UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Facultad de Ciencias

Una aplicación de la metodología OMT para diseño orientado a objetos

T E S I S

Que para obtener el título de

A C T U A R I O

p r e s e n t a

ISABEL LAURA OLIVER SUAREZ

FACULTAD DE CIENCIAS
UNAM

Director de Tesis:
M. en C. Ma. Guadalupe E. Ibargüengoitia González

1998

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:

Una aplicación de la metodología OMT para diseño orientado a objetos

realizado por **Isabel Laura Oliver Suárez**

con número de cuenta **7420912-5** , pasante de la carrera de **Actuaría**

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis Propietario	**M. en C. Ma. Guadalupe E. Ibargüengoitia González**
Propietario	**Fis. Juan Jesús Gutierrez García**
Propietario	**Fis. Gustavo Soriano Cedillo**
Suplente	**Mat. Ana Luisa Solis González Cosio**
Suplente	**Mat. Ma. de los Angeles Bautista Zamudio**

Consejo Departamental de Matemáticas
M.en A.P. Ma. del Pilar Alonso Reyes

A Memelosqui,

la persona que me ayudó a convertirme en una persona de bien, que me enseñó a ser tenaz y a luchar por lo que creo, te dedico este trabajo como una pequeña muestra de mi afectuoso agradecimiento...

Con todo mi amor
tu hija Laura.

Y por supuesto a Cape, Javier, Fabián, Bruno y Lola porque su cariño me hace fuerte...

Ofrezco mi más sincero respeto y agradecimiento a la

M. en C. Guadalupe Ibargüengoitia

por la comprensión, dedicación, gran paciencia y sobre todo por el profesionalismo que mostró al dirigir este trabajo.

Así como también a los sinodales que se tomaron el tiempo para ayudarme a enriquecer este trabajo. En orden alfabético:

Mat. Ana Luisa Solís,

Fís. Gustavo Soriano,

Fis. Juan Jesús Gutierrez, y

Mat. María de los Angeles Bautista.

Y a mis familiares y amigos que me motivaron a cerrar el broche.

Una Aplicación de la Metodología OMT para Diseño Orientado a Objetos

Indice

Introducción.

Una metodología para desarrollo de sistemas combina herramientas y técnicas para guiar el proceso de desarrollo de sistemas de información a gran escala. La evolución de las metodologías modernas empieza a finales de los 60's con el concepto de ciclo de vida del desarrollo de sistemas (CVDS). El gran avance en el hardware y el uso de lenguajes de alto nivel han provocado la construcción de sistemas más grandes y complicados. El CVDS intenta proponer un orden en el proceso de desarrollo, descomponiéndolo en fases con salidas que sirven como entrada a la siguiente fase.

El concepto de CVDS proporciona a los desarrolladores una medida de control. A principios de los 70's, las metodologías estructuradas tenían como objetivo obtener un análisis efectivo y un diseño estable y fácil de mantener. Dichas metodologías fueron en su mayoría orientadas a procesos.

Los conceptos fundamentales de la orientación a objetos como una disciplina datan de hace dos décadas, es sólo hasta mediados de los 80's que las metodologías de análisis y diseño orientado a objetos empiezan a tomar fuerza.

La orientación a objetos abarca muchos conceptos y hasta algunos la han definido como un nuevo modelo para el desarrollo de software; también es cierto que existe un gran debate en determinar si representa o no un cambio radical con respecto a las metodologías convencionales.

Yourdon[2] ha dividido en dos corrientes a los metodologistas de la orientación a objetos:

1.- Revolucionarios. Consideran que las metodologías de la programación orientada a objetos son un cambio radical convirtiendo a las metodologías tradicionales y su forma de pensar acerca del diseño en obsoletas.

2.- Conservadores. Por el contrario, ven a la orientación de objetos como una acumulación de principios de ingeniería de software, desprendidos de las metodologías tradicionales.

Sin embargo, la forma en como se vea la programación orientada a objetos depende de la infraestructura tecnológica informática con la que se cuente; así, una de las evaluaciones más importantes que debe hacer una compañía al adoptar una nueva técnica es ver si el cambio será radical o incremental; una innovación incremental requiere cambios menores, mientras que una radical necesita nuevas bases de ingeniería y de principios científicos, y en consecuencia de entrenar a su personal en la nueva técnica.

El desarrollo orientado a objetos es una nueva forma de pensar basado en la abstracción del mundo real. La escencia de la programación orientada a objetos es la identificación y organización de los conceptos involucrados en la aplicación a desarrollar, es decir, una metodología orientada a objetos es un proceso conceptual independiente del lenguaje de programación, hasta las etapas finales. Y como ya se mencionó, la orientación a objetos es fundamentalmente una nueva forma de pensar y no una nueva técnica de programación.

En el presente trabajo se estudiará la Metodología OMT de James Rumbaugh[1] para el desarrollo orientado a objetos, utilizándose para modelar una aplicación de evaluación de plantas petroquímicas, no se pretende implementar el modulo.

En el primer capítulo se describen los conceptos básicos de la orientación a objetos, para dar los fundamentos necesarios, y así entender no sólo la metodología que aquí se estudia, sino cualquier metodología orientada a objetos.

En el capítulo dos se detallan los conceptos concernientes a la metodología que aquí nos ocupa.

El capítulo tres está dedicado a describir todo el proceso que sigue la metodología de OMT.

El capítulo cuatro contiene el análisis de la aplicación en evaluación de plantas petroquímicas utilizando esta metodología para su desarrollo.

Finalmente, se dan algunas conclusiones a las que se llegó después de hacer un seguimiento de la metodología OMT.

Capítulo 1. Conceptos de la Tecnología Orientada a Objetos

Hoy en día existe una gran demanda de sistemas de alta calidad. Además las aplicaciones tienden a ser más grandes y complejas, esto las hace más difíciles de desarrollar, aunado a la volatilidad de requerimientos y la fuerte competencia en pequeños tiempos de desarrollo provoca que muchos productos se retracen, o peor aún, se liberen sin la calidad adecuada. La importancia de aplicaciones portables entre diferentes equipos y la necesidad de que sean fáciles de aprender y entender por usuarios de diferentes perfiles, hacen las cosas aún más difíciles para los diseñadores, analistas y programadores.

La necesidad de desarrollar y dar mantenimiento de una forma eficiente a sistemas de gran complejidad, en un ambiente competitivo y dinámico, ha dirigido a nuevos caminos el diseño y desarrollo de sistemas surgiendo las Metodologías Orientadas a Objetos.

Las metodologías orientadas a objetos ayudan a simplificar la forma en como se ve el mundo real y a trasladarlo a un esquema automatizado; son una nueva forma de pensamiento acerca de los problemas a resolver, a través de modelos que organizan los conceptos del mundo real, utilizando "objetos" que combinan tanto estructura de datos como comportamiento de la identidad que estén representando, a diferencia de las metodologías convencionales, donde las estructuras de datos y el comportamiento están débilmente conectados.

Se identifican tres actividades tradicionales en el ciclo de vida de la programación orientada a objetos: análisis, diseño e implementación.

En las metodologías orientadas a objetos la separación entre cada fase no está claramente delimitada, es decir, no es fácil determinar donde termina el análisis y empieza el diseño. Esto se debe básicamente a que los puntos de intéres durante las tres fases son los mismos: los objetos.

Los objetos y relaciones entre ellos se identifican tanto en la etapa de análisis como en la etapa de diseño, pero los objetos definidos en el análisis sirven no sólo como entrada del diseño, sino como una capa inicial de éste. Esta continuidad provoca una interfaz casi transparente entre las diferentes etapas, ya que analistas, diseñadores y programadores trabajan con el mismo grupo de entes.

Otra razón de que no estén claramente delimitadas estas fases es que el proceso de desarrollo orientado a objetos es un proceso iterativo y evolutivo.

Esto es realmente una ventaja, ya que desde el análisis hasta la implementación se trabaja con las mismas *clases*, es decir, no cambia la notación sólo se va complementando; claro que no debe perderse de vista que se usan con diferentes propósitos en cada fase y representan diferentes niveles de abstracción.

Antes de profundizar en la descripción de la metodología que se va a estudiar es necesario, para entenderla claramente, conocer los términos que ésta utiliza.

1.1 Conceptos Básicos de la Programación Orientada a Objetos.

Independientemente de la metodología orientada a objetos que se decida utilizar para desarrollar un sistema, los conceptos manejados son los mismos: objetos, clases, clasificación, herencia, polimorfismo y liga dinámica. Los cuatro primeros se presentan a un nivel alto de diseño y análisis, mientras que los dos últimos aparecen a un nivel bajo de diseño e implementación.

Aunque estos conceptos son básicos en la programación orientada a objetos, realmente la herencia es la única contribución de esta técnica de software y lo que aquí hace especial a los otros conceptos es la forma en como se relacionan con la herencia[3].

Los elementos básicos de un sistema orientado a objetos son precisamente los *objetos*, y no son más que algo que tiene sentido dentro de una aplicación, por ejemplo: Saint Exupery, Av. Juárez N° 14, Pedro Paramo, etc. Dicho de otra forma un objeto es un concepto, la abstracción de algo perfectamente delimitado y con significado para el ámbito del problema.

La abstracción sólo debe representar los aspectos escenciales e inherentes de un objeto, ignorando sus propiedades accidentales. En desarrollo de sistemas, esto significa enfocarse en qué es un objeto y qué hace, antes de decidir cómo debería implementarse. El uso de la abstracción preserva la libertad de tomar decisiones de implementación tanto como sea posible, para evitar entrar en detalles en forma prematura.

El representar en forma abstracta objetos durante el análisis, significa tratar únicamente con los conceptos concernientes a la aplicación, sin tomar decisiones de diseño o implementación antes de entender el problema. El uso propio de abstracción asegura mantener el mismo esquema en el análisis, el diseño, la estructura de programas, la estructura de datos y en la documentación.

Los objetos tienen dos utilidades:

a) Facilitar el entendimiento del mundo real.

b) Proporcionar bases prácticas para implementarlos en computadora.

La descomposición de un problema en objetos depende del enfoque y naturaleza del mismo. No existirá una representación correcta, tan solo adecuada.

Todos los objetos tienen identidad y se distinguen perfectamente. El término identidad significa que los objetos se diferencian por su existencia y no por sus propiedades.

Los objetos tienen un estado que está dado por los valores a sus propiedades o atributos en un momento determinado.

En la fase de programación los objetos quedan en memoria con una dirección asociada a un registro. La disposición de bits en la localidad de memoria asignada al objeto determina el estado de éste en un momento dado. Por otro lado, cada objeto se asocia a un conjunto de procedimientos y funciones llamados métodos, los cuales definen las operaciones que puede hacer el objeto. De esta forma un objeto encierra tanto estado como comportamiento.

Dicho de otra forma, la tecnología orientada a objetos forza a especificar qué es un objeto, más que cómo usarlo. Esto porque las características proporcionadas por un objeto son mucho más estables que las formas de como se usa; teniendo que los sistemas construidos sobre estructuras de objetos son más estables. El desarrollo orientado a objetos pone mayor énfasis en la definición de datos y objetos que en la estructura de procedimientos, respecto a las metodologías tradicionales.

Una *clase de objetos* o simplemente *clase* describe un grupo de objetos con las mismas propiedades (atributos), los mismos patrones de comportamiento (operaciones), mismas relaciones con otros objetos y misma semántica. Los objetos serían instancias particulares de las clases.

Un *atributo* es un dato de los objetos en una clase. Nombre, RFC, Dirección y Teléfono son atributos de una clase Persona; Marca, Modelo, Placas, Color son atributos de una clase Automóvil. Cada atributo tiene un valor único para cada instancia y varias instancias pueden tener el mismo valor para un atributo dado. El nombre del atributo es único dentro de una clase, pero varias clases pueden tener un mismo nombre de atributo. Por ejemplo, las clases Persona y Compañía pueden tener el atributo llamado Dirección.

Una operación es una función o transformación que puede aplicarse a objetos de una clase. Contratar, Despedir, Pago de dividendos son operaciones de una clase Compañía. Abrir, Cerrar, Ocultar, Desplegar son operaciones de una clase Ventana. Todos los objetos de una clase comparten las mismas operaciones.

Un método es la implementación de una operación en una clase. Una operación puede tener argumentos además de un objeto destino, dichos argumentos parametrizan la operación sin afectar la selección del método, ya que éste depende exclusivamente de la clase del objeto sobre el que tendrá lugar la operación.

Cuando una operación tiene métodos sobre varias clases es importante que tengan el mismo número y tipos de argumentos, y el mismo tipo de valores en sus resultados.

Cada operación tiene un objeto sobre el que aplicará, como argumento implícito. El comportamiento de las operaciones depende del objetivo de la clase. Una misma operación puede aplicarse a diferentes clases y por tanto puede ser polimórfica, esto significa que una misma operación puede tener un comportamiento diferente de una clase a otra, por ejemplo:

la operación MOVER tiene un comportamiento diferente para la clase VENTANA que para la clase PIEZA DE AJEDREZ. Y aunque existen varios tipos de polimorfismo en general significa la capacidad de tomar más de una forma según el ámbito en que se requiera.

Cuando se llama una operación es necesario considerar cuantas implementaciones existen de ella. El "polimorfismo" decide que implementación se usará de una función dependiendo de la clase. Por ejemplo, un programa orientado a objetos para desplegar información, figuras o texto, simplemente invocará a la operación "dibujar"; la decisión de que procedimiento usar se hace implícitamente por cada objeto que tenga y basándose en su clase. Es necesario repetir la selección del procedimiento cada vez que un programa llame a dicha operación. El mantenimiento es más fácil porque el código llamado no necesita modificarse al agregar una nueva clase.

Concluyendo, a la agrupación de objetos con las mismas características se le llama *clase*, dicho de otra forma las clases son la abstracción de propiedades importantes para la aplicación a desarrollar, ignorando aquellas que no son trascendentes para el caso.

Un objeto se dice ser una ocurrencia o instancia de su clase, cada ocurrencia tiene su propio valor para cada atributo definido en la clase, pero todas comparten los nombres de atributos y las operaciones. Por ejemplo:

Clase	**Atributos**		**Operadores**
EMPLEADO	NOMBRE RFC DIRECCIÓN TELEFONO SUELDO		DESPLEGAR IMPRIMIR BORRAR ANEXAR
Ocurrencia	**Atributos**	**Valores**	**Operadores**
EMPLEADO Trabajador del Sistema de Transporte Colectivo (Metro).	NOMBRE: RFC: DIRECCIÓN: TELEFONO: SUELDO:	José Castro CAJL651116 Lerma 32 727-4115 N$7,500	ANEXAR IMPRIMIR

La mayoría de los objetos de una clase derivan su individualidad de los valores de sus atributos y relaciones con otros objetos, aunque es posible encontrar objetos con atributos y relaciones idénticas. Los objetos en una clase comparten un propósito semántico común, esto es tienen la misma finalidad.

Si los objetos son el punto central de la modelación de objetos, ¿Por qué molestarse con clases?. Mediante la agrupación de objetos en clases, se obtiene la abstracción de un problema y la abstracción dá al modelaje su poder y capacidad de generalizar a partir de unos cuantos casos específicos. Además el uso de clase proporciona la ventaja de almacenar operaciones y nombres de atributos una vez por clase en lugar de una vez por instancia.

Desde el punto de vista de lenguajes de programación, una clase es la construcción para implementar un ente definido por el usuario. Idealmente una clase es la implementación de un Tipo Abstracto de Datos.

A la acción de agrupar objetos con la misma estructura de datos (atributos) y el mismo comportamiento (operaciones) en una misma clase se le conoce como *clasificación*. Por ejemplo: en la clase POLIGONO se clasifican los objetos triángulo, rectángulo, cuadrado, entre otros.

Las clases no actúan aisladamente dentro de un sistema sino que están relacionadas entre ellas. Cuando la relación entre clases permite definir e implementar una clase (subclase) en base a otra (superclase), la subclase *hereda* todas las características de la superclase adicionando además las propias, a las nuevas características (las de la subclase) se le conoce como extensiones.

Por ejemplo:

Superclase	**Atributos**	**Operadores**
EMPLEADO	NOMBRE RFC DIRECCIÓN TELEFONO SUELDO	DESPLEGAR IMPRIMIR BORRAR ANEXAR
Subclase	**Atributos**	**Operadores**
PILOTO	FORMACION (Comercial, Militar, etc) HORAS DE VUELO ESPECIALIDAD	PROGRAMAR ESPECIALIZAR

En este ejemplo se puede apreciar que PILOTO es un tipo de EMPLEADO, por lo que contiene los atributos y operadores de éste (Nombre, RFC, etc), además de contar con los propios(Formación, Programar, etc.) los cuales serían una extensión a las características de EMPLEADO.

Esto es, cuando un grupo de clases tienen algunos atributos en común, es factible definir una nueva clase que contenga esos atributos comunes para eliminarlos en la definición de las clases iniciales, evitando así redundancia tanto en las definiciones como en la programación al momento de la implementación. Por ejemplo, para las clases CIRCULO, POLIGONO, LINEA se puede hacer una generalización a través de la clase FIGURA GEOMETRICA. La clase que contiene los atributos comunes sería la *superclase* y las específicas las *subclases*.

La generalización es una relación entre la superclase y las subclases a menudo llamada "*es un*", porque cada instancia de una subclase "es una" instancia de la superclase, donde cada

subclase *hereda* las características de la(s) superclase(s). Como en el ejemplo anterior PILOTO "es un" EMPLEADO.

Relacionando así a las clases se puede definir una estructura jerárquica entre ellas, surgiendo con esto los términos de *padre* e *hijo*; donde padres serán las superclases e hijos las subclases.

El hecho de que se pueda establecer una relación de tipo jerárquica entre clases mediante la herencia, implica que ésta sea una propiedad transitiva, teniendo que si la clase A hereda las características de la clase B y B a su vez hereda las de la clase C, entonces A hereda las de C.

A las clases que no tienen instancias directas sino que las obtienen a través de sus hijos, se les llama *clases abstractas* y a las que sí tienen instancias directas se les llama *clases concretas*.

Las clases abstractas normalmente sirven para organizar características comunes de varias clases y son frecuentemente usadas para definir atributos y métodos que heredarán las subclases. Por otro lado, una clase abstracta puede definir una operación sin proporcionar el método correspondiente, surgiendo con esto *operaciones abstractas*.

Una operación abstracta define la forma de una operación y cada subclase concreta indica su propia implementación. Una clase concreta no puede definir operaciones abstractas.

A pesar de que una subclase hereda todas las características de sus superclases puede también condicionar atributos de sus padres; por ejemplo, un círculo es una elipse cuyos ejes mayor y menor son iguales. Los cambios arbitrarios a los valores de un atributo de una clase restringida pueden provocar una violación a sus límites, de tal forma que el resultado no pertenezca a la subclase. Además, las características heredadas pueden renombrarse en una restricción. Los ejes mayor y menor heredados de un círculo deben ser iguales y podrían renombrarse como *diámetro*.

Una clase puede heredar características de más de una superclase, donde una característica del mismo padre encontrada a lo largo de diferentes rutas se hereda sólo una vez. Una clase puede heredar de una superclase, a este hecho se le conoce como *Herencia Múltiple*.

La herencia múltiple permite a una clase tener más de una superclase y heredar características de todos sus ancestros. Esto permite mezclar información de dos o más fuentes dando como resultado una generalización más complicada que la herencia sencilla, la cual restringe la herencia de clases a un árbol.

La ventaja de la herencia múltiple es el gran poder en la especificación de clases y una mayor oportunidad para el reuso, esto trae como consecuencia que la representación de objetos se acerque más a la forma de pensar de la gente. La desventaja es una pérdida de simplicidad tanto en la concepción como en la implementación.

La generalización y herencia se puede llevar a los niveles de abstracción que la aplicación y/o el grupo desarrollador lo requieran. Como en el ejemplo del EMPLEADO y el PILOTO se podría definir la clase PERSONA que sería un ancestro de EMPLEADO y por tanto de más alto nivel.

Los términos de herencia, generalización y especialización conducen a la misma idea y con frecuencia son usados indistintamente. Sin embargo, en este trabajo generalización se refiere a la relación entre clases, mientras que herencia se refiere al mecanismo de compartir atributos y operaciones usando la generalización.

Generalización y Especialización son enfoques diferentes de la misma relación, ya que una superclase generaliza a las subclases y las subclases son una especialización de una superclase.

En algunos casos una subclase puede incluir una característica ya definida en la superclase, por supuesto debe ser bajo el mismo nombre, con el objeto de puntualizar que el comportamiento de dicha característica depende de la subclase o bien para mejorar su ejecución. Sólo debe cuidarse que la característica "redefinida" no haga inconsistente el concepto de herencia.

Como ya se dijo, a través de la herencia se pueden compartir estructuras entre varias subclases, estructuras tanto de datos como de comportamiento, evitando así redundancia, esto se traduce en compartir código, lo cual es uno de las principales ventajas de los lenguajes orientados a objetos dado que herencia es el concepto más prometedor para construir sistemas a partir de objetos ya existentes listos para reusarse, por esta razón la programación orientada a objetos con el uso de la herencia permite obtener prototipos de una manera rápida y sencilla.

La herencia no sólo soporta el reuso en la creación de un sistema, sino también facilita su expansión, ya que al integrar una nueva clase se buscaría cuales deberían ser superclases de acuerdo a sus características y a su objetivo dentro del sistema, quedando sólo por programar lo referente a los atributos y operadores que la diferencia de las otras.

Más importante que ahorrarse líneas de código es la claridad conceptual de reconocer que diferentes operaciones son la misma cosa. Esto reduce el número de casos distintos que deben entenderse y analizarse. La programación orientada a objetos no sólo permite compartir información dentro de una aplicación, sino ofrece la perspectiva de reusar diseños y código del proyecto actual en desarrollos futuros. Aunque esta posibilidad ha sido sobre-enfatizada como una justificación de la tecnología de la orientación a objetos, lo cierto es que el desarrollo bajo esta técnica proporciona herramientas, tales como la abstracción, encapsulación y herencia para construir bibliotecas de componentes reusables.

La encapsulación es una mecanismo que permite separar los aspectos externos de un objeto, los cuales pueden ser accesibles para algunos objetos desde la implementación, pero pueden estar ocultos para otros. La encapsulación previene que pequeños cambios en los programas repercutan en forma masiva. La implementación de un objeto puede cambiarse sin afectar a

las aplicaciones que lo usen, estos cambios pueden ser para mejorar, para corregir un error, etc. La encapsulación no es característica de lenguajes orientados a objetos, pero la capacidad de combinar estructura de datos y comportamiento en una entidad, hace a la encapsulación más limpia y poderosa que en los lenguajes convencionales.

La orientación a objetos no es una fórmula mágica para asegurar el reuso de rutinas, ya que el reuso no es algo que pase, sino tiene que planearse, pensando más allá de la aplicación actual y hacer un esfuerzo extra para llegar a un diseño general.

Identidad, clasificación, polimorfismo y herencia caracterizan a los lenguajes orientados a objetos. Cada uno de estos conceptos pueden usarse aisladamente, pero no tendrán el mismo valor si se utilizan conjuntamente, por esta razón los beneficios de un enfoque orientado a objetos es mayor que los que tienen cada uno de estos conceptos por separado.

De acuerdo a Thomas estos conceptos vienen, en conjunto, a crear un estilo diferente de programación, dado que hacen un mayor énfasis sobre las propiedades esenciales de un objeto para forzar al desarrollador a pensar más cuidadosa y profundamente en lo que es un objeto y lo que hace, obteniendo como resultado un sistema más limpio, más general y más robusto que si se hubiera enfatizado sólo en el uso de datos y operaciones.

1.2 Conceptos Básico de Diseño.

Se ha observado que la mayoría de los especialistas en computación conocen diferentes lenguajes de programación, varias técnicas de diagramas para representar detalles de diseño, pero no conocen más que una metodología para diseño de sistemas. Esto se debe a que es más complicado aprender una nueva técnica de desarrollo de sistemas que aprender un lenguaje de programacón, esto porque un lenguaje requiere sólo de memorizar las palabras clave que lo conforman y su sintáxis, los conceptos no cambian; mientras que una nueva técnica de desarrollo de sistemas necesita un cambio fundamental en la forma de pensar.

Un modelo de diseño se caracteriza por el proceso de descomposición del problema para llegar a la solución. Así, el problema se desglosa en una serie de actividades a realizar en un orden determinado. Dichas actividades forman bloques básicos del procedimiento de la aplicación.

Los modelos orientados a objetos enfocan el desarrollo de sistemas a la identificación de identidades involucradas en el problema a resolver. La abstracción de estas entidades se convertirá en clases en las que se basará el sistema.

No debe perderse de vista que en la programación orientada a objetos la información obtenida en el proceso del análisis pasa a ser parte integral del diseño, pero representada con una terminología diferente y con otra perspectiva. Esta transición casi transparente se debe a la homogeneidad de las entidades, descripciones y clases usadas en cada fase.

Las fases de análisis y diseño orientadas a objetos trabajan muy de cerca debido a la representación de objetos. Así, el analista identifica los objetos, mientras que el diseñador especifica objetos adicionales necesarios para una solución.

El diseño orientado a objetos tiene dos componentes que deben combinarse en el desarrollo de una aplicación: el diseño de clases y el diseño de la aplicación.

El diseño de clases se encuentra distribuido dentro del de la aplicación. El diseño de la aplicación incluye la identificación de los tipos de entidades involucradas en el problema, además de especificar lo necesario para implementar la solución. Cada tipo de entidad conduce a la descripción de una clase, la cual será la representación de un concepto. Una vez que esa descripción se terminó, la aplicación puede ser diseñada. La aplicación se desarrolla a través de la conexión de ocurrencias, representando con esto al mundo real.

La descripción de clases incluye tres partes: definición de atributos, descripción de la interfaz de la clase y el conjunto de transiciones válidas entre los posibles estados de una ocurrencia de la clase.

El modelo orientado a objetos proporciona un soporte natural para descomponer un sistema en módulos a través del agrupamiento de clases que estén conceptualmente relacionadas. Este agrupamiento se representa mediante *Clusters*, *Subsistemas* o *Frameworks*, dependiendo de la metodología usada. De aquí que las clases individualmente no son tan útiles como cuando se manejan en conjunto.

