UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

LA ORIENTACION A OBJETOS EN EL DISEÑO DE SISTEMAS DE BASE DE DATOS TRADICIONALES

T E S I S

QUE PARA OBTENER EL TITULO DE:

M A T E M A T I C O

P R E S E N T A:

ALVARO CASTILLO MARTINEZ

MEXICO, D. F.

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

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:

"LA ORIENTACION A OBJETOS EN EL DISEÑO DE SISTEMAS DE BASE DE DATOS TRADICIONALES"

realizado por ALVARO CASTILLO MARTINEZ

con número de cuenta 8955410-4 **, pasante de la carrera de** MATEMATICO

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis Propietario	M. EN C. ELISA VISO GUROVICH
Propietario	M. EN C. VIRGINIA ABRIN BATULE
Propietario	M. EN C. JAVIER GARCIA GARCIA
Suplente	DR. MARIO MAGIDIN MATLUK
Suplente	M. EN C. CLAUDINE COURT MANUEL

Consejo Departamental de Matemáticas
MAT. CESAR GUEVARA BRAVO

A mis padres.

Agradezco a las personas que me brindaron su apoyo en la realización de este trabajo, especialmente a Elisa por su paciente dirección, y a Kitten, por su eterna cercanía y por haberme revelado, sin saberlo, la dualidad compleja de la vida.

ÍNDICE

0.1 INTRODUCCIÓN

La Facultad de Ciencias de la UNAM requiere de un sistema de control escolar para el registro de reinscripciones. La reinscripción es el proceso que el alumno realiza a partir del segundo periodo lectivo dentro del plantel para quedar registrado como inscrito en las asignaturas autorizadas dentro de los grupos asignados de su preferencia, cumpliendo así parte de un plan de estudios. Esta facultad cuenta con 5 carreras y para cada materia se ofrecen varios grupos en distintos horarios a los que los alumnos se pueden inscribir hasta completar el cupo. Adicionalmente, algunas materias tienen ciertos prerrequisitos que el alumno debe cumplir. Para inscribirse, los alumnos llenan una solicitud conocida como "tira de materias" donde anotan sus datos personales (nombre y número de cuenta) y las materias a las que se desean inscribir, indicando el grupo de su preferencia. El sistema debe ser capaz de comprobar que hay cupo en el grupo escogido, así como verificar que, en su caso, el alumno haya acreditado las materias que son prerrequisito, además de hacer otras comprobaciones antes de inscribirlo, mismas que tienen su origen en el "Reglamento General de Inscripciones de la UNAM". Después de inscribirse, los alumnos reciben un comprobante de inscripción. Al finalizar el periodo de inscripción el sistema debe generar las listas de los alumnos inscritos en cada grupo de cada materia indicando el profesor, así como el número de alumnos inscritos por grupo.

Hasta ahora la encargada de los procesos de inscripción es la Dirección General de Administración Escolar (DGAE), la cual recibe la información mediante las llamadas hojas de lectura óptica, cuyos ovalos llena el alumno. Esto, como demuestra la experiencia obtenida, es fuente de muchos errores, los cuales se verán minimizados con la implantación del sistema de reinscripciones.

En el presente trabajo se hace un análisis y una comparación en la elaboración de este sistema desde 2 metodologías distintas, la modular y la orientada a objetos.

0.2 CONCEPTOS BÁSICOS

En los primeros años del procesamiento de datos, durante los años 50's, sólo existía el procesamiento secuencial de archivos. Durante los 60's, con el almacenamiento de acceso directo en disco, el procesamiento de archivos por acceso aleatorio se hizo factible y popular. A mediados de los 70's se introdujeron los primeros sistemas de bases de datos basados en una estructura jerárquica de los datos. Estos sistemas proveían la recuperación de múltiples registros asociados con un sólo registro de otro archivo. Poco después, se desarrollaron sistemas de bases de datos basados en un modelo de red, los cuales soportaban significativamente más relaciones complejas entre registros de diferentes archivos. Ambos, los modelos de bases de datos jerárquico y reticular requerían el uso de apuntadores predefinidos para ligar registros relacionados.

En 1970, el trabajo de Codd en el modelo de datos relacional revolucionó la industria de las bases de datos. Su enfoque se refería al acceso y manipulación de datos únicamente en términos de sus características lógicas..

Actualmente, un sistema de bases de datos en una organización grande consiste de hardware, software, datos y gente. La configuración del hardware comprende una o más computadoras, terminales, impresoras, unidades de disco y otros dispositivos físicos. El software incluye un sistema manejador de bases de datos (DBMS) y programas de aplicación que usan el DBMS para accesar y manipular la base de datos. Los datos residen físicamente en disco pero están lógicamente estructurados de tal forma en que se puedan accesar fácil y eficientemente.

Un sistema de bases de datos es pues, esencialmente, un sistema de mantenimiento de registros basado en computadoras, es decir, un sistema cuyo propósito general es registrar y mantener información. Sus componentes principales son datos (divididos en varias bases de datos), hardware (parte física donde residen las bases de datos), software (interfaz entre el hardware y los usuarios del sistema, denominada comúnmente "Sistema Manejador de Bases de Datos" o DBMS) y los usuarios en sí.

0.3 DISTINTOS MODELOS

Un sistema de bases de datos debe poder representar entidades (cualquier objeto distinguible del cual se registran datos) y asociaciones que vinculan esas entidades básicas y que forman parte de los datos de operación tanto como las entidades asociadas. De hecho una asociación se puede pensar como una clase particular de entidad.

Actualmente existen cuatro enfoques principales que permiten ver y manipular las asociaciones: el relacional, el jerárquico, el de red y el orientado a objetos.

De manera somera se podría decir que en el enfoque relacional las asociaciones se representan como tuplas de relaciones, mientras que en los enfoques jerárquico y de red las asociaciones se representan por medio de "ligas", las cuales son capaces de representar asociaciones de uno a muchos. En el enfoque de red las ligas pueden combinarse para modelar asociaciones más complejas de muchos a muchos, mientras que esto no es posible en el enfoque jerárquico.

Por último, el enfoque orientado a objetos, que es relativamente reciente, captura lo esencial de los lenguajes orientados a objetos. Es usado en los sistemas menejadores de bases de datos orientados a objetos (OO-DBMS). El término OO-DBMS describe una clase de sistemas con la capacidad de un DBMS y con un lenguaje que tenga las características siguientes:

- *Objetos complejos*; esto es, la habilidad de definir tipos de datos con una estructura anidada.
- *Encapsulamiento*; esto es, la habilidad de definir procedimientos aplicables sólamente a objetos de un tipo particular y la habilidad de que sólo mediante estos procedimiemtos se pueda accesar dichos objetos.
- *Identidad de objeto*; es decir, la habilidad del sistema de distinguir entre 2 objetos que "parecen" iguales en el sentido de que todos sus componentes de tipo primitivo tienen los mismos valores.

Un sistema que maneja encapsulación y objetos se dice que maneja tipos de datos abstractos (ADTs) o clases. Dicho de otra forma, una clase es una definición de una estructura junto con las definiciones de las operaciones por las que pueden ser manipulados los objetos de esa clase.

Además, en este modelo se da la jerarquía de tipos, es decir, se deja que los tipos tengan subtipos con características especiales. Los tipos de datos se construyen por formación de registros y de conjuntos. Por ejemplo, una tupla es construida de tipos primitivos (enteros, caracteres, etc.) por formación de registros, y una relación se construye de tuplas por formación de conjuntos, de tal suerte que una relación es un conjunto de tuplas con un formato de registro particular. La noción básica en este modelo es la de clases, subclases, jeraquías de clases y métodos, que son operaciones a realizar en los objetos con la estructura de esa clase.

0.4 CONCEPTOS GENERALES DE LOS DBMS

Concretamente, un Sistema Manejador de Bases de Datos (DBMS) es un programa para manejar un conjunto de datos, a este conjunto de datos se le llama *base de datos*, y es almacenada en uno o más archivos. Algunas de las razones por las que se usan los DBMS son las siguientes:

- *Protección de datos.* La base de datos está protegida de eventuales fallas del hardware y algunos errores de usuario.
- *Compartición entre usuarios.* Múltiples usuarios pueden accesar la base de datos al mismo tiempo.
- *Compartición de aplicaciones.* Múltiples programas de aplicación pueden leer y escribir datos a la misma base de datos.
- *Seguridad.* Los datos se pueden proteger contra los accesos de lectura y escritura no autorizados.
- *Extensibilidad.* Los datos se pueden añadir a la base de datos sin causar conflictos con los programas existentes, y pueden ser reorganizados para un desempeño más rápido.
- *Distribución de datos.* La base de datos puede ser particionada en varios sitios, organizaciones y plataformas de hardware.

Para muchas aplicaciones de bases de datos, la tarea más importante y difícil es el diseño de la base de datos. El diseño del código de programación relacionado usualmente es más fácil. Se debe diseñar una base de datos por muchas de las mismas razones por las que se debe diseñar cualquier programa de software: el diseño cuidadoso del software antes de la codificación mejora la calidad y reduce el costo. Un diseño de una base de datos es también referido a veces como un *modelo de datos* o *esquema.*

En general hay dos enfoques del diseño de bases de datos. El primero está basado en atributos: reunir una lista de atributos relevantes a la aplicación y sintetizar grupos de atributos que preserven dependencias funcionales. El otro enfoque está basado en entidades: descubrir entidades que sean significativas a la aplicación y describirlas.

0.5 CONCEPTOS GENERALES DE LOS RDBMS

El *modelo relacional* está basado en un concepto simple: la *tabla* o *relación.* Una base de datos relacional es básicamente una base de datos en la que los datos se perciben como tablas, el término "relación" es un término matemático para una tabla que cumple con ciertas propiedades. No obstante, es usual tratar como sinónimos los términos tabla y relación. Los sistemas manejadores de bases de datos relacionales (RDBMS) son programas de software que se usan para manejar estas tablas.

Un RDBMS contiene 3 partes principales:

- Datos que se presentan como tablas.
- Operadores para manipular las tablas.
- Reglas de integridad en las tablas.

0.5.1 Estructura lógica de un RDBMS

Una base de datos relacional aparece lógicamente como una colección de tablas, las cuales tienen un número específico de columnas y de renglones o tuplas.

La teoría de bases de datos relacionales dicta que a cada atributo debe asignársele un dominio. Un *dominio* es un conjunto de valores legales. Los dominios tienen más información que tan sólo un formato de datos y permiten un mayor cotejo de semántica. Por ejemplo, los dominios se pueden usar para prevenir operaciones en atributos incompatibles, como sumar un costo de dinero a un peso de masa. Desafortunadamente, la mayoría de los RDBMS no soportan los dominios, sino simples formatos de datos como números, fechas y cadenas de caracteres.

Cada valor en una tabla debe pertenecer al dominio de su atributo o ser nulo. *Nulo* significa que un valor de atributo es desconocido o simplemente no es aplicable. Actualmente hay muchas complicaciones teóricas concernientes a los valores nulos que frecuentemente causan problemas en las aplicaciones reales.

0.5.2 Operadores del RDBMS

El lenguaje propio del RDBMS provee operadores para manipular las tablas. Estos operadores permiten crear tablas, insertar renglones en las tablas, borrar renglones de las tablas, y otras funciones más.

0.5.3 Integridad

Un aspecto importante de los RDBMS es el soporte de la integridad de los datos. Los dos aspectos generales de la integridad en el modelo relacional son la integridad de entidades y la integridad referencial. La *integridad de entidades* establece que cada tabla debe tener exactamente una llave primaria que no permita atributos nulos. Una *llave primaria* es una combinación de uno o más atributos cuyos valores ubican de manera única a cada renglón en la tabla, y la cual se escoge de entre posiblemente varias *llaves candidatas*, es decir, llaves que cumplen con estos requisitos. La *integridad referencial* requiere que el RDBMS mantenga cada llave foránea consistente con su correspondiente llave primaria. Una *llave foránea* es una llave primaria de una tabla que está incrustada en otra (posiblemente en la misma) tabla. Algunos autores consideran a los dominios como una tercera característica general de la integridad relacional, la cual establece la *integridad de atributos*, que requiere que cada valor de un atributo se obtenga de su propio dominio.

0.5.4 Formas normales

Las formas normales son reglas desarrolladas para evitar inconsistencias lógicas en las operaciones de actualización de las tablas. Las formas normales no se deben violar inadvertidamente, salvo en casos en los que se tenga una buena razón, como un mejor desempeño.

Una tabla está en *primera forma normal* cuando cada valor de atributo no contiene un grupo repetitivo, es decir, un valor de atributo no puede contener un conjunto de valores.

Una tabla está en *segunda forma normal* cuando satisface la primera forma normal y cada renglón tiene una llave primaria. Cada campo que no sea llave primaria debe depender totalmente de la llave primaria.

Una tabla está en *tercera forma normal* cuando satisface la segunda forma normal y cada atributo que no sea llave primaria depende directamente de la llave primaria, es decir, no debe haber dependencias de la llave primaria transitivas.

§ 1

TÉCNICAS DE DISEÑO MODULAR TRADICIONAL

El diseño modular, también llamado estructurado o de descomposición funcional, es una técnica para descomponer programas de computadora en módulos independientes. El concepto es simple: diseñar un programa como una jerarquía descendente ("top-down") de módulos. Un módulo es un grupo de instrucciones, un párrafo, bloque, subprograma o subrutina. Estos módulos son diseñados con el fin de que lleven a cabo una y sólo una función. Se han desarrollado escuelas separadas de pensamiento sobre la técnica adecuada para alcanzar un diseño bien estructurado. Entre las más importantes se incluyen:

1. *Yourdon-Constantine.* Esta técnica deriva la estructura ideal del flujo de datos a través de funciones necesarias en un programa.

2. *Warnier-Orr.* Esta técnica deriva la estructura ideal de los contenidos de las salidas y entradas.

Estas técnicas usan diferentes herramientas para representar la estructura descendente de módulos.

1.1 DISEÑO ESTRUCTURADO DE YOURDON

El diseño estructurado de Yourdon busca que un programa esté dentro de la jerarquía descendente con las siguientes propiedades:

- los módulos deben ser altamente cohesivos, esto es, cada módulo debe llevar a cabo una y sólo una función.
- los módulos deben de estar libremente acoplados; en otras palabras, los módulos deben ser mínimamente dependientes uno del otro. Esto se logra obligando a que el flujo de datos entre módulos sea únicamente a través del paso de parámetros.

La especificación o modelo derivado del diseño estructurado Yourdon se llama trazo estructurado. El trazo estructurado se deriva del estudio del flujo de datos a través del sistema y simplifica las fases de construcción y entrega a través de su estructura descendente.

Los beneficios de un diseño estructurado son numerosos: programas que son tratados de acuerdo al diseño estructurado pueden ser más fáciles de escribir y probar por múltiples equipos de programadores porque las interfaces entre los módulos están bien definidas y limitadas por reglas: los módulos que al probarse en forma aislada funcionan correctamente deben funcionar también correctamente cuando se unen para formar un sistema. Las estructuras de programa descendente también simplifican el esfuerzo de programación. Por otro lado, los módulos de un programa que son desarrollados de acuerdo al diseño estructurado tienden a ser reusables, esto es porque ellos están construidos para ser cohesivos.

La reusabilidad de código ha venido a ser una cuestión de mucha importancia dentro de la industria del software por el inherente ahorro de tiempo y esfuerzo. Pero los módulos no pueden ser reusables a menos que ellos sean intencionalmente diseñados para la reusabilidad; en este sentido, el diseño estructurado representó un gran avance en la dirección correcta.

Asímismo, sistemas y programas desarrollados mediante el diseño estructurado se mantienen fácilmente. La mayor dificultad en el mantenimiento de un programa es conocido como el "efecto de onda" (cripple effect), el cual ocurre cuando un cambio en un módulo requiere cambios en

diversos módulos. El diseño estructurado busca reducir el efecto de onda minimizando conexiones intermódulo y dependencia.

El *análisis estructurado* es hasta ahora la más popular y más ampliamente practicada de las técnicas de sistemas estructurados. La técnica de análisis estructurado es simple en concepto. La especificación de nuevos sistemas va de una serie de modelos de flujo llamados diagramas de flujo de datos o DFD's, los cuales no muestran explícitamente el control de flujo a través de un sistema, sino que únicamente muestran el fujo de datos, almacenamiento de datos y los procesos que responden a un cambio de datos. A causa de su dependencia en los flujos de datos y procesos, el análisis estructurado es típicamente referido como una proposición de flujo de datos. El analista produce varios tipos de DFD's en un solo análisis estructurado.

El concepto de *sistema lógico*, algunas veces llamado sistema esencial, fué creado para solucionar adecuadamente un problema común: se tiende a dañar la creatividad pensando prematuramente en un nuevo sistema en términos de cómo debe funcionar. El análisis estructurado requiere que el analista defina lo que el sistema debe hacer (sistema lógico) antes de decidir cómo el sistema debe lograr estas metas. Sus partidarios insisten que reduciendo el sistema a su esencia lógica se obtienen los siguientes beneficios:

- El analista define más exactamente los requerimientos del usuario final y no se preocupa prematuramente de la tecnología.
- El analista es más proclive a concebir alternativas de solución más creativas en lugar de soluciones que están basadas en el sistema existente.

1.1.1 Estrategias para un diseño modular

Una de las primeras estrategias para el diseño estructurado, es la de entrada-proceso-salida de IBM, llamada HIPO. Consiste en que el diseñador descompone un programa en funciones lógicas, de tal manera que quede presentado como un diagrama estructurado. Cada módulo es eventualmente documentado con un diagrama HIPO que fuerza a la persona a detallar las entradas (estas entradas incluyen accesos a archivos y paso de parámetros a subrutinas), requerimientos de procesamiento (seudocódigo o diagrama de flujo) y salidas (incluyendo reportes, pantallas, actualización de archivos).

Ed Yourdon y Larry Constantine han desarollado lo que ha sido una estrategia popular que permite determinar una estructura óptima de diagramas para programas. Su técnica es precisamente la llamada *Diseño Estructurado*, y está basada en el uso de diagramas de flujo de datos.

El *Análisis de Transformación* es un exámen del DFD para dividir los procesos que ejecutan entrada y edición y aquellos procesos que se encargan de los cálculos.

El *Análisis de Transacción* es el examen de el DFD para identificar procesos que son distintos y por lo tanto centros de la transacción. El diagrama estructurado resulta en módulos centrales de transacción, que pueden ser divididos entonces en módulos HIPO.

Usando esta estrategia para dividir un programa en módulos se dice que hay módulos acoplados libremente y altamente cohesivos. Los módulos acoplados libremente tienden a ser casi independientes uno del otro mientras que los módulos altamente cohesivos contienen instrucciones que trabajan colectivamente para resolver una tarea específica. Los datos y símbolos de flujo de control presentados en un diagrama de estructura pueden servir como ayuda en determinar el grado de acoplamiento y cohesión de módulos.

1.1.2 Modelado

El modelado es la creación de una representación abstracta de un ente real. Un modelo está constituido con objeto de aclarar ciertos aspectos que deseamos estudiar antes de crear algo real. Se podría decir que un modelo es un simulacro a bajo costo de un sistema complejo que se desea estudiar. Se construyen por varios motivos:

- Para enfocar características importantes del sistema, a la vez que para minimizar las caracteristicas menos importantes.
- Para discutir cambios y correcciones a los requerimientos del usuario a bajo costo y con riesgo mínimo.
- Para verificar que se entiende el ambiente del usuario, y que se ha documentado de tal manera que los diseñadores y programadores puedan construir el sistema.

Existen varios tipos de modelos que se pueden construir para el usuario: modelos narrativos, modelos de prototipos, modelos gráficos diversos, etc. .

En los modelos de papel existen diagramas de flujo, diagramas HIPO, tablas de decisión, diagramas de flujo de datos, diagramas de transición de estados, árboles de decisiones, diagramas de entidad relación, diagramas de Ferstl, diagramas de Hamilton-Zeldin, diagramas PAD y una interminable serie de diagramas, tablas y gráficas que se pueden presentar al usuario.

La mayoría de los sistemas requieren de múltiples modelos: cada modelo se enfoca a un número limitado de aspectos del sistema, a la vez que minimiza otros de sus aspectos.

Desarrollo de modelos en software

Los usos básicos de modelos en el desarrollo de software nos permiten, entre otras cosas, las siguientes:

- validación por revisión
- comunicación para aquellos que no tienen conocimientos acerca de un sistema, pero que necesitan entenderlo.
- organización de las ideas del modelador.

Las principales características que una herramienta de modelado debe poseer son las siguientes:

- Ser gráficas con texto de apoyo
- Tener redundancia de control. (El modelo debería tener suficiente redundancia para ayudar al lector a ajustar las piezas juntas).
- Ser capaz de predecir el comportamiento del sistema, pudiéndose así verificar que el comportamiento sea aceptable o cambiar el modelo si el comportamiento no es aceptable.

Cuatro son los componentes del modelo del sistema:

1. El ambiente esencial del modelo para aclarar los requerimientos por la interacción del sistema con sus alrededores.

2. El comportamiento esencial del modelo para aclarar las respuestas requeridas al sistema.

3. El modelo de implementación del usuario.

4. El modelo de implementación del sistema para aclarar la principal técnica de selección de implementación.

Herramientas de modelado

Una herramienta de modelado es la base para definir diferentes aspectos de un sistema. Se necesita utilizar varias herramientas para modelar un sistema efectivo y conectarlas después para producir una gráfica conveniente. Una herramienta gráfica nos muestra los componentes del modelo y las conexiones entre los componentes. Una herramienta textual define precisamente el significado de las conexiones y los componentes.

Modelo del proceso

El trabajo básico de un sistema es la transformación de datos en información.

Se utilizan gráficas para mostrar las transformaciones, los datos que son transformados y de dónde vienen, la información y a donde vá y los datos que se guardan en las transformaciones.

Se usan textos para definir las reglas que rigen la transformación y el contenido y significado de los datos y la información.

Las herramientas gráficas y textuales se integran para darnos un completo proceso del modelado.

Modelo de datos del almacenamiento

Aquí se utilizan gráficas para mostrar grupos de información y conexiones entre los grupos. La ayuda gráfica muestra los datos actuales del almacenamiento, el significado de esos datos y la definición de las conexiones.

