UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

"ACATLAN"

TECNICA DE MODELADO DE OBJETOS PARA EL DESARROLLO DE SOFTWARE

T E S I N A

QUE PARA OBTENER EL TITULO DE:

LICENCIADO EN MATEMATICAS APLICADAS Y COMPUTACION

P R E S E N T A:

HECTOR SAUL ARELLANO ESCOBAR

ASESOR: **ING. RUBEN ROMERO RUIZ**

DICIEMBRE 2004

Esta Tesina se la dedico a mi esposa Lorena
por el gran amor, apoyo, paciencia y
comprensión que me ha tenido y a
mi hijo Saúl Ulises que es mi alegría,
orgullo y felicidad.

AGRADECIMIENTOS

Agradezco a Dios el apoyo que siempre recibí a través de mi Papá José Guadalupe, que me ha siempre me inculcó estudiar y prepararme para tener un mejor futuro.

A mi Mamá Josefina, que supo darme un buen regaño a tiempo, corrigió mi actitud, me educó con su ejemplo y cariño.

A mi hermana Yafani, por cuidarme desde el cielo. **Nunca te olvido y te llevo en mi corazón**.

A mi hermana Olga Lydia, que ha creído en mí y me ha inspirado confianza y fe.

A mi hermano José Luis que se ha preocupado por mi y siempre me ha dado su apoyo incondicional.

A la Sra. Paula y al Sr. Arturo, porque me han ayudado en todo momento.

Al profesor Rubén, que me impulso para poder titularme y confió en mí.

Al maestro Oscar Gabriel, quien ha compartido todos sus conocimientos sin esperar nada a cambio, por motivarme a seguir aprendiendo algo nuevo todos los días y por su gran amistad.

A los profesores que han brindado tiempo, esfuerzo y dedicación al enseñarme.

A la Universidad Nacional Autónoma de México, que ha sido mi segunda casa, por formarme como profesional, como persona y por dar a cada estudiante la oportunidad de superarse.

Gracias a todas las personas que están y no están, por su consejo y cariño.

CONTENIDO

INTRODUCCIÓN

Hoy en día es muy importante la creación del software de calidad, para empresas, negocios y cualquier organización. Actualmente se desarrollan aplicaciones orientadas a objetos, ya que con este nuevo paradigma se puede crear software más fácilmente y en menos tiempo, además, éste software crea menor confusión en todo el entendimiento de las aplicaciones. La tecnología orientada a objetos tiene gran aceptación, porque el software, está orientado a objetos del mundo real, cosas con las que convivimos y que abstraemos para la creación de aplicaciones. Es por ello, que el objetivo de esta tesina es presentar y utilizar la Técnica de Modelado de Objetos (TMO) como una herramienta estratégica para desarrollar software orientado a objetos.

Por otro lado, es necesario que las aplicaciones que se realicen, cumplan con lo establecido por la ingeniería del software (mostrada en el capitulo 3), para que así, se realice un mejor sistema. La TMO combina éstas dos grandes ventajas para la construcción del software, ya que es una metodología de ingeniería del software orientado a objetos. Es por ello que se tomo la decisión de realizar esta tesina, ya que además de tener las cualidades antes mencionadas, ésta técnica aún tiene vigencia, no es una herramienta que no este a la vanguardia, fue diseñada por el señor James Rumbaugh a principios de los 90's y aún cuando ya han pasado más de 10 años tiene gran aceptación, por el fácil diseño que hace para los sistemas orientados a objetos.

Además de lo dicho anteriormente, la TMO nos ayuda a emplear técnicas nuevas, para la creación de aplicaciones que se llevan a cabo durante todo el ciclo del desarrollo del software, que servirán de apoyo a los alumnos en su vida como profesionistas. En el nuevo plan de estudios de la licenciatura de Matemáticas Aplicadas y Computación, han incluido la materia de programación orientada a objetos, por la importancia que ésta tiene; pero aprender alguna metodología orientada a objetos tiene un mayor valor porque del buen

análisis y diseño que se haga en un sistema depende el funcionamiento general del software.

La TMO es una metodología de ingeniería de software porque comprende todo el ciclo de vida del software; mucha gente confunde la programación orientada a objetos con la metodología orientada a objetos, pero éstas no son iguales, la metodología incluye a la programación y abarca, muchas otras técnicas, métodos y disciplinas para crear software confiable e independiente del lenguaje que se utilice, para que pueda crecer y extenderse sin ningún problema.

Pero, ¿Por qué crear software a través de "modelos"?, porque es más fácil trabajar con modelos en diferentes proyectos para después llevarlos a la realidad. Por ejemplo, un arquitecto hace construcciones a partir de trazos mostrados en un plano, o un ingeniero automotriz que diseña carros en miniatura para después hacer un carro grande, o también un carpintero antes de realizar algún mueble lo dibuja en una hoja para después construirlo con madera. Así se debe entender la TMO, porque a partir de modelos hechos dentro del análisis y el diseño e implementación, podemos construir software, también, se pueden crear modelos en la pruebas y en el mantenimiento para mejorarlo. Dicen, que una imagen dice más que mil palabras, entonces, un modelo describe mejor el comportamiento de un sistema que el propio lenguaje natural.

Esta tesina consta de cuatro capítulos. El primero, presenta una introducción a los conceptos y principios orientados a objetos, los cuales, son fundamentales para poder entender la TMO. El capitulo dos, muestra la columna vertebral del modelado de objetos, es decir, muestra como se lleva a cabo el modelado de objetos, dinámico y funcional. En el capitulo tres, se presentan todas las etapas del ciclo de vida del software y la forma en que se desarrollan utilizando la TMO. Por último en el capitulo cuatro, muestra mediante un ejemplo, como se realiza una aplicación con ésta metodología.

INTRODUCCION AL MODELADO DE OBJETOS 1

En este capítulo se estudian, los conceptos, principios, herramientas, técnicas y la metodología TMO (Técnica de modelado de objetos) que en conjunto mostrará al lector, una introducción general que servirá de base, para conocer y adentrarse al mundo orientado a objetos.

Generalmente, buscamos la forma más fácil de hacer las cosas, esto, también lo aplicamos en la forma de solucionar los problemas a los cuales nos enfrentamos. La TMO (técnica de modelado de objetos), es una metodología que nos facilita el desarrollo de software orientado a objetos. A su vez, los objetos, nos ayudan a entender de forma clara como operan éstos, en el mundo real. Nosotros llevamos a nuestra mente la imagen de los objetos cuando se nos mencionan, como pueden ser: un libro, una silla, un escritorio, una lámpara, un teléfono, etcétera y así podemos entender que función realiza cada objeto, es decir, un teléfono no es para sentarse o con un escritorio no puede realizar una llamada, no existen incongruencias; en el sentido de que cada objeto se utiliza para llevar a cabo cierta acción. También a los objetos se le pueden hacer transacciones administrativas, como pueden ser compra, venta, préstamo, asignación, y muchos otros tipos de operaciones.

La TMO proporciona grandes ventajas para el desarrollo de software. Por otro lado, hay muchas tecnologías orientadas a objetos, para conocer las ventajas y desventajas que hay entre ellas, es necesario compararlas, para que con ello, el ingeniero de software decida cual es la más conveniente a sus necesidades para la aplicación que vaya a realizar.

1.1 Conceptos y Principios Orientados a Objetos

Vivimos en un mundo de objetos. Consideramos al mundo como un conjunto de entidades y objetos que se relacionan entre sí y se comunican entre ellos. Éstos, pueden clasificarse, combinarse, organizarse, describirse y crearse. Es por ello que la tecnología orientada a objetos propone una visión para la creación de software de computadora, a través de abstracciones del mundo real, para un mejor manejo y entendimiento del software.

El software orientado a objetos es más fácil de mantener ya que su estructura se puede descomponer fácilmente. Pero ¿Qué es un objeto? ¿Por qué se considera una tecnología orientada a objetos? ¿Cómo puedo manejar elementos del mundo real y llevarlos a cabo en el desarrollo del software? Estas son algunas preguntas que nos hacemos

cuando comenzamos a estudiar la tecnología orientada a objetos. Veamos los siguientes conceptos para entender mejor los principios orientados a objetos.

1.1.1 Objetos

Los objetos son entidades que contienen atributos (datos) y formas de comportamiento (procedimientos) particulares, un objeto es casi cualquier cosa. Un escritorio, un libro, una computadora, una impresora, una mesa, una silla, una ventana, una alfombra, etc. Son objetos y cada uno de ellos posee sus propias características; por ejemplo un escritorio tiene 4 patas, tres cajones, una dimensión, un peso y un color. Estas características son atributos que lo describen. En el ejemplo anterior, escritorio es una instancia de una clase mucho más grande de objetos que puede ser la de mobiliario.

1.1.2 Clases y Subclases

Muchos objetos pueden actuar de formas muy similares. Una clase es una descripción de un conjunto de objetos casi idénticos, consta de datos y métodos que resumen las características comunes de ese conjunto de objetos. Es por ello, que la definición de una clase ayuda a la clarificar la definición de un objeto: un objeto es un modelo o instancia de una clase.

Las subclases son clases que se derivan de otras clases. Para entender mejor este concepto, se utilizan los términos padre e hijo, que indica la relación de una clase con una subclase. Las clases padre están ubicadas por encima de las clases hijo en esta jerarquía. Las clases más altas se les llamara superclase y las clases derivadas de esas superclases, se les llama subclases. Así, en una casa de materiales para construcción donde se venden materiales de acero tendríamos el siguiente modelo de superclases y subclases Fig. 1.1.1

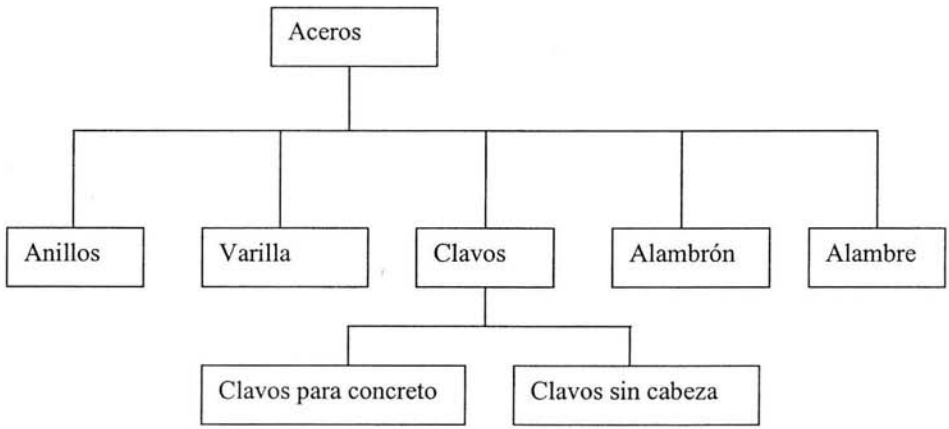


Fig. 1.1.1 Modelo de superclases y subclases de materiales de acero.

En la figura anterior se muestran varios niveles jerárquicos, la clase más alta es aceros que es a su vez superclase que contiene como subclases a los anillos, varillas, clavos, alambrón y alambre que son subclases de Acero. Sin embargo, también podemos notar que clavos es superclase de clavos para concreto y clavos sin cabeza ambos a su vez son su subclase.

1.1.3 Mensajes y Métodos

Los mensajes son el medio con el cual los objetos se relacionan. Los objetos tienen la posibilidad de actuar. La acción sucede cuando el objeto recibe un mensaje, que es una solicitud que pide a otro objeto se comporte de alguna manera. Por ejemplo, cuando un usuario solicita al objeto *documento* se imprima a sí mismo, el documento puede enviar un mensaje al objeto *impresora* solicitando un lugar en la cola de impresión; el objeto *impresora* puede enviar a su vez un mensaje a *documento* pidiendo información del formato, y así pueden interactuar muchas veces. Finalmente, el emisor del mensaje no necesita conocer la forma en que el objeto receptor esta llevando la solicitud. En otras palabras, cuando el objeto *documento* recibe el mensaje de imprimir documento sabe exactamente lo que tiene que hacer, el objeto que envía el mensaje ni sabe ni le importa como re realiza la impresión, solamente realiza la solicitud.

Los procedimientos también llamados métodos se encuentran dentro del objeto y determinan como actúa el objeto cuando recibe el mensaje. Además las instancias almacenan información o datos en el objeto. Los métodos se ejecutan en respuesta a mensajes y manipulan sus valores de cada instancia.

Al igual que las "cajas negras" de la ingeniería, la estructura interior de un objeto esta oculta a usuarios y programadores. Los mensajes que recibe el objeto son los únicos conductos que conectan al objeto con el mundo exterior.

1.1.4 Herencia

La herencia es el mecanismo para compartir automáticamente métodos y datos entre clases y subclases. Un mecanismo potente que no se encuentra en sistemas procedimentales.

La herencia permite a los programadores crear nuevas clases programando solamente las diferencias con la clase padre. Una clase almacena los datos y las funciones que realizan las operaciones sobre los objetos. La mayoría de libros y artículos de revistas sobre la programación orientada a objetos se refieren a las funciones llamándolas métodos.

La herencia además permite que el diseño de clases se realice a través de árboles o jerarquías de clases, por medio de una subclasificación, las clases llegan a ser más específicas y concretas.

Herencia simple y herencia múltiple son dos tipos de mecanismos de herencia, normalmente utilizados en el diseño y la programación orientada a objetos. Con la herencia simple, una clase puede heredar datos y métodos a una subclase simple, así como añadir sus propios datos y mensajes por sí misma. La herencia múltiple se refiere a la posibilidad de una subclase de adquirir los datos y métodos de más de una clase. La herencia múltiple es útil al construir un comportamiento compuesto a partir de más de una rama de jerarquía de clases.

1.1.5 Encapsulación

La encapsulación (encapsulamiento) es el término formal que describe el conjunto de métodos y datos dentro de un objeto, de forma que el acceso a esos datos y métodos solo se permite a través de los propios métodos del objeto y no de otros. Ninguna otra parte del programa orientado a objetos puede operar directamente sobre los datos de un objeto.

1.1.6 Abstracción

La orientación a objetos, fomenta a los analistas, programadores y usuarios piensen en términos abstractos. Comenzando con un conjunto de objetos, los programadores buscan un actor de comportamiento común y lo sitúan en clases abstractas. Las bibliotecas de clases proporcionan un depósito para los elementos comunes y reutilizables.

La maquinaría de la herencia mantiene automáticamente las relaciones entre las clases dispuestas jerárquicamente en una biblioteca de clases. Los marcos estructurales contienen las bibliotecas de clases específicas de la aplicación. Cada nivel de abstracción facilita el trabajo de programación porque hay disponible más cantidad de código reutilizable.*

1.1.7 Polimorfismo

Los objetos actúan en respuesta a los mensajes que reciben. El mismo mensaje puede originar acciones completamente diferentes al ser recibido por diferentes objetos. Este fenómeno se conoce como polimorfismo. Con el polimorfismo un usuario puede usar un mensaje genérico y dejar los detalles exactos de la realización en el objeto receptor. Así en un banco, un cliente puede tener una cuenta bancaria con un saldo que le permite retirar una cantidad de dinero, pero la misma operación puede provocar una situación diferente en un cliente que ya no tenga ningún saldo.

El polimorfismo esta fomentado por la herencia. Es muy normal almacenar las clases con funciones de utilidad en el nivel más alto que sea posible dentro de una jerarquía de clases. Las variaciones necesarias en el comportamiento de las subclases se almacenan

* Software Orientado a Objetos. ANN L. WINBLAD

en niveles más bajos de la jerarquía para sobreescribir los métodos más generales cuando sea necesario. De esta forma, los objetos están listos y pueden responder apropiadamente a mensajes de utilidad, mientras que el método que realiza la operación puede existir en la clase inmediata del objeto o varios niveles por encima del objeto.

1.1.8 Persistencia

La persistencia se refiere a la permanencia de un objeto, es decir, al tiempo durante el cual se asigna espacio y permanece accesible en la memoria de la computadora. En la mayoría de los lenguajes orientados a objetos, se crean modelos de clases a medida que el programa se ejecuta. Algunos de éstos modelos se necesitan por un breve periodo de tiempo. Cuando un objeto ya no es necesario, es destruido y recuperado el espacio de memoria que tenía asignado. La recuperación automática de espacio en memoria se denomina normalmente recolección de basura.

Después de haber ejecutado el programa orientado a objetos, los objetos ensamblados no se almacenan normalmente; es decir, los objetos dejan de ser persistentes. Una base de datos orientada a objetos, mantiene una distinción entre objetos creados solamente para el tiempo de duración de la ejecución de aquellos pensados para almacenamiento permanente. Los objetos almacenados permanentemente se llaman persistentes.

1.2 Que es la Técnica de Modelado de Objetos

La técnica del modelado de objetos es una metodología que combina tres puntos de vista para el modelado de objetos. Primero, el modelado de objetos, representa objetos "estáticos" de datos estructurales del sistema. El modelo dinámico representa el comportamiento dinámico "control" del sistema. Por último, el modelo funcional que representa la transformación del sistema "función".

En el desarrollo del software se utilizan estos tres aspectos: estructura de datos (modelado de objetos), operaciones en el tiempo (modelado dinámico) y transformación de valores (modelado funcional).

Para comprender mejor estos conceptos tenemos que comenzar por entender lo que significa el modelado como técnica de diseño.

El modelo es una abstracción de algo que queremos comprender antes de construirlo. Los modelos los podemos emplear en sistemas muy complejos, difíciles de desarrollar directamente, pero, a través de los modelos podemos obtener vistas del sistema, darnos cuenta de los requisitos necesarios para llevar a cabo la implementación, podemos verificar si satisface o no las necesidades del cliente y los objetivos trazados por la empresa.

Los modelos en el caso de la computación tienen varios objetivos:

- Probar una entidad física antes de construirla. Actualmente, con los avances tecnológicos en la computación, podemos realizar simulaciones sin tener que realizar el proyecto físicamente.

- Comunicación con el cliente. Las maquetas son proyectos que demuestran todo, o la mayoría del comportamiento de un sistema de forma explícita.

- Reducción de la Complejidad. Es mucho mejor entender el sistema parte por parte, que todo al mismo tiempo, ya que un sistema complejo abarca un extenso número de situaciones en el software.

Existen distintos modelos que se manejan hoy en día , entre ellos tenemos a los:

- Mapas: modelos bidimensionales de nuestro mundo.

- Globos terráqueos: modelos tridimensionales de nuestro mundo.

- Diagramas de flujos: representación esquemática de las decisiones y la secuencia de actividades para llevar a cabo un determinado procedimiento.

- Dibujos arquitectónicos: representación esquemática de una casa, edificio, puente, etc.
- Partituras musicales: representaciones gráficas y textuales de notas musicales y tiempos de una pieza musical.

Modelado de Objetos. El modelado de objetos describe la estructura de un sistema (identidad, relaciones entre los objetos, atributos y operaciones). No olvidemos que los objetos son unidades en que dividimos al mundo, las partes que podemos separar. El objetivo de construir un modelos de objetos, es poder describir elementos del mundo real y este modelo se puede representar gráficamente mediante diagramas que contengan clases de objetos, jerarquía, que compartan estructuras y comportamientos comunes y éstas a la vez se asocien con otras clases.

Modelo Dinámico. En el modelo dinámico lleva a cabo las acciones del sistema a través de la temporización y secuencia de operaciones. Es decir, toma el control de las operaciones y lleva a cabo cada una de éstas, secuencial, lógica y correctamente, sin tomar en cuenta lo que realicen las operaciones. El modelo dinámico se representa gráficamente mediante diagramas de estado. Cada diagrama muestra los estados y la secuencia de sucesos.

Modelo Funcional. El modelo funcional describe los resultados o efectos que transforman los valores y las restricciones del sistema. El modelo funcional se representa mediante un diagrama de flujo de datos. Este modelo captura lo que hace el sistema independientemente de cuando lo haga o cómo lo haga.

1.3 Ventajas de Software Orientado a Objetos

Las principales ventajas del método orientado a objetos descansan en su posibilidad de hacer frente a dos temas esenciales de la ingeniería de software, gestión de la complejidad y mejora de la productividad en el proceso de desarrollo de software. Así es posible:

- Escribir código reutilizable
- Escribir código fácil de mantener
- Depurar módulos de código existente
- Compartir código con otros

Los mecanismos orientados a objetos, en particular la herencia, fomentan activamente la reutilización de código ya que en vez de copiar y modificar módulos, los programadores pueden utilizar librerías de clases, obteniendo código comprobado y depurado.

La orientación a objetos presenta un cambio fundamental en la forma de desarrollar y utilizar software. La reutilización de software implica que las clases pueden mezclarse y ajustarse y ser fácilmente modificadas para construir nuevas aplicaciones. La encapsulación de datos y procedimientos cambia toda la naturaleza del proceso de programación. Además la encapsulación de datos y procedimientos simplifica el proceso, facilita el mantenimiento y reduce la posibilidad de errores en la programación.

En el método orientado a objetos, los objetivos del diseño pasan de modelar el comportamiento del mundo a modelar los objetos que existen en el mundo y sus comportamientos individuales. Si se practican correctamente las técnicas orientadas a objetos, la arquitectura de la aplicación se apega mucho más a la estructura del problema. Esto hace que el desarrollo, empleo y mantenimiento de una aplicación sea mucho más regular, fácil y rápida.

1.3.1 Reutilización

Las técnicas orientadas a objetos ofrecen una alternativa para escribir los programas una y otra vez. El programador orientado a objetos modifica la funcionalidad de un programa reemplazando los elementos y objetos antiguos por los nuevos objetos o simplemente incorporando nuevos objetos a la aplicación. En el método procedimental, los programadores centran su atención en los temas del lenguaje, mientras que en el entorno

orientado a objetos, el tema importante es crear una librería de clases o un conjunto de objetos robustos que pueden ser utilizados en diversas circunstancias.

Los métodos orientado a objetos para desarrollar interfaces gráficas de usuarios proporcionan un ejemplo actual de la reutilización del código. Cuando la interfaz gráfica de usuario se popularizó, facilitó la computarización de una amplia clase de usuarios, y en consecuencia, dificulto la programación en los programadores.

No obstante, la reutilización del software no ocurre accidentalmente, ni siquiera con los lenguajes orientados a objetos. Los diseñadores de un sistema deben de tener en cuenta las ventajas de la reutilización planear por adelantado la reutilización de lo que ya existe y diseñar los nuevos componentes que serán creados.

1.3.2 Aumento de la Productividad

Es más fácil modificar y ampliar una aplicación orientada a objetos. Se pueden añadir nuevos tipos de objetos sin cambiar la estructura existente de una aplicación. La herencia permite construir nuevos objetos a partir de los antiguos.

Las características orientadas a objetos son extremadamente útiles durante el mantenimiento. La modularidad facilita el contener los efectos de los cambios realizados en un programa. El polimorfismo reduce el número de procedimientos y por tanto el tamaño del programa que debe entender la persona encargada del mantenimiento. La herencia de clases permite construir una nueva versión de un programa sin afectar a la antigua. El mecanismo de herencia documenta los cambios en un programa como subclases que representa la historia de las modificaciones realizadas en la superclase.

La programación orientada a objetos mejora no sólo el proceso de desarrollo de software, sino también, la flexibilidad y utilidad de éste. El proceso del diseño es más intuitivo porque cada elemento del software tendrá su correspondiente elemento del mundo real en la aplicación.

Así, podemos resumir los siguientes puntos que son importantes para el desarrollo de software eficaz.

- Las principales ventajas del desarrollo de software orientado a objetos descansan en la reducción de la complejidad y aumento de la productividad por parte del desarrollador.

- Las clases dividen los problemas complejos en módulos sencillos y ocultan los detalles de la realización. La claridad de la estructura del software esta presente en el diseño, en el programa y en la aplicación final.

- Las bibliotecas de clases proporcionan una excelente plataforma para el desarrollo de aplicaciones orientadas a objetos y la reutilización del código.

- La estructura y función del software orientado a objetos aumenta su fiabilidad.

- Las aplicaciones orientadas a objetos se pueden ampliar y modificar más fácilmente, de acuerdo a las necesidades de los usuarios, reduciendo el esfuerzo por parte del programador.

Dentro de la programación también existen ventajas con el método orientado a objetos.

1.3.3 El Método Tradicional Frente al Método Orientado a Objetos

Desde el punto de vista del programador tradicional, algunas técnicas orientadas a objetos parecen ser conceptos tradicionales con nombres diferentes. Y es cierto, algunos conceptos orientados a objetos son análogos a los métodos de programación convencional. A continuación se mostrarán algunas diferencias entre términos y conceptos convencionales, y aquellos orientados a objetos.

- Un método es como un procedimiento, porque ambos contienen instrucciones de procesamiento. Las variables de clase y modelo corresponden a los datos de la programación tradicional. La diferencia fundamental consiste que la programación orientada a objetos encapsula tanto a los datos como a los métodos en una clase y la programación tradicional no.

- Una clase es como un tipo abstracto de datos, aunque para los programadores orientados a objetos, el proceso de escribir los datos no se revela fuera de la clase.

- La herencia no tiene una analogía inmediata en la programación convencional.

- El paso de mensajes reemplaza a las llamadas de función como método principal de control en los sistemas orientados a objetos.

Con las llamadas de función, los valores se presentan y el control regresa a la función que efectúa la llamada. Por el contrario, los objetos entran en acción gracias a los mensajes, y el control esta distribuido. La siguiente tabla resume la similitud entre conceptos convencionales y los orientados a objetos.

Técnicas orientadas a objetos	Técnicas Tradicionales
Métodos.	Procedimientos, funciones o subrutinas.
Variables modelo.	Datos.
Mensajes.	Llamadas a procedimientos o funciones.
Herencia.	No existe técnica similar.
Llamadas bajo control del sistema.	Llamadas bajo control del programador.

Figura 1.3.1 Tabla de comparación del paradigma orientado a objetos y el paradigma tradicional.

Dentro de la programación orientada a objetos, los procedimientos y datos se agrupan para formar una entidad llamada objeto. A diferencia de los datos pasivos y de los sistemas tradicionales, los objetos pueden actuar. Las acciones son desencadenadas por los mensajes enviados a un objeto y desarrolladas por procedimientos internos llamados métodos.

Los objetos pueden ser bastante similares a su concepción real. La descripción de un conjunto similar es una clase. Un modelo o instancia de una clase es un objeto real descrito por una clase. La abstracción de un objeto es estimulada mediante el empleo de jerarquías de clase. Para reutilizar los métodos definidos previamente, las jerarquías de clases cuentan con el mecanismo de herencia del paradigma orientado a objetos.

Como se puede apreciar, a pesar de la similitud que hay por parte de la programación convencional y la orientada a objetos hay dos caminos muy diferentes en cuanto a la forma de realizar una aplicación de software, porque en la programación orientada a objetos debemos de explotar principios como la herencia, el encapsulamiento de datos y el polimorfismo; elementos que nos ayudarán en la creación de las aplicaciones, facilitando el trabajo y su entendimiento para el mantenimiento del software.

A continuación se mostrará algunos puntos importantes de la programación orientada a objetos:

La orientación a objetos se define por medio de un conjunto de mecanismos: objetos, clases, modelo, métodos, mensajes y programación convencional. Estos

mecanismos dan origen a conceptos clave inherentes a los sistemas orientados a objetos como son: encapsulación, abstracción y polimorfismo.

Las herramientas y conceptos orientados a objetos llevan, esencialmente a los programadores a escribir código en un nivel superior de abstracción.

1.3.4 Ventajas de los Lenguajes Orientados a Objetos

Un lenguaje de programación que soporta el paradigma orientado a objetos, métodos y mensajes beneficia al desarrollador de software proporcionando una forma natural de modelar el fenómeno en el complejo mundo real. Aunque en éste paradigma los objetos, métodos y mensajes son con frecuencia difícil de entender para los programadores acostumbrados a los procedimientos y datos, la orientación a objetos ofrece modelos muy semejantes al mundo real.

Consideremos el siguiente ejemplo: El proceso de añadir, cambiar o borrar elementos en el archivo de un empleado. En la programación tradicional como se muestra en la fig. 1.3.2 AnaEmp( ) es llamado para realizar su tarea y colocar datos en los archivos.

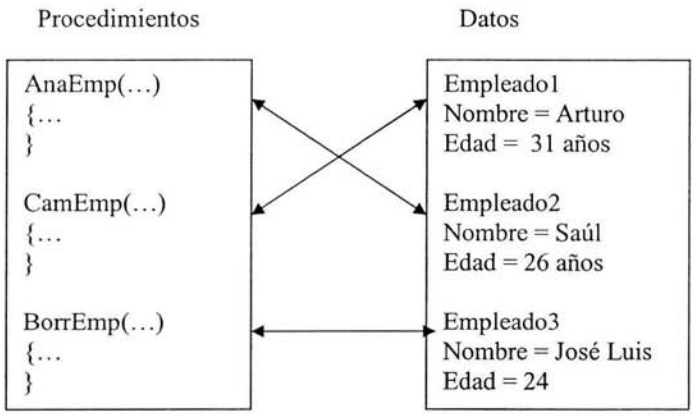


Figura 1.3.2 Método tradicional para modificar el archivo de Empleado.