El usuario de la aplicación no conocerá todos los atributos y operadores de una clase. Por ejemplo: El usuario tiene acceso a EDAD; sin embargo, es transparente para él que se obtuvo de FECHA DE NACIMIENTO y una serie de operadores para calcularla.

El diseño orientado a objetos facilita la expansión de una aplicación a través de los mecanismos de herencia. La relación de herencia fomenta el reuso de definiciones existentes haciendo más sencillo el desarrollo de nuevas definiciones. Como la estructura de herencia se va haciendo más grande progresivamente, la cantidad de especificaciones e implementaciones de herencia crecen en la definición de una nueva clase. Esto significa que en la medida que la estructura de la herencia crece el esfuerzo para implementar nuevas clases disminuye.

Los modelos orientados a objetos combinan técnicas de diseño con características de lenguajes de programación para proporcionar un mayor apoyo en el uso de rutinas predefinidas por el diseñador, así cada vez que se crea una ocurrencia se tiene que reusar el código generado para la clase correspondiente. Como parte de la fase de diseño de alto nivel, la herencia sirve como un medio de relaciones de generalización/especialización.

Estos conceptos altamente relacionados proporcionan mucho del poder del modelo orientado a objetos. La Especialización guía al diseñador en el reuso de una abstracción existente para la definición de una nueva clase que es más específica que las clases ya creadas. La Generalización soporta la explotación de aspectos comunes entre clases.

Cuando dos clases están representandas por conjuntos de atributos que se intersectan, los atributos comunes pueden sacarse creando un tercer conjunto con ellos, el cual representará una nueva clase de más alto nivel de abstracción, es decir, será la superclase de las dos iniciales.

El código de las subrutinas de clases de una aplicación se almacena en bibliotecas, de tal forma que quedan disponibles para usarse *no sólo en la aplicación actual*, sino también en las futuras. Las bibliotecas conforman lo que se conoce como Programas Base.

Capítulo 2. Modelos de la Metodología OMT.

Por miles de años se han hecho representaciones esquemáticas de diferentes aspectos del mundo real, ya sean físicos como un edificio, un avión, etc. o bien abstractos como la trayectoria de un planeta, etc., con la finalidad de resolver un problema determinado. A dichas representaciones se les ha dado el nombre de "modelos", muchas de las actividades humanas se apoyan en ellos para obtener mejores resultados.

Las representaciones esquemáticas o modelos pueden tomar diferentes formas y dimensiones, dependiendo de la escencia del problema. Así, por ejemplo: un carpintero realiza un dibujo marcando características y proporciones del mueble que va a construir; un arquitecto hace maquetas y diferentes tipos de planos (uno para cada ámbito: eléctrico, hidráulico, etc., donde cada uno es complemento de los otros), especificando cantidades, proporciones, tipos de material entre otras cosas, de los edificios antes de construirlos.

En un modelo se omiten los detalles innecesarios, enfatizando aquellos puntos que son importantes para la resolución o entendimiento del problema al que se enfrente. La creación de modelos dá al diseñador varias ventajas y mientras más complejo sea el problema más útil y necesario se vuelve el uso de modelos, incluso como en el caso del arquitecto pueden tenerse varios, donde cada uno se enfoca a detallar un punto del problema complementándose entre ellos.

Apoyarse en el uso de modelos para resolver un problema tiene varios propósitos:

- **Probar una entidad física antes de construirla**. Crear un modelo para probar una entidad antes de construirla permite al diseñador hacer cambios pertinentes antes de llevar a cabo la implementación, además se puede obtener información muy difícil o muy cara de conseguir o de manipular trabajando en el mundo real, como en el caso de un lanzamiento a la luna. Todo esto generalmente invirtiendo menos tiempo y dinero.

- **Comunicación con clientes o usuarios.** Partiendo de que un modelo imita todo o parte del comportamiento real de una entidad, el cliente puede conocer hasta donde cubrirá sus necesidades, o si es o no lo que requiere.

- **Visualización**. Al simular la realidad con un modelo, se pueden anticipar los resultados, dando al diseñador o grupo de trabajo la capacidad de corregir posibles desviaciones sobre el modelo y poder checar las repercuciones que dichos cambios tendrán, todo esto antes de llevarlo a la práctica.

- **Reducción de complejidad.** Tal vez sea la principal ventaja, ya que los sistemas muy complejos son difíciles de entender, y a través de modelos se puede particionar,

reduciendo su complejidad trabajando con un pequeño número de puntos importantes a la vez. Esta ventaja, de alguna forma encierra a las anteriores.

El desarrollo de sistemas es una actividad que también se apoya en la construcción de modelos para la optimización de resultados. En este ámbito la apariencia de los modelos depende de la herramienta de programación selecionada, debido a que cada metodología cuenta con una notación propia.

Para la representación de modelos la Metodología OMT de James Rumbaugh et al [1] se basa en la "Object Modeling Technique" , la cual maneja las fases de análisis, de diseño y la de implementación. A su vez la fase de análisis se conforma de tres modelos, donde cada uno representa un aspecto específico del sistema de tal forma que cada modelo proporciona un punto de vista diferente, dando en conjunto una descripción completa del sistema.

Los tres modelos que conforman esta técnica son:

- **Modelo de objetos.** Representa la parte estática del sistema, es decir describe la estructura de los objetos de un sistema, su identidad, sus relaciones con otros objetos, sus atributos y sus operaciones. Proporciona el marco de trabajo para los modelos dinámico y funcional. La meta del modelo de objetos es representar los aspectos importantes de la aplicación. Este modelo se esquematiza gráficamente con diagramas que contienen clases, organizándolas en estructuras jerárquicas, fomentando la comunicación con el usuario y ayudando a documentar la estructura del sistema.

- **Modelo dinámico.** Representa lo temporal, es decir describe los aspectos del sistema referentes al tiempo y secuencia de operaciones, a través de la esquematización del control de ejecución; esta parte del sistema describe la secuencia en que ocurren las operaciones, sin importar lo que hagan, sobre que operen, o como se implementen. Este modelo también se representa gráficamente, a través de diagramas de estado, los cuales muestran la secuencia de estados y eventos permitidos en el sistema para una clase de objetos. Los diagramas de estado también hacen referencia a los otros modelos.

- **Modelo funcional.** Representa lo que cambia, es decir describe los aspectos del sistema referentes a la transformación de valores, funciones, restricciones, etc., enfocándose a lo que hace el sistema, sin importar cómo o cuándo lo hace. El modelo funcional se representa mediante diagramas de flujo de datos, donde se esquematizan las dependencias entre valores y cálculos de los datos de salida a partir de los de entrada y funciones.

Existe una relación muy íntima entre los tres modelos, de tal forma que cada uno hace referencia a entidades de los otros dos; el modelo de objetos describe la estructura de datos sobre los que operan los modelos dinámico y funcional. El modelo dinámico describe el control de la estructura de objetos mostrando decisiones que dependen de los valores de los objetos y provocando acciones que los modifiquen y que invoquen funciones. El modelo funcional describe funciones llamadas por operaciones del modelo de objetos y acciones del

dinámico. Las funciones operan sobre valores de los datos especificados en el modelo de objetos. El modelo funcional también muestra restricciones sobre valores.

Cada modelo va de la mano con el ciclo de vida del desarrollo de sistemas. En la fase del análisis se obtiene un modelo de la aplicación; en la de diseño se adiciona al modelo lo referente a la solución; y en la implementación se codifican tanto la aplicación como su solución. Con esto, la palabra "modelo" toma dimensiones y en todos se usan los mismos modelos con diferentes grados de abstracción.

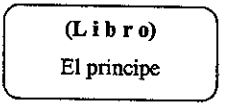
2.1 Modelo de Objetos.

Los modelos de objetos enfatizan la construcción del sistema a través de objetos en lugar de considerar la funcionalidad, ya que esto los hace menos suceptibles a cambios. Los modelos de objetos proporcionan una representación gráfica muy intuitiva del sistema utilizando lo que se conoce como Diagramas de Objetos.

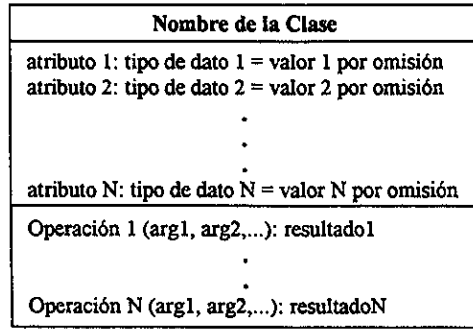
Los diagramas de objetos son una notación gráfica formal para representar clases, objetos y sus relaciones entre ellos de una manera coherente, precisa, fácil de formular, fácil de entender y práctica. Estos diagramas son útiles tanto para representar modelos abstractos como para diseño de programas. Existen dos tipos de diagramas de objetos: Diagramas de clases y diagramas de instancias.

Los Diagramas de Clase describen el caso general del modelo de un sistema, dicho de otra manera es un esquema, un patrón o una plantilla para describir muchas posibles instancias de datos, de tal forma que esquematiza una clase de objetos. Un diagrama de clase corresponde a un conjunto infinito de diagramas de instancias. El símbolo de la notación en OMT es una caja con el nombre de la clase dentro en letras negritas.

Los Diagramas de instancias describen instancias de objetos y sirven para documentar casos de prueba o discutir ejemplos con la finalidad de hacer claro algún(os) diagrama(s) de clase. El símbolo en OMT para representarlos es una caja de puntas redondeadas, dentro se especifica el nombre de la clase entre paréntesis y en negritas, abajo se indica el nombre del objeto en letra normal.



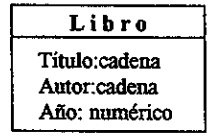
En general, los diagramas de clase se conforman por una caja dividida en tres secciones, donde la primera se usa para el nombre de la clase, la segunda para los atributos, sus tipos de datos y valores por omisión, la tercera está destinada para indicar las operaciones de la clase, mostrando argumentos y tipo de resultado.



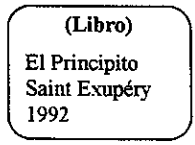
Los atributos se listan en la segunda parte de la caja de la clase, cada atributo puede complementarse por detalles opcionales tales como el tipo y el valor por omisión. El tipo va precedido por dos puntos (:) y el valor por omisión por el signo igual(=).

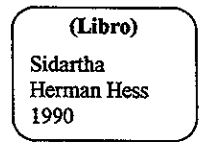
No siempre se desplegarán los atributos en el diagrama de la clase, esto dependerá del nivel de detalle que se requiera en el modelo de objetos.

A continuación se muestra un diagrama que representa la clase Libro y sus atributos.



Los siguientes diagramas representan instancias de la clase Libro, con los valores respectivos para cada atributo.





Las operaciones aparecen en la tercera sección al final de la caja de la clase separados por otra línea. Cada operación puede contener información adicional, tales como la lista de argumentos y el tipo de resultado, dependiendo del grado de detalle que se requiera. La lista de argumentos se pone inmediatamente después del nombre de la operación, entre paréntesis y los argumentos separados por comas, puede especificarse el nombre y el tipo; paréntesis encontrados indica que la operación no tiene argumentos. El tipo de resultado va precedido

por dos puntos y se recomienda no omitirlo dado que es importante distinguir las operaciones que regresan valores de las que no. Las operaciones pueden omitirse en diagramas de alto nivel.

Libro
Título: cadena Autor: cadena Año: numérico
Anexa Presta(usuario:cadena)

Figura Geométrica
Color: cadena=negro Posición:vector
Mover(delta:vector) Seleccionar(p:apuntador):booleano Rotar(ángulo:entero)

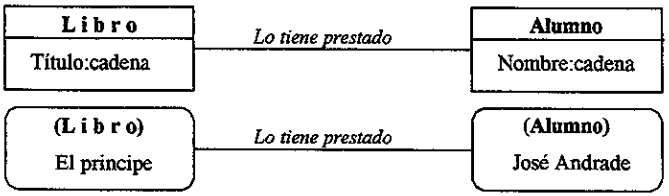
Cada objeto o clase no actúa separadamente, las ligas y asociaciones son los medios para establecer relaciones entre ellos respectivamente.

Una *liga* es una conección entre instancias, puede ser física o conceptual. Por ejemplo Roberto Martínez TRABAJA PARA Celanese. Una liga es la instancia de una asociación.

Una *asociación* describe un grupo de ligas con la misma estructura y semántica. Por ejemplo, una persona TRABAJA PARA una compañía. Todas las ligas en una asociación conectan objetos de la misma clase. Las asociaciones y ligas aparecen como verbos en las especificaciones de un problema. Una asociación describe un grupo potencial de ligas de la misma manera que una clase representa un grupo potencial de objetos.

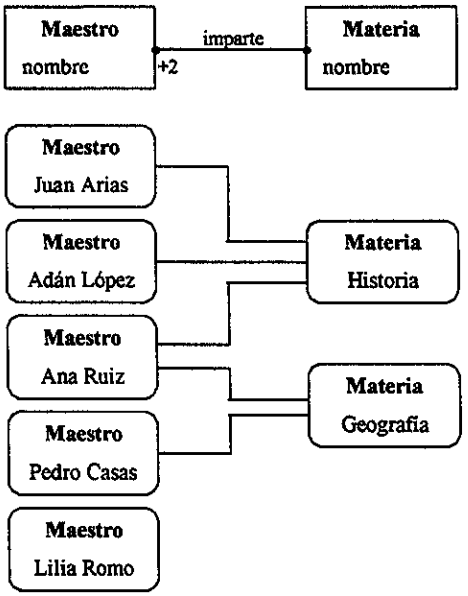
Las asociaciones son inherentemente bidireccionales. Una asociación binaria usualmente va en una dirección, aunque puedan ir en ambas. La dirección implícita de una asociación es *directa* a la otra dirección se le llama *inversa*. Así, la asociación TRABAJA PARA va en dirección *directa* de persona a compañia y la asociación EMPLEA A podría ser su *inversa* pues va de compañía a persona.

En la notación OMT las asociaciones y ligas se representan mediante una línea entre las clases y objetos relacionadas, respectivamente. El nombre de la asociación se pone sobre la línea con letras itálicas, puede omitirse si un par de clases tienen una sola asociación cuyo significado es obvio. Es recomendable acomodar las clases de tal forma que la dirección de las asociaciones vayan de izquierda a derecha.



El número de instancias de una clase que pueden estar relacionadas con una de la otra clase determina la *multiplicidad* de la asociación.

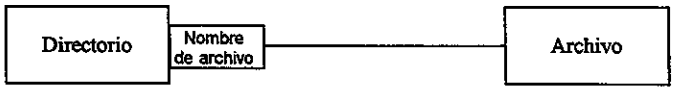
Lo más importante de la multiplicidad es saber si es de tipo "uno" o de "muchos". En notación OMT, el de tipo "uno" no se especifica, pero para el de "muchos", se escribe el valor de la multiplicidad y en los extremos de la línea se pone un punto relleno para indicar una multiplicidad de cero o más, y un punto hueco para indicar cero o uno.



El valor de la multiplicidad puede ser un número "3" (exactamente tres), una lista de números "4,6,15" (cuatro, seis y quince) o un rango "2+" (dos ó más) ó "3-5" (de tres a cinco).

La multiplicidad depende del objetivo del sistema y no es algo que se deba definir en las primeras etapas del desarrollo, primero se definirán objetos, clases y asociaciones, lo cual ayudará a aclarar puntos que facilitarán a decidir acerca de la multiplicidad.

En algunas ocasiones es posible reducir la multiplicidad de una asociación *calificando* los objetos para identificarlos dentro de la relación y así por ejemplo una asociación "uno a muchos" puede convertirse en una "uno a uno". Dentro de un diagrama de objetos el *calificador* se dibuja como una pequeña caja pegada a la clase que identificará a los objetos.

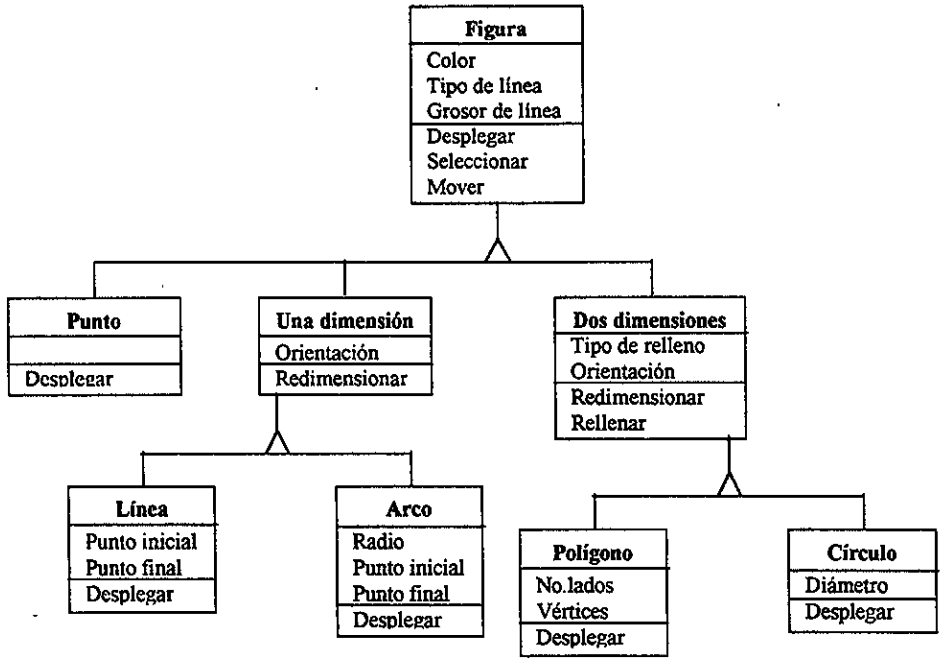


Un directorio tiene muchos archivos, pero un directorio más el nombre del archivo permiten identificar a un archivo por lo que el nombre del archivo será el identificador y de está manera se reduce la multiplicidad de la asociación, ya que un directorio tiene muchos archivos con un nombre único cada uno. El calificar objetos permite ser más preciso.

La generalización o herencia también deben plasmarse sobre los diagramas de objetos dado que son abstracciones poderosas para aprovechar las semejanzas existentes entre clases mientras preservan sus diferencias.

Dentro de los diagramas de objetos la generalización se denota mediante un triángulo conectando una superclase a sus subclases, generando una estructura jerárquica o de árbol.

En el siguiente ejemplo, los atributos Color, Tipo de línea y Grosor de línea los heredan todas las subclases, Orientación es de las clases de una y dos dimensiones y Tipo de relleno sólo de la clase de Dos dimensiones.



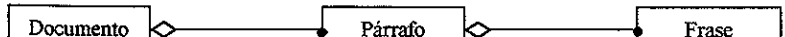
Se recomienda no crear demasiados niveles de generalización, ya que se pueden volver difíciles de entender y programar; sin embargo, el hecho de que una estructura jerárquica tenga o no muchos niveles dependerá del enfoque del problema a resolver.

La generalización es una construcción útil tanto para la esquematización conceptual como para la implementación del sistema a desarrollar, ya que facilita la creación del modelo con la estructura de clases, y la implementación con la herencia de operaciones dado que las subclases reusarán las subrrutinas hechas para sus padres.

Esto último es un punto muy importante dado que elimina la redundancia en programación (repetir las mismas líneas de código para una grupo de clase), facilitando la creación de bibliotecas de subrutinas para su uso posterior, no sólo en el sistema en curso sino para desarrollos futuros.

Para ensamblar el modelo completo se tiene que ir agregando cada elemento para formar módulos y a su vez estos se ensamblarán para llegar a la representación del sistema.

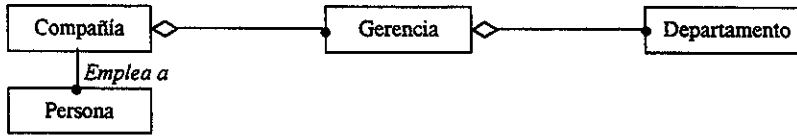
Cada *agregación* es una especie de asociación que se dibuja en los diagramas de objetos exactamente igual sólo que con un pequeño diamante pegado a la clase a la que se ensamblarán los objetos de la clase al otro extremo.



Cada objeto que se va adicionando se convierte en un componente de la agregación, conformando así un objeto lógico compuesto por varios objetos físicos.

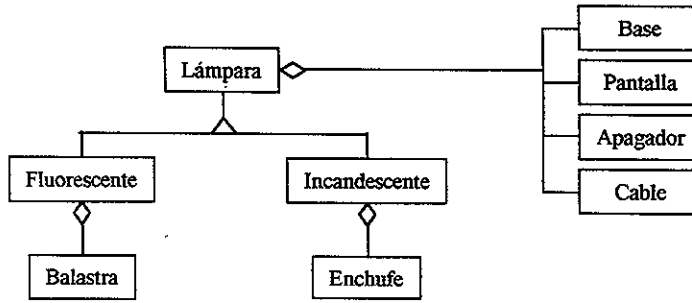
Cuando el diseñador tiene poca experiencia en esta metodología, es factible que pueda confundir una agregación con una asociación y aún más con generalización. Sin embargo, cabe notar que una agregación adiciona connotaciones lógicas. Así, si dos objetos están firmemente ligados mediante una relación de componentes está es una agregación; y si los objetos usualmente se consideran como independientes aunque pueden estar ligados, entonces se trata de una asociación.

Por ejemplo, una compañía puede ser una agregación de varias gerencias las cuales a su vez son agregaciones de departamentos; una compañía es indirectamente una agregación de sus empleados, partiendo de que COMPAÑÍA y PERSONA son clases independientes.



A las agregaciones con frecuencia se les conoce como la relación "es parte de", y a la generalización por la relación "es tipo de" o "es un".

En el siguiente diagrama se ilustra la agregación y generalización para el caso de una lámpara de mesa, cuyos componentes son una base, una pantalla, un apagador y un cable; pero pueden clasificarse en diferentes subclases: fluorescentes e incandescentes por ejemplo. Cada subclase puede tener partes específicas. Por ejemplo, una lámpara fluorescente tiene una balastra y las incandescentes enchufes.



2.2 Modelo Dinámico.

Un sistema puede entenderse mejor examinando primero su estructura estática, esto es, la estructura de sus objetos y las relaciones entre ellos, para después analizar los cambios de los objetos y sus relaciones a través del tiempo. Esos aspectos de un sistema concernientes

con el control y los cambios a través del tiempo, se esquematizan en el modelo dinámico reflejando el flujo de transformaciones en los objetos y sus relaciones. El modelo dinámico sólo describe las secuencias de operaciones que ocurren en respuesta a estímulos externos, sin importar lo qué hacen, sobre qué se efectúan o cómo se realizan.

Un modelo de objetos describe objetos, junto con sus atributos y asociaciones. Al pasar el tiempo los objetos se estimulan entre si, provocando una serie de cambios en los valores de sus atributos y ligas. Los valores de los atributos y ligas de un objeto se les conoce como *estados* y al estímulo de un objeto sobre otro se le llama *evento.*

Un evento es algo que sucede en un momento, tal como que el usuario presione el botón izquierdo del mouse o el vuelo 203 salga de Acapulco.

Un evento puede preceder o seguir a otro, pudiendo estar relacionados o ser totalmente independientes. Por ejemplo, el vuelo 203 sale de Acapulco antes de que el vuelo 203 llegue a la Ciudad de México, están relacionados; mientras que el vuelo 203 sale de Acapulco y el vuelo 394 sale de Oaxaca son eventos en los que no existe relación alguna.

A los eventos que no están relacionados se les llama *concurrentes*, aún si las localizaciones físicas de dos eventos no son distantes se consideran concurrentes si no se afectan entre sí. El orden en el que sucedan una serie de eventos concurrentes es indistinto dado que son totalmente independientes uno del otro.

Un evento es la transmisión de información desde un objeto a otro, donde cada evento es una ocurrencia única, y al igual que los objetos también se agrupan en clases sólo que de eventos, las cuales tienen un nombre para indicar comportamiento común; por ejemplo, el vuelo 203 sale de Acapulco y el vuelo 394 sale de Oaxaca son instancias de la clase de eventos SALIDAS DE VUELOS AEREOS, y nuevamente, al igual que las clases de objetos crean una estructura jerárquica.

La respuesta a un evento depende del estado del objeto receptor, y puede incluir un cambio de estado o el envio de otro evento al transmisor original o a un tercer objeto. El esquema de eventos, estados y transiciones de estados para una clase dada puede ser abstracto y representado gráficamente en un *diagrama de estado*.

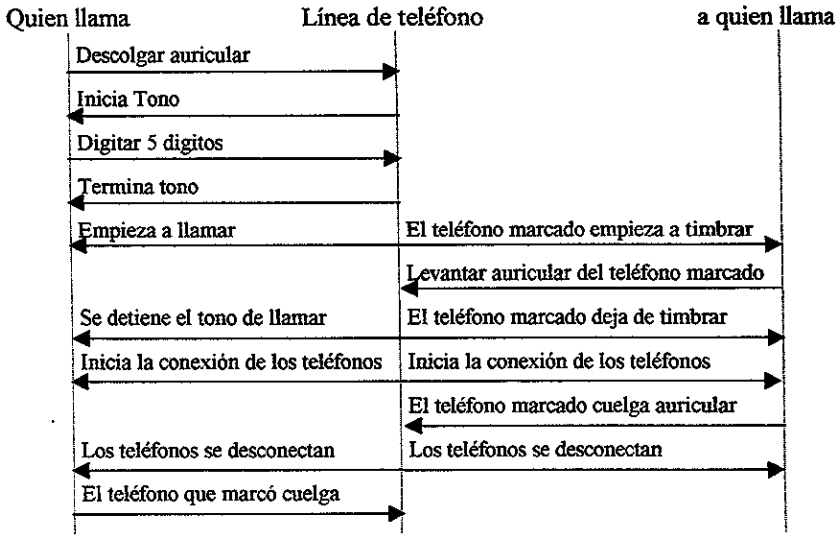
Un diagrama de estado es una red de estados y eventos de la misma forma que un diagrama de objetos es una red de clases y relaciones. El modelo dinámico consta de múltiples diagramas de estado, uno para cada clase con un comportamiento dinámico importante y mostrando un patrón de actividad para el sistema en cuestión.