Modelo de secuencia de interacciones

Se usa para describir un sistema con interacción entre personas y computadoras e interacción entre computadoras. Se necesita mostrar el control de las transformaciones y qué pasa en ellas.

Modelo de organización del programa

Para describir la organización de las instrucciones de computadora, se utilizan gráficas que muestran grupos de instrucciones, transferencia de control entre los grupos de instrucciones y la comunicación de datos entre éstos.

Este modelo de organización del programa usa apoyo textual para definir qué toma de los datos que recibe, qué datos ofrece y cómo presenta los datos que ofrece.

Modelo esencial

El modelo esencial aclara qué es lo que puede hacer un sistema para tener éxito, suprimiendo detalles de cómo ser implantado. Entiéndase por sistema una colección de actividades y almacenamiento de informacion que responde a eventos de una forma planeada y específica.

Son dos sus componentes: el *modelo ambiental* y el *modelo de comportamiento o funcional.*

El Modelo Ambiental .

Este modelo define los alcances del sistema. Consiste de:

- Declaración del propósito del sistema.
- Diagrama de contexto y definiciones secundarias. Este contexto muestra la gente, organizaciones y sistemas que están en el ambiente del sistema bajo desarrollo y las conexiones entre el sistema bajo desarrollo y su ambiente.
- Lista de eventos. Describe los eventos en el ambiente que sean necesarios para que el sistema de algún modo responda. Abarca respuestas normales y a problemas específicos.

El Modelo de Comportamiento.

Consiste de descripciones de procesos y datos esenciales almacenados que son requeridos para responder a los eventos.

Para modelar procesos esenciales es necesario modelar el proceso por el cual el sistema responde a los eventos y las interconexiones entre esos procesos.

Para modelar almacenamientos de datos esenciales se necesita la información que debemos almacenar para permitir que el sistema opere y las asociaciones entre estas piezas de información.

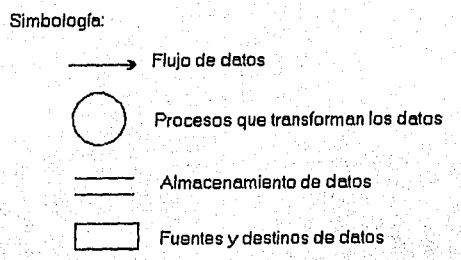


Figura 1.1: Componentes de un Diagrama de Flujo de Datos

1.1.3 Diagramas de flujo de datos

El digrama de flujo de datos (DFD) es una herramienta de modelaje que muestra a un sistema como una red de procesos conectados por trayectorias de datos. Es decir, muestra los procesos y el flujo de datos a través de estos procesos. Un diagrama de flujo de datos se usa como un primer paso en una forma de diseño estructurado y enseña el flujo de datos general a través de un sistema o programa. Es primeramente una herramienta de análisis de sistemas usada para obtener los componentes procedurales básicos y los datos que pasan a través de ellos.

Un *diagrama de datos concentrados* es un diagrama de flujo de datos primario en el que sólo se muestran los datos de entrada, el proceso global del sistema y los datos de salida que finalmente se obtienen como resultado del proceso.

Componentes de un DFD

Un diagramaa de flujo de datos se utiliza para declarar procesos y las interfaces que ocurren entre estos. Se construye con cuatro componentes básicos: un flujo de datos, un proceso, un almacén de datos y un terminador. Ver figura 1.1.

Un *flujo de datos* traza el fluir de los datos a través de varios procesos. La dirección del flujo de datos se indica con una flecha. Los datos se identifican con un nombre, el cual se escribe encima de su correspondiente flecha.

Un *proceso* transforma los datos. Por ejemplo, puede realizar operaciones

aritméticas o lógicas en los datos para producir un resultado. Cada proceso se representa por un círculo o un rectángulo con las esquinas redondeadas, en cuyo interior se escribe el nombre del proceso. Varios flujos de datos pueden entrar y salir de un proceso.

Un *almacén de datos o almacenamiento* representa un archivo lógico. Se dibuja como un par de líneas paralelas. El nombre del almacén de datos se escribe entre las líneas. Cada almacén de datos se conecta a un proceso mediante un flujo de datos. La dirección de la flecha del flujo de datos enseña si los datos son leídos del almacén de datos o producidos por el proceso y luego dirigidos a el almacén de datos.

Un *terminador* muestra el origen de los datos que usa el sistema y el recipiente último de datos producidos por el sistema. Al origen de los datos se le llama *fuente* y al recipiente de los datos, *destino*. Se representa mediante una caja rectangular.

Un diagrama de flujo para un almacenamiento puede representar:

- agregar información nueva a lo almacenado.
- cambiar el contenido de lo almacenado.
- quitar información de la que está almacenada.

El no poder nombrar un flujo de datos generalmente es una clave que permite conocer que hay un problema difícil de resolver como es el colocar juntas piezas de información que no se relacionan o colocar juntos procesos que no se relacionan.

Los DFDs se pueden usar para proveer vistas de alto nivel o detalladas de un sistema o programa. Lo que ocurre en una caja de un DFD puede enseñarse con más detalle en otro DFD mediante la *expansión de procesos.*

Más adelante volveremos a tratar estos diagramas y presentaremos un esquema.

1.1.4 Diccionario de datos

Se debe conformar un diccionario de datos que defina la información en detalle declarada en el DFD de la siguiente manera:

- describiendo el significado de los flujos y la información almacenada.

- especificando la composición de los flujos compuestos y la infomación almacenada.

- especificando las unidades y los valores de flujos elementales y lo almacenado.

La importancia de la definición de datos radica en que sin un diccionario de datos un DFD es sólamente un bosquejo.

La definición de los datos nos sirve para reducir ambigüedad y especificar detalles.

Los elementos de los datos son definidos por su significado para el sistema, su valor, su rango y sus unidades.

El significado explica la relevancia del flujo almacenado en el sistema.

El usuario necesita explicar por qué el dato es importante, además es necesario evitar las expresiones redundantes como son:

- Descripción de la composición del dato o derivación.

- Repetición del significado de los elementos ya que ellos tienen su propia definición.

- Repetición del nombre del dato.

Al crear subdefiniciones para el sujeto en cuestión en su vocabulario, frecuentemente se crean definiciones más comprensibles.

Si los datos contienen un gran número de elementos se deberá:

1. Separar los elementos en grupos.

2. Asignar un nombre significativo a cada grupo.

3. Definir los datos en términos del grupo al que pertenecen.

La especificación de los elementos se logra demostrando el rango y los valores de dichos elementos.

Idealmente, cada término en el diccionario de datos hace referencia a algo único en el modelo del sistema. Un alias es una alternativa de nombre hacia algo bien definido en el diccionario de datos.

1.1.5 El modelo esencial

Examinemos el enfoque del análisis estructurado clásico para desarrollar modelos de sistemas.

Los cuatro modelos de los sistemas son:

1. El modelo físico actual es un modelo del sistema que actualmente está empleando el usuario.

2. El modelo lógico nuevo es un modelo de los requerimientos puros o esenciales del sistema nuevo que el usuario desea.

3. El modelo lógico actual es el modelo de los requerimientos puros o esenciales que realiza el sistema actual del usuario.

4. El nuevo modelo físico es un modelo que muestra las limitaciones de implantación impuestas por el usuario. Una de las limitaciones más importante es la determinación de la frontera de automatización.

No obstante, el modelo clásico no funciona del todo porque ignoró que el proceso de desarrollar un modelo del sistema actual puede requerir tanto tiempo y esfuerzo que el usuario se frustre, se impaciente y termine por cancelar el proyecto. El problema ocurre porque el analista se distrae con la tarea de modelar el sistema actual y empieza a pensar en él como un fin en sí mismo.

Este enfoque involucra un gran desperdicio de tiempo. Se recomienda que el analista evite modelar el sistema actual de ser posible. Las herramientas de modelado deben usarse para comenzar, tan pronto como sea posible, a desarrollar un modelo del nuevo sistema que el usuario desea. Este nuevo sistema se conoce como el modelo esencial del sistema.

El modelo esencial del sistema es un modelo de lo que el sistema debe hacer para satisfacer los requerimientos del usuario, diciendo lo mínimo posible acerca de cómo se implantará. El modelo esencial debe describir el contenido de los flujos o almacenes de datos, sin describir el medio u organización física de los datos.

Componentes del modelo esencial

Como ya se mencionó, consiste en dos modelos principales:

- Modelo Ambiental
- Modelo de Comportamiento o funcional

El modelo ambiental define la frontera entre el sistema y el resto del mundo, consiste en un diagrama de contexto, una lista de acontecimientos y una descripción breve del propósito del sistema.

El modelo de comportamiento describe el comportamiento que el sistema requiere para que interactúe de manera exitosa con el ambiente.

Existen circunstancias en las cuales podría ser deseable o necesario construir un modelo de implantación antes de construir el modelo esencial del sistema. Su primer objetivo es llegar a un entendimiento y una visión generales del sistema existente. No se trata de documentar el sistema actual con detalle.

Podría ser útil reunir algunos de los documentos físicos que representarían un diccionario de datos físicos. Pero no se debe intentar escribir especificaciones de proceso para todas las funciones, ni se trate de desarrollar un diccionario de datos completos para el sistema existente.

Cuando se haya terminado de desarrolar el modelo de la implantación actual, la siguiente tarea será definirlo en términos lógicos, los cuales abarcarán los siguientes puntos:

- Buscar y separar flujos esenciales que hayan sido empaquetados de manera arbitraria en el mismo medio.
- Buscar flujos empaquetados o agregados que se envían a burbujas (que representan a computadoras, personas, etc.) que no requieren de todos los datos que hay en dichos flujos.
- Distinguir entre el trabajo esencial realizado por un proceso y la identificación del procesador que aparece en el modelo de implantación.
- Eliminar procesos cuyo único propósito sea transportar datos de un lugar a otro dentro del sistema.
- Eliminar procesos cuya labor sea verificar datos que se producen y van dentro del sistema.
- Buscar situaciones donde los almacenes esenciales se hayan empaquetado en el mismo almacén de implantación.

- Eliminar datos de los almacenamientos si ningún proceso los usa.
- Eliminar almacenes de datos que sólo existan como separadores de tiempo entre procesos, y que sean dependientes de la implantación. Esto incluye archivos intermedios, archivos de reportes, archivos de impresión diferida y otros similares.

El modelo ambiental

Para el analista la labor más difícil en la especificación de un sistema es a menudo determinar qué es parte del sistema y qué no. El modelo ambiental es un modelo que define las interfaces entre el sistema y el resto del universo (ambiente). El modelo ambiental modela el exterior del sistema y el modelo del comportamiento modela el interior. Se debe definir la frontera entre el sistema y el ambiente y también definir las interfaces entre el sistema y el ambiente. Es necesario saber qué información entra al sistema desde el ambiente exterior y qué información produce como salida al ambiente exterior. Los sistemas producen salidas como repuesta a algún acontecimiento o estímulo en el ambiente. Otro aspecto del modelo ambiental consiste en identificar los acontecimientos que ocurren en el ambiente al cual debe responder el sistema. Los acontecimientos que nos preocupan son aquellos que ocurren en el ambiente exterior y requieren una respuesta del sistema. Es importante dedicar bastante tiempo y tener suficiente participación del usuario en la elección de una frontera apropiada para el sistema. El área dentro de la frontera del sistema se conoce como el *dominio de cambios*, porque todo lo que está dentro de la frontera del sistema está sujeto a cambios, mientras todo lo que esté fuera se queda en su forma actual y no es investigado por el analista.

Herramientas usadas para definir el ambiente

El modelo ambiental consta de tres componentes:

1. Declaración de propósitos
2. Diagrama de contexto
3. Lista de acontecimientos

La declaración de propósitos es una declaración textual breve y concisa del propósito del sistema, dirigida al nivel admnistrativo superior, la administración de los usuarios, y otros que no estén directamente involucrados con el desarrollo del sistema. La intención de la declaración de propósitos no es proporcionar una descripción completa y detallada del sistema. Puede resumir los beneficios tangibles y cuantificables que se logren con el nuevo sistema, esto cuando se trata de proyectos pequeños y muy definidos. En el caso de proyectos grandes suele requerirse de un análisis de costo-beneficio.

El diagrama de contexto empieza contestando algunas de las preguntas que surgen a raíz de la declaración de propósitos y es considerado un caso especial del diagrama de flujo de datos. Este diagrama se encarga de enfatizar varias características importantes del sistema:

- Las personas, organizaciones y sistemas con los que se comunica el sistema (se conocen como terminadores)
- Los datos que el sistema recibe del mundo exterior y que deben procesarse de alguna forma.
- Los almacenes de datos que el sistema comparte con los terminadores.
- La frontera entre el sistema y el resto del mundo.

La lista de acontecimientos es una lista narrativa de los estímulos que ocurren en el mundo exterior a los cuales el sistema debe responder. Los acontecimineto pueden ser de tipo flujo, temporal o de control. El orientado a flujos es el que se asocia con un flujo de datos; es decir, el sistema se da cuenta de que ha ocurrido el acontecimiento cuando llega algún dato. Los acontecimientos de tipo temporal son aquellos que arrancan con la llegada de un momento dado en el tiempo. Los acontecimientos de control deben considerarse como un caso especial del acontecimiento temporal: un estímulo externo que ocurre en algún momento imprevisible. No se asocia con el paso regular del tiempo, por lo que el sistema no puede anticiparlo utilizando un reloj interno. No indica su presencia con el arribo de datos.

Los componentes adicionales del modelo ambiental son:

- El diccionario de datos inicial, que define todos los flujos y almacenes externos.

- El modelo de entidad-relación de los almacenes externos.

La construcción del diccionario de datos puede ser importante si las interfaces entre el sistema y los diversos terminadores están sujetas a cambios y a negociación. De manera similar, se puede construir un diagrama de entidad-relación de los almacenes externos (si hay).

Construcción del modelo ambiental

Es importante dedicar una buena cantidad de tiempo y energía al modelo ambiental, pues a menudo es el punto focal de juntas y presentaciones importantes al comienzo de la vida de un proyecto de desarrollo de sistemas. A veces es la única parte del modelo global del sistema que muchos usuarios y administradores de alto nivel llegarán a ver. Una vez construido el modelo ambiental debe ser revisado cuidadosamente por todos los representantes clave de los usuarios, además del equipo del proyecto. Entonces se estará preparado para comenzar a construir el modelo del comportamiento, es decir, el modelo del interior del sistema.

Construcción de un modelo preliminar de comportamiento

Después de desarrollar el modelo ambiental para un sistema, nuestra labor es comenzar a construir el modelo de comportamiento del sistema, es decir, el modelo del comportamiento final que el sistema debe tener para manejar con éxito el ambiente. Esto involucrará el desarrollo de un diagrama de flujo de datos y un diagrama de entidad-relación preliminares, además de la elaboración de las entradas iniciales del diccionario.

1.2 METODOLOGÍA WARNIER-ORR

La metodología de Warnier-Orr, también llamada 'Desarrollo de Sistemas Estructurados de Datos' o DSSD, fue desarrollada inicialmente por Jean Dominique Warnier y enriquecida por su colega Ken Orr. Warnier desarrolló una notación conveniente para representar información jerarquizada usando tres construcciones: secuencia, repetición, selección o alternancia y demostró que la estructura del software podría ser derivada directamente de la estructura de los datos. Orr extendió el trabajo de Warnier al hacer un

análisis más extenso del dominio de la información que evolucionó en el DSSD, el cual toma en cuenta, además de la jerarquía de los datos, el flujo de la información y las características funcionales.

Esta metodología, a diferencia de la de Yourdon, obtiene los datos de entrada y la estructura misma del programa de los datos de salida. Su filosofía consiste en que las salidas del programa determinan completa y absolutamente la estructura de los datos, la cual a su vez determina la estructura misma del programa. Es por ello que también se le conoce como análisis orientado por las salidas.

1.2.1 Diagramas de Warnier

Los diagramas de Warnier permiten al analista representar información jerarquizada en una manera compacta. Aquí, por jerarquía se puede entender una relación entre un conjunto de información o instrucciones y uno de sus subconjuntos. Un ejemplo de jerarquía lo encontramos en el juego de fútbol. Cada temporada de futbol cuenta con cierto número de partidos y cada partido cuenta con dos mitades. Esta jerarquización se representa en un diagrama de Warnier de la siguiente forma:

Temporada de futbol { juegos (1,g) { mitades (2)

Se analiza el dominio de la información y se representa la jerarquía natural de la salida. El diagrama de Warnier puede ser usado para representar la jerarquía a cualquier nivel de detalle. Las llaves son usadas para diferenciar niveles de la jerarquía de la información. Los nombres contenidos dentro de una llave representan una *secuencia* de elementos de información (cada elemento puede ser una composición de otros elementos o un elemento primario). La notación frente a algunos nombres representa *repetición*, esto es, el número de veces que un elemento en particular aparece en la jerarquía. También se acostumbra poner estas repeticiones debajo de los nombres.

El diagrama de Warnier puede usarse para dividir el dominio de la información mediante el refinamiento de elementos compuestos, para lo cual se usa el símbolo del "o exclusivo", el cual indica una ocurrencia condicional (*selección*) de un elemento de información: un proceso o el otro, pero no los dos, se llevará a cabo.

A continuación se enlistan las construcciones básicas de un diagrama de Warnier.

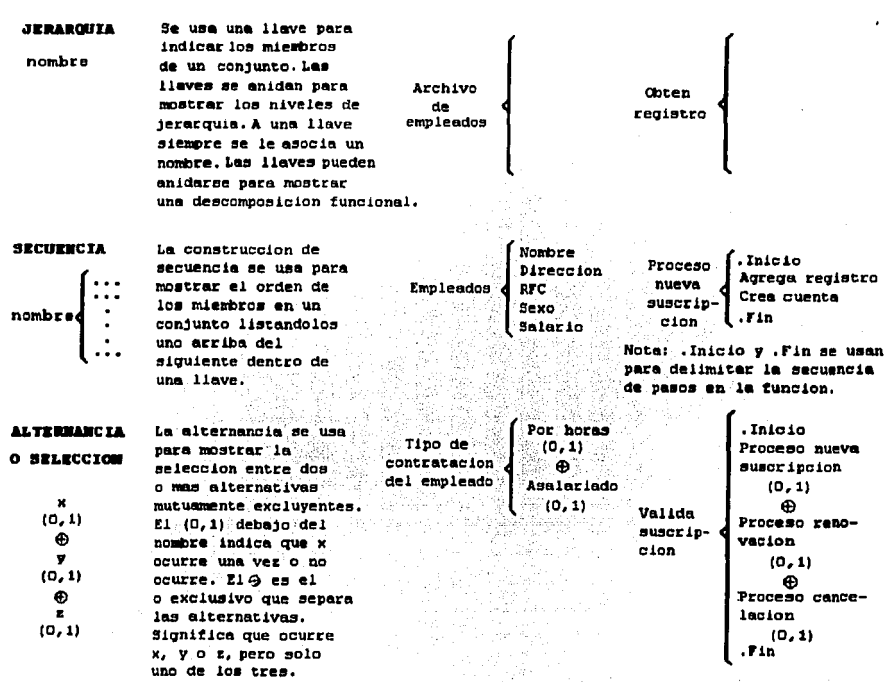


Figura 1.2: Elementos de un diagrama de Warnier.

1.2.2 El enfoque DSSD

En vez de empezar el análisis por examinar la jerarquía de la información, el DSSD primero examina el *contexto de aplicación*, esto es, cómo los datos se mueven entre productores y consumidores de información desde la perspectiva de uno de los productores o consumidores. En seguida, se señalan las *funciones de aplicación* con una representación parecida a la de los diagramas de Warnier que describe elementos de información y los procesos que deben ser llevados a cabo en ellos (esto es similar en concepto al diagrama de flujo de datos). Finalmente, se modelan los *resultados de aplicación* usando el diagrama de Warnier. Con este enfoque, el DSSD cubre todos los atributos del dominio de la información: flujo de datos, contenido y estructura.

1.2.3 El contexto de aplicación

Para determinar el contexto de aplicación del DSSD, el problema debe ser planteado de manera que nos permita contestar las siguientes preguntas:

1. ¿ Qué son los elementos de información que deben ser procesados ?
2. ¿ Qué o quienes son los productores y consumidores de información ?
3. ¿ Cómo ve la información cada productor o consumidor en el contexto de otros constituyentes ?

El DSSD propone un *diagrama de entidades* como mecanismo para responder a éstas preguntas.

El diagrama de entidad usa una notación que es muy similar al diagrama de flujo de datos. Sin embargo, símbolos similares tienen un significado distinto.

El círculo en un diagrama de entidad representa un productor o consumidor (una persona, una máquina, otro sistema) de información.

Después de que cada diagrama de entidad se revisa que esté correcto, se crea un diagrama de entidad combinado para todos los consumidores y productores de información. Las entidades que caen dentro de las fronteras del sistema propuesto se indican mediante la identificación de una *frontera de aplicación*. Los detalles dentro de la frontera de aplicación pueden estar escondidos temporalmente.

1.2.4 Funciones de aplicación

Las funciones que deben ser implementadas para realizar el sistema pueden ser discernidas mediante el examen de flujo de la información a través de la frontera de aplicación. Usando una notación parecida a la de Warnier llamada *diagrama de línea de ensamblaje* o ALD, el DSSD provee un mecanismo para acoplar información y los procesos (transformaciones o funciones) aplicados a él. Conceptualmente, el ALD juega el mismo papel que el diagrama de flujo de datos.

Un ALD se desarrolla empezando por el último flujo de información numerado y trabajando hacia atrás hacia el primer flujo numerado. El elemento de flujo de información se deriva de combinar el elemento anterior de información numerado con el procedimiento que crea el elemento deseado.

1.2.5 Resultados de la aplicación

El DSSD requiere del analista que construya un prototipo de la salida deseada para el sistema. El prototipo identifica la salida primaria del sistema y la organización de los elementos de información que componen la salida. Una vez que se ha creado el prototipo, la jerarquía de la información puede modelarse usando un diagrama de Warnier-Orr.

