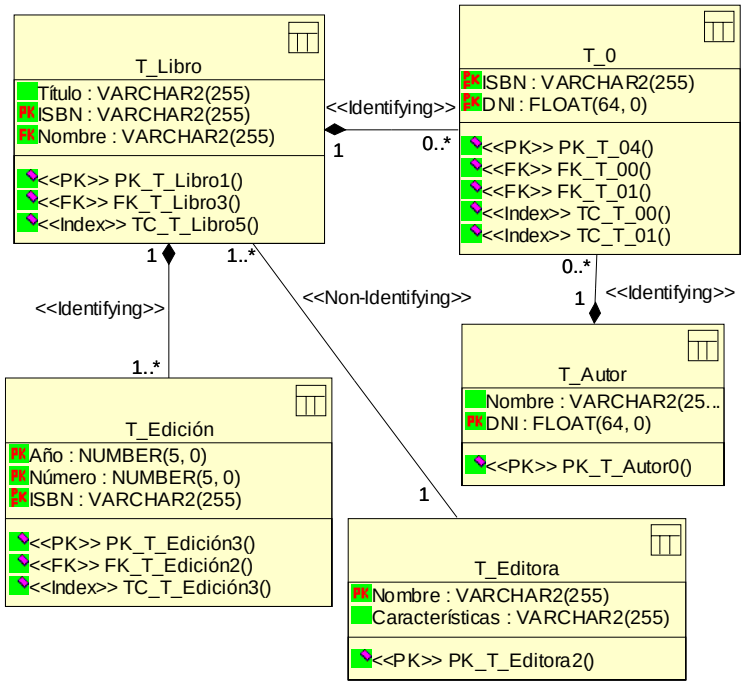

Fig. 4. Modelo de datos

3. Capa Persistente

Cuando se trabaja con un gestor de objetos, internamente se implementan los conceptos del enfoque OO y las capacidades de las BD, por lo que conceptos como los relativos a la herencia, por ejemplo, son tratados automáticamente sin que intervengan los programadores. En las variantes de utilizar como medio de almacenamiento los ficheros o un LPOO que permita relacionarse con gestores relacionales u objeto/relacionales, será responsabilidad de diseñadores y programadores definir las clases que permitan manejar los conceptos de objeto y otros relacionados con las relaciones entre clases, que no están incluidos dentro de los modelos relacional y objeto/relacional.

La posibilidad de especificar clases permite la definición de una capa de clases persistente que encapsule el trabajo con los datos almacenados. Tradicionalmente esta capa solo contiene las definiciones necesarias para implementar la interfaz entre el medio de almacenamiento y las clases del dominio [11][18]. El método DIBAO propone incluir una subcapa de especificación que describa las características de las relaciones entre los objetos en correspondencia con lo definido en el DC. Además modifica la estructura de las clases persistentes adicionando métodos que recogen las restricciones de integridad y las fórmulas dinámicas antes descritas (figura 5).

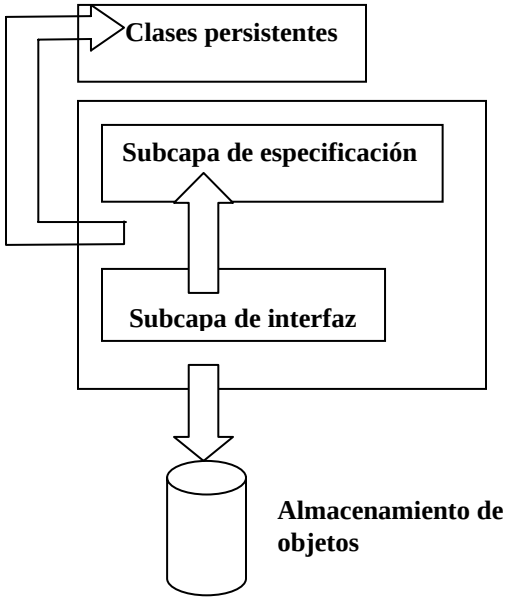

Fig. 5. Estructura de la capa de clases

Modificación de la estructura de las clases persistentes.

Asociadas a las restricciones al dominio de los atributos, la forma en que toman valor los atributos dinámicos y las fórmulas dinámicas, obtenidas al aplicar el modelo de persistencia del método DIBAO; es necesario añadir a las clases persistentes lo siguiente:

- Para cada atributo, independientemente de su clasificación, un método que verifique sus restricciones de dominio
- Un método para cada uno de los algoritmos que le asignan un valor en el caso de que sea un atributo dinámico.
- Un método que implemente la fórmula de derivación para los atributos derivados.
- Para el caso de los disparadores se definen nuevos métodos (invocados por los métodos de creación, destrucción o actualización de un objeto en dependencia de lo que indique, como evento que lo provoca, la especificación del mecanismo de disparo), que verificarán la condición, en caso de existir, e implementarán las acciones.