Durante una ejecución particular del sistema en cuestión ocurre una secuencia de eventos, a la cual se conoce como *escenario*. El alcance de un escenario puede variar; puede incluir todos los eventos en el sistema o incluir sólo aquellos que caen dentro o que se generan por ciertos objetos en el sistema. El siguiente paso después de escribir un escenario es identificar al objeto transmisor y al receptor de cada evento.

La siguiente tabla muestra un escenario para usar una línea de teléfono:

Descolgar auricular
Iniciar Tono
Digitar primer digito
Termina tono
Digitar el resto de los digitos
Empieza a llamar
El teléfono marcado empieza a timbrar
Levantar auricular del teléfono marcado
El teléfono marcado deja de timbrar
Se detiene el tono de llamar
Inicia la conexión de los teléfonos
El teléfono marcado cuelga auricular
Los teléfonos se desconectan
El teléfono que marcó cuelga

En OMT los escenarios se representan en *Diagramas de Eventos*, donde cada objeto es una línea vertical y cada evento una flecha horizontal del objeto transmisor al receptor. El tiempo aumenta de arriba hacia abajo mostrando las secuencias de eventos, el espacio entre los renglones es irrelevante.



La respuesta de un objeto a un evento puede incluir una acción o un cambio en los valores de los atributos y/o ligas, es decir en su estado. Por ejemplo, si se marca un dígito en el estado *tono*, la línea telefónica cambia al estado *llamada*; si el receptor es reemplazado en el estado *tono*, la línea telefónica ya no da línea y entra al estado *inactivo*.

Un estado permanece el tiempo que hay entre dos eventos recibidos por un objeto y como los eventos representan puntos en el tiempo, los estados representan intervalos de tiempo. Por ejemplo, desde que se levanta el auricular y hasta antes de marcar el primer dígito la línea telefónica está en estado de *tono*. El estado de un objeto depende de las secuencias anteriores de eventos que se hayan recibido, aunque en la mayoría de los casos los efectos de los eventos pasados son cancelados por eventos subsecuentes. Por ejemplo, los eventos que pasaron antes de que el teléfono haya sido colgado no tienen efecto en el comportamiento futuro; el estado *inactivo* "olvida" eventos recibidos antes del evento *colgar*.

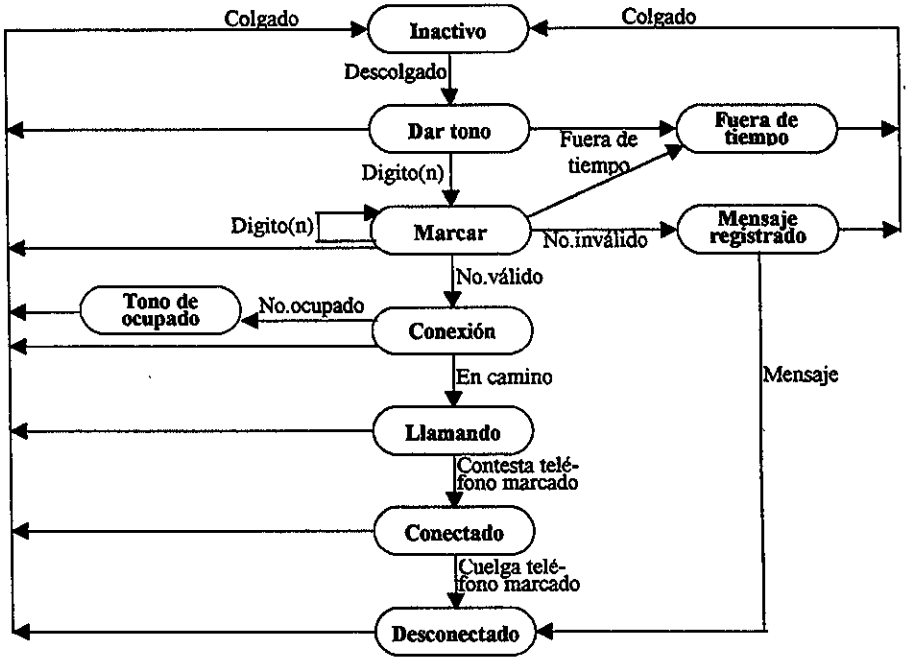
Cuando se recibe un evento, el siguiente estado depende tanto del evento como del estado actual; un cambio de estado causado por un evento se llama una *transición*.

Para representar gráficamente a los estados y transiciones la OMT utiliza *Diagramas de Estados*, los cuales relaciona eventos y estados. Un diagrama de estado es una gráfica cuyos nodos son estados y las líneas que los unen son las transiciones etiquetadas con el nombre del evento correspondiente. Un estado se dibuja como una caja ovalada que opcionalmente puede contener un nombre; una transición se representa con una flecha del estado transmisor al estado receptor; la etiqueta sobre la flecha es el nombre del estado que causa la transición. Los diagramas de estado reflejan la secuencia de estados provocados por una secuencia de eventos.

Los diagramas de estado pueden representar ciclos de vida finitos o ciclos continuos. El diagrama para la línea telefónica es un ciclo continuo, ya que el uso del teléfono es un servicio continuo; pero si se pretendiera instalar nuevas líneas, el estado inicial sería importante y en cuanto quedaran instaladas terminaría su ciclo de vida. Un diagrama finito tiene un estado inicial y uno final; el inicial entra en la creación de un objeto y la entrada del estado final indica la destrucción del objeto.

En los diagramas de estado, un estado inicial se representa por un círculo sólido, el cual puede estar etiquetado para indicar diferentes condiciones iniciales; un estado final se representa por un círculo con un punto al centro, el cual también puede estar etiquetado para distinguir diferentes condiciones finales.

A continuación se describe en un diagrama de estado el comportamiento de una línea telefónica. El diagrama está diseñado para una línea de teléfono y no para el que llama o al que le llaman.



Nótese que los estados no definen totalmente todos los valores del objeto; por ejemplo, el estado *Marcar* incluye todas las secuencias de números de teléfono incompletos. No es necesario considerar a los diferentes números como estados separados, ya que ellos tienen el mismo comportamiento; sin embargo, el número actual marcado debe guardarse como un atributo.

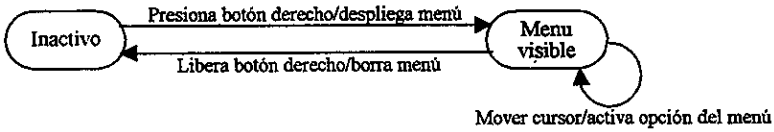
El modelo dinámico es una colección de diagramas de estado que interactúan entre sí compartiendo eventos. Un modelo de objetos representa la estructura estática del sistema, mientras que el dinámico representa la estructura de control del sistema. Un diagrama de estado, como una clase de objetos, es un esquema que describe un rango de secuencias. Un escenario es para un modelo dinámico como un diagrama de instancias para un modelo de objetos.

La ocurrencia de un evento normalmente está ligado a una condición y en consecuencia también lo están las transiciones, dado que un evento provoca una transición. Una condición es una expresión lógica, por lo que los únicos resultados posibles son Verdadero y Falso; por ejemplo, "la temperatura está bajo cero" o "temperatura<0". Una condición es válida o verdadera sólo sobre un intervalo de tiempo; por ejemplo, Alejandro Palomo Alcazar tuvo un sueldo de N$5,300 del 16 de mayo de 1996 al 30 de Octubre de 1997.

Una actividad es una operación que toma tiempo para realizarse y está asociada a un estado. Las actividades incluyen operaciones continuas, como desplegar un cuadro en la pantalla de una televisión; y operaciones secuenciales que terminan por sí mismas después de un intervalo de tiempo, tal como cerrar una válvula o realizar un cálculo. Un estado puede controlar actividades continuas, como el sonido de la campana de un teléfono que persiste hasta que el evento lo termina causando una transición del estado; y actividades secuenciales, como el levantar el brazo mecánico de un robot que se realiza mediante una serie de pasos secuenciales.

En contraste con las actividades que son operaciones que se realizan en un lapso de tiempo, existen operaciones instantáneas a las cuales se les conoce como *acciones*, y aunque no hay operaciones estrictamente instantáneas, el representarlas como acciones en un modelo indica que no es trascendente su estructura interna para propósitos de control, ya que cuando sí es importante la operación deberá representarse como actividad, con un evento inicial, uno final y posiblemente algunos intermedios.

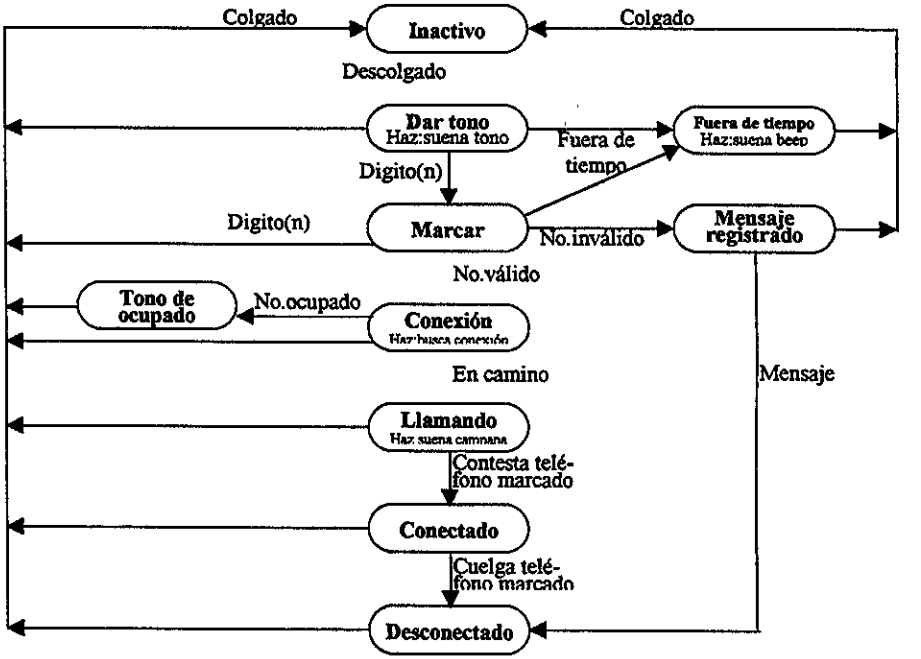
En la notación OMT una acción en una transición se presenta con una diagonal (/) seguida del nombre o descripción de la acción después del nombre del evento que la causa. A continuación se muestra el diagrama de estado para un menú pop-up en una estación de trabajo, cuando se presiona el botón derecho del mouse, el menú se despliega; cuando el botón derecho se libera, el menú se borra. Mientras el menú es visible, la opción resaltada del menú es actualizada al mover las teclas de movimiento (flechas, inicio, fin ,etc.).



Los diagramas de estado en la notación OMT toma la siguiente forma.

El nombre de estado se escribe en negritas dentro de una caja ovalada. El nombre del evento se escribe sobre la flecha de la transmisión y puede opcionalmente seguir por uno o más atributos entre paréntesis. Las condiciones se indican entre paréntesis cuadrados después del nombre del evento y los atributos, si los hay. Una actividad se indica dentro de la caja ovalada del estado mediante la palabra "*haz:*" seguida por el nombre o descripción de la actividad, es decir, en notación de la OMT el letrero "*haz:* A" dentro de una caja de estado indica que la actividad "A" empieza cuando entra el estado y se detiene cuando sale. Una acción se indica sobre una transición después del nombre del evento y separada por una diagonal(/). Todas estas construcciones son opcionales en los diagramas de estado.

A continuación se muestra el diagrama de estado para una línea telefónica, previamente mostrada pero ahora con acciones y actividades.



Como una alternativa para mostrar acciones en las transiciones, las acciones pueden asociarse con la entrada o salida de un estado.

Si se especifican operaciones múltiples en un estado, se realizan en el siguiente orden: acciones en las transiciones de entrada, acciones de entrada, actividades, acciones de salida, acciones en las transiciones de salida. Las actividades pueden interrumpirse por eventos que provoquen transiciones fuera del estado, pero las acciones de entrada y salida siempre se terminan.

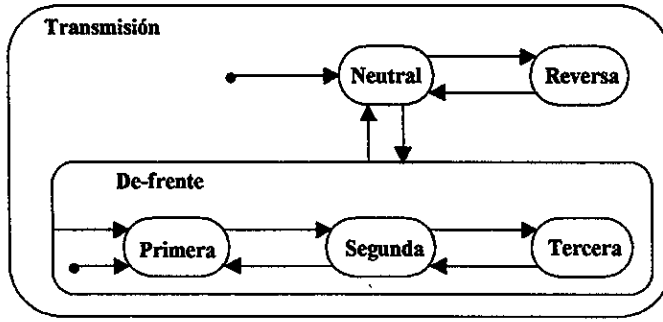
Los diagramas de estado hasta aquí descritos son no estructurados; sin embargo, pueden estructurarse para permitir descripciones concisas de sistemas complejos.

Las formas de estados estructurados son similares a las formas de objetos estructurados: generalización y agregación. La generalización equivale a expander actividades anidadas. Esto permite describir una actividad a un alto nivel, para posteriormente difundirla a un nivel más bajo adicionando detalles, similarmente a la llamada de un procedimiento anidado. Además, la generalización permite acomodar estados y eventos dentro de estructuras jerárquicas aprovechando el comportamiento común, al igual que la herencia de atributos y operaciones en clases.

La agregación permite particionar un estado en componentes independientes, con una interacción limitada entre ellos, igual que la agregación jerárquica de un objeto. La agregación es equivalente a concurrencia de estados, la cual generalmente corresponde a agregaciones de objetos.

Los estados también pueden tener subestados que heredan transiciones de sus superestados, igual que las superclases tiene subclases que heredan sus atributos y operaciones. Cualquier transición o acción que aplica a un estado aplica a todos sus subestados, a menos que los sobrepasen mediante una transición equivalente en el subestado.

Por ejemplo, en un diagrama de estado para una transmisión automática, la transmisión puede estar en REVERSA, NEUTRAL o DE-FRENTE; si está en DE-FRENTE, puede estar en PRIMERA. SEGUNDA o TERCERA velocidad. Los estados PRIMERA, SEGUNDA y TERCERA son subestados del estado DE-FRENTE. La notación de generalización para los estados es diferente a la usada en clases, un superestado se dibuja como una caja redondeada encerrando a todos los subestados. Los subestados a su vez pueden encerrar a sus subestados y así sucesivamente hasta el nivel que el problema lo amerite. Debido a que las cajas redondeadas representan a varios estados que están anidados, Harel las llamó *contornos*.



Es posible representar situaciones más complicadas, tales como una transición explícita de un subestado a otro fuera del contorno, en tales casos todos los estados deben aparecer sobre un diagrama usando la notación de contorno.

El estado de un sistema completo no puede representarse mediante un solo estado en un solo objeto; esto es el producto de los estados de todos los objetos en él.

Un diagrama de estado compuesto es una colección de diagramas de estado, y la interacción entre ellos se permite gracias a que las transiciones de un objeto dependen de otro objeto, mientras se preserva la modularidad.

Las acciones de entrada y salida son particularmente útiles en diagramas de estado anidados porque ellas permiten expresar un estado (posiblemente un subdiagrama completo) en términos de acciones correspondientes de entrada-salida sin considerar que pasa antes o

después de activar al estado. Es posible usar acciones ligadas a transiciones tanto como acciones de entrada y salida en un diagrama.

2.3 Modelo Funcional

El modelo funcional describe los cálculos que se realizan dentro de un sistema, es decir muestra como los valores de salida en una operación o función se derivan de los de entrada, sin considerar el orden en el que los valores se calculan. Así, se tiene que el modelo funcional especifica que pasa, el dinámico cuando pasa y el de objetos a que le pasa.

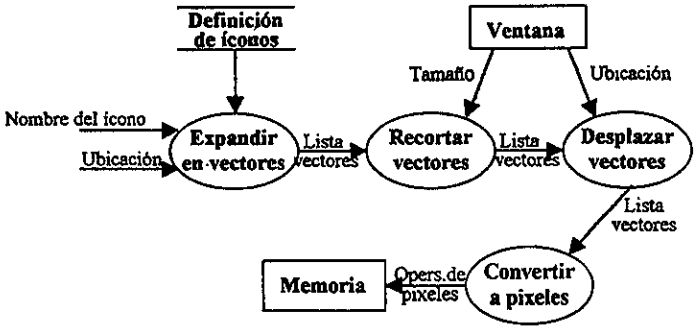
El modelo funcional dá significado a las operaciones y restricciones del modelo de objetos, así como a las acciones del modelo dinámico; dicho de otra forma un modelo funcional representa los resultados de un cálculo sin indicar cómo o cuándo se obtuvieron.

El cálculo de impuestos es una buen ejemplo de un modelo funcional. Especifica fórmulas para calcular impuestos basados en entradas, gastos, donaciones, etc.; el cálculo de impuestos define objetos y contiene información dinámica. Las formas para declarar impuestos especifican como calcularlo, basadas en una serie de valores de entrada, tales como sueldo, gastos, deducciones, etc. Dichas formas sólo proporcionan un algoritmo para calcular impuestos; no definen la función del impuesto actual. No se necesita llenar la forma en la secuencia exacta dada en el instructivo para dar las respuestas correctas.

En la notación OMT el modelo funcional se representa con diagramas de flujo de datos, los cuales muestran la trayectoria de valores desde las entradas externas hasta las salidas, pasando por almacenes internos. Un modelo funcional puede representarse por varios diagramas de flujo denotando el significado de las operaciones y restricciones.

Un diagrama de flujo de datos representa gráficamente la relación funcional de los valores calculados por un sistema, incluyendo valores de entrada, de salida y datos internos. Un diagrama de flujo de datos no muestra la información de control, tal como la hora en la que los procesos se realizan, o las decisiones entre varias rutas de datos, esta información pertenece al modelo dinámico; ni tampoco muestra la organización de valores dentro de los objetos, esta información pertenece al modelo de objetos.

Por ejemplo, el diagrama de flujo de datos para desplegar un ícono en un sistema de ventanas, muestra la secuencia de transformaciones realizadas, así como los valores y objetos externos que afectan el cálculo.



Un diagrama de flujo de datos contiene *procesos* para transformar datos, *Flujos de datos* para moverlos, *actores* los cuales son objetos que producen y consumen datos y *almacen de datos* los cuales son objetos que guardan información pasivamente.

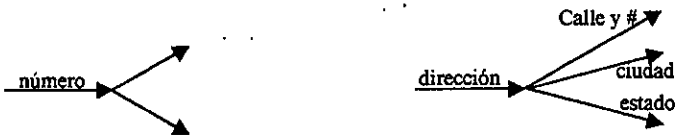
Un *proceso*, se dijo, transforma valores de datos y se implementan como métodos de operaciones (o fragmentos de ellos) de una clases de objetos. Los procesos de más bajo nivel son exclusivamente funciones sin efectos colaterales; ejemplos de funciones típicas incluyen la suma de dos números, el cargo financiero en un conjunto de transacciones de tarjeta de crédito, la ranura a través de una lista de puntos, etc. Un flujo de datos completo agrupa un proceso de alto nivel.

Un proceso se representa mediante una elipse con la descripción de la transformación y su nombre. Cada proceso tiene un número fijo de flechas que representan datos de entrada y de salida, en cada flecha se indica el tipo dato y pueden estar etiquetadas, aunque con frecuencia el tipo del valor en el flujo de datos es suficiente. Un proceso puede tener más de una salida.

Un diagrama sólo muestra el esquema de entradas y salidas, aunque los cálculos de valores a partir de los de entrada deben también especificarse.

Un *flujo de datos* conecta a través de una flecha la salida de un objeto o proceso a la entrada de otro objeto o proceso. Esto representa un valor intermedio dentro de un cálculo, el cual no se modifica por el flujo de datos.

La flecha se etiqueta con la descripción del dato, normalmente es el nombre o el tipo. El mismo valor puede enviarse a varios lugares; esto se indica con una bifurcación con varias flechas y una sola base, donde las ramificaciones no se etiquetan si representan el mismo valor, etiquetando sólo la base de donde salen.



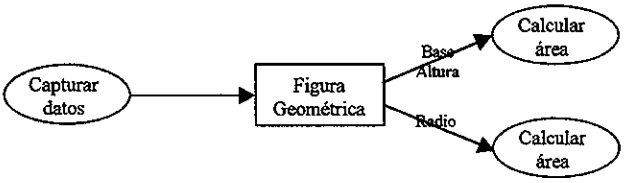
Sin embargo, puede suceder que un valor se divida en componentes, donde cada uno vaya a un proceso diferente. Esto se representa también con una bifuración, donde cada flecha de salida se etiqueta con el nombre del componente.

Cada flujo de datos representa un valor en algún punto del cálculo. El flujo interno de datos a un diagrama representa valores intermedios dentro de un cálculo y no necesariamente tiene algún significado en el mundo real.

Los flujos en los límites de un diagrama de flujo de datos son las entradas y salidas, los cuales pueden o no estar conectados a objetos, dependiendo de si el diagrama es un fragmento del sistema.

Un *actor* es un objeto activo que se dirige al diagrama de flujo de datos por los valores producidos o consumidos. Los actores se ligan a entradas y salidas de una gráfica de flujo de datos y hasta cierto punto descansan sobre los límites, por lo que algunas veces se les llama *terminales*.

Un actor se representa por un rectángulo para mostrar que es un objeto, las flechas entre el actor y el diagrama son entradas y salidas del diagrama.



Un *almacén de datos* es un objeto pasivo dentro de un diagrama de flujo de datos que almacena datos para accesarlos más tarde. A diferencia de un actor, un almacén de datos no genera ninguna operación por si mismo, sólo responde a requerimientos para conservar o accesar datos. Un almacén de datos permite accesar valores en diferente orden al que se generaron Los almacenes de datos agregados, tales como listas y tablas, proporcionan acceso de datos a través del orden de inserción o de índices.

Un almacén de datos se representa por un par de líneas paralelas con el nombre entre ellas. Flechas de entrada indican información u operaciones que modifican los valores almacenados. Flechas de salida indican información recuperada.

Un diagrama de flujo de datos es particularmente útil para mostrar la funcionalidad de alto nivel de un sistema y su separación en unidades funcionales más pequeñas. Un proceso puede expanderse en otro diagrama de flujo de datos, cada entrada y salida del proceso es una entrada y salida del nuevo diagrama, el cual puede tener almacenes de datos que no se muestren en diagramas de más alto nivel; a esta expansión se le llama anidación de diagramas de flujos de datos, donde el grupo de diagramas anidados forma un árbol. La anidación de un diagrama de datos permite a cada nivel ser coherente y entendible por sí sólo, aunque la funcionalidad global pueda ser compleja. Un diagrama que hace referencia a si mismo representa cálculos recursivos. A la anidación de diagramas también se le llama *nivelación*, ya que los diagramas se organizan en diferentes niveles.

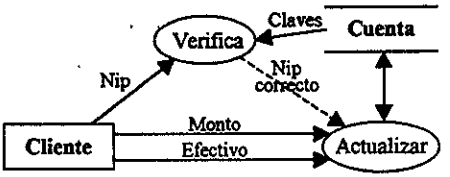
El anidamiento de diagramas normalmente termina con funciones simples, las cuales deben especificarse como operaciones, donde cada operación puede especificarse de varias formas, por ejemplo:

- Funciones matemáticas, tales como funciones trigonométricas.
- Tabla de valores de entrada y salida (enumeración) para un conjunto finito.
- Ecuaciones especificando salidas en términos de entradas.
- Pre y post-condiciones (definición axiomática).
- Tablas de decisión.
- Pseudocódigo.
- Lenguaje natural.

En algunas casos es útil incluir decisiones en lugar de proporcionar un valor a las funciones en el modelo funcional, con la finalidad de que no pasen desapercibidas y puedan mostrarse sus dependencias. Esto es posible introduciendo un *control de flujos* en un diagrama de flujo de datos. El control de flujo puede en muchas ocasiones ser útil, pero como duplican parte de la información del modelo dinámico se recomienda usarlo con moderación.

Un control de flujo genera un valor lógico que define si un reproceso es evaluado sin que sea un valor de entrada al propio proceso. El control de flujo se representa por una línea punteada que genera el valor lógico al proceso que se está controlando.

En el diagrama de flujo de control para un retiro de una cuenta bancaria, el cliente proporciona el Nip y una cuenta, el retiro ocurre sólo si el Nip se verifica exitósamente. El proceso de actualización podría expanderse con un control de flujo similar para evitar sobregiro.



Capítulo 3 Proceso de la Metodología OMT.

Las metodologías de ingeniería de software son procesos para desarrollar sistemas de una manera organizada, apoyadas en una colección de técnicas y convenciones de notación. Las etapas de producción de sistemas usualmente se organizan dentro de un ciclo de vida, el cual consta de varias fases de desarrollo, abarcando desde la formulación inicial del problema, pasando por el análisis, diseño, implementación y pruebas del software, y terminando con una fase operacional donde se realizan mantenimiento y mejoramiento.

Esto se puede llevar a cabo de dos maneras:
- Dividir el sistema y realizar todas las etapas para cada parte en forma evolutiva e iterativa.
- Realizar etapa por etapa para el sistema completo.

La metodología OMT se aplica igualmente bien en ambos casos.

En un enfoque orientado a objetos mucho del esfuerzo de desarrollo de sistemas descansa sobre la fase de análisis, lo cual muchas veces se refleja al invertir más tiempo en él; sin embargo, este esfuerzo adicional se ve más que compensado con una fácil y rápida implementación y expansión si se requiriera, dado que el diseño resultante es más limpio y adaptable.

También un enfoque orientado a objetos dirige su atención a estructuras de la información en lugar de a las funciones realizadas, permitiendo así la unificación de conceptos dentro de un sistema a través del proceso, dando como resultado el *objeto*. Los otros conceptos, tales como funciones, relaciones y eventos se organizan alrededor de los objetos por lo que la información registrada durante el análisis no se pierde ni se transforma al tomar lugar el diseño o la implementación, simplemente se complementa.

