14
2E.T.

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

"BASES DE DATOS Y ORIENTACION A OBJETOS"

T E S I S

Que para obtener el Título de

A C T U A R I O

p r e s e n t a

JAIME ANDRES BLANCO CACIQUE

México, D. F. Agosto de 1995

VNIVERSIDAD NACIONAL
AVFNºMA DE
MEXICO

M. en C. Virginia Abrín Batule
Jefe de la División de Estudios Profesionales de la
Facultad de Ciencias
P r e s e n t e

Comunicamos a usted que hemos revisado el trabajo de Tesis:

"BASES DE DATOS Y ORIENTACION A OBJETOS"

realizado por JAIME ANDRES BLANCO CACIQUE

con número de cuenta 8152800-6 , pasante de la carrera de ACTUARIA

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis
Propietario M.en C. MARIA GUADALUPE IBARGUENGOITIA GONZALEZ

Propietario MAT. ANA LUISA SOLIS GONZALEZ COSIO

Propietario ACT. JOSE CARLOS MALDONADO BAEZ

Suplente M.EN C. GUSTAVO MARQUEZ FLORES

Suplente FIS. JUAN JESUS GUTIERREZ GARCIA

Consejo Departamental de Matemáticas
MAT. CESAR GUEVARA BRAVO

A MIS PADRES :

Guadalupe Cacique Parra
Mario Blanco Iglesias

Quienes gracias a su esfuerzo, sacrificio y amor debo todo lo que soy.

A MIS HERMANOS :

Maria Victoria
Mario
Rosalinda
Raymundo Arturo
Ricardo Daniel
Guadalupe

Por su ejemplo y el apoyo que me han brindado.

A:

Familia Hamue
Carmen Camacho Q.
M. Patricia Marquez O.
I. Rodolfo Ozuna C.
J. Carlos Maldonado B.

Por la confianza que me han brindado, y el significado que tienen para mi.

A mis amigos

De los que aprendí como ser, y como no ser.

AGRADECIMIENTOS

A

M. en C. M. Guadalupe Ibargüengoitia Gonzalez

Por el tiempo invertido y paciencia en la conclusión del presente trabajo.

A

Mat. Ana Luisa Solis Gonzalez Cosio
Act. Carlos Maldonado Baez
M. en C. Gustavo Marquez Flores
Fis. Juan Jesus Gutierrez García

Por sus comentarios y tiempo dedicado en la revision del presente trabajo.

BASES DE DATOS Y ORIENTACION A OBJETOS

INTRODUCCION

En el ambiente de la informática, existen innovaciones tecnológicas. Algunas son modas, otras trascienden haciendo aportaciones importantes. Un concepto que ha trascendido es el relacionado al Modelo de Datos. Actualmente, se pueden distinguir 4 modelos de datos : Modelo Relacional, El Modelo Relacional Extendido, El Modelo Funcional, y el Modelo Orientado a Objetos.

El Modelo Relacional está basado en el álgebra relacional. Este es el modelo que predomina actualmente y está basado en el manejo de tablas o tuplas de datos; así como en la normalización de la información relacionada a la aplicación a desarrollar.

El Modelo Relacional Extendido. Dado que el modelo relacional presenta algunas debilidades para ciertas aplicaciones; la forma más sencilla de solucionarlo es extendiendo el modelo. Este modelo perfecciona el modelo relacional incorporando procedimientos, objetos, versiones, etc. No hay un solo modelo relacional extendido; existen varios, dependiendo del grado de perfeccionamiento que se desee lograr.

El Modelo Funcional. Es una forma elegante de representar a las bases de datos. Estos usan un lenguaje de acceso a datos basados en funciones matemáticas, elaborando consultas declarativas. La base del modelo funcional son los

objetos y funciones. Las funciones incorporan objetos sobre objetos y sobre valores. Las relaciones entre objetos, atributos de objetos y procedimientos asociados a objetos, son representados por funciones.

El Modelo Orientado a Objetos. Este modelo es creado a partir del paradigma de los objetos en los lenguajes de programación orientados a objetos. Los lenguajes precursores son SIMULA y SMALLTALK. No existe sólo un modelo orientado a objetos, existen varios. Sin embargo, todos ellos comparten muchas características.

Durante el presente trabajo me centraré en el modelo Orientado a Objetos y la relacion existente con los manejadores de bases de datos.

Para ello es necesario, entender los conceptos relacionados con la Orientación a Objetos, sus bases, componentes y características; de esto se hace mención durante el primer Capítulo de este trabajo.

Por otro lado, al hablar de Bases de Datos, habrá que especificar el contexto bajo el cual se tratarán las bases de datos. Esto es, explicar la diferencia entre una Base de Datos (Banco de Información) y un Manejador de Base de Datos. Abordando características comunes a los manejadores de Bases de Datos, como el modelo, integridad, acceso, diccionario de datos; y demás temas propios de los manejadores de bases de datos. Todas las características de los manejadores de bases de datos son explicadas en el Capítulo II.

Conociendo los conceptos Orientados a Objetos y lo que se entiende por Manejador de Base de Datos, se tienen los elementos suficientes como para analizar los Manejadores de Bases de Datos Orientados a Objetos, sus características y

acuerdos relacionados a su formalización. Esto se muestra durante el Capítulo III.

Ahora bien, todo Sistema Manejador de Bases de Datos tiene una base científica. En el caso de los Manejadores de Bases de Datos Orientados a Objetos las bases se encuentran en los principios Ontológicos. En el Capítulo IV se mencionan estos principios y la manera en que son formalizados.

Por último, en el Capítulo V se hace mención de la interfase entre el lenguaje Orientado a Objetos y el Manejador de Base de Datos Orientado a Objetos.

CAPITULO I

EL CONCEPTO ORIENTACIÓN A OBJETOS

Últimamente se ha mencionado mucho acerca de la orientación a objetos, algunas casas de software emplean este término, o términos relacionados a la orientación a objetos para resaltar las cualidades, características y bondades de los productos que comercializan, tales como Cliente-Servidor, Reusabilidad, Instanciación, Concurrencia, etc.

Los sistemas orientados a objetos tienen sus raíces en los lenguajes de programación, particularmente la aparición de la noción de un objeto como pieza de programación surgió en SIMULA, un lenguaje de programación para realizar simulaciones de fenómenos o determinados hechos. Más adelante se descubrió que los objetos de software podían ser útiles no sólo en simulaciones de programación, sino también para la creación de prototipos y el desarrollo de aplicaciones; por lo que se inició todo un análisis sobre el comportamiento de los objetos, estableciéndose preguntas tales como: ¿Todo es un objeto?, ¿Son las clases de objeto, un objeto en sí mismo?, ¿Existe alguna diferencia entre "Objeto del Usuario" y "Objeto del Sistema"; más aún, entre un "Objeto Activo" y un "Objeto Pasivo"?. Pero, ¿Qué significan todos estos términos?, ¿Qué es en realidad la Orientación a Objetos?. Tomaré como base una pregunta sencilla sobre la cual iré particularizando: ¿Qué es un objeto?. La respuesta a esta pregunta, depende del ambiente sobre el cual se está hablando (puesto que la pregunta es tan ambigua como lo es: ¿Quién es el usuario final?).

Si nos referimos a lenguajes de programación, los objetos estarán determinados por algoritmos, rutinas, subrutinas, procedimientos, programas, etc. los cuales serán sometidos a un proceso que los integre o transforme.

En cambio si hablamos de aplicaciones (en el desarrollo de sistemas) los objetos estarán definidos por información del medio ambiente del hombre. Por ejemplo: en un sistema de recursos humanos, los objetos serán el conjunto de las personas; y los procesos estarán dados por transformaciones que sufran los objetos (en este caso las personas) como la nómina, el cierre de año fiscal, etc. También puede darse el ejemplo cuando hablamos de bases de datos, lo cual se mencionará más adelante.

I.A. CARACTERÍSTICAS DE LOS OBJETOS.

Estamos acostumbrados a manejar los objetos de la vida real, que damos por hecho muchas características relevantes, a continuación mencionaré algunas de ellas y su equivalencia a nivel computación.

I.A.1. LEY DE IMPENETRABILIDAD.

"Dos cuerpos no pueden ocupar el mismo espacio al mismo tiempo". Sin embargo, en una computadora es posible, por ejemplo: la redundancia de información. Si el fin de un sistema es efectuar una abstracción de la realidad, lo ideal sería que no sucediera a nivel computación; el concepto Orientación a Objetos plantea este problema como identidad de los objetos.

I.A.2. ENCAPSULACIÓN.

Todo objeto cuenta con un interior y un exterior. El interior no es accesible; es decir, no tenemos acceso a ver, analizar o modificar el interior de los objetos. Dentro del concepto Orientación a Objetos esto se plantea como encapsulación.

I.A.3. HERENCIA.

"La materia no se crea ni se destruye, sólo se transforma". Este principio es fundamental. Al transladarlo a la Orientación a Objetos; ejemplo: "La madera se quema", y si una silla esta hecha de madera, podemos concluir que la silla también se quema. Esto es, la silla heredó ciertas propiedades de los elementos que la forman, por lo que tendremos que agregar la herencia de propiedades a la Orientación a Objetos. Por otro lado, retomando el caso de la silla, la madera no ha cambiado de estado sin embargo su uso exterior ha cambiado, lo que implica que pertenece a otra clase de objetos.

I.A.4. REUSABILIDAD.

Otro concepto tomado de este principio es el de Reusabilidad, por ejemplo: cuando la silla ya no es útil, con dos sillas podré crear una sola, y la madera restante usarla como leña. La *reusabilidad* es la característica de los objetos al poder ser empleados cuantas veces sea necesario, independientemente del lugar o momento en que sean referenciados. Ejemplo: en el caso de los datos, desempacar y empacar los datos numéricos. Si se trata de procesos, el calcular el número de días entre dos fechas, sin importar de que proceso es mandado llamar.

I.A.5. VARIABLES DE INSTANCIA.

Las variables de instancia o características son propias de un objeto (es decir, están en el interior de un objeto); sin embargo pueden ser comunes a una clase de objetos. Las variables de instancia sólo pueden ser modificadas a través de los métodos.

I.A.6. LOS MÉTODOS.

Los *métodos* son las funciones que definirán el comportamiento al exterior (Mensajes) o alterarán las variables de instancia (es decir, modifican el estado del objeto). Al igual que las variables de instancia, los métodos son propios del objeto o pueden ser comunes a una clase de objetos.

I.A.7. MENSAJES.

La comunicación entre objetos se da a través de mensajes. En otras palabras "Los objetos reaccionan ante estímulos externos"; aquí los *estímulos externos* son los mensajes y la reacción de los objetos son los métodos (funciones).

I.A.8. COMPORTAMIENTO.

El *comportamiento* es el conjunto de métodos de los objetos. El comportamiento varía de acuerdo al mensaje recibido y de la clase de objeto de la cual proviene el mensaje.

I.A.9. CLASES.

Las *clases* de objetos son el resultado de agrupar características y atributos comunes a determinado conjunto de objetos. En sí, las clases de objetos nos permiten identificar el comportamiento (métodos o funciones) y la forma (atributos) de un determinado conjunto de objetos.

Dependiendo del conjunto de objetos que se desee clasificar, una clase puede ser muy general o muy especializada. Un ejemplo de una clase muy general es "La Madera"; dentro de ésta clase se encuentran los elementos "Silla de Madera", "Casa de Madera", "Leña". Una forma de definir clases, es estableciendo una jerarquía entre los objetos que integran una clase muy general. En nuestro ejemplo, al establecer una jerarquía surge una subclase que son "Los Muebles de Madera"; y de los muebles de madera surge la clase de "Las Sillas". Puede establecerse una jerarquía de manera ascendente, resultando que "Las Sillas de Madera" es una subclase de "Los Muebles", y a su vez los muebles es una subclase de "La Madera".

La *herencia* es el conjunto de características o propiedades que perduran al realizarse operaciones sobre clases. Si un niño de primaria pregunta: "¿Que es una Universidad?", una respuesta podría ser "Es una escuela donde se estudian materias de una ciencia en particular". Al responderle que "Es una escuela", el niño ha heredado el concepto que él tiene de escuela; es decir, que una escuela tiene salones de clase, que van estudiantes y profesores, que hay un pizarrón, etc.

Ahora bien, todos estos principios y características de la vida real, tendremos que tomarlos, analizarlos y conceptualizarlos de manera tal que puedan ser implementados en una computadora.

Existen muchas características por analizar de los objetos del mundo real, y se puede profundizar al analizar cada una de ellas. A la aplicación de estos conceptos de los objetos reales al análisis, diseño, desarrollo y mantenimiento de software se le conoce como el paradigma de la Orientación a Objetos.

I.B COMPONENTES DE LA ORIENTACIÓN A OBJETOS

El hombre siempre ha hecho uso de los objetos reales. Dentro de los sistemas, lo ideal sería que manipulase objetos. La *orientación a objetos* se basa en el paradigma de los objetos para identificar los componentes de éstos, y así implementarlos en el ambiente de las computadoras.

I.B.1. APORTACIÓN DEL PARADIGMA DE OBJETOS.

Cuando tratamos con la arquitectura de un sistema, el diseñador de software se confronta con una elección fundamental: ¿La estructura debe estar basada en los procesos o en los datos?. Dado que al final, los procesos y los datos juegan parte importante en la estructura de un programa. La estructura de los sistemas que actualmente se encuentran en producción, está basada fundamentalmente en las limitaciones o bondades de los lenguajes de programación en los cuales fueron desarrollados dichos sistemas sin tomar en cuenta los procesos ni los datos.

En general, sin particularizar en orientación a objetos, un sistema de software es un conjunto de mecanismos para ejecutar ciertas acciones sobre ciertos datos. El desarrollo de un sistema se realiza identificando los objetos que lo

conforman y las interrelaciones que existen entre ellos para cumplir uno o varios fines específicos.

Bajo éste enfoque, el diseñador de software debe centrarse en el comportamiento de los objetos y cómo interactúan entre sí. De ahí la importancia del estudio de los objetos.

1. Los *objetos* tienen una parte interna que no es visible sino a ellos mismos. Esto es, Poseen un interior que no es accesible por el medio externo. La aportación principal es el concepto de variable de instancia (bajo el esquema anterior: Datos) ocultas a otros objetos y sólo visibles para el objeto al que pertenecen.

