HACIA LA FORMALIZACION DEL DISEÑO DE BASES DE DATOS ORIENTADAS A OBJETOS*

VICTOR HUGO MEDINA GARCIA**

RESUMEN

El presente estudio tiene como bases: la Ingeniería de Software, las metodologías estructuradas de desarrollo de software, las Bases de Datos como componentes esenciales e inseparables de la gran mayoría de Sistemas de Información, el concepto de herramientas CASE (Ingeniería de Software Asistida por Computador), y la teoría desarrollada alrededor de la orientación a objetos, que ha originado el surgimiento de una gran cantidad de paradigmas, en los que ya se vislumbran soluciones. El caracter de este estudio es investigativo y exploratorio dentro del área de la formalización del Diseño de Bases de Datos Orientadas a Objetos.

La importancia de la investigación se centra en presentar una aproximación al diseño y desarrollo de Bases de Datos Orientadas a Objetos (BDOO), cuya utilización y discusión permiten el refinamiento y formalización de una metodología, que pueda ser implementada por medios automatizados. No se pretende obtener un producto final completo ni óptimo; el interés del autor es plantear un primer enfoque al desarrollo de BDOO y el planteamiento de una serie de técnicas orientadas a objetos que permita apoyarla.

* Ingeniero de Sistemas - Universidad Distrital
Especialización en Marketing - Universidad del Rosario
Master Informática - Universidad Politecnica de Madrid
Estudios de Doctorado en Lenguajes, Sistemas Informáticos e Ingenieria de Software - Universidad Politécnica de Madrid
Coordinador de la Especialización en Gerencia Informática de la Escuela de Administración de Negocios

1. GENERALIDADES

1.1. CONCEPTO DE UNA BDOO

El desarrollo de software orientado a objetos, es una de las tecnologías más recientes y de mayores perspectivas en el área de la Ingeniería de Software. Su aplicación en la implementación de nuevos sistemas informáticos, ha creado grandes expectativas, por la supuesta simplicidad y facilidad para producir diseños con ventajas notables que garantizan el mantenimiento efectivo del software.

Las bases de datos orientadas a objetos se diferencian de las bases de datos clásicas, porque la *unidad de información es el objeto*. Además se le exigen ciertas propiedades como el encapsulamiento, la herencia, el soporte de objetos complejos (hacen referencia a otros objetos) y la posibilidad de incluir datos de tipos complejos, no previstos inicialmente por la base de datos, pero requeridos por alguna aplicación concreta (extensibilidad). A veces (no siempre) se exige que la BD sea capaz de incluir métodos (es decir código ejecutable) como parte de la información asociada a un objeto determinado. Otras admiten que los métodos sean almacenados por separado, en cuyo caso la BD sólo contendría atributos de los objetos e información sobre su organización jerárquica.

Además de poder esquematizar este concepto (Fig. 1.1) y de acuerdo a estas consideraciones, se puede deducir que una BDOO es equivalente a:

BDOO =*Orientación a Objetos + Capacidades de las BD*

En donde

Orientación a Objetos = Tipos abstractos de datos + Herencia
+ Identidad Objeto

Capacidades de las BD = Persistencia + Concurrencia + Transacciones
+ Recuperación + Interrogación + Integridad
+ Control versiones + Seguridad + Rendimiento

1.2. REGLAS DE LAS BDOO

Aunque se han realizado grandes esfuerzos para construir y comercializar sistemas de BDOO, no existe ningún estándar. No se dispone de un lenguaje estándar para la BDOO con el que programar aplicaciones. No hay un modelo estándar de datos sobre el cual fundamentar dicho lenguaje. Sin embargo el campo de las áreas de BDOO ha madurado lo suficiente como para justificar la propuesta de un conjunto de reglas, dadas por el *Comité* para el avance de las funciones de SGBD, a través del llamado *Manifiesto de las Bases de Datos Orientadas a Objetos*. Estas características, son como una equivalencia a las reglas de Codd en las BD relacionales.

Según apuntes del manifiesto, las BDOO deben satisfacer tres exigencias básicas:

1) Además de todas las capacidades y servicios de los gestores de BD convencionales, las BDOO *deben soportar estructuras de objetos y reglas*. Es decir que se requiere almacenar y manipular elementos de datos no tradiciona-

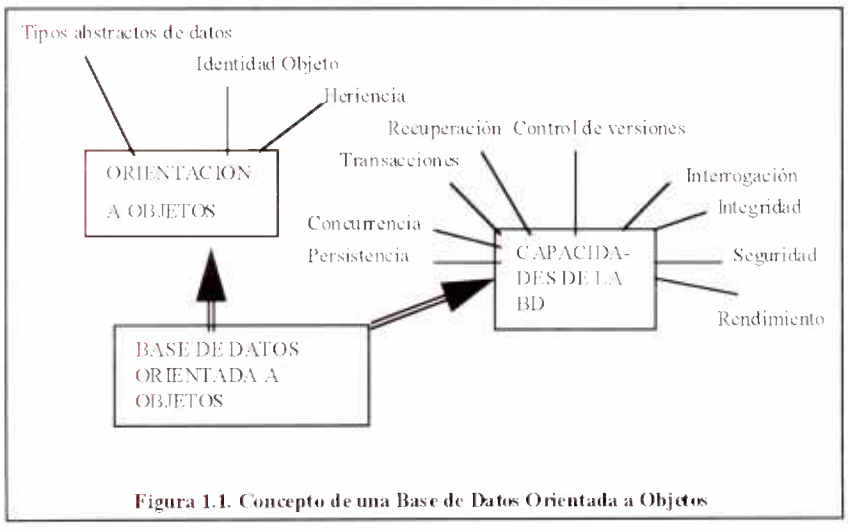

Figura 1.1. Concepto de una Base de Datos Orientada a Objetos

les tales como textos, imágenes, documentos multimedia, vídeo y otros; y además debe poder especificar reglas que solucionen con éxito cada problema.