Las estructuras de datos de una aplicación y las relaciones entre ellas son mucho menos vulnerables que los cambios requeridos en las operaciones realizadas sobre los datos. Organizar un sistema en torno a objetos más que en torno a funciones dá al proceso de desarrollo una estabilidad de la cual carece un enfoque orientado a funciones.

En la Técnica de Modelación de Objetos, el modelo de objetos desarrollado durante el análisis se usa para el diseño y la implementación, y el trabajo se canaliza progresivamente dentro del refinamiento del modelo en niveles más detallados en lugar de pasarlos de una representación a otra, para posteriormente realizar una serie de revisiones y alcanzar niveles más profundos de detalle. Cada paso adiciona, aclara o depura características en vez de modificar el trabajo que ya se ha realizado, por lo que existe menos probabilidad de introducir inconsistencias y/o errores.

Un enfoque orientado a objetos simplifica enormemente obtener un diseño limpio, modular y fácil de entender, generando con eso sistemas mucho mas sencillos de probar, de darles

mantenimiento y expander que los diseñados con alguna metodología tradicional. Por esta razón la metodología OMT abarca sólo las fases:

- Análisis
- Diseño de Sistemas,
- Diseño de Objetos, e
- Implementación.

3.1 Análisis

El propósito del análisis orientado a objetos es crear el modelo de un sistema del mundo real para que pueda entenderse. Para lograrlo, se debe plantear el problema a resolver, examinar sus requerimientos, analizar sus implicaciones y reformulárselos rigurosamente. Primero se deben abstraer aspectos importantes y dejar detalles para después. Un buen modelo de análisis indica qué se debe hacer, sin restringir cuándo o cómo hacerse, y deben evitarse decisiones de implementación. Así, el modelo de análisis es una representación precisa y concisa del problema que permite contestar preguntas y construir una solución.

El análisis empieza con la especificación del problema, la cual no tiene que ser formal, pudiendo incluso estar incompleta; a lo largo del análisis se complementará volviéndose precisa, aclarando las ambigüedades e inconsistencias que pudieran existir, es decir, la especificación del problema no deberá tomarse como inmutable, aunque si deberá servir como base para el refinamiento de requerimientos reales, pues de ahí se abstraen las características escenciales.

El modelo de análisis maneja los tres aspectos de un objeto: estructura estática (modelo de objetos), secuencia de interacciones (modelo dinámico) y transformación de datos (modelo funcional). Los tres submodelos no son igualmente importantes en cada tipo de problema. Así por ejemplo, se puede decir que casi todos los problemas derivados de entidades del mundo real tienen modelos de objetos importantes, los problemas concernientes a interacciones y tiempos, tales como interfases con usuarios y procesos de control, tiene modelos dinámicos importantes; y los problemas que con cálculos significativos, tales como compiladores y cálculos de ingeniería, tiene modelos funcionales importantes. Los tres submodelos contribuyen con operaciones que en conjunto conforman el modelo de objetos.

El procedimiento para llevar a cabo el análisis no siempre se realiza en el mismo orden, de hecho los modelos grandes se construyen en forma cíclica, esto es, primero se crea un boceto del modelo de objetos, el cual se va mejorando y complementando a través de una serie de revisiones hasta obtener una representación detallada del problema a resolver.

El analista debe estar en comunicación continua con el usuario para aclarar cualquier ambigüedad o concepto, fomentando con esto la comunicación entre las partes involucradas,

lo cual es un punto muy importante para el buen desarrollo y funcionamiento de cualquier sistema.

3.1.1 Especificación del problema.

Como ya se mencionó, el primer paso en cualquier desarrollo es formular los requerimientos. Esto aplica para todo tipo de sistemas, incluyendo a los que requieren esfuerzos de grandes grupos.

Un punto que no debe perderse de vista es que la especificación es el inicio del entendimiento del problema por lo que es común encontrar que sean ambigüas, incompletas y aún inconsistentes.

El propósito del análisis es entender totalmente el problema y sus implicaciones, el analista debe trabajar con el usuario para afinar los requerimientos significativos (para fines del usuario) y realizar una investigación exhausta para completar la información.

3.1.2 Modelo de Objetos.

Una vez que se tiene la declaración del problema se está en condiciones de iniciar con el análisis propiamente dicho, el primer paso es construir el modelo de objetos, el cual muestra la estructura estática de datos del sistema, describiendo clases de objetos del mundo real y la relación entre ellos.

El nivel más alto de organización del sistema, dentro del modelo de objetos, alcanzado mediante asociaciones es el punto más crucial, así como el punto menos crítico son las particiones de más bajo nivel dentro de clases.

El modelo de objetos precede al dinámico y al funcional porque la estructura estática es usualmente mejor definida, menos dependiente respecto a detalles de aplicación, más estable y más fácil para el entendimiento humano.

La información para el modelo de objetos se deriva de la declaración del problema, del conocimiento profundo del dominio de la aplicación y del conocimiento general del mundo real.

Para construir un modelo de objetos se pueden seguir los siguientes pasos:

- Identificar objetos y clases.
- Preparar diccionario de datos.
- Identificar asociaciones entre objetos, incluyendo agregaciones.
- Identificar atributos de objetos y ligas
- Organizar y simplificar clases de objetos mediante herencia.

- Verificar que rutas de acceso existen para consultas.
- Hacer revisiones necesarias para obtener un modelo depurado.
- Agrupar clases dentro de módulos.

El primer paso en la construcción de un modelo de objetos es identificar las clases relevantes para la aplicación, las cuales no siempre aparecen en forma explícita en la declaración del problema de tal forma que se tienen que deducir, tanto del ámbito de la aplicación como del conocimiento general que se tenga de ella. Los objetos incluyen entidades físicas, como casas, empleados, máquinas, etc., y conceptos, tales como trayectorias, asignación de asientos, horarios de pago, etc. Todas las clases deben tener sentido para el objetivo de la aplicación.

Dentro de la declaración del problema, las clases con frecuencia corresponden a sustantivos; por ejemplo, en la declaración "Un sistema de reservaciones para vender boletos de obras de varios teatros", las clases tentativas podrían ser *Reservaciones, Boletos, Obras* y *Teatros.*

En un inicio no se debe ser muy selectivo, sino por el contrario se recomienda anotar todas las clases que vengan a la mente sin preocuparse de la herencia o la relación que pudiera haber entre ellas, el único intéres en este momento es definir clases.

El siguiente paso sería depurar la lista de clases obtenida, descartando las innecesarias o incorrectas, para lo cual podrían ayudar los siguientes criterios:

- Si dos clases representan la misma información, el nombre más descriptivo debería conservarse y eliminarse el otro.

- Si la existencia de una clase no impacta el comportamiento del problema debería eliminarse. Aunque para hacerlo se recomienda evaluarlo antes, porque bajo otro contexto la clase podría ser importante.

- Si alguna de las clases definidas denota propiedades de algún objeto, se debe evaluar si su existencia independiente es importante para la resolución del problema o debe integrarse a alguna otra clase como atributo.

- Si una clase describe sólo una operación y no maneja características propias, entonces debería integrarse a otra clase en forma de operación.

Antes de eliminar cualquier clase, es importante analizar con todo detalle si dentro del contexto del problema esa clase no encaja como el rol de alguna otra y replantearse si realmente está sobrando.

Una vez que se han definido y revisado las clases de objetos, es importante describir el alcance de cada una de ellas dentro del contexto del problema a resolver, incluyendo bases y restricciones, para conformar con esto un diccionario.

El siguiente paso es identificar asociaciones entre las clases definidas, partiendo de que cualquier dependencia entre dos o más clases es una asociación, así como la referencia de

una clase a otra. Normalmente las asociaciones muestran dependencias entre clases de un mismo nivel de abstracción, y pueden implementarse en varias formas, aunque tales decisiones de implementación deberán mantenerse fuera del modelo del análisis con la finalidad de preservar libertad de diseño.

En la declaración del problema las asociaciones con frecuencia corresponden a: una localización física (siguiente a, parte de, contenido en), acciones direccionadas (conduce), comunicación (hablar a), propiedad (tiene, parte de), o satisfacción de alguna condición (trabaja para, casado con). Se sugiere anotarlas en papel primero sin tratar de afinar detalles demasiado pronto y que la lista que se obtenga se revise con el usuario y/o con los expertos en el ámbito del problema a resolver para complementarla.

Una vez que se tenga la lista revisada con los expertos, se recomienda analizarla de una forma más profunda con la finalidad de suprimir aquellas que no sirvan o que simplemente sean redundantes.

En esta revisión exhaustiva se deben eliminar todas las asociaciones que estén fuera del dominio del problema o traten con construcciones de implementación. Una asociación debería describir una propiedad estructural del dominio de la aplicación, no un evento transitorio.

El nombre de una asociación es importante por lo que debe seleccionarse con mucho cuidado; el nombre debe sugerir que representa y no cómo o por qué. Igualmente es importante indicar el rol donde lo amerite; así, el nombre del rol debe describir el papel que juega una clase en la asociación desde el punto de vista de otra clase. Por ejemplo, en la asociación *Trabaja-para*, la clase *Compañía* tiene el papel *emplear* y la clase *Persona* tiene el papel *emplearse*.

Si existe una sóla asociación entre un par de clases y el nombre de una clase describe adecuadamente su papel, se pueden omitir los nombres. Por ejemplo, los roles en la asociación *La computadora central se comunica con los cajeros automáticos* son claramente los nombres de las clases. Una asociación entre dos instancias de las misma clase (una asociación reflexiva) requiere los nombres de papeles para distinguir las instancias. Por ejemplo, la asociación *Personas maneja a personas* podría tener los papeles de *Jefe* y *Trabajador*.

Otro punto importante en una asociación es la multiplicidad, la cual puede reducirse definiendo un *calificador*. Un calificador distingue objetos en el lado "muchos" de una asociación Por ejemplo, el calificador *clave de banco* distingue a los diferente bancos de un consorcio. Cada tarjeta de efectivo necesita una clave de banco para que las transacciones puedan direccionarse al banco apropiado.

Como ya se mencionó, en una asociación es importante especificar la multiplicidad, pero no se debe hacer demasiado esfuerzo en indicarla correctamente ya que ésta cambia durante el

análisis; también se debe tomar en cuenta si los objetos necesitan ordenarse de alguna forma en especial.

El siguiente paso es identificar atributos de objetos, los cuales son propiedades de objetos individuales, tales como nombre, peso, velocidad o color. Los atributos no son objetos, aunque usan una asociación para mostrar cualquier relación entre dos objetos.

Los atributos en la declaración del problema usualmente corresponden a adjetivos seguidos de frases posesivas, tales como "el color del automóvil" o "la posición del cursor". A diferencia de clases y asociaciones, es menos probable encontrarlos en la declaración del problema.

Se debe utilizar el conocimiento que se tenga sobre el dominio del problema y del mundo real para encontrarlos. Afortunadamente los atributos pocas veces afectan la estructura básica del problema.

Es recomendable no considerar atributos en exceso, sólo aquellos que estén directamente relacionados con la aplicación. Se deben tomar en cuenta primero los atributos más importantes, los detalles finos pueden adicionarse más tarde.

El siguiente paso es organizar las clases usando herencia para compartir estructuras en común. La herencia puede aplicarse en dos direcciones: hacia arriba, generalizando aspectos comunes de clases existentes dentro de una superclase, o hacia abajo afinando clases existentes dentro de subclases especializadas.

Se puede descubrir herencia hacia arriba buscando clases con atributos, asociaciones u operaciones similares; para cada generalización se define una superclase quien contiene las características comunes. Por ejemplo, *Transacción remota* y *Transacción de caja* son similares, excepto en su iniciación, por lo que puede generalizarse con *Transacción*. Algunos atributos y aún clases pueden redefinirse levemente para ajustar propiedades. Aunque se recomienda no forzar dichos ajustes ya que se pueden tener errores de generalización.

La herencia múltiple puede usarse para explotar las características comunes de un grupo de clases, pero sólo si es estrictamente necesario, porque esto incrementa la complejidad tanto en lo conceptual como en la implementación. En el uso de la herencia múltiple, con frecuencia es posible designar una superclase primaria la cual contiene la mayor parte de la estructura heredada y de comportamiento, las superclases secundarias adicionan detalles laterales. Una vez estructurado jerárquicamente al grupo de clase se puede proceder a trazar rutas de acceso.

Un modelo de objetos raramente es correcto al primer intento, el proceso completo de desarrollo de sistemas es una iteración continua; diferentes partes del modelo están con frecuencia en diferentes etapas. Si se encuentra una deficiencia, se regresa a etapas anteriores para hacer los ajustes pertinentes, incluso algunos refinamientos pueden realizarse sólamente después de que se terminan los modelos dinámico y funcional.

Durante las revisiones del modelo de objetos se puede notar la falta o sobra de algunos objetos, asociaciones y atributos, o bien que simplemente se encuentren en el lugar equivocado.

A continuación se presentan algunos signos para detectarlo:

- Si se detectan asimetrías en asociaciones y generalizaciones, se recomienda adicionar nuevas clases por semejanza.

- Si los atributos y operaciones están disparejas en una clase, se debe dividir la clase de tal forma que cada parte sea coherente.

- Si se tiene dificultad en hacer generalizaciones, entonces alguna clase puede estar jugando dos papeles.

- Si se detectan asociaciones con el mismo propósito, se debe hacer la generalización de la superclase que falta para que las una.

- Cuando un rol forma la lógica de la clase, tal vez debería separarse. Esto frecuentemente significa convertir una asociación en una clase.

- Si una clase carece de atributos, operaciones y asociaciones, analizar a fondo por o para qué es necesaria, si lo es.

- Y si por el contrario se observa información redundante en las asociaciones, se deben eliminar aquellas que no adicionan nueva información o las que se marcan como derivadas.

En la práctica, la construcción de modelos no es tan rígidamente ordenada como se ha presentado, se pueden combinar varios pasos una vez que se va teniendo práctica. Por ejemplo, se pueden identificar clases y eliminar las incorrectas sin tener que anotarlas para después adicionarlas todas juntas al diagrama de objetos con sus asociaciones. Se puede tomar una parte del modelo y desarrollarlo en detalle, mientras otras partes son aún un bosquejo. El orden de los pasos se puede cambiar cuando sea apropiado, pero si se está aprendiendo a hacer modelos de objetos se recomienda se sigan los pasos a todo detalle las primeras veces.

El último paso de la modelación de objetos es agrupar las clases dentro de hojas y módulos. Los diagramas pueden dividirse en hojas de tamaño uniforme por conveniencia en el dibujo, impresión y consulta. Las clases estrechamente unidas deberían permanecer juntas en la medida de lo posible, aunque algunas aparezcan en varias hojas.

Un módulo es un conjunto de clases (una o más hojas) que captan algún subconjunto lógico del modelo completo y pueden variar de tamaño.

Todo esto facilitará la ampliación del sistema o el reuso de los módulos, ya que si el nuevo problema es similar a uno previo, el diseño original puede extenderse para abarcar a ambos.

Sin embargo, no se debe descartar el ser muy cuidadoso para decidir si esto es mejor a construir un nuevo diseño.

3.1.3 Modelo Dinámico.

El modelo dinámico muestra el comportamiento, a través del tiempo, del sistema y los objetos en él. El análisis dinámico empieza buscando eventos y estímulos externos, resumiendo la secuencia de eventos permisibles para cada objeto con un diagrama de estado. El algoritmo de ejecución no es relevante durante el análisis si no hay manifestaciones externas visibles, dado que los algoritmos son parte de la implementación.

Dependiendo del dominio del problema, el modelo dinámico será más o menos amplio, teniendo que es insignificante para un depósito puramente de información estática, tal como una base de datos, pero es importante para sistemas interactivos.

Para construir un modelo dinámico se sugiere seguir los siguientes pasos:

- Preparar escenarios de secuencias típicas de interacciones.
- Identificar eventos entre objetos.
- Preparar un diagrama de eventos para cada escenario.
- Construir un diagrama de estado.
- Empatar eventos entre objetos para verificar consistencia.

Los diálogos típicos entre usuario y sistema sirven para captar el comportamiento esperado del sistema. En lugar de tratar de anotar directamente el modelo dinámico general se van construyendo escenarios para asegurar que los pasos importantes no se traslapen y que el flujo completo de las interacciones sea más natural y correcto.

Algunas veces la declaración del problema describe toda la secuencia de interacciones, aunque la mayoría de las veces se tendrá que complementar. La declaración del problema puede especificar necesidades de información, pero dejar abierta la manera en la cual se obtiene. Cuando la definición de entrada de datos es una tarea mayor o la única tarea mayor, el modelo dinámico es crucial en tales aplicaciones.

Para localizar las interacciones, se recomienda primero preparar escenarios para casos "normales", es decir, interacciones sin ninguna entrada inusual o condiciones de error; después se sugiere considerar casos "especiales", tales como omitir secuencias de entrada, valores máximos y mínimos, y valores repetidos; continuar con casos de error, incluir valores inválidos y fallas de respuesta; en muchas aplicaciones, error es la parte más difícil de la implementación. Finalmente considerar otros tipos de interacciones que pueden traslaparse sobre las básicas, tales como ayuda de requerimientos y estados de consultas.

Un *escenario* es una secuencia de eventos. Un evento ocurre siempre que una información se intercambia entre un objeto en el sistema y un agente externo, tal como un usuario, un sensor u otra tarea, es decir, cualquier ocasión que entra o sale información del sistema.

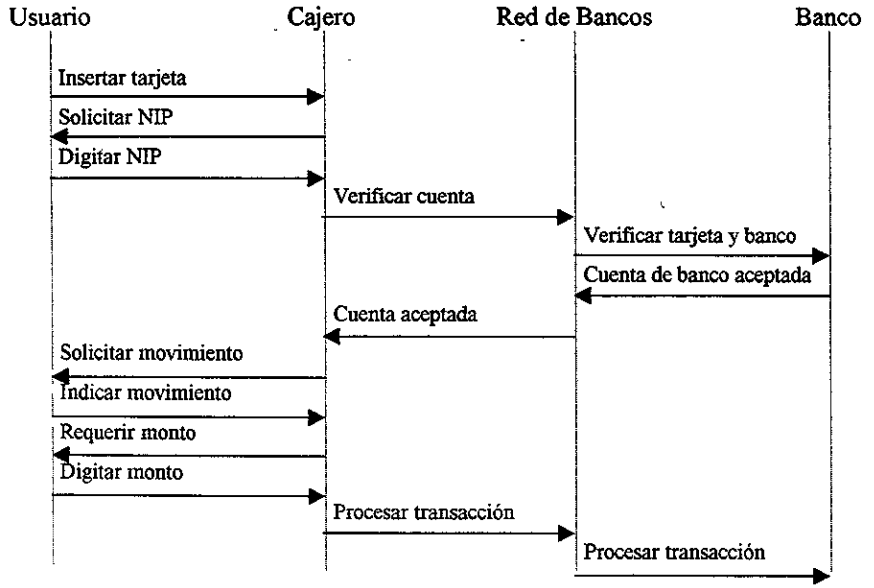
Para cada evento se debe identificar el actor (sistema, usuario u otro agente externo) que provoca el evento.

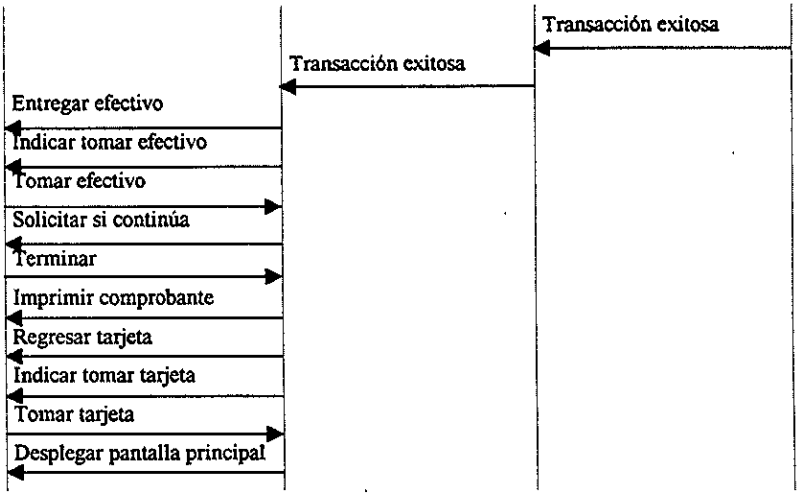
Se deben examinar los escenarios para identificar todos los eventos externos, los cuales incluyen todas las señales, entradas, decisiones, interrupciones, transiciones y acciones, ya sea a usuarios o dispositivos externos.

Una acción sobre un objeto que transmite información es un *evento*; por ejemplo, *Digita la clave* es un evento que va del agente externo *Usuario* al objeto de la aplicación *Cajero automático*.

Se sugiere agrupar bajo un solo nombre a todos los eventos que tienen el mismo efecto sobre el flujo de control, aún si los valores de parámetros son distintos. Por ejemplo, *Digita la clave* debería ser una clase de eventos ya que el valor de la clave no afecta al flujo del control. Los eventos que afectan al flujo de control deben diferenciarse.

Cada escenario se muestra como un *diagrama de eventos*, el cual es una lista ordenada de eventos entre diferentes objetos asignados a las columnas de una tabla. Si participa más de una clase en el escenario, se asigna una columna separada para cada objeto, así mediante la revisión de una columna del diagrama se pueden ver los eventos que afectan directamente a un objeto en particular. Sólo estos eventos pueden aparecer en el diagrama de estado para el objeto.





Los eventos en un grupo de clases, que bien podría ser un módulo, se esquematizan sobre un *diagrama de flujo de datos*, el cual resume eventos entre clases sin revisar la secuencia; se deben considerar eventos de todos los escenarios, incluyendo los de error. Las rutas en el diagrama de objeto muestran posibles flujos de información y las rutas en los diagramas de flujos de evento muestran posibles flujos de control.

El siguiente paso es construir el diagrama de estado para cada clase de objeto con un comportamiento dinámico no trivial, mostrando eventos de objetos receptores y transmisores. Cada escenario o diagrama de eventos corresponde a una ruta a través del diagrama de estado. Cada bifurcación en el flujo de control se representa mediante un estado con más de una transición de salida.

Después de que los eventos normales han sido considerados, se procede a adicionar los casos extremo y los especiales, considerando eventos que ocurren en momentos inoportunos, por ejemplo, un requerimiento para cancelar una transición después de que ésta ha sido presentada para proceso. En los casos cuando el usuario (u otro agente externo) no responde rápida y/o correctamente y algunos recursos pueden ser utilizados, un evento *fuera de tiempo* puede generarse después de un tiempo dado. Manejar errores de usuario claramente, con frecuencia requiere más pensamiento y código que los casos normales y complica estructuras de programación limpias y compactas, pero se debe hacer.

Se ha terminado con el diagrama de estado de una clase cuando el diagrama cubre todos los escenarios y maneja todos los eventos que pueden afectar un objeto de la clase en cada uno de sus estados. Se puede usar el diagrama de estado para sugerir nuevos escenarios, considerando si algún evento aún no manejado afecta el estado del objeto.

Peguntar ¿Qué pasa si? podría ser un buen camino para probar capacidades de manejo de error de una clase, así como para verificar si está terminado el diagrama o faltan escenarios por incluir.

Si existiesen interacciones complejas con entradas independientes, se puede organizar el modelo dinámico usando diagramas de estado anidados.

Eventualmente se tendrá la capacidad de escribir los diagramas de estado directamente sin preparar tablas de eventos. Unos cuantos escenarios son útiles, en cualquier caso.

El conjunto de diagramas de estado para clases de objetos con un comportamiento dinámico importante constituye el modelo dinámico para una aplicación.

3.1.4 Modelo Funcional.

El modelo funcional define como se calculan los valores sin revisar la secuencia, decisiones o estructura de objetos. El modelo funcional muestra que valores dependen de otros y las funciones relacionadas a ellos, es decir, los diagramas de flujo de datos son útiles para mostrar dependencias funcionales; las funciones son expresadas de varias maneras, incluyendo lenguaje natural, ecuaciones matemáticas o pseudocódigo.

Los procesos en un diagrama de flujo de datos corresponden a actividades o acciones en los diagramas de estado de las clases.

El modelo funcional se construye después de los modelos de objetos y el dinámico y para crearlo se sugieren los siguientes pasos:

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

Los valores de entrada y salida son parámetros de eventos entre el sistema y el mundo externo, así que para identificarlos se inicia obteniendo una lista de ellos, después se procede a examinar la declaración del problema para detectar cualquier otro valor de entrada o salida que pudiera faltar.

Los diagramas de datos muestran como se calcula cada valor de salida a partir de los valores de entrada. Un diagrama de flujo de datos usualmente se construye en capas, la superior

puede formarse de un solo proceso o de un proceso para cada grupo de entradas, valores calculados y salidas generadas.

Se debe ir de capas superiores a las interiores, de tal forma que se expanda cada proceso no trivial en el diagrama de nivel superior dentro de un diagrama de flujo de datos de un nivel más bajo. Si los procesos de segundo nivel aún contienen procesos no triviales pueden expanderse y así sucesivamente.

Los diagramas de flujo de datos especifican sólo dependencias entre operaciones, no muestran decisiones o secuencias de operaciones; de hecho, algunas pueden ser opcionales o mutuamente exclusivas.

Algunos valores de datos afectan decisiones en el modelo dinámico, pero las decisiones no afectan directamente los valores de salida en el modelo de flujo de datos, ya que este presenta todas las rutas posibles de cálculo.

Sin embargo, puede ser útil captar las funciones de decisiones en el modelo de flujo de datos, dado que 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 "datos sumergidos" dentro del diagrama de flujo de datos, sus salidas afectan al flujo de control en el modelo dinámico y no a valores de salida directamente; por ejemplo, *verificar clave* es una función de decisión. Y se ha mostrado que una señal de error se puede producir, pero deja implícito la flecha de control al proceso *actualizar cuenta*.