Ahora lo comparamos con el método orientado a objetos observando la figura 1.3.3

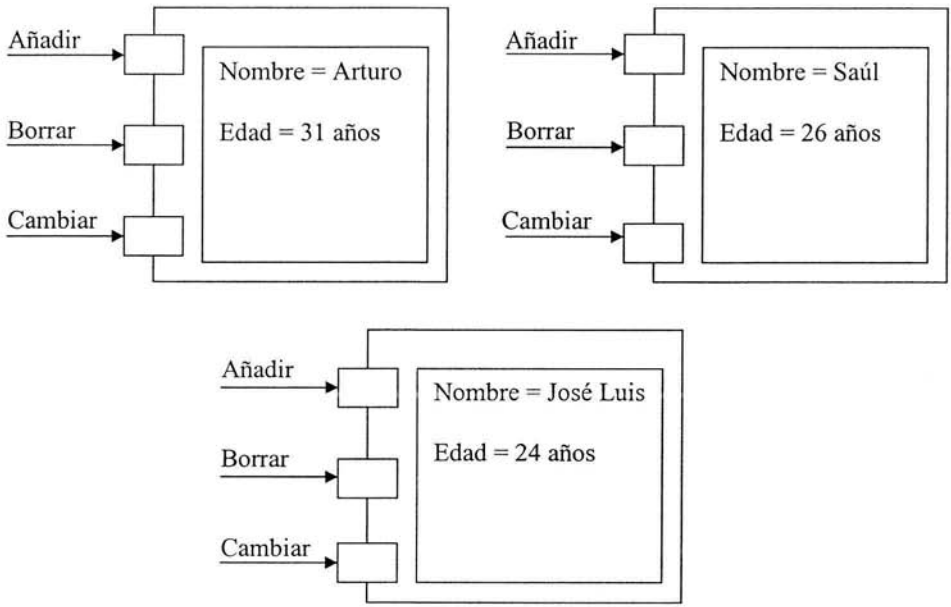


Fig. 1.3.3 Método orientado a objetos para modificar el archivo Empleado.

En el sistema orientado a objetos el mensaje añadir de la figura anterior se envía a la clase empleado. El cambio del método añadir no afecta a los otros métodos, ni a la estructura de datos del objeto.

Las subclases heredan tanto las estructuras de datos como los métodos de clases ya existentes.

Los lenguajes de programación orientados a objetos benefician al desarrollador soportando la realización de software modular. Los objetos posibilitan en mantener juntos los elementos relacionados y en particular, mantener los datos y métodos que interactúan con esos datos. Sin embargo también hay algunos inconvenientes en la programación orientada a objetos. El inconveniente más citado es que los programas orientados a objetos se ejecutan más despacio que los lenguajes procedimentales. Otro problema con los lenguajes orientados a objetos es que el programador debe aprenderse una extensa librería de clases antes de alcanzar cierta eficacia en un lenguaje orientado a objetos.

1.3.5 Bibliotecas de Clases

Una biblioteca de clases es un conjunto de clases utilizadas para una tarea determinada de programación. Cuánto más coincida una biblioteca de clases con una aplicación, menos modificación se necesita.

En un extremo se encuentra la biblioteca de clases de las unidades fundamentales de la construcción del programa, como cadena, pilas y listas enlazadas. En el otro extremo se encuentran las bibliotecas de clases que proporcionan un marco estructural completo para la categoría de aplicaciones, por ejemplo para construir interfaces gráficas de usuarios que incluye clases para menús, cursores y pantallas de ayuda.

1.3.6 Herramientas de Desarrollo

Las dos herramientas más importantes para los lenguajes orientados a objetos son los examinadores "hojeadores" y los depuradores simbólicos. Un examinador y un hojeador (Browser) de código fuente es una herramienta interactiva que proporciona una visión del código de la aplicación y de la biblioteca de clases. Es principalmente una ayuda de navegación, permitiendo al programador ver la estructura general jerárquica de la aplicación, así como pasar de una clase a sus clases antecesoras o descendientes, de un método a las clases que realizan el método o de un mensaje a las clases que realizan dicho mensaje. Además los examinadores son ayudas tutoriales porque facilitan la presentación y aprendizaje de definiciones de métodos y clases existentes y proporcionan plantillas (templates) para crear nuevas clases y métodos.

Los depuradores simbólicos para los lenguajes orientados a objetos proporcionan las mismas capacidades de los depuradores tradicionales, ampliadas al paradigma orientado a objetos. Estos depuradores orientados a objetos efectúan el seguimiento de las llamadas a los métodos, al igual que lo hacen los métodos tradicionales con las llamadas a funciones. Los depuradores permiten fijar puntos de ruptura dentro de los métodos, o en la llamada a un método. Además pueden examinar y alterar los valores de las variables modelo.

Buena parte de la potencia de un depurador radica en su habilidad de ver el programa en prueba de diferentes maneras. Un depurador orientado a objetos debería proporcionar vistas del interior de los objetos individuales durante la ejecución, vistas de las relaciones entre los objetos y vistas de las clases de las cuales se derivan los objetos.

1.3.7 Bases de Datos Orientadas a Objetos

Las bases de datos orientadas a objetos proporcionan almacenamiento central para los datos, mecanismos para compartir dichos datos entre los usuarios y formas de asegurar la integridad y seguridad de los datos. Recientemente, la tecnología de bases de datos ha luchado para encontrar formas de continuar proporcionando estas ventajas y al mismo tiempo incrementar el almacenamiento de datos complejos y diversos.

Las bases de datos orientados a objetos no almacenan datos aisladamente sino más bien siguen el paradigma orientado a objetos; vincular los datos junto con el comportamiento asociado dentro de los objetos. Las capacidades funcionales de las bases de datos orientadas a objetos incluyen soporte de construcciones ricas en modelado de datos, soporte directo de inferencia o deducción, y la posibilidad de almacenar tipos de datos como por ejemplo, imagen, audio, voz, vídeo.

El modelo relacional sufre por lo menos un inconveniente notable. Es difícil expresar la semántica de objetos complejos con solo una tabla para el almacenamiento de datos. Las aplicaciones de los años 90´s requirieron soporte para estructuras de datos extensibles y complejas, acceso a datos complejos y de alto rendimiento. ¿Cómo cuales? Los conceptos tradicionales de bases de datos se están ampliando gradualmente con capacidades orientadas a objetos para satisfacer éstas necesidades. La evolución de la tecnología de base de datos durante los últimos 30 años, como se aprecia en la figura 1.3.4 representa el giro hacía una gestión de datos más complejos.

Las bases de datos orientadas a objetos toman ventaja, en las aplicaciones en las que las relaciones entre elementos de las bases de datos llevan información clave. Es decir, los modelos orientados a objetos capturan la estructura de los datos; los modelos relacionales organizan los datos en sí. Si un registro puede ser comprendido aisladamente, entonces las base de datos relacional es adecuada. Si un registro tiene sentido solamente en el contexto de otros registros, entonces una base de datos orientada a objetos es más apropiada.

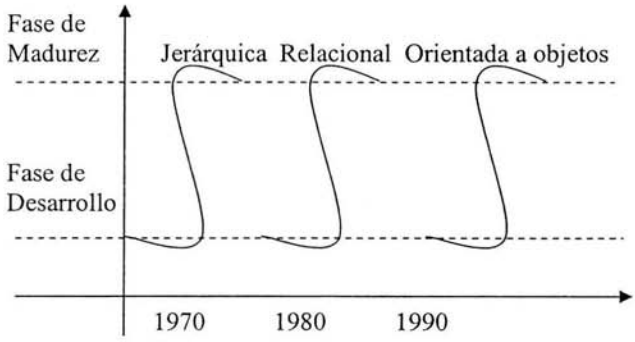


Fig. 1.3.4 Evolución de la Tecnología de Base de datos.

Las bases de datos orientadas a objetos están evolucionando como resultado de las limitaciones del modelo relacional en su incapacidad de manejar, tipos de datos numerosos y complejos. Las extensiones orientadas a objetos para las bases de datos relacionales existentes van probablemente a establecer el rumbo de las compañías de bases de datos totalmente orientadas a objetos se realizarán con toda probabilidad en áreas de aplicación CAD y CASE. Las bases de datos orientadas a objetos no sólo contemplan a los lenguajes orientados a objetos; también ofrecen ventajas sobre el método actual relacional, soportando aplicaciones complejas, perfeccionando la programabilidad y el rendimiento,

mejorando, el acceso por navegación, simplificando el control de concurrencia y reduciendo los riesgos de integridad referencial. *

1.4 Porque Utilizar TMO para el Desarrollo de Software

La Técnica del modelado de objetos es una metodología de análisis y diseño orientadas a objetos. Una justificación convincente para utilizar ésta metodología y no otras, en realidad no existe, hay muchas metodologías para crear aplicaciones orientadas a objetos. A mi parecer depende más del gusto y conocimiento que se tenga de ella, además de la sencillez para modelar y construir aplicaciones del mundo real con objetos, de forma eficiente y madura.

La Técnica de Modelado de Objetos, se puede utilizar para analizar los requisitos del problema, diseñar una solución e implementarla en un lenguaje de programación o bases de datos.

1.4.1 La TMO Utilizando la Ingeniería de Software

Una metodología de ingeniería de software produce software de forma organizada, mediante un conjunto de técnicas y notaciones predefinidas. La metodología es una serie de pasos que se utilizan para llegar a un resultado o un fin, después de que se obtuvo el objetivo, esos pasos se sistematizan para utilizarlos en el momento en que sean requeridos nuevamente. Así, en cada paso se utiliza una serie de técnicas y notaciones específicas. La metodología TMO apoya a todo el ciclo de vida del software, desde la formulación inicial del problema, pasando por el análisis, diseño, implementación, pruebas del software y mantenimiento. Es por ello, que esta metodología puede desarrollar software Orientado a Objetos de gran calidad.

1.4.2 Construcción del Software Utilizando la TMO

La metodología TMO consta de varias fases:

El análisis, se dedica a la comprensión y al modelado de la aplicación y del entorno en el cual funciona. En esta fase se realiza una descripción general del problema que hay que resolver y proporciona una visión general del sistema que se propondrá. Para obtener mayor información, se realizará un diálogo con el cliente y se observará el entorno del sistema para saber cómo opera en el mundo real.

La arquitectura del sistema se realiza mediante el Diseño del mismo. Utilizando el modelo de objetos como guía se organiza el sistema en subsistemas. Durante ésta fase se elaboran los modelos de análisis, se refinan y se optimizan para obtener un diseño práctico

* Software Orientado a Objetos
Ann L Wimbland

y con gran soporte. Además el diseño, centra su atención desde los conceptos de la computación hasta la implementación en la computadora. Así se fijan los algoritmos establecidos para implementar todas las funciones en el sistema.

En el desarrollo del software se realiza un mayor esfuerzo en la fase del análisis. A veces, resulta desconcertante, dedicar más tiempo al análisis y al diseño, pero este esfuerzo será mejor compensado en la implementación, ya que será más rápida y sencilla de realizar. Dado que el diseño es más limpio se adaptará mucho más rápido a los cambios futuros.

También la metodología orientada a objetos centra su atención en las estructuras de datos, y no en las funciones que se efectúen. Esto hace que el principal foco de atención sea el objeto.

En la Técnica de Modelado de Objetos, el modelado de objetos realizado en la fase del análisis sirve para las otras fases del desarrollo del software como son el diseño y la implementación. En cada fase se realiza un refinamiento del modelo con niveles progresivamente más detallados utilizando una misma representación.

Además, el proceso de desarrollo de software mediante la Técnica de Modelado de Objetos es iterativo más que secuencial. Esto es, no cambia de notación de una fase a otra y en cada fase se hace un refinamiento de clases de forma iterativa hasta llegar a clarificar sus características así hay menos oportunidades de producir errores.

En resumen, las principales razones por las que se debería utilizar esta metodología, radica en el hecho de que es una metodología que contempla todas las fases de desarrollo del ciclo de vida del software, además de que es contemplada en la Ingeniería de Software, lo que habla de gran soporte en la construcción del mismo. Así mismo, ésta metodología se puede utilizar en cada fase del desarrollo del software sin tener que cambiar su notación por otra y además, con los objetos como ventaja primordial se podrán realizar refinamientos de los mismos en cada iteración que se haga en cada fase.

1.5 TMO Comparada con Otras Metodologías

Antes de hacer una comparación entre metodologías, se hará un repaso de algunos conceptos importantes.

Hay un acuerdo en considerar a la metodología como un conjunto de pasos y procedimientos que deben de seguirse para el desarrollo de software. Se puede considerar también una metodología de desarrollo, como un conjunto de procedimientos, técnicas, herramientas y soporte documental que ayudan a los desarrolladores a realizar nuevo software. Por lo tanto, una metodología representa el camino para desarrollar software de manera sistemática.

Según Piattini* hay confusión entre los términos metodología, método y ciclo de vida, ya que hay autores que lo utilizan indistintamente. Vea las siguientes diferencias:

Una metodología puede seguir uno o varios modelos del ciclo de vida, esto es, el ciclo de vida indica lo que hay que obtener a lo largo del desarrollo del proyecto, pero no cómo. Esto sí lo debe indicar la metodología.

Una metodología es un concepto más amplio que el método. Así, se puede considerar como un conjunto de métodos. Inicialmente existían métodos que se centraban en una fase del ciclo de vida, métodos de programación estructurada, a los que les siguieron los de diseño estructurado, y después, análisis estructurado. A partir de aquí se produce un punto de inflexión al establecerse, un conjunto de métodos de análisis y diseño estructurado enlazado en dos fases. Desde aquí, se ve la tendencia en ir abarcando el ciclo de vida completo.

Según el diccionario de la real academia española, el término método es un modo ordenado de proceder para llegar a un resultado o fin determinado, para describir la verdad y sistematizar los conocimientos.

Existen muchas metodologías para el desarrollo de software, todas y cada una tiene sus ventajas y desventajas. Se realizará una breve descripción de algunas de ellas y se compararan con la Técnica de modelado de objetos. El objetivo será el de dar a conocer las diferentes metodologías para que, quien realice software, tenga en mente que existen diferentes formas para hacerlo y tome la decisión adecuada para emplearla en su sistema.

1.5.1 Análisis Estructurado / Diseño Estructurado (AE / DE)

En la actualidad, las metodologías de ingeniería de software que tienen mayor aceptación son las que se basan en diagramas de flujo de datos. Es por ello que Yourdon, Constantine, DeMarco se han interesado y han escrito a cerca del AE/DE.

AE / DE incluye gran número de notaciones que utiliza en la creación del software. Durante la fase del análisis, se utilizan los diagramas de flujo de datos, se especifican los procesos, se realizan diccionarios de datos, diagramas de transición de estados y diagramas entidad-relación para describir lógicamente el sistema. AE / DE divide recursivamente los procesos complejos en subdiagramas, cada vez más entendibles hasta alcanzar procesos fáciles de aplicar.

Utiliza un diccionario de datos, en donde contiene los detalles que faltan en los diagramas de flujo de datos. Además, aquí se definen los flujos, almacenes de datos y el significado de los distintos nombres.

---

* Análisis y Diseño Detallado de Aplicaciones Informáticas y de Gestión. Mario G. Piattini.

Los diagramas de transición de estados modelan el comportamiento que depende del tiempo, es muy parecido al modelo dinámico, o sea, la mayoría de los diagramas de transición describen los procesos de control y ejecución de funciones. Así como el acceso a datos desencadenado por los sucesos.

Los diagramas entidad-relación se utilizan para describir los almacenes de datos. Todo elemento de datos entidad-relación corresponde con un almacén de datos de un diagrama de flujo de datos. El objetivo primario de los diagramas entidad-relación es representar objetos de datos y sus relaciones.

Todas las herramientas anteriores se realizan durante el análisis estructurado. El diseño estructurado sigue al análisis estructurado y aborda los detalles de bajo nivel. Durante el diseño estructurado los procesos de diagramas de flujo de datos se agrupan en tareas y se asignan a procesos.

1.5.2 AE / DE Comparado con TMO

AE /DE y TMO tienen mucho en común. En la metodología de análisis estructurado y diseño estructurado, el modelo funcional es fundamental; va seguido después del modelo dinámico. Por el contrario, en TMO se considera al modelo de objetos como el más importante.

AE / DE organiza el sistema en torno a los procedimientos, en cambio, TMO organiza el sistema de acuerdo a objetos del mundo real. Si se requieren hacer cambios de requisitos en el sistema, resultarían desastrosos para el diseño basado en procedimientos, mientras que los cambios de función no representar mucho problema en el diseño orientado a objetos.

En el AE / DE la descomposición de procesos en subprocesos es arbitraria. Así, cada persona puede obtener una descomposición diferente a otra. En el diseño orientado a objetos realizan una descomposición en base a los objetos, por ello, los desarrolladores de diferentes programas tienden a encontrar objetos similares.

1.5.3 Desarrollo Estructurado de Jackson (DEJ)

Otra tecnología para el desarrollo de sistemas es el Desarrollo Estructurado de Jackson, esta metodología fue propuesta por Michael Jackson. La forma de modelar el mundo real, se hace utilizando entidades, acciones y secuencias de acciones. Las entidades son personas, objetos u organizaciones que un sistema necesita para producir o utilizar información y las acciones las cosas que suceden en el mundo real y que afectan a las entidades.

El desarrollo de software con DEJ consta de seis pasos:

- El desarrollador de software hace una lista de las entidades y acciones que abstrae del mundo real.

- Se realiza una estructura de entidades y se ordena de forma temporal.
- El modelo inicial indica la forma en que el mundo real se conecta con el modelo abstracto.
- Se utiliza un pseudocódigo para indicar las salidas o resultados de las acciones.
- Se realiza una temporización de los sistemas, que son notas informales donde se indican los resultados de las restricciones de rendimiento.
- La implementación del sistema se enfoca en los problemas de planificación y se asignan los procesadores necesarios para realizar los procesos.

1.5.4 JDS Comparado con TMO

Algunos autores consideran que la tecnología DEJ esta orientada a objetos. Sin embargo, DEJ identifica pocas entidades y muestra muy poco de su estructura. Además DEJ es difícil de entender en su totalidad por lo complejo que suele ser el fuerte uso que se hace del pseudocódigo. No hay duda de que los modelos gráficos son más fáciles de entender.

DEJ esta diseñado para realizar sistemas de tiempo real, es por ello que es útil en sistemas en el cual los procesos se sincronizan entre sí. En sistemas que requieran ser extremadamente detallados y que requieran rapidez en la solución de un problema. En la programación de computadoras paralelas. Sin embargo a diferencia de OMT no facilita el entendimiento del problema ya que no es eficiente en la abstracción y simplificación del mundo real. El diseño de DEJ es muy complejo para el desarrollo de Base de Datos por lo que es poco adecuada para su diseño.

1.5.5 Aproximación Entre el Modelo Entidad-Relación (ER) y TMO

El modelo de objetos TMO combina conceptos orientados a objetos (clases y herencia) con conceptos de modelado de información (entidades y asociaciones). El modelado de información surgió en las Bases de Datos por Peter Chen. El modelo ER es una técnica gráfica que es popular porque la notación es fácil de entender y además muy potente para la resolución de problemas reales.

El modelado TMO es también una forma mejorada del modelo ER. Se añaden conceptos como cualificación y una metodología para programación y para diseño de Base de Datos.

1.5.6 TMO comparado con Otras Metodologías Orientadas a Objetos

Se ha publicado poco a cerca de las metodologías orientadas a objetos para la ingeniería de software. Sin embargo hay gente como G. Booch.* Quien ha desarrollado trabajos orientados a objetos e indica las diferencias de las aproximaciones funcionales tradicionales del diseño, como pueden ser las basadas en flujo de datos. La metodología de Booch

* Object-Oriented Development BOOCH, G.

incluye una gran variedad de modelos que abarcan los aspectos de objetos, dinámico y funcional de un sistema de software. Una distinción fundamental entre la propuesta de Booch y OMT es el énfasis que ésta hace a las asociaciones. Booch no ha incorporado las asociaciones en su metodología. Hay mucho más similitudes que diferencias y ambas metodologías se contemplan una con otra.

Meyer no es realmente una metodología.* El libro de Meyer esta muy orientado hacía un lenguaje como vehículo para realizar un diseño. En él no se enfrenta al modelado conceptual ni al análisis.

Coad y Yourdon presentan el análisis orientado a objetos similar a la propuesta por OMT, pero ponen poca atención al diseño.**

Jacobson realiza una metodología completa para el desarrollo del software.*** Analiza el sistema en términos de entidades (un modelado de objetos) y casos prácticos(prototipos de escenarios que abarcan un comportamiento dinámico).

Como podemos observar en éstas metodologías orientadas a objetos hay más similitudes que diferencias, ya que ha pesar de que no hay una estandarización de términos o notaciones la tecnología sigue siendo la misma por el cual es difícil crear o desarrollar sistemas orientados a objetos por caminos diferentes.

* Reusable Software. MEYER, B.
** Object-Oriented Design. COAD, P & YOURDON, E.
*** Object-Oriented Software Engineering. JACOBSON, I.

TIPOS DE MODELADO DE OBJETOS 2

En este capitulo, se muestra la columna vertebral de TMO técnica de modelado de objetos, en la cual se basa el señor James Rumbaugh junto con sus colaboradores, para explicar como se lleva a cabo el modelado de los objetos utilizando ésta metodología.

Cuando realizamos un modelo, estamos llevando a cabo una representación pequeña de alguna cosa. En él, se hacen los cambios y modificaciones pertinentes para corregir los errores que se encuentren durante la construcción del sistema.

La TMO nos dice cómo modelar el sistema de estudio para realizar el software de aplicación. Es decir, primero se tiene que realizar un modelo que se asimile lo más que se pueda, a la realidad en donde se pretende implementar el software. Para modelar el sistema, se utilizan tres tipos de modelado que son: modelado de objetos, modelado dinámico y modelado funcional.

2.1 Modelado de Objetos

El modelado de objetos describe la estructura de un sistema (identidad, relaciones entre los objetos, atributos y operaciones). No olvidemos que los objetos son unidades en que dividimos al mundo, las partes que podemos separar. El Objetivo de construir unos modelos de objetos, es poder describir elementos del mundo real y este modelo se puede representar gráficamente mediante diagramas que contengan clases de objetos, jerarquías, que compartan estructuras y comportamientos comunes y estas a la vez se asocien con otras clases.

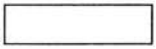
2.1.1 Diagramas de Objetos

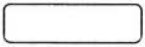
Ya se han visto y descrito algunos ejemplos de dos conceptos básicos como son las clases y los objetos. Solo se han visto en forma descrita, pero se requiere de un formalismo para expresar los modelos de objetos de forma clara y precisa.

Los diagramas de los objetos. Proporcionan una notación gráfica formal para el modelado de objetos, clases y sus relaciones entre sí, son útiles tanto para el modelado abstracto, como para diseñar programas reales. Hay dos tipos de diagramas de objetos: diagramas de clases y diagramas de instancias. Estas notaciones fueron hechas por el señor James Rumbaugh, hay otro tipo de notaciones y metodologías orientadas a objetos.*

Diagrama de Clases. Es un esquema, patrón o plantilla para describir muchas instancias de datos posibles. Los diagramas de clases describen clases de objetos.

Diagramas de Instancias. Describen la forma en que cierto conjunto de objetos se relacionan entre sí. Los diagramas de Instancias son útiles para documentar casos prácticos y para describir ejemplos.

Un rectángulo  va a contener las clases y sus atributos.

Un rectángulo ondulado  va a contener los objetos y sus valores.

Los atributos son valores que se encuentran dentro de las clases. Dentro de los atributos hay valores que se les dan a los objetos. Por ejemplo, Si tenemos una clase que se llama credencial para votar y tiene como atributos: nombre, edad, sexo, dirección; éstos tendrán valores dentro de los objetos. Como se muestran en la figura 2.1.1

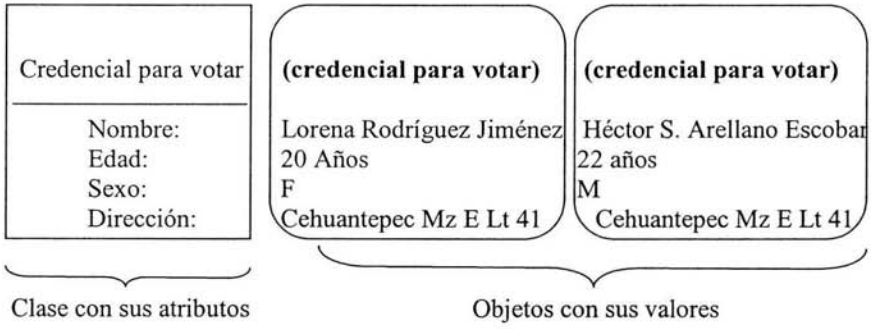


Fig. 2.1.1 Modelado del objeto credencial para votar y sus instancias.

Además podemos ver que cada atributo tiene un valor para cada instancia del objeto.

* Modelado y Diseño Orientados a Objetos RAMBAUGH, J.

2.1.2 Modelado de Operaciones y Métodos

Una operación es una transformación aplicable a los objetos de una clase. Por ejemplo, en una casa de Materiales para construcción, comprar, vender, invertir, son operaciones aplicables, a todo el material que se maneja en ella. Los objetos que pertenecen a una misma clase comparten las mismas operaciones. También una operación se puede aplicar a diferentes clases, por ejemplo, en la casa de materiales podemos tener una clase para todos los tipos de cemento que se trabajen, así como también una clase para los materiales de acero, aún cuando son distintas clases pueden aplicarse todas las operaciones mencionadas. Cuando eso sucede se dice que la operación u operaciones son polimórficas. Aunque las operaciones sean las mismas adoptan distintas formas en cada clase.

Por otro lado, un método es la implementación de una operación para una clase. Los objetos tienen comportamientos, para implementar esos comportamientos lo hacemos a través de los métodos, dentro de un lenguaje de programación. Los métodos están dentro de las clases y sólo se pueden aplicar a la clase donde están definidos.

Una clase, se representa mediante un rectángulo que puede dividirse hasta en tres segmentos. La primera división contendrá el nombre de la clase, posteriormente la lista de atributos y por último las listas de operaciones que se podrán realizar en esa clase. Los atributos pueden contener además del nombre, el tipo de dato y un valor, mientras que las operaciones pueden contener el nombre, una lista de atributos y el tipo de resultado, como se ve en la siguiente figura.

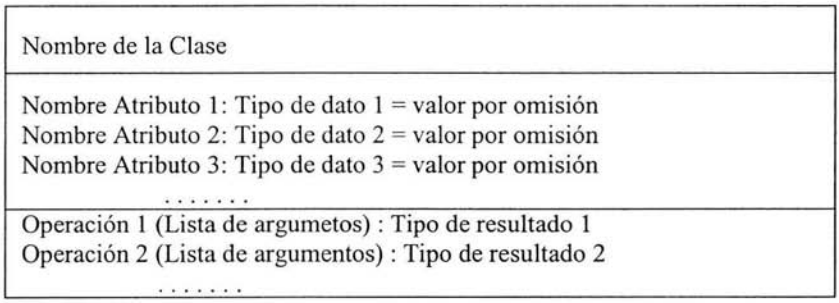


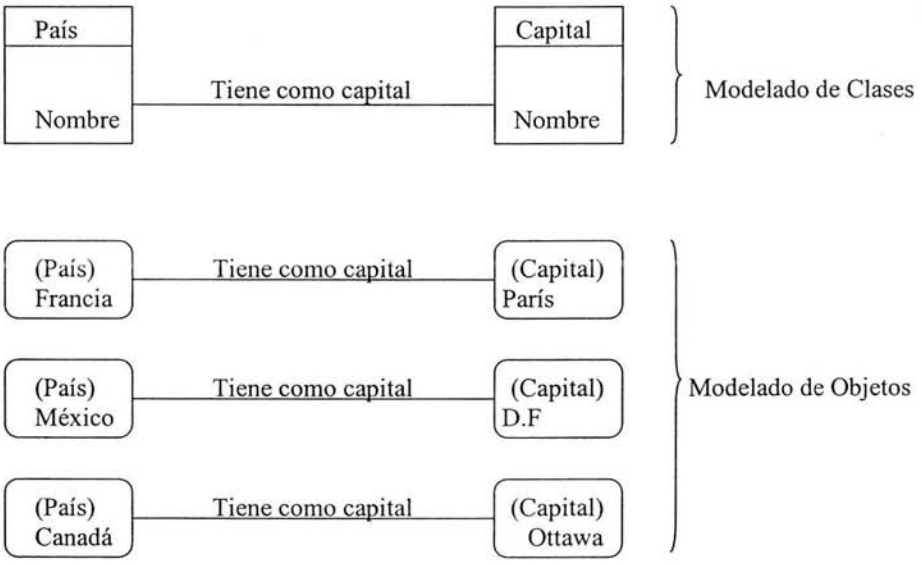
Fig. 2.1.2 Modelado de una clase con sus atributos y operaciones.

Los enlaces y las asociaciones son los medios para establecer las relaciones entre los objetos y las clases. Un enlace es una conexión física conceptual entre instancias de objetos.

Las asociaciones describen grupos de enlaces con estructuras comunes. Estas suelen implementarse en los lenguajes de programación, como un puntero que va desde un objeto hasta otro. Sin embargo, las asociaciones no deben modelarse de esta forma.

Los enlaces muestran la relación entre dos o más objetos. Todas las conexiones entre enlaces deberán modelarse como asociaciones. En la siguiente figura se muestran las asociaciones de tipo uno a uno y sus enlaces correspondientes. Cada país tiene solo una capital y a su vez dada capital pertenece a un solo país.

Fig. 2.1.3 Modelado de asociaciones entre clases.



En el modelado de objetos los enlaces se trazan a través de líneas, como se apreció en la figura anterior.

Para detallar más los enlaces veamos el siguiente ejemplo: En una tarea que se tiene que realizar mediante el diseño asistido por computadora (CAD), se tienen que hallar las redes de conectividad, dada una línea encontrar todas las líneas que la cortan y dado un punto encontrar todas las líneas pasan por él.

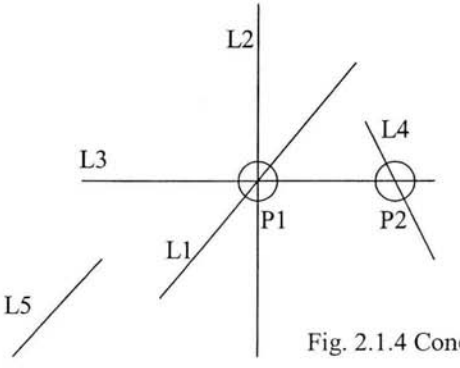
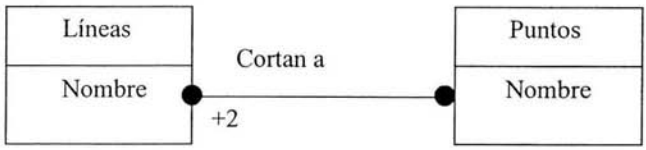


Fig. 2.1.4 Conectividad entre líneas.