2. Los *objetos* tienen un comportamiento exterior que los identifica y los distingue de los demás. Cada objeto se comporta de manera distinta, dependiendo de la clase a la que pertenezca. Esto aporta los métodos (bajo el esquema anterior: Procesos). Estos métodos u operaciones son activados cuando el objeto recibe un mensaje del exterior. En base a los dos enunciados anteriores tenemos lo siguiente: Los objetos poseen variables de instancia y métodos que reaccionan al interactuar con otros objetos por medio de mensajes.

3. Cada objeto particular es un elemento de cuando menos una clase, y de todas las clases que los comprenden. Las clases constituyen las abstracciones genéricas de los objetos. Las clases pueden ser organizadas en forma jerárquica, las clases de mayor jerarquía son las más generales. Una *clase particular* es una subclase de una clase más general. Las clases particulares "heredan" propiedades de clases más generales.

4. Las cosas nuevas las imaginamos y describimos a partir de otras que ya conocemos. La aportación de este enunciado es muy basta ya que abarca desde el concepto de objeto compuesto, hasta el de herencia de clases.

A continuación expondré conceptos de orientación a objetos resultantes de una análisis al último enunciado.

La *instanciación* es la manera en la cual se encuentran disponibles los datos relacionados a un objeto; esto es, que la información relacionada a un objeto (variables de instancia, métodos, atributos, etc.) esté disponible en determinado momento. Una *instancia* de una clase es un objeto con valores y funciones específicas. La instanciación es uno de los mecanismos básicos de reusabilidad. Los objetos pueden ser instanciados estáticamente o dinámicamente. La *instanciación estática* de los objetos es permitida al momento de la compilación y existe durante la ejecución del programa. La *instanciación dinámica* ocurre al momento de ejecución y requiere de un método para desecharla o algún procedimiento de recolección de basura.

La idea de la herencia en las clases es proveer un mecanismo sencillo y poderoso para definir nuevas clases que heredarán propiedades de clases ya existentes. La diferencia entre las distintas formas de herencia podrían mostrarse con las siguientes preguntas:

- ¿La herencia ocurre estáticamente o dinámicamente?
- ¿Quienes son los que reciben las propiedades a heredar (Clases)?.
- ¿Que propiedades pueden ser heredadas?

- ¿Que propiedades heredadas son visibles a quien la vaya a heredar?

- ¿Pueden las propiedades heredadas ser sobrepuestas o suprimidas?

- ¿Como se resuelven los conflictos?

En la herencia simple, una subclase puede heredar variables de instancia y métodos de una sola clase padre, y posiblemente agregando algunos métodos y variables de instancia por sí mismas. Por ejemplo: podemos definir una subclase "Gráfica de un Número Complejo", que heredará de la clase "Número Complejo" el número complejo, y agregará la operación 'graficar'.

La *herencia múltiple* se da cuando una subclase hereda variables y métodos (ambas son propiedades) de varias clases padre. Empleando el mismo ejemplo: la clase "Gráfica de un Número Complejo", heredará de la clase "Número Complejo" el número complejo, y de la clase "Gráfica de Objeto" la gráfica del número.

El concepto de herencia principalmente es estático, ya que cuando se crean las clases, se definen las propiedades que heredará, y esto ocurre en el momento de definirlas y no en el momento de ejecución. Una vez que la clase ha sido definida, las características de sus instancias (variables de instancia y métodos) son determinadas durante todo el tiempo.

Nos referiremos como "herencia dinámica" a los mecanismos que permiten a los objetos alterar su comportamiento en el curso de interacciones normales con otros objetos.

La herencia dinámica ocurre dentro del modelo del objeto. Para mostrar esto, considere un nuevo concepto denominado *subclasificación dinámica*; suponga que tenemos una instancia de un Número Complejo y deseamos graficarlo. Desafortunadamente no tenemos ningún método de *graficación*, lo que en realidad se desea es una instancia de "Gráfica de un Número Complejo". Con la *subclasificación dinámica* se empacaría el "Numero Complejo" dentro de "Gráfica de un Número Complejo" para aplicar el método de *graficación* y después desempacar.

Algunos puntos interesantes acerca de la herencia son los siguientes:

1. No todos los lenguajes con herencia de clases soporta la herencia múltiple (ejemplo Smalltalk). Dado que la herencia múltiple no es frecuentemente requerida, todo depende que tan homogéneo es el modelo.

2. Es importante ser capaz de sobreponer métodos de herencia. Una operación de "desplegar" es específica a un objeto, y debe ser reimplementada o alterada por una subclase que la herede.

3. Las subclases pueden o no permitir el acceso directo a variables de instancia herededas. ¿Debe una subclase, como miembro que heredará de una subclase, permitir mostrar lo qué esta normalmente oculto para las clases padre?. Cuando una subclase agrega un método que accesa variables de instancia heredadas, está violando la encapsulación del padre. Consideraré el ejemplo de "Gráfica de un Número Complejo"; si la operación de desplegar hace uso de las variables de instancia heredadas, entonces seremos capaces de alterar la representación interna de la clase padre "Número Complejo". Por otro lado, si la operación desplegar hace uso de las operaciones heredadas, tales como el valor de X y el valor de Y (el cual es calculado en coordenadas

polares), entonces habrá mayor independencia entre la clase "Número Complejo" y la subclase "Gráfica de un Número Complejo".

4. Cuando existen choques de nombres en la presencia de herencia múltiple. Es decir, si heredamos dos operaciones de 'desplegar': ¿Que método empleamos?. Actualmente, este no es un problema ya que puede ser manejado fácilmente indicando que es lo que ocurrirá cuando la operación 'desplegar' sea invocada. También el sistema provee reglas por omisión para seleccionar un método o combinación de métodos heredados, o si requiere que el programador efectúe explícitamente una elección.

Ahora bien, tomando en cuenta las clases, he hablado de la jerarquía de clases (que genera subclases) y de la herencia. Un tema interesante acerca de éstas dos características es: Si efectuamos un cambio en la jerarquía de clases (es decir, cambiamos la definición o implementación de las clases de un objeto), ¿Cómo afectará esto a la herencia de subclases?. Y es especialmente problemático cuando los elementos existentes de las clases modificadas, deben ser preservados. Ejemplo: La modificación de un sistema existente que soporte el manejo de los "Nuevos Pesos"; los nuevos elementos deben manejar la notación en "Nuevos Pesos", sin embargo, los elementos existentes deben preservar la antigua notación y almacenar el valor en el formato de "Pesos". Este problema es conocido como *evolución* en las bases de datos orientadas a objetos.

Por otro lado, la herencia tiene diferente enfoque en el campo de la representación del conocimiento. Las clases de objetos pueden representar conocimientos o creencias en lugar de datos. Una instancia de una subclase es vista como una especialización de los padres. Por ejemplo: todo lo que

sabemos cierto acerca de los mamíferos, también es cierto para los humanos, pero no al revés.

I.B.2. POLIMORFISMOS.

Una función es *polimorfa* cuando puede ser aplicada uniformemente a una variedad de objetos.

El polimorfismo permite que el código de una operación trabaje sobre diferentes tipos abstractos. Por ejemplo: la misma notación puede ser usada para sumar dos números enteros o dos números punto flotante. De forma similar para programar una función que opere sobre dos números complejos. Es decir, que la misma operación mantenga un comportamiento transparente para diferentes tipos de argumentos.

La herencia de clases está íntimamente relacionada con el polimorfismo. La misma operación que aplica a instancias de una clase padre, también aplica a instancias de sus subclases. Por supuesto que es posible tener polimorfismo, sin herencia de clases. En Unix, por ejemplo, el paradigma de un archivo siempre está presente: Las operaciones *open*, *read*, *write* y *close* se aplican polimorficamente a cualquier objeto. Recordemos que para el sistema operativo Unix todo es visto como un archivo y como tal, todas las operaciones que se aplican a los archivos, son efectuadas independientemente del tipo de archivo que se trate, ya sea terminal, impresora, etc. En cada caso, diferentes métodos son implementados para efectuar éstas operaciones. El polimorfismo perfecciona el concepto de reusabilidad haciendo posible implementar software genérico que trabajará no solamente para un rango de objetos existentes, sino que también para objetos que se sumarán en un futuro.

I.B.3. SOBRECARGA Y LIGADO DINÁMICO.

Para ejemplificarlo, considere la operación de desplegado, la cual toma información y la despliega en la pantalla. Dependiendo del tipo de objeto, usaremos diferentes mecanismos para presentarlo. Si es una fotografía tendrá que aparecer en pantalla; si son datos, tendrán que aparecer en forma de tabla; y si es una gráfica de datos, su representación será gráfica.

Considere ahora el problema de desplegar un objeto de tipo desconocido al momento de la compilación. En una aplicación usando un sistema convencional, tendremos 3 operaciones : despliega-foto, despliega-datos y despliega-gráfica. El programador preguntará por el tipo de objeto para usar la operación de desplegado correspondiente. Esto forza a que el programador tome en cuenta todos los tipos de objetos posibles para codificar la operación adecuada. Un ejemplo es:

```
for x in X do
   Begin
      case of type (x)
         Datos : despliega-datos (x)
         Foto  : despliega-foto (x)
         Grafica : despliega-gráfica (x)
      end
   end
```

En un sistema orientado a objetos, definimos la operación de desplegar a nivel tipo de objeto. Así, la operación de desplegar tiene un sólo nombre y puede ser usado indistintamente en datos, fotos o gráficas. Sin embargo, redefinimos la implementación de la operación para cada uno de los tipos de acuerdo al tipo. Esto resulta en un sólo nombre de operación(desplegar), denotando 3 programas

diferentes(esto es denominado sobrecarga). Para desplegar el conjunto de objetos, simplemente aplicamos la operación desplegado a cada uno de ellos y dejamos que el sistema seleccione la implementación apropiada al momento de ejecución. Esto es :

```
for x in X do despliega (x)
```

Así, obtenemos diferentes ventajas: se implementa en el lenguaje (o herramienta) orientada a objetos el mismo número de programas. Pero el programador de la aplicación no tiene que preocuparse de los 3 diferentes programas. Además, el código es más simple y no hay necesidad de aplicar la instrucción CASE para revisar los tipos. Finalmente, el código es más fácil de darle mantenimiento, ya que cuando un nuevo tipo es agregado, una nueva instancia del tipo es agregada también y el programa de desplegar continuará trabajando sin ninguna modificación.

Con el fin de proporcionar esta nueva funcionalidad, el sistema, no puede ligar nombres de operaciones a programas al momento de compilación, ya que habría que compilar ante cada cambio. En consecuencia, los nombres de operaciones deben ser resueltos al momento de ejecución; de forma tal que se efectúe una revision del tipo de objeto para determinar la operación sobre el tipo. Este concepto es conocido como el ligado dinámico.

Por lo tanto: A la sintaxis y secuencia de argumentos de una función es considerado la sobrecarga. Este concepto también es nombrado *firma del polimorfismo* o *polimorfismo parametrizado*.

El término de ligado dinámico se refiere a la habilidad del sistema de seleccionar la operación más apropiada al objeto en el momento de ejecución [SUBODH 94].

I.B.4 TIPOS DE OBJETOS.

La idea básica es asociar un conjunto de características dentro de algo que conocemos como *tipo de datos*. En los lenguajes de programación, los tipos son clásicamente asociados con identificadores: variables, constantes, procedimientos. Esta información nos da las bases para pensar acerca del tipo de unidades, como expresiones, al momento de compilar. Teniendo tal tipo de información se pueden prevenir errores al momento de ejecución en la cual se vea envuelta la aplicación de una operación sobre un argumento de tipo erróneo (los lenguajes que contienen implementada la revision en el tipo de información se dice que son 'robustos sobre tipos'). Esta información acerca de los tipos, algunas veces es útil para la optimización de código y eficiencia al momento de almacenarla.

El tipo de objeto es superficialmente lo mismo que las clases de objeto, la diferencia es que cuando manipulamos tipos de objeto, nos gustaría verificar si lo estamos realizando de una forma consistente; desde otro punto de vista: cuando un objeto recibe un mensaje, debería existir algo que revise si el mensaje en realidad pertenece a ése objeto, y si el mensaje llega en el momento adecuado; esta es la revision estática y está relacionada con el tipo de objeto y no con la clase de objeto.

I.B.5. CONCURRENCIA.

Existen dos formas en las cuales los lenguajes de programación, han trabajado con concurrencia:

La primera de ellas en la cual las entidades activas (procesos) se comunican indirectamente a través de objetos

compartidos pasivos. Es natural el uso de memoria como un conjunto de objetos pasivos, y ver a los procesos como una clase especial de objetos activos. En este caso, es necesario que las acciones sobre el objeto pasivo sean ejecutadas de acuerdo a su interfaz, en otras palabras, contar con un método de acceso a objetos pasivos; es decir, debemos tener un mecanismo para que el objeto activo pueda sincronizar su acceso al objeto compartido. Esto se puede realizar de diferentes formas: el uso de semáforos, o bloqueo de acceso a objetos, o empleando un monitoreo sobre el objeto pasivo.

La segunda forma de trabajo con concurrencia es cuando las entidades activas se comunican directamente una con la otra empleando mensajes. Aquí, cualquier objeto se convierte en activo en respuesta a una comunicación. El canal de control (thread) es determinado implícitamente por el paso de mensajes dondequiera que se localice un objeto 'Proceso'. Esto requiere de una sincronización explícita, ya que el paso de mensajes contempla la comunicación y la sincronización.

Cabe hacer mención que se está hablando de concurrencia en términos de objetos, no en el sentido de bases de datos.

I.B.6. IDENTIDAD.

Los modelos de datos orientados a objetos son también caracterizados por la habilidad de hacer referencias mediante el uso de la identidad de un objeto. Esta característica requiere que exista algo acerca del objeto que permanezca invariable sobre todas las posibles modificaciones del objeto.

La mayoría de los lenguajes modernos orientados a objetos contemplan la identidad del objeto, tendiendo a basarse en el valor de determinado campo para definir la

identidad. Esto es, un objeto es identificado por un subconjunto de sus atributos, lo que llamamos llave. La identidad de un objeto debe ser inmutable. No hay operación que pueda cambiar la correspondencia entre un objeto y su identidad. La identidad persistirá hasta que un objeto sea destruido. En el caso de la orientación a objetos, si borramos un objeto *x*, pueden existir otros objetos que hayan almacenado la identidad de *x*. El borrado puede acarrear referencias que queden en el aire, o producir algunas piezas de almacenamiento indefinidas. Semánticamente, la operación de borrado aún es cuestionable en el ambiente orientación a objetos (basado en el principio de la materia no se crea ni se destruye, sólo se transforma). Por lo que es preferible adoptar el término recolección de basura, en el cual los objetos no pueden ser borrados, sino que los objetos referidos son desechados. Cuando todas las referencias a un objeto han sido borradas, el sistema reclama el espacio de almacenamiento ocupado por ése objeto.