2) *Deben conservar las facilidades de las BD relacionales*. Y aunque estas tienen como principal característica el acceso de datos no procedural y la independencia de datos, no necesariamente se debe comprometer a las BDOO a que cumplan estas ventajas. El manifiesto afirma que las BDOO que utilicen el lenguaje C++ tendrán que adoptar el SQL y reducir la capacidad de ejecución a niveles relacionales.

3) Deben estar abiertas a otros subsistemas. Es decir que deben permitir accesos desde otras herramientas en ejecución, en diversos entornos, y ser compatibles con bases de datos distribuidas.

1.3. CONSIDERACIONES DE ESTRATEGIA

Para el desarrollo de un SGBDOO (Sistema de Gestión de Bases de Datos Orientadas a Objetos) hay una serie de aproximaciones que intentan incorporar las capacidades de la orientación a objetos a las BD[1] , estas son:

1) *Desarrollo de un nuevo modelo y lenguaje de datos para la Base de Datos OO*: La aproximación más agresiva consiste en desarrollar completamente un lenguaje de BD nuevo y un SGBD nuevo, incorporando las capacidades de la orientación a objetos. La mayoría de los prototipos de investigación de BDOO han seguido este camino. Como ejemplo tenemos el sistema comercial SIM, que introduce un nuevo lenguaje de manipulación de datos y un nuevo lenguaje de definición de datos basándose en el modelo semántico de datos.

2) *Ampliar un lenguaje de BD existente con las capacidades OO*: Existen lenguajes de programación que han sido ampliados con construcciones para la administración de objetos, como por ejemplo C++, Pascal Objeto y el Flavors (ampliación de Lisp). Los lenguajes de BD están tomando el mismo camino. Desde que el SQL es un estándar, la solución más razonable es ampliarlo con construcciones orientadas a objetos para que el SGBD suministre las capacidades requeridas. Esta aproximación esta siendo seguida por la mayoría de las firmas de sistemas Relacionales, entre los cuales podemos destacar los SGBDOO IRIS y ONTOS, que tienen sus propios dialectos de SQL con orientación a objetos.

Según fuentes de la red Internet se esta intensificando el desarrollo del SQL3 que soporta objetos y estará disponible probablemente en 1998. Muchas características del SQL2 (versión revisada y actualizada en 1992 del SQL estándar) ya han sido implementadas.

3) *Ampliar el lenguaje de OO con capacidades de la BD*: Consiste en introducir las capacidades de las BD a un lenguaje de programación orientado a objetos. En esta ampliación se incorporan capacidades de BD como transacciones, persistencia, control de versiones, etc. El sistema GEMSTONE, ha seguido esta aproximación, ya que básicamente resulta ser una ampliación del lenguaje SMALLTALK al que se le han añadido clases y primitivas para la administración de BD; esta ampliación ha constituido el lenguaje OPAL.

4) *Suministrar librerías orientadas a objetos al SGBD*. Mientras que Servo Logic introdujo nuevas construcciones a un lenguaje OO existente, ONTOS introdujo una librería C++ para la administración de la BD. Estas librerías incluyen clases como conjuntos, listas, arrays y tipos. También se han añadido métodos para el manejo de transaciones, manejo de excepciones y agrupación de objetos.

5) *Construcciones de lenguajes de BDOO embebidas en un lenguaje HOST*: Los lenguajes de BD pueden estar embebidos en lenguajes HOST; por ejemplo, Las sentencias SQL pueden estar embebidas en PL/I, C, Fortran y ADA. Algunas BDOO han tomado esta aproximación, como puede constituirlo el sistema O_2[2], que suministra extensiones embebidas en C (llamadas CO_2) y Basic.

---

[1] SANTOS, Eugenio. Seminarioo sobre Bases de Datos Orientadas a Objetos. U. Politécnica de Madrid. 1992.

[2] DAYAL U., Buchmann A., McCarthy D. Rules are objects too: A knowledge model for an active OODB system. Advances in object oriented databses system. Sep 1988. pp. 129-143.

6) *Productos de aplicaciones específicas con un SGBDOO subyacente*. Otra aproximación importante es el desarrollo de herramientas específicas para dominios de aplicación y entornos que utilicen tecnología de SGBDOO o suministren aspectos de BDOO a los dominios de aplicación específicos. Un ejemplo puede ser TEAMONE, que utiliza una configuración del sistema de administración de TeamOne Systems, Inc. Además suministra una extensión lógica en el sistema de ficheros Unix y una aproximación OO, en un repositorio de objetos.

2. ESTANDARES PARA LA INTERACCION DE OBJETOS

La tecnología OO no puede llegar a alcanzar su potencial hasta que haya una industria estándar, en la que las clases (objetos) de un proveedor sean capaces de interactuar con las clases de otros proveedores. Las clases pueden ser ejecutadas en una red de computadores de diferentes fabricantes con diferentes: sistemas operacionales, sistemas de BD, e interfaces de usuario.

Las herramientas CASE para el diseño OO y la generación de código ayudan a forzar éstos estándares, generando peticiones desde formatos estándares y empleando clases estándares desde un repositorio para obtener servicios de objetos y facilidades comunes.

2.1.1. Grupo de Gestión de Objetos

La principal organización interesada en el establecimiento de una industria estándar es la OMG (Object Management Group). Es una asociación de comercio, internacional, sin ánimo de lucro, fundada por cerca de 200 compañías de computadores y de software, cuya misión a grandes rasgos es: [3]

- Dedicada a la maximización de la *portabilidad, reusabilidad e interoperabilidad del software*. Con el fin de producir estructuras y especificaciones para la comercialización de entornos orientados a objetos.

- Proveer una *Arquitectura de Referencia* con términos y definiciones sobre las cuales deben estar basadas las especificaciones. La OMG creará una industria estándar para la comercialización de sistemas orientados a objetos, enfocándose sobre: el acceso de redes de objetos, encapsulación de aplicaciones existentes, y las interfaces de BD de objetos.