Ahora se modelara las clases y objetos de la figura 2.1.5

Diagrama de Clases



Por tanto, el diagrama de objetos quedaría de la siguiente forma:

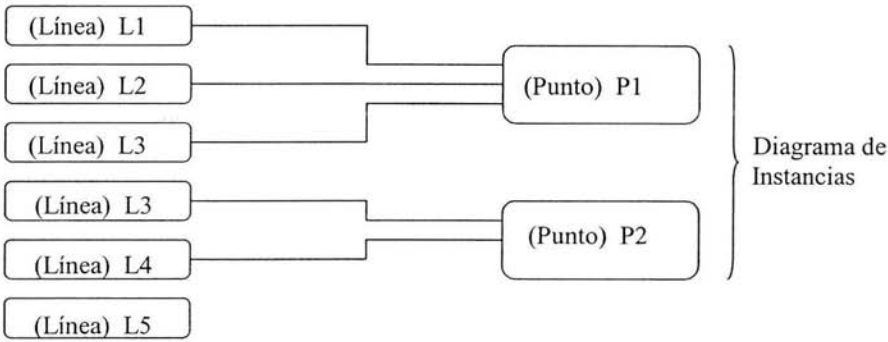


Fig. 2.1.5 Asociaciones y enlaces muchos a uno.

Como vemos, en el enlace entre clases hay dos círculos negros y un "+2", estos símbolos se refieren a la multiplicidad. La multiplicidad especifica el número de instancias de una clase que pueden estar relacionadas con otra instancia de otra clase. En este ejemplo, +2 indica que dos o más líneas interceptan o cortan a un punto y cada línea tiene cero o más puntos de intersección.

Existen también asociaciones ternarias, éstas actúan como unidades mínimas, es decir no se pueden dividir en asociaciones binarias porque perderían información. Un ejemplo de este tipo de asociaciones puede ser el siguiente: Cuando se trabaja en proyectos de desarrollo de software, son necesarios los programadores en la fase de desarrollo final, es por ello que se requiere de un programador que maneje un lenguaje para ese proyecto. Pero además, un programador puede programar en muchos lenguajes que pueden servir para diferentes proyectos. Para entenderlo mejor veamos la siguiente figura.

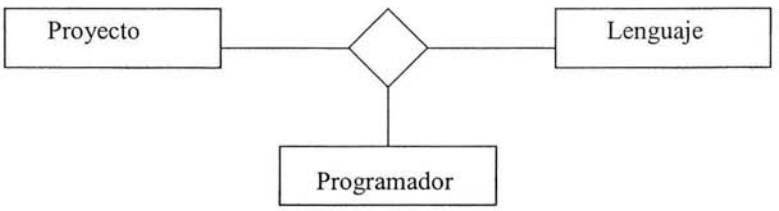


Fig. 2.1.6 Asociaciones ternarias

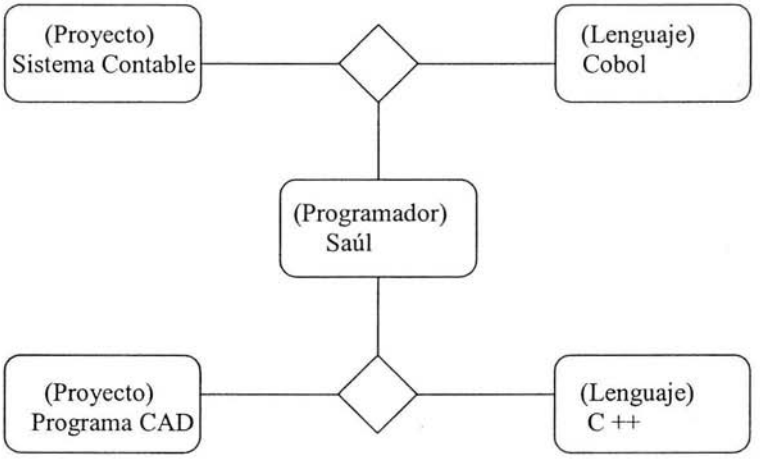


Fig. 2.1.7 Diagrama de Instancias

Vemos que un programador puede participar en diferentes proyectos utilizando diferentes lenguajes de programación, se necesita tres clases: proyecto, lenguaje y programador para modelar correctamente el problema, de esta forma tendremos toda la información que necesitamos, a esto se le llama una asociación ternaria. De no hacer éste tipo de asociaciones se perdería información, o no se tendría la información correcta para poder modelar el problema.

Como ya se vio, la multiplicidad especifica él numero de instancias asociadas a una determinada clase, mediante un número o mediante intervalos, como se describe aquí:

- "1". Indica (uno exactamente)
- "1+". Indica (uno o más)
- "3-5". Indica (entre 3 y 5) incluyendo a ambos
- "2,4,18" Indica (2 4 o bien 18)

También hay terminadores con símbolos especiales:

● Circulo negro. Denota "muchos", o sea, cero o más.
○ Circulo blanco. Indica "opcional", o sea cero o uno.
——— Línea. Sin símbolos de multiplicidad denota una asociación uno a uno.

La multiplicidad va a depender de la forma en que se defina el problema.

2.1.3 Importancia de las Asociaciones

Las asociaciones son estructuras de modelado útiles, tanto para programas como para sistemas de bases de datos y del mundo real, independientemente de cómo estén implementadas. La mayoría de los lenguajes de programación llevan a cabo las asociaciones mediante punteros de objetos, éstos pueden optimizar la implementación pero no se deben utilizar en la fase del diseño. Para conocer más los enlaces y asociaciones, se verán los siguientes conceptos.

Atributos de enlace. Un atributo de enlace es una característica que describe al enlace de una asociación. La notación que TMO utiliza para un atributo de enlace de una asociación, es, un cuadro ligado a la asociación mediante un lazo. Dentro del cuadro pueden aparecer uno o más atributos de enlace. En las asociaciones muchos a muchos generalmente se utilizan los atributos de enlace. Sin duda los atributos deben pertenecer al enlace y no se deben asociar a ningún objeto.

El siguiente ejemplo nos muestra como un archivo es compartido a través de una red, este archivo puede ser de solo lectura o de lectura y escritura, puede ser visible siempre y cuando el usuario conozca el password para poder acceder a él, una vez que haya accesado podrá leerlo o cambiarlo, además uno o más usuarios pueden tener acceso a un mismo archivo. El modelado de este ejemplo quedaría así:

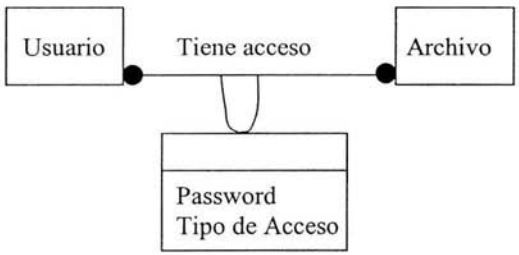


Fig. 2.1.8 Atributos de enlace para asociaciones muchos a muchos.

Usuario	Password	Tipo de acceso	Archivo
Héctor Saúl	***	Lectura	Llopez\Tesis.doc
Lorena	***	Lectura	Llopez\Bdatos.mdb
Melitón	***	Lectura y escritura	Llopez\Tesis.doc

Como vimos en la figura anterior, el password y el tipo de acceso es una propiedad conjunta de archivo y usuario y no puede ser asociada ni a archivo ni a usuario ya que se perdería información. Es por ello que utilizamos a otra clase que contiene información importante y esta enlazada a la clase usuario y a la clase archivo.

Pero los atributos de enlace no solo pueden utilizarse en asociaciones muchos a muchos también en asociaciones uno a muchos y uno a uno, como lo muestra la figura 2.1.9. Una empresa, da trabajo a muchas personas, que reciben un sueldo y cada persona tiene un cargo. Además el jefe evalúa el rendimiento de cada empleado.

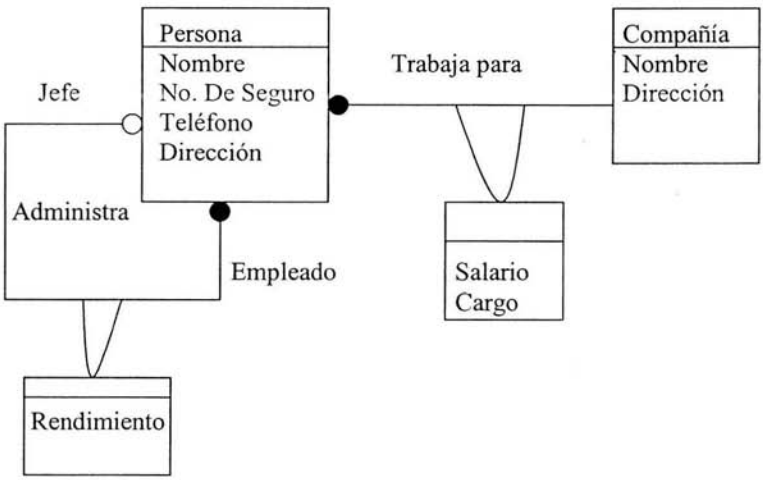


Figura 2.1.9 Asociaciones de enlace uno a muchos.

A un cuando salario y cargo podrían ser atributos de persona, es preferible tenerlos como atributos de enlace ya que ambos dependen de las dos clases como lo son persona y compañía.

En asociaciones ternarias también puede haber atributos de enlace. Para atletas de alto rendimiento como lo es, la carrera de 400 mts. Libres para mujeres, se tiene que realizar un modelo que muestre, que competencia realizó una atleta, en que cede, el año en que participo, además del lugar que ocupo.

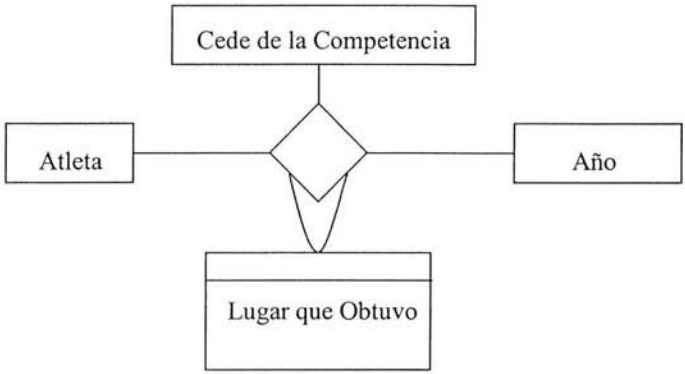


Fig. 2.1.10 Atributos enlaces de en asociaciones ternarias.

Atleta	Cede de la competencia	Año	Lugar Ocupado
Ana Gabriela Guevara	Australia	2000	5to. Lugar
Ana Gabriela Guevara	Francia	2002	1er. Lugar
Ana Gabriela Guevara	España	2002	1er. Lugar

Así como hay atributos de enlaces también puede haber enlace de clases. La siguiente figura representa la autorización que se da a los usuarios para trabajar en una estación de trabajo. La autorización tiene como fin dar privilegios y acceso a los usuarios. Cada usuario tiene un directorio origen para cada estación de trabajo, un usuario puede tener acceso a diferentes estaciones de trabajo y además un mismo archivo puede ser compartido por muchos usuarios. Así, tenemos el siguiente modelo:

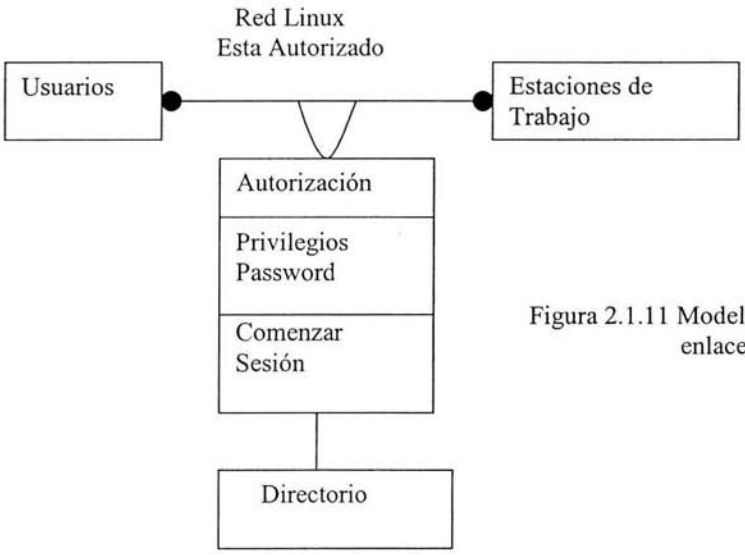


Figura 2.1.11 Modelado de enlace de clases.

Resulta útil modelar las asociaciones como clases, cuando los enlaces pueden participar en asociaciones con otros objetos o cuando están sometidas a operaciones.

2.1.4 Descripción de Roles en las Asociaciones

Cada asociación donde participan dos objetos juegan un rol; distinguimos que papel interpretan cada uno en relación con el otro. Las asociaciones se modelan a través de líneas que unen a los objetos. Aunque el nombre del rol es opcional, nos proporciona información útil y hace más sencillo el entendimiento del modelado de objetos que se este escribiendo. Los nombres del rol se ponen por encima de la línea de asociación a un lado de la clase que describa. Así una persona que trabaja en la ENEP "Acatlán" desempeña diferentes roles.

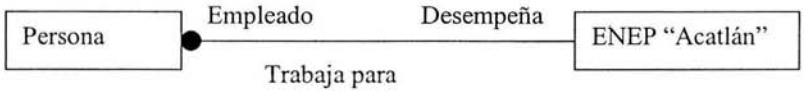


Figura 2.1.12 Descripción de roles en las asociaciones.

Oscar Caballero	Profesor
...	Directora
.....	Secretario Particular

Ningún nombre de rol debe ser igual a un atributo de la clase origen.

2.1.5 Cualificación

Llamamos cualificación a un atributo especial que reduce la multiplicidad de una asociación. Por ejemplo: En una tienda de discos un cliente quiere comprar un disco de Andreas Boccelli que contenga la canción de "Time to say good bye".* El interprete tiene muchos discos cada uno con su respectivo nombre, así, cada nombre de CD pertenece a un sólo disco. De esta forma tendríamos una asociación muchos a uno.

Figura 2.1.13 Asociación uno a muchos

Esta asociación de "muchos a uno" puede cambiar a una asociación uno a uno, de la siguiente forma: Si nosotros conocemos el intérprete y el nombre del disco, tendremos un valor exacto de lo que buscamos. Esto reduce el problema que un interprete tenga varios discos, ya que si decimos el nombre del interprete y el nombre del disco sabremos con exactitud lo que el cliente busca.

*Nota: Hay CD's que traen varios intérpretes, éste es un ejemplo para una asociación específica.

Así la asociación ahora es uno a uno.

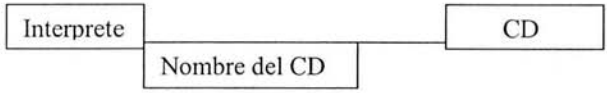


Figura 2.1.14 Cualificación que reduce la asociación muchos a uno, en uno a uno.

De esta manera nombre de CD es un cualificador, se reduce la multiplicidad y se incrementa la visibilidad del problema. Un cualificador se pone dentro de un rectángulo pequeño, en el extremo de la línea de asociación que se encuentra más próximo a la clase que cualifica, ver la figura anterior. La cualificación suele reducir las asociaciones muchos a uno, pero, esto no siempre es posible.

2.1.6 Técnica de Agregación Para el Diseño

La agregación es o son objetos que forman "una parte de algo" o "una parte de todo". Por ejemplo, un nombre con una lista de argumentos y una o más sentencias forman parte de la definición de una función en el lenguaje C, que a su vez forman parte de un programa completo.

La propiedad transitiva es una parte importante en la agregación, ya que si A pertenece a B y B pertenece a C entonces A pertenece a C. Por otra parte, es antisimétrica ya que si A es parte de B, B no es parte de A.

Dentro del modelado orientado a objetos la agregación es aquella que se relaciona con otra clase para formar un ensamblaje, que a su vez puede relacionarse con una clase componente. Entonces una clase con muchas clases relacionadas tiene muchas relaciones de agregación.

Las agregaciones se dibujan igual que las asociaciones, salvo por un pequeño rombo que indica el extremo del ensamblaje que se pone en el extremo de la relación. Por ejemplo, cuando ensamblamos una computadora, partiendo del sistema mínimo que lo compone; un CPU, monitor, teclado y mouse. Comenzando por el CPU, dentro de él hay una tarjeta madre, en donde se conectan las unidades de almacenamiento como pueden ser A (para disckets de $3^{1/2}$) unidad de CD, éstas últimas a su vez se conectan a una tarjeta de sonido, ya ensamblada en la tarjeta madre. También la tarjeta madre tiene muchos circuitos integrados, un procesador y un ventilador, etc. El CPU tiene también un chasis para dar soporte a todos los elementos antes mencionados y su caparazón para cubrirlos. El CPU en sí mismo ya es una clase ensamblada, pero por si solo no forma a la computadora sino necesita de otras clases para formar todo el conjunto que la completarán. Para entender mejor este planteamiento veamos la figura 2.1.14.

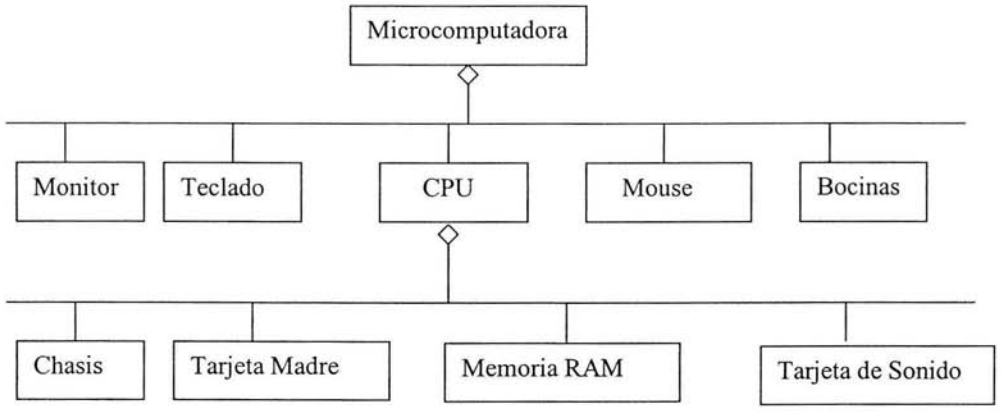


Figura 2.1.15 Ensamblaje de una Microcomputadora utilizando la agregación.

Observe que la agregación puede tener un número ilimitado de niveles.

2.1.7 Generalización y Herencia

Se le llama generalización a la relación que existe en una clase, con un subconjunto de clases de ella misma, es decir, a la relación de una clase con una refinación de ella misma. Lo que se refina lo llamamos clase y lo refinado lo llamamos subclase. Por ejemplo en una casa de materiales se vende material de aceros, que contempla varios objetos, entre ellos, las varillas, el alambrón, alambre, clavos, etc. Cada uno de ellos forman materiales de acero, pero a su vez, tienen características propias que las distinguen de las otras clases. Para modelar este ejemplo se haría lo siguiente:

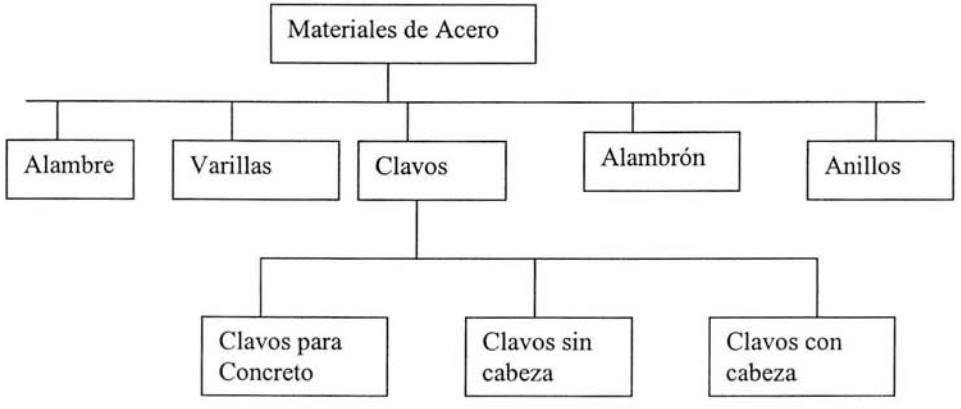


Figura 2.1.16 Generalización de Clases.

Se puede apreciar que la generalización es transitiva en un gran número de niveles. Toda operación que se realice a una clase antecesora puede realizarse a una o más instancias de esa clase. Las operaciones que se heredan resultan de gran valor cuando se requiere reutilizar código.

Se sabe que los lenguajes orientados a objetos dan un fuerte apoyo a la noción de herencia. Los sistemas de bases de datos proporcionan poco o ningún apoyo para la herencia. Actualmente se está tratando de corregir esa situación.

Las palabras generalización, herencia y especialización son aspectos que se requieren para la misma idea, como ya se dijo, la generalización es la relación que existe entre las clases, es decir, la superclase con las subclases; los atributos y operaciones las heredan las subclases de las superclases; la especialización es la refinación que se hace al momento en que una clase se especializa en un uso particular.

2.1.8 Módulos

Los módulos son utilizados para construir agrupamientos de clases, asociaciones y generalizaciones. A partir de los módulos se captura una vista o perspectiva de una situación dada. Por ejemplo la compra, venta y fletes en una tienda de muebles son vistas de un sistema. Los módulos son útiles y necesarios para descomponer el modelo de objetos en segmentos fáciles de manipular. El nombre del módulo suele enumerarse en la parte superior de la hoja, no existe notación especial para los módulos.

2.1.9 Conceptos Avanzados del Modelo OO

La agregación como ya se dijo es una asociación que tiene entre otras características la propiedad transitiva, es decir, las operaciones efectuadas sobre los agregados se extiende hasta sus componentes.

La herencia múltiple permite que una subclase herede características de más de una clase. Una clase que tiene más de una superclase se llama clase mixta.

2.1.9.1 Claves Candidatas

Una clave candidata es un conjunto mínimo de atributos que define de forma única un objeto o enlace.

Dentro de las asociaciones muchos a muchos, se requiere que ambos objetos identifiquen de forma única cada enlace. En este ejemplo se ve de manera sencilla la forma en que utilizamos las claves candidatas. Un alumno toma muchas materias y una materia es tomada por muchos alumnos. La figura 2.1.17 nos muestra cómo se obtienen las claves candidatas.

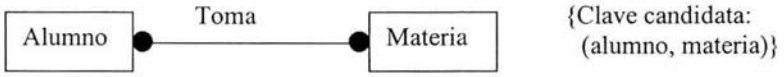


Figura 2.1.17 Claves candidatas en asociaciones muchos a muchos.

En este tipo de notaciones es necesario que la clave candidata este formada por la combinación tanto de alumno como de materia.

Para las asociaciones uno a muchos solo debe haber una clave candidata; que es el objeto en el extremo múltiple. Así una persona solo nace en un estado de la República Mexicana, y en un estado de la República mexicana nacen muchas personas.

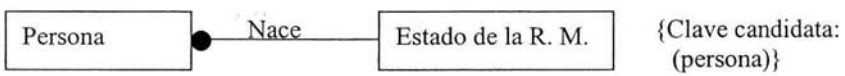


Figura 2.1.18 Clave candidata en asociaciones uno a muchos.

Como se observa el Objeto extremo múltiple es la persona por lo tanto la clave candidata estará formada por ese objeto.

Y por último tenemos asociaciones del tipo uno a uno, dentro de éstas tenemos claves candidatas que pueden ser cualquiera de los objetos. Por ejemplo: Una persona esta registrada en una sola acta de nacimiento y en cada acta de nacimiento esta registrada una persona. De esta manera tenemos la siguiente relación:

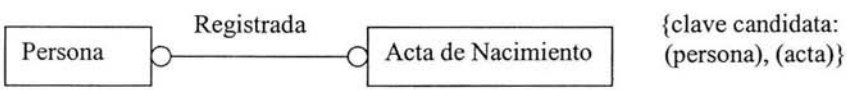


Figura 2.1.19 Claves candidata para asociaciones uno a uno.

Hay asociaciones ternarias en cuyo caso las claves candidatas constan de tres objetos. Por ejemplo: En un proyecto donde se requiere de personas que manejen cierto lenguaje de programación, sería necesario la combinación de las tres clases para tener una información completa de los tres objetos (Proyecto, Persona y Lenguaje). La combinación de solo dos de ellas no sería suficiente, porque no tendríamos toda la información que se requiera de las tres clases, además, sería poco comprensible. La figura 1.2.18 muestra la forma adecuada de escoger las claves candidatas en problemas de este tipo.

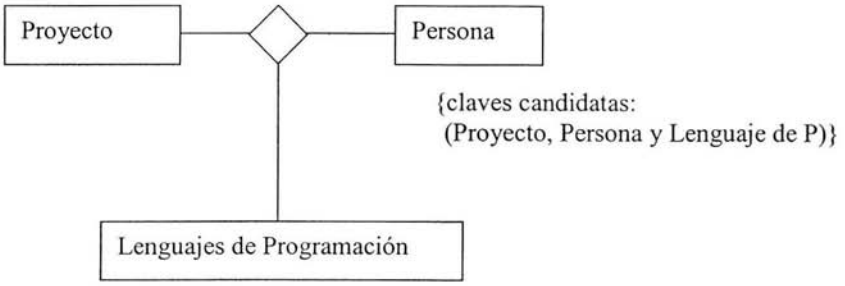


Figura 2.1.20 Claves candidatas para asociaciones ternarias.

Hay claves candidatas ternarias que no necesitan de los tres objetos, por ello debemos apegarnos al problema que queremos resolver.

2.2 Modelado Dinámico

El modelo dinámico es difícil de entender, es por ello que primero se requiere de la estructura del modelo de datos (modelo estático), ya que sobre la base de ese modelo podemos emplear el modelo dinámico del sistema. Se tienen que examinar también, los cambios en los objetos y las relaciones de esos cambios con el tiempo. El modelo dinámico consiste en llevar el control sistemático y secuencial de las operaciones que se realizan en respuesta a estímulos provocados por otros objeto. Este modelo no toma en cuenta la forma en que se realizan las operaciones ni el resultado final de las mismas.

Los conceptos importantes que hay que aprenderse para llevar acabo correctamente el modelado dinámico son los sucesos, que son los estímulos externos y los estados representan los valores de los objetos. Al igual que las clases y sus relaciones con las subclases, los sucesos y estados se pueden modelar en forma jerárquica, para que compartan una estructura y un comportamiento.

2.2.1 Estados

A lo largo del tiempo los objetos se estimulan unos con otros, dando lugar a una serie de cambios en sus estados. Un estado, se le denomina a los atributos y los enlaces mantenidos por un objeto. Un diagrama de estados es una red de estados y sucesos de los objetos a través de las clases y sus relaciones. Los cambios de estado ocurren frecuentemente y los cambios en los objetos son independientes.

2.2.2 Sucesos

Un suceso es el estímulo individual que proviene de un objeto fuente y hace que tenga una serie de reacciones el objeto destino. El suceso es algo que transcurre durante un periodo de tiempo, los sucesos, además, no tienen una duración de tiempo exacta. Es por ello, que un suceso es una transmisión de información directa entre los objetos. El objeto fuente tal vez, espera recibir una respuesta por parte del objeto destino pero la reacción o suceso que realice el segundo objeto dependerá de sí mismo.

El modelo dinámico, como ya dije lleva el control de las acciones que hay que realizarse, los sucesos llevarán información de un objeto a otro, pero hay diferentes sucesos. Por una parte hay sucesos que sólo indican que sucedió algo y hay sucesos que aportan datos o valores al objeto. Los atributos se muestran entre paréntesis después de la clase del suceso. En la figura 2.2.1 se muestran algunos sucesos de ejemplo.

Salidas de vuelo (línea aérea, numero de vuelo, ciudad) Teclea cadena (texto) Llamada al servidor de internet no responde Digito marcado (digito)

Fig. 2.2.1 Clases de sucesos con atributos y sin atributos.

2.2.3 Seguimiento de Sucesos

La secuencia de sucesos y de los objetos que intercambian sucesos, se pueden mostrar mediante escenarios denominados diagramas de diseño de seguimiento de suceso.

Los escenarios son una secuencia de sucesos que se producen en la ejecución de un sistema. En ellos se muestra la forma como se llevan a cabo los sucesos ofreciendo una mejor percepción de las acciones que lleva a cabo ese sistema. Para observar mejor lo que es un escenario veamos la figura 2.2.2

El emisor descuelga el teléfono
Comienza el tono para marcar
El emisor comienza por marcar (5)
Marca numero 2
Marca numero 3
Marca numero 4
Marca numero 5
Marca numero 6
Marca numero 7
Marca numero 8
Señal de llamada
Teléfono comienza a sonar
Espera respuesta de llamada
El destinatario contesta
El teléfono del destinatario deja de sonar
El destinatario Cuelga el teléfono
El emisor cuelga el teléfono

Fig. 2.2.2 Escenario para realizar una llamada telefónica.

Este sistema es relativamente sencillo, lleva a cabo una llamada telefónica. Como se observa, los sucesos se realizan de manera secuencial y ordenada, además tienen que ejecutarse cada paso para llevar a cabo el objetivo del sistema, se tienen que cumplir, en caso contrario el sistema no se realizará la tarea exitosamente.

