Universidad Central de Venezuela

Facultad de Ciencias

Escuela de Computación

Lecturas en Ciencias de la Computación

ISSN 1316-6239

Fundamentación Teórica de las Bases de Datos Orientadas a Objetos

Prof. Yosly Hernández Bieliukas

Prof. Antonio Silva Sprock

ND 2009-02

Centro de Investigación en Sistemas de Información (CISI)

Caracas, Octubre 2009.

FUNDAMENTACIÓN TEÓRICA DE LAS BASES DE DATOS ORIENTADAS A OBJETOS

Yosly Hernández Bieliukas[1, 2], Antonio Silva Sprock [2]

[1] Universidad Central de Venezuela, Facultad de Ciencias, Coordinación de Extensión, Unidad de Educación a Distancia

Av. Los Ilustres, Los Chaguaramos, Caracas, 1043, Venezuela.

yosly.hernandez@ciens.ucv.ve

[2] Universidad Central de Venezuela, Facultad de Ciencias, Escuela de Computación.

Av. Los Ilustres, Los Chaguaramos, Caracas, 1043, Venezuela.

antonio.silva@ciens.ucv.ve

Resumen

Hasta la aparición de las Bases de Datos Orientadas a Objetos (BDOO), las Bases de Datos (BD) tradicionales no estaban diseñadas para almacenar objetos, con lo que al guardar los datos de un programa bajo el enfoque Orientado a Objeto incrementaba significativamente la complejidad del programa, dando lugar a más código y más esfuerzos de programación, así como al problema del Desfase de la Impedancia (diferencia de esquemas).La tecnología utilizada en BDOO es la consumación de la programación orientada a objetos y la tecnología de BD (modelado de objetos y las BD tradicionales). Muchos de los propósitos de las BDOO son los mismos que los de las bases de datos tradicionales, pero con la ventaja adicional de poder representar modelos de datos más complejos (objetos cuyo valor de algún atributo es otro objeto) en un marco mucho más eficiente, además de permitir la persistencia de los objetos, manteniendo la integridad y las relaciones entre ellos.

TABLA DE CONTENIDO

1.- INTRODUCCIÓN

Las Bases de Datos Relacionales (BDR) lideran el mercado en vista de que son las más utilizadas, son ideales para aplicaciones tradicionales que soportan tareas administrativas y de negocio, sin embargo, como resultado de avances recientes en hardware y software, han surgido aplicaciones más sofisticadas, sistemas multimedia, sistemas de información geográfica (GIS) y médica, aplicaciones 3D y sistemas inteligentes, aplicaciones de bioinformática, telecomunicaciones y robótica, entre otras, las cuales tienen requisitos y características diferentes a las aplicaciones tradicionales, que pueden caracterizarse por estar compuestas de elementos complejos, transacciones de mayor duración, nuevos tipos de datos para almacenar elementos multimedia, y la necesidad de definir operaciones no estándar para la aplicación. Dentro de las aplicaciones se definen las Orientadas a Objeto (OO), y en general el paradigma de Programación Orientada a Objeto (POO) cuyos elementos complejos, antes referidos, son Objetos.

En tal sentido, las BDR no están diseñadas para almacenar estos Objetos, con lo que al guardar los datos de un programa OO incrementa significativamente la complejidad del programa, dando lugar a más código y más esfuerzos de programación, así como al problema del Desfase de la Impedancia, definido esto como la diferencia de esquemas entre la característica del repositorio y los elementos a almacenar.

Por ello, las Bases de Datos Orientadas a Objetos (BDOO) se propusieron con la idea de satisfacer las necesidades de estas aplicaciones más complejas aprovechando las ventajas que ofrece la Programación Orientada a Objetos (POO) como una forma de resolver problemas utilizando modelos que se han organizado en base a conceptos del mundo real.

Las bases de datos tradicionales almacenan sólo datos, mientras que las BDOO almacenan objetos, con una estructura arbitraria y un comportamiento. Una simple metáfora de Esther Dyson referenciada por Martínez (S/F), ayuda a ilustrar

la diferencia entre ambos modelos. Consideremos el problema de almacenar un automóvil en un garaje al final del día. En un sistema de objetos el automóvil es un objeto, el garaje es un objeto, y hay una operación simple que es almacenar-automóvil -en-garaje. En un sistema relacional, todos los datos deben ser traducidos a tablas, de esta forma el automóvil debe ser desarmado, y todos los pistones almacenados en una tabla, todas las ruedas, en otros elementos del automóvil. Por la mañana, antes de irse a trabajar hay que componer de nuevo el automóvil para poder conducir, el problema surge al componer las piezas porque puede salir una moto en vez de un automóvil. Es por ello que las BDOO son ideales para almacenar y recuperar datos complejos permitiendo a los usuarios su navegación directa (sin un mapeo entre distintas representaciones).

2.- CONCEPTOS BÁSICOS

A continuación se describen los conceptos básicos importantes para entender las BDOO.

a) **Clases:** abstracción conceptual que permite describir un conjunto de objetos que tienen el mismo tipo (Marín, 2001). Una clase no es más que un patrón en el que se basan aquellos objetos que tienen propiedades similares. Por ejemplo, la clase Persona donde los atributos que lo componen son nombre, apellido y edad.

b) **Instanciación:** mecanismo que permite crear objetos de una clase determinada. Por ejemplo, si se tiene la clase Persona a partir de este mecanismo se crea el objeto Juan, Marta, entre otros.

c) **Objetos*:*** corresponden a todos los elementos que se manipulan dentro de una BDOO. Un Objeto es una representación abstracta del mundo real, el cual está compuesto por un estado (las propiedades y sus respectivos valores), y un comportamiento (las operaciones asociadas que permiten interactuar con otros objetos y consigo mismo). Un Objeto no es más que

una instancia de una determinada clase, por ejemplo el objeto Juan el cual se obtiene a través de la Clase Persona.

d) **Variable de Instancia*:*** según (Marín, 2001) son cada uno de los atributos que caracterizan el estado de un objeto. Por ejemplo, si se tiene el atributo edad, la variable de instancia será el valor que tenga asignado, para este caso 27.

e) **Identidad de un Objeto:** se implementa a través de un identificador único, **OID (*Object Identifier*)**, generado por el sistema. El valor de un OID no es visible para el usuario externo, el sistema lo emplea internamente para identificar cada objeto de manera única, así como también, crear y manejar referencias entre objetos. El concepto de identidad hace que sea necesario distinguir:

   - **Igualdad de identidad:** dos objetos son iguales si tienen el mismo OID. En este caso se suele decir que son el mismo objeto. Normalmente se representa por el símbolo '='.