- La OMG provee un *foro abierto para su discusión* a nivel industrial, educativo y de promoción de la tecnología de objetos. Coordina sus actividades con organizaciones relacionadas y actúa como un centro para la tecnología y mercadeo del software orientado a objetos.

2.1.2. Arquitectura de Gestión de Objetos

La OMG tiene como *Modelo de Referencia* la llamada Arquitectura de Gestión de Objetos, cuya meta es la de permitir software diferente de diferentes proveedores para trabajar en conjunto. Es decir que diversas soluciones de diseño pueden ser acomodadas. El Modelo de Referencia direcciona principalmente lo siguiente:

- Cómo los objetos hacen y reciben peticiones y respuestas.

- Las operaciones básicas que podrían suministrar para cada objeto.

- Las interfaces de objetos que suministran facilidades comunes, útiles en muchas aplicaciones.

La Arquitectura de Gestión de Objetos (7), consta de cuatro partes tal como se aprecia en la siguiente gráfica (Fig. 2.1): [4]

- *Aplicaciones de Objetos*: Son aplicaciones de usuario final, construidas por diversos vendedores o casas de software.

[3] MARTIN, James y Odell, James. Object-Oriented Analysis & Desing. Prentice-Hall: New Yersey. 1992.
[4] MARTIN, James. Principles of Object-Oriented. Analysis and Desing. New Jersey: Prentice-Hall. 1993.

- *Facilidades comunes*: Son objetos y clases que suministran capacidades de propósito general, útiles en muchas aplicaciones.

- *Servicios de Objetos*: Es un conjunto de servicios que suministra funciones básicas para la realización y mantenimiento de objetos. Incluye ficheros o sistemas de BD, manejadores de transacciones, servicios de directorio, etc.

- *Agente de Solicitudes de Objetos*: Es la cabeza de esta arquitectura, permite a los objetos comunicarse independientemente de las plataformas y técnicas especificadas. La meta es garantizar que los objetos puedan interoperar con otros, ya sea que estén en la misma máquina o en máquinas conectadas a un cliente-servidor o en sistemas de redes heterogéneas.

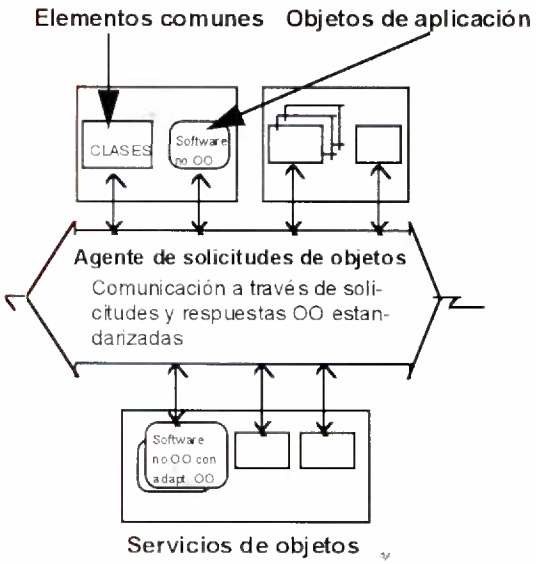

Fig. 2.1 - Arquitectura de Gestión de Objetos (Modelo de Referencia de la OMG)

Un objeto hace una petición en modo estándar y se ejecuta el *Agente de Solicitudes de Objetos*, para que la solicitud sea procesada. Este induce a algún método para ser invocado, y transfiere el resultado requerido, algunas funciones que puede direccionar son: nombre de servicios, solicitud de envío, parámetros, liberaciones, sincronización, activación, manejo de excepciones, y mecanismos de seguridad.

La arquitectura OMG, también piensa en términos de *objetos cliente y servidor*. Un objeto *cliente* solicita servicios de un objeto *servidor*, el cual acepta las solicitudes y ejecuta el servicio. El cliente y el servidor podrían estar sobre la misma máquina o sobre máquinas separadas.

2.1.3. Modelamiento con objetos

Los sistemas orientados a objetos se presentan dentro de un nuevo paradigma, que ya hace parte de muchas especialidades de la ciencia de la computación, e.g.: lenguajes de programación, bases de datos, sistemas de tiempo real, simulación, inteligencia artificial; además es claro que en la comunidad informática existe un acuerdo sobre la tendencia de los Sistemas de Información tradicionales a convertirse en Sistemas Orientados a Objetos. Este consenso general se debe en gran parte al *poder de modelamiento* que se logra con la noción de objeto, presentando una herramienta relativamente universal para representar el mundo real.

El modelamiento del mundo real ha sido una práctica utilizada por científicos e ingenieros para obtener respuestas de él. En realidad la *resolución de problemas constituye la actividad fundamental de las personas que trabajan en ciencias e ingeniería*.

El modelamiento utilizando metodologías orientadas a objetos, permite *un enfoque más natural* de los modelos con el mundo real. En el método orientado a objetos las nociones intutitivas de las entidades del mundo real se asocian con comportamientos y características de las mismas; esta noción intuitiva, es el fundamento del modelamiento orientado a objetos, donde un *objeto* es un conjunto de atributos y comportamientos, que junto con la interacción con otros objetos, permite modelar sistemas complejos paralelos a la realidad.

El desarrollo y ubicación de los objetos, han estimulado el desarrollo de nuevas metodologías de análisis y diseño de sistemas que permiten sistematizar el proceso de modelamiento, de ahí el planteamiento de la formalización de una metodología orientada a objetos, que pueda ser aplicada con la ayuda de herramientas automatizadas (OOCASE), tema de esta investigación.

3. FORMALIZACION DE UNA METODOLOGIA DE DISEÑO DE BDOO

La tecnología orientada a objetos y la tecnología CASE son aptas para trabajar en conjunto, ya que están basadas en un repositorio que permite integrar herramientas CASE y generar código libre de errores.

El *desafío de las herramientas CASE es la modelación de la realidad,* de forma tal, que sea lo más precisa para los usuarios finales y para poder transformar los modelos en requerimientos del sistema tan automáticamente como sea posible, ya que las técnicas OO son de gran ayuda para conseguirlo.