2.2.4 Seguimiento de Sucesos a Través de Trazas

La secuencia de sucesos se puede representar en un escenario mejorado mediante un diagrama llamado seguimiento traza de sucesos. Este diagrama consiste en mostrar los objetos de manera vertical y los sucesos de manera horizontal la información fluye desde el objeto fuente hasta el objeto receptor. Hay que recordar que en los diagramas muestran la secuencia de sucesos no así el tiempo exacto en que se tienen que realizar.

Ahora se mostrará el ejemplo anterior en la figura 2.2.3 utilizando el diagrama de seguimiento traza de sucesos.

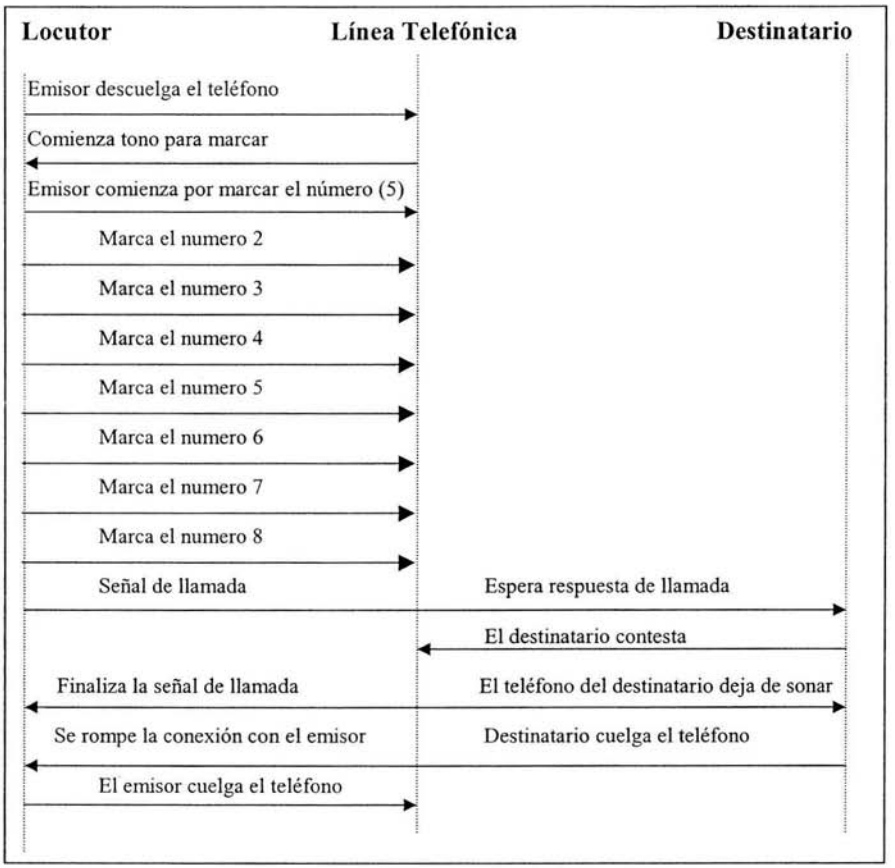


Fig. 2.2.3 Seguimiento de sucesos a través de trazas.

2.2.5 Propiedades de los Estados

Como se señaló antes, los valores de los atributos y los enlaces entre objetos determinan los estados. La respuesta de un objeto a una acción o suceso puede hacer que el objeto cambie de estado. Por ejemplo cuando se descuelga el teléfono para realizar una llamada, el estado del teléfono cambia a ocupado por lo cual el teléfono no puede recibir una llamada hasta que no cuelgue y el estado cambie a libre.

Una cosa importante de los estados, es que aun cuando un estado depende de todas las acciones o sucesos anteriores, se olvidan o quedan ocultos cuando ocurre el cambio de

estado; así por ejemplo, después de que se realizó una llamada, se puede realizar otra y todos los sucesos anteriores ya no importan y no son tomados en cuenta por el objeto.

Los estados suelen estar asociados a ciertas condiciones que determinarán el estado que debe de tener el objeto. Por ejemplo, en un automóvil, si el carro tiene la velocidad de reversa el estado del carro es hacía atrás, si las velocidades están en primera, segunda, tercera o cuarta el estado del carro es hacía adelante. Como vemos los objetos reaccionan a estímulos y su reacción es el cambio de uno a otro estado. Los estados se pueden caracterizar de diferentes maneras como se muestra en la figura 2.2.4 En donde se describen los sucesos y estados de forma escrita.

Estado de inicio (E_1)
Descripción: En este momento se puede introducir un número

Introducir numero 1
Cifra 1 = numero 1
E1 = almacenamiento del primer número

Introducir símbolo de operación
Operación = símbolo de operación
E2 = almacenamiento del símbolo de operación

Introducir número 2
Cifra 2 = numero 2
E3 = almacenamiento del segundo número

Condición que caracteriza al estado

Si la siguiente tecla introducida es " = "
Resultado = Cifra 1 (operación) Cifra 2
Estado = Estado Final(E4)

Si es un símbolo de operación
Pasar a E2

Fig. 2.2.4 Una distinta caracterización de estados para una calculadora.

En este ejemplo hay 4 estados E1, E2, E3 y E4, además, también hay varios sucesos que hacen que el objeto cambie su estado una y otra vez. El estado inicial es E1, esto quiere decir que la calculadora esta preparada para que se le introduzcan dígitos, por lo que al introducirlos estaremos realizando un suceso, mientras ese estado no reciba un símbolo de operación seguirá igual, al introducir el símbolo (+, -, *, /) es otro suceso y ahora sí el

estado pasará a E2, que será el estado que almacene el símbolo de operación, seguido de este estado con el siguiente suceso que deberá de ser número, el estado cambiara a E3 donde almacenará el segundo número, después de esto, habrá dos opciones, una es oprimir el botón que contiene el símbolo "=", de suceder esto el estado pasará a E4 que es el estado final y se verá el resultado, en caso de ser un símbolo de operación se verá el resultado pero pasara al Estado E2 para que almacene el símbolo de operación y espere que sea introducido otro número.

En caso contrario mostrará el resultado anterior pasando a E2 y esperando a que se introduzca un nuevo número.

2.2.6 Diagramas de Estado

Un diagrama de estados relaciona sucesos y estados. Un cambio de estado ocasionado por un suceso se llama transición. Dentro de este diagrama los estados son nodos y a los arcos dirigidos se les llama sucesos. Los estados son representados como cuadros redondeados y dentro de él contiene un nombre opcional. Los sucesos se representan mediante flechas que inician en el estado fuente y son dirigidas al estado destino. Puede haber más de un suceso que salga de un mismo estado, aquí se tomará en cuenta el primer suceso que ocurra.

Ahora, se utilizara el diagrama de estados para graficar el problema de la calculadora. Veamos la siguiente figura 2.2.5

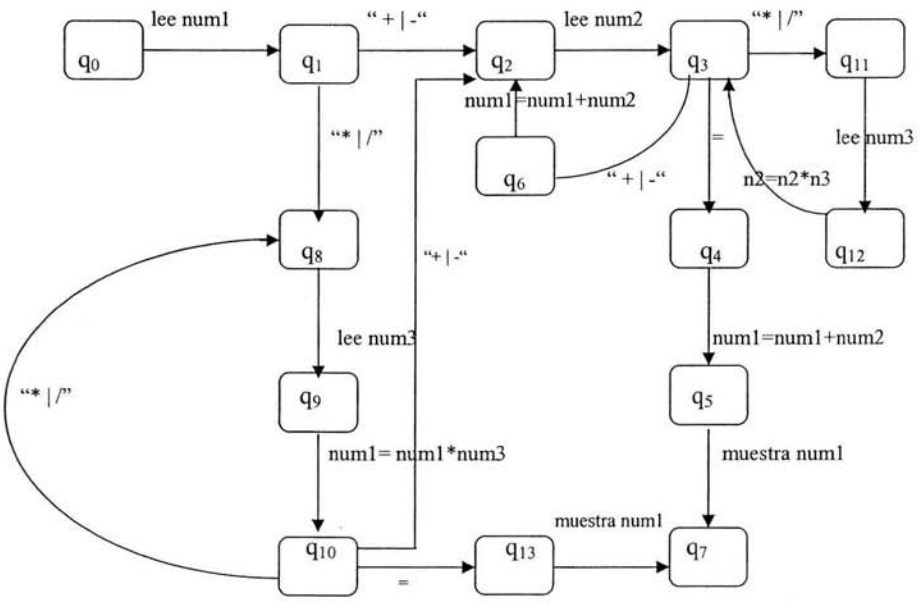


Figura 2.2.5 Diagrama de Estados y Sucesos.

Los diagramas de estados, como se puede apreciar, se pueden hacer lo más detallado que se quiera. En el diagrama anterior, tenemos mucho más estados que con la primera descripción que se hizo de la calculadora. Con ello, los diagramas ofrecen mucho más ventajas en la resolución de problemas, porque llevan a cabo las operaciones que se tienen que realizar de forma sistemática. Este diagrama se puede llevar más fácilmente a la programación.

Los diagramas de estados representan instancias de una sola clase, ya que tienen un mismo comportamiento. Sin embargo, como cada objeto tiene sus propios atributos cada uno de ellos tendrá su propio estado. Por otra parte, los diagramas de estados tienen duración finita y sus respectivos estados iniciales y finales. Al iniciar en un estado se crea el objeto cuando finaliza el estado se destruye el objeto. El estado inicial es representado mediante un círculo negro y el final con un círculo blanco encerrando a uno negro. Así por ejemplo en un juego de ajedrez (muy simplificado), seguimos el distinto estado en que se encuentran los movimientos de las piezas blancas y las negras Fig. 2.2.6

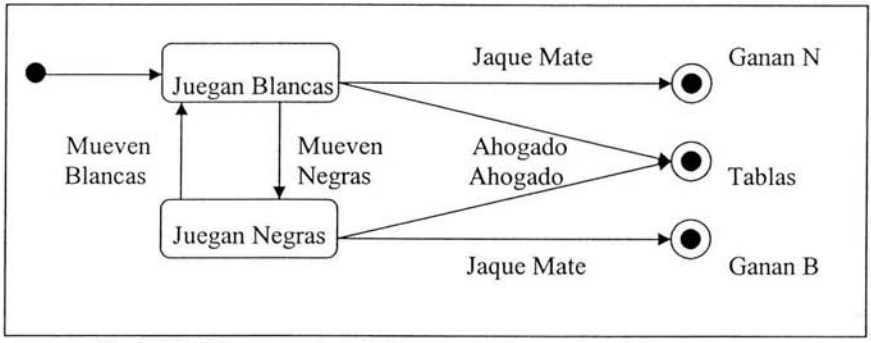


Fig. 2.2.6 Diagrama de estados para un juego de ajedrez.

Como se ha visto el diseño dinámico, es una colección de diagramas que interactúan entre sí a través de sucesos compartidos, muestra una estructura de control del sistema. Además los diagramas dinámicos pueden cumplir condiciones lógicas, que son diferentes a los sucesos, las condiciones son representadas por medio de corchetes, mediante esas condiciones se establece si hay o no una transición de estados.

2.2.7 Control de las Operaciones

Un diagrama de estados tendría muy poca utilidad si sólo describieran sucesos entre estados, con las operaciones asociadas a los estados la herramienta se vuelve mucho más potente. Las operaciones son actividades que se realizan en un cierto tiempo. Entre las actividades se encuentran las operaciones continuas, como por ejemplo el hacer sonar el teléfono, hasta que el destinatario cuelgue o se termine el tiempo de respuesta; es por ello

que una actividad continua se puede ver como una actividad secuencial que tiene una duración indefinida.

Una acción es una operación cuya realización es instantánea y esta asociada con un suceso. Las acciones tienen una duración insignificante comparada con el tiempo en que dura el diagrama de estados hasta llegar a su objetivo. Por ejemplo, cuando colgamos el teléfono es una acción del suceso colgar y es instantánea.

La notación para describir una acción que afecta a una transición es una diagonal "/" junto con el nombre o descripción de esa acción, va después del suceso. Veamos el siguiente ejemplo en la figura 2.2.7

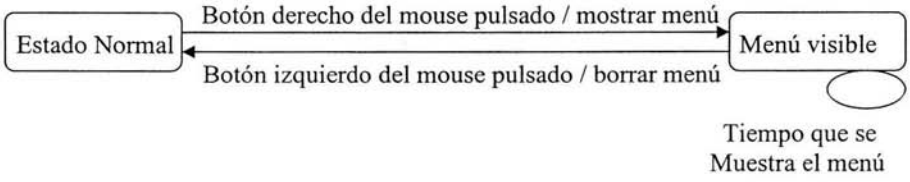


Fig. 2.2.7 Notación que se utiliza para mostrar las acciones que se realizan en los sucesos.

También se pueden estructurar los diagramas de estado, de forma muy parecida a los objetos. Es decir, utilizando la generalización y la agregación de estados. La generalización permite que se estructuren los estados por jerárquica de tal forma que se puedan heredar estructuras y comportamientos a subclases de estados. La agregación permite que los estados se descompongan en componentes más pequeños, y en relación directa con el estado.

2.2.8 Generalización de Estados

Como ya se mencionó, los estados pueden poseer subestados que heredan las transiciones de los superestados. Toda transición o acción que sea aplicable a un estado es aplicable también a los subestados. Los superestados se dibujan como un gran cuadro redondeado que encierran a sus subestados, que a su vez pueden contener otros subestados y se enlazan para formar un anidamiento en los diagramas de estados. El siguiente ejemplo, se trata, de utilizar la generalización en el diagrama de estados para realizar los movimientos en la palanca de velocidades de un automóvil para realizar cambios de velocidades desde primera hasta cuarta y también para la reversa. Ver figura 2.2.8

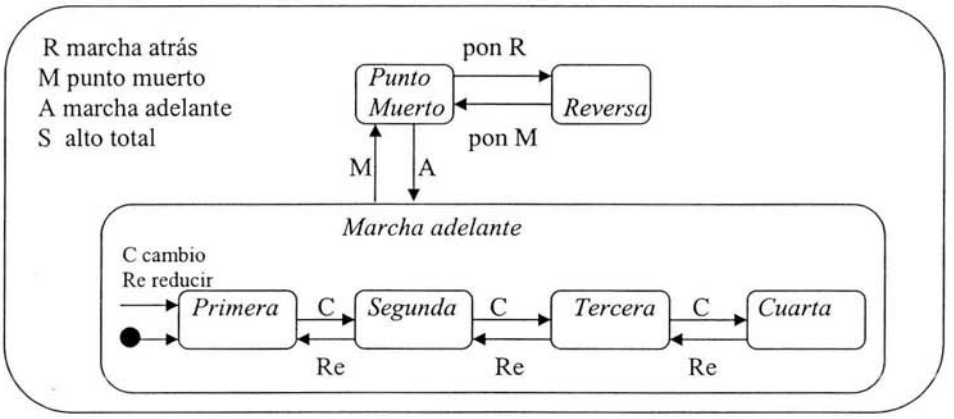


Fig. 2.2.8 Generalización en el diagrama de estados.

En la figura anterior se muestra, como se mueve un carro a partir del punto muerto. Marcha atrás y marcha adelante, cambio y reducir son sucesos que fueron heredadas por una subclase de estados (Marcha adelante). Hay un anidamiento de estados ya que primera, segunda, tercera y cuarta son subestados del superestado marcha adelante.

2.2.9 Concurrencia en el Modelo Dinámico

El modelo dinámico describe un conjunto de objetos concurrentes, cada uno tiene su propio estado y su diagrama de estados. Además los objetos pueden cambiar de estado independientemente. La agregación implica recurrencia, es por ello, que el estado agregado corresponde a los estados combinados de todos los diagramas componentes.

Para entender mejor el concepto de concurrencia, observemos las siguientes figuras 2.2.9A, 2.2.9B y 2.2.9C donde se establece concurrencia entre estados. El ejemplo, es el de un carro que tiene componentes recurrentes de agregación: encendido, transmisión, acelerador y frenos. Cada estado tiene a su vez subestados y es un componente de coche. Además la transición que se hacen entre los subestados de un componente, se realizan en paralelo con todos los demás.

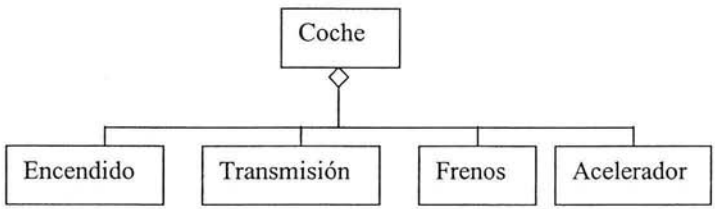


Figura 2.2.9A Agregación de estados.

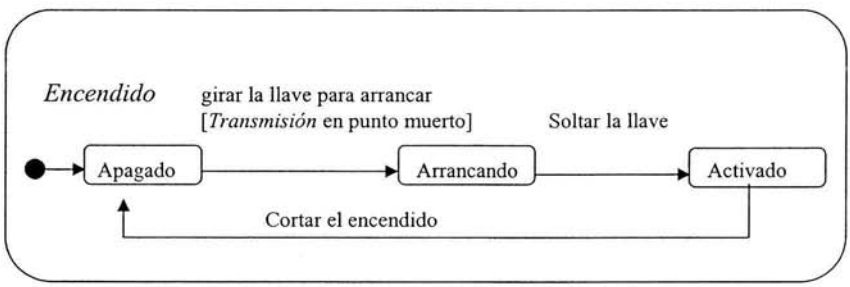


Figura 2.2.9B Diagrama de estados.

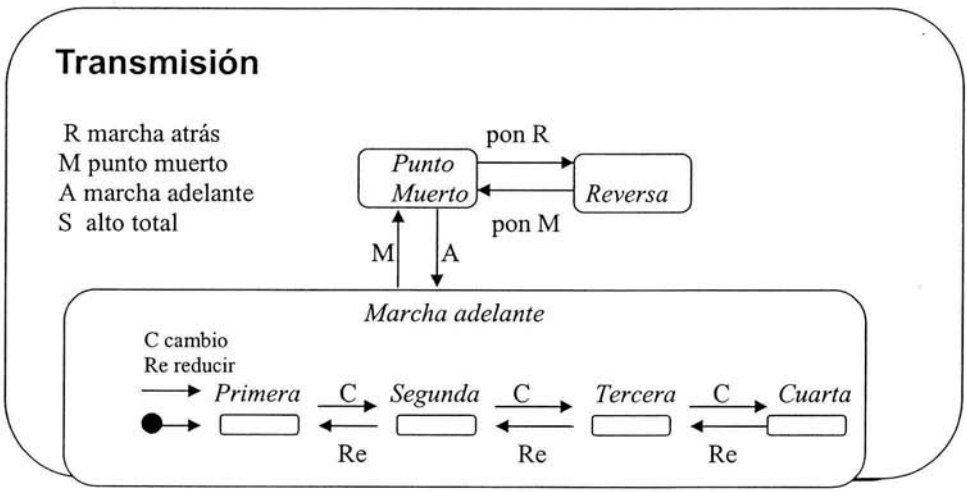


Figura 2.2.9C Concurrencia entre los estados.

La concurrencia del estado de un único objeto surge cuando se puede descomponer el objeto en subconjuntos de atributos o enlaces. En la figura anterior, los estados de los componentes, son casi, pero no del todo independientes. Ya que, el carro no arranca a menos de que la transmisión este en punto muerto.

2.2.10 Acciones de Entrada y Salida

Hay otra alternativa para mostrar las acciones en las transacciones de estados, éstas se pueden asociar a las entradas y salidas de un estado. Veamos ahora la fig. 2.2.10, en ella se mostrará un mecanismo para abrir y cerrar una puerta. Los sucesos que participan son pulsar botón, que abren o cierran la puerta. Cada que el usuario pulsa el botón la puerta cambia de dirección, pero si se esta abriendo tiene que abrirla completamente, luego cerrarla y viceversa. El control para llevar a cabo esta tarea, pone motor hacía arriba para abrir y motor hacía abajo para cerrar que serán acciones efectuadas junto con los sucesos.

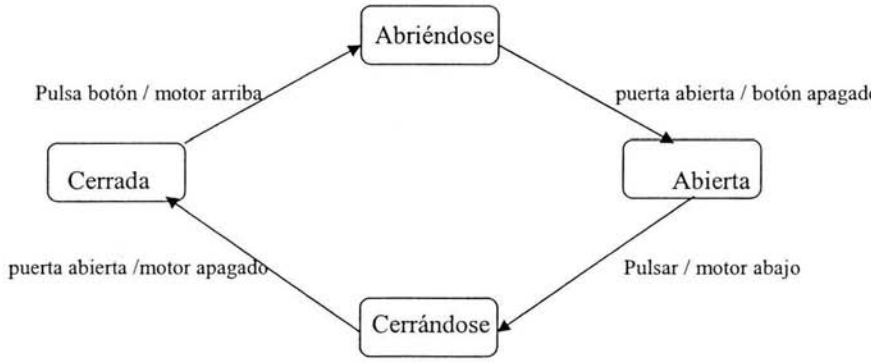


Figura 2.2.10 Acciones de transición de estados.

2.3 Modelado Funcional

El modelado funcional consta de múltiples diagramas de flujo de datos, en donde se muestran los valores de entradas de procesos, operaciones que se realizan, almacenes de datos internos hasta los resultados o la salida final. Es por ello, se dice, que el modelo funcional especifica "lo que sucede", el modelo dinámico especifica "cuando y cómo sucede" y el modelo de objetos nos dice "a que le sucede". El modelo funcional no lleva a cabo el control, ni la estructura de datos, eso pertenece al modelo dinámico y de objetos respectivamente.

2.3.1 Modelos Funcionales

El modelo funcional se encarga de señalar los resultados de los cálculos, sin someterse en detalle a cómo y cuando se deben de realizar. Especifica el significado de las operaciones dentro del modelo de objeto y las acciones dentro del modelo dinámico, así como las restricciones en el modelado de objetos. Las bases de datos suelen tener un modelado funcional trivial, porque su propósito es organizar y almacenar datos y no transformarlos.

Se dice, que un compilador es un cálculo puro. Porque lo único que se necesita es proporcionar un archivo texto con instrucciones en algún lenguaje específico. Una vez que se realice la compilación se obtendrá un archivo objeto que contendrá a la vez instrucciones a nivel máquina. La mecánica de cómo se lleva a cabo la compilación es irrelevante para la aplicación. Para llevar a cabo ésta tarea se requirió como entrada un archivo fuente, la operación en este caso fue la compilación y la salida es un archivo objeto.

2.3.2 Diagramas de Flujo de Datos

Un diagrama de flujo de datos muestra las relaciones entre los valores introducidos a un sistema, los valores que se obtienen y además los datos internos almacenados. En otras palabras, un diagrama de flujo de datos, es un grafo que muestra los valores de los datos que van desde su introducción al sistema, pasando por los procesos que los transforman, hasta llegar a su destino. Cuando se habla de un proceso no se especifica cómo y cuando se realizan las operaciones necesarias para llegar a un resultado, sólo se estipula que el proceso se tiene que realizar.

Un diagrama de flujo de datos no sólo contienen procesos que transforman datos, también contienen flujos de datos, que son los encargados de trasladar los datos a objetos actores, que tanto producen como consumen almacenes de datos. Por lo tanto, un diagrama de flujo de datos, muestra la secuencia en que se tiene que llevar a cabo los procesos, así como los valores externos y los objetos que afectan al cálculo.

Yourdon* establece que los diagramas de flujo de datos consisten en procesos, agregados de datos, flujos y terminadores:

- Los procesos representan las diversas funciones individuales que el sistema lleva a cabo. Las funciones transforman las estradas en salidas.

- Los flujos son las conexiones entre los procesos (funciones del sistema) y representan la información que dichos procesos requieren como entrada o la información que generan como salida.

---

* Análisis Estructurado. EDWARD YOURDON

- Los agregados de datos muestran las colecciones (o agregados) de datos que el sistema debe recordar por un periodo de tiempo. Cuando los diseñadores y los programadores terminan de construir el sistema, los agregados existirán como archivos o bases de datos.

- Los terminadores muestran las entidades externas con las que el sistema se comunica. Típicamente se trata de individuos o grupos de personas (por ejemplo, otro departamento o división dentro de la organización), sistemas de computo externos y organizaciones externas.

Como se verá a lo largo del capitulo, el señor Rambaugh también hace mención a cada uno de los conceptos que maneja Edward Yourdon, sólo que su notación es diferente, además, que lo explota en el paradigma orientado a objetos, mientras Yourdon lo hace para el análisis y diseño estructurado.

2.3.2.1 Procesos

Los procesos son los encargados de transformar los datos, los de más bajo nivel son funciones sin efectos laterales, como puede ser la suma de dos números. Un grafo completo de flujo de datos es un grafo de alto nivel. Los procesos pueden tener efectos laterales sobre todo si se relacionan con almacenes de datos o también pueden ser ocasionados por objetos externos.

La forma en que se ilustran los procesos, es a través de elipses que contienen un nombre apropiado y relacionado a la función que desempeñará. Las entradas a los procesos se especifican mediante flechas de datos así como también las salidas. Se deben especificar los valores de salida a partir de los valores de entrada. En la implementación, los procesos son los métodos de las clases de objetos.

2.3.2.2 Flujo de Datos

Los flujos de datos conectan a los objetos, mediante la salida de uno de ellos y la entrada de otro. Se dibujan mediante flechas entre el productor (objeto fuente) y el consumidor (objeto destino). Dentro de los flujos de datos hay valores agregados, es decir que el mismo flujo de datos se puede descomponer en sus componentes. La combinación de varios componentes en un valor agregado es lo contrario. Veamos la siguiente figura 2.3.1 donde se ve como un valor agregado, sufre una bifurcación.

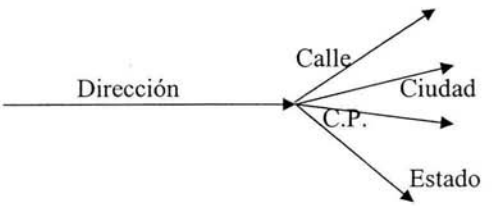


Figura 2.3.1
Flujo de datos agregados.

Cada flujo de datos es un valor necesario para realizar algún cálculo.

2.3.2.3 Actores

Un actor es un objeto que esta activo, produce o consume valores de los flujos de datos. Ellos, están asociados a las entradas y salidas de los flujos de datos. Los actores se representan a través de rectángulos para mostrar que son objetos.

2.3.2.4 Almacenes de datos

Como su nombre lo indica, un almacén de datos, es el encargado de almacenar datos para su utilización posterior, es además, un objeto. A diferencia de los actores los almacenes de datos no generan ningún proceso, solo se dedican a responder a solicitudes, acceso y actualización de datos. Estos almacenes permiten el acceso a los datos en un orden distinto en el que fueron introducidos, ya que se puede acceder a ellos mediante un nombre o una clave índice que los identifica de manera única.

Para reconocer los almacenes de datos, éstos se encuentran dentro de un par de líneas paralelas donde tienen el nombre del almacén. Las flechas de entrada a los almacenes, indican información o u operaciones que modificarán a los datos almacenados, como pueden ser: agregar elementos, borrarlos o actualizarlos. Las flechas de salida indican que un valor ha sido extraído del almacén.

A continuación, se muestran varios ejemplos de cómo operan los almacenes de datos. En el primer ejemplo, se captura la temperatura cada hora y al final del día se obtiene la temperatura máxima y mínima de las que se generaron en el día. Ver figura 2.3.2

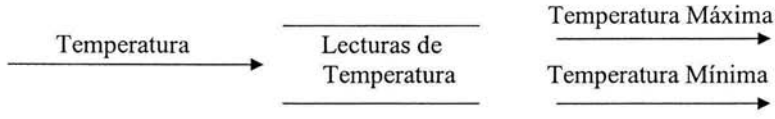


Figura 2.3.2 Entradas y salidas de datos de un almacén.

Las entradas en este ejemplo es la temperatura dada cada hora, y las salidas son la temperatura máxima y la mínima obtenidas durante un día.

El segundo ejemplo, trata de los movimientos que se tienen que realizar en una cuenta bancaria, cuando un cliente realiza un retiro de fondos. Ver la figura 2.3.3.

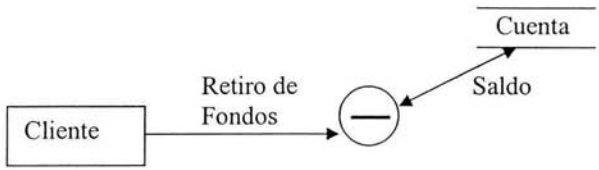


Figura 2.3.3 Utilización de un almacén para una cuenta bancaria.

En la figura anterior, observe la flecha en doble sentido que tiene el almacén, ésta indica que hubo tanto una entrada como una salida en el almacén. Es decir, se accedió a la cuenta del cliente y al momento de hacer la transacción hubo una actualización dentro de la misma cuenta. Estas operaciones son muy frecuentes en las aplicaciones.

La figura 2.3.4 muestra como se pueden obtener los pesos atómicos de los elementos químicos de una tabla periódica. Como se sabe, los pesos de los elementos químicos son constantes, nunca cambian, esto hace que el acceso al almacén (tabla periódica) sea de forma directa, al ser un almacén que contiene valores constantes no tiene porque existir flujos de entrada solo de salida.

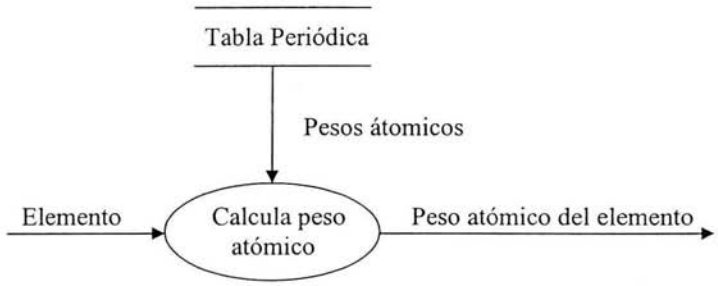


Figura 2.3.4 Acceso a los pesos atómicos en una tabla periódica.