1.2.6 Requerimientos físicos

La notación del DSSD que incluye entidad, línea de ensamblado y diagramas de Warnier-Orr se usa para modelar requerimientos de software desde el punto de vista lógico. Los requerimientos físicos deben también determinarse como parte del análisis. Dentro de los requerimientos físicos que deben considerarse están el desempeño, la seguridad, la confianza, el hardware y las interfaces. Estos puntos, cuando se acoplan con los requerimientos lógicos derivados mediante el DSSD, proveen al analista con un método y notación para obtener los requerimientos del software.

1.2.7 Construcción lógica de programas y sistemas

Dentro del contexto del diseño orientado a la estructura de los datos, se tiene la Construcción Lógica de Programas y Sistemas (LCP), desarrollada por

Warnier para definir un conjunto de reglas y leyes que gobiernan la estructura de la información y la organización resultante del software obtenido.

La LCP presenta procedimientos para el análisis y el diseño. Empezando con una representación formal de la estructura de los datos mediante el diagrama de Warnier-Orr, el método conduce a la derivación de procedimientos y culmina con métodos sistemáticos para la generación de seudocódigo, verificación y optimización.

El enfoque del diseño LCP comienza con la especificación de las estructuras de los datos de entrada y de salida con el uso de diagramas de Warnier. Una evaluación de los requerimientos del software conlleva hacia la derivación de una representación del software.

Un proceso de jerarquización para un programa se deriva de la estructura de datos de entrada. La representación del diagrama de Warnier para un software determinado puede transformarse en una representación de diagrama de flujo más convencional, en la que estructuras de repetición se mapean en construcciones repite-hasta y ocurrencias condicionales en construcciones si-entonces-si no; de tal forma que sea posible interpretar cada caja del diagrama de flujo como un módulo. Sin embargo, Warnier toma un punto de vista en cuanto a los procedimientos que impide la definición directa de módulos.

La LCP intenta extender la metodología de diseño en un dominio que otros métodos evitan mediante una técnica llamada *organización detallada*, en la que un conjunto de instrucciones detalladas pueden desarrollarse sistemáticamente de la organización lógica del programa.

Warnier define los siguientes tipos de instrucciones:

- Entradas y preparación de entradas
- Ramificaciones y ramificaciones preliminares
- Cálculos
- Salidas y preparación de salidas
- Llamadas a módulos (subprogramas)

Una organización detallada se desarrolla mediante la generación de listas de instrucciones por tipos. La instrucción es escrita y correlacionada al bloque apropiado de procedimientos con una indicación numérica. Después de que

se preparan listas de instrucciones, instrucciones con el mismo identificador de bloque de procedimientos se agrupan y organizan en una secuencia de entrada-procesamiento-salida.

La organización detallada provee al diseñador con una técnica para desarrollar una descripción de diseño detallada de una forma sistemática.

Estructuras complejas

Cuando la organización lógica de un programa es compleja, se requiere de técnicas adicionales de diseño para representar y simplificar condiciones y procesos. La LCP recomienda el uso del álgebra booleana para ayudar a reducir la complejidad lógica, ayudando así al diseñador en la especificación de la organización detallada. El enfoque LCP para la simplificación lógica requiere evaluación rigurosa de acciones y condiciones.

La simplificación lógica también puede usarse durante el mantenimiento de software existente no estructurado. Muchos programas tienen un flujo de control disperso que es difícil de entender e imposible de mantener. Warnier sugiere un enfoque para restructurar tales programas con el uso de estructuras alternativas complejas:

- Desarrollar un diagrama de flujo para el software.
- Escribir una expresión booleana para cada secuencia de proceso.
- Construir una tabla de verdad.
- Reconstruir el software usando técnicas para estructuras alternativas complejas, añadiendo modificaciones conforme se requieran.

La LCP ofrece una segunda metodología de diseño orientada a la estructura de datos.Usando un conjunto de reglas desarrolladas a partir de los fundamentos de las ciencias de la computación,Warnier propone un enfoque de diseño riguroso que es conducido por la jerarquía de la información. Ofrece un conjunto de técnicas que extienden el diseño de software en una especificación de procedimientos detallada, simplificación lógica e incluso en la reestructuración del software existente.

1.2.8 Desarrollo de sistemas estructurados de datos: DSSD

Como se mencionó anteriormente, el DSSD extiende los conceptos básicos desarrollados por Warnier en una metodología más comprensible para el análisis y diseño de sistemas basados en computadoras.

Como entrada para el proceso de diseño del DSSD se toma la información de análisis de requerimientos que incluye el contexto de aplicación, la descripción funcional y los resultados de la aplicación. Los diagramas y datos contenidos en estas representaciones se usan como el fundamento del diseño lógico y físico del DSSD. El diseño lógico se enfoca en las salidas, interfaces y el diseño de procedimientos del software. El diseño físico evoluciona a partir del diseño lógico y se enfoca en el "empaquetamiento" del software para poder alcanzar de la mejor forma el desempeño deseado, mantenimiento y otros factores de diseño impuestos por el ambiente del sistema.

Los pasos principales que se siguen en el proceso de diseño son:

1. *Definición de las salidas del proceso.* Cada salida del programa se representa como una estructura de datos jerárquica.
2. *Definición de la base de datos lógica.* Se definen todos los datos necesarios para producir la salida del sistema.
3. *Realización de análisis de los eventos.* Se definen todos los eventos que pueden afectar (cambiar) los datos en la base de datos lógica.
4. *Desarrollo de la base de datos física.* Se definen los archivos físicos para los datos de entrada.
5. *Diseño del proceso lógico.* Se diseña la lógica de procesamiento del programa necesaria para producir la salida deseada de la entrada.
6. *Diseño del proceso físico.* Se añade la lógica de control y los procesos que permitan manejar los archivos para completar el diseño del programa.

Un enfoque de diseño simplificado

Para diseños no muy complicados, la metodología se vuelve muy tediosa para seguir en detalle. En vez de ello muchos diseñadores encuentran suficiente

emplear una versión abreviada de la metodología que empieza definiendo la estructura de salida usando un diagrama de Warnier y luego provee la lógica necesaria para producir tal salida. El proceso de diseño lógico puede dividirse en dos actividades principales: la derivación de la *Estructura de Salida Lógica* (LOS) y la definición resultante de la *Estructura de Proceso Lógica* (LPS).

Para la obtención de la LOS, los datos que son parte del dominio de información de un problema se organizan de manera jerárquica. Posteriormente se aplican los siguientes pasos:

1. Se evalúa el problema o la información de los requerimientos relativos y se listan todos los datos distintos ("átomos") que no pueden subdividirse más.
2. Se especifica la frecuencia de ocurrencia de cada átomo.
3. Se evalúan los datos que pueden ser subdivididos ("datos universales").
4. Se desarrolla una representación esquemática de la Estructura de Salida Lógica.

Ahondemos un poco más en este proceso.

Obtención de la Estructura de Salida Lógica (LOS)

La LOS es una representación jerárquica de los datos que componen la salida de un sistema computarizado. El primer paso en la obtención de la LOS consiste en aislar todos los datos que no pueden ser subdivididos ("átomos"), lo cual se puede lograr revisando el enunciado del problema. En seguida, se debe notar la frecuencia de ocurrencia de cada átomo. Una vez que se definan todos los átomos y su frecuencia, el diseñador empieza un examen de los "universales", datos o categorías que son compuestas de otros universales y átomos.

Obtención de la Estructura de Proceso Lógica (LPS)

La LPS es una representación de procedimientos de software requerida para procesar la correspondiente LOS. Cada dato universal deviene una construcción de repetición a la cual se añaden instrucciones de procesamiento. Los pasos siguientes conducen a la obtención de la LPS:

1. Todos los átomos son listados del diagrama de Warnier para la LOS.
2. Se definen todas las instrucciones o procesos de inicialización y terminación.
3. Se especifica todo proceso computacional o no numérico.
4. Se especifican todas las instrucciones y procesos de salida/entrada.

§ 2

DISEÑO DE BASES DE DATOS RELACIONALES USANDO UNA METODOLOGÍA ORIENTADA A OBJETOS

El diseño de bases de datos o modelado de datos es una faceta de la ingeniería de software. Un modelo de datos es el primer paso en el diseño de una base de datos para una aplicación, el cual permite definir la estructura misma de la base de datos. En el caso de un sistema manejador de una base de datos relacional, esta estructura incluye detalles como definición de atributos y tablas así como especificar reglas que garanticen la integridad de dichas tablas. Las aplicaciones llenan la estructura de la base de datos y hacen la información accesible al usuario.

La meta de un modelado de bases de datos es poder diseñar una mejor base de datos. Algunos criterios importantes son:

1. Desempeño: ¿La estructura de la base de datos promueve la disponibilidad de los datos? ¿Los usuarios pueden recuperar y actualizar datos relevantes rápidamente?;

2. Integridad: ¿Hasta qué punto la base de datos garantiza que los datos correctos estén almacenados? (la definición de correctos depende de la aplicación);

3. Entendibilidad: ¿Qué tan coherente es la estructura de la base de datos a los usuarios finales, a otras arquitecturas de bases de datos y a los diseñadores originales después de un periodo de tiempo?

4. Extensibilidad: ¿Qué tan fácilmente puede extenderse la base de datos a nuevas aplicaciones sin irrumpir con el trabajo actual?

La metodología OMT aplicada al diseño de bases de datos relacionales ha probado su efectividad en el alcance de esta meta.

En este capítulo se dará una visión general sobre el funcionamiento de esta metodología de desarrollo orientada a objetos, misma que por su extensión es imposible revisarla con detalle en unas cuantas páginas.

2.1 TRABAJOS RELATIVOS

2.1.1 El modelo entidad-relación de Chen.

El enfoque de entidad-relación (ER) es la técnica más aceptada para el modelado lógico de datos. Como su nombre lo indica, el modelo ER soporta entidades y relaciones. Una *entidad* es algo que existe y que se puede distinguir. Un grupo de entidades similares forman un conjunto de entidades. Entidades y relaciones se describen por atributos.

Es útil resumir la información en un diseño usando diagramas de entidad-relación, donde:

1. Rectángulos representan conjuntos de entidades

2. Círculos representan atributos y están ligados a sus conjuntos de entidades respectivos mediante aristas. A veces se subrayan los atributos que sean parte de la llave de su entidad respectiva.

3. Diamantes representan relaciones, las cuales están unidas mediante aristas a sus conjuntos de entidades constitutivos.

La figura 2.1 muestra un diagrama simple de entidad-relación con 3 conjuntos de entidades: EMPS, DEPTS Y GERENTES. Los primeros dos están relacionados por la relación ASIGNADO_A y el segundo y tercero están

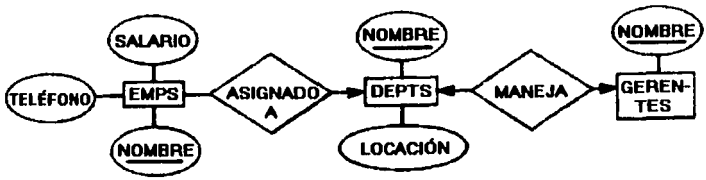


Figura 2.1: Ejemplo de diagrama entidad-relación.

relacionados por MANEJA. Para EMPS, por ejemplo, hay tres atributos: NOMBRE, TELEFONO y SALARIO.

Los diagramas ER son más expresivos que las tablas relacionales puesto que representan niveles más altos de abstracción. No obstante, a pesar de su utilidad, el método ER no es capaz de capturar completamente la intención de los modeladores de datos pues carece de una subesctructura para entidades y relaciones. Por ello se necesita de herramientas de modelado más poderosas.

2.1.2 Metodología de diseño relacional lógico de Teorey (LRDM)

Esta metodología es una extensión de la anterior. Se trata de una técnica gráfica de modelado de datos que maneja cuatro conceptos básicos: entidades, generalización, agregación y asociación, de los cuales se hablará más adelante. Por ahora, es suficiente decir que los conceptos adicionales mejoran el poder expresivo de esta metodología. El modelo ER representa una vasta mejoría sobre las tablas simples. Similarmente la LRDM es más poderosa que aquél.

2.1.3 Técnica de Modelado de Objetos (OMT)

Esta metodología se basa en conceptos de orientación a objetos.

Hay generalmente 4 características que conforman un enfoque orientado a objetos:

- *Identidad* – Significa que los datos están compactados en entidades discretas y distinguibles llamadas *objetos*, los cuales pueden ser concretos o conceptuales. Cada objeto tiene una identidad inherente que lo distingue. Un objeto es simplemente algo que tiene sentido en un contexto de aplicación. Los objetos sirven para promover un mejor entendimiento del mundo real y proveen una base práctica para la implementación computacional.

- *Clasificación* – Significa que objetos con las mismas estructuras de datos (atributos), comportamiento (operaciones), relaciones con otros objetos y la misma semántica, son agrupados en una *clase*. Una clase es una abstracción que describe propiedades importantes para una aplicación. Un objeto se puede ver como instancia de una clase.

- *Polimorfismo* – Significa que una misma operación puede comportarse de manera distinta en diferentes clases. Una operación es una acción o transformación que realiza un objeto o de la cual es objeto.

  Un *método* es una implementación específica de una operación por una cierta clase.

- *Herencia* – Es el compartir atributos y operaciones entre clases basadas en relaciones jerárquicas. Una clase puede ser refinada en varias subclases, cada una de las cuales hereda todas las propiedades de su superclase y adiciona las suyas propias.

  La habilidad para sintetizar propiedades comunes de varias clases en una sola puede reducir repeticiones dentro de diseños y programas.

El modelado y diseño orientado a objetos es una nueva manera de pensar en problemas usando modelos organizados al rededor de conceptos del mundo real. La construcción fundamental es el *objeto*, que combina atributos y operaciones en una sola entidad.

Los modelos de datos orientados a objetos comparten muchas de las características y beneficios de los programas orientados a objetos. Un programa orientado a objetos encapsula datos con procedimientos que actúan sobre esos datos. Cada paquete de datos y operaciones es llamado un objeto. Los objetos separan claramente la especificación externa de la implementación interna y se agrupan para facilitar el reuso de código similar,

siendo esta tecnología de objetos la más apropiada para problemas complejos, con estructura sofisticada.

En este enfoque, una base de datos almacena el componente pasivo de los objetos, esto es, sus datos privados o estado interno; mientras que las aplicaciones combinan estos datos con un componente activo, los procedimientos u operaciones.

La OMT, desarrollada a principios de los noventa, consta esencialmente de los siguientes pasos:

- Desarrollo de un modelo de análisis que contiene *objetos* que se encuentran en el dominio de aplicación, incluyendo una descripción de las propiedades de los objetos y su comportamiento.
- Se toman decisiones de diseño y se añaden detalles al modelo para describir y optimizar la implementación.

  Los objetos del dominio de aplicación forman la plataforma del modelo de diseño, y son implementados en términos de objetos del dominio informático.
- Finalmente el modelo de diseño es implementado en un lenguaje de programación, base de datos o hardware.

Para representar los modelos orientados a objetos se usa notación gráfica.

La OMT mejora los enfoques ER y LRDM. Una ventaja de los modelos de datos orientados a objetos es que permiten una integración sencilla con los programas orientados a objetos. Se podría considerar que en general es difícil fundir la interacción de bases de datos con el código de procedimientos; no obstante, el uso de una metáfora de objetos común y la misma notación de diseño para el modelado de datos y el desarrollo de programas facilitan esta situación.

2.2 DESARROLLO ORIENTADO A OBJETOS

El desarrollo orientado a objetos es una nueva manera de pensar acerca del software basada en abstracciones del mundo real. Desarrollo se refiere a los 3 puntos del ciclo de vida del software: análisis, diseño e implementación.

La esencia del desarrollo orientado a objetos es la identificación y organización de conceptos del dominio de aplicación.

El desarrollo orientado a objetos es un proceso conceptual independiente de un lenguaje de programación hasta las últimas etapas, permitiéndose así una mejor comunicación entre las personas involucradas. Aún como herramienta de programación puede tener varios destinos, incluyendo lenguajes y bases de datos convencionales así como orientadas a objetos.

2.2.1 Metodología Orientada a Objetos

Una metodología para el desarrollo orientado a objetos consiste en construir un modelo de un dominio de aplicación y después añadirle detalles de implementación durante el diseño de un sistema. A este enfoque se le llama OMT y consta de 4 etapas:

- *Análisis* – Partiendo de un planteamiento del problema, el analista construye un modelo de la situación real del problema enseñando sus propiedades importantes.

  El modelo de análisis es una abstracción precisa y concisa de 'qué' debe hacer el sistema.

- *Diseño del sistema* – El diseñador del sistema toma decisiones de alto nivel acerca de la arquitectura en general. El sistema destino se organiza en subsistemas basados en la estructura de análisis y la arquitectura propuesta. El diseñador del sistema debe decidir qué características optimizar y escoger una estrategia para atacar el problema, entre otras cosas.

- *Diseño de objetos* – Se construye un modelo de diseño basado en el modelo de análisis pero conteniendo detalles de implementación. El foco de atención aquí es la estructura de datos y los algoritmos necesarios para implementar cada clase. Las clases de objetos del análisis se aumentan con estructuras de datos del dominio informático y algoritmos usados para optimizar.

- *Implementación* – Las clases de objetos y relaciones desarrolladas durante el diseño de objetos son finalmente trasladadas en una implementación de un lenguaje de programación.

Los mismos conceptos orientados a objetos de identidad, clasificación, polimorfismo y herencia se aplican durante el ciclo entero de desarrollo.

Algunas clases no son parte del análisis sino que se introducen en parte del diseño o implementación.

La metodología OMT usa 3 clases de modelos para describir un sistema a lo largo de todas las etapas de desarrollo:

- *El modelo de objetos* – Describe la estructura estática de los objetos en un sistema así como sus relaciones. Contiene diagramas de objetos (gráficas cuyos nodos son clases y cuyos arcos son relaciones entre clases).
- *El modelo dinámico* – Describe las interacciones entre objetos en el sistema. Se usa para especificar e implementar los aspectos de control de un sistema. Contiene diagramas de estado (gráficas cuyos nodos son estados y cuyos arcos son transacciones entre estados causadas por eventos).
- *El modelo funcional* – Describe las transformaciones de datos del sistema. Contiene diagramas de flujo de datos (gráficas cuyos nodos son procesos y cuyos arcos son flujos de datos).

A diferencia del enfoque funcional o estructurado, el orientado a objetos se centra primero en identificar objetos del dominio de aplicación, colocando después procedimientos alrededor de ellos.

Profundicemos ahora en esta metodología.

2.3 METODOLOGÍA DE DISEÑO

La metodología OMT soporta en su totalidad, como otras metodologías tradicionales, el ciclo de vida del software. Una metodología de la ingeniería de software es un proceso para la producción organizada del software, usando una colección de técnicas predefinidas y convenciones en la notación. Una metodología se presenta usualmente como una serie de pasos, con técnicas y notación asociada con cada paso. Los pasos en la producción del software se organizan usualmente en un ciclo de vida consistente en varias fases de desarrollo. El ciclo de vida completo del software incluye la formulación inicial

del problema, el análisis, diseño, implementación, pruebas del software y una fase de operación durante la cual se realiza el mantenimiento y extensión. En general las fases del ciclo de vida se simplifican mediante un enfoque orientado a objetos, no obstante los métodos tradicionales usados en las pruebas y el mantenimiento no se alteran significativamente.Sin embargo, un enfoque orientado a objetos produce un diseño limpio y bien entendido que es más fácil de probar, mantener y extender que los diseños tradicionales puesto que las clases de objetos proveen una unidad natural de modularidad.

La metodología OMT está basada en el desarrollo de un modelo tripartita del sistema compuesto por los modelos de objetos, dinámico y funcional, el cual es después refinado y optimizado para constituir un diseño. El *modelo de objetos* captura los objetos en el sistema y sus relaciones. El *modelo dinámico* describe la reacción de los objetos en el sistema ante eventos, y la interacción entre los objetos. El *modelo funcional* especifica las transformaciones de los valores de objetos y las restricciones de éstas.

Las fases que constituyen esta metodología son el análisis, el diseño y el diseño de objetos.

El análisis consiste en entender y modelar la aplicación y el dominio dentro del cual opera. La entrada inicial a la fase de análisis es un planteamiento del problema que describe el problema a ser resuelto y provee una visión conceptual del sistema propuesto. La salida del análisis es un modelo formal que captura los 3 aspectos esenciales del sistema: los objetos y sus relaciones, el flujo de control dinámico y la transformación funcional de los datos sujeta a posibles condiciones.

La arquitectura completa del sistema se determina durante el diseño del sistema, usando el modelo de objetos como guía, el sistema se organiza en subsistemas. La concurrencia se organiza mediante el agrupamiento de objetos en tareas concurrentes. Se toman decisiones generales acerca de la comunicación entre procesos, el almacenamiento de datos y la implementación del modelo dinámico.

Durante la fase del diseño de objetos, los modelos del análisis se elaboran, refinan y optimizan para producir un diseño práctico. Durante esta fase hay un desplazamiento en el énfasis de los conceptos de aplicación hacia los conceptos de cómputo. Primero, se escogen los algoritmos básicos para implementar cada función principal del sistema. Basada en estos algoritmos, la estructura del modelo de objetos después se optimiza para una implementación eficiente. Se determina la implementación de cada asociación

y atributo y finalmente los subsistemas se empaquetan en módulos.

2.3.1 Impacto del enfoque orientado a objetos

La metodología OMT es un enfoque de construcción de software orientado a objetos que difiere de enfoques de desarrollo de software tradicionales. Estas diferencias afectan el proceso de desarrollo de software y al mismo producto de software en sí.

Desplazamiento del esfuerzo de desarrollo al análisis. Un enfoque orientado a objetos mueve mucho del esfuerzo de desarrollo del software a la fase de análisis del ciclo de vida. Es a veces desconcertante gastar mucho tiempo durante el análisis y diseño, pero este esfuerzo es más que compensado por una implementación más fácil y rápida. Además, los cambios futuros son más fáciles porque el diseño resultante es más claro y más adaptable.