Cuando el diagrama de flujo de datos ha sido lo suficientemente refinado, se está en condiciones de describir cada función, enfocándose en qué hace y no en cómo se implementará.

La descripción puede estar en lenguaje natural, en ecuaciones matemáticas, pseudocódigo, tablas de decisiones o en alguna otra forma apropiada; puede ser declarativa o de procedimiento. Una descripción declarativa especifica las relaciones entre valores de entrada y salida, y las relaciones entre los valores de salida. Una descripción de procedimiento especifica una función dando un algoritmo que la calcule, el propósito del algoritmo es sólo indicar qué hace la función; durante la implementación cualquier otro algoritmo que calcule los mismos valores puede ser utilizado.

Son preferibles las descripciones declarativas a las de procedimientos porque no requieren de una implementación, pero si la descripción de procedimiento es mucho más fácil de escribir se recomienda usarla.

El siguiente paso es identificar restricciones entre objetos que no son mas que dependencias funcionales entre objetos que no están relacionados con la entrada-salida. Las restricciones pueden actuar sobre dos objetos al mismo tiempo, entre instancias del mismo objeto en diferentes momentos o entre instancias de diferentes objetos en diferentes momentos, aunque las últimas son usualmente funciones de entrada-salida. Las pre-condiciones sobre funciones son restricciones que los valores de entrada deben satisfacer y las post-

condiciones son restricciones que los valores de salida se comprometen mantener. Por lo que se recomienda formular los tiempos o condiciones bajo los cuales las restricciones se sostienen.

El modelo de análisis completo puede mostrar inconsistencias, por lo que se recomienda revisar las diferentes etapas para obtener un diseño más claro y coherente, mejorar las definiciones de objetos para optimizar el manejo de herencia y mejorar la estructura, y aumentar detalles que se olvidaron durante la primera revisión; no se deben evitar cambios que se requieran sólo porque ya se tiene un modelo, es más fácil hacerlo ahora que dejarlo para después.

El modelo final deberá verificarse con el usuario y expertos del dominio de la aplicación para asegurarse que se está representando correctamente el mundo real.

La verificación final del modelo de análisis sirve de base para la arquitectura del sistema, diseño e implementación. La declaración original del problema se revisará para incluir correcciones y conceptos que se descubrieron durante el análisis.

La meta del análisis es especificar totalmente el problema y el dominio de la aplicación sin introducir una inclinación a alguna implementación en particular, pero en la práctica es imposible evitar todas las mañas de implementación. No existe una línea absoluta entre las diferentes fases de diseño, ni existe un análisis perfecto. No se debe tratar de seguir las reglas que se dieron de una manera muy rigurosa. El propósito de las reglas es preservar flexibilidad y permitir cambios posteriores, pero no debe perderse de vista que la meta del modelaje es realizar un trabajo completo y la flexibilidad es sólo un medio para conseguirlo.

3.2 Diseño de Sistemas.

La fase de análisis se enfoca exclusivamente a *qué se necesita*, sin importar *como se conseguirá*, y no es sino hasta el diseño de sistemas donde se toman decisiones de cómo se resolverá el problema, es decir, es la etapa en donde se selecciona el enfoque básico para atacar el problema, partiendo de un nivel alto para ir obteniendo diferentes grados de detalle.

Durante el diseño del sistema se definen tanto la estructura completa como el estilo, esto es, la arquitectura del sistema, que no es más que la organización completa del mismo mediante componentes llamados *subsistemas*.

La arquitectura proporciona el contexto para tomar decisiones a más detalle en etapas subsecuentes. A través de las decisiones de alto nivel aplicadas al sistema entero, el diseñador divide el problema en módulos y así puedan realizarse trabajos posteriores mediante diseñadores independientes para los diferentes subsistemas.

El diseñador de sistemas debe tomar las siguientes decisiones:

- Organizar el sistema en subsistemas.

- Identificar concurrencia inherente en el problema.
- Asignar subsistemas a procesadores.
- Seleccionar un enfoque para manejar almacenes de datos.
- Manejar accesos a recursos globales.
- Seleccionar la implementación de control en los programas.
- Manejar condiciones de frontera.
- Establecer prioridades en intercambios.

Con frecuencia la arquitecutra completa de un sistema puede obtenerse basándose en similitudes con sistemas previos, encontrando que ciertos tipos de arquitecturas de sistemas son útiles para resolver una amplia gama de problemas.

Un subsistema no es un objeto ni una función, es un paquete de clases, asociaciones, operaciones, eventos y restricciones que están interelacionados, con una meta bien definida y, con optimismo, una pequeña interfase con otros subsistemas. Un subsistema usualmente se identifica por los *servicios* que proporciona. Un *servicio* es un grupo de funciones que comparten algunos propósitos comunes, tales como procesos de entrada y salida, dibujo de cuadros o cálculos aritméticos.

Un subsistema define una forma coherente de ver un aspecto del problema. Por ejemplo, el archivo de comandos dentro de un sistema operativo es un subsistema; comprende un grupo de abstracciones relacionadas importantes, además de otras abstracciones independientes en otros subsistemas, tales como el de manejo de memoria o el de control de proceso.

3.3 Diseño de Objetos.

El análisis determina qué debe hacer la implementación, mientras el diseño del sistema define el plan de ataque. La fase del diseño de objetos determina las definiciones completas de las clases y asociaciones usadas en la implementación, así como las interfases y algoritmos de los métodos usados para implementar operaciones; adiciona objetos internos para la implementación y optimización de estructuras de datos y algoritmos.

Durante el diseño de objetos se deben realizar los siguientes pasos:

- Combinar los tres modelos para obtener operaciones sobre clases.
- Diseñar algoritmos para implementar operaciones.
- Implementar control para interacciones externas.

- Ajustar estructura de clases para incrementar herencia.
- Diseñar asociaciones.
- Determinar representación de objeto.
- Empaquetar clases y asociaciones dentro de módulos.

El diseño orientado a objetos es un proceso cíclico, es decir, cuando se piensa que está completo a un nivel de abstracción, se pueden agregar más detalle y descomponer el diseño en un nivel más fino de detalle, incluso se puede encontrar que se tienen que agregar nuevas operaciones y atributos a las clases en el modelo de objetos y probablemente hasta se identifiquen nuevas clases, por lo que es muy recomendable revisar las relaciones entre objetos, incluyendo cambios en la jerarquía de herencia. No es extraño que se tenga que iterar varias veces.

3.4 Implementación.

La implementación es la última fase del desarrollo de sistemas a través de la metodología de OMT y consiste en transcribir los resultados del análisis y diseño a un lenguaje de programación o bien a un manejador de bases de datos, dependiendo del tipo de problema a resolver; si la tarea principal es el acceso continuo de información se recomienda un sistema de bases de datos, de otra forma un lenguaje de programación es la herramienta correcta para la implementación.

Como se había mencionado ya en algún momento, la herencia, el reuso, la claridad y el fácil crecimiento no es algo que se de por el simple hecho de utilizar una metodología orientada a objetos, por lo que la forma en que se escriban y estructuren los programas es muy importante, es decir , es recomendable seguir un estilo de programación para lograrlo.

La selección del lenguaje de programación depende de varios factores del tipo de problema, del equipo de cómputo que se tenga, de la economía de la empresa, etc.

Se pueden utilizar lenguajes de programación orientados a objetos o lenguajes no orientados a objetos, por supuesto que estos últimos requieren de un mayor cuidado y disciplina debido a que no cuentan con las ayudas necesarias para definir los conceptos básicos de la programación orientada a objetos de una manera automática y por tanto no detectan desviaciones del programador.

La implementación de un diseño orientado a objetos es más natural con un lenguaje orientado a objetos debido a que las estructuras de diseño son similares a las estructuras del lenguaje y se deben considerar los siguientes pasos:

- Definir clases.

- Crear objetos.
- Hacer llamadas a operaciones.
- Usar herencia.
- Implementar asociaciones.

Sólo cuando sea imprescindible se recomienda utilizar un lenguaje de programación no orientado a objetos y aún en este caso, el diseño orientado a objetos es beneficioso. Los pasos a seguir son prácticamente los mismos que cuando se utiliza un lenguaje orientado a objetos, sólo que el programador debe implementar los conceptos orientados a objetos. Los pasos a seguir son:

- Traducir clases a estructuras de datos.
- Pasar argumentos a métodos.
- Reservar espacio para objetos.
- Implementar herencia en estructuras de datos.
- Implementar resolución de métodos.
- Implementar asociaciones.
- Resolver concurrencia.
- Encapsular detalles internos de clases.

Si el problema amerita utilizar bases de datos se recomienda un manejador relacional debido a que son más flexibles y funcionales que los jerárquicos y los de red.

Capítulo 4. Planteamiento de una Aplicación.

De acuerdo al proceso de la metodología OMT de Rumbaugh el primer paso para el desarrollo de una aplicación es definir el problema que se quiere resolver, con el objeto de introducirse en el ámbito del sistema y tener un punto de partida para el análisis. En esta ocasión la aplicación que se planteará es: "La evaluación de una serie de plantas petroquímicas", y siguiendo con las indicaciones de la metodología aquí estudiada se presenta a continuación la especificación del problema.

4.1 Especificación del problema.

Durante las primeras entrevistas con el usuario sobre cuales son sus requerimientos de información y proceso se elabora la especificación del problema. Que para este caso es:

> **Evaluar el rendimiento de las diferentes plantas de un grupo petroquímico desde el punto de vista financiero, es decir, calcular las utilidades o pérdidas a partir de los ingresos y erogaciones que generan la producción; permitiendo definir varios escenarios y hacer comparaciones entre ellos con la finalidad de detectar posibles desviaciones, por ejemplo: definir los escenarios de lo programado y lo realizado para compararlos y así verificar que se cumplan los objetivos planteados.**
>
> **Este grupo está organizado mediante centros petroquímicos distribuidos a lo largo del país, los cuales son congregaciones de plantas. Cada planta requiere de materias primas, químicos, catalizadores y servicios auxiliares para obtener un producto principal y en ocasiones uno o más productos secundarios, todo esto representa egresos (materias primas, químicos, catalizadores y servicios auxiliares) e ingresos (productos y subproductos); además existen otros gastos asociados a la producción como sueldos, conservación y mantenimiento, almacenaje y depreciación, entre otros.**

Una vez que se cuenta con la especificación del problema se está en condiciones de proceder con el análisis propiamente dicho, ya que se tiene la información mínima para empezar a descomponer el problema en partes más fáciles de entender y/o plantear.

4.2 Análisis

El primer paso del análisis es definir las clases de objetos del sistema, para lo cual se debe revisar la especificación del problema y extraer una lista de los sustantivos que ahí aparezcan. Dicha lista es un indicio de las clases que la aplicación pudiera tener.

La lista de los sustantivos, es decir, de las posibles clases, se presenta a continuación en orden de aparición:

Rendimiento,
Plantas,
Grupo petroquímico,
Utilidades,
Pérdidas,
Ingresos,
Producción,
Escenarios,
Desviaciones,
Centros Petroquímicos,
País,
Congregaciones de Plantas,
Materias Primas,
Químicos y catalizadores,
Servicios Auxiliares,
Producto principal,
Productos secundarios,
Otros gastos.

Lo primero que salta a la vista al revisar esta lista de posibles clases con el usuario y expertos en la materia, es que un *Rendimiento* positivo es una *Utilidad* y uno negativo es una *Pérdida*, por lo tanto la existencia de las tres sería redundante; dado que para esta aplicación *Rendimiento* es más representativa se decidió eliminar *Utilidades y Pérdidas.* De igual forma *Centros Petroquímicos* y *Congregaciones de plantas* representan al mismo ente, es decir también son redundantes, por lo que debe eliminarse una de las dos. Dado que el nombre de *Centros Petroquímicos* es más descriptivo se desecha *Congregaciones de plantas*

Otro punto al que se llegó es que al usuario no le interesa un detalle contable de cada planta, sólo desea saber en términos generales cuánto gastó en materia prima, cuánto produjo, cuánto vendió y por supuesto a cuánto ascienden los gastos asociados tanto a la producción como a la venta de sus productos, para con esto poder calcular las ganancias o pérdidas de las diferentes plantas; y sólo si resultan datos inusuales remitirse al área pertinente. Por ejemplo, si un mes resulta muy alto el gasto en materia prima, entonces se revisarán las cuentas o subcuentas afectadas del sistema contable por posibles errores al registrar la información o bien revisar el área de producción por posibles desperdicios o mal

funcionamiento de la planta. Dicho de otra manera el sistema que se requiere permitirá llevar el control del funcionamiento de las plantas petroquímicas, sin entrar en demasiados detalles, ya que está dirigido a la alta dirección de la empresa.

Por lo anterior las clases de *Rendimiento*, *Ingresos* y *Egresos* no son importantes para el alcance de este sistema, dado que no impactan el comportamiento de la aplicación, en su lugar se definió la clase *Estados Financieros*, la cual es un resumen del comportamiento de una planta y las clases antes mencionadas tal vez pasarían a ser algunos de sus atributos. Aunque estos atributos serían derivados ya que *Rendimiento* es la diferencia entre *Ingresos* y *Egresos*, y a su vez los *Ingresos* son igual al volumen producido por el precio del producto respectivo y los *Egresos* son igual al volumen de materias primas, químicos y catalizadores, y servicios auxiliares por el precio de ellos; más sueldos, mantenimiento, etc.

También se observa que la clase *País* puede ser una muy importante pensando en modificaciones o expansiones futuras o bien como una clase que se reuse en otras aplicaciones; por ejemplo, se puede desarrollar un sistema de "BenchMarking", el cual consiste en comparar, en este ámbito, dos plantas gemelas (con la misma capacidad, la misma edad y el mismo proceso químico) de países diferentes. Algo típico es obtener diferencias funcionales y estructurales de plantas de México con respecto a las de Estados Unidos, obteniendo los datos de éstas últimas de publicaciones internacionales. Sin embargo, *País* no es una clase determinante a este punto de desarrollo por lo que se consideró desecharla por el momento.

Al revisar nuevamente la lista de clases candidatos, se encontró que *Productos Principales* y *Subproductos* son productos que obtiene una planta, la diferencia es que el principal es el objetivo, mientras el secundario es una implicación del proceso químico, por lo tanto estas dos clases se engloban en una sola llamada *Productos*.

También se llegó al acuerdo de renombrar *Otros Gastos* con *Costos de Operación*

En base a los comentarios anteriores las posibles clases se convierten en la siguiente lista de clases candidatos:

Grupo petroquímico,
Centros petroquímicos,
Plantas,
Materias Primas,
Químicos y Catalizadores
Servicios Auxiliares,
Productos Principales,
Costos de Operación,
Estados Financieros,
Escenarios y
Desviaciones.

El siguiente paso es construir el diccionario de clases, sin que esto implique que la lista de clases ya no pueda modificarse, pues como se mencionó en capítulos anteriores el proceso de la metodología OMT es un mecanismo iterativo, que termina hasta pulir todos aquellos detalles que de una forma u otra puedan afectar el comportamiento de la aplicación.

Siempre bajo la supervisión y aprobación del usuario, y con base en el conocimiento que se va adquiriendo sobre la aplicación a desarrollar se crea el diccionario de clases:

- *Grupo Petroquímico.*- Es una empresa cuyo objetivo es producir y comercializar los diferentes productos petroquímicos, organizada mediante una serie de centros petroquímicos y controlada bajo una sola administración, con la finalidad de supervisar que cada uno de los centros dé los mejores rendimientos posibles.

- *Centros Petroquímicos.*- Son conjuntos de plantas ubicadas físicamente en la misma área y organizadas bajo la misma administración. Existen dos tipos: los productivos y los de costos. Los centros productivos son aquellos cuyos componentes son plantas productivas, mientras que los de costos son donde se realizan trabajos administrativos o de comercialización (agencias de ventas). Cada centro es responsable del buen funcionamiento de sus elementos.

- *Plantas Petroquímicas.*- Son dispositivos donde se realizan una serie de procesos que efectúan transformaciones sobre derivados del petróleo mediante reacciones químicas para obtener uno o más productos petroquímicos.

- *Materias Primas.*- Son los "productos" que entran a una planta para someterse a las reacciones químicas y obtener uno o más productos petroquímicos. Pueden ser derivados del gas, del crudo e incluso petroquímicos, como *etano, gas natural, benceno, diesel*; o bien pueden ser cualquier tipo de mezcla o compuesto, como *aire, cloro o agua.* El registro de volúmenes y montos de materias primas es mensual y el corte del periodo es anual.

- *Químicos y Catalizadores.*- Son sustancias químicas que ayudan a realizar y controlar los procesos de las diferentes plantas sobre las materias primas. El registro de volúmenes y montos de los químicos es mensual y el corte del periodo es anual; el de catalizadores se distribuye en forma mensual de acuerdo a la vida útil de estos.

- *Servicios Auxiliares.*- Son los servicios que requiere una planta para que funcione, tales como *energía eléctrica, vapor, gas natural, agua de enfriamiento,* etc. Normalmente un centro productivo genera todos los servicios auxiliares que requieren sus plantas, esto es, cada centro tiene, por decirlo así, un módulo de servicios auxiliares que abastece a todas las plantas de dicho centro, aunque existen casos donde el centro vende servicios ya sea a otros centros o bien a clientes externos. El registro de volúmenes y montos de servicios auxiliares es mensual y el corte del periodo es anual.

- *Productos* - Son los productos petroquímicos que se obtienen como resultado de someter las materias primas a una serie de procesos químicos a través de una planta. Los productos principales siempre son petroquímicos, pero los secundarios no

necesariamente, como en el caso de las plantas de amoniaco cuyo producto principal es amoniaco -un petroquímico- y como producto secundario anhídrido carbónico -un no petroquímico-. El registro de productos es mensual y el corte del periodo es anual.

- *Costos de Operación.*- Son los gastos fijos asociados a cada planta, para que funcionen debidamente; para el caso de las plantas estas son erogaciones independientes de la adquisición de materias primas y servicios auxiliares (gastos variables). Mantenimiento, sueldos, viáticos, indemnizaciones, adquisición de materiales y depreciación son algunos rubros de los costos de operación; su registro es mensual y el corte del periodo anual.

- *Estados Financieros.*- Esta clase contiene un resumen de los gastos por compras de materias primas, químicos y catalizadores y servicios, de los gastos de operación, de los ingresos por ventas y por tanto de las utilidades o pérdidas. El nivel del resumen puede ser por planta, por centro o por todo el grupo petroquímico.

- *Escenarios.*- Partiendo de que:

  Rendimiento = Ventas - Compras - Costos de Operación

  donde los valores de ventas, compras y costos de operación representan un escenario. La variación de alguna(s) de estas variables genera un nuevo escenario.

- *Desviaciones.*- Son las diferencias en ventas, compras, costos de operación y por tanto de rendimientos en periodos y/o escenarios diferentes, sin importar si son o no del mismo año, siempre y cuando se compare el mismo número de meses. Algo usual es comparar el mismo periodo de lo programado contra lo real.

Una vez depurada la lista de clases el siguiente paso es definir las asociaciones entre ellas, tomando como base tanto la especificación del problema como el diccionario de clases. Las asociaciones encontradas para nuestro problema son:

Un grupo petroquímico está formado de centros,
Un centro contiene plantas,
Una planta procesa materias primas,
Una planta utiliza Químicos y Catalizadores,
Una planta funciona con servicios auxiliares,
Una planta obtiene productos,
Una planta genera costos de operación,
Un estado financiero evalúa el rendimiento de una planta,
Un estado financiero evalúa el rendimiento de un centro,
Un estado financiero evalúa el rendimiento de un grupo petroquímico,
Un escenario agrupa montos de materias primas, de servicios auxiliares, de productos y de gastos de operación.
Una desviación compara escenarios.

Al revisar la lista de asociaciones se encontró que un grupo petroquímico es un conjunto de centros y a su vez un centro es un conjunto de plantas, por tanto las asociaciones:

Un estado financiero evalúa el rendimiento de una planta,
Un estado financiero evalúa el rendimiento de un centro, y
Un estado financiero evalúa el rendimiento de un grupo petroquímico;

pueden sustituirse por una sola asociación:

Un estado financiero evalúa el rendimiento de un grupo de plantas,

donde el grupo de plantas puede contener una sola, las plantas de un centro o bien todas las del grupo petroquímico. Analizando cuidadosamente esta asociación se notó que no es binaria, ya que implícitamente están involucradas las clases *Materias Primas, Servicios Auxiliares, Productos* y *Costos de operación*, pues en conjunto contienen la información necesaria para calcular el rendimiento. De acuerdo con las recomendaciones de James Rumbaugh, se deben tener asociaciones binarias en la medida de lo posible, por lo que el diseñador decide descomponer esta asociación en las siguientes:

Un estado financiero resume los gastos por materias primas de un grupo de plantas,
Un estado financiero resume los gastos por servicios de un grupo de plantas,
Un estado financiero resume los gastos de operación de un grupo de plantas, y
Un estado financiero resume los ingresos por ventas de un grupo de plantas;

que aunque son ternarias, se pueden convertir en binarias calificándolas.

Otra observación es que la asociación

Un escenario agrupa montos de materias primas, de servicios auxiliares, de productos y de gastos de operación

también debe eliminarse debido a que en realidad es una construcción de implementación.

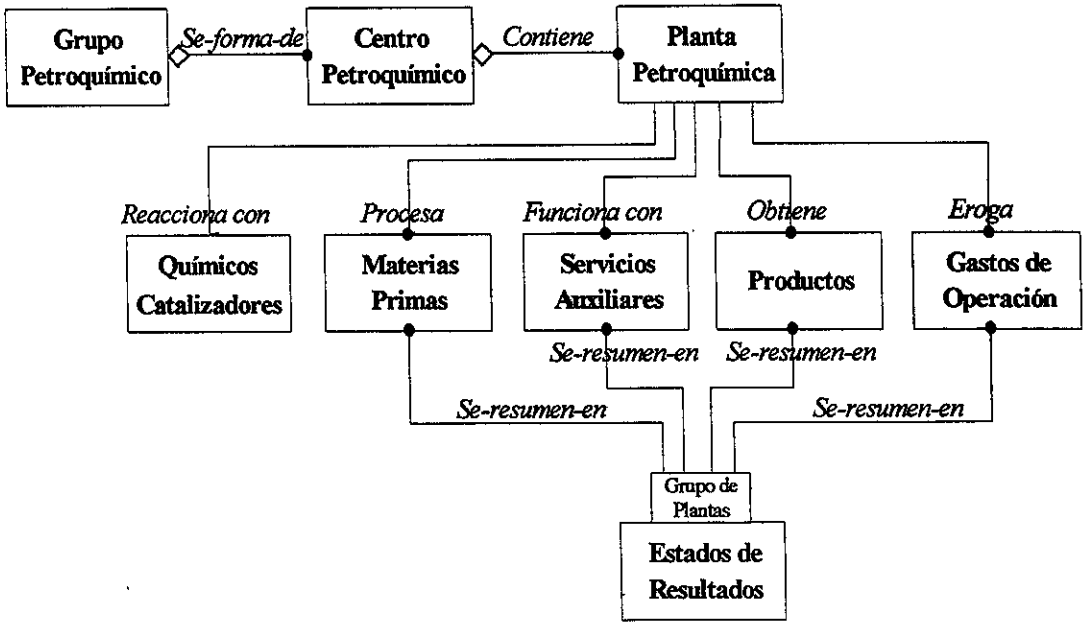
Finalmente, la asociación

Una desviación compara escenarios

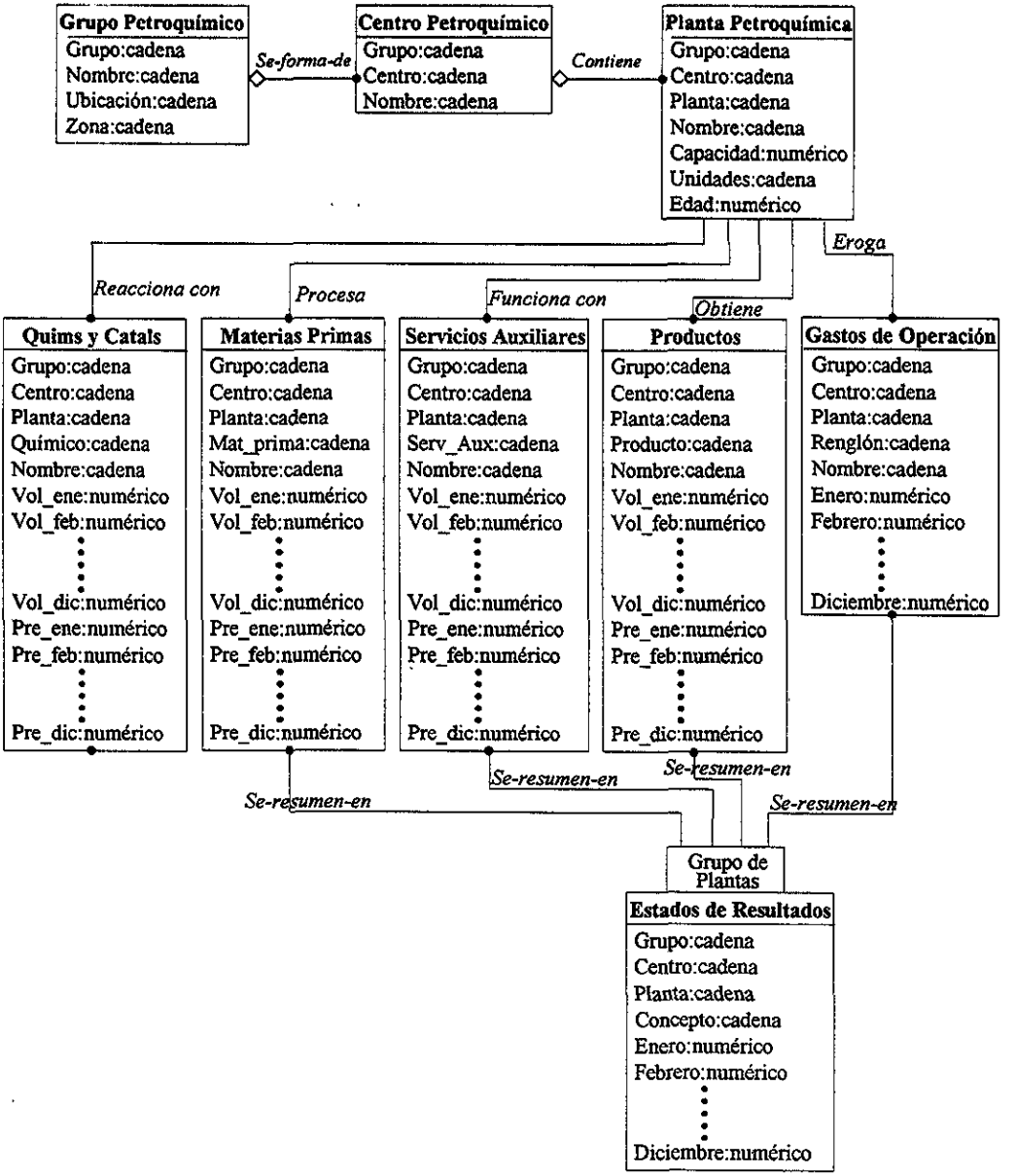
es una relación temporal ya que aunque dos escenarios pueden existir permanentemente, la comparación se da sólo en el momento que el usuario requiera verificar diferencias entre ellos, en consecuencia se decide también descartarla. De esta manera las asociaciones resultantes son todas ellas binarias.