El último ejemplo de un almacén de datos, muestra como los datos dentro de un almacén sufren cambios continuos, ya que en ocasiones pueden variar mucho. La figura 2.3.5 muestra como calcular el costo de un producto, cuando ha subido su precio. Por una parte, el almacén recibe el nombre y el precio para actualizar sus valores dentro del almacén. Para calcular sus precios en la venta al público necesitan saber el nombre del producto y su precio.

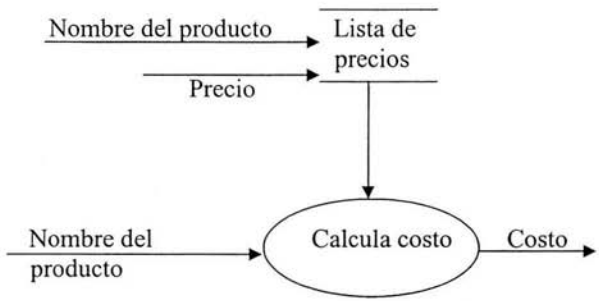


Figura 2.3.5 Almacén que contiene entradas y salidas de datos.

En este ejemplo, es necesario que el almacén contenga entradas, no es como en el anterior donde los valores son constantes, para ello, debe de actualizar continuamente el contenido de los datos.

2.3.2.5 Diagrama de Flujo de Datos Anidados

Un diagrama de flujo de datos es útil para mostrar la funcionalidad del sistema, desde su más alto nivel y luego a través de la descomposición de sus unidades hasta llegar a las funcionalidades más pequeñas. Los diagramas se pueden anidar desde donde el diseñador lo desee de manera arbitraría. Con el anidamiento se detallan las funciones de los componentes, lo que hace que el sistema sea más entendible, llegando a los detalles más internos.

2.3.2.6 Diagramas de Control

Un flujo de control es un valor booleano que decide que proceso se tiene que realizar, no decide ni cómo ni cuando se efectuará ya que ese es trabajo del modelo dinámico, éstos se muestran a través de una línea discontinua, que va desde el proceso donde produce el valor booleano hasta el proceso donde se evaluara la información. Véase la figura 2.3.6 donde se usa el flujo de control. La figura muestra como un cliente retira un monto de su cuenta que tiene en un banco. Para que el pueda acceder a ella tiene que verificar una contraseña antes de llevar a cabo la transacción.

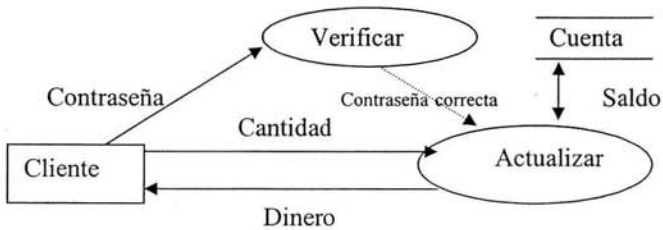


Figura 2.3.6 Uso del flujo de control.

El cliente envía dos datos que son la contraseña y la cantidad que va a retirar, la transacción se efectúa sólo si la contraseña es la correcta. Cuando verifica que la contraseña es la correcta se hace el retiro.

Los flujos de control son útiles en ocasiones pero provocan duplicidad de procesos, ya que quien lleva el control es en realidad el modelo dinámico, así que es conveniente utilizarlos lo menos posible.

En general el modelo funcional muestra "lo que hay que hacer" por parte del sistema. Los procesos que aparecen en el modelo funcional determinan la relación que hay entre los objetos. Un proceso suele implementarse como un método. Un diagrama de flujo de datos es un grafo de procesos, donde intervienen flujos de datos, almacenes de datos y actores. Los actores son objetos consumidores y productores de datos, usan el flujo de datos para enviar o recibir datos. Los almacenes también son objetos, sólo que actúan de forma pasiva, ellos se encargan de almacenar información y proveerla cuando se la pidan.

Los diagramas de flujo de datos son especialmente útiles para mostrar la funcionalidad de alto nivel de un sistema y las transformaciones complejas con múltiples entradas, salidas y valores intermedios.

INGENIERIA DE SOFTWARE ORIENTADA A OBJETOS 3

En este capítulo se mostrará a la TMO como una metodología de ingeniería de software para la construcción del mismo.

La ingeniería del software es un proceso para producir software de forma organizada, empleando una colección de técnicas y convenciones de notación predefinidas.

La TMO se presenta como una serie de pasos, con técnicas y notaciones asociadas a cada paso. Los pasos de producción del software suelen organizarse en un ciclo de vida que consta de varias fases de desarrollo. El ciclo de vida completo del software abarca desde la formulación inicial del problema, pasando por el análisis, diseño, implementación y pruebas de software, hasta una fase operacional durante la cual se llevan a cabo el mantenimiento y las mejoras.*

En éste capitulo se muestran todas las fases del ciclo de vida que abarca la TMO para lograr el objetivo final, que es un software de calidad.

3.1 Análisis Orientado a Objetos

El primer paso de la técnica de modelado de objetos es sin duda, el análisis, que se refiere a la obtención de un modelo, preciso, correcto, comprensible y entendible del mundo real. Es necesario que antes de construir o crear un sistema de software, se conozcan las necesidades, requisitos y entorno en el cual el software va a operar. Es por ello, que el propósito del análisis orientado a objetos es el de modelar el sistema lo más cercano y parecido posible a la realidad para un mayor entendimiento. Durante el análisis orientado a objetos se deben abstraer las características importantes que describan al sistema en el mundo real y dejar al último los detalles menos importantes.

* Modelado y Diseño Orientados a Objetos. J. RUMBAUGH.

3.1.1 Dónde Comenzar

El análisis comienza por la definición del problema, generada por los clientes y, posiblemente, por los desarrolladores. Esta información puede ser incompleta o informal. A un cuando la definición del problema sea incorrecta, debe de servir como base, ya que, poco a poco se tendrán que refinar los detalles, quitar ambigüedades o incongruencias para dejar el planteamiento del problema limpio y claro.

En general los enunciados escritos en un lenguaje natural, suelen ser incompletos e imprecisos, es por ello que el propósito del análisis es el de crear modelos que sean representaciones precisas y concisas del problema, de esta forma, se podrán generar preguntas y dudas a las cuales se les debe de dar solución.

Rumbaugh nos presenta una visión general de cómo se lleva a cabo el proceso de análisis orientado a objetos. Ver Figura 1.3.1

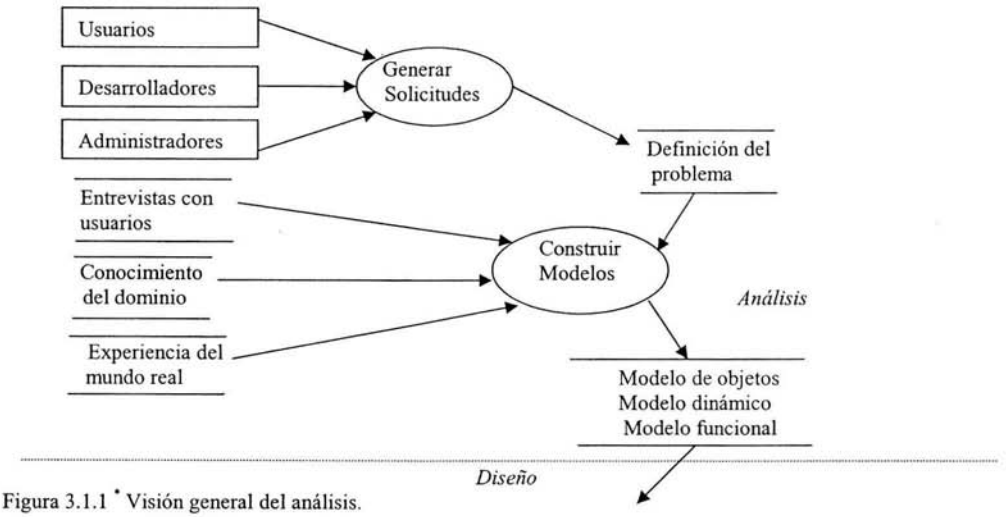


Figura 3.1.1 * Visión general del análisis.

Es importante realizar un riguroso modelado del sistema, ya que así, el desarrollador se enfrentará a problemas de comprensión que se pueden corregir durante la primera etapa de la construcción del software de forma fácil, en vez de corregirlos después de avanzado el trabajo.

Desde la fase del análisis ya podemos emplear los tres modelos de la metodología TMO que son: modelado de objetos, modelado dinámico y modelado funcional.

* Modelado y Diseño Orientado a Objetos. J. RUMBAUGH.

El análisis no siempre se puede realizar de manera exacta en una sola pasada. Los modelos se construyen de forma iterativa, para encontrar errores y corregirlos. Además los modelos de sistemas grandes y complejos se dividen en modelos más pequeños y se integran al último para la comprensión completa del sistema.

3.1.2 Definición del Problema

La definición del problema debe indicar lo que hay que hacer, y no cómo hacerlo. Se deben exponer las necesidades y no las posibles soluciones. El solicitante debe indicar las características obligatorias que se deben de llevar a cabo necesariamente y las que tengan poca relevancia o que no sean obligatorias. Es importante hacer hincapié, en que las definiciones del problema, suelen ser ambiguas, incompletas y con errores en las especificaciones. El analista debe documentar todos los detalles que le parezcan importantes, para realizar un mejor planteamiento. Además, debe de estar en contacto constante con el solicitante, para refinar los requisitos del sistema de tal forma que se ataque al verdadero problema. También, debe cuestionar los requisitos y buscar información importante que sea necesaria para la definición del problema. De no hacerlo, podría ocurrir lo siguiente: hacer exactamente lo que pida el cliente, pero no satisfacer las necesidades reales y lo más probable es que culpen al analista.

3.1.3 El Modelado de Objetos en el Análisis

Una vez que se ha planteado el problema, se tiene que hacer un análisis de requisitos, para ello se utilizan los modelados de la técnica de modelado de objetos. El modelado de objetos representa la estructura estática del sistema, es decir se modelan las clases y sus relaciones.

El modelado de objetos precede al dinámico y al funcional, ya que las estructuras están mejor estructuradas y mejor definidas. La información del modelo de objetos se extrae de la definición del problema, del conocimiento que tenga la persona que maneja las operaciones que se realizan en la empresa o sistema, y del conocimiento que se tenga del mundo real.

El primer paso para realizar el modelo de objetos, es identificar las clases y asociaciones; después se añaden los atributos necesarios para la descripción de las mismas. Lo siguiente es realizar la organización de las clases utilizando la herencia. Los modelos funcional y dinámico se encargarán de describir las operaciones que se tengan que realizar sobre las clases. Como el modelo dinámico modifica a los objetos no se pueden especificar hasta que no se comprenda la funcionalidad del sistema.

Otra recomendación que se hace al analista, es que, debe de escribir las ideas que le lleguen a la mente, para no perder de vista los detalles y así implementarlos en el momento adecuado.

La técnica TMO indica que son necesarios los siguientes pasos para construir el modelo de objetos:

- Identificar los objetos y las clases.
- Preparar un diccionario de datos.
- Identificar las asociaciones entre objetos.
- Organizar y simplificar las clases de los objetos empleando la herencia.
- Verificar que existen las vías de acceso adecuadas para posibles consultas.
- Iterar y refinar el modelo de objetos.
- Agrupar las clases en módulos.

A continuación se analizan estos puntos con mayor profundidad.

3.1.3.1 Identificación de Clases y de Objetos

Lo primero que se tiene que hacer para crear el modelo de objetos, es identificar las clases relevantes del dominio que se tenga del sistema. Entre ellas, se deben identificar las entidades físicas, como: persona, producto, maquinaria, etc. También se deben identificar conceptos como: reservación de asientos, notas de venta, planes de pagos etc. Se deben evitar las estructuras de implementación, sólo se debe atender la estructura estática del sistema.

Hay que escoger las clases candidatas y enumerarlas. Es fundamental poner todas las clases, aún aquellas que en un momento se eliminen o se hagan a un lado, se tienen que escribir todas las que se consideren necesarias. Vea la siguiente figura 3.1.2 para que sepa cual es la secuencia para identificar las clases.

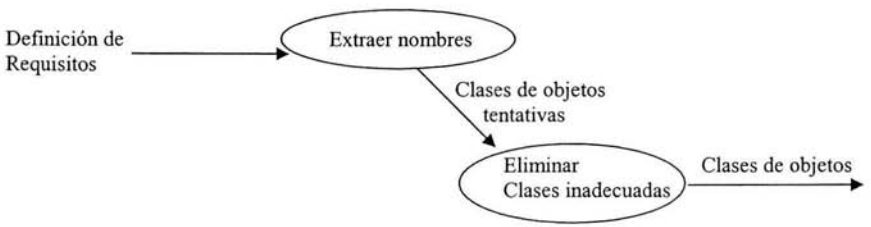


Figura 3.1.2 Identificación de clases de objetos.

Hasta el momento lo más importante ha sido identificar las clases, por ahora no se tomará en cuanta la herencia ni otros conceptos. Por ejemplo, si se analiza un sistema de préstamos de una biblioteca, hay que identificar que se tipo de material que se presta, como pueden ser: libros, revistas, periódicos, videos, cd's, etc. Después se podrán clasificar en diferentes categorías.

3.1.3.1.1 Conservación de las Clases Correctas

Ahora, el siguiente paso es escoger y quedarse con las clases correctas utilizando los siguientes criterios:

- Clases redundantes. Si hay dos o más clases que representan y expresan la misma información sólo se queda con una de ellas y se escoge la que tiene un nombre más descriptivo de la clase a la cual pertenece. Por ejemplo: producto y material pueden ser una misma clase, entonces se debe escoger una y eliminar la otra, se escoge la que vaya de a cuerdo al sistema interpretado.

- Clases irrelevantes. Si una clase no tiene nada que ver con el problema que estamos tratando, sé deshecha, pero es necesario primero ver si la clase es importante o no de acuerdo al contexto que maneje el desarrollador.

- Clases vagas. Toda clase debe representar algo en específico, ya que se pueden tomar clases que no delimitan lo que representan. Por ejemplo, si ponemos una clase llamada sistema, estaremos construyendo una clase vaga, ya que no delimita la estructura de si misma.

- Atributos. No debemos confundir los atributos con los objetos y viceversa. Los atributos son características de los objetos como pueden ser: nombre, apellido, dirección, teléfono, etc.

- Operaciones. Si un nombre describe una operación entonces no es una clase. Sin embargo, toda operación que tenga características propias puede ser una clase. Por ejemplo: Llamada, puede ser una operación de una clase importante cuyos atributos pueden ser, número de llamada, hora, fecha, mensaje.

- Roles. El nombre de una clase debe representar su naturaleza intrínseca y no el del rol o papel que juegue en las relaciones que tenga con otras clases.

3.1.3.2 Preparar un Diccionario de Datos

Para entender con precisión cada palabra e interpretación que se le da a todas las entidades del modelado, es necesario tener un diccionario de datos.

Todas las clases de objetos tienen que describirse claramente, dónde se utiliza y con que otras clases se relaciona, lo mismo se debe hacer con los atributos y operaciones, miembros de cada clase. Pero ¿Qué es un diccionario de datos? Yourdon* hace la siguiente definición de diccionario de datos:

El diccionario de datos es un listado organizado de todos los elementos de datos que son pertinentes para el sistema, con definiciones precisas y rigurosas que permiten que el usuario y el analista del sistema tengan una misma comprensión de las entradas, salidas, componentes de los almacenes y también de los cálculos intermedios.

El formato del diccionario de datos no siempre es el mismo, a continuación se mostrarán aspectos que en general debe de contener un diccionario de datos.

- Nombre. Es el nombre principal que describirá al elemento que se maneje y puede ser, un elemento de datos, de control, de un almacén de datos o de una entidad externa.

- Alias. Otro nombre o sinónimo que se le da al nombre principal.

- Donde se usa y como se usa. Se debe indicar donde y como se usa cada elemento descrito en el diccionario de datos.

- Descripción del contenido. Se describen detalles importantes de los datos.

- Información adicional. Existe también otra información a cerca de los tipos de datos, valores implícitos, restricciones o limitaciones.

Cuando se escoge un alias para un nombre, debe de haber consistencia en ese nombre, es decir, la denominación de un dato en el diccionario de datos también debe de estar dentro del análisis hecho al sistema, además, no se debe repetir el nombre del alias para otro producto, operación o componente que ya se ha definido con anterioridad.

Para saber dónde se usa o cómo se usa, basta con revisar los diagramas de flujo de datos, es una de las grandes ventajas que proporciona el diccionario de datos. Así al hacer referencia a "algo" dentro del sistema, sabremos cómo actúa, a quien afecta y cómo se comporta.

Para empresas grandes, el diccionario de datos crece en complejidad, el mantenimiento no se puede hacer manualmente, afortunadamente hay herramientas CASE que se encargan de hacerlo.

---

* Análisis Estructurado. EDWARD YOURDON.

3.1.3.3 Identificar Asociaciones Entre Objetos

Todas las dependencias entre dos o más clases es una asociación, así como también las clases que hagan referencia a otras clases. Los atributos no hacen alusión a una asociación, es decir, si un empleado labora en una empresa no tiene como atributo compañía, en lugar de eso, utiliza la asociación que puede escribirse *trabaja-para*.

En la mayoría de los casos las asociaciones se relacionan a través de verbos y algún complemento, por ejemplo: conduce, habla con, junto a, parte de, etc. Hay que escoger el más adecuado que describa la asociación a la cual esta haciendo alusión. Las asociaciones se encuentran en la definición del problema.

3.1.3.3.1 Cómo Retener las Asociaciones Correctas

Habrá que descartar algunas asociaciones que estén de más, empleando los siguientes criterios:

- Cuando es necesario eliminar una clase, se eliminan también todas las asociaciones o se restauran en términos de otras clases.

- Asociaciones irrelevantes. Hay que eliminar las asociaciones que se encuentren fuera del domino del problema planteado.

- Acciones. No hay que confundir las acciones con las asociaciones, éstas describen la estructura del sistema mientras que las acciones representan sucesos transitorios.

- Asociaciones derivadas. Hay que quitar las asociaciones derivadas de otras, de no hacerlo así, se tendrán asociaciones redundantes. Un ejemplo de esto, sería la asociación *abuelo*, cuya asociación ya existe y se llama *padre-de*; la primera es una clase redundante ya que se deduce de última asociación.

- Asociaciones con nombre incorrecto. La asociación debe indicar "lo que es" y no "cómo se produce" la relación ni tampoco porqué. El nombre de la asociación es importante ya que hace más comprensible las relaciones entre las clases.

- Nombres de rol. Tienen que escribirse en donde convenga, no es necesario que todos los roles desempeñados por una clase se tengan que escribir. Los nombres de rol describen la asociación entre las clases.

Además, de los puntos anteriores, se deben agregar asociaciones conforme se desarrolle el análisis.

3.1.3.4 Organizar y Simplificar las Clases de los Objetos Empleando la Herencia

El siguiente paso es organizar las clases para crear una estructura común. Se puede realizar de dos formas: ascendente, generalizando los aspectos comunes de clases de las clases

existentes en una superclase o bien, refinando en subclases a partir de las clases existentes en forma descendente. Aunque generalmente se usan ambas simultáneamente.

En la forma ascendente lo que se hace, es buscar atributos, asociaciones y operaciones similares de las clases. Posteriormente se refinan las clases, mediante la herencia y se crean las subclases, poniendo sólo los atributos generalizados, para filtrar los atributos y especializar a las clases.

3.1.3.5 Verificar que Existen las Vías de Acceso Adecuadas

Con el modelado de objetos es fácil observar el tipo de resultados obtenidos en los procesos. Se analizan las entradas, procesos y salidas para saber si el modelo es correcto o habrá que modificarlo.

3.1.3.6 Iteración en el Modelo de Objetos

Es raro que un modelo de objetos sea correcto en una sola pasada. Además, todo el proceso de desarrollo de software es una continua iteración. Si se encuentra un error, se tiene que regresar a la fase anterior para corregirlo. Entre los problemas que se encuentran comúnmente se tienen los siguientes:

- Atributos y operaciones de diferente tipo en una clase.
- Clases que desempeñan dos papeles.
- Operaciones que no tienen clase destino.
- Contar con clases innecesarias.
- Asociaciones que se pasan por alto, sin tomarse en cuenta.
- Asociaciones que son innecesarias.
- Nombres de roles demasiado amplios.

Estos errores suelen ocurrir, cuando no se tiene experiencia en el modelado de objetos. El orden de los pasos se puede intercambiar cuando sea necesario, por ejemplo, se pueden identificar las clases directamente y no anotar las incorrectas, añadirlas al diagrama de objetos con sus asociaciones.

3.1.3.7 Agrupamiento de Clases en Módulos

Ahora, se tienen que agrupar las clases en módulos y hojas. Se dividen los diagramas en hojas, para imprimirlos y estudiarlos. En una hoja deben de estar descritas las asociaciones que representa dicho diagrama. Después se dividirá el sistema en subsistemas que representan varios procesos, como no es posible hacer los diagramas en una sola hoja se crean módulos (conjunto de hojas) que representa dicho subsistema.

3.1.4 El Modelo Dinámico en el Análisis

El modelo dinámico muestra el comportamiento del sistema y de los objetos que varían con el tiempo. Muestra además, los sucesos, que son estímulos que actúan en los objetos y la respuesta de éstos ante esos sucesos. Para llevar a cabo este modelo hay que seguir los siguientes pasos:

- Preparar escenarios de secuencias de iteración.
- Identificar sucesos que actúen entre objetos .
- Construir un diagrama de estados.

3.1.4.1 Preparar Escenarios

Los escenarios se preparan, primero obteniendo información del usuario con respecto al comportamiento del sistema. Deben mostrar los intercambios de información entre el usuario y el sistema durante los procesos.

Los sucesos se producen cuando se intercambia información entre un objeto del sistema y una agente externo, que puede ser, un usuario, un censor, una tarea, etc. Los valores intercambiados son parámetros de los sucesos, por ejemplo, el acceso a un sistema puede requerir de un password el cual es un parámetro.

3.1.4.2 Identificación de Sucesos

Los sucesos se identifican mediante los escenarios, ejemplos de sucesos son: las señales, entradas, decisiones, interrupciones, transiciones y acciones procedentes o destinadas al usuario o a un dispositivo externo. Los escenarios deben de mostrarse como una traza de sucesos, esto es, una lista ordenada de objetos que se encuentran en diferentes columnas de una tabla.

Siguiendo cada columna de la traza se pueden observar los sucesos que afectan directamente a un objeto. Los sucesos que se representen en una traza son los únicos que tienen que aparecer en el diagrama de estados, para ese objeto. En la siguiente figura 3.1.3 se muestra la interacción entre un usuario y un cajero automático que se muestra mediante una traza de sucesos.

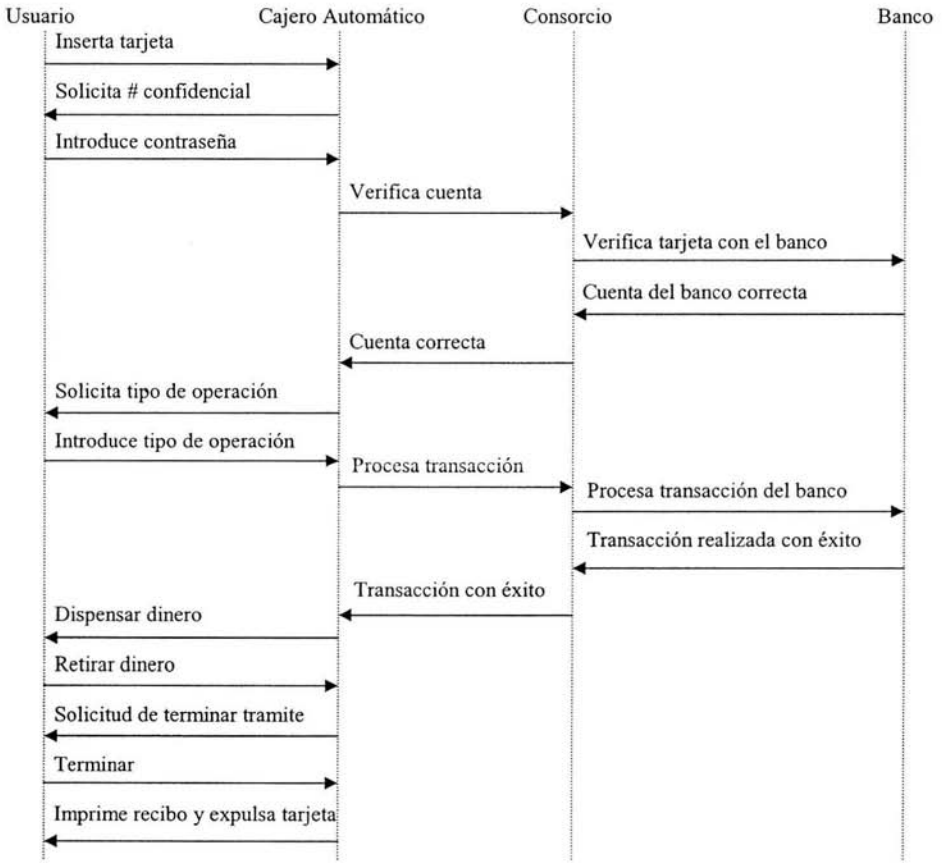


Fig. 3.1.3 Seguimiento de sucesos mediante una traza.

3.1.4.3 Construcción del Diagrama de Estados

En este momento, se debe preparar un diagrama de estados para todas las clases que tengan un comportamiento dinámico no trivial, este diagrama debe indicar los sucesos enviados y recibidos por los objetos.

Para realizar dichos diagramas se comienza con analizar los escenarios o trazas de sucesos relacionados con los diagramas de estados. Como la traza mostrada en la figura 3.1.3 , la cual muestra una interacción considerándose solo los sucesos que afecten a un único objeto. Estos contienen sucesos de entrada y de salida que se encuentran a lo largo de la columna de la traza. El intervalo entre dos sucesos es un estado. El diagrama de estados debe ser una secuencia entre sucesos y estados.

El diagrama de estados concluirá cuando abarque todos los escenarios y maneje todos los sucesos que afecten a un objeto de la clase y cada uno de sus estados. Veamos la siguiente figura 4.1.4 que muestra como se lleva a cabo el diagrama de estados para el ejemplo del cajero automático.

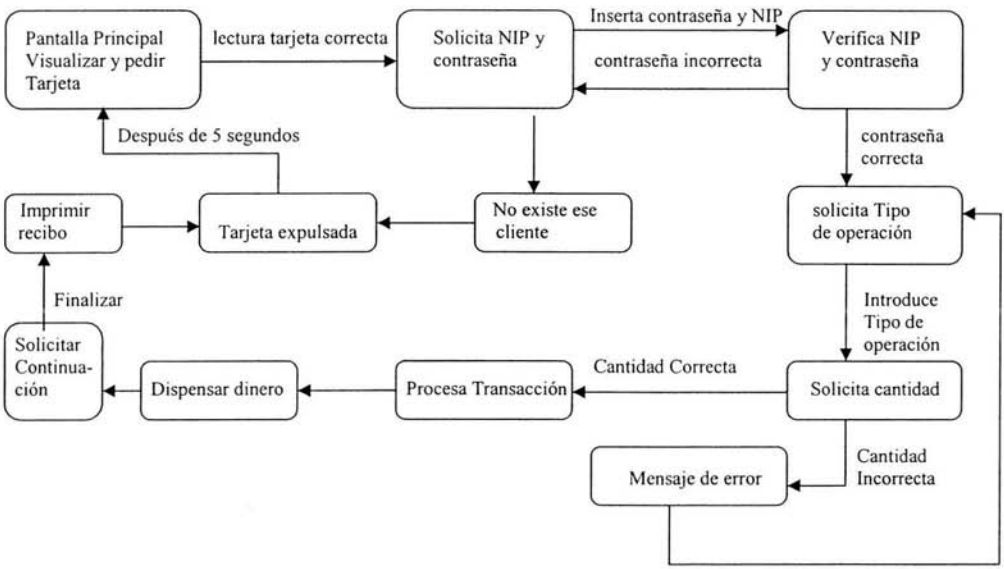


Fig. 3.1.4 Diagrama de estados para el cajero Automático.

El conjunto de diagramas de estados para clases de objetos que tienen un comportamiento dinámico importante es lo que constituye el modelo dinámico de la aplicación.

3.1.5 El Modelo Funcional en el Análisis

El modelo funcional muestra la forma en que se calculan las cosas, sin tomar en cuenta las secuencias, decisiones ni la estructura de los objetos. Cada diagrama de flujo de datos corresponde a un diagrama de estados propuesto en el modelo dinámico. Es por ello, que es mejor realizar el modelo de objetos y dinámico antes de hacer el modelo funcional.

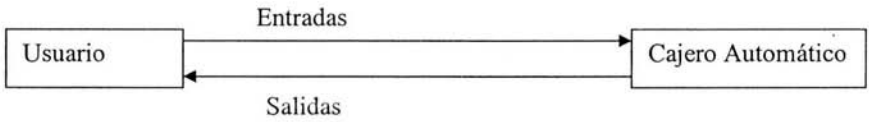
Para construir el modelo funcional es conveniente seguir los siguientes pasos:

- Identificar los valores de entrada y salida.
- Construir diagramas de flujos de datos.

- Describir las funciones.
- Identificar las restricciones.

3.1.5.1 Identificar los Valores de Entrada y Salida

Se enumeran los valores de entrada y de salida, éstos valores son los parámetros de los sucesos que se intercambian entre el sistema y el mundo exterior. Véase la figura 3.1.5 donde se muestran valores de entrada y salida para el cajero automático.



Entradas
1. Tarjeta
2. NIP
3. Password
4. Tipo de Transacción
5. Cantidad

Salidas
1. Dinero
2. Recibo
3. Mensajes

Figura 3.1.5 Valores de entrada y salida de un Cajero Automático.