La *formalización* que se plantea en esta metodología (Fig. 3.1), se basa principalmente en representaciones gráficas que permiten suministrar un alto nivel de abstracción para que sea entendida fácilmente por los usuarios. La *interface gráfica de manipulación de objetos, es con el fin de poderse implementar a través de herramientas automatizadas de desarrollo* (OOCASE).

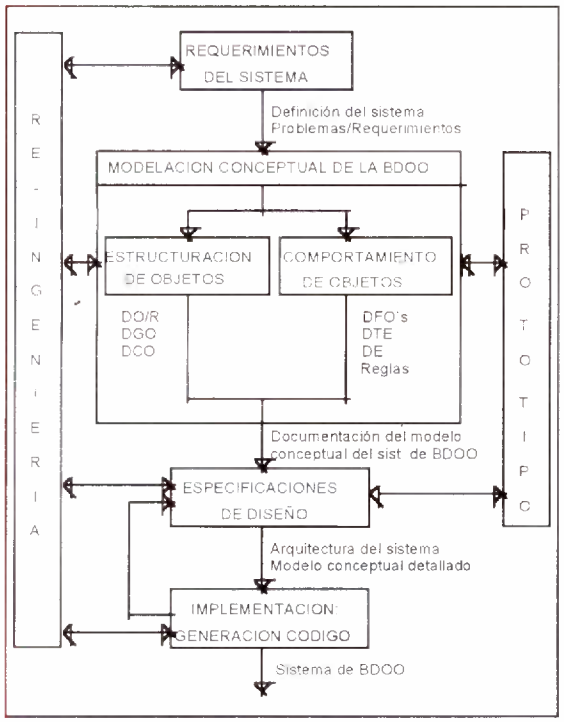

Fig. 3.1 - Formalización de una metodología de BDOO

3.1. REQUERIMIENTOS DEL SISTEMA

El objetivo de ésta etapa es lograr el conocimiento y entendimiento de la estructura de la organización, las funciones y el desempeño sistémico del sistema. Igualmente se propone determinar un análisis de costos y beneficios del sistema para determinar por ejemplo la continuidad del proyecto.

Se desarrollan básicamente las siguientes actividades:

- Investigación inicial del sistema o factibilidad: Esta primera etapa es opcional y se hace un estudio del sistema actual con el propósito de identificar la prioridad del proyecto, mediante la determinación del problema u oportunidades.

- Identificación de los requerimientos del sistema: Comprende las definiciones de los criterios que el sistema de BD debe satisfacer, entendiendo y documentando la estructura del sistema actual y los requerimientos del nuevo sistema.

3.2. MODELACION CONCEPTUAL DEL SISTEMA DE BDOO

La modelación del sistema de BDOO, se describe en términos de su estructura y su conducta o comportamiento. La *estructura* del mundo es una metáfora visual y espacial, que hace referencia a la visión estática de como los objetos están colocados en el espacio. La *conducta o comportamiento,* en contraste, se refiere a la forma como los objetos cambian de estado a través del tiempo, dentro de la estructura definida. El *estado de un objeto* es la colección de todos los tipos de objetos que son aplicados a un objeto.

Las máquinas de estado finito ofrecen una forma útil para describir los cambios de estado. Las componentes primarias para la especificación de máquinas de estado finito incluyen: tipos de eventos, reglas de activación, condiciones de control y operaciones. Las instancias de estos son: *eventos, activadores, evaluaciones de control de condiciones y la invocación de operaciones* respectivamente. Los *eventos* son cambios de estado en el objeto. Los *activadores* responden a los eventos por invocación de operaciones con los argu-

mentos requeridos por los objetos. Las *evaluaciones de control de condiciones* actúan como guardias que aseguran que la operación es ejecutada sólo cuando la condición de prueba es verdadera. Cada *operación invocada* es un proceso empleado activamente en los cambios de estado del objeto por implementación de un método especifico.

El esquema de eventos es un enfoque formal para expresar la conducta y está basado en las fundamentaciones de las máquinas de estado finito. Este incorpora las nociones de operación, nivelamiento de especificaciones, reusabilidad de operaciones, concurrencia e independencia de mecanismos de implementación. El esquema de eventos incorpora también estructuras expresadas por el esquema de objetos.

3.2.1. Modelación de la Estructuración de Objetos

La estructuración de objetos es un análisis que define los objetos que percibimos y la forma como están asociados o relacionados, y la composición entre objetos complejos.

En esta fase se identifica la siguiente información:

- Los tipos de objetos y su relación entre ellos. La identificación de objetos y sus asociaciones son representadas por un esquema de objetos, llamado modelo Objeto/Relación. Esta información guía al diseñador para la definición de clases y la estructura de datos.

- Identifica como los tipos de objetos están organizados en supertipos y subtipos. La jerarquía de generalización puede ser diagramada e indica las direcciones de herencia para el diseñador.

- Además se identifica la composición de objetos complejos. Las jerarquías de composición también pueden ser diagramadas. La composición permite definir mecanismos para la gestión correcta de objetos.

Se realizan las siguientes actividades:

- Identificación de objetos.

- Identificación de asociaciones entre objetos (Diagrama Objeto/Relación - DO/R).

- Organización y simplificación de clases de objetos: herencia (Diagramas de Generalización de Objetos o Herencia - DGO).

- Composición y Jerarquía de Objetos (Diagramas de Composición de Objetos DCO).

- Diccionario de Objetos.

. Modelo Objeto/Relación (Diagrama O/R)

Para definir el modelo Objeto/Relación que se propone, es conveniente tener claro la conceptualización y sentido de los términos *entidad* y *objeto*:

Una *entidad* es algo donde podemos almacenar datos. Es un objeto poseedor de existencia propia y conforme a las decisiones de gestión de la empresa; cuando se implementa, cada entidad es almacenada como un registro; las entidades del mismo tipo, son almacenadas como una colección de registros conocidos como archivo o relación.