Énfasis en la estructura de los datos antes que en las funciones. Un enfoque orientado a objetos concentra su atención en la estructura de los datos en vez de en las funciones que se van a realizar. Este cambio de énfasis le da al proceso de desarrollo una base más estable y permite el uso de un concepto de software unificante único a lo largo del proceso: el concepto de un objeto. Todos los demás conceptos tales como funciones, relaciones y eventos, se organizan alrededor de los objetos de tal forma que la información recogida durante el análisis no se pierde o transforma cuando viene el diseño y la implementación. Es menos probable que cambien las estructuras de datos de una aplicaión y las relaciones entre ellas que las operaciones realizadas en los datos. Organizar un sistema alrededor de objetos en vez de funciones le da al proceso de desarrollo una estabilidad que falta en enfoques orientados a funciones. Los objetos encapsulados están más protegidos de los efectos de cambio mediante interfaces públicas que esconden su implementación interna privada.

Proceso de desarrollo continuo. Puesto que un enfoque orientado a objetos define un conjunto de objetos orientados al problema desde el principio y continúa usando y extendiendo estos objetos a lo largo del ciclo de desarrollo, la separación entre fases del ciclo de vida es mucho menos marcada. En la Técnica de Modelado de Objetos, el modelo de objetos desarrollado durante el análisis es usado por el diseño y la implementación, y el trabajo se encamina a refinar el modelo a niveles progresivamente más detallados en vez de convertir de una representación a otra. El proceso es

continuo porque no hay discontinuidades en las cuales una notación en una fase se reemplace por una notación diferente en otra fase.

Iterativo en vez de secuencial Aunque la descripción de la OMT es por necesidad lineal, el proceso de desarrollo real es iterativo. La continuidad del desarrollo orientado a objetos hace más fácil repetir los pasos de desarrollo a niveles de detalle progresivamente más finos. Cada iteración añade o clarifica características en vez de modificar el trabajo que ya se ha hecho, con lo cual hay menos posibilidades de introducir inconsistencias y errores.

2.3.2 Conceptos básicos en el modelado de objetos

Un modelo de objetos describe la estructura de datos estática de objetos, clases y sus relaciones entre sí. Un *objeto* es un concepto, abstracción o ente con fronteras bien definidas y un significado propio para una aplicación determinada. Todos los objetos tienen una identidad y son distinguibles. Una *clase de objetos* o simplemente *clase*, describe un conjunto de objetos con propiedades, comportamiento, relaciones con otros objetos y semántica similares. Al agrupar los objetos comunes en clases, se está abstrayendo el problema. Objetos y clases a menudo aparecen como sustantivos en las descripciones de problemas.

Un *atributo* o *campo* es una propiedad de los objetos en una clase; cada atributo tiene un valor para cada objeto. En estos términos, un objeto es una *instancia* de una *clase* descrita por atributos. La noción de un objeto es sinónima de la de entidad en los modelos ER y LRDM.

Las cajas en la figura 2.2 denotan clases de objetos. La clase equipo, por ejemplo, tiene campos nombre, costo y peso.

Una *operación* es una acción, función o transformación que puede aplicarse o ser aplicada por los objetos. Una misma operación se puede aplicar a distintas clases, en cuyo caso es *polimórfica*, esto es, la misma operación toma formas diferentes en clases diferentes. Un *método* es la implementación de una operación en una clase. Por ejemplo, la clase *Archivo* de la figura 2.3 podría tener una función *Imprime*. Métodos diferentes podrían implementarse para imprimir distintos tipos de archivos como ASCII o binarios. Las operaciones, de existir, se listan en el tercio más bajo de una caja de clase pero se pueden emitir de los diagramas de alto nivel.

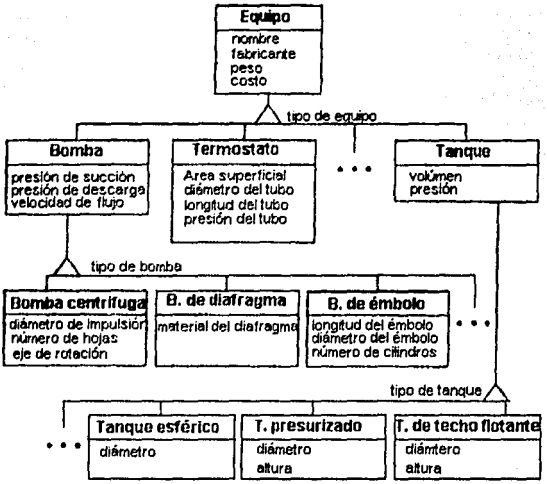


Figura 2.2: Generalización

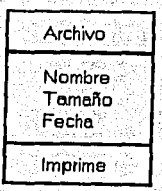


Figura 2.3: Operaciones

Relaciones

Una *relación* es una liga lógica entre objetos. Hay tres tipos de relaciones: *generalización, agregación y asociación.* Una relación se indica con una o varias lineas entre objetos.

Símbolos especiales en los extremos de una línea de relación indican cuántos objetos de una clase se relacionan con cada objeto de otra clase, es decir, la *multiplicidad* de la relación. Por ejemplo, un pequeño círculo relleno significa muchos. Muchos, en este contexto, es cero o más. Un pequeño círculo vacío significa cero o uno. Una línea recta que termine sin algún símbolo denota exactamente uno.

Relación de generalización

Una relación de generalización particiona una clase en subclases mutuamente exclusivas. La generalización puede tener un número arbitrario de niveles. Los triángulos en la figura 2.2 simbolizan una generalización. Para la generalización de arriba , "equipo" es la clase principal o *super clase*, "bomba"y "tanque" son subclases. La superclase guarda datos generales tales como nombre, costo y peso; las subclases guardan datos particulares para cada tipo de equipo. Análogamente, para la generalización de abajo, "bomba" es la super clase mientras que "bomba centrífuga" y "bomba de diafragma" son subclases.

Cada caja en la figura 2.2 corresponde a una clase de objetos, no a un objeto en sí. Un mismo objeto es representado en cada nivel de la generalización. Y los atributos son heredados desde el nivel más alto hacia abajo. Cada bomba centrífuga tiene un nombre de equipo, costo, peso, presión de succión, presión de descarga, velocidad de flujo, diámetro de impulsión, número de paletas y eje de rotación.

La OMT soporta herencia múltiple. Cada objeto puede participar en más de una generalización.

Relación de agregación

La agregación es una relación del tipo "parte de". Combina objetos de niveles bajos en objetos compuestos. La agregación puede ser multinivel y recursiva. Por ejemplo, una estructura de datos puede referirse a sí misma recursivamente.

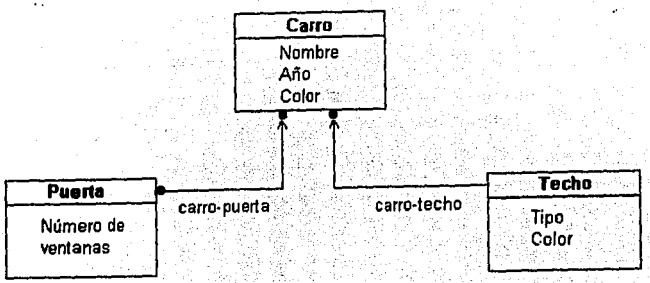


Figura 2.4: Agregación

Como se muestra en la figura 2.4, un techo es parte de un carro, muchas puertas son parte de un carro. El mismo tipo de puerta y techo puede ser usado para una variedad de carros. En este caso, "carro" es un ensamble y "puerta" y "techo" son componentes, con las flechas apuntando hacia el objeto compuesto (a veces en lugar de flechas se usan pequeños rombos).

Relación de asociación

Una asociación relaciona 2 o más objetos independientes. Usualmente aparecen como verbos en la sentencia del problema. Las asociaciones pueden ser binarias, ternarias o de un orden mayor, aunque en la práctica la gran mayoría son binarias. Asociaciones de ordenes mayores pueden convertirse en una serie de asociaciones binarias. Las asociaciones son bidireccionales inherentemente.

La figura 2.5 muestra que muchos empleados trabajan para una compañía y un empleado supervisa a otros empleados. Arbitrariamente un empleado se restringió a trabajar para una sola compañía. La elección de objetos, relaciones y su multiplicidad depende del dominio del problema. Las asociaciones pueden tener una o más propiedades.

Un *atributo de asociación* es una propiedad de la asociación. La notación para un atributo de asociación es una caja pegada a la asociación mediante un arco, uno o varios atributos aparecen en la segunda región de la caja. En

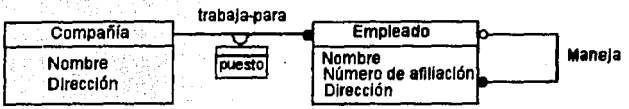


Figura 2.5: Asociación

la figura 2.5 se muestra que la asociación "trabaja-para" tiene el"puesto" como atributo.

Calificación de relaciones

Una calificación relaciona 2 clases de objetos y un calificador. El calificador es un atributo especial que reduce la multiplicidad efectiva de una asociación. Una asociación calificada puede considerarse una forma de asociación ternaria. Sólo las relaciones 1-n o n-n pueden ser calificadas.

La figura 2.6 representa una agregación con y sin calificación. Una planta tiene muchas piezas de equipos que se distinguen por el nombre del equipo, este último es el calificador. En ambas formas se refleja que una planta tiene muchas piezas de equipo, sin embargo, la forma calificada añade un único nombre a cada pieza de equipo en una planta dada. Para hallar una pieza de equipo primero se escoge la planta y luego se especifica el nombre del equipo.

La calificación es una ventaja mayor del enfoque OMT. Ocurre frecuentemente y proporciona un soporte semántico especial.

2.3.3 Análisis

El análisis, primer paso de esta metodología, se encarga de encontrar un modelo del mundo real preciso, conciso, entendible y correcto. El propósito del análisis orientado a objetos es modelar el sistema del mundo real de tal forma que pueda ser entendido. Para hacer esto se deben examinar requerimientos, analizar sus implicaciones y volverlos a establecer rigurosamente. Primero se deben abstraer características importantes del mundo real y dejar detalles pequeños para después. El modelo de análisis

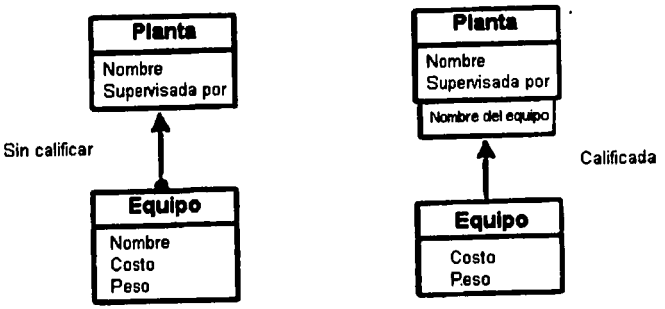


Figura 2.6: Relaciones de agregación

exitoso establece qué debe hacerse, sin restringirse a cómo se debe hacer y evita decisiones de implementación. El resultado del análisis deberá ser entender el problema como una preparación para el diseño.

El modelo de análisis sirve para varios propósitos: clarifica los requerimientos, provee una base para la concordancia entre el desarrollador del software y el encuestador y deviene en la plataforma para el futuro diseño e implementación.

El análisis empieza con un planteamiento del problema generado por clientes y posiblemente los desarrolladores. El planteamiento puede ser incompleto o informal, el análisis lo hace más preciso y descubre ambigüedades e inconsistencias.

En seguida se debe entender el sistema del mundo real descrito por el planteamiento del problema y sus características esenciales deben abstraerse en un modelo. Los planteamientos en lenguaje natural son generalmente ambiguos, incompletos e inconsistentes. El modelo de análisis es una representación concisa y precisa del problema que permite responder preguntas y construir soluciones. Los pasos subsecuentes de diseño se refieren al modelo de análisis en vez de al planteamiento del problema original. Tal vez aún más importante, el proceso de construir un modelo riguroso del dominio del problema fuerza al desarrollador a confrontar malentendidos desde el principio en el proceso de desarrollo, cuando aún son fáciles de corregir.

El modelo de análisis cubre los tres aspectos básicos de un objeto:

estructura estática (modelo de objetos), secuencia de interacciones (modelo dinámico) y transformaciones de datos (modelo funcional).

Planteamiento del problema

El primer paso al desarrollar algo es establecer los requerimientos. El planteamiento del problema debe ser un planteamiento de necesidades, no un propósito de solución, y es sólo un punto de partida. El propósito del análisis subsecuente es entender completamente el problema y sus implicaciones. El analista debe trabajar con el encuestador para refinar los requerimientos.

2.3.4 Modelado de objetos

El primer paso al analizar los requerimientos es construir un modelo de objetos, el cual muestra la estructura de datos estática del sistema del mundo real y la organiza en piezas manejables. El modelo de objetos describe clases de objetos del mundo real y sus relaciones entre ellas.

La información para el modelo de objetos viene del planteamiento del problema, el conocimiento experto del dominio de aplicación y el conocimiento general del mundo real.

Para construir un modelo de objetos se siguen los siguientes pasos :

- Identificar objetos y clases
- Preparar un diccionario de datos
- Identificar asociaciones entre objetos
- Identificar atributos de objetos y ligas
- Organizar y simplificar clases de objetos usando herencia
- Verificar que existan rutas de acceso para preguntas comunes
- Iterar y refinar el modelo

Identificación de clases de objetos

El primer paso para construir un modelo de objetos es identificar clases de objetos relevantes del dominio de aplicación, los cuales pueden ser entidades físicas o conceptuales. No todas las clases son explícitas en el planteamiento del problema, algunas están implícitas en el dominio de aplicación o conocimiento general.

Se empieza por listar clases de objetos candidatas encontradas en la descripción escrita del problema. Las clases a menudo corresponden a los sustantivos. Al principio no debe preocupar mucho la herencia o clases de alto nivel. Más tarde se pueden organizar en categorías más amplias buscando similitudes y diferencias entre las clases básicas.

En seguida se descartan las clases innecesarias e incorrectas de acuerdo a los siguientes lineamientos generales:

- *Clases redundantes.* Si dos clases expresan la misma información, se debe quedar el nombre más descriptivo.

- *Clases irrelevantes.* Si una clase tiene poco o nada que ver con el problema, debe ser eliminada. Esto implica emitir un juicio de valor, porque en otro contexto la clase pudiera ser importante.

- *Clases vagas.* Una clase debe ser específica. Algunas clases tentativas pudieran tener fronteras no bien definidas o abarcar mucho en su alcance.

- *Atributos.* Deben restablecerse como atributos aquellos nombres que describen preliminarmente objetos individuales. Si la existencia independiente de una propiedad es importante, entonces se hace clase y no atributo.

- *Operaciones.* Si un nombre describe una operación que se aplica a objetos y no se manipula correctamente, entonces no es una clase. Una operación que tiene características de sí misma debe modelarse como una clase.

- *Roles.* El nombre de una clase debe reflejar su naturaleza intrínseca y no un rol que juega en la asociación. Una entidad física a veces corresponde a varias clases.

- *Construcciones de implementación.* Deben eliminarse del modelo de análisis aquellas construcciones extrañas al mundo real; tal vez se necesiten más tarde durante el diseño.

Preparación de un diccionario de datos.

Las palabras solas tienen muchas interpretaciones, por tanto se debe preparar un diccionario de datos para todas las entidades modeladas, escribiendo un párrafo donde se describa de manera precisa cada clase de objetos. Se describe el alcance de la clase dentro del problema, incluyendo cualquier restricción en su uso o calidad de miembro. El diccionario de datos también describe asociaciones, atributos y operaciones.

Identificación de asociaciones.

Cualquier dependencia entre dos o más clases es una asociación. Una referencia de una clase a otra es también una asociación. Las asociaciones frecuentemente corresponden a verbos o frases verbales, las cuales incluyen ubicación física (parte de, contenido en), acciones dirigidas (conduce), comunicación (habla a), propiedad (tiene, parte de) o satisfacción de alguna condición (trabaja para, casado con, maneja).

En seguida se descartan las asociaciones innecesarias e incorrectas usando los siguientes criterios:

- *Asociaciones entre clases eliminadas.* Si una de las clases en la asociación ha sido eliminada, entonces la asociación se debe eliminar o reestablecerse en términos de otras clases.

- *Asociaciones irrelevantes o de implementación.* Se elimina cualquier asociación que esté fuera del dominio del problema o trate con construcciones de implementación.

- *Acciones.* Una asociación debe describir una propiedad estructural del dominio de aplicación, no un evento transitorio. A veces un requerimiento expresado como acción implica una relación estructural esencial y debe ser establecida de acuerdo a ello.

- *Asociaciones ternarias.* Muchas de las asociaciones entre tres o más clases se pueden descomponer en asociaciones binarias o establecerse

como asociaciones calificadas. Si un término en una asociación ternaria es puramente descriptivo y no tiene características propias, entonces el término es un atributo de liga en una asociación binaria. Ocasionalmente se requiere una asociación ternaria general, al no poder descomponerse en asociaciones binarias sin pérdida de información.

- *Asociaciones derivadas.* Se omiten asociaciones que puedan ser definidas en términos de otras asociaciones porque son redundantes. También se omiten asociaciones definidas por condiciones en los atributos de objeto.

Se debe procurar lo más posible que las clases, atributos y asociaciones en el modelo de objetos representen información independiente. Las rutas múltiples entre clases a menudo indican asociaciones derivadas que son la composición de asociaciones primitivas. No obstante, no todas las asociaciones que forman rutas múltiples entre clases indican redundancia. A veces la existencia de alguna asociación puede derivarse de dos o más asociaciones primitivas pero no así la multiplicidad. Si la multiplicidad es importante, la asociación extra se debe mantener.

Aunque las asociaciones derivadas no añaden información, son útiles en el mundo real y en el diseño. Se pueden enseñar asociaciones derivadas en los diagramas de objetos, pero deben dibujarse usando lineas punteadas para indicar su estado dependiente y para distinguirlas de asociaciones fundamentales.

La semántica de las asociaciones se especifica teniendo en cuenta:

- *Asociaciones mal llamadas.* No se debe decir cómo o por qué se produce una situación, simplemente se debe decir qué es. Los nombres son importantes para entender y deben esogerse con mucho cuidado.

- *Nombres de papeles.* El nombre del papel describe el papel que juega una clase en la asociación desde el punto de vista de la otra clase. Se deben añadir nombres de papel donde sea apropiado, para así distinguir mejor el papel de las clases en la asociación. Si sólo hay una asociación entre un par de clases y el nombre de la clase describe su papel adecuadamente, éstos se pueden omitir. Una asociación entre dos instancias de la misma clase (asociación reflexiva) requiere nombres de papel para distinguir las instancias.

- *Asociaciones calificadas.* Usualmente un nombre identifica un objeto dentro de un contexto; muchos de los nombres no son únicos globalmente. El contexto se combina con el nombre para identificar al objeto de manera única. Un calificador distingue objetos en el lado de multiplicidad "muchos" de una asociación.

- *Asociaciones faltantes.* Se añade cualquier asociación faltante que sea descubierta.

Si una asociación tiene importancia relevante dentro del problema, puede convertirse en clase, con atributos propios de la asociación.

Identificación de atributos.

Los atributos son propiedades de objetos individuales tales como nombre, peso, velocidad o color. Los atributos no deben ser objetos; se usa una asociación para enseñar cualquier relación entre dos objetos. Se deben considerar atributos que se relacionen directamente a una aplicación particular.

Durante el análisis, evítense atributos que son solamente para la implementación.

Los atributos derivados deben omitirse; éstos, como objetos y asociaciones derivadas, pueden ser útiles en abstraer propiedades significativas de una aplicación, pero deben ser claramente distinguibles de atributos básicos que definen el estado del objeto. Los atributos derivados no deben expresarse como operaciones, como obten_edad, aunque eventualmente pueden implementarse como tal.

Los atributos de liga también se deben identificar. Un atributo de liga es una propiedad de la liga entre dos objetos, en vez de ser una propiedad de un objeto individual.

Se eliminan los atributos innecesarios e incorrectos con los siguientes criterios:

- *Objetos.* Si la existencia independiente de una propiedad es importante, en vez de sólo su valor, entonces es un objeto. La distinción depende de la aplicación.

- *Calificadores.* Si el valor de un atributo depende de un contexto en particular, entonces el atributo debe ser establecido como calificador.

- *Nombres.* Los nombres son a veces mejor modelados como calificadores en vez de atributos de objetos. Un nombre es un atributo de objeto cuando no depende del contexto, especialmente cuando no necesita ser único.
- *Atributos de liga.*Si una propiedad depende de la presencia de una liga, entonces la propiedad es un atributo de la liga y no del objeto.
- *Valores internos.* Si un atributo describe el estado interno de un objeto que es invisible fuera del objeto, se debe eliminar del análisis.
- *Detalles finos.* Omítanse aquellos atributos menores que no afecten operaciones.
- *Atributos discordantes.* Un atributo que parezca completamente diferente y sin conexión a los otros atributos puede indicar una clase que debe partirse en dos clases distintas. Una clase debe ser simple y coherente.

Refinación con herencia

El siguiente paso es organizar clases mediante el uso de la herencia para compartir estructuras comunes. La herencia puede añadirse en dos direcciones: generalizando aspectos comunes de clases existentes en una superclase o refinando clases existentes en subclases especializadas. En el primer caso, la herencia se puede descubrir buscando clases con atributos, asociaciones u operaciones similares. Para cada generalización se define una superclase para compartir características comunes. En el otro, las especializaciones son frecuentemente aparentes del dominio de aplicación, donde los subcasos enumerados son la fuente más frecuente de especialización. Si las especializaciones propuestas son incompatibles con una clase existente, puede ser que ésta esté formulada de manera impropia.

La herencia múltiple se puede usar para incrementar el compartir, pero sólo si es necesario, ya que aumenta la complejidad conceptual y de implementación.

Cuando el mismo nombre de asociación aparece más de una vez con el mismo significado sustancialmente, se debe tratar de generalizar las clases asociadas. Los atributos y asociaciones deben asignarse a la clase más general para la cual sea apropiado.