4.3 Modelo de objetos

Una vez identificadas las clases involucradas en el problema así como sus asociaciones, se está en condiciones de construir el diagrama de objetos inicial para someterlo a un proceso de depuración hasta llegar al que servirá como base para el Modelo Dinámico.

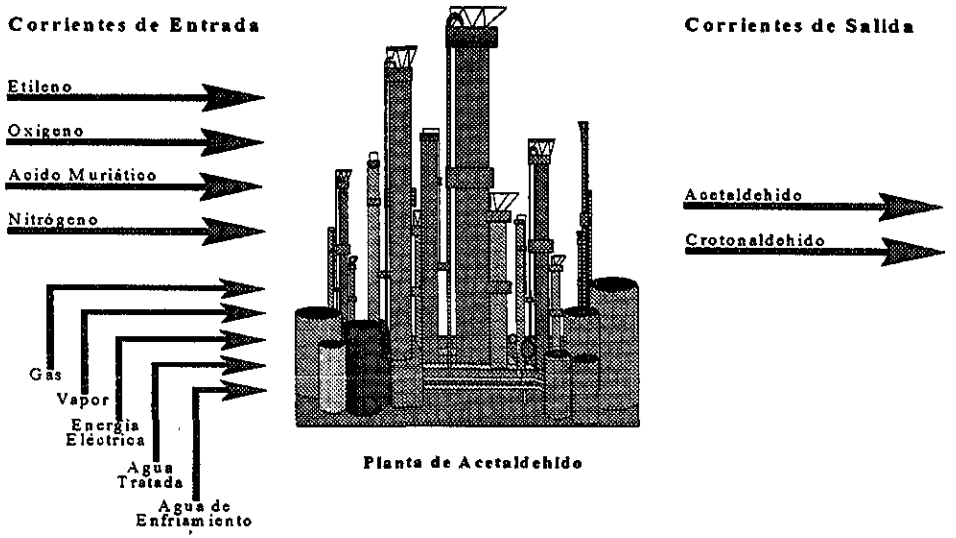


El siguiente punto es definir los atributos de cada una de las clases. El diagrama anterior se vería de la siguiente manera ya con atributos:

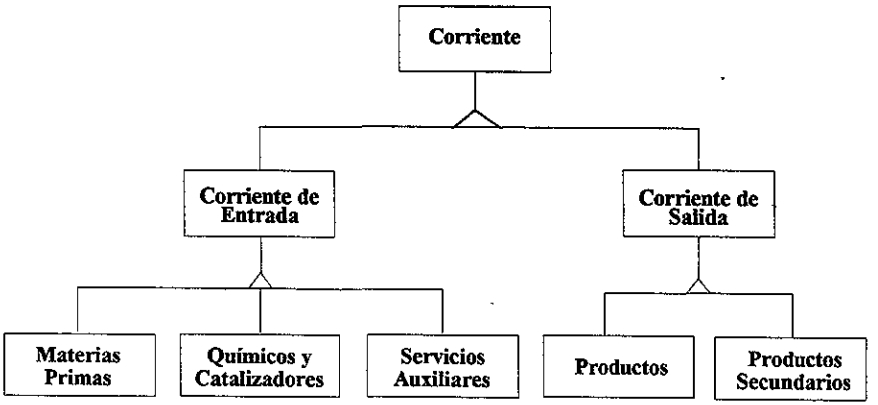


Al analizar los atributos definidos en las clases para organizarlas usando herencia, se notó que *Químicos y Catalizadores, Materias Primas, Servicios Auxiliares* y *Productos* contienen exactamente los mismos atributos, por lo que se hizo una generalización de ellas, surgiendo las superclases *Corrientes de Entrada* y *Corrientes de Salida*, pero estas a su vez

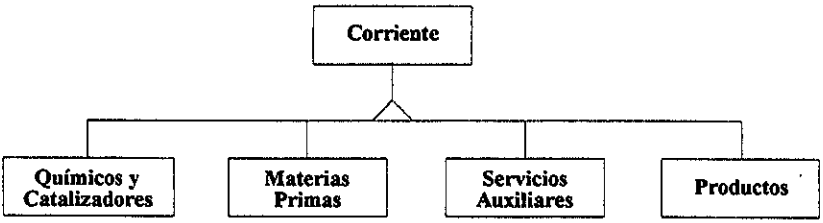
se generalizan mediante la clase *Corriente*. Bajo este esquema *Materias Primas* y *Servicios Auxiliares* son *Corrientes de Entrada* y los *Productos* son *Corrientes de Salida*.



Esta parte del diagrama de objetos se vería de la siguiente manera:

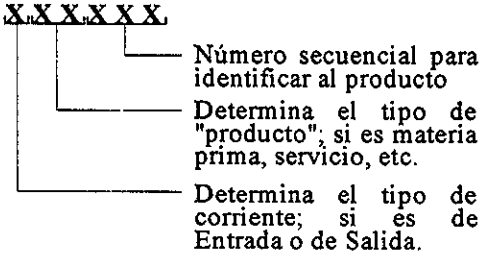


Al revisar este diagrama se encontró que *Corrientes de Entrada* y *Corrientes de Salida* son clases abstractas, es decir, no tienen objetos propios, sus instancias son sólo las que hereda de *Materias Primas, Servicios Auxiliares* y *Productos*. Debido a que no son clases determinantes para la organización de la aplicación se decidió eliminarlas, manteniendo sólo la clase *Corrientes*.

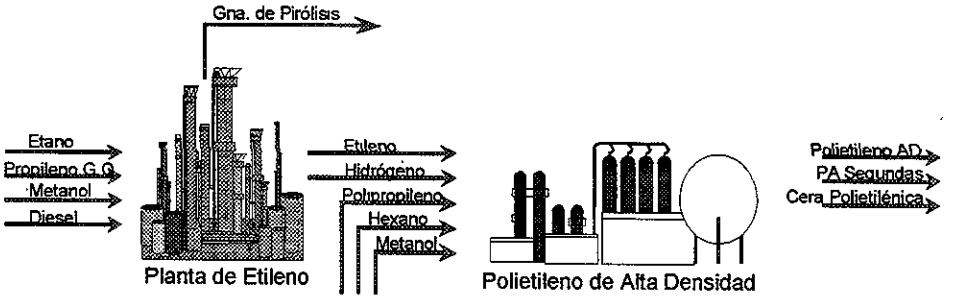


Al profundizar un poco más se observa que *Corriente* también es una clase abstracta y que *Materias Primas, Químicos y Catalizadores, Servicios Auxiliares* y *Productos* tienen exactamente los mismos atributos, por lo que se decide eliminar estas últimas y conservar únicamente *Corrientes*. Esto es posible anexando un atributo que determine el tipo de corriente, si es materia prima, químicos, servicio auxiliar, producto principal o producto secundario. Dicho de otra forma, estas clases pasan a ser atributos.

El atributo PROD_ID puede fungir como identificador de la corriente y al mismo tiempo determinar el tipo, siempre y cuando se estructure apropiadamente, por ejemplo:



Al revisar las corrientes de cada planta se notó que un "producto" puede requerirse por varias plantas y además en una puede ser de entrada y en otra de salida, por ejemplo:



Las plantas de Etileno generan como producto principal la corriente de salida "Etileno", mientras las plantas de Polietileno tienen entre sus materias primas la corriente de entrada "Etileno", por lo que de acuerdo a como se definió la clase *Corrientes* se necesitaría un registro por cada planta de Etileno y de Polietileno para definir la corriente etileno. Para evitar redundancias el diseñador recomienda redefinir *Corrientes* y crear otra clase que

administre los volúmenes tanto de consumos y compras de materias primas, químicos y catalizadores, y servicios auxiliares, como de las producciones y ventas. A esta nueva clase se le llamará *Balance de Materiales*, la cual estará relacionada con *Corrientes*, *Almacén*, *Plantas* y *Estados de Resultados*.

De aquí se detecta otro concepto interesante para el usuario: *Cadenas Productivas*, lo cual es una serie de plantas relacionadas entre sí, porque las corrientes de entrada de una son las de salida de otra y así sucesivamente; cada cadena tiene una planta inicial y otra final. Así, para evaluar la productividad de una cadena, todas las corrientes de entrada externas se valuarán a un precio de mercado y todas las internas, es decir, las que se traspasan entre las plantas de la cadena, se valuarán a costo.

De esta manera se cuenta con una herramienta para determinar si un producto puede ser competitivo a un precio de oportunidad, y por otra parte, dado que se tiene la facilidad de evaluar la rentabilidad de una planta en forma independiente a precio de mercado, permite detectar puntos débiles en una cadena productiva.

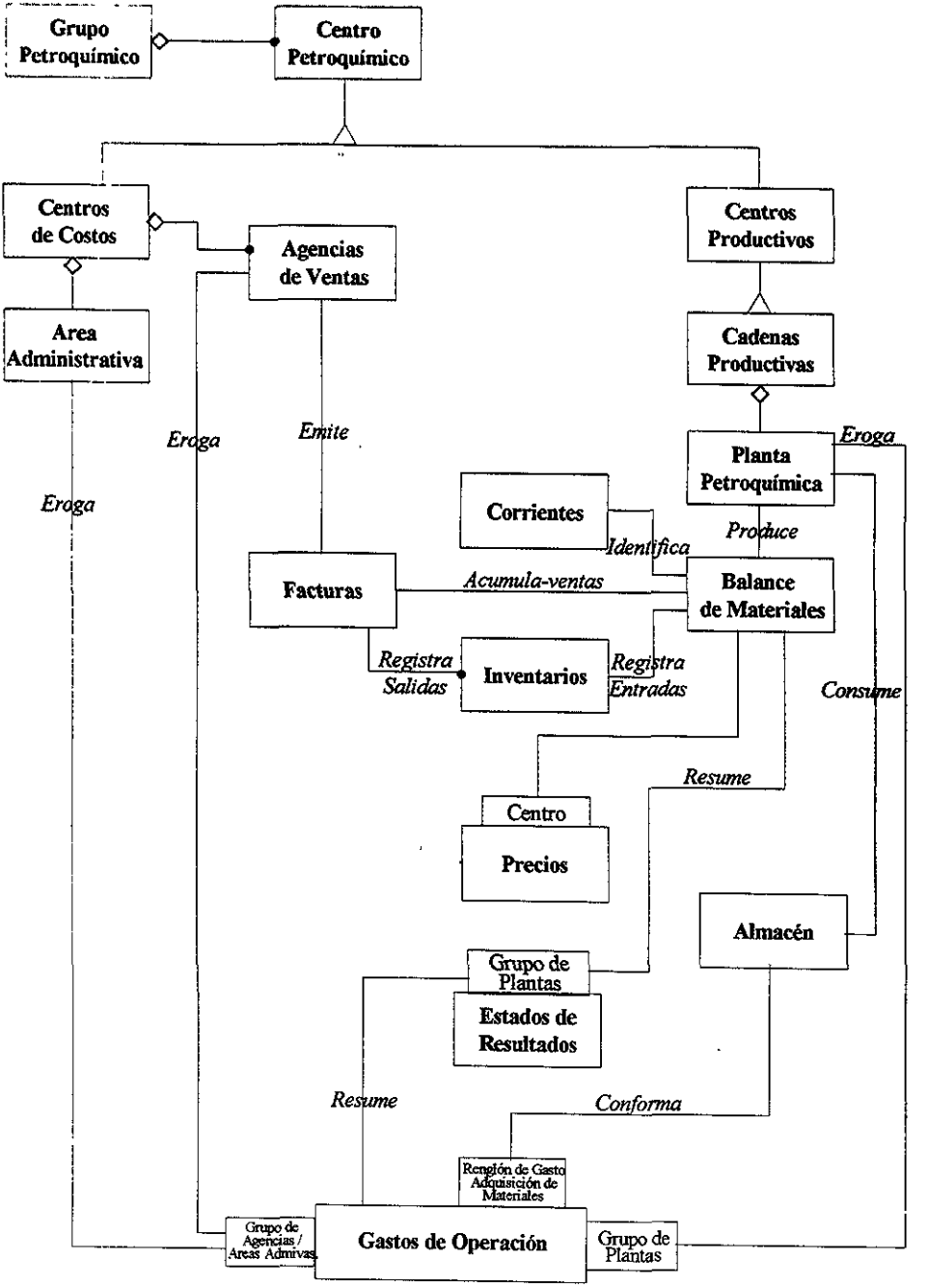
El hecho de que *Cadenas Productivas* sean agrupaciones de plantas deja ver que es una subclase de *Centros Petroquímicos*.

Continuando la investigación con los expertos en la aplicación se encontró que existen dos tipos de centros: Productivos y de Costos. Los centros productivos se forman mediante un grupo de plantas y los de costos con agencias de ventas, organizaciones que comercializan los petroquímicos obtenidos de los diferentes centros productivos, y con áreas netamente administrativas.

Para llevar a cabo la comercialización de los petroquímicos las agencias de ventas emiten facturas para controlar tanto la procedencia de los productos, es decir la planta y centro que los producen, como los volúmenes vendidos, el precio y el monto de la venta. Los precios se definen por centro y no por corriente, por lo que también para evitar redundancias es recomendable manejarlos como una clase y no como un atributo, pues así en lugar de requerir un precio por Producto, se define uno por cada Centro-Producto.

Las áreas administrativas se encargan de la organización de las plantas, agencias de ventas y centros petroquímicos para su buen funcionamiento. Tanto las agencias de ventas como las áreas administrativas requieren de una serie de costos fijos para su correcta operación, los cuales vendrían a ser las veces de *Costos de Operación*.

Por lo anterior, *Centros Petroquímicos* se convierte en la superclase de las subclases *Centros Productivos* y *Centros de Costos*, y se definen las clases de *Agencias de Ventas*, *Areas Administrativas*, *Precios* y *Facturas*. Al definir nuevas clases también se crean nuevas relaciones, las cuales se muestran en el siguiente diagrama:


Grupo Petroquímico
Centro Petroquímico
Centros de Costos
Agencias de Ventas
Centros Productivos
Area Administrativa
Cadenas Productivas
Eroga
Emite
Eroga
Eroga
Planta Petroquímica
Eroga
Corrientes
Identifica
Produce
Facturas
Acumula-ventas
Balance de Materiales
Registra Salidas
Inventarios
Registra Entradas
Consume
Resume
Centro
Precios
Almacén
Grupo de Plantas
Estados de Resultados
Resume
Conforma
Renglón de Gasto Adquisición de Materiales
Grupo de Agencias / Areas Admvas.
Gastos de Operación
Grupo de Plantas

El diccionario de las nuevas clases se presenta a continuación:

- *Centros Productivos.*- Son conjuntos de plantas petroquímicas ubicadas fisicamente en la misma área y organizadas bajo la misma administración. Estos centros generan tanto Ingresos (con las ventas de los productos) y Egresos (costos fijos y costos variables). Cada centro es responsable del buen funcionamiento de sus plantas.

- *Centros de Costos.*- Son conjuntos de agencias de ventas y/o áreas administrativas, es decir, los elementos de los centros de costos no son entes productivos, se concretan a la comercialización y a la organización, por lo tanto no generan ingresos. Cada centro de costos es responsable del buen funcionamiento de sus elementos.

- *Cadenas Productivas.*- Son agrupaciones de plantas organizadas mediante un tren productivo, de tal forma que las corrientes de salida de una son las corrientes de entrada de la siguiente en la fila, contando con una planta inicial y otra final. Para algunos fines una cadena productiva se considera como una caja negra, tomando en cuenta únicamente las corrientes de entrada y de salida externas, mientras que para otros debe considerarse todo el detalle. Todas las plantas de una cadena pertenecen a un solo centro productivo. Existen algunos casos en los que una planta pueda pertenecer a más de una cadena productiva.

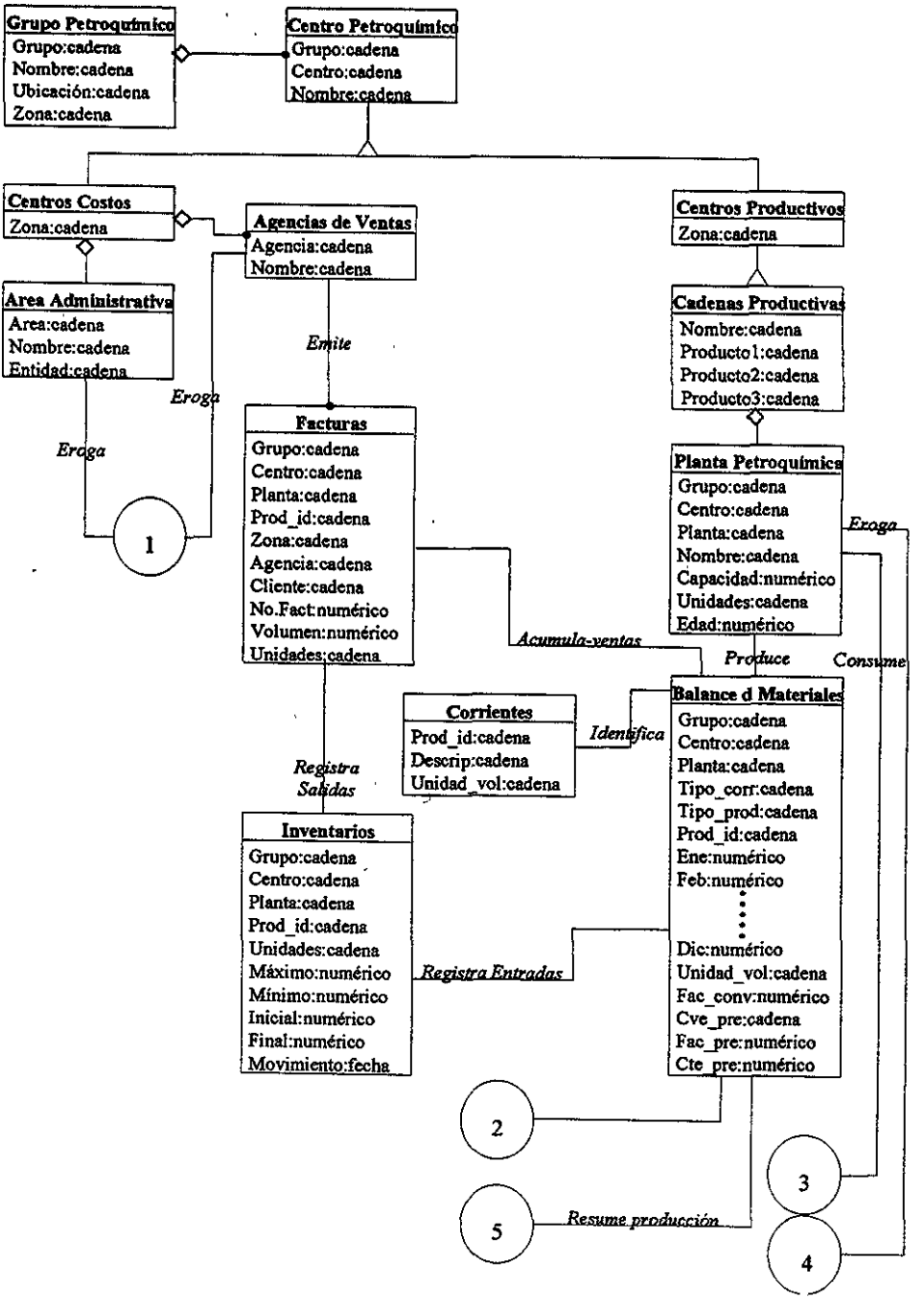
- *Agencias de Ventas.*- Son las organizaciones a través de las cuales se realizan la comercialización de los diferentes petroquímicos. Las agencias de ventas emiten una factura por venta hecha, registrando procedencia y destino del producto, esto es Centro-Planta que lo produjo y el cliente, además volumen, precio, monto y si es nacional o internacional.

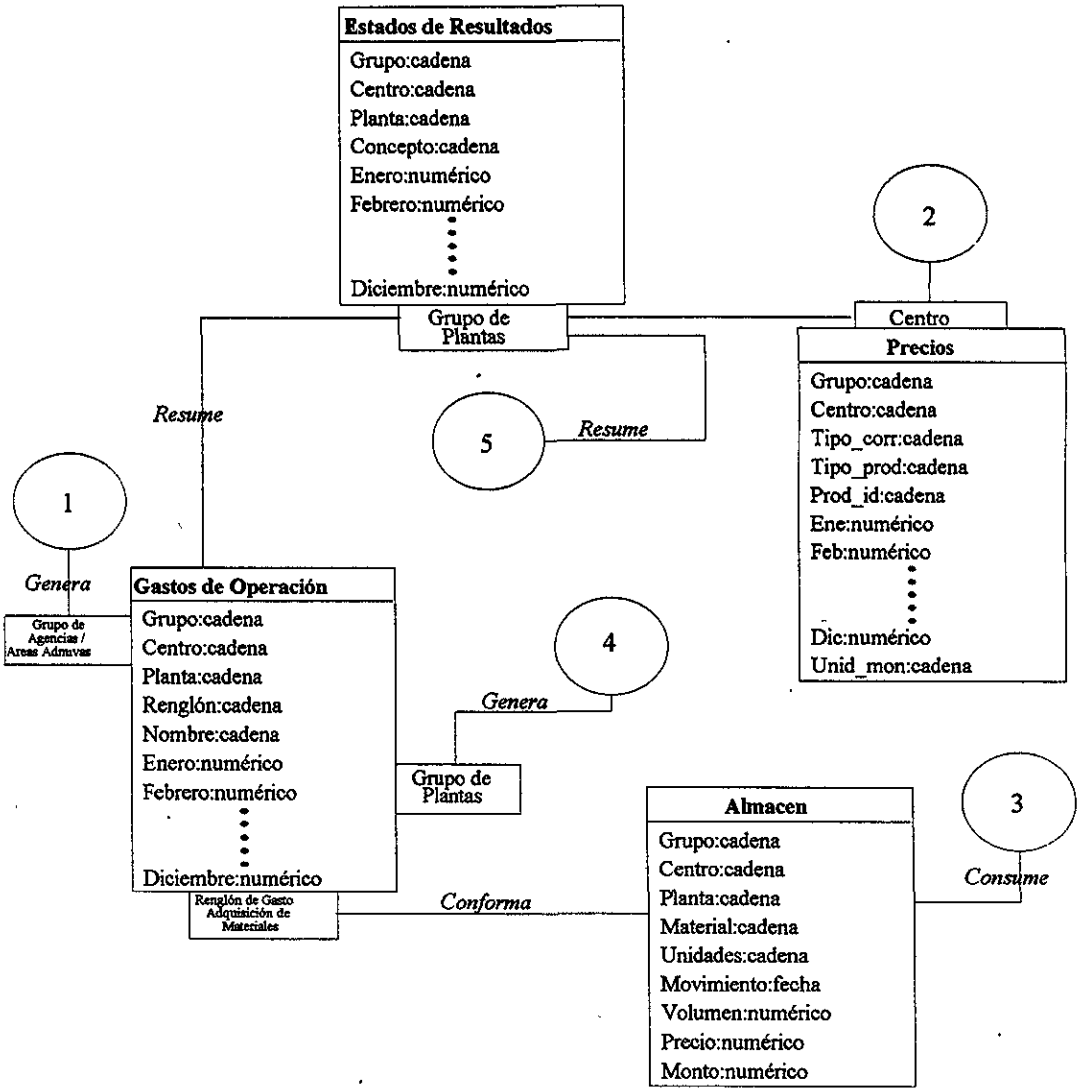
- *Areas administrativas.*- Son organizaciones que se encargan de supervisar el buen funcionamiento tanto de las plantas productivas como de las agencias de ventas, por ejemplo, son quienes definen estrategias de mercado para determinar precios y productividad de algún petroquímico para ser competitivo a nivel nacional e internacional, son quienes realizan la emisión de la nómina y controlan el manejo del personal en general, etc.

- *Facturas.*- Son los documentos que emiten las agencias de ventas a sus clientes para realizar una transacción, pero que también permiten llevar un control de los volúmenes y montos vendidos de cada producto por planta y centro, así como una bitácora de precios por cliente (descuentos y precios especiales).

- *Precios.*- Son los valores por unidad de volumen al que se realiza una venta, asociados cada uno de los productos por centro productivo. El modelo para calcular el precio de un producto se forma de un precio base multiplicado por un factor y adicionándole una constante; tanto el factor como la constante son únicos para cada centro productivo por producto, los descuentos y precios especiales se definen en la factura.