Las entidades están representadas por agrupaciones de elementos de datos. En el diseño OO, los tipos de objetos tienen una estructura de datos que puede estar compuesta por muchos tipos de entidades.

Un *objeto es diferente de una entidad* por las siguientes razones:

- La encapsulación de datos y métodos (operaciones o procesos).

- La estructura de datos es más compleja.

- Enfasis en la herencia.

El diagrama Objeto/Relación es similar al diagrama Entidad/Relación, en el cual también se establecen relaciones entre objetos. Debido a que los analistas están familiarizados con los diagramas

de E/R, se propone usar las mismas convenciones para su diagramación con algunas variaciones en sus símbolos y se introduce como una extensión del modelo Entidad/Relación. Sus componentes son:

- *Los objetos*, descritos mediante sus atributos y métodos.

- *Las relaciones*, descritas mediante sus atributos: se definen dos tipos de relaciones: relación de herencia y relación entre clases (uso).

. Posibilidades y limitaciones del modelo O/R

- El modelo tiene una base matemática importante que debería permitir la generación automática de código.

- Las relaciones entre los conceptos del mundo real son descritas mucho mejor que mediante el OOD clásico.

- El modelo asegura una transición suave entre la especificación y la implementación.

- Provee una aproximación única para la modelización tanto de la aplicación como de la BD asociada (cuando existe).

- La transición del modelo al diseño OO es muy suave y fácil para el personal familiarizado con los modelos Entidad/Relación.

- Actualmente no hay herramientas que soporten el modelo.

. Diagrama de Generalización de Objetos o Herencia

El propósito del diagrama de herencia es describir las relaciones de pertenencia de herencia entre las clases de un programa simple, una librería o un entorno. El diagrama de herencia esta destinado para ser usado por los diseñadores quienes desean construir o modificar la estructura de herencia o para modificar las clases embebidas en la estructura. En resumen, el diagrama de herencia es útil durante la depuración.

. Diagrama de Composición

Es un tipo especial de relación entre tipos de objetos o clases, este puede ser leído como: "esta compuesto de".

Los diagramas jerárquicos son usados para mostrar como un tipo de objeto (o clase) esta compuesto por otro tipo de objeto (o clase).

3.2.2. Modelación del Comportamiento de Objetos

El comportamiento o conducta de objetos, tiene que ver con lo que le sucede a los objetos en el tiempo.

Se identifica la siguiente información:

- Los estados que puede tener un objeto. Estos pueden ser dibujados sobre un diagrama de transición de estados.

- Los eventos que ocurren; los eventos son cambios de estado de un objeto, los cuales pueden ser descritos por medio de un Diagrama de Eventos, donde se muestra la secuencia de operaciones y eventos.

- Las interacciones que ocurren entre objetos: un diagrama puede ser dibujado para mostrar el paso de mensajes entre clases.

- Las reglas que gobiernan los objetos y su conducta. La activación de reglas son usadas para reaccionar sobre los eventos.

- Cómo están las operaciones representadas en los métodos? Los diagramas, los elementos declarativos y otros medios son usados para especificar los métodos con suficiente detalle para generar código.

Se realizan las siguientes actividades:

- Identificación de procesos y funciones del sistema (Diagrama de Flujo de Objetos - DFO´s).

- Identificación de cambios de estado (Diagrama de Transición de Estados - DTE).

- Definición de eventos (Diagrama de Eventos - DE)

- Descripción de reglas y funciones.

. Diagrama de Flujo de Objetos (DFO´s)

Los diagramas de flujo de objetos (DFO´s), son similares a los diagramas de flujo de datos (DFD´s - utilizados en el análisis tradicional de aplicaciones), porque representan actividades que hacen interface con otras actividades. En los DFD´s la interface que se pasa son datos (a través de los llamados flujos de datos), en cambio en los DFO´s se puede representar cualquier clase de objeto que pasa de una actividad a otra, como por ejemplo: ordenes de pedido, partes, buenos acabados, diseños, servicios, hardware, software o inclusive datos. En resumen los DFO´s, nos indican los objetos que son producidos y las actividades que producen e intercambian dichos objetos.

Los DFO´s permiten representar actividades claves de la empresa o sistema, enlazados por los productos que producen e intercambian las actividades[5].

Los DFO´s son empleados a nivel estratégico como un método en un manejador OO. Adicionalmente representan los requerimientos de procesamiento de forma tal que los planificadores de negocios puedan aplicar y entender los ejercicios de planificación estratégica.

El diagrama de flujo de objetos es útil para representar estructuras de control y el flujo de procesamiento, cuando no son comprensibles los eventos y activadores dinámicos.

. Diagrama de Transición de Estados (DTE)

Amplía el modelo conceptual O/R mediante la exposición de la conducta o comportamiento de cada objeto o de las relaciones en este modelo. El modelo de estado se documenta mediante la utilización de un diagrama estándar de transición de estado o de una tabla de transición de estado. Dicha tabla de transición de estado, no es más que una lista de sucesos para cada objeto.

Un objeto puede tener uno o muchos estados, por ej. un semáforo puede tener de uno a tres estados: rojo, amarillo y verde. Cuando su estado cambia, varias acciones tienen lugar. Un cambio notable de estado es un *evento* o suceso que activa una operación, por ej: el cambio de luz de un semáforo, es un evento para motoristas y peatones; la recepción de una orden de compra es un evento y también cambia de estado al objeto *Orden de Compra*.

Es decir que la conducta de un objeto, concierne con que eventos ocurren, el correspondiente cambio de estado, y las operaciones que resultan desde estos cambios de estado. Algunas veces un objeto es creado, pasa a través de varios estados y finalmente es destrozado. En un lenguaje OO, se envían peticiones que pueden causar la activación de métodos, estos métodos cambian el estado de un objeto, el estado es registrado en los datos del objeto.

. Diagrama de Eventos (DE)

Un *evento* ocurre en un punto en el tiempo. Las operaciones son procesos que pueden ser requeridos por los objetos. Cuando las operaciones son exitosas, ocurren eventos.