2.3.5 Modelo dinámico

El modelo dinámico muestra el comportamiento dependiente del tiempo del sistema y los objetos en él. Los valores de atributos y relaciones de un objeto definen su *estado*. Durante el tiempo, los objetos se estimulan entre sí, resultando en una serie de cambios en sus estados. Un *evento* es un estímulo individual de un objeto a otro. El análisis dinámico se empieza buscando eventos-estímulos y respuestas externamente visibles. Luego se resumen secuencias de eventos permisibles para cada objeto con un *diagrama de estado*. Un diagrama de estado es una red de estados y eventos. El modelo dinámico consiste de múltiples diagramas de estado, uno para cada clase con un comportamiento dinámico importante, y muestra el patrón de actividad para el sistema entero.

El modelo dinámico es importante para sistemas interactivos.

Primero se preparan escenarios de diálogos típicos. Aunque estos escenarios pueden no cubrir cada contingencia, al menos aseguran que interacciones comunes no sean pasadas por alto. De los escenarios se extraen eventos. Es mejor primero identificar eventos y después asignar cada evento a su objeto correspondiente. Luego se organiza la secuencia de eventos y estados en un diagrama de estado.

Preparación de un escenario

Un *escenario* es una secuencia de eventos. Un evento ocurre cuando se intercambia información entre un objeto del sistema y un agente externo, como un usuario, un censor u otra tarea. Los valores de la información intercambiada son los parámetros del evento, y opcionalmente se muestran entre paréntesis, después del nombre del evento. Los eventos sin parámetros carecen de significado. La información en tales eventos es el hecho de que ha ocurrido, una mera señal. Un evento ocurre cada vez que se introduce información al sistema o se saca del sistema.

Para cada evento se identifica al actor (sistema, usuario o agente externo) que causó el evento y los parámetros del evento, en caso de ser relevantes. Algunos ejemplos de eventos con parámetros o atributos son: cadena introducida (texto), dígito marcado (dígito).

Se prepara uno o más diálogos típicos entre usuario y sistema para obtener una semblanza del comportamiento esperado del sistema. Estos

escenarios enseñan las interacciones mayores, formatos de despliegue externos e intercambios de la información. El modelo dinámico se aproxima por escenarios.

A veces, el planteamineto del problema describe la secuencia de interacción completa pero las más de las veces se tiene que inventar el formato de interacción. Primero se preparan escenarios para casos "normales", interacciones sin entradas extrañas ni errores de condición. Después se consideran casos especiales como secuencias de entrada omitidas, valores mínimo y máximo y valores repetidos. Luego se consideran casos de errores de usuarios. Finalmente se consideran varios otros tipos de interacciones que pueden recaer en interacciones básicas, tales como solicitud de ayuda y status.

La figura 2.7 muestra un escenario para usar una línea telefónica. Éste escenario sólo contiene eventos que afectan la línea telefónica.

Formatos de interfaz

Muchas de las interacciones se pueden separar en dos partes: la lógica de la aplicación y la interfaz del usuario. El análisis se debe concentrar primero en el flujo y control de la información en vez del formato de presentación. El modelo dinámico captura la lógica de control de la aplicación.

Identificación de eventos

Se examinan los escenarios para identificar todos los eventos externos. Los eventos incluyen todas las señales, entradas, decisiones, transiciones y acciones de o para usuarios o dispositivos externos. Los cómputos internos no son eventos, a excepción de los puntos de decisión que interactúan con el mundo externo. Se usan los escenarios para encontrar eventos normales, pero no deben olvidarse el manejo de errores de condición y de eventos poco usuales.

Una acción de un objeto que transmite información también es un evento. La mayoría de las interacciones y operaciones objeto a objeto corresponden a eventos. Se agrupan juntos bajo un sólo nombre aquellos eventos que tienen el mismo efecto en el flujo de control, aún si sus parámetros difieren.

Se relaciona cada tipo de evento a las clases de objeto que lo envían y reciben. El evento es un evento de salida para el que lo envía y un evento de

- El que llama levanta la bocina
- Comienza tono de marcar
- El que llama marca el número (5)
- Finaliza el tono de marcar
- El que llama marca el número (5)
- El que llama marca el número (5)
- El que llama marca el número (1)
- El que llama marca el número (2)
- El que llama marca el número (3)
- El que llama marca el número (4)
- El teléfono receptor empieza a sonar
- Comienza el tono de llamar en el teléfono que llama
- Contestan a donde se llama
- El teléfono al que se llama deja de sonar
- Desaparece el tono de llamar en el teléfono que llama
- Los teléfonos se conectan
- Cuelga la persona a quien se llama
- Los teléfonos se desconectan
- El que llama cuelga la bocina

Figura 2.7: Escenario para una llamada telefónica

entrada para el receptor. A veces un objeto se envía un evento a sí mismo, en cuyo caso el evento es a la vez una entrada y una salida para la misma clase.

El siguiente paso después de escribir un escenario es identificar para cada evento los objetos que envían y reciben.

Se muestra cada escenario como una *traza de eventos* -una lista ordenada de eventos entre objetos diferentes asignadas a columnas en una tabla. Este diagrama muestra a cada objeto como una línea vertical, y a cada evento como una flecha horizontal que vá del objeto que envía al objeto receptor. El tiempo se incrementa de arriba hacia abajo. Viendo una columna particular en la traza se pueden obtener los eventos que afectan directamente un objeto en particular. Sólo estos eventos pueden aparecer en el diagrama de estados para el objeto.

La figura 2.8 muestra una traza de eventos para una llamada telefónica.

Se muestran los eventos entre un grupo de clases en un *diagrama de flujo de eventos.* Este diagrama resume los eventos entre clases, sin considerar la secuencia. Se incluyen eventos de todos los escenarios, incluyendo eventos de error. El diagrama de flujo de eventos es la contraparte dinámica de un diagrama de objetos.

Construcción de un diagrama de estados

Se prepara un diagrama de estados para cada clase con comportamiento no trivial, enseñando los eventos que el objeto recibe y envía. Cada escenario o traza de eventos corresponde a una ruta a través del diagrama de estados.

Se empieza con los diagramas de traza de eventos que afectan la clase que está siendo modelada. Se escoge una traza que enseñe una interacción típica y sólo se consideran los eventos que afectan un sólo objeto. Los eventos se arreglan en una ruta cuyos arcos se etiquetan con los eventos de entrada y salida encontrados a lo largo de una columna en la traza. El intervalo entre cualesquiera dos eventos es un *estado*, y especifica el contexto en el que se interpretan los eventos. Una transición entre estados representa la respuesta a un evento, incluyendo el siguiente estado y posibles acciones y eventos enviados a otros objetos. Se le da un nombre a cada estado si el nombre es significativo. El diagrama inicial será una secuencia de eventos y estados. Cuando un evento se recibe, el siguiente estado depende del estado actual y del evento; un cambio de estado causado por un evento es una *transición.*

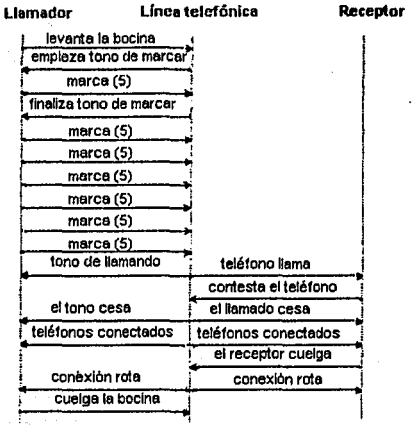


Figura 2.8: Traza de eventos para una llamada telefónica

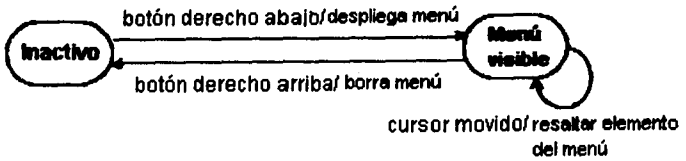


Figura 2.9: Acciones para un menú "pop-up"

Un diagrama de estados es una gráfica cuyos nodos son estados y cuyos arcos dirigidos son transiciones etiquetadas por eventos. Una *acción* es una operación instantánea en respuesta a un evento. Un tipo de acción es enviar un evento a otro objeto. Una *actividad* es una secuencia de acciones que toman algún tiempo en completarse y se puede igualar con un estado o un diagrama de estado completo. El resultado de una actividad puede usarse como una decisión para escoger el siguiente estado.

Un estado se dibuja como una caja redondeada que contiene un nombre opcional. Una transición se dibuja como una flecha que va del estado receptor al estado destino; la etiqueta en la flecha es el nombre del evento causante de la transición. La notación "Haz: A" dentro de una caja de estado indica que la actividad A empieza al entrar al estado y finaliza al salir. La notación para una acción en una transición es una diagonal y el nombre o descripción de la acción, siguiendo el nombre del evento que la causa. La figura 2.9 muestra el diagrama de estado para un menú "pop-up" de Windows.

A continuación, se mezclan otros escenarios en el diagrama de estados. Se encuentra el punto en cada escenario donde diverge de escenarios previos. Este punto corresponde a un estado existente en el diagrama. Se incrusta la nueva secuencia de eventos al estado existente como una ruta alternativa. Mientras se examinan estados y escenarios puede ser que se piense en otros eventos posibles que pueden ocurrir en cada estado, en cuyo caso se deberán añadir también al diagrama de estados.

Después de que se han considerado eventos normales, se añaden casos frontera y casos especiales. El manejar errores de usuario de una manera

clara frecuentemente requiere más trabajo y código que en casos normales, complicando la estructura del programa de otra manera limpia y clara, pero debe hacerse así.

El diagrama de estados de una clase se termina cuando el diagrama cubre todos todos los escenarios posibles y maneja todos los eventos que pueden afectar a un objeto de la clase en cada uno de sus estados. Se puede usar el diagrama de estados para sugerir nuevos escenarios considerando la manera como afecta el estado del objeto algún evento no tomado en cuenta.

Se repite el procedimiento anterior de construir diagramas de estados para cada clase de objetos, concentrándose en clases con interacciones importantes.

No todas las clases necesitan un diagrama de estados. Muchos objetos responden a eventos de entrada independientemente de su historia pasada, o capturan toda la historia importante en forma de parámetros que no afectan el control. Tales objetos pueden recibir y enviar eventos. Se listan los eventos de entrada para cada objeto y los eventos de salida enviados en respuesta a cada evento de entrada, pero no habrá más estructura de estados.

La figura 2.10 muestra el diagrama de estado para la línea telefónica.

Apareamiento de eventos con objetos

Cuando los diagramas de estados para cada clase estén completos, se debe cotejar que sean consistentes entre sí a nivel del sistema. Cada evento debe tener un remitente y un receptor, que ocasionalmente puede ser el mismo objeto. Los efectos de un evento de entrada se siguen de un objeto a otro a lo largo del sistema para estar seguros de que concuerdan con los escenarios. Se debe asegurar que eventos correspondientes en diferentes diagramas de estados sean consistentes. El conjunto de diagramas de estados para clases de objetos con un comportamiento dinámico importante constituye el modelo dinámico para una aplicación.

2.3.6 Modelo funcional

El modelo funcional muestra cómo se calculan los valores, sin considerar secuencia, decisiones o la estructura de los objetos. El modelo funcional refleja qué valores dependen de qué otros valores y las funciones que los relacionan.

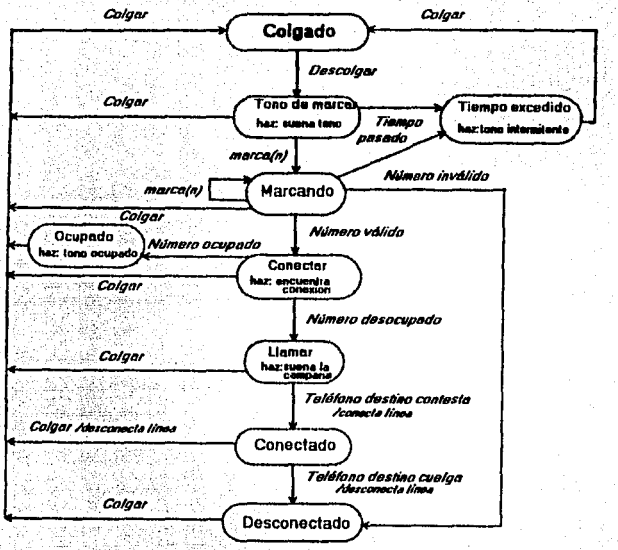


Figura 2.10: Diagrama de estado para la línea telefónica.

El modelo funcional consiste de múltiples diagramas de flujos de datos. Recordemos que un *diagrama de flujo de datos* (DFD) muestra las relaciones funcionales de los valores calculados por un sistema, incluyendo valores de entrada, de salida y almacenes internos de datos. Un DFD contiene *procesos* que transforman los datos, *flujos de datos* que mueven los datos, *objetos actores* que producen y consumen datos, y objetos *almacenes de datos* que guardan los datos pasivamente.

Los diagramas de flujo de datos son útiles para mostrar dependencias funcionales. Las funciones se expresan de varias formas, incluyendo lenguaje natural, ecuaciones matemáticas y seudocódigo.

Un DFD es una gráfica de procesos, flujos de datos, almacenes de datos y actores. Los *procesos* transforman los valores de los datos. Los procesos de bajo nivel son operaciones simples en los objetos, pero los procesos de alto nivel pueden contener almacenes internos de datos sujetos a efectos colaterales. Un DFD es un proceso en sí. Los flujos de datos relacionan los valores en los procesos, almacenes de datos y actores. Los actores son objetos independientes que consumen y producen valores. Los almacenes de datos son objetos pasivos que rompen con el flujo de control introduciendo retardos entre la creación y el uso de datos.

Los DFD's pueden ser anidados jerárquicamente, y los procesos últimos se deben especificar directamente como operaciones. Las operaciones se pueden especificar por muchos medios, incluyendo ecuaciones matemáticas, tablas y restricciones entre las entradas y las salidas, y también en seudocódigo. Las consultas son operaciones sin efectos laterales y se pueden implementar como funciones puras. Las acciones son operaciones con efectos secundarios pero sin duración, pueden ser implementadas como procedimientos. Las actividades son operaciones con efectos laterales y duración, deben ser implementadas como tareas. Las operaciones pueden pegarse a las clases dentro del modelo de objetos e implementarse como métodos.

Los procesos en un diagrama de flujo de datos corresponden a actividades o acciones en los diagramas de estados de las clases. Los flujos en un diagrama de flujo de datos corresponden a objetos o valores de atributo en un diagrama de objetos. Es mejor construir el modelo funcional después de haber hecho los modelos de objetos y dinámico.

Se llevan a cabo los siguientes pasos para construir el modelo funcional:

- Identificar valores de entrada y salida.

- Construir diagramas de flujo de datos mostrando dependencias funcionales.
- Describir funciones.
- Identificar restricciones.
- Especificar criterios de optimización.

Identificar valores de entrada y salida.

Se empieza por listar valores de entrada y salida. Los valores de entrada y salida son parámetros de eventos entre el sistema y el mundo exterior. Se examina el planteamiento del problema para encontrar cualquier valor de entrada o salida que se haya olvidado. Los eventos de entrada que sólo afecten el flujo de control, tales como cancelar, terminar o continuar, no generan valores de entrada.

Construcción de diagramas de flujo de datos

En seguida se construye un diagrama de flujo de datos enseñando cómo se calcula cada valor de salida a partir de otros valores y sobre todo a partir de los valores de entrada. Los diagramas de flujo de datos interactúan con objetos internos que sirven como almacenes de datos a través de iteraciones, los cuales se representan mediante barras paralelas.

Un diagrama de flujo de datos se construye usualmente por capas. La capa más alta puede consistir de un sólo proceso o tal vez varios, cada uno de los cuales reúne entradas, calcula valores y genera salidas.

Dentro de cada capa del diagrama de flujo de datos, se trabaja hacia atrás desde cada valor de salida para determinar la función que lo calcula. Si las entradas a la operación son todas entradas del diagrama completo, se habrá terminado. De otro modo algunas de las entradas de la operación son valores intermedios que deben ser seguidos hacia atrás. También se podría llevar un seguimiento hacia adelante de las entradas a las salidas, pero usualmente es más difícil identificar todos los usos de una entrada que identificar todos los orígenes de una salida.

Se expande cada proceso no trivial en el diagrama de nivel más alto en un diagrama de flujo de datos de un nivel más bajo. Si los procesos del segundo nivel aún contienen procesos no triviales, pueden expandirse recursivamente.

Muchos sistemas contienen objetos de almacenamiento interno que retienen valores a través de iteraciones. Un almacén interno puede distinguirse de un flujo de datos o un proceso porque recibe valores que no resultan en salidas inmediatas sino que se van a utilizar en un momento posterior, probablemente lejano.

Los diagramas de flujo de datos especifican sólo dependencias entre las operaciones; no enseñan decisiones o secuencias de operaciones: algunas operaciones pueden ser opcionales o mutuamente exclusivas. Tales decisiones de secuencia son parte del modelo dinámico, no del modelo funcional. Algunos valores de datos afectan decisiones en el modelo dinámico. Las decisiones no afectan directamente valores de salida en el modelo de flujo de datos. Sin embargo puede ser útil capturar funciones de decisión en el modelo de flujo de datos, pues pueden ser funciones complicadas de valores de entrada. Las funciones de decisión pueden mostrarse en el diagrama de flujo de datos, pero sus salidas son señales de control, indicadas por flechas de salida punteadas. Estas funciones son "sumideros de datos" dentro del diagrama de flujo de datos; sus salidas afectan el flujo de control en el modelo dinámico y no valores de salida directamente. Ejemplos de funciones de decisión son aquéllas que en un momento dado verifican ciertos datos.

La figura 2.11 muestra un diagrama de flujo de datos para el despliegue de un ícono en un sistema de ventanas. El diagrama muestra la secuencia de transformaciones realizadas, así como los valores externos y objetos que afectan el cálculo.

Descripción de funciones

Cuando el diagrama de flujo de datos ha sido lo suficientemente refinado, se escribe una descripción de cada función. La descripción puede ser en lenguaje natural, ecuaciones matemáticas, seudocódigo, tablas de decisión u otra forma apropiada, enfocándose en qué es lo que hace la función, no en cómo implementarla. La descripción puede ser declarativa o procedural. Por ejemplo, una descripción declarativa de una función que ordene y descarte valores duplicados pudiera ser:"cada valor en la lista de entrada aparece exactamente una sola vez en la lista de salida, la cual únicamente contiene valores de la lista de entrada y los valores en la lista de salida están en orden estrictamente creciente". Una descripción procedural especifica una función mediante un algoritmo para calcularla. El propósito del algoritmo es sólo

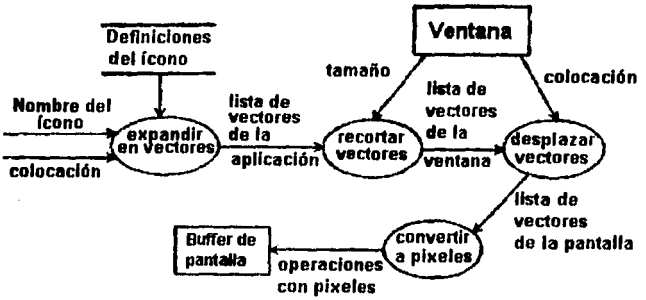


Figura 2.11: Diagrama de flujo de datos para un despliegue gráfico de ventanas

especificar lo que hace la función, ya que durante la implementación puede usarse cualquier otro algoritmo que calcule los mismos valores. Es preferible usar descripciones declarativas puesto que no implican una implementación, pero si es más fácil y clara una descripción procedural, ésta debería usarse.

Identificación de restricciones entre objetos

Las restricciones son dependencias funcionales entre objetos que no están relacionadas por una dependencia de entrada-salida. Pueden darse entre dos objetos al mismo tiempo, entre instancias del mismo objeto en momentos distintos (una *invariante*) o entre instancias de diferentes objetos en momentos distintos (aunque estas últimas son generalmente funciones de entrada-salida). Las precondiciones en funciones son restricciones que los valores de entrada deben satisfacer, y las postcondiciones son restricciones que los valores de salida deben cumplir. Se deben establecer los tiempos o condiciones bajo los cuales se cumplan las restricciones. El analista debe incorporar las restricciones en los modelos dinámico y funcional para completar la especificación.

Un ejemplo de restricción en un problema contable sería: "Ningún balance de cuenta puede ser negativo".

Especificación de criterios de optimización

Se especifican valores a ser maximizados, minimizados u optimizados de algún otro modo, sin ser necesario ser muy precisos en este punto.

Un criterio de optimización para nuestro ejemplo sería minimizar el tiempo en que un registro de un alumno se bloquea, si es que tal bloqueo es necesario.

Agregado de operaciones

Las operaciones pueden corresponder a preguntas acerca de atributos o asociaciones en el modelo de objetos, eventos en el modelo dinámico y funciones en el modelo funcional. Es más útil distinguir estos tipos de operaciones durante el análisis. Las operaciones clave deben resumirse ahora en el modelo de objetos. Se descubren durante los pasos subsecuentes del análisis.

Operaciones del modelo de objetos

Las operaciones de la estructura de objetos incluyen leer y escribir valores de atributos y ligas de asociaciones. Estas operaciones no necesitan mostrarse explícitamente en el modelo de objetos, pero están implícitas por la presencia de un atributo. Durante el análisis se asume que todos los atributos son accesibles.

Operaciones de eventos

Cada evento enviado a un objeto corresponde a una operación en el objeto. Dependiendo de la arquitectura del sistema los eventos pueden convertirse en métodos explícitos. Durante el análisis, los eventos se representan mejor como niveles en las transiciones de estado y no deben ser listados explícitamente en el modelo de objetos.

Operaciones de funciones

Cada función en el diagrama de flujo de datos corresponde a una operación en un objeto (o posiblemente varios objetos). Estas funciones frecuentemente

tienen una estructura computacional interesante y deben ser resumidas en el modelo de objetos. Las funciones se organizan en operaciones sobre objetos.

Si la misma serie de ecuaciones o fragmentos de seudocódigo describen a más de una función, entonces se puede introducir una nueva operación para simplificar el modelo funcional.