A través de la herencia redefiniendo métodos o incluyendo nuevos métodos o clases según corresponda, se pueden implementar cambios en el esquema.

Subcapa de especificación.

En la fase de diseño de la BD, el método DIBAO ha adicionado características a las clases y sus relaciones que, aunque están relacionadas con el comportamiento de los objetos en el mundo real (por ejemplo, la dimensión entre compuesta y componente), no son propiedades semánticas como el color de los ojos o del cabello. La definición de nuevos atributos tiene aparejada la creación de métodos que los actualicen y garanticen las restricciones a ellos asociadas.

Como los nuevos atributos y métodos que pudieran aparecer son características del objeto que deben persistir junto al resto de las propiedades, se recomienda en esta subcapa, extender las estructura de las clases a través de la herencia, de forma tal que se incluya:

- Definir la clase THerencia con la siguiente especificación:
  - Herencia de la clase que el lenguaje defina como persistente.

- Atributos:

Atributo	Tipo	Descripción
Total	Lógico	Falso- indica que al menos un objeto del tipo de la clase padre no se corresponden con alguno de sus hijos.
Solapamiento	Lógico	Falso- indica que un objeto del tipo de la clase padre solo se corresponde con un objeto en alguno de sus hijos.
Hija	Lógico	Verdadero- indica que tiene ancestros en la jerarquía del dominio.
Atributo de especialización	Clase	Clase que contiene el atributo que provoca la especialización y el valor que toma para esta hija en particular.
Árbol de herencia	Clase base del árbol de jerarquía del lenguaje	Colección de clases hijas en el árbol de jerarquía.

  - Redefinir los métodos de salvar y restaurar de acuerdo a sus nuevos atributos.
  - Crear los métodos que permitan crear y destruir una instancia de este tipo y le dan valor y consultan a sus atributos.

- Para todas las clases o un ancestro en su árbol de jerarquía, heredar de la clase THerencia (definida por el programador en el lenguaje que utilice). Esta clase recoge las características de una relación padre/hija lo que convertiría a todas las clases por defecto en posibles padres.
- Definir una nueva clase para cada una de las clases que tengan atributos que referencien a otros objetos. En todos los casos por cada clase referenciada se definen cuatro atributos que indican las cardinalidades máxima y mínima en cada extremo. En el caso de las clases compuestas hay que adicionar por cada componente un atributo de tipo lógico que indique la dimensión (Verdadero - Estática, Falso – Dinámica). Esto implica que hay que definir métodos que verifiquen que se cumplan las restricciones asociadas a los valores de estos atributos.

Subcapa de interfaz.

Ambler Scott en [1] describe diferentes formas de implementar una capa de clases persistente para el trabajo con la información almacenada en una BDR. En este trabajo se utilizan dos de ellas. En la variante de implementación hacía ficheros, se incluye en la especificación de las clases los métodos para salvar y recuperar datos. En este caso la capa de clases está embebida dentro de las clases del dominio lo que permite escribir código rápidamente (útil para aplicaciones pequeñas y prototipos), pero que implica un fuerte acoplamiento entre las clases del dominio y el medio de almacenamiento, por lo que un cambio afecta a los métodos. Pero, la forma en que los lenguajes de programación implementan esta forma de almacenamiento, obliga a utilizar esta variante a pesar de sus problemas.

Cuando se trabaja con un LPOO, pero la información se almacena en un BDR o BDOR, es posible encapsular el trabajo con el medio de almacenamiento en una estructura intermedia que sea la única afectada si cambia el gestor o la estructura de las tablas en que se almacena la información de un objeto. Evidentemente esta solución hace más compleja la construcción de aplicaciones, pero crea una disciplina que mejora la calidad del producto final, facilita el mantenimiento y permite el desarrollo de aplicaciones más complejas. La idea consiste en:

- Definir la clase TPersistenciaBD que incluya:
  - Atributos:

Atributo	Tipo	Descripción
TablaBase	Clase base del árbol de jerarquía del lenguaje	Nombre de la tabla a la que directamente se transforma la clase
Lista de tablas asociadas	Clase base del árbol de jerarquía del lenguaje	Listado que contiene las tablas asociadas a la clase base y fue generado a partir de las conversiones de clases a tablas.