   - **Igualdad de valor:** dos objetos son iguales si los valores de sus atributos son iguales. Normalmente se representa por el símbolo '=='.

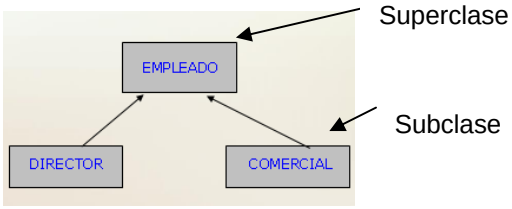
f) **Método:** es un procedimiento algorítmico a través del cual se realiza una determinada operación sobre el comportamiento de un objeto, un método se caracteriza por tener su nombre, sus parámetros formales y su valor de retorno (si es el caso).Los métodos de los objetos pueden ser de tres tipos: observadores, que devuelven información acerca del estado de un objeto; modificadores, que cambian un objeto de un estado válido a otro; y constructores, que permiten la creación de objetos. (Marín, 2001).

g) **Mensajes:** refieren a las señales que envía un objeto a otro para que ejecute una determinada operación, no es más que el medio de comunicación entre los objetos.

h) Tipo: modelo de los rasgos comunes de un conjunto de elementos que tienen las mismas características. Por ejemplo, los objetos de tipo persona.

i) Herencia: mecanismo mediante el cual una clase puede ser definida sobre la base de la definición de de otra clase. Por este mecanismo, la subclase *hereda* los atributos que definen la estructura de la superclase y los métodos que caracterizan su comportamiento. Además, la subclase puede añadir nuevos atributos y métodos para completar su definición. Los tipos de herencia son: **simple** (una clase hereda de una única superclase) y **múltiple** (una clase hereda de más de una superclase). (Bertino & Martino, 1995).

Ejemplo



j) Polimorfismo: mecanismo que permite definir e invocar funciones que comparten la misma interfaz pero tienen una implementación diferente (Marín, 2001). Por ejemplo, si se tiene el método CalcularArea() depende del objeto que lo implemente se ejecutará, porque calcular el área de un cuadrado no es igual a calcular el área de un círculo.

k) Persistencia: capacidad del programador para que sus datos se conserven al finalizar la ejecución de un proceso, de forma que se puedan reutilizar en otros procesos. Los datos tienen que persistir sin necesidad de que el usuario explícitamente los copie (Marín, 2001).

l) Un Objeto persistente: es un objeto que sobrevive a la ejecución del proceso que lo creó. Mientras que un **Objeto transitorio** es un objeto que deja de existir cuando el proceso que lo creó termina su ejecución.

m) Encapsulamiento: ocultar la implementación de un método, dejando visible la especificación. Puede tener niveles de: privada, público y protegida (Marín, 2001).

3.- CARACTERÍSTICAS DE LAS BDOO

Las BDOO aparecen a finales de los 80`s, están estructuradas para simplificar la programación OO, permiten almacenar los objetos directamente en la Base de Dato (BD), utilizando las mismas estructuras y relaciones que los lenguajes de POO, destacando que surgen de la combinación de éstos con las BD tradicionales.

Dentro de las principales características de las BDOO se tiene:

- Permiten trabajar de una manera transparente y eficiente en un entorno de programación basado en objetos, soportan todos los conceptos de la OO.
- Gestión de datos complejos como lo son por ejemplo los datos multimedia (imagen, video, entre otros)
- Permiten la persistencia transparente de los objetos.
- Presentan en muchos casos una arquitectura distribuidaProcesamiento transaccional que soporta la concurrencia.

Adicionalmente, en general soportan las siguientes características (aunque depende de cada Sistemas Manejadores de Base de Datos Orientados a Objetos (SMBDOO)) la Integridad de datos, versionamiento de objetos, indexación, seguridad, y tolerancia a fallos, entre otras.

4.- MANIFIESTOS ACERCA DE LOS SMBDOO

- Manifiesto de los Sistemas de Bases de Datos al Objeto puras de Atkinson (1989): enfoque purista que sostiene que los SMBDOO deben soportar una

modelo de objetos puros y no basarse en extensiones semánticas de modelos clásicos como el relacional.

- Manifiesto de los SBD de Tercera Generación de Stonebraker (1990): SMBD Relacionales extendidos que sean capaces de soportar los conceptos de orientación al objeto. Es la postura que propugnan los principales vendedores de productos relacionales.

- Tercer manifiesto de Darwen y Date (1995): reinterpretan el modelo relacional bajo la visión orientada al objeto.

4.1.- Manifiesto de Atkinson. Características de los SMBDOO Puros.

Las características que debe seguir una sistema de gestión de bases de datos para considerarse orientado a objetos se dividen en tres grupos:

- Obligatorias – Trece reglas

- Opcionales, aquellas que pueden añadirse para mejorar el sistema pero que no son obligatorias – Cinco reglas

- Abiertas, aquellas que el diseñador puede incluir para adaptar el sistema a sus necesidades. – Cuatro reglas.

Características Obligatorias: "Las Reglas de Oro".

I. Tipos Complejos: un sistema de BBDDOO debe poder dar cabida a diferentes tipos de datos, desde los más sencillos a objetos más complejos, como:
   - Tipo abstracto para construir otros tipos: permite construir nuevos tipos a partir de caracteres, float, enteros. Por ejemplo: punteros, números complejos, cadenas de bits.
   - Array: son matrices de elementos de datos, como los que podemos encontrar en innumerables aplicaciones científicas.
   - Secuencia de tipo: insertar elementos en una matriz en cualquier posición.

- Tuplas: forma natural de representar las propiedades de una entidad.
- Conjunto: forma natural de representar los grupos del mundo real.
- Funciones: para crear, modificar y borrar otros objetos.
- Uniones: elementos de datos que pueden tomar diferentes valores de los diferentes tipos, es decir, matrices heterogéneas.
- Composición recursiva del resto de tipos: se utiliza para representar objetos complejos, tales como documentos, espacios geométricos.

II. Identidad de Objeto

III. Encapsulamiento

IV. Tipos y Clases: existen dos categorías principales de sistemas orientados a objetos, los que soportan el concepto de **clase** y los que soportan el concepto de **tipo**. Un tipo en un sistema orientado a objetos se corresponde con el concepto de tipo abstracto de datos.

Es un conjunto de objetos que tienen un mismo comportamiento (comparten una misma funcionalidad) que se puede observar desde afuera. Esto significa que el tipo al cual un objeto pertenece depende de qué operaciones puedan invocarse sobre el objeto, cuál es el orden y tipo de sus argumentos y cuál es el tipo del resultado.