Simplificación de las operaciones

Se examina el modelo de objetos para encontrar operaciones similares y variaciones en cuanto a forma de una misma operación. Se trata de ampliar la definición de una operación para cubrir tales variaciones y casos especiales. Se usa la herencia donde sea posible para reducir el número de operaciones distintas. Se introducen nuevas superclases conforme se necesiten para simplificar las operaciones, siempre y cuando las nuevas superclases no sean forzadas y sean naturales. Luego se localiza cada operación al nivel correcto dentro de la jerarquía de clases.

2.3.7 Iteración del análisis

La mayoría de los modelos de análisis requieren de más de un paso para estar completos. La mayoría de los planteamientos de problemas contienen circularidades y la mayoría de las aplicaciones no se pueden enfocar de una manera completamente lineal, puesto que interactúan distintas partes del problema. Para entender un problema con todas sus implicaciones, se debe atacar el análisis iterativamente, preparando una primera aproximación al modelo y después iterando el análisis conforme el entendimiento se incrementa. No hay una línea divisoria clara entre análisis y diseño. El análisis final se debe verificar con los expertos en el dominio de aplicación.

2.3.8 Refinamiento del modelo de análisis

El modelo de análisis puede mostrar inconsistencias dentro de algunos modelos. Se hace necesario iterar los distintos pasos para producir un diseño más limpio y más coherente. Trátese de refinar las definiciones de objetos para incrementar lo que se comparte y mejorar la estructura, añadiendo detalles que se pasaron de alto durante el primer paso.

Algunas construcciones parecerá que no se acoplan correctamente. En tal caso se deben reexaminar cuidadosamente, pues se pudieran tener conceptos erróneos.

A veces, se necesita una restructuración mayor del modelo conforme se incrementa el conocimiento que se tiene. Cuando hay muchas construcciones que parecen similares pero no cuadran juntas de manera correcta, tal vez se olvidó un concepto más general. Se debe entonces de buscar generalizaciones erróneas hechas en atributos.

Una omisión común es un objeto físico que tiene dos aspectos lógicamente distintos. Estos deben ser modelados con un objeto distinto para cada aspecto.

Indicaciones que se deben buscar incluyen excepciones, muchos casos especiales, falta de simetría esperada y un objeto con dos o más conjuntos de atributos u operaciones sin relación. Se debe considerar restructurar el modelo para capturar mejor las restricciones dentro de su estructura. Se deben quitar aquellas asociaciones u objetos que parecían útiles primero pero ahora parecen extraños. Frecuentemente dos objetos distintos en el análisis pueden combinarse porque la distinción entre ellos no afecta al resto del modelo significativamente.

El análisis verificado final sirve como la base para la arquitectura del sistema, diseño e implementación. El planteamiento del problema original debe ser revisado para incorporar correcciones y entender los descubrimientos realizados durante el análisis.

2.3.9 Diseño lógico de bases de datos

La palabra modelo se usa frecuentemente en tres diferentes niveles. En el nivel más alto, un modelo es realmente una metodología para modelar. Por tanto, el modelo orientado a objetos es una metodología que se puede usar para crear modelos en el segundo nivel. Un modelo de segundo nivel es un esquema de base de datos que define los tipos de datos a capturarse para una aplicación en particular. Se designa un modelo de datos a este nivel para capturar sólo esos detalles acerca de las operaciones que los usuarios juzgan relevante. Se ignoran aquellos datos irrelevantes que consumirían recursos innecesariamente. Cuando se implementa un esquema de base de datos y se capturan datos reales, se construye una base de datos. Esta base de datos implementada es un modelo de tercer nivel, pues modela el estado actual del

dominio por el cual existe.

Una base de datos incorpora un modelo de la realidad. El DBMS maneja la base de datos de tal forma que cada usuario pueda grabar, accesar y manipular los datos que constituyen su modelo de la realidad. Si el mapeo entre los elementos reales y los elementos del modelo se lleva a cabo apropiadamente, entonces el modelo puede usarse para resolver el problema, de lo contrario no se producirá una solución correcta. En este contexto, el modelo orientado a objetos es una metodología para crear esquemas de bases de datos para situaciones de aplicación particular. Estos esquemas de bases de datos son modelos en sí mismos que proveen la estructura lógica para capturar hechos acerca de una porción particular de la realidad. Cuando estos hechos son capturados y grabados en un sistema computacional de bases de datos, entonces la base de datos en sí misma es un modelo del estado actual de la realidad.

El término "orientado a objetos" presume una representación computacional de las entidades del mundo real como 'objetos' que tienen atributos y participan en relaciones, no como registros en los sistemas tradicionales orientados a archivos. Los modelos de datos orientados a objetos reciben también el nombre de modelos semánticos porque mapean el significado de las cosas en la realidad a construcciones en el modelo.

Los modelos orientados a objetos consisten de conjuntos de objetos, relaciones, atributos, indicadores de cardinalidad (multiplicidad) y llaves. Los conjuntos de objetos pueden ser léxicos, si contienen instancias que pueden ser impresas, o abstractos, si contienen instancias que no pueden ser impresas. Las instancias en los conjuntos de objetos abstractos se representan por medio de 'llaves sustitutas', que son identificadores internos sin significado externo.

Las relaciones establecen conexiones entre instancias de conjuntos de objetos, y pueden ser vistas como conjuntos de objetos que se llaman conjuntos de objetos agregados, los cuales pueden tener atributos y participar en otras relaciones. A las relaciones que involucran 2 conjuntos de objetos se les llama binarias y a las que involucran 3 o más, relaciones de alto nivel o n -adas. Cualquier relación de alto nivel puede ser descompuesta en una serie de relaciones binarias y se puede asumir que éstas son relaciones binarias de muchos a muchos. La cardinalidad de una relación se refiere al máximo número de instancias en un conjunto de objetos que están relacionadas con una única instancia del otro conjunto de objetos. Las cardinalidades entre

relaciones son de uno a uno, de uno a muchos y de muchos a muchos.

Para cualquier instancia dada en un conjunto de objetos, el valor de su atributo está determinado de manera única. Si una instancia de objeto en particular no tiene valor para alguno de sus atributos, se dice que ese atributo tiene un valor nulo para la instancia de objeto. Los atributos deben permanecer conceptualmente separados de los objetos que describen: los valores de los atributos pueden cambiar pero el objeto asociado con ellos es el mismo.

Los conjuntos de objetos de especialización son subconjuntos de otro conjunto de objetos. Por otro lado, un conjunto de objetos de generalización contiene a otro conjunto de objetos. Si un objeto es una especialización de otro objeto entonces el objeto de especialización hereda todos los atributos y relaciones del objeto del cual se especializa. La herencia es la propiedad de un conjunto de especialización que le permite tener todos los atributos y relaciones de su conjunto de generalización. Los conjuntos de especialización son valiosos porque proveen un medio para definir atributos para algunas instancias sin la necesidad de definirlos para otras.

Las llaves se usan para identificar objetos de manera única. Las llaves sustitutas son identificadores internos que no pueden usarse fuera del sistema, lo cual se logra con el uso de llaves externas, que son conjuntos de atributos léxicos que pueden usarse conjuntamente para identificar un elemento de un conjunto de objetos.

Al analizar las preguntas que los usuarios quieren responder y los reportes que la organización necesita, se pueden construir modelos de datos orientados a objetos. Estos modelos de datos identifican conjuntos de objetos, sus atributos, conjuntos de especialización y relaciones, conjuntamente con las cardinalidades de las relaciones.

En un proyecto grande de desarrollo de bases de datos, analistas diferentes trabajan con grupos de usuarios diferentes para crear varios modelos de datos. Para crear una única base de datos unificada, se deben integrar estos modelos de datos o vistas de la base de datos. El proceso involucra dejar las vistas intactas en lo posible, quitando sólo esos conjuntos de objetos, relaciones y atributos que son redundantes entre las vistas, y conectando éstas definiendo nuevas relaciones entre conjuntos de objetos en las diferentes vistas conforme sea necesario. Los proyectos de desarrollo de bases de datos que son particularmente grandes pueden provocar una decisión de crear un número de bases de datos más pequeñas en vez de una sola.

2.3.10 Aplicación de la OMT al diseño de bases de datos relacionales

El paradigma orientado a objetos OMT es versátil. No sólo provee una base para diseñar sistemas y programar código, sino que también puede ser usado para diseñar bases de datos. El uso de un diseño orientado a objetos trasciende el modelo de la base de datos que se escoja: no sólo se pueden diseñar bases de datos relacionales con este enfoque, sino también jerárquicas, de red y orientadas a objetos.

Una base de datos se diseña mediante la realización del análisis descrito con anterioridad y construyendo un *modelo de objetos.*

A continuación se consideran cuestiones de implementación que permitirán mapear un modelo de objetos a estructuras de bases de datos en *sí.*

Tres Niveles de representación

La figura 2.12 resume esta metodología de diseño de bases de datos.

Esta metodología usa tres niveles de representación: el nivel *alto*, el *medio* y el *bajo.* Estos tres niveles describen el mismo problema en diferentes niveles de abstracción.

El modelo de datos lógico inicial es sucesivamente transformado en tablas relacionales ideales y después en comandos de definición de datos de un sistema manejador de bases de datos. Los niveles múltiples resultan ser una construcción útil para fomentar el diseño lógico de bases de datos mientras se está aún adecuando la fase de implementación.

El *nivel alto* se enfoca a la estructura de datos fundamental. Este nivel es un subconjunto de la notación gráfica que fue desarrollada por Loomis, Stah y Rumbaugh para la programación orientada a objetos (OMT). Un diseño de bases de datos orientadas a objetos presenta una abstracción lógica de datos simple y concisa que es sencilla de implementar con un DBMS comercial.

El *nivel medio* contiene tablas genéricas independientes de un DBMS y en él se manejan cuestiones tales como el mapeo de estructuras objeto en tablas, dominios y llaves.

El *nivel bajo* es el lenguaje de definición de datos del DBMS a usar. Este nivel contiene los comandos del DBMS que crean tablas, atributos e índices y considera detalles específicos tales como la colocación de tablas dentro de

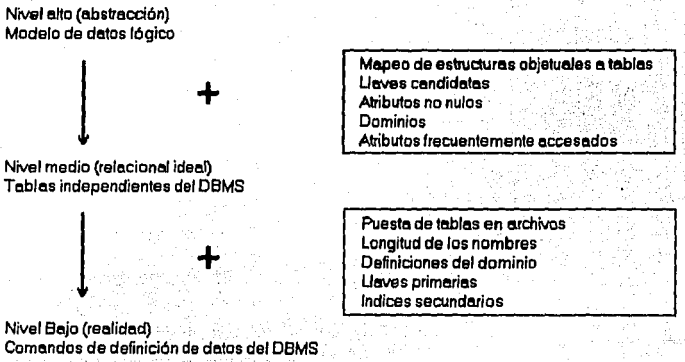


Figura 2.12: Tres niveles de representación

archivos de bases de datos, un conjunto limitado de tipos de datos y restricciones arbitrarias.

Representación del nivel alto

Es la representación que se hace de objetos, relaciones y calificación de relaciones, ya descrita anteriormente.

Representación del nivel medio

El nivel medio mapea las estructuras objeto del nivel alto en tablas genéricas. El nivel medio se ocupa del problema general de mapear clases en tablas tomando como punto de partida la idiosincracia misma de un DBMS, lográndose con esto mejorar la documentación y facilitar la migración a un nuevo DBMS. Además se ha observado que las tablas resultantes tienden a estar en tercera forma normal; por esto se puede decir que la tercera forma normal es un beneficio intrínseco del modelado de objetos. Las formas normales mejoran la integridad de los datos. Recordemos que una tabla está en primera forma normal cuando cada valor de atributo es atómico y no contiene un grupo repetido. Una tabla está en segunda forma normal cuando está en primera forma normal y cada renglón tiene una única llave. Una tabla está en tercera forma normal cuando satisface la segunda forma normal y cada atributo que no sea llave depende directamente de la llave primaria.

En este contexto, la primera formal normal se alcanza descomponiendo objetos complejos. El alcance de esta descomposición depende del significado de *atómico* y la aplicación. Por ejemplo, podría ser perfectamente razonable considerar un arreglo como objeto atómico cuando éste guarda la composición de un fluído. Sin embargo, en un contexto diferente un arreglo podría requerir de una descomposición.

Veamos ahora por qué las tablas derivadas de objetos satisfacen la segunda forma normal. Anteriormente se definieron los objetos como entes que existen y se pueden distinguir; por tanto, los objetos tienen una única llave cuando se proveen con información de distinguibilidad. Las relaciones se dan entre objetos, por tanto éstas también tienen una única llave, la que resulta de la combinación de llaves de los objetos involucrados.

La demanda de la tercera forma normal es en cambio un poco débil. La mayoría de las violaciones de la tercera forma normal parecen ocurrir

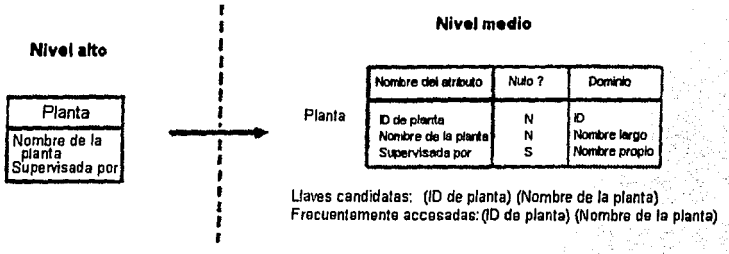


Figura 2.13: Nivel medio para un objeto

cuando se introduce información extraña en una tabla o la tabla carece de un enfoque suficientemente bueno. Las tablas relacionales permiten construcciones irreales y están a un nivel muy bajo para diseñar. El paradigma de objetos está a un nivel de abstracción más alto y tiende a bloquear diseños irracionales. Los objetos son menos flexibles y menos peligrosos que las tablas relacionales. El método de construir un modelo de datos de un número pequeño de entidades coherentes es superior al enfoque tradicional de reunir todos los atributos, indagar las dependencias funcionales y sintetizar tablas.

Objetos

Cada clase de objetos se mapea directamente a una tabla. Todos los campos objeto devienen atributos de las tablas. En la figura 2.13 se introduce un

atributo adicional: "Identificador de Planta". Cada objeto tiene un único identificador, todas las referencias a los objetos se hacen por medio de su identificador. La identidad de los objetos está implícita en diagramas de objetos y debe ser explícita en las tablas.

Hay muchas razones para adoptar un fuerte sentido de identidad de objetos. Una ventaja es que los identificadores de objetos son inmutables y completamente independientes de cambios en valores de datos y localizaciones físicas. La estabilidad de identificadores de objetos es particularmente importante para las relaciones puesto que éstas se refieren a objetos, contrastando ésto con el hecho de referirse a los objetos por su nombre. Cambiar un nombre requiere actualizar muchas relaciones. La identidad de objetos provee un mecanismo uniforme para referenciar todos los objetos.

El nivel medio controla el uso de valores nulos. *Nulo* significa que un valor de atributo es desconocido o no es aplicable para un renglón dado. 'N' los olvida, 'S' los permite. Los atributos en llaves candidatas no deben ser nulos. Esta columna le da al modelador de datos la opción de requerir valores para campos adicionales. Cada atributo tiene un dominio o conjunto de valores de atributos legales, los cuales deben ser consistentes. Los dominios aseguran decisiones en la longitud del atributo que son consistentes y previenen operaciones en entidades incompatibles.

La figura 2.13 lista llaves candidatas. Recuérdese que una llave candidata es un conjunto de atributos que identifican de manera única cada renglón. Cada atributo puede pertenecer a cero, una o más llaves candidatas. La figura 2.13 también lista grupos o atributos que son susceptibles de experimentar accesos frecuentes. Estos grupos serían candidatos primarios para indexar. El orden de los atributos dentro de un grupo pueden o no ser relevantes para la implementación del nivel bajo.

Relación de generalización

Una relación de generalización tiene una 'tabla superclase' y múltiples 'tablas subclases'. La figura 2.14 ilustra el mecanismo general. Para cada pieza de equipo, hay un 'renglón superclase' y un 'renglón subclase' con un identificador de equipo común. Recuérdese que literalmente el mismo objeto se representa en cada nivel de la generalización. El tipo de equipo es el campo superclase discriminador que particiona las subclases. Cada valor del tipo de equipo corresponde a una tabla subclase.

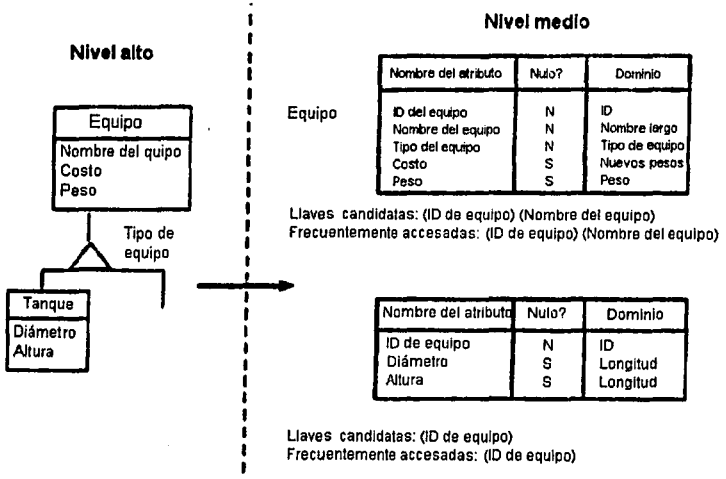


Figura 2.14: Nivel medio para una relación de generalización

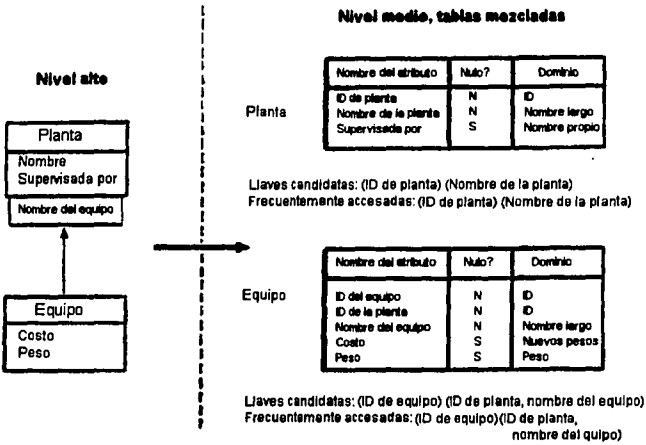


Figura 2.15: Nivel medio para una agregación de existencia-dependencia

Relación de agregación

Por necesidad, las relaciones de muchos a muchos se mapean a tablas distintas.

Relaciones uno a uno y uno a muchos se pueden mapear a tablas distintas o fusionarse con un objeto participante. El manejo de las agregaciones de uno a uno y uno a muchos depende del contexto. Las agregaciones dependientes existentes se fusionan con una tabla objeto para simplificar la ejecución de la integridad. Dentro del contexto de la aplicación para la figura 2.15, cada pieza de equipo debe asignarse a una planta. Las agregaciones libres se guardan en distintas tablas. En la figura 2.16, cada tipo de techo se usa para diferentes modelos de carros. Un techo es una parte que existe independientemente de un carro en particular.

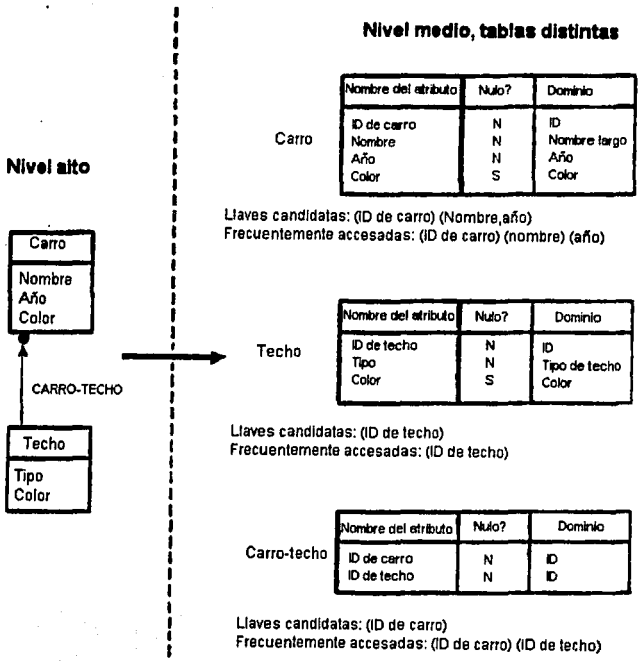


Figura 2.16: Nivel medio para una agregación libre

Relación de asociación

Como una regla, las asociaciones se mapean a tablas distintas, como en la figura 2.16. Las propiedades de una asociación devienen atributos de la tabla de asociación. En este caso, las asociaciones no se colapsan con un objeto correspondiente, como en la figura 2.15. Hay muchas razones para hacer externas las asociaciones:

1. Las asociaciones se dan entre objetos independientes de igual peso sintáctico. En general, parece inapropiado contaminar objetos con conocimientos de otros objetos.

2. El colapsar en objetos asociaciones con propiedades descriptivas, puede violar la tercera forma normal.

3. Es difícil obtener multiplicidades justo en los primeros pasos del diseño. La elección de multiplicidades es a veces una decisión arbitraria y puede cambiar conforme evoluciona el subconjunto del universo que está siendo modelado. Asociaciones uno-a-uno o uno-a-muchos pueden o no ser externalizadas, mientras que las asociaciones muchos a muchos deben serlo.

4. Una representación simétrica simplifica la búsqueda y la actualización.

Representación del nivel bajo

El nivel bajo es el lenguaje de definición de datos del DBMS destino. Este nivel contiene los comandos reales del DBMS que crean tablas, atributos e índices. Este nivel explota las características del DBMS y compensa defectos y rarezas. Los detalles específicos del nivel bajo dependen de la elección del DBMS.