I.B.7. COMPLETEZ COMPUTACIONAL.

Esta característica exige que todas las operaciones aplicables sobre los objetos, no se realicen sino a través del lenguaje explícitamente especificado para ello y no haciendo uso de otras herramientas de menor nivel.

Otra forma de expresarlo es la siguiente: el lenguaje de programación debe ser capaz de tener acceso a todos los recursos del sistema dentro del mismo lenguaje.

I.C REGLAS PARA DEFINIR UN SISTEMA ORIENTADO A OBJETOS.

Ahora bien, para que un sistema sea considerado Orientado a Objetos, será necesario que cumpla con un conjunto de características, las fundamentales son:

1. Objetos.
2. Identidad de los objetos.
3. Encapsulamiento.
4. Organización en tipos o clases.
5. Herencia.
6. Polimorfismo.

Estas se consideran las características obligatorias para que un sistema Orientado a Objetos sea considerado como tal. Además de las anteriores, existen otras características deseables. Esto es, no son obligatorias; y contribuyen al enriquecimiento de la Orientación a Objetos.

8. Herencia Múltiple
9. Revision de Tipos
10. Concurrencia.

CAPITULO II

CARACTERÍSTICAS DE MANEJADORES DE BASES DE DATOS

El diseño tradicional de sistemas de bases de datos ha sido determinado en gran medida para responder a las necesidades típicas de aplicaciones en los negocios. Antes de que los sistemas manejadores de bases de datos fuesen introducidos al ambiente comercial, los programas poseían su propio conjunto de archivos, cada uno de los cuales tenía su propia idiosincrasia en cuanto a la estructura y al formato de la información. Al crear nuevas aplicaciones sobre información existente, los datos deberían tomarse de fuentes de información muy dispersas. En pocas palabras, los programas dependían de la estructura de los datos almacenados, dificultando así el mantenimiento de aplicaciones y el cambio de estas estructuras.

Los sistemas manejadores de bases de datos perfeccionaron el proceso de desarrollo de aplicaciones en ambientes de grandes volúmenes de información. Proporcionando una vista de los datos simple y uniforme, expresada en términos independientes de la estructura de los datos. Otro beneficio de los sistemas manejadores de bases de datos, es el concerniente a la integridad de la información, ya que el propio sistema controla la consistencia e integridad de información, de esta forma se elimina de cada programa de la aplicación, el chequeo de la integridad antes de almacenarla. Más aún, los sistemas manejadores de bases de datos poseen rutinas para que el acceso físico a los datos sea óptimo.

Antes que nada, hay que analizar: ¿Qué se entiende por un sistema manejador de base de datos?. Los Sistemas manejadores de bases de datos presentan las siguientes características:

II.A. MODELO Y LENGUAJE.

Un manejador de base de datos (DBMS) posee un modelo no trivial y un lenguaje. Esto es, el manejador de base de datos entiende alguna estructura sobre los datos (el modelo de datos) que contiene, y provee operaciones para manipularla. Comúnmente el modelo de datos provee una estructura para los registros o conjunto de registros ordenados, tales como conjuntos (sets), o árboles (trees), listas, o relaciones (relations). Las operaciones incluyen la creación, destrucción y modificación de registros; y un mecanismo para la búsqueda, dentro de un conjunto de registros, de un valor dado.

II.B. RENDIMIENTO (PERFORMANCE).

Un manejador de base de datos provee un almacenamiento consistente y estable. Se entiende por consistente el que un dato sea accesible hasta el final del proceso que lo crea. Por estable, que el dato tenga permanencia en caso de alguna falla ya sea del sistema, del medio de almacenamiento, o del proceso. En el caso de una falla del proceso, aborta el proceso que hace uso de la base de datos. En el caso de una falla del sistema, la computadora en la cual reside la base de datos, deja de funcionar debido a un error de hardware, del sistema operativo o inclusive del manejador de base de datos. En un error del medio de almacenamiento, el dispositivo de almacenamiento se corrompe. La mayoría de los

manejadores de bases de datos con procedimientos de recuperación, escriben información acerca de los cambios a la base de datos en un medio secundario usándolo para efectuar correcciones a los datos después de la falla. Otros manejadores de base de datos agregan un atributo correspondiente al estado de la información; es decir, si se ha almacenado ya, o se encuentra en el limbo. El estado de el limbo es principalmente usado en las bases de datos distribuidas .

II.C. COMPARTIR INFORMACIÓN.

Un sistema manejador de base de datos debe permitir que los datos sean compartidos. A un nivel muy simple, un manejador de base de datos permite que un dato sea creado por un usuario o programa, y sea usado por otros. Este requisito excluye a los sistemas que simplemente crean un área de trabajo individual, en algunos sistemas se les llama área de datos local (Local Data Área); los cuales no permiten el uso simultáneo entre dos o más usuarios. Los manejadores de bases de datos convencionales poseen mecanismos de control de concurrencia los cuales previenen a los usuarios de ejecutar acciones inconsistentes en la base de datos. Gran parte del control de concurrencia está relacionado con las restricciones de integridad, de las cuales se hablará más adelante. Por ahora, lo importante es que cuando un usuario termine una transacción sobre la base de datos, en ese momento sea disponible a cualquier programa o usuario.

II.D. RESTRICCIONES DE INTEGRIDAD.

Un manejador de base de datos debe ayudar a asegurar la consistencia de información. Esto es reforzar la integridad por medio de restricciones de manera tal que la información

contenida en la base de datos siempre sea válida. Algunas clases comunes de restricciones son: el especificar un rango de valores válido para un campo. Uso de llaves, las cuales nos identifican un valor único en un conjunto de registros. Y las restricciones de referencia integral las cuales limitan las acciones dentro de un conjunto de registros. Estas reglas son denominadas como restricciones.

II.E. TAMAÑO ARBITRARIO DE LA BASE DE DATOS.

El espacio de almacenamiento de un manejador de base de datos no debe limitarse ni condicionarse a las características físicas del procesador. De manera tal que el tamaño de la base de datos no sea limitado por la cantidad de memoria, rango de direccionamiento de memoria virtual, etc..

II.F. CONTROL DE ACCESO O SEGURIDAD.

Algunos sistemas manejadores de bases de datos poseen la característica de que el usuario es propietario de la información que él crea; esto es, que el usuario es propietario de determinada información, y solamente él puede otorgar o restringir el acceso de otros usuarios a la información que le pertenece.

Existen otros manejadores de bases de datos que únicamente controlan el acceso a la información; esto es, a determinados usuarios les permite accesar toda la información, y a otro conjunto de usuarios solamente les permite ver un subconjunto de la información.

II.G. CONSULTAS. (LENGUAJE PARA CONSULTAR DIRECTAMENTE LA BASE DE DATOS)

Especialmente los manejadores de base de datos relacionales poseen la facilidad de expresar la consulta requerida o modificación a la base de datos sin necesidad de expresar como va a ser efectuada la lectura a los dispositivos de almacenamiento. En otras palabras: el usuario dice qué información solicita, y el manejador de base de datos determina cómo va a ser leída la información en los dispositivos de almacenamiento. De esta forma muchos manejadores de bases de datos soportan la creación y mantenimiento de estructuras auxiliares para el acceso a los datos, ya sean índices o listas invertidas, las cuales determinan una trayectoria alterna para localizar datos.

II.H. ESQUEMA DE SEPARACIÓN.

La mayoría de manejadores de bases de datos mantienen un esquema central, un catálogo de tipos definidos en la base de datos y de los objetos declarados de esos tipos. Diferentes programas y usuarios comparten esta meta-información. Lo contrario a este mecanismo es cuando un programa contiene sus propias declaraciones, tipos, y variables.

II.I. VISTAS.

Esta característica permite la definición de datos virtuales, los cuales no están explícitamente almacenados. Las vistas son empleadas para dar un acceso selectivo a los datos, y para presentar al usuario la información en una forma más accesible y que la pueda comprender fácilmente.

II.J. ADMINISTRADOR DE LA BASE DE DATOS.

Esta es una interfaz especial para que el administrador de la base de datos efectúe tareas como:

- Reorganización. Cambiar la estructura física o lógica de los datos, de forma transparente a los usuarios.
- Monitoreo. Obtener estadísticas sobre la frecuencia de las consultas en los datos.
- Auditoría. Llevar un registro del uso de los datos, quién lo grabó, quién lo modificó, a qué hora, etc..
- Respaldo. Obtener respaldos periódicos de toda o una parte de la base de datos.

II.K. GENERADOR DE FORMAS Y DE REPORTES

A partir de que en las aplicaciones comúnmente se efectúan procesos de captura, reportes, gráficas, formato de información; muchos proveedores de manejadores de bases de datos incluyen o venden un generador de formas o de reportes con especificaciones de alto nivel.

II.L. DICCIONARIO DE DATOS.

Un diccionario de datos es una extensión del esquema de base de datos. Además es empleado como documentación de la base de datos. Este es el lugar donde se centraliza la definición de la base de datos.

II.M. DISTRIBUCIÓN.

Esto se enfoca a distribuir la información en diferentes computadoras en red. Generalmente se efectúa la distribución de información con el propósito de perfeccionar el rendimiento de las aplicaciones e incrementar la disponibilidad de datos.

De todas éstas características que poseen los manejadores de bases de datos, las fundamentales son:

El Modelo y el Lenguaje.
Rendimiento.
El Compartir Información.
Tamaño Arbitrario de la Base de Datos.
Integridad de Información.

Frecuentemente y aunado a lo anterior se encuentran las siguientes características:

El Lenguaje de Consulta.
Vistas.
Administración de la Base de Datos

Y con menos frecuencia están presentes:

Generador de Reportes y Formas.
Diccionario de Datos.
Distribución.

CAPITULO III

SISTEMAS MANEJADORES DE BASES DE DATOS ORIENTADOS A OBJETOS

Actualmente no existe un modelo lo suficientemente fuerte en la orientación a objetos para considerarlo como tal; es decir, no existe aún, como lo es la teoría relacional para el Modelo Relacional.

III.A. CONCEPTOS PROPIOS DE MANEJADORES DE BASES DE DATOS ORIENTADOS A OBJETOS.

Los nuevos conceptos como lo son: persistencia, la identidad, polimorfismo, tipos, clases, herencia, etc.; provocaron cambios en la conceptualización de un manejador de base de datos. Teniéndose que incorporar éstos al manejador de base de datos. A continuación, trataré cada uno de ellos.

III.A.1. PERSISTENCIA Y ALCANCE.

En un lenguaje de programación, todos los objetos manipulados desaparecen al término de la ejecución del programa que los creó y manipuló. Por el contrario, en el manejador de base de datos orientado a objetos, todos los objetos son creados para subsistir y poder ser consultados y modificados posteriormente por las aplicaciones. Esto es, la propiedad de que los objetos sobrevivan a la ejecución de un

proceso, con el fin de que eventualmente sea utilizado por otros procesos. La persistencia debe ser *ortogonal*; es decir, que cada objeto, independientemente de su tipo, se le permita existir como tal. También debe ser implícita: el usuario no tiene que mover o copiar explícitamente el objeto para hacerlo persistente. La persistencia es entonces la primera característica que deberá ofrecer todo manejador de base de datos orientado a objetos.

III.A.2. TIPOS DE OBJETOS Y CLASES.

Existen dos categorías de sistemas orientados a objetos: Aquellos que soportan la noción de clase, y los que soportan la noción de tipo.

El tipo en un sistema orientado a objetos, resume todos los rasgos distintivos de un conjunto de objetos con las mismas características. Corresponde a la noción de tipo de dato abstracto. El tipo esta formado por dos componentes: la inteface y la implementación.

Solamente la interface es visible por el usuario y consiste en una lista de operaciones (con los parámetros de entrada y el resultado).

La implementación solamente es vista por el diseñador del tipo, y esta formada a su vez por dos partes: los datos y la operación. La parte de datos describe la estructura interna de los datos del objeto; la estructura de los datos puede ser compleja o sencilla. La parte de la operación, consiste en la función que se efectuará cuando sea llamada por el usuario.

En los lenguajes de programación, los tipos son herramientas para incrementar la productividad del

programador, asegurando la corrección del programa. Si el tipo es diseñado cuidadosamente, el sistema efectúa la revision de tipos al momento de la compilación; de otra forma, no se efectuará en ningún momento. Además, no se pueden modificar los tipos al momento de ejecución.

La noción de clase es diferente a la de tipo. Comprende dos aspectos: la fabrica de objetos y el almacén de objetos. La fabrica de objetos es usada para crear nuevos objetos, o duplicar objetos existentes. El almacén de objetos significa accesar las clases y sus extensiones, es decir, el conjunto de objetos que son instancias de una clase. El usuario puede manipular el almacén aplicando operaciones a todos los elementos de una clase. Las clases no son usadas para revisar la corrección de un programa, sino para crear y manipular objetos. En la mayoría de los sistemas que emplean el mecanismo de clase, ésta se puede manipular en el momento de ejecución.

III.A.3. JERARQUÍAS

La jerarquía, esta íntimamente relacionada con la herencia y las clases o tipos. La jerarquía consiste en establecer el orden bajo el cual se heredarán características, propiedades y operaciones, entre las clases, generando así subclases.

La definición de la jerarquía depende del tipo de herencia que se esté empleando, de forma tal que cada tipo de herencia definirá una reorganización de clases, postulando una jerarquía entre ellas.

Las formas de herencia que afectan la jerarquía y la clasificación son: Herencia por Substitución, Herencia por Inclusión, Herencia Restrictiva.

Herencia por Substitución: Decimos que una clase *X* hereda de la clase *Y*, si podemos efectuar más operaciones sobre objetos de la clase *X* que en objetos de la clase *Y*. Así, en cualquier lugar que podamos tener un objeto de la clase *Y*, lo podemos substituir por un objeto de la clase *X*. Este tipo de herencia está basado en el comportamiento y no en el valor.

Herencia por Inclusión: Corresponde a la noción de clasificación. Estipula que *Y* es una subclase de *X* si cada objeto de la clase *Y* es también un objeto de la clase *X*. Este tipo de herencia está basado en la estructura y no en las operaciones.