3.1.5.2 Construcción de Diagramas de Flujo de Datos

El diagrama de flujo de datos muestra la forma en que se deben calcular los valores de salida a partir de los de entrada. Los diagramas de flujo constan de entradas, procesos y salidas. La siguiente figura 3.1.6 muestra la forma como se realiza un diagrama de flujo de datos para el cajero automático.

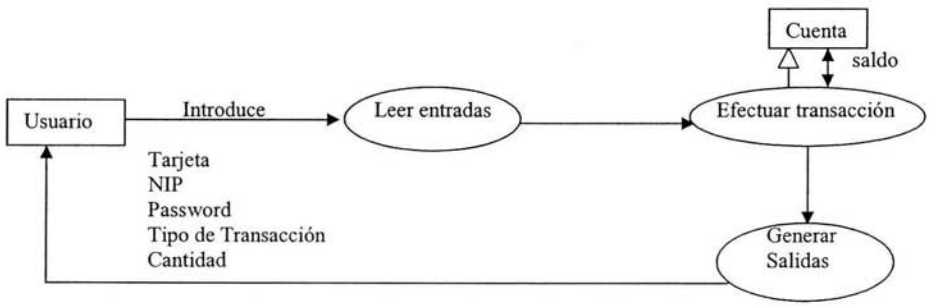


Figura 3.1.6 Diagrama de flujo de datos para el cajero automático.

Se puede expandir el proceso efectuar transacción para mostrar de forma más clara ese proceso, quedando de la siguiente forma figura 3.1.7

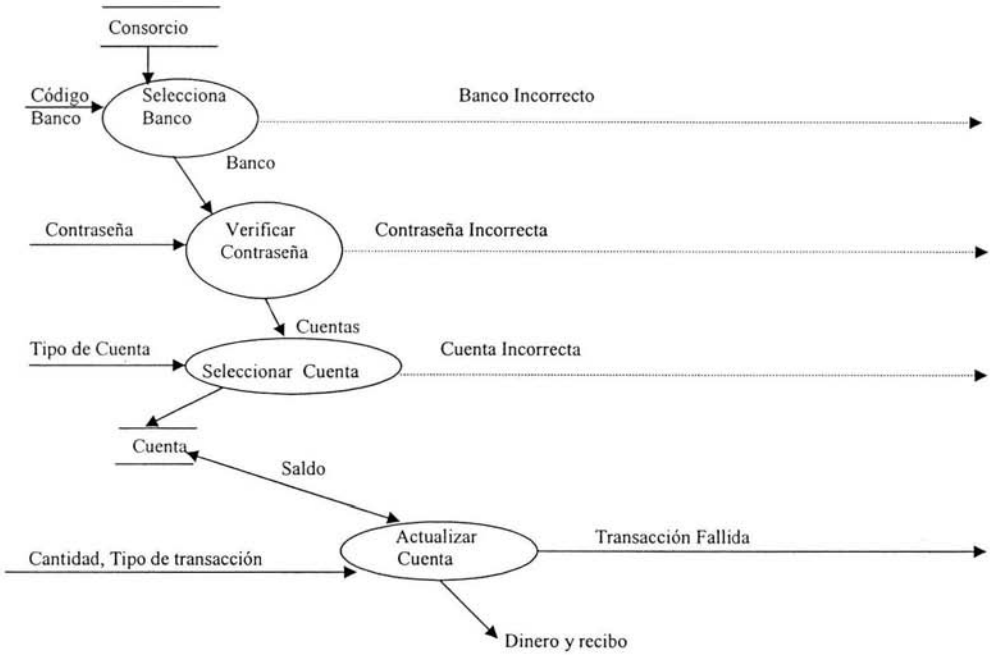


Figura 3.1.7 Modelo funcional del cajero automático.

Las funciones de decisión se pueden mostrar en el diagrama de flujo de datos, pero sus salidas son señales de control, que se indican mediante flechas de salidas discontinuas.

3.1.5.3 Descripción de Funciones

Cuando se ha realizado un diagrama de flujos suficientemente detallado, hay que describir las funciones que se realizan en él, que puede ser en lenguaje natural, pseudocódigo, tablas de decisión o en algún formato adecuado a cada función. Las descripciones de estas funciones se pueden realizar de dos maneras: declarativa o de procedimientos. Las declaraciones declarativas especifican las relaciones entre los valores de entrada y los de salida. Las descripciones procedimentales especifican la función dando un algoritmo para calcularla. Las descripciones declarativas son preferibles a las procedimentales, ya que no implican una implementación, pero si la descripción procedimental es más fácil de escribir se debe de utilizar.

3.1.5.4 Especificación de Restricciones

Es importante identificar las restricciones de los objetos. En todo sistema hay restricciones que deben de tomarse en cuenta, de lo contrario podría acarrear consecuencias irreparables. Por ejemplo: en el cajero automático no se deben admitir saldos negativos, o alguna cantidad que exceda a las especificaciones de algún banco. Para tener un control sobre estas restricciones, el analista debe agregar las restricciones tanto en el modelo funcional como en el dinámico para completar las especificaciones.

En la siguiente figura 3.1.8 se pueden ver algunas especificaciones que se pueden implementar en el cajero automático.

Actualizar Cuenta (cuenta, cantidad, tipo de transacción)⟶ dinero, recibo y mensaje.

Si la cantidad que se intenta retirar supera al saldo disponible, se rechaza la transacción y no hay entrega de dinero.

Si la cantidad que se intenta retirar, no supera al saldo disponible ni al impuesto por el banco, cargar el importe y realizar retiro especificado.

Si la transacción es un ingreso, recoger el efectivo y abonar el importe.

Cuando se ha realizado una transacción el cajero debe de proveer un recibo con la hora, fecha, tipo de transacción, importe y saldo actual.

Figura 3.1.8 Restricciones del cajero automático.

Puede haber especificaciones más detalladas, por lo que el analista debe prever en esta fase del análisis toda restricción significativa del sistema.

Casi todos los modelos de análisis requieren de más de una revisión para finalizar. Para comprender un problema con todas las implicaciones que se manejan alrededor de él, es preciso revisar repetidamente el análisis, preparando una aproximación al modelo. Con las iteraciones se tendrá una mejor comprensión del sistema o subsistema que se este desarrollando. Por último, se tiene que revisar el análisis con el solicitante y los expertos de la aplicación, para dejar bien claro el problema a resolver.

3.2 Diseño Orientado a Objetos

El diseño Orientado a Objetos, transforma el modelo del análisis en un modelo de diseño que sirve como anteproyecto para el desarrollo de software, este diseño se apoya en cuatro conceptos importantes que son: abstracción, ocultación de información, independencia funcional y modularidad.

3.2.1 Diseño de Sistemas Orientado a Objetos

Para entender mejor el diseño de sistemas, vea la siguiente pirámide propuesta por Pressman*, las cuales divide en las siguientes capas:

Figura 3.2.1 Diseño orientada a objetos.

La capa del subsistema. Aquí se representan las partes (subsistemas) que se interrelacionan para formar el sistema.

La capa de las clases y objetos, representa las jerarquías de clases que permiten crear el sistema utilizando generalizaciones y especializaciones.

La capa de diseño de mensajes. Detalla la forma en que los objetos se comunican entre sí. Esta etapa establece las interfaces internas y externas para el sistema.

La capa de responsabilidades. Contiene las estructuras de datos, algoritmos y operaciones de cada objeto.

La técnica de modelado de objetos abarca el diseño de dos niveles diferentes de abstracción. El diseño de sistemas y el diseño de objetos que describe de forma detallada el objeto de forma individual.

---

* Ingeniería de Software un enfoque práctico. ROGER S. PRESSMAN.

El diseño que propone James Rumbaugh se basa en los siguientes puntos:

- Realizar un diseño de sistemas.
  Particionar el modelo de análisis en subsistemas.
  Identificar la concurrencia dictada por el problema.
  Asignar subsistemas a procesadores y tareas.
  Elegir una estrategia básica para la implementación de la gestión de datos.

- Identificar recursos globales y los mecanismos de control requeridos para acceder a ellos.
  Diseñar un mecanismo de control apropiado para el sistema.
  Considerar cómo se deben manejar las condiciones limite.
  Revisar y considerar intercambios.

- Conducir un diseño de objetos.
  Seleccionar operaciones del modelo de análisis.
  Definir algoritmos para cada operación.
  Seleccionar las estructuras de datos apropiadas para los algoritmos.
  Definir todas las clases internas.
  Revisar la organización de clases para optimizar el acceso a los datos y mejorar el rendimiento computacional.
  Diseñar los atributos de la clase.

- Implementar los mecanismos de control definidos en el diseño del sistema.

- Ajustar la estructura de una clase a una herencia fuerte.

- Diseñar el intercambio de mensajes para implementar relaciones entre objetos(asociaciones).

- Empaquetar las clases y asociaciones en módulos.

3.2.1.1 Descomposición de un Sistema en Subsistemas

En todas las aplicaciones, salvo en las más pequeñas, el primer paso del sistema consiste en dividir el sistema en pequeños componentes. Cada componente principal es un pequeño sistema llamado subsistema.

Un subsistema, no es ni una función, ni un objeto, sino un conjunto de clases, asociaciones, operaciones, sucesos y restricciones interrelacionados que tienen una interfaz bien definida; normalmente un subsistema se identifica por los servicios que proporciona. Por tanto, cada subsistema se puede diseñar entonces, independientemente sin afectar a los demás.

Entonces al definir subsistemas, éstos deben de seguir los siguientes criterios de diseño, según Pressman* .

- El subsistema debe poseer una interfaz bien definida a través de la cual ocurre toda la comunicación con el resto del sistema.
- Con excepción de un pequeño número de <<clases de comunicación>>, las clases de un subsistemas deben colaborar únicamente con otras clases dentro del subsistema.
- El número de subsistemas debe mantenerse pequeño.
- Los subsistemas pueden dividirse internamente para ayudar a reducir la complejidad.

3.2.1.2 Identificación de la Concurrencia

El modelo dinámico nos permite identificar la concurrencia. La concurrencia ocurre cuando dos o más objetos pueden recibir sucesos al mismo tiempo sin interactuar. Si el flujo de sucesos indica que sólo hay un objeto activo en ese momento, sólo se sigue un hilo de control, éste dura hasta que llegue respuesta al objeto que ha enviado el mensaje.

3.2.1.3 Tareas Concurrentes

Se podría pensar que todos los objetos son concurrentes, sin embargo, en realidad muchos objetos en un sistema son independientes. Dentro de los diagramas de estados encontramos un hilo de control, en donde solamente esta activo en cada instante. Los hilos de control son implementados como tareas en los sistemas computacionales. Por ejemplo, en el cajero automático, mientras el banco verifica la cuenta para realizar la transacción el cajero en ese momento no realiza ninguna actividad ya que esta esperando respuesta del objeto banco para poder interactuar de nuevo con el usuario, en este momento esta llevando en conjunto una sola tarea.

3.2.1.4 Gestión de Datos

Generalmente los almacenes de datos pueden contener estructuras de datos, archivos y bases de datos implementadas en memoria o en dispositivos de almacenamiento secundario. Los tipos de almacenamiento de datos proporcionan diferentes ventajas así como desventajas que se deben analizar para elegir cual es la más adecuada a la aplicación que se esta llevando a cabo.

Los archivos son una forma de almacenamiento de datos barata, sencilla y permanente. Una de sus desventajas es que las operaciones que se hacen en un archivo son

* Roger S. Pressman. Ingeniería de software un enfoque práctico.

de bajo nivel por lo que la aplicación debe agregar código adicional para proporcionar un nivel de abstracción adecuado y fácil de entender.

Las bases de datos son gestionadas por un sistema manejador de base de datos (SMBD), son otro tipo de almacenamiento. Las bases de datos son muy potentes y hacen que las aplicaciones se puedan transportar a diferentes plataformas y sistemas operativos. Una desventaja de los SMBD es que poseen una interfaz compleja y se integran con alguna dificultad con los lenguajes de programación. ¿Cómo decidir si los datos deben pertenecer a un archivo o a una base de datos?

Datos que deben pertenecer a una base de datos formal.

- Datos que deban ser accedidos por múltiples usuarios.
- Datos que deban ser usados eficientemente por un SMBD.
- Datos que se deban transportar por diferentes sistemas operativos y plataformas de hardware.
- Datos que tengan que ser accedidos por diferentes aplicaciones.

Líneas que suelen caracterizar a los datos que están dentro de un archivo.

- Datos que sean muy grandes y difíciles de estructurar para un SMBD (tal como un archivo gráfico bmp).
- Datos que sean voluminosos y contengan poca información, tal como ficheros de archivos o registros históricos.
- Datos que se manejen por un corto periodo de tiempo y se desechen después.

Las ventajas y desventajas que podemos encontrar en una base de datos se presenta por los siguientes puntos.

Una base de datos representa una gran ventaja comparada con simples archivos, a continuación se presentan, estas ventajas:

- En una base de datos hay muchas características que permiten, el acceso compartido entre múltiples usuarios o entre múltiples aplicaciones, la distribución de datos, la integridad de los datos, la extensibilidad y transacciones programadas por distribuidores de SMBD.

- Existe sólo una interfaz para todas las aplicaciones. Cada aplicación accede a un subconjunto de datos que necesita e ignora el resto.

- Un lenguaje estándar para acceder a los datos es el SQL es admitido para la mayoría de los sistemas relacionales para la administración de la Base de datos.

Así como las bases de datos tienen ventajas muy marcadas con respecto a los archivos, también tienen desventajas que a continuación se presentan.

- Costos temporales en su rendimiento. Solo algunos SMBD pueden realizar muchas transacciones (más de 50) por segundo. Para grandes aplicaciones los diseñadores deben de trabajar en conjunto con los proveedores de SMBD para optimizar el rendimiento de las base de datos junto con las aplicaciones.

- Poca funcionalidad para aplicaciones avanzadas. Los SMBD están enfocados, la mayoría de las veces, para aplicaciones de negocios que tienen que manejar una gran cantidad de datos con una estructura sencilla. Sin embargo, hay aplicaciones que necesitan realizar operaciones con tipos de datos complejos, difíciles de implementar, en donde las bases de datos no son muy útiles.

Con las nuevas bases de datos orientadas a objetos éstas desventajas poco a poco van desapareciendo.

3.2.2 Manejo de Recursos Globales

El diseñador tiene que identificar los recursos globales para determinar como se tendrá acceso a ellos. Dentro de los recursos globales que se deben de tomar en cuenta, están las unidades de disco, torre de discos, procesador, líneas de comunicación, pantallas de estación de trabajo, etc. Si el recurso es físico se puede autocontrolar así mismo, pero si es lógico, tal como una base de datos se corren grandes riesgos o peligros de que el acceso produzca conflictos en el acceso compartido. Es por ello, que todo recurso global debe contener un "objeto guardián" que controle los accesos a cada recurso. A su vez todo acceso a los recursos deben de pasar por el "objeto guardián".

3.2.2.1 Condiciones Límite

El diseñador debe de tomar en cuenta las condiciones límite o de contorno del sistema, que son: Iniciación, terminación y fallos, a continuación se explica la importancia de cada uno de ellos.

Iniciación. El inicio del sistema debe permanecer en un estado estacionario, hay que iniciar constantes, parámetros, tareas, objetos guardianes y tal vez una jerarquía de clases.

Terminación. La terminación es mucho más fácil de realizar que la iniciación, ya que cualquier objeto se puede simplemente liberar, así como los recursos externos que se hayan utilizado. Es importante en un sistema concurrente que cada tarea indique la terminación de las demás.

Fallos. Un fallo es la terminación no planeada de un sistema*. Los fallos pueden surgir por errores del usuario, agotamiento de recursos del sistema o de algún fallo externo.

* Modelado y Diseño Orientado a Objetos. J. RUMBAUGH.

Se deben planear los fallos de manera que el sistema quede tan limpio como sea posible, grabando o imprimiendo la mayor cantidad de información acerca del fallo.

3.2.2.2 Establecimiento de Prioridades

Durante el diseño se deben de establecer las prioridades necesarias para llevar a cabo los objetivos de la aplicación trazados en el análisis. El diseñador debe determinar los criterios de prioridad para que se tomen las decisiones correctamente. El éxito o fracaso del producto final dependerá de que los objetivos hayan sido seleccionados correctamente. En caso de que no se establezcan prioridades que abarquen todo el sistema, éste podría desperdiciar recursos, ya que en ocasiones el programador se olvida de los objetivos reales y se obsesiona con la "eficiencia" de la aplicación, a un cuando ésta no sea realmente importante.

3.2.3 Diseño de Objetos

Los objetos que se descubren durante el análisis sirven como base para el diseñador quien debe escoger las distintas formas de implementarlos con el fin de minimizar el tiempo de ejecución, memoria y costo. Ahora, las operaciones especificadas en el análisis deben de expresarse en algoritmos. Las clases, atributos y asociaciones se deben de representar en estructuras de datos bien definidas. Se debe exagerar en el diseño del sistema ya que esto traerá como consecuencia la facilidad de implementación y de mantenimiento.

El diseño de objetos, es un proceso de adición de detalles y de toma de decisiones para la implementación. El diseño de objetos es principalmente un proceso de refinamiento y detalles finos.

3.2.3.1 Diseño de Algoritmos

Toda operación especificada en el modelo de análisis debe tener su propio algoritmo. Un algoritmo se puede dividir en operaciones más sencillas sucesivamente, hasta que se puedan implementar directamente.

Para diseñar los algoritmos se deben de tomar en cuenta los siguientes puntos:*

- Seleccionar algoritmos que reduzcan costos al implementar las operaciones.
- Seleccionar estructuras de datos adecuadas a los algoritmos.
- Definir nuevas clases y operaciones internas necesarias.
- Asignar las operaciones a las clases adecuadas.

Si se requiere escoger entre varios algoritmos alternativos se debe de tomar en cuenta lo siguiente:

* Modelado y Diseño Orientado a Objetos. J. RUMBAUGH.

Complejidad computacional. Es esencial pensar en la complejidad del algoritmo, es decir, la forma en que aumenta el tiempo de ejecución, ya que puede ser constante, lineal, cuadrático o exponencial. Se deben de identificar los puntos clave para maximizar la eficiencia de la aplicación y dejar a un lado los pequeños factores de eficiencia.

Facilidad de implementación y comprensibilidad. Vale la pena perder un poco de rendimiento en operaciones no críticas e implementar algoritmos sencillos y fáciles de comprender.

Flexibilidad. La mayoría de los programas cambian o se amplían con el tiempo, es por ello, que un programa rígido puede resultar obsoleto a mediano plazo e ineficiente en poco tiempo.

Un algoritmo óptimo suele hacer a un lado la legibilidad y facilidad, pero hace más eficiente el algoritmo dentro del programa. En cambio, un algoritmo sencillo es más fácil de implementar, pero, ineficiente al momento de ejecutarse. Esto sólo se sabrá en el momento de trabajar con la aplicación.

3.2.3.2 Selección de Estructuras de Datos

La selección de algoritmos implica la selección de estructuras de datos en las que se aplican. Entre las estructuras de datos se encuentran las matrices, listas, pilas, colas, conjuntos, diccionarios, árboles y las variaciones de cada uno de ellos. La mayoría de los lenguajes orientados a objetos contienen estructuras genéricas que forman parte de su biblioteca de clases predefinidas.

3.2.3.3 Definición de Clases y Operaciones Internas

Es posible que durante la introducción de algoritmos se necesiten nuevas clases para que almacenen resultados intermedios y se tengan que incrementar, a su vez, nuevas operaciones que lleven a cabo ciertas tareas. El diseñador debe tener presente, que se pueden agregar nuevas clases que no fueron tomadas en cuenta durante el análisis, éstas serán elementos que se utilizarán en la implementación.

3.2.4 Implementación de Control

El diseñador debe establecer una estrategia para construir el modelo dinámico visto en el capitulo 2.2, vea los siguientes puntos:

- Utilizar dentro del programa diagramas de estados.
- Implementación de una máquina de estados.

3.2.4.1 El Estado Como Posición Dentro del Programa

La posición dentro del control en el programa define, en que estado se encuentra el programa. Una vez leída la entrada hay una transición de estados, dependiendo del acceso de datos que haya recibido la clase. Para llevar a cabo el control de un diagrama de estados dentro del programa se sigue estos puntos:

- Se comienza con el estado inicial, se realiza el recorrido a través de una secuencia de sucesos, éstos pasan a ser sentencias dentro del programa y se llega a la conclusión del proceso.

- Se identifican los recorridos que salgan del estado inicial y vuelvan a regresar a él, aquí se presentan condiciones para el programa.

- Se identifican recorridos en los cuales, salen de un estado y después de un número de transiciones vuelven al mismo estado, éstos en realidad son bucles dentro del programa.

Observe el siguiente modelo de estados aplicado al cajero automático. La figura 3.2.2 muestra el modelo dinámico y el pseudocódigo derivado del anterior.

Pseudocódigo

```
Hacer para siempre
Mostrar pantalla principal
Leer tarjeta
Repetir
        Pedir login y password
        Leer login y password
        Verificar cuenta
Hasta que la cuenta sea correcta
Repetir
        Preguntar tipo de transacción
        Leer clase
        Preguntar cantidad
        Leer cantidad
        Comenzar Transacción
Hasta que la transacción sea correcta
Dispensar efectivo
Imprimir recibo
Expulsar tarjeta
Regresara a la pantalla principal
```

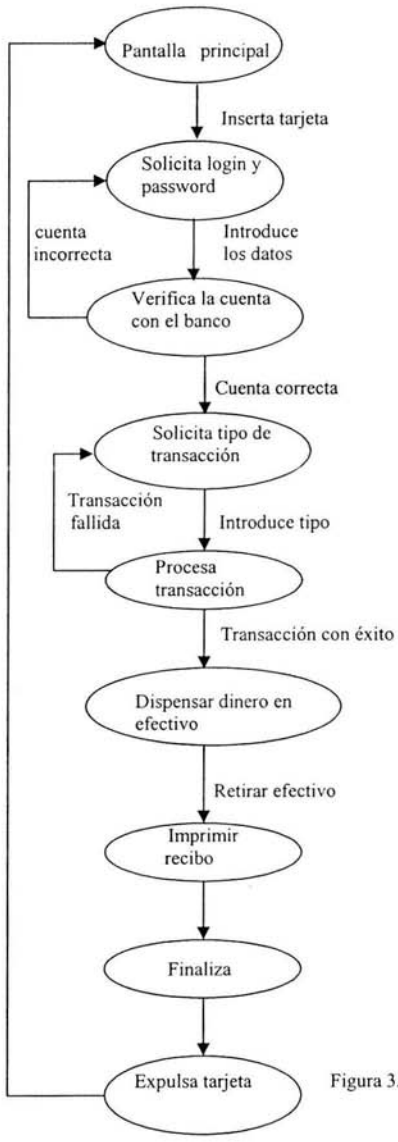


Figura 3.2.2 Control mediante una secuencia de estados del cajero automático.

Se puede apreciar que el hilo de control con el modelo de estados, es fácil de llevar. El estado de inicio es la pantalla principal, cada suceso ocasiona un cambio de estado; donde hay dos o más sucesos que salen de un mismo estado se implementa una sentencia condicional, si un estado dentro de un número de pasos vuelve a tener el control es que hay un bucle o un proceso que se repite.

3.2.4.2 Motor de Máquinas de Estado

Otra aproximación más directa para implementar el control es utilizando las máquinas de estado. La cual se representa a través de una tabla de transiciones y acciones proporcionadas por la aplicación. Un analizador sintáctico, tal como el yacc o lex* de unix, produce una máquina de estados implícita para implementar una interfaz de usuario.

3.2.5 Ajuste en la Herencia

A medida que progresa el diseño de objetos, se pueden incrementar tanto objetos como operaciones, es preciso que el diseñador deba:

- Reorganizar y ajustar las clases y operaciones para incrementar la herencia.
- Abstraer el comportamiento común de los grupos de clases.

3.2.5.1 Reorganizar las Clases y Operaciones

Hay ocasiones en que una operación se define en varias clases y es fácil de heredar de una clase común; frecuentemente las operaciones de las clases son parecidas pero no idénticas. Mediante pequeñas modificaciones en las clases y operaciones, se puede hacer coincidir, de tal forma que se herede una sola operación.

3.2.5.2 Abstracción del Comportamiento Común

Es necesario volver a checar la herencia en la etapa del diseño, ya que durante el análisis se pueden omitir detalles comunes entre clases. Para ello, es necesario crear superclases que implementan características compartidas, dejando las especializadas para las subclases. Cuando se finalice el proyecto se deben recoger, documentar y generalizar las clases potencialmente reutilizables para poder usarlas en proyectos futuros.

La herencia es un mecanismo que permite emplear la generalización, con la cual, el comportamiento de una clase es compartido por todas sus subclases. El programador, a veces, declara diferentes clases que tienen un mismo comportamiento definiendo más clases de las necesarias.

3.2.6 Diseño de Asociaciones

Las asociaciones son el "pegamento" del modelo de objetos y proporcionan las vías de acceso entre las entidades . Durante el diseño es necesario realizar una estrategia adecuada para implementar dichas asociaciones.

* Modelado y Diseño Orientado a Objetos. J. RUMBAUGH.

3.2.6.1 Analizando el Recorrido de las Asociaciones

Las asociaciones son inherentemente bidireccionales, es decir, la relación va de un objeto a otro indistintamente. En algunas ocasiones, solo recorre en una sola dirección, lo que puede simplificar su implementación. Si una asociación recorre en una sola dirección, es posible implementarla mediante un puntero. Si la multiplicidad es de uno a muchos se implementa un conjunto de punteros. En cambio las asociaciones bidireccionales pueden implementarse de dos formas, la primera, consiste en implementar los atributos en ambas direcciones y cuando se requiere realizar una búsqueda ésta se hace hacía atrás. La segunda consiste en implementar, también, los atributos en ambas direcciones para tener un acceso lo más rápido y poder realizar actualizaciones eficientemente.

3.2.7 Empaquetamiento Físico

Los lenguajes orientados a objetos tienen distintos niveles de empaquetamiento. El empaquetamiento implica los asuntos siguientes:

- Ocultar información interna a los ojos de los usuarios.
- Coherencia de entidades.
- Construcción de módulos físicos.

3.2.7.1 Ocultamiento de la Información

Uno de los objetivos de diseñar es tratar a las clases como "cajas negras", en la cual la interfaz es pública pero los detalles internos quedan ocultos. El diseñador decide, a que atributos se pueden acceder desde el exterior de la clase.

Durante el diseño se debe limitar a las operaciones de las clases para que solo interactué con un número limitado de clases. Los siguientes puntos ayudan a limitar el ámbito de las operaciones proporcionadas por Rumbaugh.

- Hay que asignar a cada clase la responsabilidad de llevar a cabo las operaciones y de proporcionar la información que corresponda a ella.

- Hay que invocar una operación para acceder a los atributos que pertenezcan a un objeto de otra clase.

- Hay que definir las interfaces con el mayor nivel de abstracción posible.

3.2.8 Construcción de Módulos

Los módulos deben realizarse con el fin de mostrar interfaces mínimas y bien definidas. Las interfaces entre dos módulos constan de las relaciones y asociaciones entre las

diferentes clases y las operaciones que acceden a las clases cruzando los límites de los módulos.

Una clase o módulo son coherentes si están organizados congruentemente y todas sus partes encajan, llevando a cabo un objetivo común. Es por ello, que las clases no deben servir para demasiadas cosas a la vez. Si resultan muy complicadas, se pueden descomponer empleando la generalización o agregación; porque los segmentos más pequeños tienen mayor posibilidad de ser reutilizables que los segmentos grandes y complicados.

3.3 Implementación

La escritura del código es un extensión del diseño. Se supone, que la escritura del código debe ser sencilla, casi mecánica, porque ya se debieron tomar las decisiones difíciles durante el diseño. Será de gran importancia el código del programa para su mantenimiento y extensión.

3.3.1 Implementación a Través de un Lenguaje de Programación

La mayoría de los lenguajes de programación son capaces de implementar tres conceptos en el software: estructura de datos, flujo dinámico de control y transformación funcional.

Las sentencias utilizadas para declarar las estructuras de datos están en lenguajes orientados a procedimientos. El flujo de control se puede expresar mediante procedimientos (condicionales, bucles y llamadas), o bien, sin procedimientos (reglas, restricciones, tablas y máquinas de estados).

Las transformaciones funcionales se utilizan en términos de operadores primitivos del lenguaje así como en forma de llamadas a subprogramas.

La forma más fácil de implementar el diseño orientado a objetos es utilizando un lenguaje orientado a objetos. Sin embargo, el diseño orientado a objetos resulta beneficioso para los lenguajes procedimentales o no orientados a objetos; ya que en todo caso, el código de los lenguajes de programación terminan convirtiéndose en código máquina. El uso de los lenguajes no orientados a objetos exige al programador, mayor cuidado y disciplina a la hora de mantener la estructura de datos y el programador no puede obtener ayuda para este tipo de problemas.

De cualquier forma, los lenguajes orientados a objetos y los que no los son, pueden ser utilizados bien o mal. Los programas orientados a objetos mejoran muchísimo en expresividad, aunque los programadores descuidados pueden aumentar la falta de claridad.

3.3.2 Implementación de los Sistemas con una Base de Datos

Cuando el problema requiere de un acceso persistente de datos, las bases de datos pueden ser la forma más correcta para su implementación. El objetivo principal de las bases de datos son: las estructuras y restricciones a las que están sujetos los datos. Las ordenes de los SMBD afectan al conjunto de datos dentro de una base de datos. Las bases de datos ofrecen operaciones concurrentes de datos por parte de distintos usuarios como parte fundamental de su estructura.

Algunos sistemas de bases de datos orientadas a objetos intentan integrar un lenguaje orientado a objetos con una base de datos como un todo.