El concepto de clase es diferente al de tipo. Su especificación es la misma que la de un tipo, pero es una noción de tiempo de ejecución. Contiene dos aspectos:

- La fábrica de objetos: crea nuevos objetos de la clase.
- Almacén de objetos: conjunto de objetos que son las instancias de la clase, su extensión.

V. Herencia

VI.- Polimorfismo

VI.- Completitud de cálculos: puede expresarse cualquier función computable utilizando el lenguaje de modificación de datos de un sistema de base de datos.

VIII.- Extensibilidad: el conjunto de tipos predefinidos que aporta el sistema de base de datos debe ser extensible mediante algún mecanismo que permita definir tipos nuevos. No debe haber distinción en cuanto al uso de los tipos definidos por el sistema y los extendidos.

IX.- Persistencia

X.- Gestión de almacenamiento: la gestión del almacenamiento secundario es soportada por un conjunto de mecanismos que no son visibles al usuario, tales como gestión de índices, agrupación de datos, selección del camino de acceso, optimización de consultas, etc. Estos mecanismos evitan que se tengan que escribir programas para mantener índices, asignar el almacenamiento en disco, o trasladar los datos entre el disco y la memoria principal, creándose de esta forma una independencia entre los niveles lógicos y físicos del sistema.

XI.- Concurrencia: el SMBDOO debe gestionar el acceso de múltiples usuarios a la vez. Soportando la noción de atomicidad de una secuencia de operaciones y la compartición controlada.

Debe ofrecerse serialización de las operaciones. Si se introduce concurrencia en nuestro sistema hay que considerar cómo los objetos activos sincronizan sus actividades con otros, así como con objetos puramente secuenciales. Por ejemplo, si dos objetos activos intentan enviar mensajes a un tercer objeto, hay que tener la seguridad de que existe algún mecanismo de exclusión mutua, de forma que el estado del objeto sobre el que se actúa no está corrupto cuando los dos objetos activos intentan actualizarlo simultáneamente. En presencia de la concurrencia no hay que definir simplemente los métodos de un objeto; hay que asegurarse también de que la semántica de estos métodos se mantiene a pesar de la existencia de múltiples usuarios concurrentes.

XII.- Recuperación: el sistema debe proporcionar como mínimo el mismo nivel de recuperación que los sistemas de bases de datos actuales. De forma que, tanto en caso de fallo de hardware como de fallo de software, el sistema pueda retroceder hasta un estado coherente de los datos. Los fallos de los que un sistema debe ser capaz de recuperarse pueden producirse por diferentes motivos, por ejemplo:

- Interrupción en el suministro de energía, de forma que se pierda la información almacenada en la memoria principal y en los registros de uso general.
- Errores de software, debido a que los resultados generados son incorrectos, lo que provoca respuestas erróneas a los usuarios y que la base de datos entre en un estado incongruente.

Una vez que el subsistema de recuperación de la base de datos ha detectado el fallo, procede a su clasificación (ya que cada uno de ellos debe manejarse de manera diferente), para posteriormente restaurar la base de datos al estado anterior al de la ocurrencia del fallo.

XIII.- Facilidad de consultas *ad-hoc*: el sistema debería proporcionar la funcionalidad de un lenguaje de consulta *ad hoc*, es decir, permitir al usuario hacer cuestiones sencillas a la base de datos. Este tipo de consultas tienen como misión proporcionar la información solicitada por el usuario de una forma correcta y rápida.

En los SBDOO los lenguajes de consulta constituyen una funcionalidad importante.

Estos lenguajes de consulta deben satisfacer cuatro criterios:

- Ser de alto nivel, es decir, deben ser capaces de expresar consultas no triviales en pocas palabras.
- Ser declarativas, es decir, que la atención se dirija hacia el *QUE* se desea, y no hacia *CÓMO* se obtiene.

- Ser eficientes, esto es, la formulación de la consulta debe permitir alguna forma de optimización.
- Ser independientes de la aplicación, es decir, debería trabajar sobre cualquier base de datos.

Pero a diferencia de los SMBD relacionales, en los sistemas orientados a objetos, los lenguajes de consulta no constituyen la única forma de acceder a los datos. En los SMBDOO también se puede acceder a los datos de forma navegacional. Esta forma de acceso se basa en los identificadores de los objetos y en la jerarquía de agregación, de forma que dado un IDO, el sistema accede al objeto directamente y navega a través de los objetos a los que se refieren sus atributos.

Características Opcionales

I.- Herencia Múltiple

II.- Comprobación de tipos: el SMBDOO debe realizar una comprobación de tipos para evitar posibles problemas de rebasamiento y compatibilidad de tipos en ejecución.

III.- Distribución: los objetos de las bases de datos pueden estar almacenados en diferentes ubicaciones, sin que el usuario al acceder a ellos sea consciente de esta circunstancia, el sistema debe proporcionar una forma de realizar esta operación de manera transparente.

IV.- Transacciones de diseño

V. Versiones: la mayoría de aplicaciones CAD/CAM y CASE, comprenden una actividad de diseño que requiere alguna forma de versionado. Una versión es una instantánea semánticamente significativa tomada al objeto de diseño en un momento dado en el tiempo. Una versión de un objeto se crea a partir de las modificaciones hechas en el tiempo a las versiones previas de éste, comenzando

con una versión inicial. La traza de estas *derivaciones* se mantiene a través de una *historia* de las versiones. Una versión de un objeto complejo puede constar de las versiones específicas de sus objetos componentes.

Opciones Abiertas

I.- Paradigma de programación

II.- Sistema de representación

III.- Sistema de tipos

IV.- Uniformidad

4.2- Manifiesto de Stonebraker

Se considera el manifiesto de definición de sistemas gestores de bases de datos de tercera generación siendo

- SGBD primera generación: Sistemas Jerárquicos y de Red.
- SGBD segunda generación: Sistemas Relacionales.
- SGBD tercera generación: siguiente generación de sistemas de bases de datos cuyos principios básicos de desarrollo presenta el manifiesto.

Las características se recogen en tres principios básicos junto con trece proposiciones que indican los requerimientos más específicos de estos sistemas, las cuales no difieren mucho de las trece características obligatorias que indicaba el manifiesto de Atkinson. Así los tres principios y las proposiciones para conseguirlos son los siguientes:

Primer Principio

“Además de los servicios tradicionales de gestión de datos, los SMBD de tercera generación proporcionarán gestión de objetos y reglas más ricas”

Proposiciones:

1. Los SGBD de la tercera generación debe tener un sistema de tipos rico
2. La herencia es aconsejable
3. La reutilización y la encapsulación son aconsejables.
4. Se deberían asignar IDO para los registros sólo si no está disponible una clave primaria.
5. Las reglas de convertirán en una característica primordial de los futuros sistemas. Las reglas no deberían asociarse con una función específica.