Los atributos de estas nuevas clases se presentan a continuación:



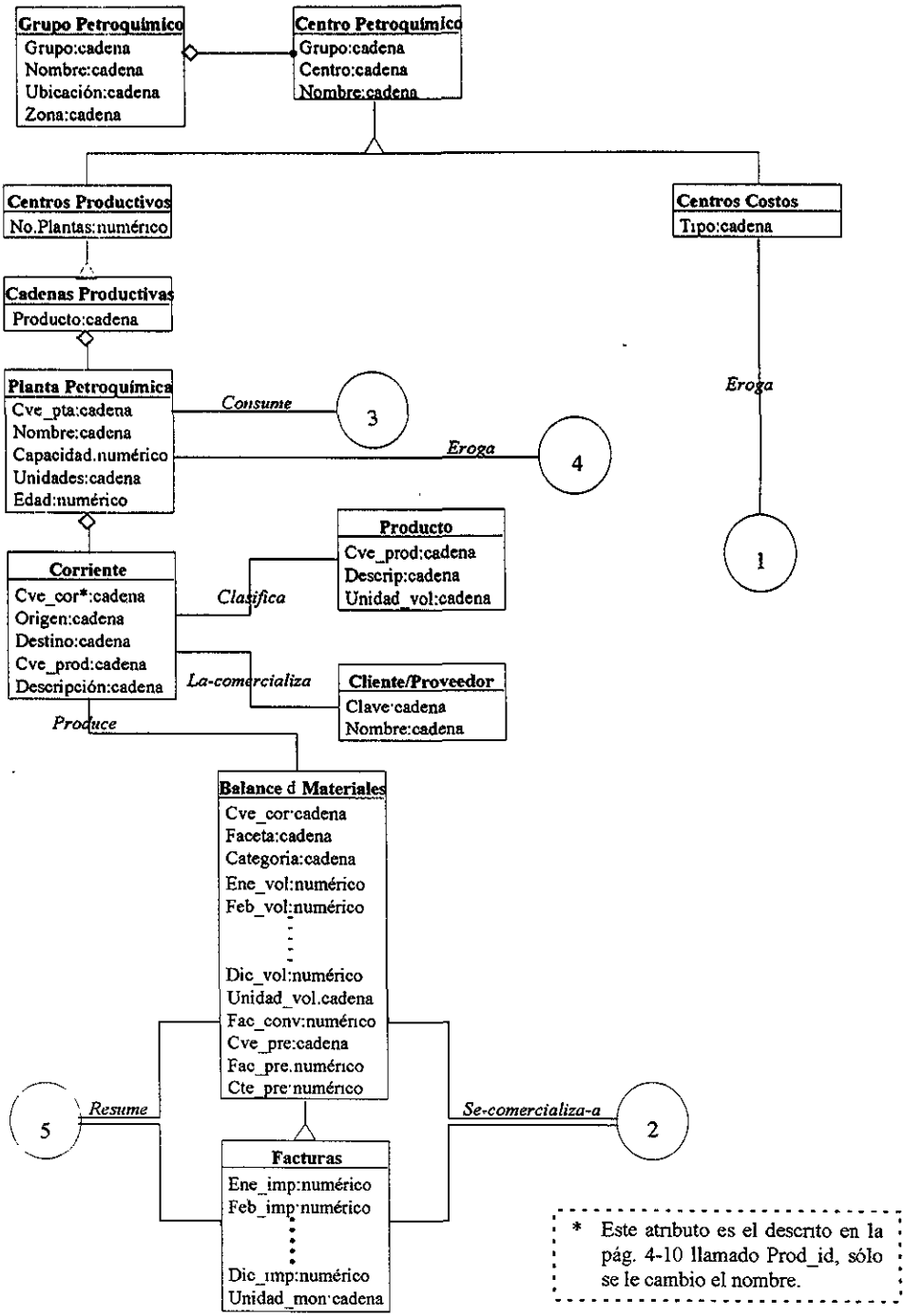

Estados de Resultados
Grupo:cadena
Centro:cadena
Planta:cadena
Concepto:cadena
Enero:numérico
Febrero:numérico
Diciembre:numérico
Grupo de Plantas
2
Centro
Precios
Grupo:cadena
Centro:cadena
Tipo_corr:cadena
Tipo_prod:cadena
Prod_id:cadena
Ene:numérico
Feb:numérico
Dic:numérico
Unid_mon:cadena
Resume
5
Resume
1
Genera
Grupo de Agencias / Areas Admvas
Gastos de Operación
Grupo:cadena
Centro:cadena
Planta:cadena
Renglón:cadena
Nombre:cadena
Enero:numérico
Febrero:numérico
Diciembre:numérico
4
Genera
Grupo de Plantas
Renglón de Gasto Adquisición de Materiales
Conforma
Almacen
Grupo:cadena
Centro:cadena
Planta:cadena
Material:cadena
Unidades:cadena
Movimiento:fecha
Volumen:numérico
Precio:numérico
Monto:numérico
3
Consume

Revisando una vez más con el usuario las asociaciones y los atributos de las nuevas clases, y sin perder de vista que se desea obtener un sistema gerencial, se llegó a que no es importante para los fines de este desarrollo tener el detalle del manejo de inventarios, así como de las facturas y agencias que comercializan cada producto; con llevar un registro mensual de las ventas que se realizan es suficiente. Al usuario sólo le interesa saber el volumen e importe mensual de los diferentes tipos de ventas que el producto de una planta pueda tener y la variación de inventario.

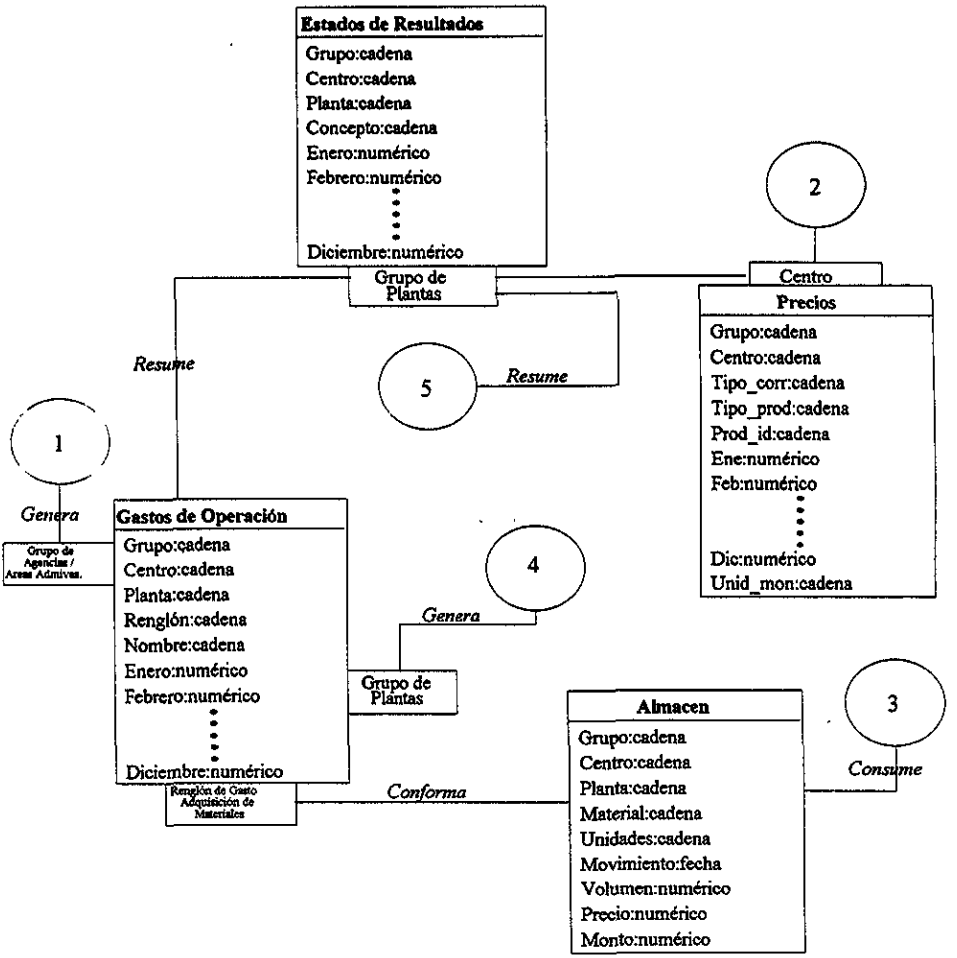
En esta revisión también se pudo detectar que es importante tener la información, de compras y ventas por un lado, y de consumos y producciones por otro. Los datos requeridos de compras y ventas son el origen y destino, es decir quien compra y quien vende, además de los volúmenes e importes mensuales. Las ventas pueden realizarse ya sea mediante una agencia de venta o bien directamente por el centro productivo; para el usuario es importante distinguir una de otra.

A través de la variación de inventarios y pérdidas se hará la confrontación entre compras y consumos, y producciones y ventas.

De acuerdo a estos comentarios la clase de *Agencias de Ventas* y la de *Inventarios* se eliminan, la de *Facturas* y la de *Corrientes* se modifican y se crea la clase de *Clientes y Proveedores*. Con esto también surgen cambios, en las relaciones entre las clases, llegando así al siguiente diagrama de objetos:



* Este atributo es el descrito en la pág. 4-10 llamado Prod_id, sólo se le cambio el nombre.



4.4 Modelo Dinámico.

Partiendo de la especificación del problema y de la estructura estática representada en el diagrama de objetos anterior se construirá un diagrama de estado, analizando los cambios que cada objeto pueda tener a través del tiempo. El primer paso es preparar diálogos típicos entre usuario y sistema a través de escenarios.

4.4.1 Escenarios normales.

A continuación se presenta el escenario normal de acceso al sistema:

El sistema solicita clave de acceso, el usuario la digita,
El sistema verifica y acepta la clave,
El sistema solicita versión y periodo con el que se trabajará,
El usuario selecciona la versión y digita el periodo,
El sistema presenta las operaciones disponibles,
El usuario selecciona una opción.

Identificar más diálogos del sistema con el exterior no se hace fácil para esta aplicación, por lo que se solicita a diferentes tipos de usuarios definan: 1). Qué pretenden obtener de este sistema; y 2). Qué tareas piensan desarrollar con él.

La información obtenida se clasificó de acuerdo al objeto que afecta de la siguiente manera:

Estado de Resultados.
- Saber si una planta genera ganancias o pérdidas en un periodo determinado evaluándola a diferentes precios, costos de operación y volúmenes.
- Saber si una cadena productiva genera ganancias o pérdidas en un periodo determinado evaluándola a diferentes precios, costos de operación y volúmenes.
- Saber si un centro productivo genera ganancias o pérdidas en un periodo determinado evaluándolo a diferentes precios, costos de operación y volúmenes.
- Saber si un grupo petroquímico genera ganancias o pérdidas en un periodo determinado evaluándolo a diferentes precios, costos de operación y volúmenes.
- Comparar la rentabilidad de una planta, una cadena y/o un centro para diferentes periodos o evaluados a diferentes precios, costos de operación y volúmenes.

Balance de Materiales.
- Registrar diferentes tipos de volúmenes de las materias primas y/o servicios que consume una planta en forma mensual.
- Registrar diferentes tipos de volúmenes de productos y/o servicios que genera una planta en forma mensual.
- Calcular cuanto consume mensualmente una cadena, un centro productivo o un grupo petroquímico de una materia prima o un servicio determinado.
- Calcular cuanto produce mensualmente una cadena, un centro productivo o un grupo petroquímico de un producto o servicio determinado.

Precios.
- Registrar diferentes tipos de precios promedio mensual para cada materia prima, producto y servicio, ya sea por planta, por cadena, por centro productivo o por grupo petroquímico.
- Calcular precios a partir de facturación y volúmenes.

Facturación
- Saber el gasto por cada materia prima y servicio de una planta, una cadena, un centro o de un grupo petroquímico en forma mensual.
- Conocer los ingresos de cada producto y servicio de una planta, una cadena, un centro o de un grupo petroquímico en forma mensual.

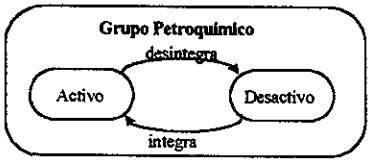
Gastos de Operación.
- Conocer cuales fueron los gastos asociados a la operación en forma mensual.

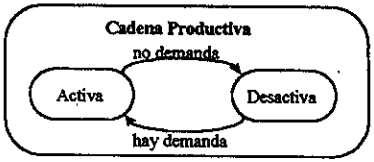
De las necesidades de los diferentes tipos de usuarios se observa que la interacción con el exterior es mínima, y debido a que la parte fundamental es administrar una base de datos para cubrir todos los requerimientos planteados, no existen diálogos típicos importantes.

El hecho de que la interacción con el exterior sea prácticamente nula facilita la creación de los diagramas de estado de cada objeto sin necesidad de tener los diálogos.

4.4.2 Diagramas de estado.

Siguiendo en orden el modelo de objetos resultantes, a continuación se presentan los diagramas de estado para los objetos Grupo Petroquímico y Cadena Productiva.





Los diagramas para Centro Petroquímico, Productivo y de Costos son exactamente igual al de Grupo Petroquímico.

Los de Planta, Balance de Materiales y Almacén están dados por el evento de Mantenimiento, ya sea preventivo o correctivo y por el comportamiento del mercado de los productos petroquímicos.

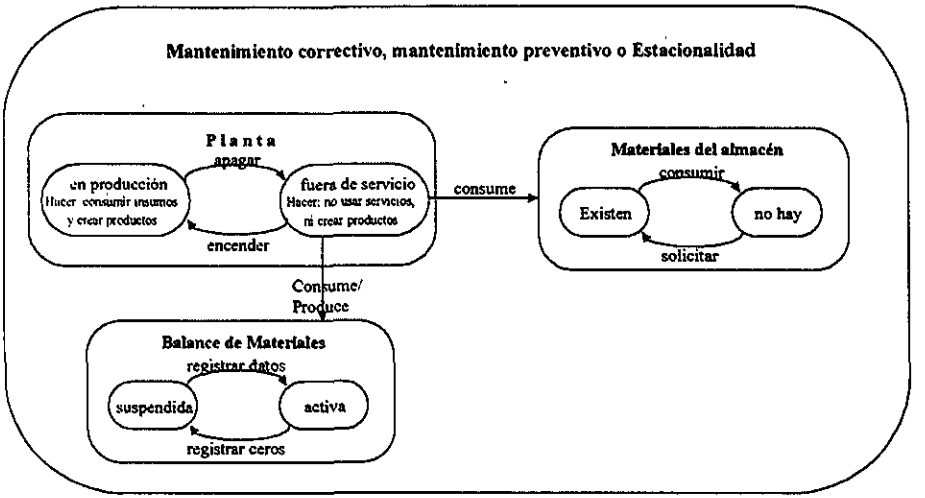
Existe un programa anual de mantenimiento preventivo que cada jefe de planta genera. Su periodicidad y duración depende de las características físicas de la planta y del comportamiento en el mercado del producto(s) que obtiene.

Cuando, a pesar del mantenimiento preventivo, una planta no funciona correctamente se debe parar y corregir el problema, dando lugar al mantenimiento correctivo. Esto es muy importante debido a que cualquier falla puede provocar accidentes fatales.

Por otro lado, el mercado internacional de petroquímicos es cíclico, hay temporadas en las que un producto es muy demandado y por tanto sus ganancias son altas, esto provoca que todas las plantas del mundo que lo producen se activen, lo que a su vez causa que el mercado se sature y las ganancias bajen hasta convertirse en pérdidas, antes de que esto suceda, los grupos petroquímicos van poniendo fuera de servicio estas plantas.

La demanda de un producto también es por temporal; por ejemplo, la demanda del amoniaco estará en función de la época de siembra de la región donde se comercialice debido a que se utiliza para producir fertilizantes.

Conjugando todos estos aspectos, el diagrama de estado para los objetos Planta, Balance de Materiales y Almacén se vería de la siguiente forma:



Al revisar detalles con el usuario sobre las cuestiones externas que determinan la forma de este diagrama, se encontró que el comportamiento de los objetos de Precios, Gastos de Operación y el mismo Estados de Resultados también resulta afectado por el mantenimiento y la estacionalidad.

Los precios están en función de la demanda y los volúmenes disponibles, es decir a una mayor demanda y/o menor disponibilidad de producto, un mayor precio. Algo muy parecido sucede con Gastos de Operación y con Estados de Resultados.

Al revisar el planteamiento del problema y discutirlo con el usuario para hacer un análisis más profundo se llegó a que para el objetivo del sistema no es determinante este comportamiento debido a que no se pretende llevar un control de los productos petroquímicos en el mercado nacional e internacional, ni llevar un seguimiento del mantenimiento de plantas. Tal vez si el objetivo fuera contar con un sistema para control de producción sí sería importante.

Así, para no desviarse del objetivo de nuestro trabajo, lo recomendable por la metodología James Rumbaugh es manejar este comportamiento como atributos de los objetos involucrados, lo cual ya está incluido con los atributos de enero a diciembre. A Planta se puede agregar el atributo estado para saber si está prendida o apagada; Almacén se puede controlar con el atributo de volumen y Balance de Materiales con los valores de los atributos de enero a diciembre: si están en ceros significa que la planta correspondiente está parada.

Lo mismo sucede con los objetos de Grupo Petroquímico, Cadena Productiva, Centro Petroquímico, Productivo y de Costos donde se puede agregar el atributo de *estado* cuyos valores posibles sean *activo* y *desactivo*.

Los objetos Producto y Cliente/Proveedor no tienen un comportamiento dinámico por tratarse de catálogos.

Por todo lo anterior, se llegó a la conclusión de que este sistema no tiene un modelo dinámico importante, por lo que se procederá a construir el modelo funcional.

4.5 Modelo Funcional.

Nuevamente basándose en la especificación del problema donde dice que hay que "calcular las utilidades o pérdidas a partir de los ingresos y erogaciones que generan la producción" se puede partir para la contrucción del modelo funcional, considerando de manera general lo siguiente:

$$\text{Utilidad/Pérdida} = \Sigma\text{Ingresos} - \Sigma\text{Egresos}.$$

Revisando el modelo de objetos junto con los usuarios, se tiene:

$$\text{Ingresos} = \text{Volumen productos y servicios vendidos} * \text{Precio}$$

Esto idealmente debería corresponder a las facturas a favor del Grupo Petroquímico, es decir:

$$\text{Ingresos}_1 = \text{Facturas de productos} + \text{Facturas de servicios},$$

Donde se debería cumplir,

$$\text{Ingresos} = \text{Ingresos}_1$$

Sin embargo, el usuario dice que normalmente esto no sucede y que es necesario conocer la diferencia entre los ingresos reales y los ingresos ideales, es decir se debe calcular:

$$\text{Diferencia} = \text{Ingresos}_1 - \text{Ingresos}$$

Los egresos se calcularían de la siguiente manera:

$$\text{Egresos} = \text{Volumen materias primas y servicios comprados} * \text{Precio} + \text{Costos de Operación}$$

Sucede lo mismo que con Ingresos, idealmente lo anterior debería ser igual a:

$$\text{Egresos}_1 = \text{Facturas de materias primas y servicios} + \text{Costos de Operación},$$

Pero como casi nunca coinciden, también se calcula:

Diferencia = $Egresos_1$ – Egresos

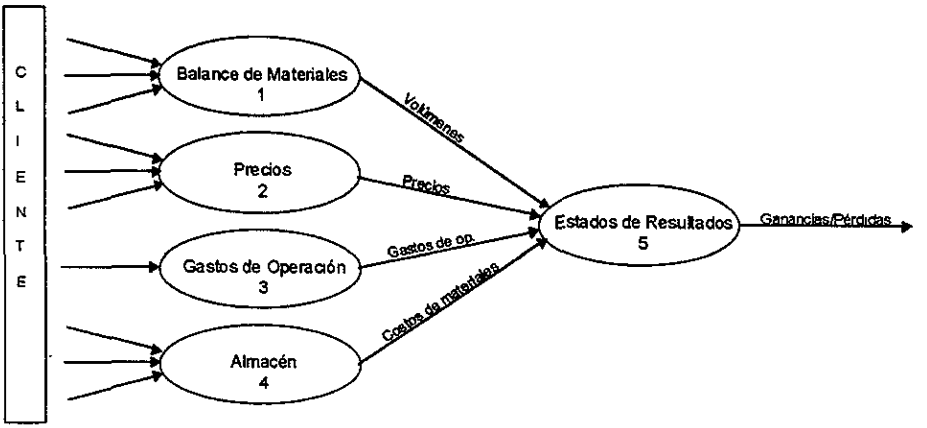
Con todo lo anterior se tiene la base para iniciar la construcción de los diagramas de flujo que conformen el modelo funcional.

4.5.1 Diagramas de Flujo.

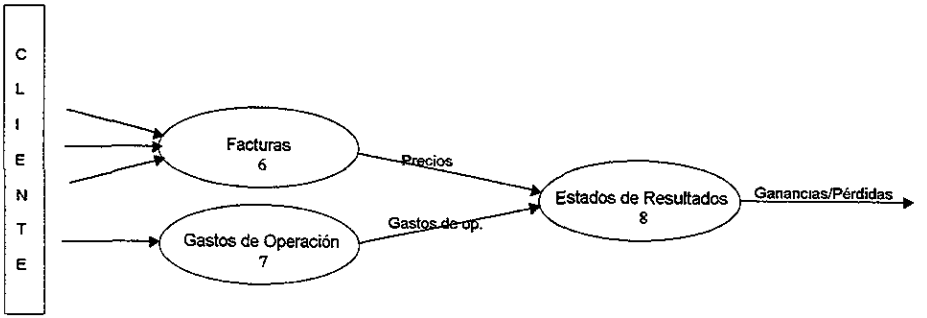
Se iniciará con un diagrama de flujo general para después construir los de nivel más bajo.

Sin embargo, al tratar de crear el diagrama de flujo de más alto nivel se encontró que se pueden tener varios de acuerdo a las siguientes necesidades planteadas por el usuario:

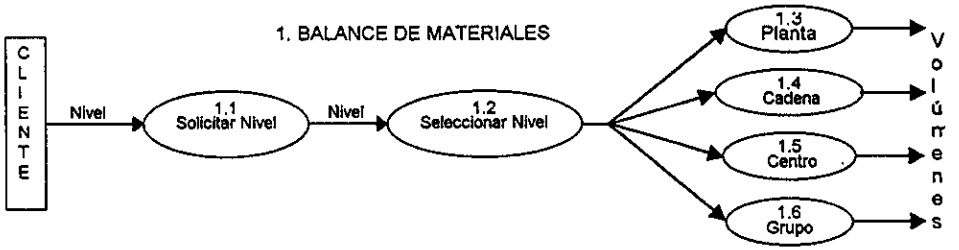
- Estados de resultados en base a la producción, no las ventas, cuyo diagrama se vería como sigue:



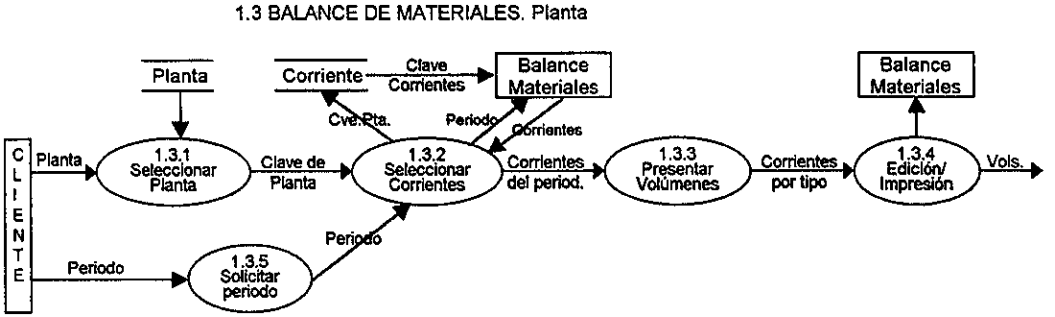
Estados de resultados en base a las ventas reales



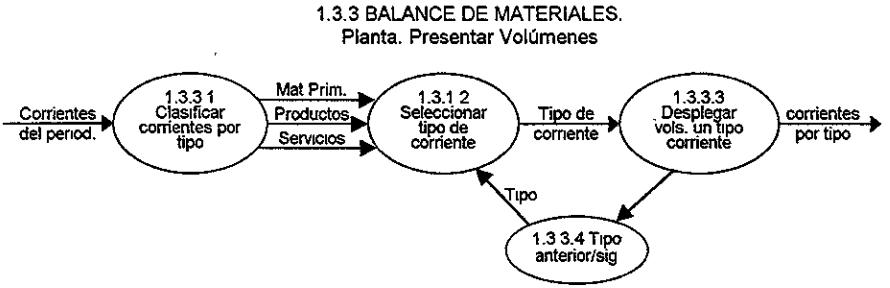
En primera instancia se desarrollarán los diagramas para calcular Estados de Resultados con base a la producción y consumos.