3.3.3 Programación Orientada a Objetos

Los programas que siguen correctamente al diseño tienen más probabilidades de ser correctos, reutilizables, extensibles y fáciles de corregir. A continuación se describen éstos conceptos más detalladamente.

Reutilización. El software reutilizable reduce costos, codificación y comprobación, porque reduce la cantidad de código y simplifica la comprensión.

En cuanto a las clases hay dos tipos de reutilización, la primera es, compartir el código recién escrito en un proyecto y volver a utilizar ése código en proyectos futuros. Otra forma de reutilizar el código es dentro del mismo proyecto, utilizando las capacidades del lenguaje de programación y mediante el descubrimiento de partes redundantes dentro en el diseño.

La reutilización es una parte importante y una gran ventaja en la programación orientada a objetos, es por ello, que requiere inversión de tiempo y esfuerzo para su implementación.

Extensibilidad. La mayor parte del software se extiende en formas que los realizadores originales no lo habrían pensado. Pero esta extensión se puede ajustar y no afectar al software si se tiene cuidado en los siguientes puntos:

Encapsulamiento de clases. La clase estará encapsulada si su herencia interna esta oculta a las demás clases. Solo los métodos de la clase deberían de acceder a los datos en su implementación.

No recorrer múltiples enlaces o métodos. Lo correcto es solo acceder a las clases que están relacionadas directamente con otras clases y evitar acceder a otras mediante un enlace a partir de una tercera.

Distinguir las clases públicas de las privadas. Las operaciones y datos públicos son visibles fuera de las clases y tienen interfaces publicas. Hay que tener cuidado al definir las operaciones públicas ya que pueden acarrear errores muy costosos, por el acceso a los datos

o métodos que tienen las demás clases. En las clases privadas las operaciones y datos quedan limitados a la misma clase.

3.3.4 Robustez

A un método se le llama robusto cuando no falla ni siquiera cuando se introducen parámetros incorrectos. Hay que buscar la eficiencia cuando se escriben los métodos. El software debe de tener cuidado contra entradas incorrectas por parte del usuario; una entrada incorrecta nunca debe producir la caída de un programa. Se debe proteger al programa de errores fatales que puedan traer consecuencias irreparables.

Otro punto que hay que cuidar, es el de optimizar el programa después de que funcione, ya que en ocasiones, el programador pone mucha atención y esfuerzo para optimizar partes del código que se ejecutan muy pocas veces.

3.3.5 Programación a Gran Escala

En los proyectos de sistemas grandes se requiere de programas enormes y complicados, entonces se necesita de emplear a un grupo de programadores. En tales proyectos, es necesario e indispensable la comunicación humana para manejar métodos comprensibles, legibles, utilizar los mismos nombres y conceptos que en el modelo de objetos antes de programar en forma prematura. Además hay que documentar las clases y los objetos correctamente.

3.4 Pruebas Orientadas a Objetos

Es muy importante realizar pruebas a los sistemas orientados a objetos de forma adecuada. Algunos aspectos que se tienen que considerar se refieren a:

- Detectar los errores en los modelos de análisis y diseño orientados a objetos.
- Realizar pruebas a unidades de integración.
- Hacer un diseño de casos de prueba para software orientado a objetos.

En cada etapa, pueden realizarse pruebas a los modelos para encontrar errores antes de que se propaguen.

3.4.1 Pruebas en el Análisis y Diseño Orientado a Objetos

Un modelo esta sintáctica y semánticamente correctos, cuando éste refleja al mundo real de forma exacta, y para saber si realmente lo refleja, el ingeniero de software debe examinar las jerarquías de clases y la definición de cada una de ellas para buscar omisiones y ambigüedades.

Otro punto importante cuando se realizan las pruebas, es la consistencia y ésta se observa analizando las relaciones entre las entidades en el modelo. Hay que poner atención en cada clase con sus respectivas conexiones, la responsabilidad, colaboración y aportación dentro del modelo.

También se deben analizar los diseños tanto del sistema como de los objetos. En el primero, se pueden revisar los subsistemas que componen al sistema, y las asignaciones de cada clase a un subsistema. Mientras el modelo de objetos, sirve para revisar los detalles de las clases, además, de presentar el intercambio de mensajes que existe entre ellas.

3.4.2 Pruebas de Unidades Orientadas a Objetos

Las pruebas de software comienzan con pruebas a <<pequeñas escalas>>, es por ello, que se incia con las pruebas de unidad y finalizan con la validación del sistema. En el software orientado a objetos el concepto de unidad se refiere a la clase. Por eso, el concepto importante en la definición de una clase es la encapsulación. En vez de módulos, la unidad mínima a probar es la clase encapsulada, ya que en ella, se empaquetan tanto atributos como operaciones que los manipulan. Las pruebas para clases orientadas a objetos esta dirigida a las operaciones encapsuladas, los cambios de estado y el comportamiento de cada clase. La validación de software orientado a objetos se centra en las acciones visibles al usuario y las salidas que el sistema le ofrece.

3.4.3 Diseño de Casos de Prueba

Las pruebas de casos de prueba se encuentra en la etapa de la definición del problema. Estas pruebas se centran en el diseño de secuencias apropiadas de operaciones para determinar los estados de la clase.

3.4.4 Pruebas en Fallos

Las pruebas basadas en fallos comienza en el modelo de análisis, su objetivo es, el detectar el mayor número de errores posibles, mediante una planificación preliminar en el diseño y el código.

3.4.5 Diseño de Pruebas de los Escenarios

Este tipo de pruebas se pueden utilizar esencialmente para corregir: (1) especificaciones incorrectas, y (2) interacciones entre subsistemas. Cuando ocurren errores asociados con especificaciones incorrectas, es que el producto no hace lo que el cliente desea. Tal vez hace cosas incorrectas o se omitieron algunas cosas importantes. Los errores en las

interacciones entre subsistemas aparece cuando el comportamiento de un subsistema propicia que otro subsistema falle.

Como se pudo apreciar en los puntos anteriores, el objetivo de la prueba del software es el detectar el mayor número de errores con el menor esfuerzo posible. Las pruebas de software orientado a objetos debe incluir la revisión de los modelos de análisis y diseño. También se tienen que realizar pruebas en la programación, para hacer esto, se diseñan secuencias de pruebas para asegurar que se realicen operaciones adecuadas al sistema.*

* Ingeniería de Software un enfoque práctico. ROGER S. PRESSMAN.

EJEMPLO DE LA TMO PARA EL DESARROLLO DE SOFTWARE 4

En este capítulo se presenta un ejemplo para mostrar el uso de la metodología, Técnica de Modelado de Objetos para el desarrollo de software. La aplicación que se emplea, es para una casa de materiales para construcción, la cual, actualmente lleva sus procedimientos administrativos manualmente.

Como los procesos son grandes y complejos, el ejemplo se enfoca solamente a desarrollar una aplicación para llevar un control de inventario del sistema. Esto, es solo una parte del sistema, ya que no se toman en cuenta los gastos administrativos, de papelería o nomina etcétera, el alumno, puede tomar alguno de estos subsistemas para desarrollarlo si así los desea.

4.1 Análisis del Sistema

De acuerdo a las entrevistas hechas con el dueño, él tiene algunos problemas para controlar el inventario de su negocio. A continuación se especifican estos problemas y se tratará de identificar el problema.

Diariamente el propietario realiza tanto compras como ventas del material, por lo que tiene que realizar un cálculo de tiempo y presupuesto para planear los días siguientes,

y así prever, ¿Cuánto puede gastar y en qué? Así que tiene que saber cuales fueron los costos e ingresos diarios del material.

Cuando el dueño realiza la compra de material de manera directa, es decir, cuando manda alguno de sus vehículos por el material, quiere aprovechar el viaje y traer otros productos que le hagan falta pero, no siempre se da cuenta que material le falta porque no lleva un control de inventario correcto.

Hay materiales que son pequeños, por lo que es difícil ponerles atención, a cerca de su existencia; cuando los clientes llegan y preguntan por ellos, el encargado le hace esperar, para buscarlo y se da cuenta que no hay en existencia, por lo que hace perder tiempo al cliente.

Como los materiales cambian continuamente de precio, tiene un problema porque existe más de un encargado, entonces suele no avisarle a tiempo a alguno de ellos, por lo que realizan la venta con precios anteriores obteniendo una perdida en sus ingresos.

Otra dificultad a la que se enfrenta, es cuando vende material y lo tiene que entregar, como en el almacén cuenta con poco espacio, tiene poco material; cuando llega otro cliente que quiere el mismo material y es atendido por otro encargado, como no se trabaja de manera coordinada, se vende el material que ya no esta disponible, lo que ocasiona que algunas veces no se entregue el material a tiempo, porque tiene que traer el material que se acabo, para entregar el pedido al primer cliente que lo hizo.

4.1.1 Planteamiento del Problema

De los puntos anteriores se puede deducir el planteamiento del problema. Por lo señalado anteriormente, se entiende que es muy importante, tener un buen control de inventario, de hecho es indispensable, ya que en él descansa la base para administrar una micro, pequeña, mediana y grande empresa. Es por ello, que se realizará un sistema computarizado (software) que lleve a cabo el control del inventario y además realice el cálculo de los costos e ingresos diarios de los materiales.

Con el enunciado anterior ya se puede comenzar a construir la aplicación para el problema.

4.1.2 Comprensión del Sistema

Ahora que se conoce el planteamiento del problema, se llevará a cabo el siguiente paso, que es el de comprender el sistema lo más que se pueda para modelarlo lo más cercano posible a la realidad. Primero se describe el sistema en un lenguaje natural para después utilizar los modelados de objetos, dinámico y funcional.

4.1.3 Conociendo al Sistema

A continuación se analizan los puntos esenciales para manejar el inventario del negocio de la casa de materiales para construcción.

- El control del inventario se lleva a cabo manualmente, mediante dos procesos que son la compra y venta del material.

- La compra de material se hace de dos formas: la primera consiste en realizar el pedido con el proveedor para que envié el material; la otra forma es, cuando el mismo propietario envía a alguno de sus carros con el proveedor para traer el material, en ambos procesos se manejan las notas de compra y el pago se hace en efectivo. Aquí también se hace la actualización del inventario a través de las notas de compra.

  Una nota de compra contiene los siguientes conceptos:

  * Nombre de la Casa de Materiales del Proveedor.
  * Dirección.
  * Teléfono.
  * No. De Nota de Venta.
  * Fecha.
  * Cantidad.
  * Descripción.
  * Precio Unitario.
  * Importe.
  * Subtotal.
  * IVA.
  * Total.

  A continuación se presenta un ejemplo de una nota de compra de un proveedor.

PROVEEDORA DE MATERIAL AJUSCO S.A. DE C. V.
Carretera Picacho Ajusco, Col. Sto. Tomás Ajusco Km. 37.5
Tlalpan, D.F.

C.P. 14710 Teléfonos: 58-46-45-46 y 58-46-48-56

Enviar a: José Guadalupe Arellano Soto

Fecha: 06/Mayo/2003 | No. Nota 2124

CANT	DESCRIPCIÓN	PRECIO U.	IMPORTE
140	Bultos de Cto. Cruz Azul	$ 75.00	$ 10,500.00
150	Varillas de 3/8"	36.00	5,400.00

Pagado y Entregado

Cantidad el letra	SUBTOTAL	15,900.00
	IVA	
	TOTAL	15,900.00

Figura 4.1.1 Ejemplo de una nota de compra.

➢ La venta del material, también se maneja de dos formas: una de ellas es, que el cliente llegue a la casa de materiales "ARESCO", compre el material y se lo lleve el mismo. La segunda forma es, que el cliente haga un pedido de material, para que posteriormente se le envié en un flete, aunque el flete puede hacerse el mismo día o en los días siguientes. A diferencia de las notas de compra donde se paga en efectivo una vez entregado el material, en las notas de venta, el cliente puede pagarlo todo cuando hace el

pedido, dejar algo A/Cuenta o bien, pagar todo en el momento en que se entregue el material.

Las notas de venta contienen los siguientes datos:

* Nombre de la Casa de Materiales.
* Nombre del Cliente
* CURP del Propietario
* CURP del Propietario
* Código Postal
* Total
* Fecha
* Dirección
* Cantidad
* Importe
* Concepto

MATERIALES PARA CONSTRUCCIÓN "ARESCO" S.A. DE C. V.

R.F.C. AREJ-600810-M54 CURP AREJ-600810HYGDFG10
Huistepec #16 Col. Mesa los Hornos entre Cehuantepec y Chantepec
Tlalpan, D.F. C.P. 14420 México D.F.

Nombre: Juan Manuel Montes
Dirección: Chantepec Mz I Lt 20 Col. Mesa los Horno
Fecha: 14/Mayo/2003

No. Nota

CANT	DESCRIPCIÓN	PRECIO U.	IMPORTE
10	Bultos de cemento Tolteca	$ 80.00	$ 800.00
15	Varillas de 3/8"	37.50	562.50
17	Kgs. De Anillos 10 x 25	7.50	127.50
5	Kgs. De Alambre	7.50	37.50

Especificaciones:
A/cuenta dejo $800.00
Resta $727.50
Entregar : 16/Mayo/2003

SUBTOTAL	1,527.50
IVA	
TOTAL	1,527.50

Figura 4.1.2 Ejemplo de una nota de venta.

En la figura anterior se mostró la forma de llenar una nota de venta. Cuando se entrega el pedido se pone el sello de entregado y se cobra lo estipulado en la nota.

Con lo anterior ya se tiene un conocimiento mayor del sistema. En este momento es importante conocer el material con el que cuenta la casa de materiales, cómo se vende (kilogramos, pieza, tonelada, metros cúbicos, etc.) A continuación se hacen varias listas del material con que cuenta el negocio agrupándolas según sus características.

Lista de Materiales de Cemento			
Nombre	Marca	Precio	Unidad
Cemento Gris	Tolteca	79.00	Bulto (50 kgs)
Cemento Gris	Cruz azul	81.00	Bulto (50 kgs)
Cemento Blanco	Tolteca	78.00	Bulto (25 kgs)
Cemento Blanco	Cruz azul	80.00	Bulto (25 kgs)
Cemento Blanco	Tolteca	156.00	Bulto (50 kgs)
Cemento Blanco	Cruz azul	160.00	Bulto (50 kgs)
Mortero	Tolteca	55.00	Bulto (50 kgs)
Cal	Enterocal	24.00	Bulto (25 kgs)
Cal	Calidra	24.00	Bulto (25 kgs)
Yeso	Las Peñitas	25.00	Bulto (35 kgs)
Yeso	Sn. Luis	25.00	Bulto (35 kgs)
Yeso	Sn. Martin	25.00	Bulto (35 kgs)
Pegazulejo	Crest	75.00	Bulto (20 Kgs)
Pegazulejo	Perdura	54.00	Bulto (20 Kgs)
Pegazulejo	Interceramic	57.00	Bulto (20 Kgs)
Pastablanca	Tolteca	57.00	Bulto (20 Kgs)
Pegazulejo	Niasa	48.00	Bulto (20 Kgs)
CeroFino	-	25.00	-
Cero Grueso	-	25.00	-

Tabla 1 Materiales que se venden en bultos.

Este material, llamado cementos, se organizó de esa manera ya que la compra y venta de éste material se realiza por unidades llamadas bultos. El precio, varía aún cuando sea un mismo producto ya que depende de la marca a la cual pertenezca.

En la tabla anterior no se tomo en cuenta la cantidad en unidades con que cuenta el negocio, así como tampoco se mostró la inversión que hay de ese material, posteriormente se hará porque es necesario, ya que dentro del objetivo del problema, está el de obtener las ganancias diarias por la venta de estos productos y el número de unidades con que se cuenta.

A continuación se muestra otra tabla que contiene otro tipo de productos con otras características.

Lista de Materiales de Aceros				
Nombre	Medida	Marca	Precio	Unidad
Varilla	3/8"	Sn. Luis	37.50	Pieza
Varilla	3/8"	Hylsa	37.50	Pieza
Varilla	1/2"	Sn. Luis	68.00	Pieza
Varilla	1/2"	Hylsa	68.00	Pieza
Alambre	-	-	7.50	Kg.
Alambrón	-	-	5.50	Kg.
Anillos	10x15	-	7.50	Kg.
Anillos	10x20	-	7.50	Kg.
Anillos	10x25	-	7.50	Kg.
Anillos	10x30	-	7.50	Kg.
Anillos	15x15	-	7.50	Kg.
Anillos	15x20	-	7.50	Kg.
Anillos	15x25	-	7.50	Kg.
Anillos	20x20	-	7.50	Kg.
Anillos	20x25	-	7.50	Kg.
Clavos c/cabeza	2 1/2"	DeAcero	10.00	Kg.
Clavos c/cabeza	4"	A y R	10.00	Kg.
Clavos c/cabeza	2 1/2"	Sicartsa	10.00	Kg.
Clavos p/concreto	1 1/2"	-	0.35	Pieza
Clavos p/concreto	2"	-	0.50	Pieza
Clavos p/concreto	2 1/2"	-	0.50	Pieza
Registro Reforzado	60x40	-	56.00	Pieza

Tabla 2 Materiales de Acero.

En la tabla anterior se mostraron materiales que no cuentan con una marca, que son los anillos y los clavos para concreto. Solo se pusieron junto con la varilla y los clavos con cabeza porque son de acero. Ahora se muestran los productos utilizados para la instalación eléctrica en una construcción.

Materiales para Luz			
Nombre	Medida	Precio	Unidad
Caja de p/Luz	Chica	2.50	Pieza
Caja de p/Luz	Grande	4.00	Pieza
Chalupa	-	2.50	Pieza
Manguera p/Luz	1/2"	115.00	Rollo (100 Mts)
Manguera p/Luz	3/4"	150.00	Rollo (100 Mts)
Codo p/manguera	1/2"	0.50	Pieza
Codo p/manguera	3/4"	0.50	Pieza
Caja exagonal	-	12.00	Pieza

Tabla 3 Materiales para instalación eléctrica.

Ahora veamos la siguiente tabla de materiales de minas, llamados así porque así se me ocurrió.

Materiales de Minas		
Nombre	Medida	Precio
Arena	1 mt. Cub.	165.00
Grava	1 mt. Cub.	165.00
Grava Ligera	1 mt. Cub.	175.00
Tabicón pesado	Pieza	1.50
Tabicón Ligero	Pieza	1.50
Tabique Rojo Normal	Pieza	1.40

Tabla 4 Material de minas.

Es importante mencionar que los materiales mostrados anteriormente, no los tiene la casa de materiales, es decir, no hay en existencia; éstos se entregan sobre pedido. Se toman en cuenta porque tienen un ingreso para la casa de materiales.

Por último, tenemos la tabla 5 que contiene otros materiales que también se venden en la casa de materiales.

Materiales PVC			
Nombre	Medida	Precio	Unidad
Tubo PVC	2"	56.00	Pieza (6 Mts)
Tubo PVC	3"	70.00	Pieza (6 Mts)
Tubo PVC	4"	80.00	Pieza (6 Mts)
Codo PVC	2"x90	7.00	Pieza
Codo PVC	3"x90	10.00	Pieza
Codo PVC	4"x90	12.00	Pieza
Codo PVC	2"x45	7.00	Pieza
Codo PVC	3"x45	10.00	Pieza
Codo PVC	4"x45	12.00	Pieza
TEE PVC	2"	8.00	Pieza
TEE PVC	3"	10.00	Pieza
TEE PVC	4"	12.00	Pieza
YEE PVC	2"	8.00	Pieza
YEE PVC	3"	10.00	Pieza
YEE PVC	4"	12.00	Pieza
Coladera PVC	-	12.00	Pieza
Cople PVC	2"	4.00	Pieza
Cople PVC	3"	6.50	Pieza
Cople PVC	4"	8.00	Pieza

Reducción PVC	3"x2"	7.00	Pieza
Reducción PVC	4"x2"	8.00	Pieza
Reducción PVC	4"x3"	8.00	Pieza
Pegamento PVC	40ml	10.00	Pieza
Pegamento PVC	75ml	16.00	Pieza

Tabla 5 Materiales de PVC.

Estos son todos los productos que vende la casa de materiales; algunas veces varía porque aparecen nuevos productos en el mercado, los cuales quieren dar a conocer y se hacen las pruebas necesarias para saber si funciona o no.

4.1.4 Identificación de Clases

El siguiente paso es identificar las clases para crear el modelo de objetos del sistema de inventario. De acuerdo a la información anterior se pueden obtener las siguientes clases candidatas:

- Almacén de materiales
- Notas de Compra
- Notas de Venta
- Caja
- Proveedor
- Clientes

Estas son clases inicialmente candidatas, que se deben analizar para construir la estructura del sistema. La descripción de cada una de ellas se hará a continuación, aunque la definición de clases y atributos se deben hacer en el diccionario de datos.

Materiales. Son los productos con que cuenta el negocio y los materiales que puede vender aun cuando no se tengan en existencia.

Almacén. El almacén en realidad es la casa de materiales para construcción, en ella se encuentran casi todos los productos que se venden.

Notas de Compra. Con ellas se lleva el control de todo el material que se compra diariamente.

Notas de Venta. Con ellas se lleva el control del material que se vende a diario.

Caja. Es la encargada de pagar y recibir los costos e ingresos de la casa de materiales.

Proveedor. Son otras empresas que abastecen de productos a la casa de materiales "ARESCO".

Cliente. Es el consumidor, es decir, quien compra los materiales.

A continuación se hace los diagramas que muestran las relaciones entre las clases. Vea las siguientes figuras.

Un almacén recibe varias notas de compra.
Una nota de compra es recibida por un almacén.

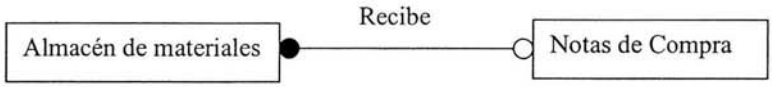


Figura 4.1.3 Relación entre almacén de materiales y notas de compra.

Un almacén verifica varias notas de venta.
Una nota de venta es verificada por un solo almacén.

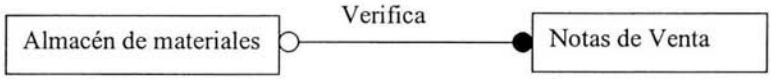


Figura 4.1.4 Relación entre almacén y notas de venta

Un cliente tiene varias notas de venta .
Una nota de venta tiene un solo cliente.

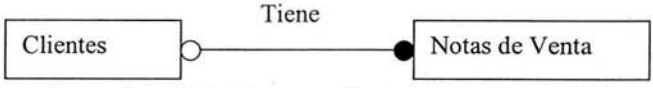


Figura 4.1.5 Relación entre clientes y notas de venta.

Una caja cobra muchas notas de venta.
Una nota de venta es cobrada por una sola caja.

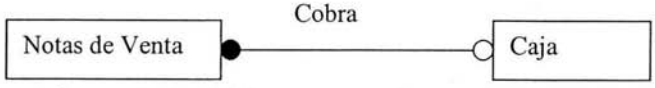


Figura 4.1.6 Relación entre notas de venta y caja.

Un proveedor proporciona muchas notas de compra.
Una nota de compra es proporcionada por un solo proveedor.

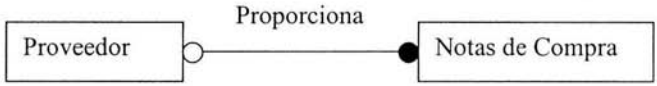


Figura 4.1.7 Relación entre proveedor y notas de compra.

Una caja paga varias notas de compra.
Una nota de compra es pagada por una sola caja.

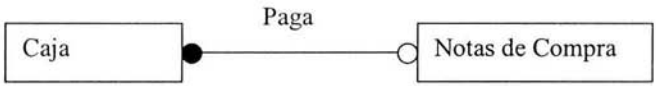


Figura 4.1.8 Relación entre caja y notas de compra.

Un diagrama global de todas las clases relacionadas quedaría de la siguiente forma Figura 4.1.9

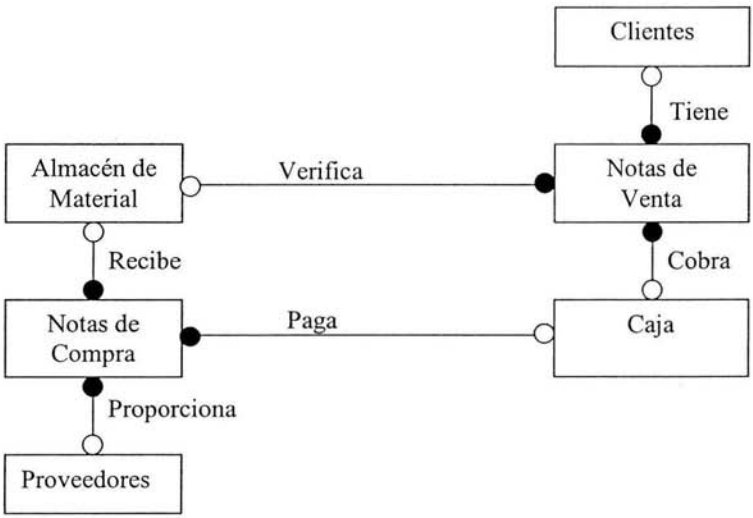


Figura 4.1.9 Diagrama de relaciones entre clases.

El siguiente paso es identificar los atributos de cada una delas clases, a través del planteamiento del problema y del conocimiento del sistema se puede llevar acabo.

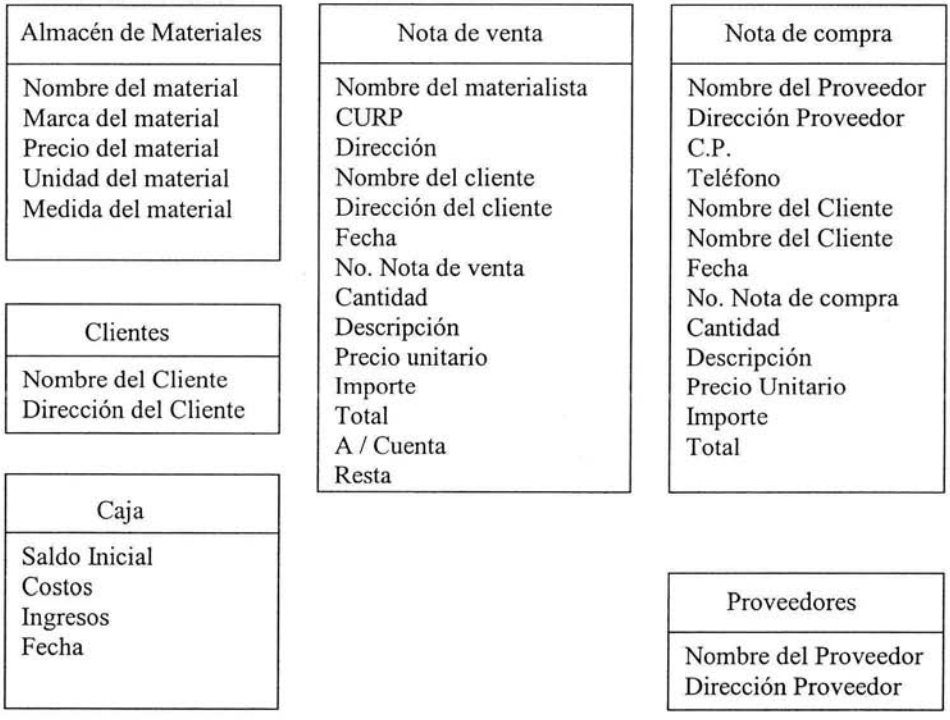


Figura 4.1.10 Atributos de cada una de las clases

Ya se tienen los atributos de las clases, ahora hay que refinar esos atributos para encontrar las características propias de cada objeto. Para hacerlo utilizamos el concepto de herencia. De las clases anteriores, se utilizó la herencia solamente para el caso de los materiales, ya que en ella se puede explotar esta herramienta.

Los materiales, constan de diferentes atributos como se vio anteriormente, sin embargo, para el caso de la clase materiales, hay atributos que no se contemplaron, ya que dependen de dos clases diferentes, a estos nuevos atributos se les llama atributos de enlace*.

A continuación se presenta el modelo que se hizo para los materiales utilizando la herencia, y a partir de ese modelo se muestra cómo obtuvieron los atributos de enlace.

* Véase el capitulo 1.2.7 Atributos de Enlace.

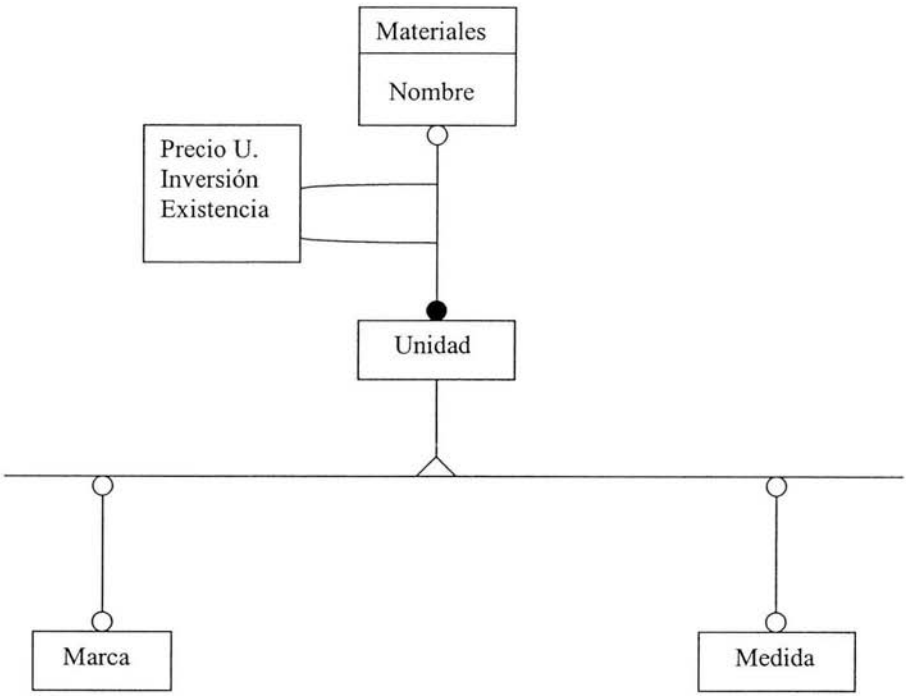


Figura 4.1.11 Muestra la herencia de la clase materiales y atributos de enlace.