Segunda Principio

"Los SGBD de tercera generación deben incluir a los SGBS de segunda"

Proposiciones

1. Un SGBD de la tercera generación debe tener un lenguaje de acceso declarativo y de alto nivel.
2. Deben existir dos formas de especificar colecciones: por enumeración de sus miembros o mediante un lenguaje de consulta.
3. Las vistas deben ser actualizables.
4. Los indicadores de resultados no deben aparecer en los datos.

Tercer Principio

"Los SGBD de tercera generación deben estar abiertos a otros subsistemas"

Proposiciones

1. Se puede acceder a un SGBD de tercera generación desde múltiples lenguajes de alto nivel.

2. Debe soportar la persistencia de las variables.

3. El lenguaje SQL es una forma universal de expresión de datos.

4. Las consulta y sus respuestas deben constituir el nivel más bajo de comunicación entre un cliente y un servidor.

4.3.- Tercer Manifiesto

El enfoque purista ha sido duramente criticado por expertos de bases de datos relacionales:

“Los productos relacionales se apoyan en un lenguaje basado en la lógica, que lleva más de dos mil años demostrando su validez”.

“Sería una pena desperdiciar más de veinte años de investigación y desarrollo en bases de datos relacionales”.

El Tercer Manifiesto de Darwen y Date (1995)

- Reinterpreta el modelo relacional bajo una visión orientada al objeto.

- Propone un lenguaje D que proporciona algunas ventajas de la orientación al objeto, como los tipos de datos y la herencia, manteniendo el fundamento teórico del modelo relacional. No se trata de una extensión del lenguaje SQL.

- Según el manifiesto, tal lenguaje D, debe estar sujeto a una serie de prescripciones, proscripciones y lo que denomina “sugerencias muy fuertes” las cuales divide en categorías.

    - RM: surgen del Modelo Relacional

    - OO: no surgen del Modelo relacional

5. VENTAJAS Y DESVENTAJAS DE LASBDOO

Los SMBDOO adquieren gran importancia debido a que éstas resuelven gran parte de los inconvenientes que los sistemas tradicionales no podían resolver.

- La cantidad de información que maneja un SMBDOO es mucho mayor, además del modelado de dicha información es mucho más sencillo y orientado a la extensibilidad.

- Los SMBDOO permiten el manejo de objetos más complejos, lo que permite soportar la integración de bases de datos multimedia y diversos tipos especializados.

- La estructura de la base de datos está dada por referencias (o apuntadores lógicos) entre objetos.

- Los SMBDOO posibilitan el versionado de objetos, lo que permite ayudar en los cambios de modelado que se produzcan en el sistema.

- Rapidez en el desarrollo de aplicaciones y en su mantenimiento debido a la reutilización de clases.

- Las clases genéricas cuentan con una mayor potencia, pero lo más importante es que pueden ser reutilizadas. Como las clases pueden ser reutilizadas, el material redundante no necesita ser diseñado, y esto proporciona una facilidad en el desarrollo y mantenimiento de las aplicaciones.

- La utilización de una BDOO simplifica la conceptualización ya que la utilización de objetos permite representar de una manera más natural la información que se quiere guardar.

- Mejora el flujo de comunicación entre los usuarios, los diseñadores y los analistas.

- Simplifican la POO

A pesar de las ventajas, los inconvenientes implícitos también son importantes.

- Educar a las personas en el paradigma orientado a objetos requiere una cantidad de tiempo considerable, dinero y otro tipo de recursos.

- Otra desventaja es que es estrictamente necesaria una forma de comunicación y una forma de trabajo conjunta entre los sistemas tradicionales y los OODBMS. Esto no existe e implica un costo temporal, monetario y de recursos.

- No existe un lenguaje de consulta específico como SQL, aunque resulta más facil realizar consultas complejas a un SMBDOO. Además no hay ningún estándar lo suficientemente establecido para el diseño y la implementación aunque la ODMG (*Object Database Management Group*) trabaja en ello.

- Los métodos definidos sin un SMBDOO para el diseño de clases establecen un restricción lógica sobre cómo pueden ser accedidos y manipulados los datos actuales, haciendo que las consultas realizadas “ad hoc” sean difíciles de ejecutar sin la BDOO. Para ello, “*The Object Oriented Database System Manifiesto*” propone un *browser* gráfico que proporcione la funcionalidad necesaria para construir consultas simples para el usuario.

- En cuanto a la gestión de transacciones, el problema surge debido a que los métodos de las BDOO incluyen gran cantidad de datos complejos y no se deben perder cuando ocurra *roolback.* Para ello, igual que el modelo relacional, se proponen los *chekpoints* en transacciones largas para realizar *rollback*s parciales. El problema, sigue siendo determinar dónde ponerlos y cuándo.

6.- DIFERENCIAS ENTRE LOS SMBDR, SMBDOO y SMBDRO

Adicional a los ya conocidas SMBDR y los SMDBOO, tratados a lo largo de este tema, se debe mencionar los SMBDRO (Sistema Manejador de Base de Datos Relacional Objeto), que reinterpreta el modelo relacional bajo una visión orientada al objeto (Ver tabla 1)

Tabla 1.- Comparación entre los SMBDR SMBDRO y SMBDOO

SMBDR	SMBDOO	SMBDOR
Utiliza la clave primaria	Utiliza el OID	Utiliza un alias para cada tabla, el cual debe ser único
No se pueden crear nuevos tipos de datos.	Permite la creación de nuevos tipos de datos	Igual a SMBDOO
Utiliza tablas	Utiliza	Igual a SMBDR
Utiliza solo tecnología de las BDR	Utiliza las técnicas de la POO	Combina la POO con la tecnología de las BDR
En consultas utiliza las sentencias de SQL: Select, From, Where	En consultas utiliza las sentencias de SQL, pero el lenguaje es llamado OQL: Select, From, Where	Introduce un API separado (basado en SQL) para manipular los datos almacenados, las definiciones de clase se deben mapear a los tipos de datos soportados por el sistema. En desarrollo SQL3
No puede manipular datos complejos	Permite la manipulación de datos complejos	Igual a SMBDOO
Utiliza un Gestor de Memoria intermedia	La asignación de las posiciones de memoria de los objetos se realiza basada en las relaciones entre los objetos	Igual a SMBDR

7.- DISEÑO DE LAS BDOO

Bertino & Martino (1995) menciona sobre el modelo de datos que no hay un modelo común a utilizar como punto de referencia, ningún fundamento formal, tampoco ningún estándar para los modelos OO, como si lo hay en el caso del modelo relacional.