- Métodos para salvar, borrar, recuperar y modificar objetos, que se redefinen en cada clase hija de acuerdo a las tablas asociadas a cada clase. Su implementación mezcla sentencias del lenguaje de programación y del lenguaje de consulta. Estos métodos son los únicos que se afectarían por un cambio en el esquema de la BD o el gestor utilizado.
- Incluir en las clases persistentes una referencia a la clase de la capa persistente porque los métodos que necesitan de los valores de los atributos del objeto, solo tendrán acceso a ellos a través de esta capa de clases.
- Definir una nueva clase para cada clase persistente que herede de TPersistenciaBD.

4. Experiencias en la Utilización del Método

La propuesta de método de diseño de la BD que se describe en este artículo se incluye dentro de los cursos de Ingeniería de Software que reciben los estudiantes de 4to año de la carrera de Ingeniería en Informática en Cuba, desde el año 2000. Además, desde ese mismo período forma parte de los cursos análisis y diseño que se imparten a empresas del país.

La versión actual del método se ha aplicado a proyectos que manipulan grandes volúmenes de información por lo que trabajan con gestores relacionales y objeto/relacionales (fundamentalmente SQL, Oracle y Paradox). Para otras aplicaciones de propósito general se ha utilizado la persistencia hacia ficheros. Los proyectos desarrollados han estado vinculados principalmente con la gestión empresarial, desarrollos en la medicina y otros trabajos de ingeniería.

Actualmente también se incluye este método de diseño dentro del curso de análisis y diseño que reciben los estudiantes de la carrera de Computación y Sistemas de la Universidad Peruana de Ciencias Aplicadas (Lima, Perú) y en la carrera de Sistemas Automatizados de Dirección del Instituto Técnico Militar José Martí (Cuba).

5. Conclusiones

Estática y dinámica son dos perspectivas importantes que se han tener en cuenta cuando se diseña la BD. Por un lado se busca definir estructura y por el otro cambios permitidos sobre esa estructura. Obviar alguna de estas vistas en el proceso de diseño nos puede llevar a una BD inconsistente.

En ésta propuesta de método de diseño de la BD se incluyeron los conceptos de las definiciones estática y dinámica referidos por [8][21]. Se usó como referencia a estos autores porque recogen en sus criterios tópicos abordados por otros también de reconocimiento en éste campo.

Partir, como premisa para el diseño de la BD, de la clasificación de sus atributos, permite minimizar éste proceso al centrarse el diseñador en lo que puede afectar al atributo y lo que debe saber de cada uno.

El método de diseño sistematiza los pasos a seguir en el diseño así como recomendaciones para llevarlos a cabo, teniendo en cuenta todas las variantes, en cuanto a forma de almacenar la información, con que se pueden enfrentar los diseñadores.

Además, describe cómo reflejar las características estáticas y dinámicas utilizando los DC, DE y formatos de especificación de algunas restricciones de integridad y fórmulas dinámicas.

La conversión de clases a tablas no desecha por completo todas las características del enfoque de objetos ya que:

- Encapsular el trabajo con las tablas con la creación de clases persistentes que actúan de interfaz en su relación con otras clases que requieren de la información almacenada.
- Recomienda la creación de nuevas clases con vistas a futuros cambios en el esquema.
- Representa de la agregación como una clase para conservar el objeto del mundo real con su significado.
- Recomienda siempre como tendencia la implementación del código utilizando las potencialidades del enfoque OO.

El modelo de objetos que se obtiene como resultado, cumple las características definidas por ODMG. La capa de persistencia incluye dos subcapas que permiten la definición de nuevas clases que responden al modelo de persistencia y asilan a las clases del dominio del medio de almacenamiento. En el caso de las clases persistentes, se propone adicionar nuevos procedimientos asociados a las validaciones de las restricciones estáticas y fórmulas dinámicas.