La *Herencia Restrictiva* es un caso especial de la herencia por inclusión. Una clase *Y* es una subclase de *X* si todos los objetos de la clase *Y* cumplen una restricción dada.

III.A.4. POLIMORFISMOS.

Aquí, el concepto de polimorfismo es el mismo que se menciona en el punto I.B.2 (página 14) del presente trabajo. En este caso, se recurre a la revision de tipos de variables enviados en el mensaje al momento de ejecución para determinar el método que se empleará.

III.A.5. IDENTIDAD.

En las bases de datos orientadas a objetos que se basan en la identidad, es importante mencionar que la forma en que los objetos son relacionados puede, o no estar basado en esta característica. Las referencias que hacen uso de la identidad de un objeto directamente, son como los apuntadores en lenguajes convencionales. En dichas bases de datos, es

posible relacionar un objeto x a otro objeto, o conjunto de objetos S, almacenando la identidad de S (o de los miembros de S) en x, ya que todos los accesos a un objeto x son a través de mensajes definidos sobre su tipo. Sin embargo la asociación entre x y S debe ser hecha por un método. Este método puede hacer uso de la identidad almacenada o puede usar una expresión basada en el valor (este último en un ambiente relacional). De esta forma, las bases de datos orientadas a objetos proveen un marco para hacer un acceso único basados en el valor y un acceso basados en la identidad.

Una observación de interés es que es diferente el concepto de llave primaria en una base de datos relacional y la identidad de un objeto. La llave primaria no es única en todo el sistema, solamente es única dentro de una sola relación. Por otro lado, la identidad de un objeto es única en toda la base de datos, esta referencia a objetos hace fácil el crear colecciones con objetos de tipos heterogéneos. El concepto de llave juega un papel importante dentro de las bases de datos orientadas a objetos; bajo este ambiente, la llave es una propiedad o conjunto de propiedades que identifican a un objeto único dentro de un conjunto o colección. La llave no es algo que pueda ser asociado a un tipo, ya que la unicidad de la llave es garantizada solamente con una colección de objetos.

I.A.6. OBJETOS COMPLEJOS.

Los objetos complejos son construidos desde su forma más simple, aplicándoles construcciones. Los objetos más simples, son objetos como enteros, caracteres, conjuntos de bytes de cualquier tamaño y booleanos. Hay varias construcciones de objetos, como: tuplas, conjuntos, listas, arreglos; estos forman los objetos compuestos.

Los conjuntos son básicos por que son una forma natural de representar colecciones del mundo real. Las tuplas son básicas por que es una forma natural de representar propiedades de una entidad. Ambos, conjuntos y tuplas son importantes por que tienen gran aceptación como constructores en un mundo relacional.

Las listas y arreglos son importantes por que captan el orden que existe en el mundo real. También son útiles en aplicaciones científicas, donde se necesitan matrices o series de tiempo.

Los constructores de objetos deben de ser ortogonales; esto es: cualquier constructor debe aplicarse a cualquier objeto. Los constructores del modelo relacional, no son ortogonales, ya que el conjunto de constructores pueden ser aplicados solamente a tuplas, y el constructor de tuplas puede solamente aplicarse a valores atómicos.

Cabe hacer notar que los objetos complejos requieren que el operador apropiado sea proporcionado para tratar con tales objetos. Esto es, que las operaciones sobre un objeto complejo se propaguen transitivamente a todos sus componentes.

Una de las limitaciones del modelo relacional, citada con mayor frecuencia, es su descripción de las entidades del mundo real a través de estructuras "planas", con muy pobre riqueza semántica (fidelidad en la descripción de los objetos de la realidad). Los modelos para objetos complejos proponen la posibilidad de construir estructuras que no estén en Primera Forma Normal, gracias al manejo de atributos con valores no atómicos, sino de tipo tupla, conjunto, lista, o resultado de una aplicación ortogonal de tales constructores.

Un ejemplo de este tipo de objetos es el objeto "Gráfica de un número complejo".

III.B. POSTULADOS.

Ahora bien, la tendencia de cada una de las investigaciones trajeron consigo diversos puntos de vista, características y propiedades de los objetos. A continuación mencionaré algunos de ellos:

III.B.1 EL MODELO DEL UMBRAL Y EL MODELO DE REFERENCIA.

Uno de los primeros esfuerzos por definir y postular las características de los manejadores de bases de datos orientados a objetos se debe a Stanley Zdonik [ZDONIK 90]. En su trabajo menciona las características de los manejadores de bases de datos relacionales:

El modelo y el lenguaje
Relaciones
Lenguaje de definición de datos
Lenguaje de manipulación de datos
Vistas
Distribución
Diccionario de datos
Restricciones de integridad

El modelo del Umbral deberá cumplir con los siguientes requisitos:

1. Deberá proveer la funcionalidad de las bases de datos relacionales.

2. Deberá soportar la identidad del objeto

3. Deberá proveer de un método de encapsulación.

4. Deberá soportar el estado complejo de objetos.

Como se habrá notado, faltan aún muchas características de la orientación a objetos; sin embargo, lo anterior solamente postula un modelo base, y dependiendo de la tendencia (ya sea inteligencia artificial, bases de datos de conocimientos, lenguajes, etc.) se extenderá este modelo.

El modelo de referencia deberá contemplar los siguientes puntos:

1. Deberá cumplir con el modelo del Umbral

2. Representación estructurada de objetos.

3. Persistencia.

4. Tipos de objetos y variables.

5. Jerarquías entre clases, en estructura de árbol.

6. Polimorfismos.

7. Colecciones o agrupamiento de objetos.

8. Consultas en línea o existencia de índices.

III.B.2 EL MANIFIESTO DE LOS SISTEMAS DE BASES DE DATOS ORIENTADOS A OBJETOS.

Más adelante (1990), se conjuntaron los siguientes investigadores: Malcolm Atkinson, Francois Bancilhon, David DeWitt, Klaus Dittrich, David Maier y Stanley Zdonik. El propósito fue acordar cuando un sistema deberá nombrarse orientado a objetos. Como resultado se obtuvo el Manifiesto [ZICARI 90]. El cual postula que todas las características caen bajo tres rubros: Obligatorias, Opcionales y Abiertas.

OBLIGATORIAS. Aquí se considera cumplir con 2 condiciones: deberá ser un manejador de base de datos y deberá ser orientado a objetos. Para que se considere un manejador de base de datos tendrá que cumplir con:

- Persistencia.
- Manejador de almacenamiento.
- Concurrencia.
- Recuperación de datos.
- Herramienta de consultas ad-hoc.

Ahora bien, para que se considere orientado a objetos, tendrá que satisfacer:

- La identidad del objeto.
- Encapsulación.
- Tipos y/o Clases.
- Jerarquía de Clases y/o Tipos.
- Sobrecarga y ligado dinámico
- Completes computacional.
- Extensibilidad.

OPCIONALES. Este tipo de características, perfeccionan el funcionamiento del sistema en general. Y sirven a aplicaciones particulares, como por ejemplo CAD/CAM, CASE, Automatización de oficinas. Estas características son:

- Herencia múltiple.
- Revision de tipos.
- Distribución.
- Diseño de transacciones.
- Versiones.

ABIERTOS. Hasta aquí se han dictado algunas reglas para los sistemas orientados a objetos. Sin embargo, cuando se diseñan sistemas orientados a objetos, existen muchas alternativas para implementarlo. Estas son libertades en la implementación de sistemas orientados a objetos. Estas formas de implementación, difieren de las características OBLIGATORIAS, en el sentido que aún no se ha alcanzado ninguna conclusión dentro de la comunidad científica.

Un ejemplo de estas diferencias, es el grado de uniformidad que se espera para un sistema orientado a objetos; esto es, considerar preguntas tales como: ¿es un tipo, un objeto?, ¿es un método de un objeto?, ¿o deberán ser tratados los tres de forma diferente?. Podemos ver el problema en tres niveles diferentes: nivel de implementación, nivel de lenguaje de programación y el nivel de interface.

A nivel de implementación, uno debe decidir el tipo de información que debe ser almacenada como objeto, y el tipo de información que debe ser almacenada en el objeto.

A nivel lenguaje de programación, se basa principalmente en el nivel en que se efectúan las acciones, ya sea sintácticamente o semánticamente.

Finalmente a nivel de interface. Uno desearía presentar al usuario una vista uniforme de los tipos, los objetos y los métodos; aún cuando, semánticamente en el lenguaje de programación sean de diferente naturaleza. Por el contrario, se pueden presentar como diferentes entidades, aún cuando en el lenguaje de programación sean lo mismo.

CAPITULO IV

PROPUESTA FORMAL AL MODELO DE OBJETOS.

Como se ha mencionado anteriormente, los manejadores de bases de datos están basados en un modelo. Sin embargo, no existe una teoría formal sobre los cuales este basado el paradigma de objetos. En 1979 M. Bunge hace el primer intento de formalizar las bases para un desarrollo teórico del modelo orientado a objetos [WON 89]. Principalmente hace uso de los principios Ontológicos. La ontología es una rama de la Filosofía que estudia el modelar la existencia de cosas en el mundo real. Se toma como base la ciencia de la filosofía con el propósito de encontrar formalmente la noción de objeto.

IV.A. PRINCIPIOS ONTOLÓGICOS.

A continuación mencionaré las bases del formalismo de M. Bunge. Comenzando con los principios fundamentales:

El mundo está compuesto de cosas. Consecuentemente las ciencias de la realidad (natural o social) estudian cosas, sus propiedades y cambios.

Las formas son propiedades de las cosas.

a) Se estudia y se modifican las propiedades por medio de la examinación de las cosas, forzándolas a cambiar.

b) Las funciones sobre objetos son representadas por operaciones definidas sobre dominios, los cuales son conjuntos de objetos concretos.

Las cosas están agrupadas en sistemas o se agregan en componentes que interactúan; no hay nada que deje de ser parte de al menos un sistema. No hay cosas independientes: los límites que se trazan entre entidades son con frecuencia imaginarias. Lo que en realidad hay, son sistemas.

Todo cambia. Otra forma de comprenderlo es considerar que no existe absolutamente nada que permanezca siendo lo mismo en el mismo estado.

Nada proviene de nada y nada se reduce a nada. Nada se crea espontáneamente, todo fenómeno tiene antecedentes. A su vez, nada desaparece espontáneamente.

Todo obedece a leyes. Ya sea naturales o sociales, las leyes son relaciones invariables sobre las propiedades.

IV.A.1. LAS COSAS Y SUS PROPIEDADES.

El mundo esta formado por dos categorías de cosas: cosas concretas que llamaremos *entidades* o *individuos substanciales*, *y cosas conceptuales*. Para nuestros propósitos trataremos con individuos substanciales, así, al mencionar entidad, individuo y cosa se estará haciendo referencia a lo mismo. De aquí en adelante se usará indistintamente entidad, individuo y cosa. Es importante que estas sean cosas concretas y no conceptos o tipos de cosas. Un individuo puede ser simple o compuesto; literalmente, compuesto de otros individuos. La idea de que un objeto puede ser combinado para generar otros objetos es formalizado en el siguiente postulado:

POSTULADO. Existe una operación binaria, llamada *asociación* o *concatenación*, tal que:

1. Si a_1 y a_2 son individuos, entonces $a_1 \cdot a_2$ es también un individuo.

2. $a \cdot a = a$ (individuos substanciales son indempotentes sobre la asociación)

3. La operación de asociación es conmutativa.

4. La operación de asociación es asociativa.

Básicamente este postulado menciona que cualquier colección de individuos puede ser visto como un individuo y que el orden en el cual se combinen no es importante. También que un individuo no puede ser combinado con él mismo para crear algo nuevo. Sin embargo a continuación se menciona cómo un individuo puede estar formado por un conjunto de individuos.

DEFINICIÓN. Un individuo x es compuesto si solamente si existe un individuo substancial y y z tal que $y \neq x$, $z \neq x$ y $x = y \cdot z$. De otra forma el individuo es *simple*.

El significado de la definición anterior es que: un individuo se considera compuesto si puede ser descrito como formado por dos individuos diferentes. Por ejemplo: si un componente electrónico puede ser partido en dos sub-componentes (donde cada uno puede ser también partido), entonces es considerado un objeto compuesto.

DEFINICIÓN. Sea x y y individuos substanciales. Entonces x es-parte-de y si solamente si $x \cdot y = y$.

DEFINICIÓN. La *composición* de un individuo es el conjunto de sus partes:

$C(x)=\{y|y$ es-parte-de $x\}$

Un concepto importante es que las entidades poseen propiedades. Las propiedades de individuos substanciales son llamadas *propiedades substanciales*. Se hace una distinción entre lo que es propiedad y lo que es atributo. Un individuo puede tener una propiedad que es conocida para nosotros. En contraste, un atributo es una característica asignada por nosotros a un objeto. Algunos atributos de un objeto reflejarán sus propiedades, en consecuencia: reconocemos a las propiedades solamente a través de sus atributos. Una propiedad conocida, debe tener al menos un atributo que lo represente.

Las propiedades no existen por sí mismas, sino que son asignadas a entidades. Por otro lado, las entidades no son solo un conjunto de propiedades. Así, se puede decir que el componente fundamental del mundo son las entidades, que las entidades son conocidas por nosotros a través de sus propiedades, y que las propiedades se materializan en términos de atributos.

Las propiedades de cosas compuestas pueden ser relacionadas a las propiedades de las cosas en su integración. Aquí las propiedades de las cosas compuestas caen en dos clases: *hereditarias*, esto es propiedades que pertenecen a los componentes de una entidad compuesta; y *no hereditarias*, las cuales llamaremos propiedades emergentes.

DEFINICIÓN. Sea P una propiedad de una entidad x con composición $C(x)$. Entonces:

1. **P** es una propiedad hereditaria o resultante de x si solamente si existe $y \in \mathcal{C}(x)$, $y \neq x$ tal que **P** es una propiedad de y.

2. **P** es una propiedad emergente de x si no existe $y \in \mathcal{C}(x)$, $y \neq x$ tal que **P** es una propiedad de y.

Para ejemplificar lo anterior, se considera el ejemplo de una computadora: la capacidad de memoria de la computadora es la misma que el tamaño de la memoria, por lo que es una propiedad hereditaria. Sin embargo, el rendimiento de la computadora depende de como interactúen todos los componentes, por lo que se trata de una propiedad emergente.

DEFINICIÓN. Sea T un subconjunto de los individuos substanciales. Sea P el conjunto de propiedades.