Llaves primarias

La llave primaria debe ser única y ninguno de los campos que la constituyen debe ser nulo. El nivel medio identifica las llaves candidatas, una de las cuales debe escogerse como llave primaria. En general, se usa el identificador de objeto como llave primaria para las tablas objeto. La llave primaria para las tablas relaciones puede ser uno o más identificadores de los objetos

involucrados. Es conveniente hacer de los identificadores la llave primaria aún cuando éstos no tengan un significado inherente para el usuario. La mayoría de las aplicaciones son estructuralmente complejas y difíciles de navegar. Por tanto las aplicaciones complejas deben mediar el acceso del usuario con programas usuales. Si se va a restringir el acceso de las bases de datos a través de un programa, también se debería hacer el acceso a través de identificadores, los cuales nunca cambian y tienen un pequeño tamaño fijo (lo cual podría implementarse como un entero) que aumenta la velocidad de selecciones y uniones.

Índices secundarios

Los índices secundarios en la mayoría de los DBMS relacionales, juegan un doble papel. Por un lado mejoran el desempeño del sistema mediante una búsqueda rápida de los renglones con ciertos valores de atributo, y por otro pueden forzar la unicidad de las llaves candidatas.

§ 3

CASO PARTICULAR

Como ejemplo de desarrollo se presenta a continuación el problema planteado inicialmente referente a el proceso de reinscripción de alumnos en la Facultad de Ciencias, el cual se tratará desde las dos metodologías.

3.1 METODOLOGÍA TRADICIONAL

Para indicar el flujo de la información nos referiremos al diagrama de flujo de datos. Presentamos a continuación los diagramas correspondientes más importantes.

3.1.1 Yourdon-Constantine

El proceso principal de esta metodología es la descomposición funcional, iniciando con un diagrama de datos concentrados (figura 3.1) y haciendo expansiones sucesivas de procesos (figuras 3.2 y 3.3).

Para obtener el contexto de aplicación se usa el diagrama de entidad–relación de la figura 3.4, el cual se ha simplificado de tal forma que sólo contiene entidades, y las relaciones están representadas con nombres encima de las ligas entre esas entidades.

3.1.2 Warnier-Orr

Dentro del proceso de diseño lógico ubicamos las salidas del sistema, que son dos: el comprobante de inscripción y las listas de alumnos inscritos por

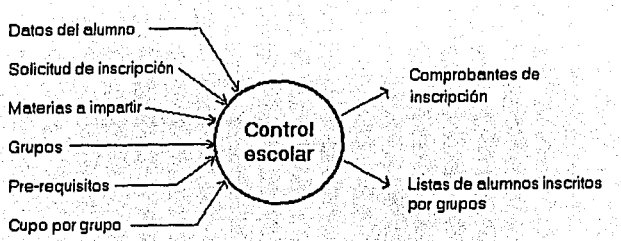


Figura 3.1: Diagrama de datos concentrados

grupos; ver figura 3.5.

Del anterior prototipo se deriva la estructura lógica de salida de la figura 3.6.

A continuación, del esquema de la estructura lógica de salida se obtiene el diagrama de la estructura lógica del proceso de la figura 3.7.

De lo anterior se obtiene finalmente la disposición de los archivos de entrada que necesitamos, ordenados por el campo subrayado. Ver figura 3.8.

3.2 METODOLOGÍA ORIENTADA A OBJETOS OMT

3.2.1 Modelo de objetos

Dentro del nivel alto del esquema de abstracción, el producto final es el diagrama del modelo de objetos en sí de la figura 3.9. En este diagrama se muestran las clases Materia, Grupo, Gpo. actual, Gpo. Anterior, Alumno y Adeudo. Los atributos de la clase Materia son clave mat.(la clave de la materia), nombre, periodo (año y semestre), carrera a la que pertenece, créditos que cubre, tipo (para indicar si es optativa u obligatoria) y nivel

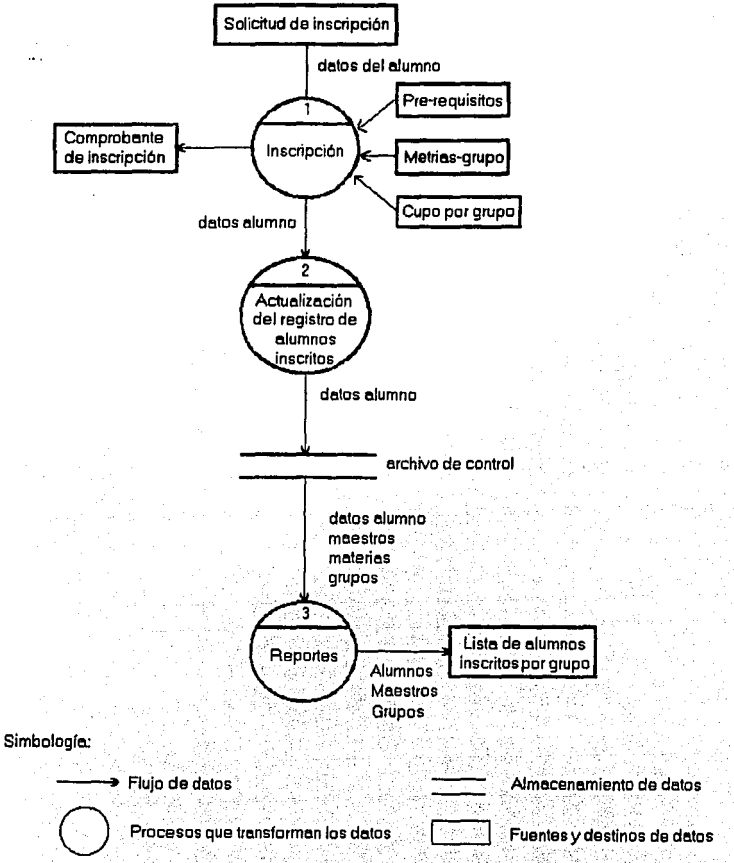


Figura 3.2: Diagrama de flujo de datos

Expansión del proceso 1:

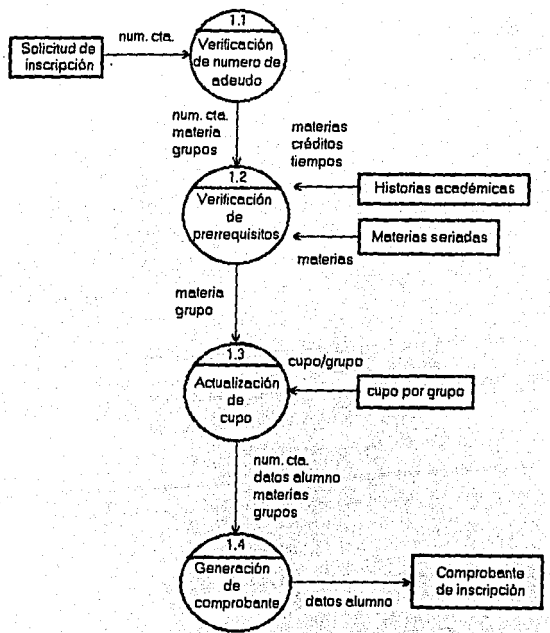


Figura 3.3: Diagrama de flujo de datos. Expansión del proceso Inscripción

Para determinar el contexto de aplicación se usa un diagrama de entidad-relación :

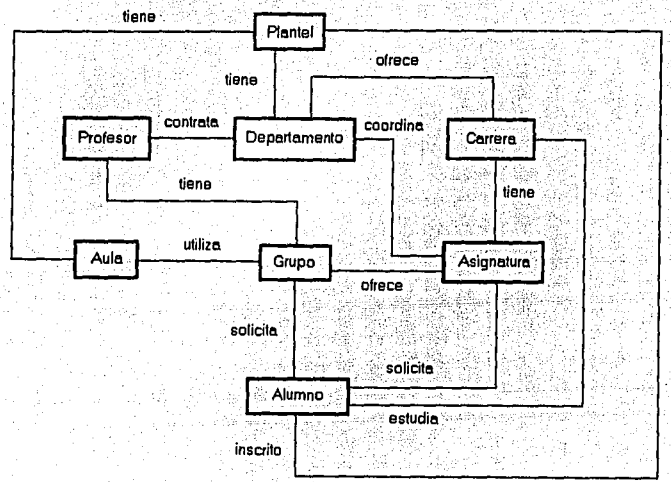


Figura 3.4: Diagrama de entidad–relación

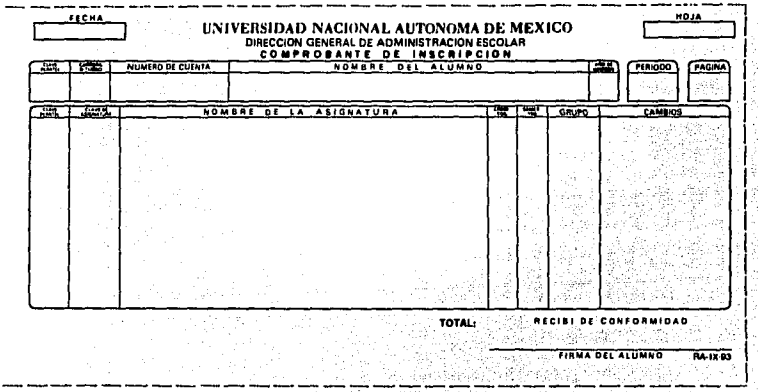


Figura 3.5: Prototipo del comprobante de inscripción

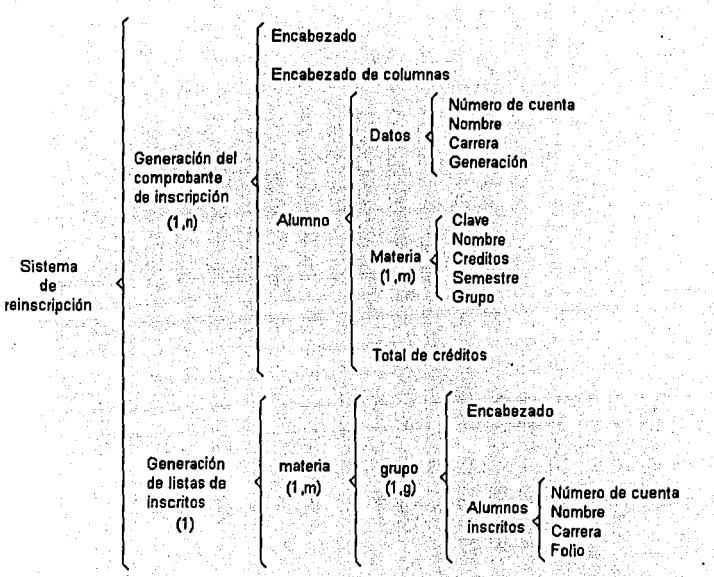


Figura 3.6: Esquema lógico de datos

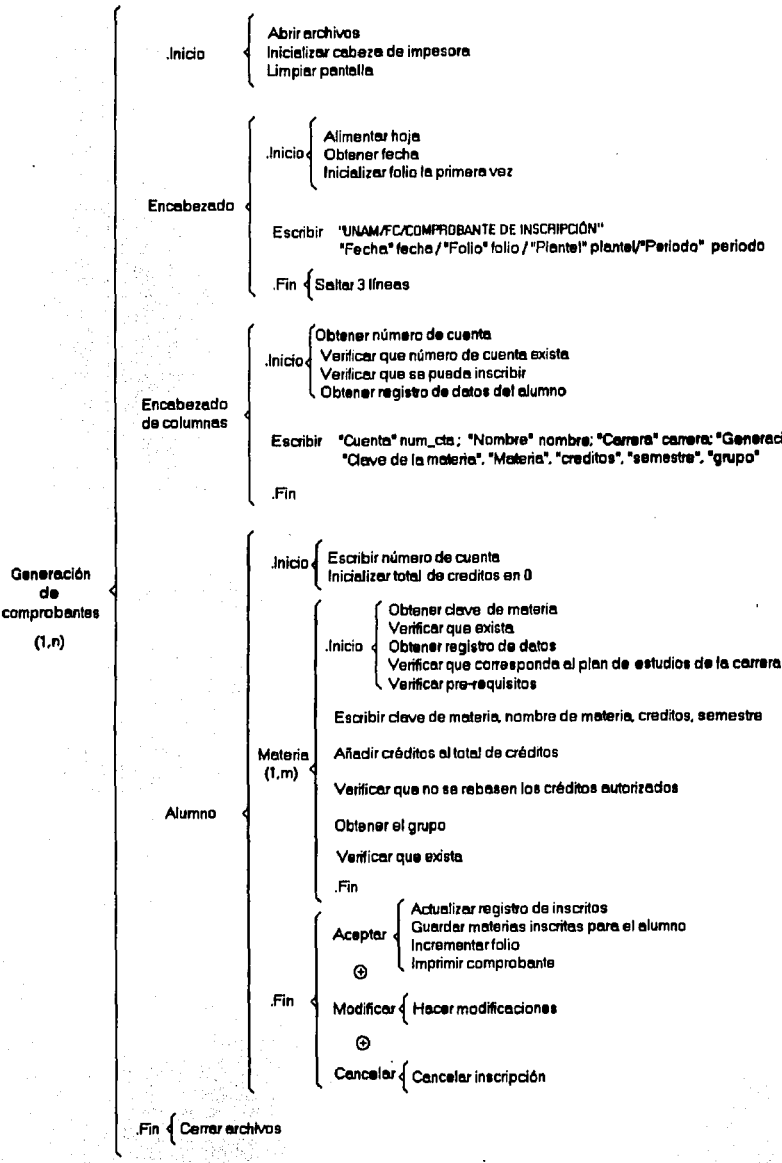


Figura 3.7: Esquema lógico del proceso.

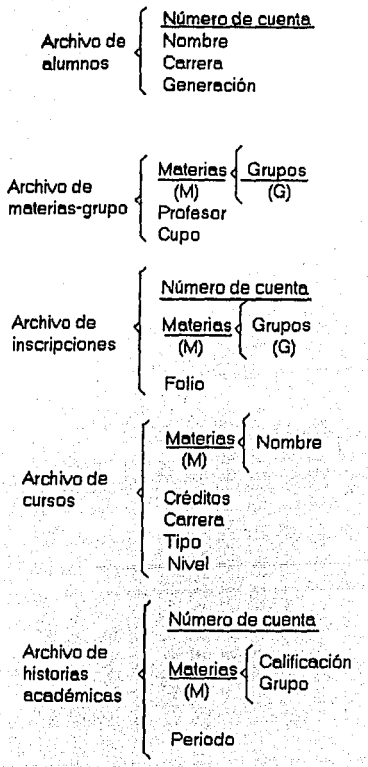


Figura 3.8: Archivos ideales de datos.

(para señalar el nivel al que pertenece dentro del plan de estudios). Entre la clase Materia y Grupo hay una relación de agregación calificada, donde el calificador es la clave del grupo, indicándose así que para una materia dada hay varios grupos que se distinguen por la clave del grupo; de esta forma para ubicar un grupo en particular primero se escoge la materia y luego la clave del grupo en sí, el cual además tiene como atributos un horario, un profesor, un cupo y un salón. La relación de generalización entre las clases Grupo, Gpo. actual y Gpo. anterior está determinada por el periodo en el que se imparte o impartió cada materia-grupo. Entre Gpo. actual y Alumno se tiene la relación "Se inscribe en", la cual tiene como tal el atributo folio, esto denota el hecho de que a una inscripción particular le corresponde un folio único, además de los datos de los grupos a los que se inscribe un alumno y los datos mismos de éste. La multiplicidad en esta relación es de muchos a muchos, ya que un alumno se puede inscribir en varios grupos y, recíprocamente, un grupo puede tener varios alumnos inscritos. Por otra parte, si el periodo de la materia-grupo no corresponde al periodo actual, se tiene la relación "Se inscribió en", la cual tiene como atributos el tipo de examen en que se presentó, el folio del acta, la calificación que obtuvo el alumno y el periodo en que se inscribió. Finalmente, entre la clase alumno y la clase Adeudo se da una relación de asociación simple, un alumno puede o no tener un adeudo de cierto tipo, mientras que varios alumnos pueden tener un mismo tipo de adeudo.

Del modelo de objetos se obtienen las tablas relacionales (figura 3.10) mediante las reglas de mapeo descritas; conformándose así el nivel medio. Obsérvense las llaves foráneas:

Clave mat. de Materia-grupo referencia a Clave mat. de Materias. Num.cta. de Inscripción referencia a Num.cta. de Alumno. Clave mat. de Inscripción referencia a Clave mat. de Materias. Num.cta. de Historias referencia a Num.cta. de Alumno. Clave mat. de Historias referencia a Clave mat. de Materias. Num.cta. de Adeudo referencia a num.cta. de Alumno.

Las relaciones cumplen con los requisitos de integridad de entidades e integridad referencial, pues todas cuentan con llaves primarias que no tienen valores nulos y los atributos de las llaves foráneas se encuantran todos contenidos en las llaves primarias. Así mismo, las relaciones están normalizadas. Todos los atributos de las relaciones son atómicos, no existen dependencias transitivas y los atributos de las relaciones dependen directamente de las llaves primarias.

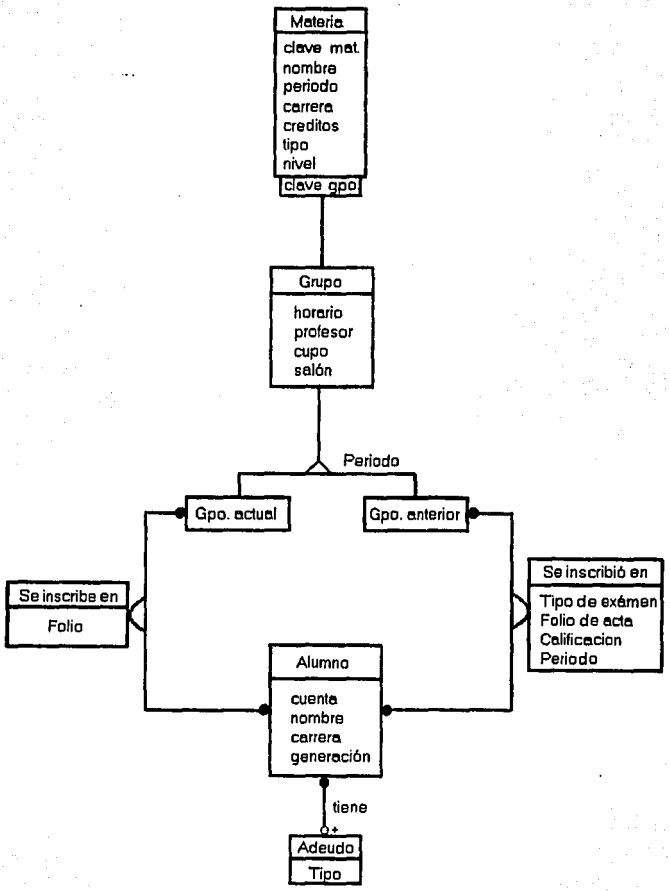


Figura 3.9: Diagrama de objetos. Nivel alto.

Nivel medio: Del modelo de objetos (nivel alto) se obtienen las siguientes tablas :

De la clase Alumno:

Alumno

Atributo	¿Nulo?	Dominio
Num. cta.	N	(c, 8)
Nombre	N	(c, 32)
Carrera	N	(c, 2)
Generación	N	(c, 2)

Llave candidata: (Num. cta.)

De la clase Materia:

Materias:

Atributo	¿Nulo?	Dominio
Clave mat.	N	(c, 4)
Nombre	N	(c, 15)
Periodo	N	(c, 3)
Carrera	N	(c, 2)
Créditos	N	(n, 2)
Tipo	N	(c, 1)
Nivel	N	(c, 1)

Llave candidata: (Clave mat)

De la asociación calificada entre materia y grupo:

Materia-grupo

Atributo	¿Nulo?	Dominio
Clave mat.	N	(c, 4)
Clave gpo.	N	(c, 4)
Horario	S	(c, 10)
Profesor	S	(c, 32)
Cupo	S	(n, 3)
Salón	S	(c, 4)

Llave candidata: (Clave mat)
(Clave gpo)

De la asociación "se inscribe en":

Inscripción

Atributo	¿Nulo?	Dominio
Num. cta.	N	(c, 8)
Clave mat.	N	(c, 4)
Clave gpo.	N	(c, 4)
Folio	N	(n, 4)
Créditos	N	(n, 3)

Llave candidata: (Num.cta.)
(Clave mat.)

De la asociación "se inscribió en":

Historias

Atributo	¿Nulo?	Dominio
Num. cta.	N	(c, 8)
Clave mat.	N	(c, 4)
Clave gpo.	N	(c, 4)
Periodo	N	(c, 3)
Tipo de ex.	N	(c, 2)
Folio	N	(n, 4)
Calif.	N	(c, 2)

Llave candidata: (Num.cta.)
(Clave mat.)

De la clase Adeudo y la asociación Alumno-tiene-Adeudo:

Adeudo

Atributo	¿Nulo?	Dominio
Num. cta.	N	(c, 8)
Tipo	N	(c, 1)

Llave candidata: (Num. cta.) (Tipo)

Figura 3.10: Nivel medio.

3.2.2 Modelo dinámico

En esta etapa se crea el escenario principal del sistema (figura 3.11), en el cual se tienen dos actores: el sistema y el usuario, siendo los parámetros de los eventos los datos que el usuario introduce y los mensajes que genera el sistema. De este escenario se obtiene la respectiva traza de eventos de la figura 3.12. Posteriormente obtenemos el diagrama de flujo de eventos de la figura 3.13 para finalmente construir el diagrama de estados de la figura 3.14.

3.2.3 Modelo funcional

Para el modelo funcional se empieza identificando los valores de entrada y salida del sistema (figura 3.15). En seguida obtenemos el diagrama de flujo de datos general (figura 3.16), del cual se derivan otros diagramas de flujo para los principales procesos (figuras 3.17 y 3.18).