Referencias

1. **Ambler, S.** "Design a robust persistence layer" (Part 4 of 4). Software Development. 6(4) 73-75, April 1998.
2. **Ambler, S.** "Persistence layer requirement". Software develpoment. 6(1) 70-71, January 1998.
3. **Ambler, S.** "Persistence modeling in the UML". Software Development Magazine. http://www.sdma_gazine.com/articles/1999/0008a.htm. August 1999.
4. **Ambler, S.** "The Design of Robust Persistence Layer for Relational Database". http://www.amysoft.com/_persistencelayer.pdf. October 21, 2000.
5. **Barry, D.** "ODMG 2.0: An overview". Dr. Dobb's Sourcebook. http://www.com/aerticles/1997/9717a/9717a.html. September/October. 1997.
6. **Bussert, J.** "The rule of triggers". MIDRAGE Systems. 11(8) 50, June 15, 1998.
7. **Coleman, D.; Hayes, F. and Bear, S.** "Introducing ObjectCarts or How to use statecharts in object-oriented Design". IEEE transactions on Software Engineering. 8(1). January 1992.
8. **Cooling, J.E.** "Real-time software systems: An introduction to structure and object-oriented design". International Jhonson Publishing Inc. EUA. 1997.
9. **Dalton, P.** "Microsoft SQL Server black book". The Corilis group, Inc. International Thomson Publishing Company. 1997.
10. **Delgado, M.** "Automatización del Diagrama de transición de estado para la metodología ADOOSI". Tesis para optar por el título de maestra en Informática. ISPJAE. Cuba. Noviembre 1997.
11. **Embley, D**. "Object database Development: concepts and principles". Addison Wesley Longman, Inc. EUA.1998.
12. **Hernández, A.** Tesis presentada en opción al grado científico de Doctor en Ciencias Técnicas: "DIBAO, un método para el diseño de la Base de datos a partir del modelo orientado a objetos". ISPJAE. Cuba. Diciembre 2001.
13. **Hofstede, ter A.H.M. and H.A. Proper**. "How to Formalize It?. Formalization Principles for Information Systems Development Methods". Technical report #4/97, Faculty of Information Technology, Queensland University of Technology, Brisbane, Australia. http://www.icisqut.edu.au/~authur/ Work.ps.Z. June 1997.
14. **Hubbers. W.G.M. and A.H.M. ter Hofstede.** Exploring the Jungle of Object-Oriented Conceptual Data Modeling. In Chris McDonald, editor, Proceedings of the 9th Australasian Database Conference, ADC'98, Springer, .Perth, Australia. Australian Computer Science Communications. 20(2) 65-76,. February 1998.
15. **Jordan, D.** "Identifying ODMG objects". SIGS Publications, USA. http://www.borland.be/borlandcpp/news/report/CR9503jordan.html. 1996.
16. **Keller, W.; Mitterbauer, C. and Wagner, K.** "Object-oriented data integration". In Chaudhri,A. and Loomis, M. "Object Database in practice". Prentice-Hall, Inc. Hewlett-Packard Company. 3-20, 1998.
17. **Korth, H. And Siberschatz, A**. "Fundamentos de Bases de Datos". 2da edición. McGraw-Hill/Iberoamericana de España, S.A. 1993.
18. **Liberty, J.** "Begining Object-Oriented Analysis and Design with C++". Wrox Pres., Canadá.1998.
19. **Martin, J. and Odell, J.** "Object-oriented methods: a foundation". Second edition. Prentice-Hall, Inc. EUA.1998.
20. **Pastor, O.** "Diseño y desarrollo de un entorno de ejecución automática de software en el modelo orientado a objetos". Tesis doctoral. DSIC-UPV, España. Abril 1992.
21. **Pastor, O. y García, R**. "Uso de Oracle como base de datos relacional para implementar un diseño orientado a objetos". Generalitat Valenciana, Conselerria de Senitat i Consum. 3 de Mayo de 1993.
22. **Pastor, O.; Pelechano, V.; Bonet, B. y Ramos, I**. "OO-Method 2.0: una metodología de análisis y diseño orientado a objetos". Reporte de investigación DSIC,UPV. España. 1996.
23. **Rumbaugh, J.; Blaha, M.; Premarlini, W.; Eddy, F. y Lorense, W.** "Modelado y diseño orientado a objetos. Metodología OMT." Prentice-Hall Hispanoamericana. S.A. 1996.
24. **Rumbaugh, J.; Jacobson, I. and Booch, G.** "The unified modeling language: reference manual". Adison-Wesley Longman, Inc. Canadá.1999.
25. **Wayna, M.; Christiansen, J.; Hield, Ch. and Simunick, K.** "Modeling battefield: sensor environment". In Chaudhri,A. and Loomis, M. "Object Database in practice". Prentice-Hall, Inc. Hewlett-Packard Company. 210-215, 1998.

Anaisa Hernández González. *Graduada de Ingeniería en Sistemas Automatizados de Dirección en 1993 y Master en Informática Aplicada a la Ingeniería y la Arquitectura en 1996, ambos títulos otorgados por el Instituto Superior Politécnico José Antonio Echeverría (ISPJAE), en Cuba. En el 2002 recibió el título de Doctor en Ciencias Técnicas otorgado por la Academia de Ciencias de Cuba. Ha publicado numerosos artículos y participado en eventos internacionales. Es profesora desde el año 1993 del ISPJAE y actualmente dirige un proyecto de formación de profesores de Ciencias Informáticas. Sus áreas de investigación están vinculadas con la Ingeniería y Gestión de software, el Diseño de la Base de Datos y la Dirección de proyectos informáticos, obteniendo sus principales resultados en el desarrollo de metodologías, métodos, procedimientos y herramientas para el análisis y diseño de sistemas.*