Un evento representa el cambio de estado de un objeto, por ejemplo: cuando se ingresa dinero a una cuenta bancaria, se actualiza.

[5] MARTIN, James y Odell, James. Object-Oriented Analysis & Desing. New Jersey: Prentice-Hall. 1992.

. Reglas

Otra de las metas del desarrollo orientado a objetos debería ser la de evitar la programación hasta donde sea posible.

Además, en lo posible, el código para el funcionamiento del sistema debe ser generado a partir de los modelos, ya que son fáciles de entender y experimentado por los usuarios finales.

La conducta del sistema, puede describirse con la ayuda de reglas, las cuales necesariamente deben ser rigurosas porque son básicas para la generación de código. Sin embargo es importante que las reglas sean entendidas por los usuarios finales, para permitirles chequear y corregir errores de las políticas del negocio y la conducta deseada del sistema que se esta representando. Es decir que las reglas encapsulan el conocimiento de los negocios.

Hay una distinción entre los lenguajes declarativos y procedurales. Los lenguajes de programación convencionales son procedurales, donde se da un conjunto de instrucciones que son ejecutadas en una secuencia especifica, la cual puede variar dependiendo de las condiciones de prueba. Los lenguajes declarativos como su nombre lo indica declaran un conjunto de hechos y reglas, no se especifica una secuencia de pasos para su procesamiento.

Las sentencias declarativas son mucho más fáciles de comprender y validar que un lenguaje procedural.

Hasta donde sea posible, se deben construir sistemas con sentencias declarativas que puedan ser enlazadas a los diagramas OO y que sean entendidas por los usuarios finales.

. Diccionario de objetos

Constituye un depósito de los objetos del sistema en desarrollo, que contiene los detalles sobre los objetos, los atributos, relaciones entre los objetos elementales, las memorias, los procesos y el ambiente del sistema de los DFO´s. Estos diccionarios constituyen una base importante de la documentación del sistema. A través de formatos estándares podemos documentar la descripción de los objetos/clases, las relaciones o asociaciones y los atributos.

3.3. ESPECIFICACIONES DE DISEÑO

Durante las especificaciones de diseño se realiza un análisis de los modelos de objetos, con el fin de suministrar los detalles básicos para la implementación. Se toman decisiones que son necesarias para el funcionamiento del sistema sin descender a los detalles particulares de un lenguaje o de una BD.

En esta fase se determinan las definiciones completas de las clases y sus asociaciones, así como también las interfaces y algoritmos de los métodos utilizados para implementar las operaciones. No es necesario hacer transformaciones de un modelo a otro, sino que hay que describir y complementar las especificaciones del sistema de BD, formando un proceso de transición al diseño.

3.3.1. Transición al diseño

El análisis se aplica a un área completa de la empresa o negocio, y el diseño se aplica a un sistema. Para un sistema, la distinción entre análisis y diseño puede ser enfocado, en flujos de análisis al diseño. El análisis es críticamente importante porque la programación OO sin un buen análisis alcanza pocos beneficios.

En la transición o avance del análisis al diseño (Ver Figura 3.2), se realizan las siguientes actividades:

- Los tipos de objeto son expresados en más detalle como *clases*.

- La jerarquía de generalización de tipos de objeto se convierte en *clases jerárquicas* con herencia.

- Las operaciones, al lado de sus eventos, reglas y condiciones de control, llegan a ser *métodos*.

- Los modelos de datos llegan a ser estructuras de datos usadas por las clases.

- La interface de usuario es diseñada y prototipada.

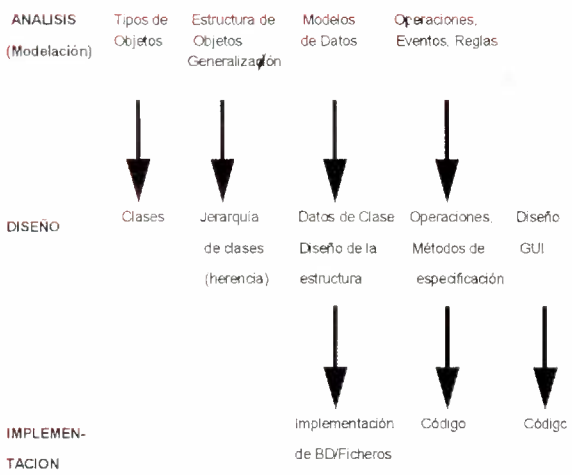

Fig. 3.2 - Transición en las etapas de desarrollo OO: del análisis al diseño y del diseño a la implementación.

Un generador de código puede ser utilizado para que el código sea implementado inmediatamente desde los métodos de especificación. El código de la descripción de la BD es implementado directamente desde la estructura de datos del diseño. La interface de usuario, preferiblemente gráfica (GUI) es generada y ajustada para hacer el prototipo. De ahí que las herramientas OOCASE deben ser seleccionadas para acoplar la generación de código, tan automáticamente como sea posible.

3.3.2. Clases

En la estructuración de objetos de la etapa de modelación, se identifican los tipos de objetos, y la implementación de estos tipos de objetos se hace en el diseño.

De ahí el concepto de que una clase es la implementación de un tipo de objeto, donde se especifica la estructura de datos y los métodos para implementar cada operación.

Un objeto, es una instancia de una clase. Los datos y las operaciones están encapsuladas y son especificadas por sus clases.

3.3.3. Herencia de clases

La generalización es una notación conceptual. La herencia de clases (normalmente se referencia como herencia) es una implementación de la generalización.

La herencia de clases permite que la estructura de datos y las operaciones de una clase, estén disponibles para ser reutilizadas físicamente por sus subclases. La herencia de operaciones desde una superclase facilita compartir código, mejor que la redefinición de código entre clases. Finalmente la herencia de estructura de datos facilita la estructura reutilizable.