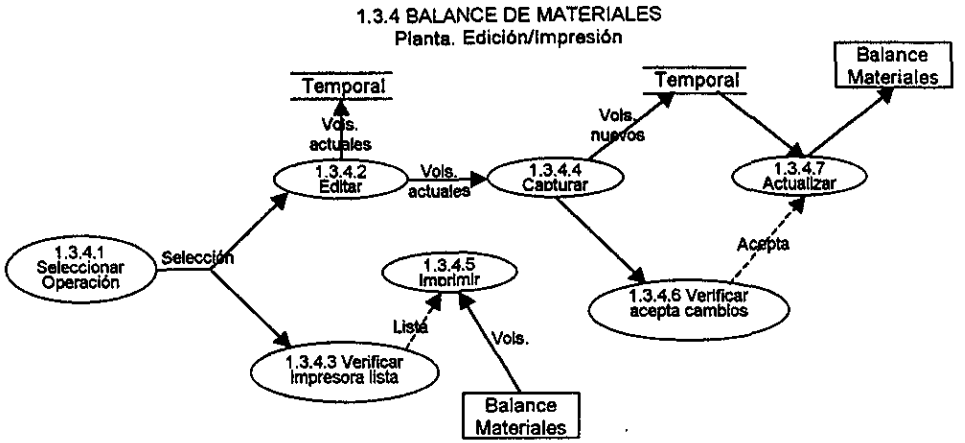
A continuación se presentan los diagramas referentes a planta



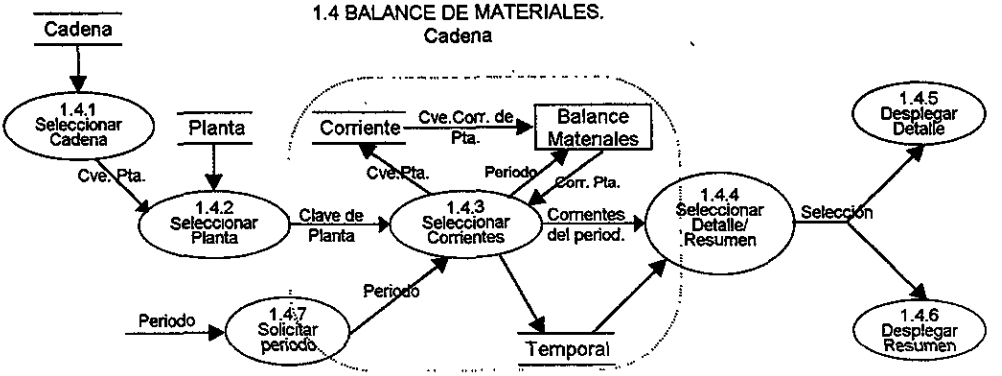
Dentro del diagrama de *Planta* se desarrollará el de *Presentar Volúmenes* presentado a continuación



Otro diagrama de *Planta* que se detallará es *Edición/Impresión*:

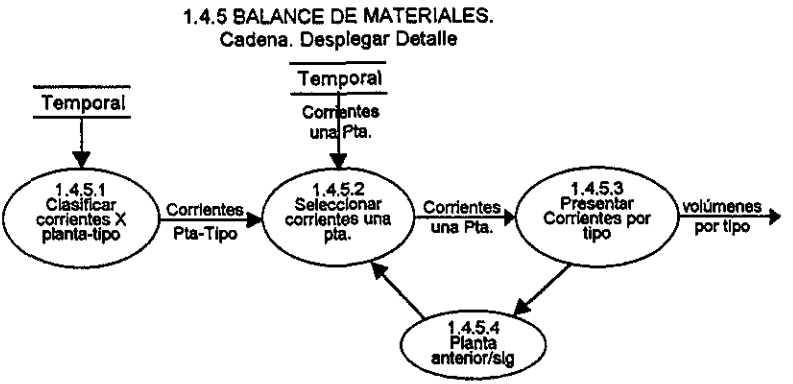


El siguiente punto es desarrollar el diagrama de flujo de *Cadena* de *Balance de Materiales*:



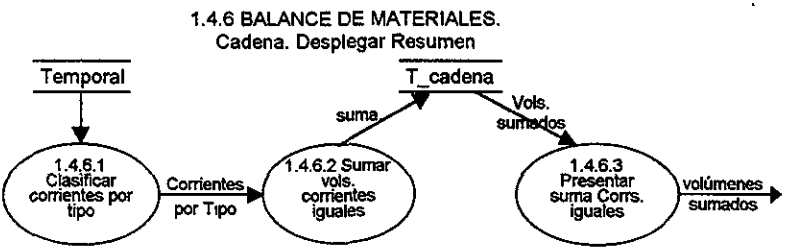
Lo encerrado en el cuadro punteado se repetirá para cada planta que componga la cadena productiva, con el objeto de registrar en el archivo *Temporal* todas las corrientes de las plantas de tal cadena.

Del diagrama de *Cadena* se desglosará el de *Desplegar Detalle*:

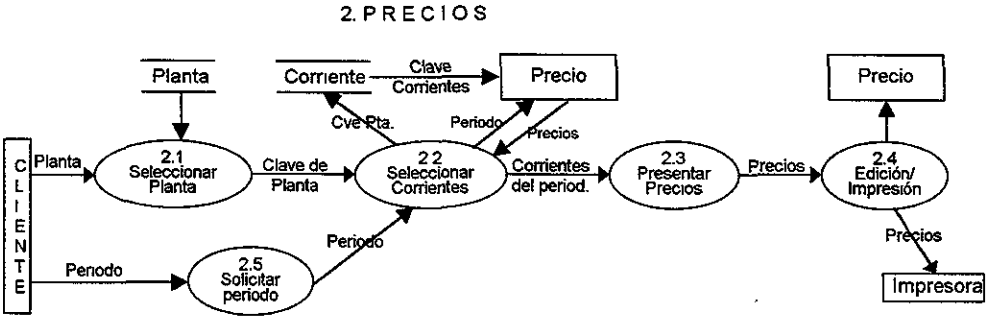


El desglose de *Presentar Corrientes por tipo* es igual al de *PLANTA. Presentar Volúmenes*, ya detallado.

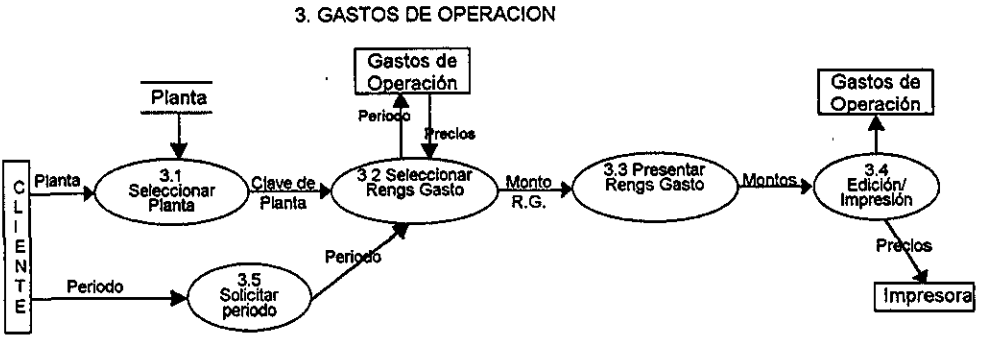
El siguiente diagrama es el de *Desplegar Resumen*:



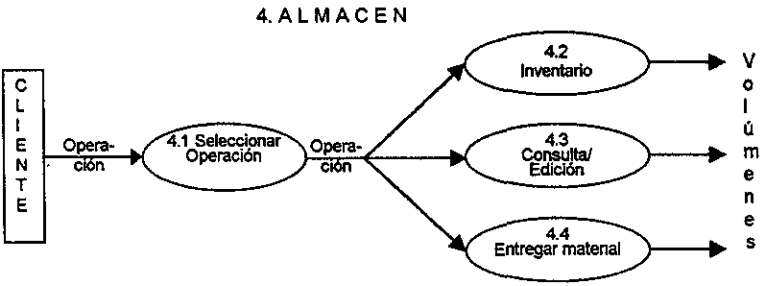
Los diagramas de *Centro* y *Grupo* son prácticamente iguales a los de *Cadena*, cambiando el proceso *Seleccionar Cadena* por *Seleccionar Centro* o *Seleccionar Grupo* respectivamente, y así agrupar todas las plantas del Centro o Grupo elegidos. Por tanto se procederá a presentar el diagrama correspondiente al módulo de *Precios* :



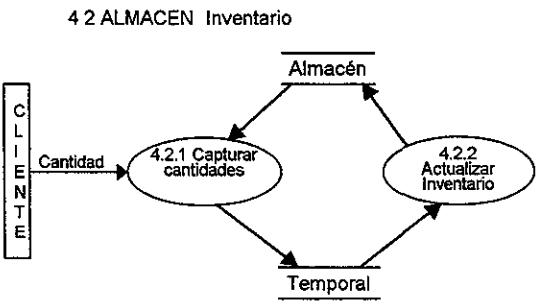
Los procesos de *Presentar Precios* y de *Edición/Impresión* son muy parecidos a los de *Balance de Materiales*, sólo sustituyendo el objeto de *Balance* por *Precio* y algún otro detalle. A continuación se presenta el diagrama de *Gastos de Operación*:



El siguiente diagrama es el de *Almacén*:

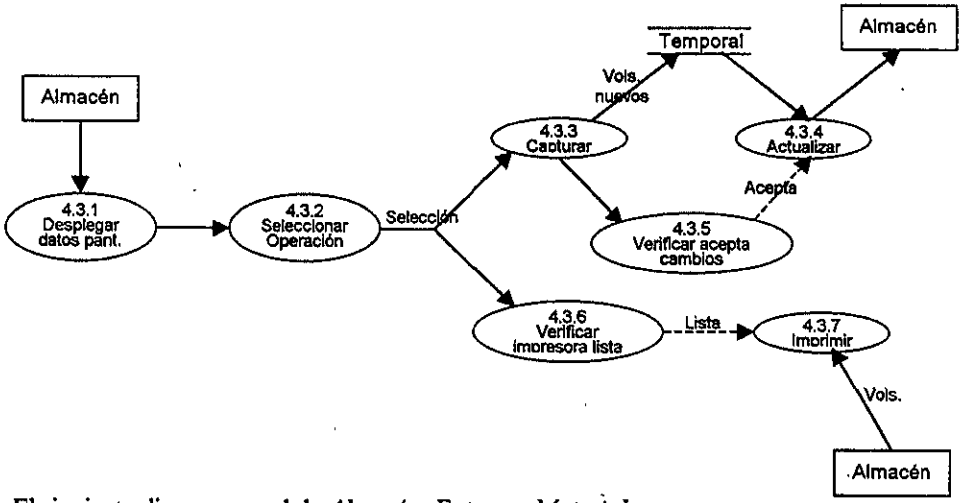


A continuación se detalla el diagrama *Almacén. Inventario*:



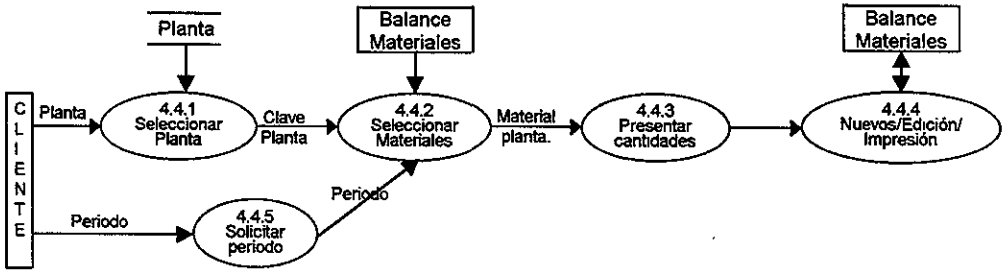
El diagrama de *Almacén. Consulta/Edición* se vería como sigue:

4.3 ALMACEN. Consulta/Edición



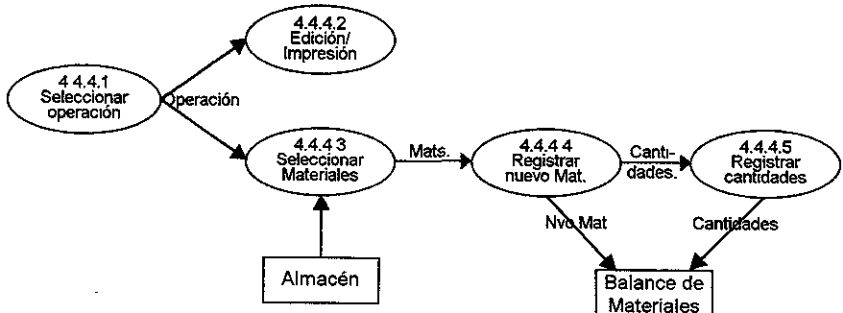
El siguiente diagrama es el de *Almacén. Entregar Material*:

4.4 ALMACEN. Entregar material



Para terminar con los diagramas de *Almacén* a continuación se presenta el de *Nuevo/Edición/Impresión*:

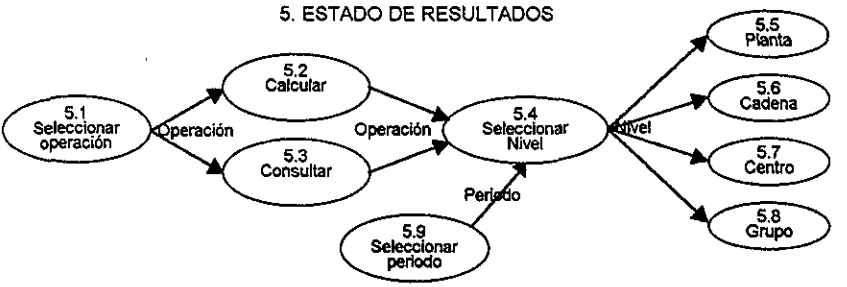
4.4.4 ALMACEN. Entregar Nuevo/Edición/Impresión



El diagrama de *Edición/Impresión* es igual al de *Almacén. Consulta/Edición.*

Con los procesos de los diagramas desarrollados hasta el momento se cuenta con la información necesaria para calcular Estado de Resultados con base a la producción y consumos, debido a que los volúmenes mensuales de las producciones y de los consumos de materias primas, servicios y materiales están almacenados en *Balance de Materiales*; los precios promedio mensuales en *Precios* y los montos de los renglones del gasto en *Gastos de Operación.*

Asi el diagrama de *Estado de Resultados* se vería de la siguiente manera:



Primero se desarrollará el diagrama de *Estado de Resultados. Calcular. Planta*, en el cual se obtienen por cada mes del periodo indicado los siguientes datos para cada una de las plantas:
Total por materias primas,
Total por ventas,
Total por servicios, y
Total de Gastos de Operaciones

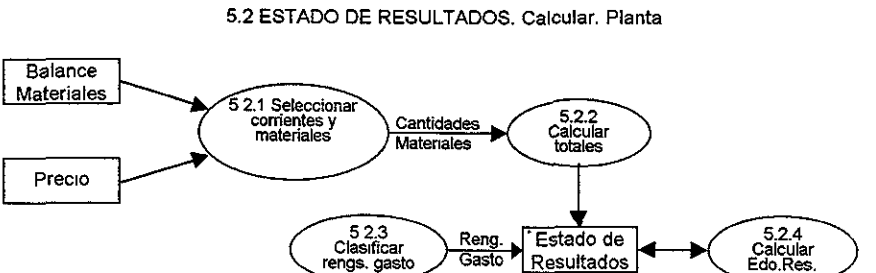
Con esto se calcula los Estado de Resultados

Estado de Resultados = Ventas – (Materias Primas+Servicios+Gastos de Operación)

Teniendo así la ganancia o pérdida mensual por planta, es decir, a través de esta herramienta se puede saber si una planta es o no rentable.

La suma de los totales de cada mes da como resultado la rentabilidad de la planta para el periodo.

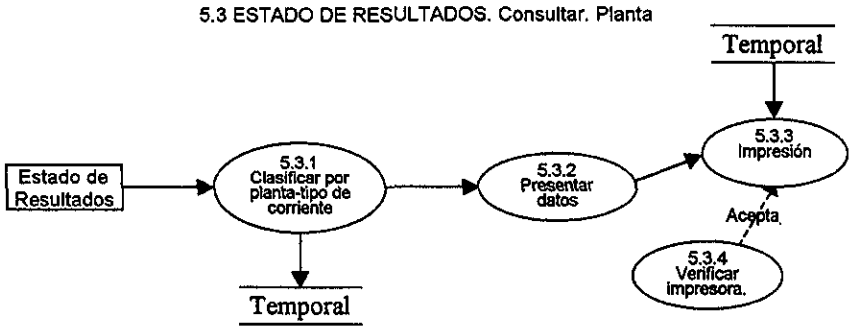
El diagrama de *Estado de Resultados. Calcular. Planta* se vería de la siguiente manera:



El diagrama de *Estado de Resultados. Calcular. Cadena* es igual al de *Planta*, sólo que en el proceso de *Seleccionar corrientes y materiales* agrupa corrientes y materiales por cadena y elimina corrientes que van de una planta a otra de la misma cadena, de tal forma que sólo quedan las corrientes externas a la cadena; también concentra en un solo registro todas las corrientes de una misma materia prima, servicio, material o producto.

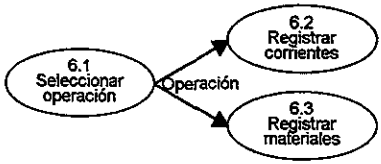
Los diagramas para Centro y Grupo son idénticos al de cadena sólo agrupando por Centro y Grupo respectivamente en lugar de por cadena.

Por tanto, se procederá a desarrollar los diagramas de *Consultar*:

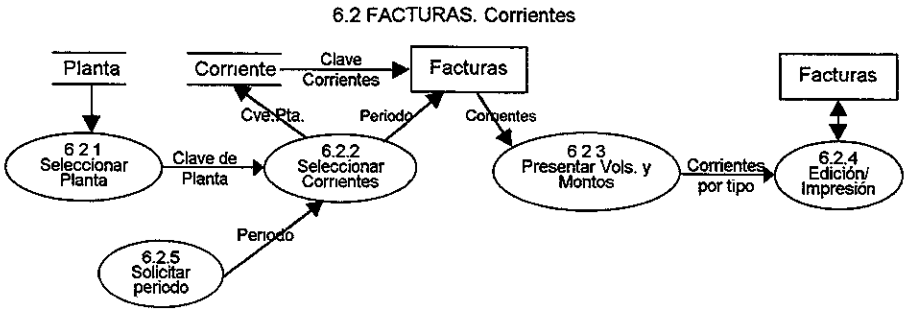


Al igual que en los diagramas de *Calcular*, los de *Consultar* para *Cadena, Centro* y *Grupo* son iguales a los de *Planta* sólo que los procesos de *Clasificar* y *Presentar* se hacen por cadena, centro y grupo respectivamente.

Con esto se terminan los diagramas de flujo para Estado de Resultados con base a la producción; por lo que se procederá a desarrollar los diagramas de Estados de resultados con base a las ventas reales. Se desglosará el de *Facturas*:

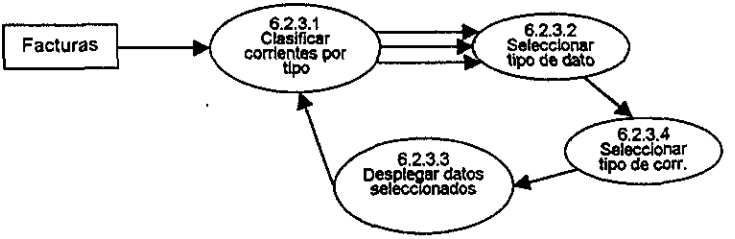


El diagrama de *Registrar corrientes* se ve como sigue:



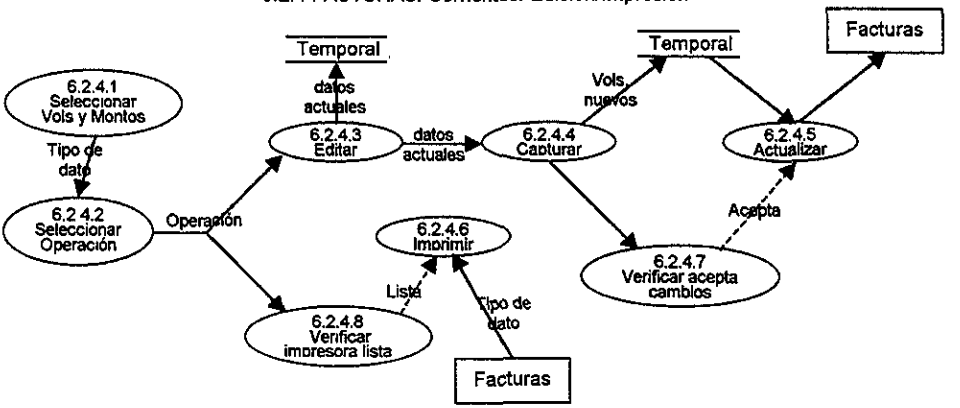
El diagrama que se desarrollará a continuación es el de *Presentar volúmenes y montos*:

6.2.3 FACTURAS. Corrientes. Presentar



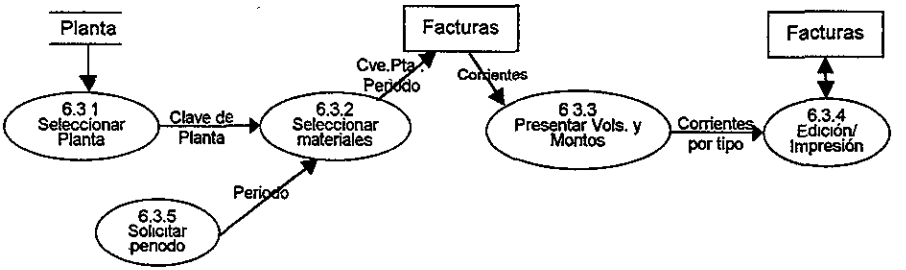
A continuación se desglosará el diagrama correspondiente a Edición/Impresión de Facturas:

6.2.4 FACTURAS. Corrientes. Edición/Impresión



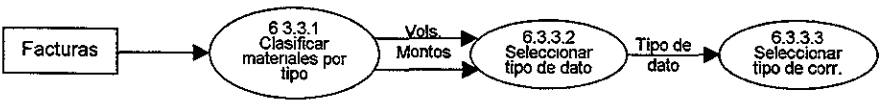
El diagrama de Facturas Materiales se presenta a continuación:

6.3 FACTURAS. Materiales



De este diagrama se desarrollará el de *Presentar volúmenes y montos*:

6.3.3 FACTURAS. Materiales Presentar



Con esto se terminaría el modelo funcional ya que los diagramas de Gasto de operación y los de Estado de Resultados serían prácticamente iguales a los presentados para estado de resultados con base a la producción.

Los resultados que se obtengan dependerán, como en cualquier sistema, de los datos que se alimenten. Así, en lugar de capturar producciones reales se podrían suministrar producciones programadas o producciones ideales para tener un rendimiento máximo de la planta, cadena, centro o grupo; o bien valuar las producciones reales a precios de exportación o a precios nacionales, etc., es decir se pueden manejar diferentes versiones de cada objeto: Balance de Materiales, Precio, etc.

De esta manera el sistema no sólo servirá para calcular Estado de Resultados, sino también como un simulador o una herramienta de seguimiento, ya que se podría comparar producciones reales contra las programadas y así detectar desviaciones y poder tomar acciones pertinentes.

El análisis aquí efectuado permite entender y describir todos los componentes de estado de resultados.

Este es un problema real y como tal no es factible resolverlo de manera ideal, se deben considerar todas las condiciones y políticas de la empresa, así como las excepciones, que para efectos de este trabajo harían un diseño e implantación muy complicados.

Sin embargo, cabe mencionar que tanto el diseño como la implantación de este sistema sí se desarrollaron y hasta la fecha sigue operando, no sólo como una herramienta para evaluar una planta, cadena o centro, sino como un simulador que permite hacer comparaciones entre versiones (datos reales, programados, etc.) y proyecciones bajo diferentes condiciones.

Conclusiones

El desarrollo orientado a objetos es un proceso ordenado y al mismo tiempo flexible que facilita la representación del mundo real. Además conduce a analistas y desarrolladores de sistemas a realizar un trabajo más limpio, sencillo de entender y documentado.

La técnica orientada a objetos combina dos aspectos importantes:

- La comunicación continua desde el inicio del proyecto hasta el final entre usuarios e informáticos, lo cual facilita que los analistas capten todos los detalles necesarios para resolver el problema planteado; y

- La representación gráfica de las diferentes fases del análisis, lo cual ayuda a desglosar el problema en partes más fáciles de entender y por tanto de detectar interacciones, dependencias y comportamiento común.

La combinación de estos dos aspectos reduce la probabilidad de terminar con un sistema que no cumpla con las espectativas planteadas, ya que los modelos representan el problema tal como lo concibe el desarrollador, teniendo así el usuario la oportunidad de detectar cualquier desviación a tiempo.

Utilizar una metodología orientada a objetos no es una tarea fácil al principio ya que requiere de una disciplina por parte de usuarios, analistas y desarrolladores, pero recordemos que "nunca hay tiempo para hacerlo bien, aunque siempre lo hay para volver a hacerlo".

Además implica toda una nueva manera de pensar respecto al desarrollo de una aplicación, ya que se debe enfocar más hacia el dominio del problema que en cómo se resolverá en la computadora.

Las metodologías orientadas a objetos requieren invertir más tiempo en el análisis y diseño, lo cual queda más que compensado en la implantación, el mantenimiento y por supuesto en el desempeño del sistema.

En lo que se refiere a la metodología OMT, en general proporciona una buena dirección. En el Modelo de Objetos maneja una notación fácil de entender que permite plasmar la realidad de una manera natural y clara, como se puede observar en el problema desarrollado en el capítulo 4 "La evaluación de plantas petroquímicas".

Del Modelo Dinámico no se puede decir lo mismo, pues aunque la metodología delimita perfectamente los alcances de este modelo, no proporciona los mecanismos suficientes para detectar fácilmente los escenarios normales y con error, para construir los diagramas de

estado a partir de ellos. De aquí que para el desarrollo de la aplicación planteada en el capítulo 4 se tuvo que buscar apoyo en los casos de uso del usuario.

Encontrando que para las aplicaciones donde el manejo de bases de datos es muy importante el Modelo Dinámico es prácticamente nulo pues las bases de datos sólo se crean, se modifican y se destruyen, es decir, no tienen un ciclo de vida interesante.

En cambio el Modelo Funcional sí juega un papel esencial, ya que para explotar adecuadamente una base de datos se necesita realizar una serie de operaciones.

Es importante la metodología pero más importante es entender perfectamente cada uno de los conceptos de la tecnología orientada a objetos, de esta manera analistas y desarrolladores serán capaces de librar los puntos débiles que pudiera tener la metodología utilizada.

Bibliografía

[1] Modelado y diseño orientado a objetos. Metodología OMT
Autor: James Rumbaugh
Michael Blaha
William Premerlani
Frederick Eddy
William Lorensen
Editorial: Prentice Hall

[2] Computer. ID- 0018-9162/92/1000-0022
Artículo: Object-oriented and conventional analysis and design methodologies
Autor: Robert G. Fichman & Chris F. Kemerer

[3] Communications of the ACM. Vol.33 No.9
Artículo: Understanding object-oriented: a unifying paradigm
Autor: Tim Korson & John D. McGregor
Artículo: Applying object-oriented analysis and design
Autor: Jean-Marc Nerson