1. El conjunto de propiedades del individuo $x \in T$ es $p(x)=\{P \in P | x$ posee P$\}$.

2. El conjunto de propiedades substanciales de T es $p(T)=\{P \in P |$ para todo $x \in T$, x posee P$\}$.

Es importante que dos cosas concretas no sean lo mismo. Esto es formalizado en la siguiente premisa.

POSTULADO. No existen dos individuos substanciales que tengan las mismas propiedades.

Si se perciben que dos entidades son idénticas, es solo por que no se asignan atributos a todas sus propiedades substanciales. Ahora podemos resumir las premisas principales sobre una cosa:

1.- Todas las cosas poseen propiedades.

2.- Las propiedades no existen independientemente de las entidades.

3.- Las entidades no son conjunto de propiedades.

4.- La totalidad de propiedades de una entidad (individuo substancial) es requerida para diferenciarla de cualquier otra entidad.

DEFINICIÓN. Sea x un individuo substancial y $p(x)$ el conjunto de sus propiedades. Entonces el individuo con sus propiedades es llamado un objeto concreto X:X=<x,$p(x)$>.

Las cosas son percibidas (en ciencias, ontología, u otras representaciones simbólicas) como conceptos o modelos. Esto es formalizado a través de la noción de un esquema conceptual. Un esquema conceptual de una cosa será llamado un modelo. Una *cosa modelo* es definida en términos de un marco de referencia, el cual es un punto de vista, y un conjunto de funciones que representan las propiedades que son importantes para el propósito por el cual modelamos. Bunge emplea la física para demostrar el concepto; por ejemplo: un punto de masa es visto diferente desde varios marcos de referencia, en diferentes momentos, y puede ser descrito por varios conjuntos de propiedades (tales como <masa, posición, fuerza>). Estos conceptos pueden extenderse a tipos de sistemas de información de negocios; por ejemplo: considere un producto que es manufacturado, distribuido y vendido a través de distribuidores. El mismo producto puede ser visto de diferente manera por el fabricante, el distribuidor y el cliente. Cada uno de ellos puede asignarle atributos diferentes debido a las propiedades del producto que le son relevantes.

DEFINICIÓN. Sea X=<x,$p(x)$> un objeto de la clase T. Un esquema funcional X_m de X es un cierto conjunto no vacío M, junto con

una secuencia **F** de funciones sobre M, cada una de las cuales representa una propiedad de cosas en T. Abreviando:

$X_m=<M,F>$ donde $F=[F_i|F_i$ es una función sobre M y $1\leq i \leq m <\infty]$

En esta definición entenderemos por M todos los posibles marcos de referencia o puntos de vista.

POSTULADO. Cualquier cosa puede ser modelada como un esquema funcional.

IV.A.2. ESTADOS Y LEYES.

En un esquema funcional **F** cada componente es una función $F_i:M\rightarrow V_i$ donde V_i es el dominio de valores para la propiedad representada por la función F_i. Las propiedades F_i son llamadas *variables de estado* o *funciones de estado*. El conjunto de todos los valores F_i es llamado *estado* de una cosa. Todo, en un momento dado, esta en un estado. Formalmente:

DEFINICIÓN. En un esquema funcional $X_m=<M,F>$, F_i para $1\leq i \leq n$ es llamado el i-esimo estado funcional para X; **F** es el total de funciones de estado para X y sus valores $F_m=<F_1,\ldots,F_n>(m)=<F_1(m),\ldots,F_n(m)>$ para $m \in M$ se dice que representa el estado de X en m.

Cabe hacer notar que no hay un estado absoluto de la función. Elegir un estado de la función específica depende del conocimiento, metas, vistas, etc..

En la realidad, no toda combinación de valores de las funciones de estado puede tomarse como estado de una cosa. Este es limitado por la naturaleza de las cosas. Lo anterior se formaliza en el concepto de leyes.

DEFINICIÓN. Sea X_m=<M,F> un esquema funcional para una cosa X. Cualquier restricción sobre los posibles valores de los componentes de F o cualquier relación entre dos o más componentes de F es llamado una *ley*.

Dado que la ley contiene información acerca de las cosas, las leyes son propiedades de las cosas. De otra forma, una ley puede ser vista como la descripción de los posibles estados de una cosa (es un esquema funcional dado) en el conjunto {Legal, Ilegal}. Siendo una función sobre los valores de propiedades, que son funciones sobre M (el posible marco de referencia), una ley es también una función sobre M.

El concepto de una ley es fundamental en el modelado de las cosas debido a que contienen el conocimiento de lo que puede ser una cosa y de lo que no puede ser. En este sentido, las leyes contienen información dinámica de las cosas. Aún más, las leyes definen el conjunto de estados en que una cosa puede estar, opuesto al estado concebible que son las posibles combinaciones de valores para las funciones de estado. De ahí la siguiente definición.

DEFINICIÓN. Sea X_m =<M,F> un esquema funcional para la cosa X, donde $F:<F_1,\ldots,F_n>:M\rightarrow V_1\times\ldots\times V_n$ es el estado de la función y sea L(x) el conjunto de todas las leyes de X. Entonces el conjunto del codominio $V_1\times V_2\times\ldots\times V_n$ de F restringido por las condiciones (leyes) en L(x) es llamado el *espacio de estados legales* de X en la representación de X_m o $S_L(x)$.

$S_L(X)=\{<X_1,\ldots,X_n> \in V_1\times\ldots\times V_n | F$ satisface conjuntamente todo miembro de $L(X)\}$

Y todo punto de $S_L(x)$ es llamado estado legal (o realidad posible) de X en representación de X_m.

Considerese el ejemplo: la cosa es una cuenta contable y todas las transacciones relacionadas. Las leyes son: "Balance es la suma algebraica del monto de transacciones" y "Ninguna fecha de transacción es previa a la fecha en que la cuenta fue creada". En este caso, el espacio de estados legales de la cuenta es aquel en el cual todas las transacciones pertenecen a fechas después de una fecha dada y el balance es igual a la suma de las transacciones.

IV.A.3. CLASES Y ESPECIES.

Desde que las cosas son conocidas a través de sus propiedades, las cosas que poseen las mismas propiedades son similares en cierto sentido. Así, las propiedades pueden ser utilizadas para clasificar las cosas. Esta noción es formalizada en las siguientes definiciones.

DEFINICIÓN. (The scope) La extensión de una propiedad substancial es la colección de entidades que la poseen: $G(P)=\{x|P\in p(x)\}$.

DEFINICIÓN. Un subconjunto X de individuos es llamado una *clase* si y solamente si existe una propiedad substancial tal que X es la extensión de P.

DEFINICIÓN. Sea R un subconjunto de propiedades. El conjunto $\cap\{G(P)|P\in R\}$ es llamado la *especie*-R.

Una especie-R es una clase con respecto a una "propiedad compuesta" formada por todas las propiedades en R.

Recordar que la leyes también son propiedades de las cosas; de ahí que, la definición de una especie-R puede ser enfocada al conjunto de propiedades que son leyes. Esto trae la siguiente definición.

DEFINICIÓN. Sea L* un subconjunto de leyes. El conjunto $\{G(L) | L \in L^*\}$ es llamado la *especie natural*.

Todas las cosas de la misma especie natural tendrán el mismo espacio de estado legal. El espacio de estado legal es en consecuencia una forma de describir las propiedades generales de la misma especie.

Para mostrar lo anterior considere los estudiantes de la universidad. Ellos pertenecen a una clase dado que hay dos propiedades en común para todos y cada uno de ellos:

1) Todos ellos son seres humanos.

2) Están inscritos en una universidad dada.

Asumir ahora que la universidad tiene regulaciones específicas sobre los cursos que se deben terminar a fin de graduarse. Entonces, para los estudiantes a graduarse, el estado de variables describiendo los cursos tomados, tendrá valores de acuerdo a los requisitos de graduación, esto es: su estado será legal con respecto a las reglas de graduación. Los graduados serán una especie natural con respecto a los requisitos de graduación.

En base a lo anterior, una cosa puede pertenecer a más de una especie:

TEOREMA. El conjunto de todas las especies forman una reticulación (lattice) completa sobre la inclusión.

En lugar de demostrar el teorema, lo haré de manera intuitiva: El teorema especifica que el conjunto de todas las especies forma una jerarquía múltiple. Sea k_1 una especie y k_2 un subconjunto propio de k_1. Entonces, todos los elementos de

k_2 tienen un conjunto de propiedades común que los elementos de k_1 poseen. También, una especie puede ser el subconjunto de dos especies, y la relación de inclusión es asimétrica. De ahí que el conjunto de todas las especies puede ser descrita como una gráfica acíclica directa.

IV.A.4. CAMBIOS, EVENTOS E INTERACCIÓN.

Un conocimiento completo de las cosas requiere información acerca de como las cosas pueden cambiar.

POSTULADO. Todas las cosas (concretas) tienen al menos dos estados distintos.

Cuando una cosa sufre un cambio, al menos una propiedad tendrá que cambiar en valor; de ahí que el cambio de una cosa es manifestado como el cambio de estado. De ahí se sigue que: para que un cambio sea posible, la cosa debe tener al menos un estado. Sin embargo, puede surgir una pregunta: cuando una cosa *x* cambia ¿Deja de ser *x* para convertiste en una nueva cosa *y*?; por ejemplo: si una cosa "Automóvil" ha sido pintado, ¿continúa siendo el mismo automóvil?. Para responder a esto, recordemos que una cosa es descrita a través de un esquema funcional, y que el esquema funcional describe un punto de vista de las cosas, en consecuencia, todo depende de la persona que opine.

Una cosa, si existe, retendrá su nombre a través de su historia tanto tiempo hasta que exista un cambio en su especie natural. Un cambio, el cual afecte el nombre de la cosa.

El cambio de una cosa será denominado como evento. Un evento puede ser descrito como un par ordenado $<s_1, s_2>$ donde s_1 y s_2 son los estados antes y después del cambio

respectivamente. Considere el espacio de estados S(X) de una cosa X. El conjunto de todos los eventos posibles es E(X)=S(X)×S(X). Esto incluye el evento identidad para designar que no existió ningún cambio: $\langle s, s\rangle$ $s \in S(X)$. También dos eventos pueden ser combinados en otro evento si el primero termina antes de iniciar el segundo. Esta combinación es designada como $\langle a, b\rangle\langle b, c\rangle=\langle a, c\rangle$ y $\langle a, c\rangle$ será llamado un evento complejo.

No todos los eventos posibles pueden suceder en la realidad, debido a que la transición entre estados (aún cuando se trate de leyes) no siempre es permitida.

DEFINICIÓN. Sea $S_L(X)$ un espacio de estados legales para una cosa X. Entonces la familia de *transformaciones legales* del espacio en sí mismo es el conjunto de funciones:

$G_L(X)=\{g$ es una función $\mid g:S_L(X)\rightarrow S_L(X)$ y g es compatible con la ley de X$\}$

DEFINICIÓN. Sea $G_L(X)$ una familia de transformaciones legales sobre $S_L(X)$. Entonces el *espacio de eventos legales* es:

$$E_L(X)=\{\langle s,s'\rangle \in S_L(X)\odot S_L(X) \mid s'=g(s) \text{ y } g \in G_L(X)\}$$

Para ilustrar lo anterior, considere nuevamente el ejemplo de la universidad. La familia de transformaciones legales está definida por las transiciones permitidas al estado de cada estudiante; por ejemplo, las condiciones para inscribirse en un curso, terminar un curso, terminar un programa de estudios y graduarse. El espacio de eventos legales incluirá pares de leyes de estados tales como <el estudiante es inscrito en un curso a , el estudiante termina el curso a> , este conjunto de tuplas, prueba que el estudiante ha cumplido con todos los requisitos del curso (de

otra forma, no hay transformación legal que pueda causar este evento).

Recordando el principio de "Todas las cosas cambian". Este principio requiere que una cosa cambie de estado con el tiempo.

DEFINICIÓN. Sea F una función de estado para la cosa X relativo a un marco de referencia f . Sea el conjunto de estados para el marco de referencia $S(f)$. Asuma que hay una función uno a uno $S(f) \rightarrow S(X)$, y sea el estado de $S(f)$ denotado por $t \in S(f)$. La *historia* de X *relativa* a f es el conjunto de pares ordenados $h(X)=\{<t,F(t)> \mid t \in S(f)\}$.

El caso más común es que el estado de un marco de referencia f haga uso de f en diferentes momentos a través del tiempo. En este caso, la función $S(f) \rightarrow S(X)$ significa que para cualquier momento dado la cosa X está en un estado único. Tomando el ejemplo de la universidad, en un momento dado, el estudiante estará en un estado descrito por los cursos terminados y por los cursos que actualmente toma. La historia del estudiante será la descripción del estado del estudiante en diferentes momentos a través del tiempo.

La noción de historia nos permite definir la interacción de objetos. Básicamente, si dos objetos interactúan, entonces al menos uno de ellos no preservará el mismo estado, como lo haría si el otro no existiera.

DEFINICIÓN. Sean X y Y dos cosas diferentes con funciones de estado F_X y F_Y. Sean $h(X)$ y $h(Y)$ sus respectivas historias. X *actúa sobre* Y si la historia de Y dado X, es diferente de $h(Y)$. En otras palabras: $h(Y/X) \neq h(Y)$.

Para ejemplificar esto, asuma que el estudiante intercambia información acerca de los cursos. Podría suceder

que un estudiante decidiera no inscribirse en un curso, después de haber consultado con sus compañeros; por lo que la historia del estudiante sería diferente si no lo hubiese consultado.

DEFINICIÓN. Dos cosas diferentes *interactúan* si y solamente si cada una interactúa sobre la otra.

DEFINICIÓN. Dos cosas diferentes están ligadas si y solamente si al menos una de ellas actúa sobre otra.

Un principio importante es que ninguna cosa es observable si no puede ser actuada por otras cosas. Esto se expresa mejor en el siguiente postulado.

POSTULADO. Todo actúa, y es actuado por otras cosas.

IV.B. FORMALIZANDO LOS PRINCIPIOS ONTOLÓGICOS BAJO EL CONCEPTO ORIENTACIÓN A OBJETOS.

"En un sistema orientado a objetos, todas las entidades conceptuales son modeladas como objetos". Cada objeto tiene su memoria (privada) que guarda el estado de su información. Esta información es retenida como variables de instancia. Identificamos cinco conceptos que son atribuibles a la noción de objeto.