3.3.4. Transición de operaciones a métodos

Las operaciones son procesos que pueden ser solicitados como unidades. Los métodos son especificaciones procedimentales (procedurales) de una operación en una clase. En otras palabras, las operaciones son las solicitudes o peticiones de servicio y los métodos son las especificaciones en código de programación para dichas operaciones.

Los métodos en una clase manipulan *solamente* la estructura de datos de esa clase, y no pueden acceder la estructura de datos de otra clase diferente.

Los diseñadores toman operaciones del Diagrama de Eventos y le agregan más detalles para especificar su diseño. Por ejemplo, del diagrama de eventos realizado en la fase de modelación, se puede obtener la siguiente operación: *Crear orden de compra*.

En el diseño se convierten tipos de objetos en clases y operaciones en métodos, así por ejemplo: el tipo de objeto *Ordenes de compra* se convierte en una clase con el mismo nombre. El diagrama de transición de estados indica las múltiples operaciones que son utilizadas con estos tipos de objetos y sus correspondientes clases; cada estado de transición puede ser relacionado con una operación. El diseñador toma cada operación y especifica el método o métodos; algunas veces son necesarios varios métodos para crear una

operación. En el ejemplo de la operación *Crear Orden de Compra* se puede calcular el total de la *Orden de Compra*, el método puede especificar la forma de calcular ese total. Para hacerlo, el método adquiere el precio de un ítem de la orden enviando una petición a la clase *Producto* que almacena los datos *Precio* de cada ítem. El método hace esto para cada ítem de la *Orden de Compra*.

3.3.5. Documentación de las decisiones del diseño

La documentación de las especificaciones y decisiones que se toman en el diseño, debe ser realizada tan pronto se realiza éste para evitar confusiones. Además cuando se trabaja con otros desarrolladores, es casi imposible recordar detalles de diseño después de un tiempo y además parte de la documentación es básica para llevar a cabo la implementación. También es la mejor forma de facilitar la transmisión del diseño a otros desarrolladores y permite ser utilizada como referencia en el mantenimiento del sistema.

El documento del diseño realmente es una una extensión del documento de los requerimientos dados en la modelación. En los formatos gráficos (diagramas de objetos) y en los formatos textuales (descripción de clases) se incluyen descripciones revisadas y mucho más detalladas de los diferentes modelos de objetos.

3.4. IMPLEMENTACIÓN: GENERACIÓN DE CÓDIGO

Las herramientas OOCASE no necesariamente utilizan lenguajes de programación orientados a objetos para la generación de código, ya que lo pueden hacer en otros lenguajes, tales como Cobol o C. El problema con este tipo de lenguajes es que no tienen las ventajas del enlace dinámico.

Algunos OOCASE generan C++, que si tienen enlace dinámico, como una opción requerida para clases que pertenecen a una superclase.

Un principio de la integración de CASE, es que la depuración no cubre el cambio de instrucciones a nivel de código, pero algunas veces cambia el diseño efectuado por los interpretadores o el generador de código, el cual crea código sin errores de sintaxis, que puede ser probado como una caja negra y el código fuente puede ser cambiado cuando sea necesario. Esto es muy importante cuando la herramienta genera código en un lenguaje tradicional como Cobol o C.

Esto realmente se conoce también con el nombre de CASE inmediato, ya que ayuda al diseñador a una alta creatividad, permitiéndole observar como escultor su creación y modificaciones a dicho desarrollo. De ahí que los desarrolladores encuentran mucho más rápido este CASE inmediato y con mayor satisfacción que las tradicionales herramientas CASE, que requieren grandes esperas para la generación y compilación de código.

Los sistemas CASE, construidos con interpretador, son altamente deseables porque los desarrolladores pueden ejecutar inmediatamente el código obtenido y repetitivamente pueden crearlo y modificarlo.

3.5. PROTOTIPO DE LA APLICACIÓN

Un prototipo es un modelo (p.e. arquetipo sobre el cual se diseña un sistema) ejecutable de la aplicación que consta inicialmente de mapas (diseño de pantallas con solo literales) y una estructura de la aplicación que define el flujo de control.

El prototipo de la aplicación puede ser opcional en el desarrollo del sistema. El prototipo generalmente facilita la comunicación con el usuario final. Ofrece un modelo ejecutable que muestra al usuario final entre otros, respuestas a lo siguiente[6]:

¿Cuáles son las características esenciales del modelo del sistema?

¿Qué información le suministra el sistema?

¿Qué información se puede encontrar en las pantallas?

[6] DAVIS Gordon y Olsson M. Sistemas de información gerencial. Bogotá: McGraw-Hill. 1987.

¿Cómo conseguir o acceder a cada pantalla?

¿Qué nuevas estructuras de objetos se requieren?

Es decir que el usuario final tiene una versión funcional de la aplicación antes de empezar la construcción de programas, permitiendo hacer los cambios oportunamente. Con esta información el usuario puede validar funciones y respuestas de la aplicación, mostrar pantallas y ejecutar rutas o caminos. Ante todo un prototipo es un modelo ejecutable de aspectos seleccionados o críticos de un sistema - objetivo. En lo que sigue se establecerá la construcción del Prototipo, se establecerá además su relación con los métodos tradicionales, ventajas y desventajas.

3.6. EL MODELO DE RE-INGENIERÍA

Según Bachman[7] más de un 80% de los recursos de información se dedican a las aplicaciones actuales mostrando una tendencia: *«No hay desarrollos nuevos sino un mantenimiento radical de los sistemas de información actuales»* lo cual implica dos aspectos: Uno bueno y otro malo. El *primero* es que la mayoría de los sistemas de información ya están informatizados lo cual implica a su vez la necesidad de mejorar lo existente y el aspecto malo es que estos sistemas han crecido tanto y de pronto en forma tan desordenada que se vuelve muy complejo su actualización. Bachman introduce el término de *Re-ingeniería* como la habilidad para retrotraer el diseño de ingeniería sumido en el código fuente de los archivos existentes y los diseños de bases de datos para mejorarlos y lograr las nuevas necesidades y entonces hacer la Ingeniería Directa.