Sin embargo, como se está trabajando en un entorno de POO, para realizar el diseño conceptual de una BDOO se utiliza el Lenguaje UML (Unified Modeling Language), como herramienta para representar simbólicamente el conjunto de datos persistentes relacionados entre sí. UML 2.0 provee un conjunto de diagramas que permiten representar gráficamente tanto la estructura (parte estática) como el comportamiento (parte dinámica) de la BDOO.

Por ello, para modelar la estructura o vista lógica de la BD, se utiliza el **Diagrama de clases** que permite presentar las clases con sus respectivas relaciones estructurales y de herencia, además del **Diagrama de Objetos** cuando no está muy claro y preciso cómo serían las instancias de las clases o para especificar más el Diagrama de Clases.

Mientras que, para modelar la parte dinámica, la interacción y comportamiento entre los objetos, se emplearía el **Diagrama de Secuencia** para presentar las interacciones entre los objetos organizados en una secuencia temporal y describir como estos objetos colaboran; así como también, el **Diagrama de Estado** para mostrar los posibles estados en que puede encontrarse un objeto y las transacciones que pueden causar un cambio de estado, luego que ocurre un evento. Cabe destacar, que pueden ser utilizados otros diagramas pero esto dependerá de cuan complejo sea la BDOO que se desea modelar, es necesario utilizar los diagramas de una manera precisa para no redundar en la información que se representa.

8.- ESTÁNDARES DE PERSISTENCIA

Existen dos estándares importantes, utilizados para el manejo de la persistencia en las BDOO, los cuales son:

8.1.- Estándar Object Database Management Group 3.0 (ODMG 3.0)

ODMG 3.0 es un modelo estándar para la semántica de los objetos de una BDOO, adopta una arquitectura que consta de un sistema de gestión que soporta un lenguaje de BDOO, con una sintaxis similar a un lenguaje de programación OO. Tiene por objetivo establecer un conjunto de especificaciones que permitan que un desarrollador pueda escribir aplicaciones portables, que puedan correr sobre más de un producto.

8.1.1- Características

Dentro de las principales características se tienen:

- Persistencia transparente
- Garantiza la atomicidad, consistencia, aislamiento y durabilidad de en una transacción
- Soporta expresiones de consultas complejas
- Conexión con lenguajes de programación como: Java, C++, y Smalltalk
- El modelo de objetos ODMG 3.0 permite que tanto los diseños como las implementaciones, sean portables entre los sistemas que lo soportan.

8.1.2.- Componentes

ODMG 3.0 está compuesto por cuatro elementos fundamentales, los cuales son:

a.- Modelo de Objetos.

ODMG se basó en el Modelo de Objetos del OMG que sigue una arquitectura de "núcleo + componentes". Especifica la semántica que se puede definir explícitamente, además de las construcciones que son soportadas por un SMBDOO. Los componentes del Modelo de Objetos son:

- **Objetos:** cada objeto tiene un **OID,** que no cambia y que no se reutiliza cuando el objeto se borra. Además tienen un **Nombre, Tiempo de vida** pueden ser **transitorios o persistentes, estado y comportamiento.** Los tipos de objetos se descomponen en **atómicos** (definidos por el usuario: boolean, short, long, unsigned short, unsigned long, float, double, octet, char, string, enum), **colecciones** (Set, Bag, List, Array y Dictionary) y tipos **estructurados** (Date, Time, Timestamp e Interval). Los objetos **colección** identificados por el estándar son los siguientes: Set<tipo> (grupo desordenado de objetos del mismo tipo. No se permiten duplicados), Bag<tipo>: (igual a Set pero permiten duplicados), List<tipo> (grupo ordenado de objetos del mismo tipo. Se permiten duplicados), Array<tipo> (grupo ordenado de objetos del mismo tipo que se pueden acceder por su posición. Su tamaño es dinámico y los elementos se pueden insertar y borrar de cualquier posición) y Dictionary<clave, valor> (es como un índice. Está formado por claves ordenadas, cada una de ellas emparejada con un solo valor).

- **Literales**: son valores constantes que no pueden ser modificados y no tienen identificador, además se encuentran embebidos en los objetos y no pueden ser referenciados de modo individual. Ejemplo `hola`, 5, `literal`

- **Clases**: permiten describir el estado y comportamiento de un conjunto de objetos que tienen el mismo tipo.

- **Comportamiento**: se especifica a través de un conjunto de operaciones,en cada operación se define el nombre, el nombre y el tipo de cada argumento,

el tipo de valor retornado y el nombre de las excepciones que la operación puede ocasionar.

- **Interfaz:** describe los atributos, relaciones y operaciones visibles de una clase a otros objetos. No son instanciables pero sirven para definir operaciones heredables por objetos definidos por el usuario.

- **Propiedades:** corresponden a los atributos (define el estado del objeto) y las relaciones que tienen con otros objetos, que pueden ser de uno a uno, de uno a muchos y de muchos a muchos (la cual se representa a través de los tipos de colecciones de objetos).

- **Herencia:** soporta la herencia simple de clases e interfaces. Las interfaces no son instanciables, se suelen utilizar para especificar operaciones abstractas que pueden ser heredadas por clases o por otras interfaces. A esto se le denomina herencia de comportamiento y se especifica mediante el símbolo ":".

- Aunado a ello, el Modelo de Objeto ODMG define una relación EXTENDS para la herencia de estado y comportamiento entre clases. Las clases que extienden a otra clase obtienen acceso a todos los estados y comportamientos de la superclase, incluyendo cualquier cosa que haya adquirido a través de la herencia de otras interfaces o clases.

- **Extents:** corresponde a una colección (extensión) de un determinado tipo de objeto que tiene un nombre e incluye todas las instancias de objetos persistentes creadas a partir de dicho tipo. Esta extensión se puede indexar para que el acceso a su contenido sea más rápido.

b.- Lenguaje de Definición de Objetos (ODL)

Las implementaciones asociadas a un tipo son separadas léxicamente en el ODL, el cual es usado para expresar la estructura y condiciones de integridad sobre el esquema de la BD. ODMG 3.0 distingue tres modos de definición de

tipos: Interfaces, Clases y Literales. Al especificar el diseño de las clases en ODL se deben describir las características del Modelo de Objetos, a saber: atributos, Relaciones y Métodos

Castell (1994) plantea que los principios básicos de ODL son:

- Soporta la semántica completa del modelo de objetos de ODMG.
- Independiente del lenguaje de programación.
- Compatible con el lenguaje de programación.
- Extensible (nuevas funcionalidades, optimizaciones físicas)

c.- Lenguaje de Consultas de Objetos (OQL)

Es un lenguaje de especificación de consultas sobre un esquema de BD definido mediante ODL. OQL, es un lenguaje declarativo del tipo Structured Query Language (SQL) utilizado en las BDR, que permite realizar consultas de modo eficiente sobre BDOO, incluyendo primitivas de alto nivel para conjuntos de objetos y estructuras. La sintaxis básica es una estructura *SELECT...FROM...WHERE...*

Aunado a ello, presenta las siguientes características:

- Es un Lenguaje Funcional, donde los operadores pueden ser compuestos libremente.
- Se basa en el Modelo de Objetos de ODMG.
- No proporciona operadores de actualización explícitos.
- Proporciona primitivas de alto nivel para trabajar con herramientas específicas.