Principios básicos.
Abstracción de datos o encapsulación.
Independencia y persistencia.
Intercambio de mensajes.
Homogeneidad.
Herencia.
Encapsulación.

IV.B.1. PRINCIPIOS BÁSICOS.

El mundo está hecho de objetos. Los objetos describen cosas concretas; esto es, entidades específicas en lugar de tipos o clases. Por ejemplo: cuando hablamos de gente, cada individuo será visto como un objeto.

Los objetos son conocidos (observables) para el mundo a través de sus propiedades.

Las propiedades son de dos clases: atributos observables y leyes. Un *atributo observable* es una descripción del conjunto de objetos que tienen esta propiedad en un dominio dado; formalmente, es una propiedad a la cual se le pueden asignar valores específicos. Las *leyes* son restricciones sobre las combinaciones permitidas de los valores de atributos.

El conjunto específico de propiedades usado para describir un objeto dado depende del punto de vista y del propósito del modelado. Este conjunto es llamado un esquema funcional.

El valor de un atributo en un momento dado es llamado el estado de la variable.

El conjunto de estados de variables define el estado del objeto.

Los estados permitidos para un objeto son determinados por el conjunto de leyes del objeto. Las leyes también son propiedades de las cosas. Un objeto solamente puede estar en un estado consistente con su ley. Tal estado es llamado estado legal.

Dos o más objetos pueden ser combinados a través de la asociación para formar otra cosa. Un objeto que puede ser descrito como una asociación entre otros objetos, es llamado un objeto compuesto. Un objeto que no es compuesto, es un objeto simple.

Las propiedades de los objetos compuestos son:

1.- Resultante o hereditario.- Cuando la propiedad pertenece también a un objeto de la composición del objeto compuesto.

2.- Emergente.- Cuando la propiedad no pertenece a ningún objeto de la composición del objeto compuesto.

Los objetos solamente pueden existir en un estado legal.

Un objeto que está en un estado ilegal cambiará su estado a un estado legal, únicamente definido por el estado ilegal.

Los objetos no alcanzan un estado ilegal espontáneamente. Para mostrar esto, considere el ejemplo de la cuenta contable y las transacciones. Cuando una nueva transacción ocurre, el balance no será igual a la suma de las transacciones, de esta forma tendrá que cambiar. También, el balance no cambiará a menos que suceda una transacción. Es importante notar que los objetos no se someten a las leyes. En vez de eso, las leyes son reglas describiendo los estados que un objeto puede asumir. Si un objeto pertenece a un estado considerado ilegal, entonces la ley no fue bien definida.

Un objeto puede afectar el estado de otro objeto. Esto es hecho mediante una ley que liga el estado de las variables de los dos objetos.

Si un objeto afecta el estado de otro objeto, podría ser puesto en un estado ilegal; entonces el objeto afectado cambiará a un estado legal.

IV.B.2 ABSTRACCIÓN DE DATOS.

"El concepto más importante en la aproximación a la orientación a objetos es la abstracción de datos, Por esto entendemos que estamos más interesados en el comportamiento de los objetos, que en su representación." El comportamiento de los objetos es encapsulado en sus métodos. Los métodos son mecanismos que tienen acceso a un objeto y pueden cambiar el estado del objeto. Así, un objeto tipo es descrito en términos de la forma (variables de instancia), de sus instancias y las operaciones (métodos) aplicables a sus variables de instancia. Las variables de instancia, junto con sus métodos son llamados: las propiedades del objeto.

Ontológicamente: las cosas deben cambiar y los cambios no pueden ser descritos separadamente de las cosas. Acerca de esto, la encapsulación es un principio fundamental. Sin embargo, la noción de método o cualquier otro mecanismo para cambiar el estado de un objeto, no existe. En su lugar, el concepto de ley define las leyes o estados permitidos (es decir, combinaciones de estados de variables). Las leyes son vistas como propiedades de las cosas.

IV.B.3. INDEPENDENCIA.

La noción de independencia incorpora dos características a un objeto. Una es relacionada al estado de un objeto, y la otra es su existencia. Esto es: "Los objetos también tienen control sobre su propio estado. Una vez creado, un objeto

continuará existiendo (o persistirá) aún cuando su creador muera".

La *independencia* implica que la única forma en que un objeto puede cambiar su estado es a través de la acción de sus métodos; literalmente, a través de sus características dinámicas. En consecuencia, la encapsulación es una condición necesaria para la independencia.

En términos ontológicos, los posibles estados de una cosa son dictados por las leyes de la cosa. Estas leyes son consideradas propiedades de la cosa. La persistencia de una cosa se manifiesta mediante el principio de invarianza nominal que permite cambios de estado sin cambiar la esencia de la cosa. No hay mecanismos explícitos para la creación o destrucción de las cosas.

IV.B.4. INTERCAMBIO DE MENSAJES.

En la aproximación a la orientación a objetos, los objetos pueden comunicarse solamente mediante el intercambio de mensajes. Un mensaje puede causar que un objeto se comporte como se encuentra definido en el método del objeto. Así los objetos pueden afectarse unos a otros a través del intercambio de mensajes y ya que los objetos son independientes, es la única manera en que pueden hacerlo. De esta forma, los métodos pueden ser vistos como definiciones de respuestas a los posibles mensajes.

Ontológicamente, sobre todo objeto actúan otros objetos. El efecto de que una cosa actúe sobre otra lo vemos en el historial de las cosas; formalmente, su estado se transforma en el tiempo. Nuevamente, la ontología no se enfoca a los mecanismos específicos con los cuales la interacción entre las cosas se lleva a cabo.

IV.B.5. HOMOGENEIDAD.

La *homogeneidad* implica que todo es un objeto. En particular, para una homogeneidad completa, los mensajes y propiedades deben ser vistas como objetos. Lo anterior acarrea un círculo vicioso: Si los mensajes son vistos como objetos, entonces ellos deben mandar mensajes para comunicarse con objetos. Similarmente, si los atributos son objetos, ellos deben tener sus propios atributos, los cuales son objetos, etc. En consecuencia, uno tiene que parar arbitrariamente a cierto nivel para romper el círculo vicioso.

Ontológicamente se hace una distinción clara entre las cosas y sus propiedades. En este sentido, no hay una homogeneidad completa. Más aún, las propiedades no pueden existir independientemente de las cosas; así, hay una asimetría en las características fundamentales de los conceptos. Ontológicamente está permitido que las cosas estén compuestas de otras cosas, y la especificación hace que existan cosas simples las cuales no se pueden descomponer. De esta forma, la existencia de un nivel básico, es un principio y no una decisión arbitraria. Es posible que ciertas propiedades pertenezcan a una cosa que está en la composición de una cosa dada. También, ontológicamente se agrega la distinción entre propiedad resultante y propiedad emergente, así se formaliza la idea de que una cosa puede ser totalmente diferente de una colección de cosas en su composición. Esto implica que el estado de las variables de una cosa pertenecen a los objetos que la componen o solamente a la cosa en cuestión.

IV.B.5. HERENCIA.

Los objetos pueden ser agrupados en clases. Una clase es la definición de un tipo de objeto. Todos los objetos en una clase tienen las mismas variables de instancia y métodos; además, responden a los mismos mensajes. "Una clase describe la forma (variables de instancia) de sus instancias y las operaciones (Métodos) aplicables a sus instancias". Los objetos pueden ser categorizados en subtipos o subclases mediante la especialización. Los objetos de una subclase heredan todas las variables de instancia y métodos de su superclase, adicionalmente puede tener sus propias variables se instancia y métodos. Es importante notar que en la literatura existente de orientación a objetos, la herencia de clases es vista como un mecanismo para proveer un código reusable, ahorrando información en la base de datos, programación y simplicidad en programación.

En términos ontológicos, las clases son definidas basados en la noción de extensión de propiedades. El concepto de clase es extendido al de especie considerando las combinaciones de extensiones de propiedades; y aún más, a una especie natural considerando la extensión del conjunto de leyes. Así, las propiedades estáticas (atributos) y las propiedades dinámicas (leyes) son usadas en la definición de jerarquía de especies. Se puede probar que la colección de especies forma una reticulación completa bajo la inclusión. Así, el punto de vista ontológico de clases es similar al de clases en la orientación a objetos.

La noción de encapsulación es soportada de varias formas:

1. No es necesario que un objeto sepa el estado de otro objeto. También esto no es imposible, ya que un objeto

puede observar sus propios cambios de estado, resultantes de una ley de interfaz; estos cambios pueden interpretar información acerca del otro objeto.

2. No es necesario que un objeto sepa si ha forzado un cambio de estado sobre otros objetos.

3. Un objeto no tiene que cuidar que otros objetos hayan cambiado de acuerdo a sus leyes.

IV.B.6 NOTACIÓN FORMAL Y DEFINICIONES.

El conjunto de estados de un objeto *i* será denotado por $S(i)$.

DEFINICIÓN. Una ley es una función sobre el conjunto de estados:

$$L:S(i) \rightarrow S(i)$$

Un estado para el cual $L(s)=s$ será llamado un estado legal.

Un estado para el cual $L(s)\neq s$ será llamado un estado ilegal.

Mientras que el concepto de ley especifica estados permitidos, o a qué estado cambiará un estado legal, no hará mención de como cambiará de estado; esto es un concepto de aseveración no de procedimiento.

DEFINICIÓN. Sean *i*, *j* dos objetos. Una *interfaz* o ley entre objetos es una función :

$$A: S(i) \otimes S(j) \rightarrow S(i) \otimes S(j)$$

Si dos objetos no interactúan, entonces cada uno de ellos puede asumir cualquier estado (legal) en el conjunto de estados. Sin embargo, si el objeto interactúa, no todas las posibles combinaciones de sus estados (legales) serán permitidos. La idea de una interfaz es una extensión del concepto de ley que se define para un solo objeto.

A continuación se usa s^k para el estado de un objeto k (en su respectivo conjunto de estados) y se denota $A((s^i,s^j))=(s^i,s^j)$.

DEFINICIÓN. Un estado A para el cual $(s^i, s^j)=(s^i, s^j)$ será llamado un *estado estable entre objetos*. Si $(s^i, s^j)\neq(s^i, s^j)$ entonces el estado será llamado *estado inestable entre objetos*. Aún más, si $s^i\neq s^i$ diremos que el objeto j afecta al objeto i. Para $s^j \neq s^j$ diremos que el objeto i afecta al objeto j.

Para mostrar la noción de estados entre objetos y la interacción entre objetos, considere dos estaciones de trabajo, donde la estación 1 le manda señales a la estación 2. El estado de la estación de trabajo k está definido por la pareja $<I_k, O_k>$ donde I_k y O_k son: la entrada y la salida del proceso que está corriendo en la estación de trabajo k. El estado entre objetos será $(<I_1, O_1>, <I_2, O_2>)$. Asuma que existe la siguiente ley entre objetos: $I_2=O_1$. En términos de nuestra notación, esto es expresado como $A(<w,x>,<y,z>)=(<w,x>,<x,z>)$; de esta forma, si $y=x$, el estado entre objetos es estable; y si $y\neq z$ entonces es inestable.

Los dos tipos de ley, Ley de objetos y Ley entre objetos, contienen la información sobre cual estado cambiará, y cuales son esos estados a los que cambia. Así, las leyes contienen la dinámica de los objetos. La dinámica de los objetos puede ser descrita en términos de sus cambios de estado. Tales cambios son llamados eventos.

DEFINICIÓN. Una transición entre dos estados será denominada como un *evento*. Un evento será denotado por un par ordenado $e = \langle s_1, s_2 \rangle$ donde s_1 y s_2 son los estados antes y después de la transacción respectivamente.

Dado que una ley describe los posibles cambios de estado, una ley se puede definir explícitamente enumerando todos los eventos que empiezan en determinado estado legal (el estado legal no es cambiado por una ley).

Usando la notación y las definiciones anteriores, podemos describir la dinámica de la interacción de objetos. Asuma que la pareja de objetos i, j está en un estado inestable (s^i_1, s^j_1). Considere el objeto i, y asuma ahora que está en un estado legal $s^i_1 \in S_L(i)$. Suponga que debido a la interacción con el objeto j (esto es, la ley de interfaz entre i y j), i cambia al estado s^i_2. Si s^i_2 es legal, entonces no sucederá nada. Sin embargo, si s^i_2 es ilegal, entonces el objeto i cambiará su estado a un nuevo estado s^i_3 de acuerdo a su ley $\{L^i\}$. La acción del objeto j sobre i es modelada en términos de dos eventos $\langle s^i_1, s^i_2 \rangle$ debido a la ley de interfaz, y $\langle s^i_2, s^i_3 \rangle$ debido a la ley del objeto. En el caso del estado para el objeto j puede analizarse de la misma manera.

Para mostrar esto, considere el ejemplo de las dos estaciones de trabajo que están comunicadas. Asuma que la estación de trabajo k está gobernada por una ley que describe el proceso de correr en ella misma, relacionando la salida de la siguiente forma: $O_k = f_k(I_k)$. Ahora bien, si la salida de la estación de trabajo cambia, la ley entre objetos determinará un cambio en la entrada de la estación de trabajo 2. El estado $\langle I_2, O_2 2 \rangle$ puede ser ilegal, y así cambiará a un estado $\langle I_2, f_2(I_2) \rangle$.

Algunos comentarios acerca de lo anterior, en comparación con la noción común de dinámica de objetos a través de mensajes y métodos son las siguientes:

Primero: Los métodos son vistos como operaciones que encapsulan el comportamiento de un objeto. Los métodos son capaces de responder a mensajes específicos. En contraste, en el modelo actual no existe el concepto de operación. La dinámica interna de un objeto es definida mediante un concepto de aseveración, una ley. La forma en que una ley es implementada no se incluye en el modelo. Una ley es activada si el objeto se encuentra en un estado ilegal. Esto puede suceder debido a la ley de interfaz que conecta a los dos objetos; formalmente, debido al efecto de otros objetos.

Segundo: No se ha especificado ningún mecanismo para que interactúen los objetos. Sin embargo, se han implementado varios. Nuevos mecanismos son implementados dependiendo de la comunicación. Una posibilidad es asumir que un mensaje es mandado de un objeto a otro, y que este mensaje pueda cambiar el estado de éste último a un estado ilegal, de esta forma, el objeto tendrá que responder mediante un cambio de estado. Otra posibilidad es crear un área de comunicación pública, y asumir que los objetos pueden depositar información en esta área y/o leer información de esta área. En general, el paradigma de intercambio de mensajes puede ser visto como una implementación de un caso especial donde solamente un objeto cambia su estado cuando dicho estado es inestable. Sea cual fuere la implementación, el punto importante es que ambos, intercambio de mensajes y área común, son vistas como metáforas para la interacción de objetos o formas de implementación.