- El sistema pide un número de cuenta; el usuario introduce '9478657-4'.
- El sistema verifica el número de cuenta con el directorio de alumnos y lo reconoce como válido.
- El sistema verifica que el alumno tenga derecho a inscripción.
- El sistema despliega datos generales del alumno.
- El sistema pide la clave de la materia a inscribir, el usuario introduce '0001'
- El sistema verifica que la clave sea correcta y se imparta en el semestre corriente.
- El sistema verifica que el alumno no la haya dado anteriormente, que no la tenga aprobada o reprobada 2 veces y que cumpla con los requisitos para poder cursarla.
- El sistema pide la clave del grupo para la materia; el usuario introduce '2000'
- El sistema verifica que el grupo sea correcto y que corresponda a la materia dada.
- El sistema pide confirmación para repetir el proceso, el usuario se la niega.
- El sistema despliega opciones de aceptar, cancelar o modificar, el usuario acepta.
- El sistema imprime el comprobante de inscripción para el alumno.
- El sistema pide un número de cuenta, hasta que se teclee 'fin'.
- El usuario teclea 'fin'
- El sistema despliega mensaje de despedida

Figura 3.11: Escenario normal del sistema

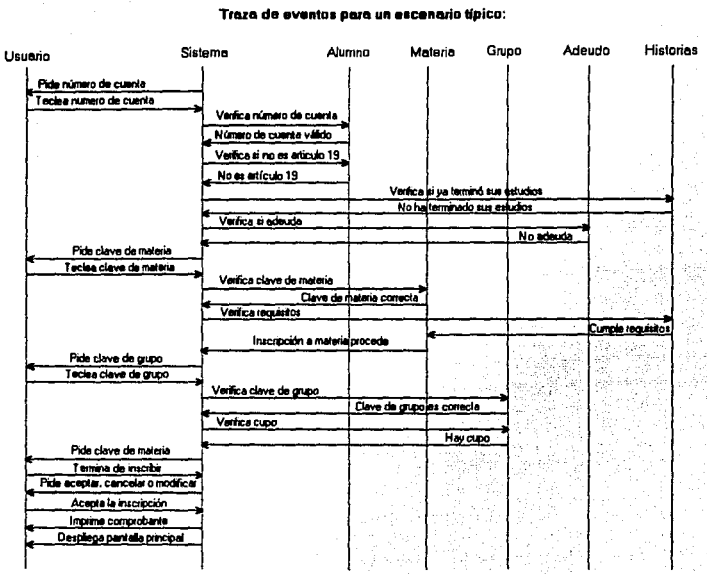


Figura 3.12: Traza de eventos para un escenario típico

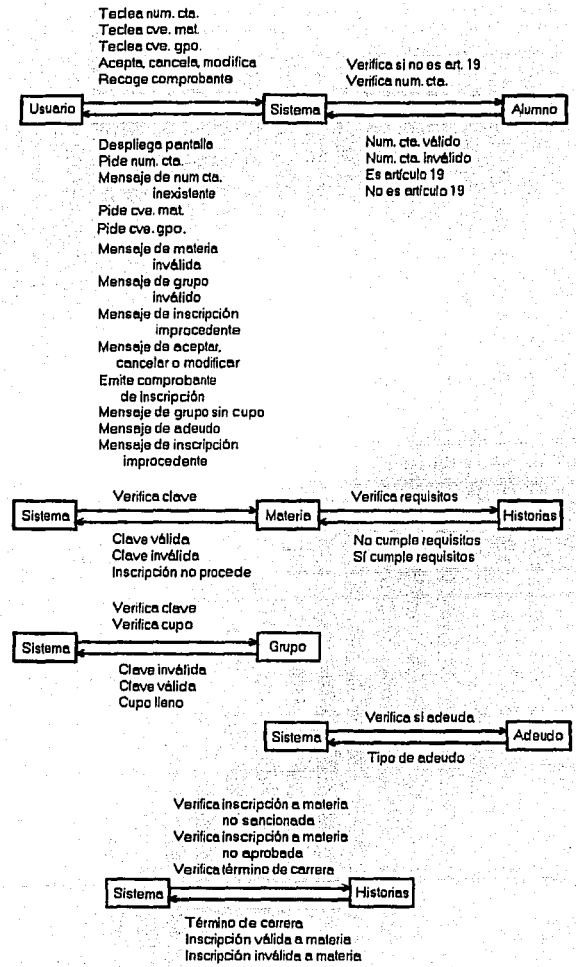
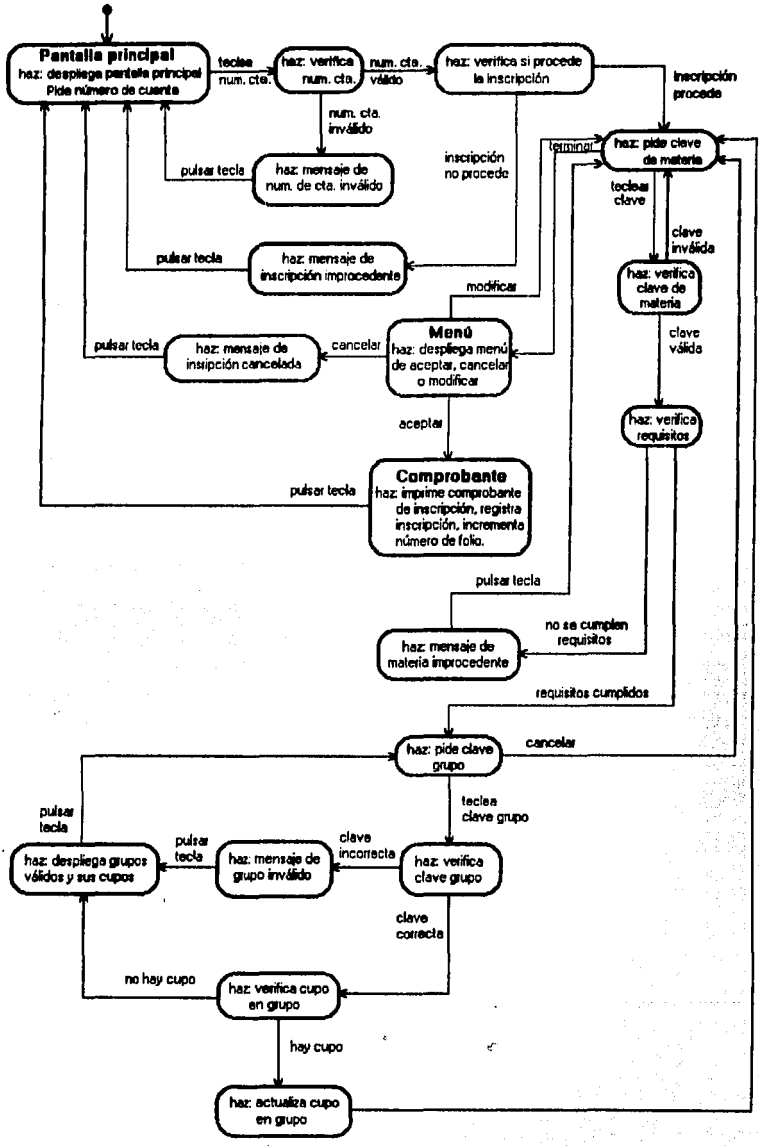


Figura 3.13: Diagramas de flujo de eventos.


Pantalla principal
haz: despliega pantalla principal
Pide número de cuenta
teclea num. cta.
haz: verifica num. cta.
num. cta. válido
haz: verifica si procede la inscripción
inscripción procede
num. cta. inválido
haz: mensaje de num. de cta. inválido
pulsar tecla
inscripción no procede
terminar
haz: pide clave de materia
teclea clave
clave inválida
haz: verifica clave de materia
pulsar tecla
haz: mensaje de inscripción improcedente
modificar
clave válida
Menú
haz: despliega menú de aceptar, cancelar o modificar
pulsar tecla
haz: mensaje de insipción cancelada
cancelar
haz: verifica requisitos
aceptar
Comprobante
haz: imprime comprobante de inscripción, registra inscripción, incrementa número de folio.
pulsar tecla
pulsar tecla
haz: mensaje de materia improcedente
no se cumplen requisitos
requisitos cumplidos
haz: pide clave grupo
cancelar
pulsar tecla
teclea clave grupo
haz: despliega grupos válidos y sus cupos
pulsar tecla
haz: mensaje de grupo inválido
clave incorrecta
haz: verifica clave grupo
clave correcta
no hay cupo
haz: verifica cupo en grupo
hay cupo
haz: actualiza cupo en grupo

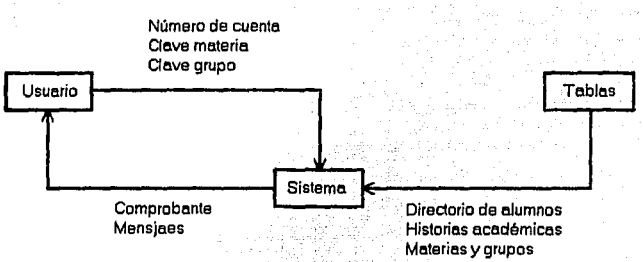


Figura 3.15: Identificación de valores de entrada y salida

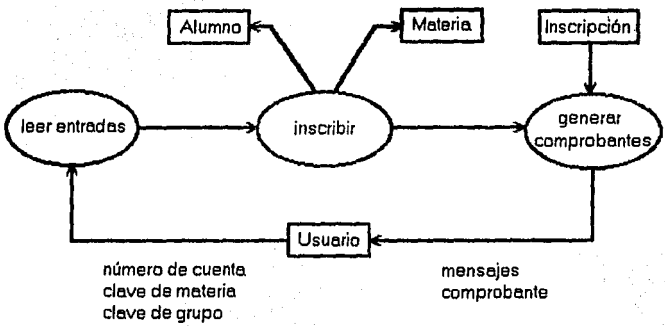


Figura 3.16: Diagrama de flujo de datos

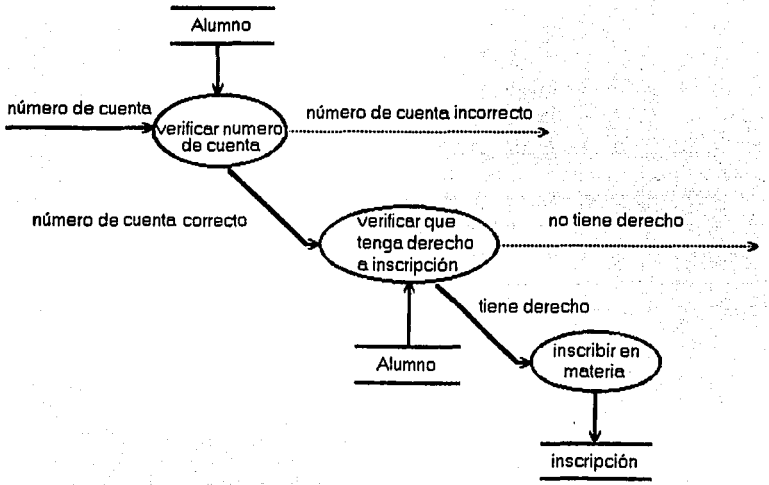


Figura 3.17: Diagrama de flujo de datos para el proceso de inscribir

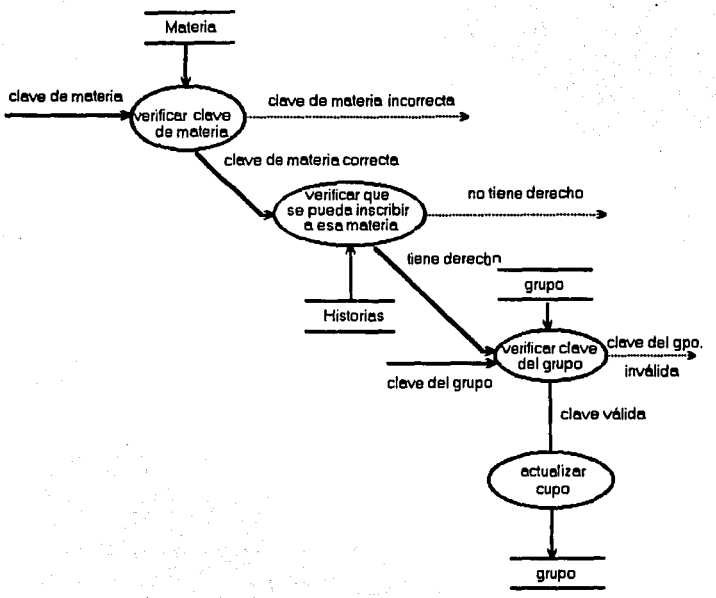


Figura 3.18: Diagrama de flujo de datos para el proceso de inscribir en materia

§ 4

CONCLUSIONES

Al igual que los métodos estructurados en su momento, el paradigma de los objetos aporta nuevas técnicas de análisis, diseño y construcción de software. Ambos elaboran modelos del mundo real que luego implantan en computadoras; sólo que, mientras que los modelos estructurados conciben la realidad como un flujo de procesos que manejan la información, los modelos orientados a objetos la entienden de modo similar a como lo hace la investigación científica: como un *sistema de objetos* portadores de información y procesos que comparten estructuras de información y comportamiento con todos los de su clase y que se comunican a través de mensajes a los cuales reaccionan polimórficamente.

En efecto, los objetos del mundo real poseen *información* y realizan *procesos* en su interior; y así son los objetos en el nuevo enfoque. Los objetos reales *heredan* de su clase propiedades y comportamiento; y así sucede con los objetos informáticos. Los objetos reales surgen y eventualmente desaparecen, los objetos informáticos se construyen y destruyen a partir de su clase. Los objetos reales se comunican de diversas maneras, los objetos implementados en computadoras reciben y emiten diferentes *mensajes*. Los objetos reales reaccionan de muchas formas ante el mismo estímulo, son polimórficos, y también lo son los objetos elaborados con el nuevo paradigma.

Los métodos estructurados fueron instrumentos poderosos para reducir la *complejidad*; los métodos orientados a objetos constituyen el siguiente paso en esta dirección. Los métodos estructurados permitieron multiplicar varias veces la capacidad humana para desarrollar software frente a los métodos artesanales, los métodos orientados a objetos permiten multiplicar

otras tantas veces esta capacidad frente a los métodos estructurados. La programación estructurada nos enseñó a modularizar procedimientos, pero mantuvo globales los datos; la programación orientada a objetos nos enseña a *encapsular* los datos en una sola unidad (el *objeto*) junto con los procedimientos. Por ello, la orientación a objetos nos permite obtener software más modificable y mantenible y hace su administración más efectiva. La programación estructurada demandó más recursos de hardware que la programación artesanal, la programación orientada a objetos permite obtener software más poderoso y compacto que la programación estructurada. Los métodos estructurados redujeron el costo de fabricación del software y aumentaron considerablemente la productividad del personal de desarrollo; los métodos orientados a objetos hacen otro tanto con relación a la programación estructurada.

La metodología estructurada de Yourdon, basada en diagramas de flujo de datos, en lo general incorpora componentes de modelado similares a los usados en la OMT. La diferencia entre ambas metodologías radica en el énfasis y estilo relativo que ponen en estos componentes. Los diseños OMT están dominados por el modelo de objetos. El paradigma en el mundo real de objetos y relaciones provee el contexto para entender el comportamiento dinámico y funcional. En contraste, el análisis y diseño estructurado acentúa la descomposición funcional. El enfoque estructurado organiza un sistema alrededor de procedimientos, en cambio, un diseño orientado a objetos organiza un sistema alrededor de objetos del mundo real o conceptuales que existen en la perspectiva que el usuario tiene del mundo. Muchos de los cambios en los requerimientos son cambios en las funciones en vez de serlo en los objetos, por lo que un cambio puede ser desastrozo en un diseño basado en funciones o procedimientos.

En cuanto al diseño de bases de datos con el enfoque objetual, se podría decir que es expresivo, eficiente, coherente y menos propenso a los problemas de actualización que se tienen con otras técnicas de diseño de bases de datos basadas en atributos. Así mismo, los sistemas manejadores de bases de datos relacionales son un vehículo de implementación viable para modelos de objetos puesto que la tecnología es madura y se tienen disponibles muchos paquetes comerciales.

En particular, en nuestro caso de análisis presentado pudimos comprobar que el enfoque del diseño de bases de datos basado en la OMT ofrece varias ventajas:

1. Es intuitivo, fácil de usar y fácil de entender.

2. Es expresivo. Provee un conjunto de construcciones para el modelado de datos más rico que enfoques alternativos.

3. Es extensible. Da un lugar para los cambios dentro del alcance del modelo de datos y se puede portar a otro DBMS.

4. Nivel de abstracción muy útil. Ataca de una buena manera los problemas del mundo real y los mapea naturalmente a un RDBMS.

5. Buen desempeño. Es fácil visualizar los patrones de acceso a los datos cuando se usa la OMT.

6. Promueve la integridad en las bases de datos. Las tablas derivadas de los objetos tienden a estar en tercera forma normal.

7. Mejora la integración. El paradigma de los objetos ayuda a cubrir la separación semántica que hay entre bases de datos y aplicaciones.

8. El modelado de objetos mejora la claridad del pensamiento y la habilidad para comunicarse durante el proceso de diseño.

Al no contarse con un OO-DBMS, el sistema se desarrolló en su totalidad con el compilador Clipper, el cual es un sistema manejador de bases de datos comercial que si bien no es en sí relacional , se escogió básicamente por las facilidades que proporciona para trabajar bajo ambientes de red. Esta elección no resultó ser un inconveniente pues como se señaló anteriormente, el diseño es independiente del lenguaje de programación que se use en la implementación.

Una de las principales diferencias entre las metodologías empleadas radica en la forma de obtener las bases de datos del sistema. Con el diseño orientado a objetos, una vez obtenido el modelo de objetos, es más fácil visualizar las relaciones entre sus elementos y mapear directamente en tablas relacionales, con las ventajas inherentes ya descritas; mientras que en el caso de la metodología Warnierr-Orr, este proceso viene a ser el resultado último del desarrollo. Además, los criterios conducentes a obtener una 'buena' base de datos deben aplicarse de manera separada. Esto mismo ocurre con el desarrollo estructurado. En ambos casos el diseño de la base de datos sigue

un enfoque guiado por atributos, mientras que en el caso objetual se sigue un enfoque guiado por entidades.

Tal vez lo más conveniente sea desarrollar separadamente estas metodologías, de tal manera que puedan verse como complementarias en la creación de un sistema desde distintas perspectivas.

APÉNDICE A

Hacia un enfoque OO/Relacional

Recientemente Hugh Darwen y C.J. Date dieron a conocer un manifiesto para la dirección futura que tomarán los datos y los sistemas manejadores de bases de datos, el cual consiste en una serie de prescripciones, proscripciones y sugerencias.

Los autores ven en el modelo relacional la base que permita una fundamentación firme para el futuro de los datos y su manipulación. Respecto a ciertas características que han sido extensamente discutidas recientemente, incluyendo algunas referentes a aspectos del enfoque objetual, creen que son ortogonales al modelo relacional, por lo que este último modelo no necesita extensiones, correciones ni perversiones para que estas características se puedan acomodar en algún lenguaje de bases de datos que pudiera representar la fundamentación buscada, sino que más bien se debería buscar un nuevo enfoque que conjugue lo mejor de esas dos tecnologías. Tal enfoque debería estar fundamentado firmemente en el modelo relacional, el cual es finalmente la fundamentación de la tecnología moderna de bases de datos en general. Es decir, lo que se quiere de los sistemas relacionales es que evolucionen para incorporar las características, o al menos las buenas características, de lo orientado a objetos.

La idea principal para ese nuevo enfoque deseado es la noción relacional de los **dominios**. Considérense los siguientes puntos:

- La construcción fundamental en los sistemas OO es la *clase de objetos*, la cual es básicamente un tipo de dato encapsulado definido por el usuario y de complejidad interna arbitraria.

- La construcción fundamental en los sistemas relacionales (la mayoría de las veces no implementada) en los productos relacionales actuales, es el *dominio*, que es básicamente (de nuevo) un tipo de dato definido por el usuario, encapsulado y de una complejidad interna arbitraria.

En otras palabras, un dominio y una clase de objetos son lo mismo, por lo que los dominios (y en consecuencia los atributos relacionales) pueden contener cualquier cosa, arreglos, listas, pilas, documentos, fotografías, mapas, etc. Bajo este planteamiento podemos deducir que los dominios encapsulan mientras que las relaciones no.

Por tanto se tendría que un sistema relacional que soporte relaciones y dominios propiamente, sería capaz de hacer cualquier cosa que los sistemas orientados a objetos pueden hacer y los relacionales no pueden.

Lo anterior contradice el enfoque que se siguió en este trabajo al identificar una clase con una relación y, por consiguiente, una tupla con un objeto, pues tenemos que bajo este enfoque el álgebra relacional, al permitir la manipulación directa de las relaciones y sus datos, pone al descubierto la estructura del objeto en sí, negando con ello un principio fundamental del paradigma orientado a objetos: el ocultamiento de la información; mientras que haciendo la ecuación clase=dominio, este problema no se dá. A la fecha, la discusión sobre esta contradicción aún continúa, dado que ambos enfoques presentan cada uno sus ventajas y desventajas. Si bien este es un tema al que todavía le queda mucho por desarrollar, podemos decir que básicamente se tiene un problema de decisión entre qué tanto es lo que se quiere encapsular en el diseño de las bases de datos.

BIBLIOGRAFÍA

[1] Blaha, M.; Premerlani, W.; Rumbaugh, J. *Relational data base design using an object-oriented methodology.* Communications of the ACM. April, 1988.

[2] Blaha, M.; Premerlani, W.; Shen, H. *Converting object-oriented models into RDBMS schema.* IEEE Software. May, 1994.

[3] Rumbaugh, J.; et. al. *Object-oriented modeling and design.* Prentice Hall, 1991.

[4] Hansen, G.; Hansen, J. *Database management and design.* Prentice Hall, 1992.

[5] Date, C. J. *An introduction to database systems.* Addison Wesley, 1995.

[6] Ullman, J. D. *Principles of database and knowledge-base systems. Vol. I.* Computer Science Press, 1989.

[7] Yourdon, E.; Constantine, L. *Structured design.* Yourdon Press, 1979.

[8] Yourdon, E.; Constantine, L. *Modern structured analysis.* Yourdon Press, 1989.

[9] Higgins, D. *Program design and construction.* Prentice Hall, 1979.

[10] Martin, J.; McClure, C. *Structured techniques. The basis for CASE.* Prentice Hall, 1988.

[11] Pressman, R. *Software engineering.* McGraw Hill, 1993.

[12] Sommerville, I. *Software engineering.* Addison Wesley, 1992.

[13] Darwen, H.; Date, C. J. *Introducing the third manifesto.* Database Programming and Design. January, 1995.