- Obtención como resultado un elemento, además de colecciones de elementos.

- Uso de operadores de colecciones: funciones de agregados (max, min, count, sum, avg) y cuantificadores (for all, exists)

- Uso de group by.

d.- Conexión con los lenguajes C++ , Java Binding y Smalltalk.

ODMG 3.0 provee conexiones con distintos lenguajes como lo son C++ , Java Binding y Smalltalk. Esta interfaz ("bindings", enlazamientos), especifica como se debe hacer la programación de una aplicación sobre una BD que ha sido previamente declarada en ODL, en función del lenguaje con el que se trabaje.

8.2 Estándar Java Data Objects (JDO)

Es un estándar Java para gestionar la persistencia de objetos, hereda el trabajo de ODMG 3.0 para modificarlo y adaptarlo a las características peculiares de Java, parte de SUN Microsystems, y pretende ser una especificación de objetos de datos independiente del medio de persistencia. El Modelo de Objetos es básicamente el modelo de objetos de Java, incluyendo todos los tipos básicos, referencias, colecciones e interfaces.

8.2.1.- Características

Jordan y Russel (SF), describieron que JDO posee las siguientes características:

- Especifica los contratos entre sus clases persistentes y su entorno de ejecución.

- Soporta el Modelo de objetos, incluyendo referencias, colecciones, interfaces, herencia, entre otros.

- La persistencia es totalmente transparente: esto permite hacer el objeto comercial totalmente independiente de cualquier tecnología de la BD.

- Reduce el ciclo de desarrollo.
- Persistencia Universal.
- JDO es una solución sencilla para sistemas de información grandes.
- Modelo Transaccional robusto.

8.2.2.-Componentes

La arquitectura de JDO está constituida por un conjunto de paquetes los cuales poseen cada uno características en particular, a continuación se presentan los paquetes JDO y su respectiva utilidad. (ver tabla 2)

Tabla 2.- Paquetes de JDO

PersistentCapable	Se implementa esta interfaz en las clases que pueden tener instancias persistentes, la cual manipula el ciclo de vida del objeto.
PersistenceManager	Representa conexiones a la fuente de los datos, para realizar consultas sobre los objetos almacenados. Una aplicación puede abrir uno o más PersistenceManagers
PersistenceManagerFactory	Permite recibir nuevos instancias de PersistenceManager de las fuentes de los datos, es decir, la creación de las instancias de PersistenceManager. Se utiliza para obtener referencias válidas a objetos del tipo PersistenceManager.
Transaction	Permite delimitar las transacciones
Query	utilizando lenguaje de consulta JDO, se puede de manera explicita y declarativa obtener objetos de la fuente de datos, además los objetos también pueden ser obtenidos de manera implícita y transparente de la fuente de datos utilizando la navegación básica
InstanceCallback	Define algunos mecanismos que permiten realizar “cosas especiales” (como inicializar atributos transitorios) durante las operaciones de la BD (como antes / después de lectura, antes / después de escritura).
JDOException	Las excepciones que se levantan durante las operaciones de JDO

Para alcanzar la persistencia completamente JDO define un mecanismo llamado "*Enhancement*". La idea es quitar cualquier código dependiente de la BD explícita de clases del negocio. El mapeo con fuentes de datos existentes o nuevas entonces se define usando archivos externos XML. El reforzador de JDO toma los archivos Java compilados (.class) y aplica las reglas de la persistencia según lo definido en el archivo metadata, como se observa en la Figura 1.

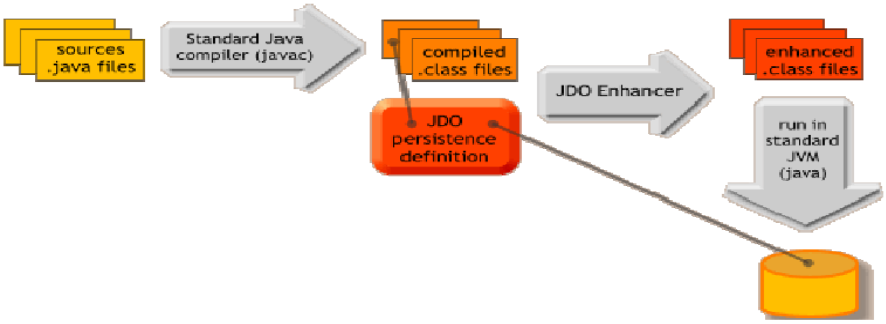


Figura 1. Mecanismo Enhancement para alcanzar la persistencia.

8.3.- Comparación entre ODMG 3.0 y JDO

En la tabla 3 se puede apreciar un cuadro comparativo entre los estándares ODMG 3.0 y JDO, en función del modelo de objetos, ciclo de vida, Bases de Datos y Transacciones, además de las consultas.

Tabla 3. Comparación entre los estándares ODMG 3.0 y JDO (Hernández&Montilla,2005)

ODMG 3.0	JDO
Modelo de Objeto	
Persistencia transparente para el usuario	Persistencia transparente para el usuario
Persistencia por herencia	Persistencia por herencia
Posee varios lenguajes para el binding que se integran al modelo objeto de ODMG, tal como; Java, C++ y Smalltalk	Integración con el modelo objeto de Java
Se proporcionan interfaces específicas Provee cinco colecciones básicas	Proporciona la interfaz PersistenceCapable Provee el Collection factory y las clases de objetos
La identidad de objeto del sistema es manejada por el sistema manejador de objetos	Provee tres modelos para la identidad del objeto, los cuales son: Application Identity, data store identity y Non-data store identity
Ciclo de Vida	
Los objetos viven hasta que la transacción termina	Los objetos viven mientras son instanciados con el PersistenceManager
BD y Transacciones	
Fácil de usar, maneja tanto la BD como las transacciones	BD, conexiones y transacciones son manejadas por el API de Java
No define transacciones distribuidas	Soporta transacciones distribuidas
Especifica extensiones para el manejo de ambientes robustos (EJB)	Soporta el manejo de ambientes robustos (EJB)
Permite búsquedas explícitas de objetos	No permite búsquedas explícitas
Consultas (Query)	
Extensión del OQL	Consultas basados sobre Java -- JDOQL
Expresiones complejas de búsqueda	Secuencias simples de filtrado
Los resultados pueden ser composiciones, proyecciones o apenas un número entero	Los resultados siempre son colecciones de objetos capaces de ser persistentes
El nombre de la clase es parte de secuencia de la búsqueda	Explícitamente los clases de objetos, son usados como parámetros