Finalmente, dado que no existe una relación directa entre métodos y mensajes a nivel de modelado, la noción de que los

métodos puedan responder a mensajes específicos, no ha podido ser realizado.

Sea X un conjunto de objetos. Sea $p(x)$ el conjunto de propiedades de $x \in X$. Denotamos $P(X)$ al conjunto de todas las propiedades poseídas por los objetos en **X**: $P(X)=\cup\{p(x) \mid x \in X\}$. Y denotamos por 2^X el poder del conjunto **X** (el conjunto de todos los subconjuntos de **X**).

DEFINICIÓN. La *extensión* de una propiedad P en X es el subconjunto de objetos que poseen la propiedad P :

$$G(P;X)=\{x \mid x \in X \text{ y } P \in p(x)\} 2^X$$

Recordemos que las propiedades incluyen atributos observables y las leyes. Por lo que esta definición de extensión incluye ambos. Ahora, una clase puede ser definida en términos de extensión.

DEFINICIÓN. Sea $p=\{P\}$ un subconjunto de $P(X)$. Una clase con respecto a p en X es el conjunto de todos los objetos que poseen todas las propiedades en p:

$$C(p;X)=\cap\{G(P) \mid P \in p\}$$

LEMA. Sean p_1, p_2 dos subconjuntos de $P(X)$. Entonces p_1 es un subconjunto propio de p_2 si y solamente si $C(p_2;X)$ es un subconjunto propio de $C(p_1;X)$.

TEOREMA. Sea $2^{P(X)}$ el conjunto de todos los subconjuntos de propiedades del conjunto X. Entonces el conjunto $CL=\{C(p,X) \mid p \in 2^{P(X)}\}$ es una partícula bajo la inclusión.

El lema y teorema anteriores proveen las bases para la jerarquía de clases y la herencia de propiedades. Para entender esto, considere dos conjuntos de propiedades p_1 y p_2

tales que $p_1 \subset p_2$. Denotamos el conjunto de todos los objetos que poseen la propiedad p_i como C_i. Todo objeto que posee las propiedades p_2 también posee las propiedades p_1; así $C_2 \subseteq C_1$. También considere cualquier subconjunto de C_1; todos los objetos en este subconjunto tienen propiedades p_1. En consecuencia, se puede decir que C_2 es una subclase de C_1, o alternativamente, que C_1 es una superclase de C_2. Además, suponga también que $p_3 \subset p_2$. Entonces C_3 es también una superclase de C_2. Así, la definición de clase crea una jerarquía múltiple de herencia.

La base para incluir los objetos compuestos es la siguiente: Cada par de objetos puede ser combinado a través de la operación de asociación (denotada •) sobre otro objeto. La operación es conmutativa y asociativa.

DEFINICIÓN. Sean x y y objetos tales que $x \bullet y = x, x \neq y$. entonces y es parte de x, denotada como y p-o x.

DEFINICIÓN. La composición de un objeto es igual al conjunto de sus partes.

$$C(x) = \{y \mid y \text{ p-o } x\}$$

DEFINICIÓN. x es un objeto simple si y solamente si:
y p-o $x \Rightarrow y = x$.

TEOREMA. El conjunto de todos los objetos es parcialmente ordenado bajo la relación p-o.

DEFINICIÓN. Sea $P \in p(x)$ una propiedad del objeto x. Entonces P es una propiedad resultante o hereditaria si existe $y \in C(x)$ tal que $P \in p(y)$, $y \neq x$. Por otro lado, P es una propiedad emergente si para todo y tal que $y \in C(x)$ y $y \neq x$, $P \notin p(y)$.

La noción de propiedad emergente nos permite atribuir a un objeto propiedades generales. Recordemos que un objeto puede estar solamente en un estado legal, y que el estado legal es determinado por la ley del objeto. Debido a que los objetos pueden tener algunas variables que son emergentes y otras que son resultantes, la ley del objeto establecerá una liga entre estas dos propiedades:

Las propiedades emergentes de un objeto son ligadas por las leyes de los objetos a las propiedades de los objetos que lo componen.

Aún más, asumimos que todas las propiedades de un objeto compuesto están relacionadas a las propiedades de los objetos que lo componen.

Sea P una propiedad emergente del objeto x. Entonces existe una ley definida sobre x que determina en forma única el valor de P para toda combinación permitida de valores de las propiedades del objeto en la composición $C(x)$ de x.

En este sentido, cada variable de estado de un objeto compuesto está relacionada a un objeto en su composición, o es una función para las variables de estado que son relacionadas al objeto en su composición. Esta percepción de un objeto compuesto puede ser interpretada como una generalización del concepto de variable de instancia como objetos. Más aún, partiendo de que las variables de estado emergentes y las variables de estado resultantes parecen ser parte del estado, el objeto que interactúa con otro objeto dado, no tiene que cuidar la composición del otro objeto. Esto puede ser interpretado como la encapsulación e independencia.

CAPITULO V

INTEGRACIÓN DEL LENGUAJE Y BASE DE DATOS

V.A. PROCESOS - DATOS.

Las metodología y herramientas para diseño y desarrollo de aplicaciones hacen una división tajante entre procesos y datos. Inclusive, entre las herramientas CASE se maneja este esquema de separación: Procesos - Datos; para poder así dividir el trabajo de programas, del manejo de datos.

Sin embargo, al hablar en términos orientado a objetos, el concepto de datos cambia, ya que existen variables únicamente visibles al objeto que la posee; en este caso, surge la pregunta ¿Donde deberán ser almacenadas estas variables, en el lenguaje o en la base de datos ?.

Los lenguajes de programación, junto con el compilador, ha sido considerada una disciplina separada de las bases de datos en el campo de los sistemas operativos. Se considera conjuntar estas dos especialidades en una integración de software paulatina, necesaria y benéfica para crear aplicaciones más robustas. Profundizando un poco más:

Es importante notar que el interés está en las aplicaciones enfocadas a los datos: aquellas en las que el acceso y mantenimiento a los datos es crítico. Esto cubre un amplio rango de programas útiles a los usuarios finales, pero excluye a muchas aplicaciones en tiempo real, como sistemas de control, los cuales tienen la función principal de observar y responder a eventos del mundo real. Por otro lado,

existen muchas aplicaciones numéricas enfocadas a efectuar cálculos como resultado final y no en el almacenamiento de información.

El esquema de separación Procesos - Datos es útil para identificar los programas (procesos) y determinar la información que se empleará en la aplicación (datos). Una vez realizado lo anterior, los datos son analizados dependiendo del manejador de base de datos a utilizar con el propósito de definir la base de datos empleando instrucciones SQL o el lenguaje de definición de datos (DDL).

Ahora bien, la metodología Orientada a Objetos propicia un mejor modelado de la realidad, proporcionando recursos que permiten organizar y modificar las aplicaciones; dejando en cierta obsolescencia algunos términos como CASE, Ingeniería de Software, programación estructurada [KARIB95].

Gran parte de esto es motivado por el hecho de que el lenguaje orientado a objetos solo hace uso de la base de datos orientada a objetos sin necesidad de definir una estructura para la base de datos. De hecho los lenguajes orientados a objetos Borlan C++ y Microsoft C++ ofrecen una biblioteca de Clases; solamente haciendo uso de ellas al momento de programar. Lo anterior rompe con el esquema tradicional Procesos - Datos.

V.B. CONVERTIDORES DE OBJETOS

Otro punto importante es que los manejadores de bases de datos convencionales fueron desarrollados para almacenar construcciones simples de tipos de datos, como texto, números, fechas, arreglos, apuntadores. Estos sistemas no poseen medios para el manejo de una variedad infinita de tipos de datos, los cuales las herramientas orientadas a

objetos si lo permiten. Más aún, no contemplan almacenar métodos propios de los objetos.

Una forma de solucionar esto, es agregar una capa entre el Lenguaje Orientado a Objetos y el Manejador de base de datos, que convierta los objetos en estructuras más simples que el manejador pueda soportar [TAYLOR91]. Esta es una de las principales aportaciones de la Orientación a Objetos en el mundo comercial. Ya que solamente es necesario un módulo intermedio (Conector de Base de Datos) para poder accesar diferentes Bases de Datos relacionales en una misma aplicación.

Otra forma de solucionarlo es usar un nuevo manejador de base de datos que pueda almacenar toda la información relacionada a los objetos; esto es, variables de instancia, clases, métodos, etc. Sin que exista la necesidad de desarmar el objeto, almacenarlo y armarlo nuevamente al momento de usarlo. Esta nueva clase son los manejadores de bases de datos Orientados a Objetos.

V.C. ESTRUCTURA DE LOS MANEJADORES DE BASES DE DATOS ORIENTADOS A OBJETOS.

Ante este planteamiento, quedan las siguientes disyuntivas:

- ¿Que modelo de datos se empleará; Modelo del Umbral/Referencia o el Manifiesto de los Sistemas Orientados a Objetos?

- Independientemente del Modelo: ¿Que Bases se considerarán para construir el manejador de base de datos; los principios Ontológicos o se partirá de bases existentes como el Modelo Relacional Extendido y el Modelo Semántico de Datos?

- ¿Habrá que construir un nuevo manejador de almacenamiento?

Aunado a lo anterior , existen conceptos que caen en el ámbito de los Lenguajes y las Bases de Datos considerándolos Orientados a Objetos; por ejemplo:

La función básica de una Clase, es definir un particular tipo de objeto. Y la clase puede contener cualquier número de instancias únicas de la clase. La diferencia entre una instancia de una clase (Objeto) y otra instancia de la misma Clase, son los valores de las variables.

Cuando el objeto recibe un mensaje, busca en su clase el método o variable que aplica a su propio conjunto de valores. Si no lo encuentra, busca en la superclase inmediata superior. Si no lo encuentra, busca en la superclase superior. Y así sucesivamente hasta que encuentra el método o la variable; esto es, una *búsqueda en la base de datos*.

Sin embargo, esta búsqueda es lenta, ya que puede recorrer todas las clases para localizar una variable. Por lo que existe otra técnica, que establece una *liga directa* al método o variable, eliminando la búsqueda en la base de datos.

V.D. MUNDO CIENTÍFICO Y MUNDO COMERCIAL.

Esta simbiosis es tema de grandes estudios. Aplicado a este tema, es importante hacer notar bajo que ambiente se está tratando. Un ejemplo es el siguiente: El lenguaje de programación SIMULA surgió de una inquietud científica, aportando la Orientación a Objetos; que ahora ha trascendido en una competencia comercial, la cual tiene que ser respaldada en una investigación científica.

Otro ejemplo: En el mundo comercial al postular que un producto de software es Orientado a Objetos implica cumplir con las premisas de la Orientación a Objetos (Modelo Umbral/Referencia, Manifiesto), que a su vez emergen del mundo científico; sin embargo no se han limitado las premisas para determinar la Orientación a Objetos, ya que las bases (Principios Ontológicos) no han sido probadas satisfactoriamente por la comunidad de investigadores [MARCOS95].

Para lograr lo anterior; habrá que establecer un estándar que en ocasiones es determinado por el mundo comercial (por ejemplo: SQL, Microsoft).

Bajo un ambiente científico se hace referencia a un ambiente Orientado a Objetos en términos de las premisas que se cumplen. De esta forma si tiene sentido decir que un producto de software es mas Orientado a Objetos que otro.

También se mencionan las características Orientadas a Objetos implantadas en el producto, y bajo que rama del conocimiento se está hablando (Sistemas Operativos Orientados a Objetos, Redes Orientadas a Objetos, Lenguajes Orientados a Objetos, Bases de Datos Orientados a Objetos, etc.).

En el ambiente comercial se hace mención de productos basados en objetos, los cuales solo cumplen con algunas características Orientadas a Objetos. Más Aún, existen productos que argumentan la interfase gráfica como premisa de Orientación a Objetos.

Por otro lado, para poder determinar si un producto de software es orientado a objetos, habrá que probarlo contra cada una de las premisas de la orientación a objetos.

Pero: ¿Cuáles premisas?, ¿Las del Modelo de Referencia? o ¿ Del Manifiesto?. Es importante estandarizar las características, premisas y bases que determinan si un producto es Orientado a objetos o no lo es. Sin embargo, aún no han fructificado los intentos por establecer estos estándares.

V.E. INTEGRACIÓN DE SOFTWARE.

Uno de los principales beneficios de la estandarización, es contar con los elementos suficientes para integrar: Lenguajes, Manejadores de Bases de Datos, Sistemas Operativos y Redes, todos ellos Orientados a Objetos.

Esta integración puede ser sencilla; sin embargo, sus repercusiones son inmediatas, ya que, por ejemplo, desaparecería el concepto Cliente/Servidor, que en esencia es un término Orientado a Objetos.

En el mercado ya existen productos tendientes a integrar estos ambientes Orientados a Objetos. En este caso, se encuentra NEXT que cuenta con un Sistema Operativo (NEXTSTEP), una Base de Datos (NeXT DATABASE KIT), y herramientas de desarrollo (NEXTSTEP DEVELOPER). Aquí, el lenguaje y la base de datos comparten bibliotecas en común, todo bajo un mismo ambiente de trabajo; pudiendo acceder a bibliotecas almacenadas en Sybase y en Oracle.

V. F. UNIFICACIÓN DEL LENGUAJE Y EL MANEJADOR DE LA BASE DE DATOS

La principal ventaja de considerar un lenguaje y manejador de base de datos orientado a objetos a la vez, es

que ambas partes están basadas en el mismo modelo. Aún así existen diferentes problemas a solucionar:

Estos problemas caen dentro de tres categorías: Primero, se tiene que determinar la funcionalidad del sistema, esto es como va a actuar y que funciones soportará. Segundo, se tiene que determinar el momento en el cual se obtiene un rendimiento de la aplicación aceptable. Tercero, existen problemas de aceptación de sistemas que van más allá de temas técnicos [CATELL 91].

Uno de los principales temas es el modelo de datos. Existen lenguajes que soportan la definición de clases y otros lenguajes que soportan la definición de tipos. Al definir una jerarquía y herencia, esta deberá ser congruente con el modelo del manejador de la base de datos. Otro problema es causado por los tipos de variables, en los lenguajes orientados a objetos, la mayor parte de los tipos de variables son tomados de los manejadores de bases de datos ya existentes; por otro lado, los manejadores de bases de datos postulan nuevas estructuras para el manejo de variables.