Bachman plantea que, con las modernas herramientas de re-ingeniería de Software se puede realizar un ciclo completo de *re-ingeniería de software*. Dicha metodología se describe así:

1) Recuperar los programas existentes escritos con varios años de antelación, e.g. con 5 a 20 años, en forma mal estructurada, programación *"espaghetti"* con variedad de sistemas de acceso a ficheros y variedad de reportes y pantallas.

2) Hacer retro-ingeniería a aquellos programas aplicativos, en los cuales las definiciones de una o varias transacciones empresariales, expresadas en comunicaciones de datos, almacenamiento de datos y formas de recuperación que sean independientes de cualquier tecnología específica de implementación.

3) Mejorar la determinación y definición de aquellas transacciones empresariales, según las necesidades de los nuevos requerimientos del sistema de información.

4) Seleccionar metas de las plataformas DBMS y CMS (i.e. del sistema de administración de bases de datos y de comunicaciones). Hacer ingeniería directa a aquellas transacciones reversadas hacia un conjunto de programas nuevos ejecutándose frente a los sistemas escogidos DBMS y CMS.

5) Optimizar aquellos programas nuevos para satisfacer los requisitos de desempeño y los tiempos de respuesta del sistema.

6) Generar el código (en forma automática) para los nuevos programas fuente, Cobol, i.e. listos para su compilación.

4. CONCLUSIONES

La base fundamental del desarrollo de esta investigación, ha sido el estudio, análisis y evaluación de los diferentes conceptos, especificaciones, técnicas, métodos y metodologías aplicadas al diseño de bases de datos orientadas a objetos con el apoyo de herramientas CASE, con el fin de crear o plantear una propuesta de formalización o normalización de estándares para su desarrollo; teniendo en cuenta que hay una creciente preocupación tanto de la comunidades académicas e investigativas, así como de las instituciones encargadas de la adopción de estándares, para llegar a una unificación de criterios que permitan un desarrollo industrial de software más rápido y más fácil de implementar y mantener.

[7] BACHMAN C. A personal Chronicle: creating better information system with some guiding principles. IEEE Knowledge and Data Engineering. Vol. 1 No. 1. 1989.

A pesar de que existen organizaciones internacionales que están trabajando en la adopción de estándares, se espera que esta investigación contribuya desde el punto de vista académico a su proceso: teniendo en cuenta que se han logrado cumplir los objetivos que se plantearon al inicio de la investigación, concretamente en:

- La **formalización de una metodología** de diseño de base de datos orientada a objetos.

- El soporte con técnicas gráficas orientadas a objetos, las cuales son básicas para su aplicación a través de herramientas OOCASE. Es decir que después de un análisis de la aplicabilidad de diagramas en las diferentes etapas de desarrollo, se han adaptado y estandarizado los **diagramas que más fortalecen la metodología en mención.**

- La estandarización de conceptos de orientación a objetos, que sin embargo en mi opinión seguirán siendo muy sesgados de acuerdo al enfoque que se quiera tratar. Dentro de éste concepto de estandarización se proponen complementariamente, unos **estándares generales para el desarrollo de sistemas de BDOO**

- Visión de la problemática del desarrollo de software y el planteamiento de soluciones viables a través de modernas técnicas de desarrollo y actualización, tales como la orientación a objetos, las **herramientas OOCASE** y la re-ingeniería.

- Documento de soporte al aprendizaje de la orientación a objetos y apoyo a futuras investigaciones e implementaciones.

- La adaptación de modelos tradicionales a modelos de orientación a objetos, con el fin de ofrecer una continuidad y una facilidad en el desarrollo de software, sin desechar conocimientos y experiencias adquiridas y aplicadas en la vida real.

BIBLIOGRAFIA

BACHMAN C. A personal chronicle: creating better information system with some guiding principles. IEE Knowledge and Data Engineering. Vol. 1 No. 1. 1989.

BROWN, Alan. Object Oriented Databases - Applications in Software Engineering. London: McGraw-Hill. 1991.

BUTLER, Martin; Bloor, Robin y Beach Paul. Database: An Evaluation and Comparison. Marc Beishon. United Kingdom. 1990.

DAVIS, Gordon y Olsson M. Sistemas de información gerencial. Bogotá: McGraw-Hill. 1987.

DAYAL, U., Buchmann, A. y Mcarthy D. Rules are objects Too: A Knowledge Model for an Active OODB System. Advances in Object-oriented Databses System. Sep 1988. pp. 129-143.

DITTRICH, Klaus; Dayal, Umeshwar y Buchamann, Alejandro. On Object-Oriented Database Systems. New York: Springer-Verlag. 1991.

MARTIN, James y Odell, James. Object-Orientes Analysis & Desing. Prentice-Hall: New Yersey. 1992.

MARTIN, James. Principles of Object-Oriented. Analysis and Design. New Jersey: Prentice Hall. 1993.

McCLURE, Carma. CASE is Software Automation. New Jersey: Prentice Hall. 1989.

MEDINA G., Víctor H. y Pérez, G., Alfonso. Desarrollo de Sistemas, Prototipos y Re-ingeniería con el apoyo del Computador: Herramientas CASE. Bogotá: U. Nacional. 1993.

MEDINA G., Víctor H. Diseño de Sistemas Informáticos en entorno CASE. U. Politécnica de Madrid. 1990.

RUMBAUGH, James; Blaha, Michael; Premerlani, William; Eddy, Frederick y Lorensen, William. Object oriented modeling and desing. New Jersey: Prentice-Hall. 1991

SANTOS, Eugenio. Seminario: Bases de Datos Orientadas a Objetos. U. Politécnica de Madrid. 1992.

SHLAER, Sally y MELLOR, Stephen J. Object Lifecycles - Modeling the World in States. New Jersey: Yourdon Press. 1992.

YOURDON, Edward y Constantine, Larry. Structured Design Fundamentals of a Discipline of Computer Program And Systems Design. New Jersey : Yourdon Press. Prentice Hall. 1979.