9.- LOS SISTEMAS MANEJADORES DE BASES DE DATOS ORIENTADAS A OBJETOS

Un SMBDOO es la integración de la tecnología de objetos con las capacidades de las BD, almacena objetos incorporando y empleando todas las ventajas de la OO, lo que refiere a que se pueden tratar directamente los objetos sin tener que hacer la traducción a tablas o registros, los objetos se conservan y pueden ser

gestionados aunque su tamaño sea muy grande, compartidos entre múltiples usuarios, además de mantener tanto su integridad como sus relaciones.

9.1 FastObjects

Un SMBDOO comercial que permite almacenar los objetos que han sido creados usando tecnología de C++ o de Java directamente en una BDOO. La BD de FastObjects reconoce las relaciones entre los objetos y reproduce estas relaciones cuando los objetos se recuperan de la BD. Un servidor de BD multiusuario para aplicaciones de múltiples capas escritas en Java o en C++. Los componentes principales son: **SDK FastObjects, Archivo de configuración (Ptj.opt), la cache activa, Enhancer FastObjects, Bases de Datos FastObjects** (objects.dat y objects.idx**) y el diccionario.**

9.1.1.- Arquitectura

Consiste en un sistema servidor de BD el cual proporciona la funcionalidad esencial para poner a la disposición una biblioteca de archivos. El servidor de FastObjects recibe y mantiene conexiones del cliente sobre una colección de sockets de TCP/IP. El servidor maneja todas las transacciones que funcionan con la BD proporcionando control de concurrencia del objeto. Los objetos de Java o de C++ se almacenan en el sistema de archivo local del servidor de FastObjects.

9.1.2.- Manejo de memoria

Lo hace a través de los siguientes mecanismos:

Transmisión Diferida de los Objetos y Sub-Objetos

Para minimizar el tráfico en las redes de datos, aumentar los tiempos de respuesta y disminuir el consumo de memoria, el servidor de BD FastObjects difiere la transmisión de los objetos hasta que sus atributos son accedidos o solicitados explícitamente por la aplicación. Un sub-objeto es aquel que esta asociado con un objeto base, ya que, forma parte de uno de sus atributos.

Si el acceso no es diferido, entonces toda la línea ancestral es transmitida desde el servidor cuando el cliente solicite un objeto. El problema de esta forma de acceder a los objetos es que cada vez que se accede a un sub-objeto del objeto solicitado esto resulta en una solicitud adicional al servidor para tener acceso a los datos de este sub-objeto.

Patrones de Acceso

Permiten decirle al manejador de BD de FastObjects que cuando se solicite un determinado tipo de objeto cuáles objetos asociados a éste van a ser necesitados, entonces el servidor puede transferir esos objetos específicos en conjunto con el objeto solicitado y de esta forma lograr se busca reducir el tiempo de acceso, y de esta forma tener acceso a cada uno de los objetos asociados durante su uso en la aplicación de forma rápida y eficiente.

9.1.3.-Índices

Una clase debe tener por lo menos un índice basado en los identificadores asociados internamente a los objetos. Este índice establece el orden de los objetos cuando se recobran desde la colección de los mismos. FastObjects permite definir índices secundarios para la amplificación de la clase, provee métodos para establecer y asignar que índice será usado. Estos índices deben expresarse en el archivo de configuración, asociados con la extensión de la clase a la cual se refieren, y pueden contener más de un campo como clave.

9.1.4.-Diccionario

Es una BD especializada para mantener la descripción de las clases que son instanciadas para obtener los objetos que se encuentran almacenados en la BDOO, estas descripciones son referenciadas como el esquema de clases que conforma la estructura del diccionario. Fastobjects utiliza el diccionario para determinar cómo los objetos son almacenados y recuperados. Por ello, para cada clase almacenada en el esquema de la BD, el diccionario contiene el nombre de la

clase, el identificador, la descripción de los elementos y de los índices definidos, además de las relaciones entre las clases.

9.1.5.-Seguridad

FastObjects provee un sofisticado sistema de autorización que permite definir que usuarios específicos o grupos de usuarios (roles) pueden efectuar operaciones con los objetos de la BD. Cada rol o usuario se le puede ser individualmente autorizado el derecho a leer, escribir o borrar objetos en la BD en cualquier combinación. Con la autorización de usuarios de FastObjects se puede afinar aún más la seguridad de la BD concediendo el acceso de roles o usuarios a tipos específicos de clases o inclusive a instancias de clase con valores específicos de datos.

9.1.6.-Integridad

FastObjects apoya totalmente todos los rasgos de integridad de datos tradicionales como transacciones, logging y bloqueos. Las transacciones aseguran que la ejecución de una operación sea atómica, debido a que la transacción que es escrita se almacena establemente. Cuando la transacción es realizada completamente es luego archivada mediante archivo log, luego ocurre un cambio en la transacción primero se escribe en archivo log y después en un archivo de la BD. Si el logging falla, es que no hay nada escrito en la BD, si la BD al escribir, falla debido a un evento extraordinario, como una caída del sistema, FastObjects repite el escrito desde el archivo log. Además, cada objeto escrito en la BD del FastObjects está codificado con un checksum para protegerlos de las fallas. En caso de que un bloque de disco falla, cualquier objeto afectado por algún fracaso es capturado por sus checksum garantizando que la aplicación nunca utilice esos datos defectuosos.

9.1.7.-Mecanismos de control de concurrencia

Fastobjects está diseñado para el acceso concurrente, controlado por el servidor de la BD, garantizando así la consistencia del objeto, establece cuatro

formas de bloquear, en cuanto a la lectura (read), escritura (write), actualización (upgrade) y borrado (delete) de los objetos.

En FastObject todas las operaciones de BD deben realizarse dentro de una transacción las cuales pueden poseer checkpoints que almacenan las modificaciones realizadas en los objetos hasta ese momento, aunque manteniendo sus bloqueos.