Unificar el mismo modelo de datos no es suficiente, otra diferencia entre el lenguaje y el manejador de base de datos puede vislumbrarse con la siguiente pregunta: ¿Como deberá aplicarse el paradigma de objetos: procedural, lógica de programación, ecuaciones, funciones, basado en reglas, como producto de un sistema, o habrá que encontrar una forma en la cual se puedan tratar con cada uno de los anteriores de manera clara y efectiva ?. Este tema hasta ahora ha sido considerado exclusivo del lenguaje orientado a objetos; sin embargo, al hablar de la integración lenguaje-base de datos, los manejadores de base de datos tendrán que abordarlo.

Otro motivo por el cual deben estar integrados lenguaje y base de datos lo expondré bajo una secuencia de eventos, considere:

1.Un objeto posee variables de instancia que solamente ése objeto puede ver.

2.Este objeto, recibe un mensaje.

3.El mensaje provoca la ejecución de una o varias funciones propias del objeto.

¿ Como deberán ser leídas las variables de instancia del objeto, en la base de datos para ejecutar la función ?

Pueden considerarse dos alternativas: dinámica o estáticamente. Analicemos cada una de ellas:

Estáticamente: Para que la lectura sea estática, en el momento que se activa una función, son leídas las variables de instancia. Sin embargo, esto afecta la concurrencia, ya que durante la ejecución de la función pueden ser alterados los valores de las variables de instancia en el manejador de base de datos. Esto afecta el rendimiento de la aplicación, ya que al finalizar la función, se tendrá que revisar si el valor de las variables de instancia no han sido afectados.

Dinámicamente. Aquí, la lectura de las variables de instancia se efectúan en el momento que son usadas dichas variables, por lo que el lenguaje tendrá que contemplar el uso del valor de las variables de instancia, y no bajo el esquema de lectura, modificación y almacenamiento del valor de la variable. Además el manejador de la base de datos deberá contemplar el mejor y más rápido acceso a la información relacionada con el objeto. Repercutiendo nuevamente en el rendimiento de la aplicación.

Es necesario tener un ambiente Orientado a Objetos que soporte a los objetos en el momento de ejecución, un ejemplo de los problemas latentes son los siguientes [WON 89] :

- *Persistencia*: Salvar automáticamente a los objetos entre procesos; usando, ya sea espacio de memoria virtual persistente o mediante el manejador de base de datos.

- *Recolección de Basura*: Borrado automático a objetos no referenciados.

- *Concurrencia*: Acceso concurrente múltiple; esto es, control de concurrencia y comunicaciones mediante el paso de mensajes o el manejador de base de datos.

- *Distribución*: Nombres de objetos a nivel Global, paso de mensajes en forma remota y métodos de invocación de objetos.

- *Seguridad*: manejo de propietarios de objetos, derechos de acceso, o permisos para poder invocar a un objeto.

Dependiendo de la naturaleza de las aplicaciones, estas funcionalidades deben ser suministradas ya sea por el sistema operativo o por el manejador de base de datos Orientado a Objetos. Esto es, en un ambiente el cual esta formado por una gran cantidad de objetos y concurrencia, deberá centrarse en soluciones a los problemas de almacenamiento, consulta, indexación, optimización, etc. Para otras aplicaciones, como integración de sistemas, habrá que tratar con problemas de 2 filosofías de trabajo, como por ejemplo: una estructura de base de datos relacional y los procesos orientados a objetos.

Otro tema importante es el concerniente a: ¿ Hasta que momento es conveniente mantener la barrera entre procesos y datos ?. Es decir, actualmente el diseño y desarrollo de sistemas está basado en el esquema de separación entre los procesos o programas, y datos o bases de datos. Sin embargo, el concepto Orientación a Objetos cambia radicalmente esta idea, ya que existe información inherente a los objetos que el usuario (programador) no puede accesar.

V.G. PRODUCTOS DE SOFTWARE ORIENTADOS A OBJETOS

Basados en el modelo de datos, podemos encontrar los siguientes prototipos y productos:

Relacional Extendido: Podemos encontrar los siguientes productos: POSTGRES, STARBURST, INGRES V6.3, SYBASE, PENGUIN.

Modelo Funcional: Como por ejemplo: IRIS, PROBE, VISION, SIM.

Orientado a Objetos: Como: ONTOS, ObjectStore, O2, GemStone, ORION, VERSANT, STATICE.

A continuación se describe la información más relevante acerca de algunos de estos productos:

POSTGRES : 1986. Desarrollado en la Universidad de California en Berkeley. Basado en el modelo relacional extendido. Provee de las siguientes características: objetos, identificadores de objetos, objetos compuestos, herencia múltiple, histórico de datos, procedimientos y un lenguaje de consulta.

STARTBURST : 1986. Desarrollado en el centro de investigación de IBM en Almaden. Basado en el modelo relacional extendido. Comentarios importantes de este sistema son: Un lenguaje de consulta basado en una extensión del álgebra relacional, permitiendo consultas recursivas. No provee de encapsulación de tipos en procedimientos, resolviendo esto mediante el manejo de privilegios para usuarios específicos. Permite al usuario definir métodos de acceso y almacenamiento de datos. Permite al usuario extender el lenguaje de consulta mediante el uso de tablas de funciones.

IRIS : 1988. Desarrollado en los laboratorios de Hewlett-Packard en Palo Alto, California. Basado en el modelo funcional. Sin embargo, para el almacenamiento de información emplea el modelo relacional extendido. Comentarios importantes de este son los siguientes: Las funciones pueden ser definidas explícitamente. Posee un lenguaje de consulta particularmente dirigido a objetos. IRIS puede ser independiente a la orientación a objetos en el sentido que los objetos solamente pueden ser manipulados usando funciones (excluyendo los mensajes) y las funciones pueden ser agrupadas con los tipos de objetos sobre los cuales operan. No provee ningún mecanismo de encapsulación, y la sintaxis en la notación es funcional en lugar de orientada a objetos.

PROBE : 1985. Desarrollado por el CCA (Computer Corporation of America) en Cambridge Massachusetts. Basado en el modelo funcional. Es la continuación de las investigaciones previas sobre DAPLEX. Aquí, los objetos son identificados únicamente por el identificador de objetos generado por el sistema.

ONTOS : 1989. Es un producto comercial de ONTOLOGIC. Este substituye todos los trabajos previos de ONTOLOGIC, como Vbase. Basado en un modelo Orientado a Objetos. Incorpora los estados de objeto: desactivos (almacenados en disco) y activos (cuando el programa hace uso de ellos). Además, incorpora una variante de SQL haciendo a los atributos del objeto visibles al usuario.

ObjectStore : 1990. Producto comercial de Object Design. Implementado en C++ . Provee de identidad del objeto, herencia múltiple, una librería de tipos, incluyendo conjuntos y listas. Posee un lenguaje para manipulación de datos (DML) el cual tiene que ser ligado usando C++ . Maneja concurrencia mediante 2 métodos: Concurrencia de corta duración implementado con bloqueo de dos fases a nivel página; y Concurrencia de larga duración, basados en mecanismos de control de versiones.

O2 : 1988. Es un prototipo Orientado a Objetos que ha sido llevado al mundo comercial. Desarrollado por GIP Altair, en Le Chesnay, Francia. Es un Manejador de Base de Datos Orientado a Objetos exclusivamente, soporta identidad del objeto, encapsulación, herencia múltiple, ligado dinámico, listas, conjuntos, transacciones y un lenguaje de consulta global. O2 puede ser usado con una extensión orientada a objetos de C, denominada CO2 .

GemStone : 1988. Producto comercial de SERVIO Corporation. Es un Manejador de Base de Datos Orientado a Objetos. Originalmente fue un intento por incorporarlo a SmallTalk. Sin embargo, ahora usa OPAL como lenguaje de consulta y programación; y se ha integrado a C++ . La

arquitectura de GemStone incluye 2 procesos : GEM provee los mecanismos necesarios para integrarse a los lenguajes de programación, OPAL y C++ en este caso. STONE es el manejador de datos, controla concurrencia, uso de disco, autorizaciones a usuarios, transacciones y servicios de recuperación de información.

ORION : 1988. Prototipo llevado al ambiente comercial. Desarrollado por ITASCA Systems en Mineapolis, Minesota. También es conocido como ITASCA. Es un Manejador de Base de Datos Orientado a Objetos implementado en LISP. Se han liberado varias versiones de este producto ya que inició como un proyecto de investigación. Incorpora identificadores de objetos, herencia múltiple, objetos compuestos, indexación, transacciones y distribución.

Algunos productos de Software Orientados a Objetos que han surgidos y no han dejado ver las bases o el enfoque sobre el cual están construidos son:

Power Builder : Producto de PowerSoft. Herramienta de desarrollo que emplea conceptos de Orientación a Objetos, accesando información de bases de datos relacionales.

Delphi : Producto de Borland. Herramienta de desarrollo de aplicaciones que emplea un conector de base de datos para accesar diferentes bases de datos.

NEXT : Compañia que está creando todo un ambiente orientado a objetos, tendiente a emplear sistemas abiertos; esto es, poder convivir con cualquier producto ya sea de software como hardware.

CONCLUSIONES

El presente trabajo muestra los fundamentos de la Orientación a Objetos, características de los Manejadores de Bases de Datos, una de las bases de la Orientación a Objetos (principios Ontológicos), y la integración los lenguajes Orientados a Objetos con los Manejadores de Bases de Datos.

Sea cual fuere el análisis realizado, puede observarse que los nuevos Sistemas Orientados a Objetos están lo suficientemente avanzados para revolucionar la investigación y desarrollo en las diferentes ramas de la computación (DBMS, CASE, Lenguajes, etc.).

Aún no puede hablarse de Manejadores de Bases de Datos Orientadas a Objetos comerciales. Sin embargo, si tiene sentido hablar de Bases de Datos Orientadas a Objetos. Muchos productos de Software almacenan información en los Manejadores de Bases de Datos Relacionales, haciendo transparente para el usuario (programador) la ubicación de los datos o información; haciendo uso de ella solamente.

Así, algunas investigaciones están dirigidas hacia el establecimiento de puentes entre los Manejadores de Bases de Datos y los Sistemas Orientados a Objetos , de manera que proporcionen ambientes productivos.

Los Manejadores de Bases de Datos Orientados a Objetos se enfrentan a 2 factores importantes: tendencias en la investigación, y tendencias en el mercado.

La tendencia en la investigación es sentar las bases para crear un producto lo suficientemente robusto: esto es, los principios sobre los cuales se sustentan los manejadores de bases de datos, presentan ciertas debilidades; estas son principalmente en la unificación de criterios para abarcar una amplia gama de aplicaciones, como pueden ser las bases de datos de conocimientos y bases de datos de información. Existen intentos por crear dicha unificación, sin embargo no han sido aceptados totalmente.

La tendencia en el mercado es un factor decisivo. La comunidad acepta los sistemas Orientados a Objetos; sin embargo ante el manejo de datos (Bancos de Información), los usuarios prefieren integrar información, que centralizarla.

Por otro lado, los nuevos modelos (Relacional Extendido, Funcional, Orientado a Objetos) generalmente proveen una funcionalidad muy similar. De hecho, el término "Modelo Semántico de Datos" ha sido usado para incluir todos los modelos.

Una observación importante acerca de esta confusión entre modelos de datos, es que todos los modelos están convergiendo lentamente en un solo modelo que combina todas sus características. Convirtiendo el modelo de datos en una forma de distinguir el origen del sistema que se está tratando.

También es importante notar que el modelo de datos no es lo único que distingue a un sistema. De hecho existen sistemas que no poseen un modelo de datos, o permiten que el usuario elija su propio modelo. Los sistemas difieren de

acuerdo a la arquitectura de la base de datos, la cual es la forma en que es implementado e integrado con el lenguaje de programación de la aplicación.

Algo si es claro, los Manejadores de Bases de Datos Orientados a Objetos están siendo muy populares. La investigación a cerca del tema avanza muy rápido, además está echando raíces para lanzarse a los ambientes productivos y llenar las necesidades que los Manejadores de Bases de Datos tradicionales no han sido capaces de satisfacer.

REFERENCIAS BIBLIOGRÁFICAS

[CATELL 91] OBJECT DATA MANAGEMENT
Object-Orientes and Extended Relational Data Base Systems
R.G.G. Cattell
De. Addison-Wesley Publishing Co. 1991

[KARIB95] Guadalupe Ibargüengoitia
Orientación a Objetos. Una oportunidad para la Empresa y la Academia
SOLUCIONES AVANZADAS 15-Jun-1995
Una entrvista con el Dr. Miguel Karib Mora

[MARCOS95] Marcos Calderón Macías
Una Visión General de los Sistemas Manejadores de Bases de Datos Orientados a Objetos
SOLUCIONES AVANZADAS 15-Jun.-1995

[MCLEOD 90] RESEARCH FOUNDATIONS IN OBJECT-ORIENTED AND SEMANTIC DATABASE SYSTEMS
Alfonso F. Cardenas; Dennis McLeod
Prentice Hall International 1990

[MEYER 88] OBJECT-ORIENTED SOFTWARE CONSTRUCTION
Bertrand Meyer
Prentice Hall International 1988

[OFELIA 93] Ofelia Cervantes de Poty
"Bases de Datos Orientadas a Objetos"
SOLUCIONES AVANZADAS Año I Número 6
Bases de datos Orientadas a Objetos

[SUBODH 94] OBJECT-ORIENTED NETWORKS
Models for Architecture, operations and Managment
Subodh Bapat
De. Prentice Hall 1994

[TAYLOR91] OBJECT ORIENTED TECHNOLOGY
David A. Taylor
Addison-Wesley Publishing Company 1991

[WON 89] OBJECT-ORIENTED CONCEPTS, DATABASES AND APPLICATIONS
Won Kim , Frederick H. Lochovsky
Addison-Wesley Publishing Company 1989

[ZDONIK 90] READINGS IN OBJECT ORIENTED DATABASE SYSTEMS
Atanley B. Zdonik y David Maier
Ed. Morgan Kaufmann 1990

[ZICARI 90] OBJECT ORIENTED DATABASE SYSTEM: THEORY AND PRACTICE
ta. Escuela de Invierno. Morelia Mich 1990
Robert Zicari