9.1.8.-Mecanismos de respaldo y recuperación

FastObject proporciona una opción de configuración para usar archivos de recuperación de forma que si se produce un fallo mientras se actualiza la BD se puede usar este archivo para reescribir la actualización. Además, proporciona funciones de administración que incluyen lo que FastObject llama 'External Backups'. Esto permite a un cliente suspender la BD con el fin de hacer una copia sin afectar a otros clientes que estén usándola. Los cambios hechos por los otros clientes son almacenados hasta que la BD esté de nuevo en línea. Adicionalmente provee estrategias de replicación a través de las cuales las bases de datos replicadas tienen sus cambios escritos al servidor primario y secundario de la BD usando un agente de replicación en red.

9.2 DB4Objects

Db4Objects (Db4o) es un nombre que proviene de la expresión Database for objects, que significa BD para objetos, se trata de un framework de persistencia que permite almacenar objetos Java o .NET de forma directa y transparente. Los principales componentes son:**El motor db4(archivo .jar), el API, ObjectContainer y Object Manager**.

9.2.1.- Arquitectura

Está bajo el modo cliente/servidor. El API de Db4o incluye la funcionalidad necesaria para que Db4o pueda ejecutarse como un servidor y permitir la definición de clientes que interactúen con el servidor. En un ambiente distribuido

se puede implementar esta modalidad para aceptar conexiones desde otros computadores o desde PDAs, dispositivos o teléfonos celulares.

9.2.2.- Manejo de memoria

La responsabilidad de limpiar la memoria es cedida a la máquina virtual. En Java se utilizan referencias débiles para las instancias de objetos. Si un objeto no es referenciado por la aplicación durante un largo tiempo, el Garbage Colletor de la Máquina Virtual limpiará éstas referencias débiles. Con respecto a los objetos persistentes que no han sido referenciados, el Garbage Collection no tiene influencia sobre ellos, éstos continuarán siendo almacenados en la BD, incluso si no son referenciados por un largo tiempo.

9.2.3.-Índices

Db4o permite indexar objetos para maximizar el rendimiento de las consultas. A través de los objetos del API se pueden crear índices asociados a determinadas clases. Para crear o eliminar índices en clases que poseen una gran cantidad de objetos se proponen dos estrategias:

- Importar todos los objetos con la propiedad “indexing” en off, configurar el índice y reabrir el ObjectContainer/ObjectServer.

- Importar todos los objetos con “indexing” en on y efectuar commit regularmente cada cierta cantidad de objetos.

Db4O permite definir índices de clase, de atributos y de colecciones.

9.2.4.-Diccionario

Db4o no posee un diccionario como tal, para mantener la información de las clases que conforman el esquema de la BD utiliza una clase especial llamada metadata, la cual contiene de cada una de las clases, el nombre, atributos, métodos, antecesores, y las relaciones

9.2.5.- Seguridad

Db4o maneja la seguridad mediante protección por password y mecanismos de encriptación, como soluciones para una alta seguridad, permitiendo a cualquier usuario escoger su propio mecanismo de encriptación. Es importante destacar que no proveen seguridad en la comunicación entre cliente y servidor.

9.2.6.- Integridad

Db4O garantiza integridad implícita, al manejar las identidades de los objetos, para lo que se debe mantener el ObjectContainer en ejecución para garantizarla. Si se cerrara este y los objetos de manipulan en memoria, se corre el riesgo de duplicidades al guardar objetos en Db4o

9.2.7.- Mecanismos de control de concurrencia

Db4o trabaja bajo un esquema de manejo de concurrencia optimista, en el cual se hace bloqueo de los objetos para la escritura y lectura, pero no se hace un chequeo de las colisiones al momento de acceder concurrentemente a los objetos. Este manejador le deja la responsabilidad de crear un sistema de control de concurrencia al programador del sistema.

9.2.8.- Mecanismos de respaldo y recuperación

El Object Container de Db4o usa una memoria caché de objetos donde se guardan los objetos y las transacciones que se están usando y todas las operaciones son realizadas allí en la caché y sólo son escritos a la BD en el momento en que se cierra la conexión. Si ocurre un error de sistema no se ve comprometida la BD sino sólo los objetos que se encuentran en la memoria caché. Además provee mecanismo de rollback para deshacer las transacciones que no terminan con éxito y mecanismos para sincronizar los datos de la caché con los de la BD. Db4o también provee como mecanismo de recuperación la opción de crear respaldos (backups) de la BD está activa y existen aplicaciones corriendo sobre ella.

10.- CONCLUSIÓN

Las BDOO tienen la capacidad de cubrir las necesidades de datos de aplicaciones donde la tecnología relacional comienza a tener problemas de desempeño, escalabilidad, flexibilidad y/o complejidad de mantenimiento. La importancia de las BDOO radica en el manejo de objetos complejos y a la persistencia de ellos. Por otro lado, los estándares de persistencia ODMG y JDO han repercutido de manera importante en el desarrollo y auge de los SMBDOO, mancando importante influencia. Para finalizar, es importante destacar que aún existe inmadurez en el mercado para la adopción de esta tecnología OO, por lo que debe analizarse con detalle la presencia del proveedor para adoptar su producto en una línea de producción sustantiva.

11.- REFERENCIAS

- Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D., Zdonik, S. (1989). The Object-Oriented Database System Manifesto. Actas del 1st International Conference on Deductive and Object-Oriented Databases (DOOD). Japón

- Stonebraker, M. (1990). The Third-Generation Database Manifiesto: A Brief Retrospection.

- Darwen, H., Date, C. (1995). The Third Manifesto.

- Bertino, E. y Martino, L. (1995) "Sistemas de Bases de Datos Orientadas a Objetos", Adisson Wesley.

- Catell, R. (1994). The Object Database Standard**.** Disponible en http://kybele.escet.urjc.es/documentos/BDA/Parte2/OdmgV3.PDF

- Hernández, Y., Montilla J (2005). Estudio comparativo del manejo de objetos multimedia en los sistemas manejadores de bases de datos orientado a objetos (SMBDOO) FastObjects y Jasmine II. Caracas: Universidad Central de Venezuela, Facultad de Ciencias, Escuela de Computación.

- Jordan, D. & Russell, C. (S/F). Java Data Objects. Disponible en: http://www.aqs.es/web/files/JavaDataObjects_cap1.pdf

- Marín,N.(2001). Tesis Doctoral "Estudio de la Vaguedad en los Sistemas de Bases de Datos Orientados a Objetos: Tipos Difusos y sus Aplicaciones". Universidad de Granada.

- Martínez, A.(S/F). Introducción a los Sistemas de Gestión de Bases de Datos Orientadas a Objetos. Universidad de Oviedo, España.

- FastObjects. http://www.versant.com/products/fastobjects/t7

- ODMG. http://www.odmg.org/
- JDO. http://java.sun.com/products/jdo/
- Db4Objects. http://www.db4o.com/