Para comprender mejor el modelo anterior, veamos porque no todos los materiales tienen los mismos atributos y porque es conveniente manejar la herencia en especial para esta clase.

Como se presento en la figura anterior vimos que la primer subclase es la unidad, eso quiere decir que hay materiales que solo dependen del nombre del material y de la unidad, y con sólo esos atributos determinan el precio del producto. A continuación se presenta la tabla para esos materiales.

Unidad	Nombre	Precio	Inversión	Existencia
Kilogramo	Alambre	7.50	xxx	xxx
Metro Cúbico	Arena	165.00	xxx	xxx
Saco	Cero fino	25.00	xxx	xxx
Saco	Cero grueso	25.00	xxx	xxx
Pieza	Caja p/luz Hex.	12.00	xxx	xxx
Pieza	Chalupa	2.50	xxx	xxx

Pieza	Caja p/luz Chica	2.50	xxx	xxx
Metro Cúbico	Granzón	165.00	xxx	xxx
Metro Cúbico	Grava normal	165.00	xxx	xxx
Metro Cúbico	Grava ligera	175.00	xxx	xxx
Pieza	Registro Ref.	56.00	xxx	xxx
Metro Cúbico	Tezontle	170.00	xxx	xxx
Pieza	Tabicón Pesado	1.50	xxx	xxx
Pieza	Tabicón Ligero	1.50	xxx	xxx
Pieza	Tabique rojo	1.40	xxx	xxx
Pieza	Ladrillo	1.30	xxx	xxx

Tabla 6 Precio de materiales que solo cuentan con nombre y unidad.

Ahora, siguen los materiales que dependen de la unidad, nombre y medida para determinar su precio.

Unidad	Nombre	Medida	Precio	Inversión	Existencia
Kilogramo	Anillos	10 x 15 cms.	7.50	xxx	xxx
Kilogramo	Anillos	10 x 20 cms.	7.50	xxx	xxx
Kilogramo	Anillos	10 x 25 cms.	7.50	xxx	xxx
Kilogramo	Anillos	15 x 15 cms.	7.50	xxx	xxx
Kilogramo	Anillos	15 x 20 cms.	7.50	xxx	xxx
Kilogramo	Anillos	15 x 25 cms.	7.50	xxx	xxx
Kilogramo	Anillos	20 x 20 cms.	7.50	xxx	xxx
Kilogramo	Anillos	20 x 25 cms.	7.50	xxx	xxx
Kilogramos	Clavos c/cabeza	2 1/2"	10.00	xxx	xxx
Kilogramos	Clavos c/cabeza	3"	10.00	xxx	xxx
Kilogramos	Clavos c/cabeza	4"	10.00	xxx	xxx
Pieza	Clavos p/concreto	1 1/2"	0.33	xxx	xxx
Pieza	Clavos p/concreto	2"	0.50	xxx	xxx
Pieza	Clavos p/concreto	2 1/2"	0.50	xxx	xxx
Pieza	Reducción PVC	4" a 2"	5.00	xxx	xxx
Pieza	Reducción PVC	4" a 3"	7.00	xxx	xxx
Pieza	Tee PVC	2"	6.00	xxx	xxx
Pieza	Tee PVC	3"	8.00	xxx	xxx
Pieza	Tee PVC	4"	10.00	xxx	xxx
Pieza	Yee PVC	2"	6.00	xxx	xxx
Pieza	Yee PVC	3"	8.00	xxx	xxx
Pieza	Yee PVC	4"	10.00	xxx	xxx
Mt. Lineal	Tubo PVC	2"	8.00	xxx	xxx
Mt. Lineal	Tubo PVC	3"	10.00	xxx	xxx
Mt. Lineal	Tubo PVC	4"	12.00	xxx	xxx

Tabla 7 Precio de materiales que cuentan con la unidad, nombre y medida.

Siguiendo con el modelo, se muestran los siguientes materiales que dependen de la unidad, el nombre del material y la marca, vea la tabla 8.

Unidad	Nombre	Marca	Precio	Inversión	Existencia
Bulto (25 Kgs.)	Cal	Calidra	24.00	xxx	xxx
Bulto (25 Kgs.)	Cal	Enterocal	24.00	xxx	xxx
Bulto (50 Kgs.)	Cemento Gris	Cruz Azul	80.00	xxx	xxx
Bulto (50 Kgs.)	Cemento Gris	Tolteca	78.00	xxx	xxx
Bulto (25 Kgs.)	Cemento Blanco	Tolteca	79.00	xxx	xxx
Bulto (25 Kgs.)	Cemento Blanco	Cruz Azul	79.00	xxx	xxx
Bulto (50 Kgs.)	Cemento Blanco	Tolteca	156.00	xxx	xxx
Bulto (50 Kgs.)	Cemento Blanco	Cruz Azul	156.00	xxx	xxx
Bulto (20 Kgs.)	Pegazulejo	Crest	78.00	xxx	xxx
Bulto (20 Kgs.)	Pegazulejo	Niasa	45.00	xxx	xxx
Bulto (20 Kgs.)	Pegazulejo	Pegarex	45.00	xxx	xxx
Bulto (20 Kgs.)	Pegazulejo	Perdura	55.00	xxx	xxx
Bulto (20 Kgs.)	Pegazulejo	Tolteca	55.00	xxx	xxx
Saco	Yeso	San Martin	25.00	xxx	xxx
Saco	Yeso	San Luis	25.00	xxx	xxx
Saco	Yeso	Las Peñitas	25.00	xxx	xxx

Tabla 8 Precio de materiales que cuentan con unidad, nombre y marca.

Por último tenemos a los materiales que utilizan a los 4 atributos de la clase materiales: unidad, nombre, marca y medida, vea la tabla 9.

Unidad	Nombre	Marca	Medida	Precio	Inversión	Existencia
Pieza	Varilla	Sicarsa	3/8"	37.50	xxx	xxx
Pieza	Varilla	Hylsa	1/2"	37.50	xxx	xxx
Pieza	Varilla	San Luis	5/8"	37.50	xxx	xxx
Pieza	Varilla	Sicarsa	3/8"	68.00	xxx	xxx
Pieza	Varilla	Hylsa	1/2"	68.00	xxx	xxx
Pieza	Varilla	San Luis	5/8"	68.00	xxx	xxx
Pieza	Varilla	Sicarsa	3/8"	68.00	xxx	xxx
Pieza	Varilla	Hylsa	1/2"	105.00	xxx	xxx
Pieza	Varilla	San Luis	5/8"	105.00	xxx	xxx
Pieza	Varilla	Sicarsa	3/8"	105.00	xxx	xxx

Tabla 9 Precio de materiales que tienen unidad, nombre, marca y medida.

Se puede apreciar que las tablas, cambiaron a partir del modelo creado con la herencia. Esto es parte de lo que ofrece el modelo orientado a objetos, para facilitar el trabajo de los objetos. Ahora las tablas se clasificaron a partir de las características que comparten generando un otro tipo de tablas, que son con las que se trabajaran en la aplicación.

Ahora que ya se han identificado los atributos de las clases y la herencia que se puede derivar de cada una de ellas, es el momento de mostrar un nuevo modelo de las relaciones entre clases con sus atributos. Vea la siguiente figura 4.16.

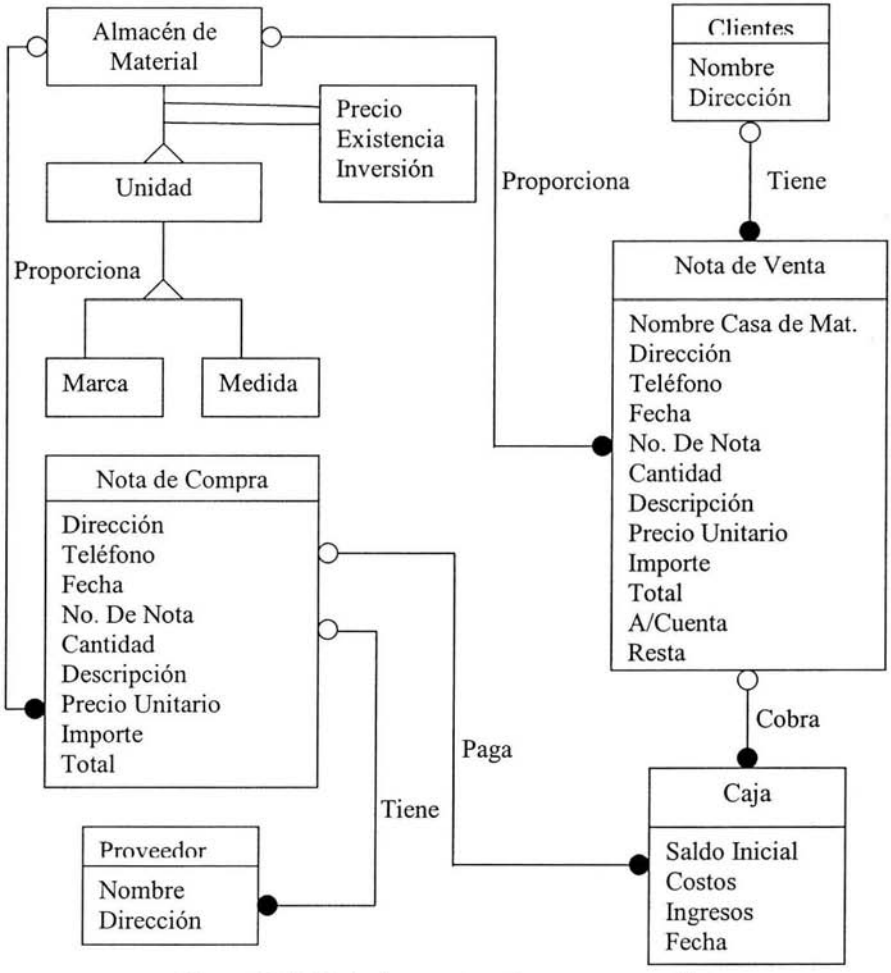


Figura 4.1.12 Relaciones entre clases con sus atributos.

En la figura anterior se muestra el modelo de datos, con las clases, atributos, y sus relaciones. Es importante utilizar sólo, los atributos que nos interesen de una clase y desechar aquellos que no sean de utilidad.

El siguiente paso es analizar el comportamiento dinámico del sistema, es decir, ver cuales son los estímulos que provocan una respuesta por parte del sistema. Para llevar a cabo lo antes mencionado utilizamos los escenarios.*

Se comienza por preparar los diálogos entre el usuario y el sistema para simular el comportamiento de la realidad. Comenzare por mostrar el escenario para realizar la venta del material.

- Llega un cliente a comprar material.
- El encargado checa que haya el material que se ha pedido. Si existe, realiza la nota y actualiza el inventario, si no informa al cliente de la ausencia del material.
- Una vez terminada la nota cobra el total.
- Entrega la nota al cliente quien la lleva al almacenista para que le entreguen el material.
- El almacenista entrega el material y pone el sello de entregado a la nota.

En este caso el cliente en el momento que realiza la compra se lleva su material.

Otro escenario seria cuando:

- El cliente llega y realiza un pedido a la casa de materiales.
- El encargado realiza la nota y una copia.
- El cliente solo paga una parte del total de la nota.
- El encargado anota el efectivo que se paga y lo que resta del total.
- El encargado envía la nota al almacenista para que mande el material en la camioneta.
- El almacenista entrega la nota al chofer para que la entregue y cobre el resto del total.
- El chofer entrega el material cobra lo señalado en la nota al cliente y lo entrega al encargado.

Un tercer escenario para la realización de una nota de venta sería el siguiente:

- El cliente realiza el pedido a través de una llamada telefónica.
- El encargado toma el pedido y realiza la nota.
- El encargado pone en la nota que es por cobrar. En este caso el cliente paga el total de la nota cuando se le entrega el material.
- El encargado entrega la nota al almacenista, para que mande a la camioneta con el material que se encuentra en la nota.
- El almacenista entrega la nota al chofer para que cobre el total del material.

* Ver Capitulo 2.2.3 Seguimiento de Sucesos.

- El chofer le entrega el material al cliente y lo cobra.
- El chofer entrega el dinero al encargado.

Estos son los escenarios que se llevan a cabo cuando se hace una venta de material. Aclaró que cuando un encargado hace la nota se le toman los datos del cliente como son nombre, dirección, fecha en que se hizo el pedido, cantidad descripción, precio unitario, importe, total y los conceptos A/Cuenta y resta. Además, el encargado recibe el dinero pero en realidad el dinero ingresa a la caja.

Para el caso de la compra de material, se tienen 2 escenarios. El primero es:

- El propietario de la casa de materiales manda a una de sus camionetas por el material que le hace falta.
- El proveedor realiza la nota de venta para el materialista.
- La compra se realiza en efectivo por lo que se paga en ese momento.
- Posteriormente el almacenista recibe el material que compro el propietario, verifica que la cantidad de material que se compro sea la correcta y actualiza el inventario.

El segundo escenario seria el siguiente:

- El propietario realiza el pedido por teléfono.
- El proveedor toma sus datos y envía el material especificado.
- Una vez que llega el material el almacenista verifica que la cantidad de material sea correcta.
- Se descarga el material y se actualiza el inventario.
- El encargado paga el efectivo del total de la nota.

En estos escenarios no se indica de donde se pagan las notas de compra, sin embargo, es obvio que la caja es la que paga el costo del material.

Una mejor forma de mostrar los escenarios es a través de las trazas de sucesos.* A continuación se muestran las traza de sucesos para la nota de venta. Cada traza muestra un escenario diferente de un mismo concepto, que es la nota de venta.

* Vea el capitulo 2.2.4 Seguimiento de Sucesos a Través de Trazas.

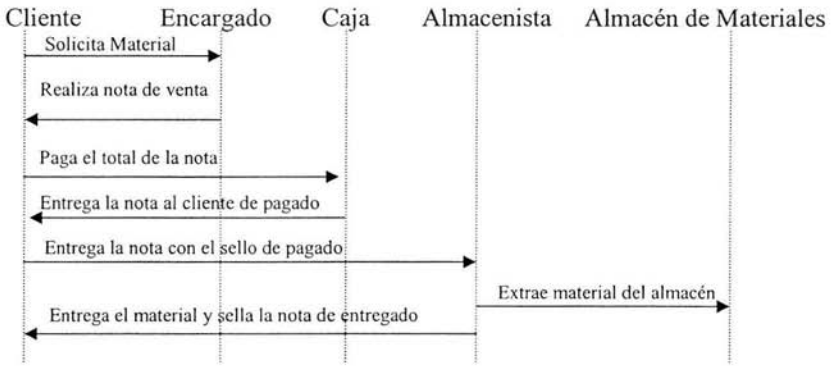


Figura 4.1.13 Traza de la nota de venta del primer escenario.

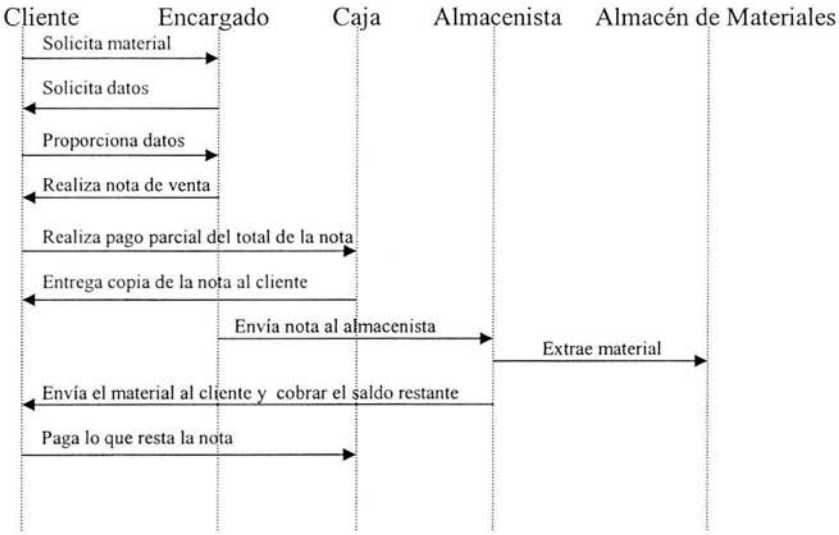


Figura 4.1.14 Traza de la nota de venta del segundo escenario.

A continuación el último escenario de la nota de venta.

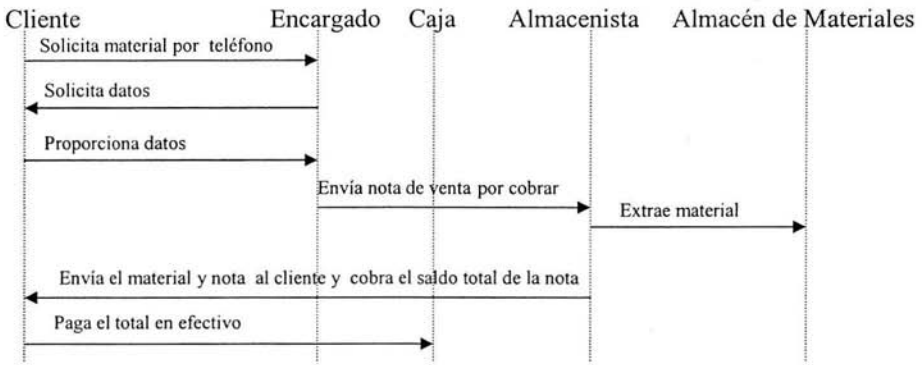


Figura 4.1.15 Traza de la nota de venta del tercer escenario.

Ahora se mostrará las trazas de sucesos de los escenarios de la nota de compra.

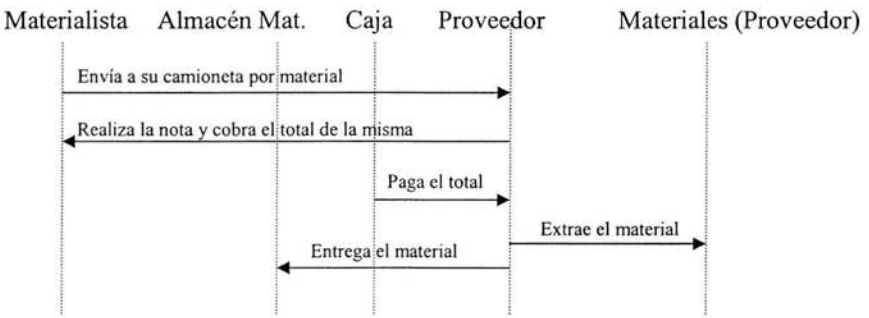


Figura 4.1.16 Traza de la nota de compra del primer escenario.

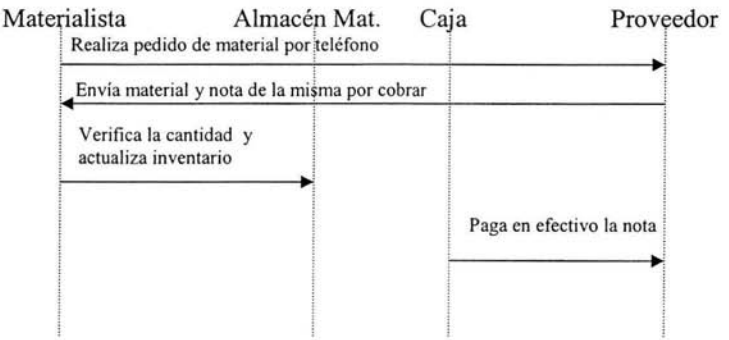


Figura 4.1.17 Traza de la nota de compra del segundo escenario.

Es el momento de detallar más la secuencia de sucesos y estados, para hacerlo se tienen que construir los diagramas de estados.* Se unificaran los escenarios tanto para la notas de venta como para la nota de compra.

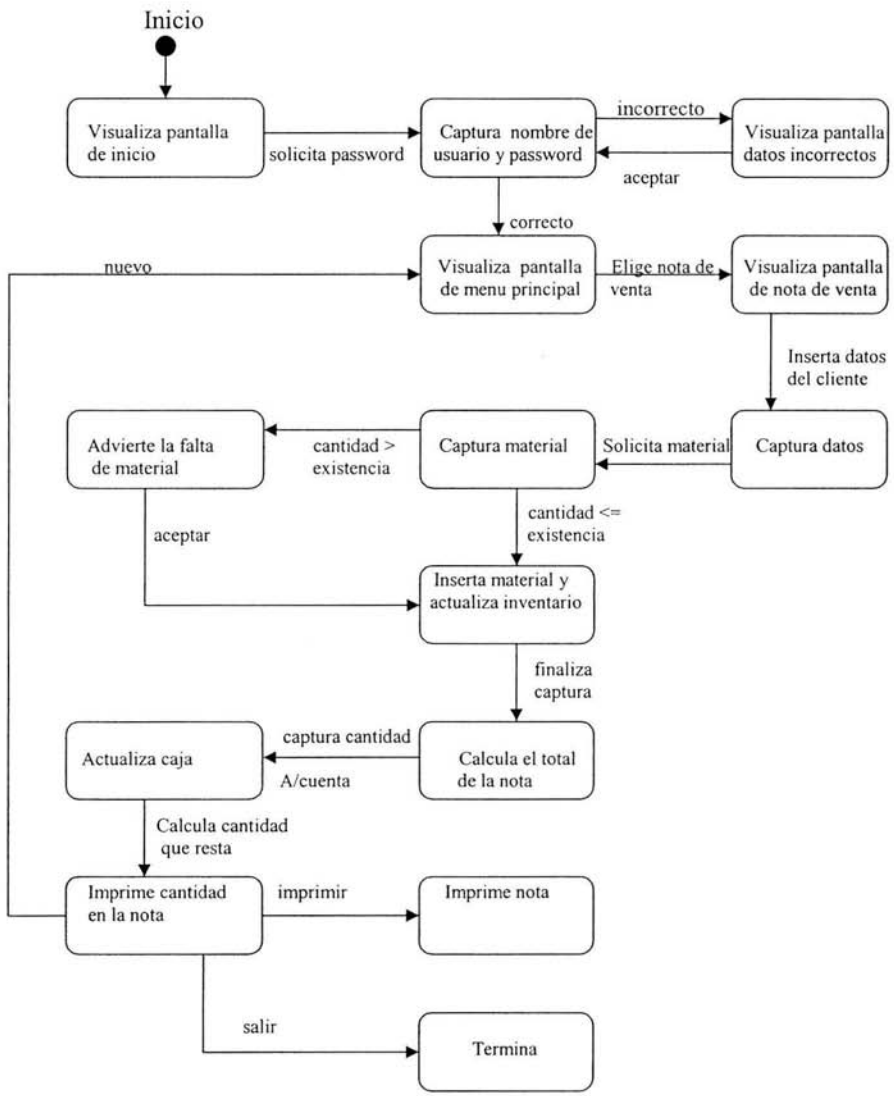


Figura 4.1.18 Diagrama de estado para la nota de venta.

* Ver el capitulo 2.2.6 Diagramas de Estados.

Este es el diagrama de estados para los escenarios de la nota de compra.

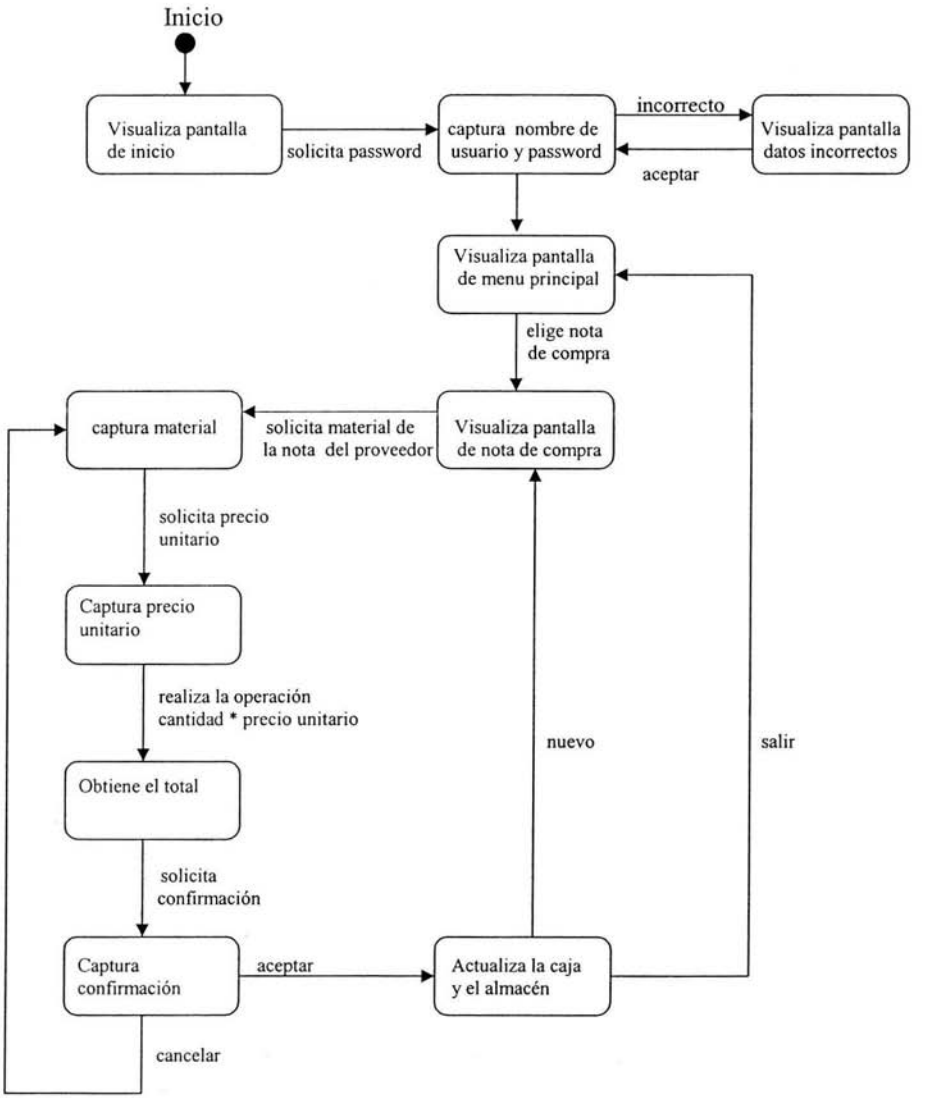


Figura 4.1.19 Diagrama de estados para la nota de compra.

Ya se describieron y mostraron los modelos de objetos (estático), también el modelo dinámico falta el modelo funcional* dentro del análisis. Para ello se deben conocer los valores de entrada y salida. Observe las siguientes figuras donde se muestran estos valores.

* Ver el capitulo 2.3 Modelo Funcional.

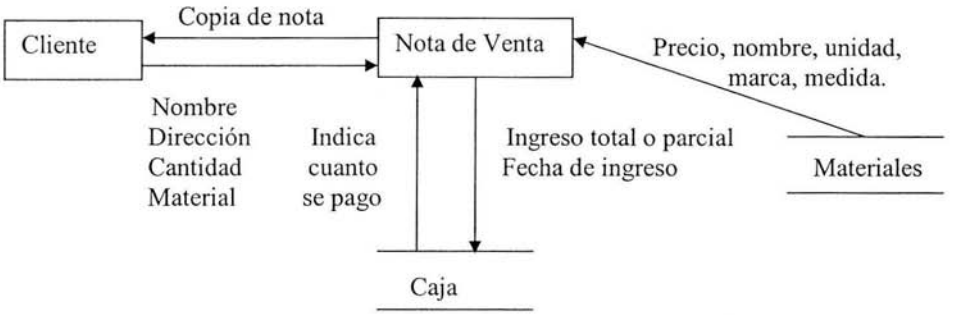


Figura 4.1.20 Entradas y salidas de las notas de venta.

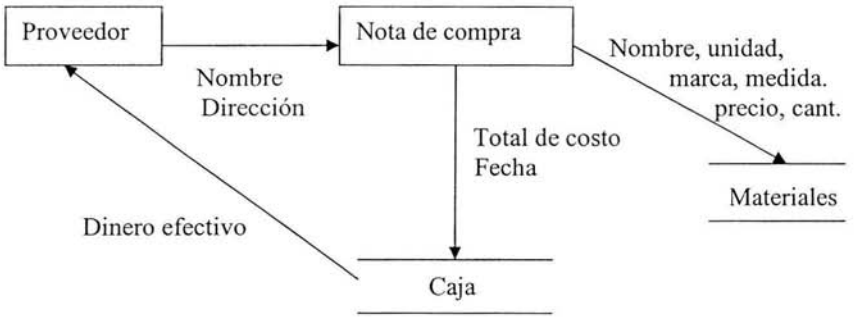


Figura 4.1.21 Entradas y salidas de las notas de compra.

En la figura 4.1.20 la nota de venta tiene como entradas del cliente, nombre y dirección, su finalidad es de entregar el material adecuado a la persona indicada y para cualquier aclaración que haga el cliente. La salida es una copia de la nota para lo antes señalado y como comprobante de la compra del material.

Las entradas del material hacia la nota de venta, tiene tres funciones especificas. La primera, es para indicar si el almacén tiene suficiente material en existencia, no puede vender algo que no tiene y el cliente necesita en ese momento. La segunda es para realizar la venta del material aunque no se tenga en existencia, ya que el cliente puede no necesitarlo en el momento y simplemente hace el pedido para que se le pueda llevar posteriormente. La tercera, es para actualizar el inventario del material que se vendió.

Las funciones de la nota de compra en relación a las entradas y salidas son las se encuentran en la figura 4.1.21

Toda nota de compra debe tener nombre del proveedor, dirección y teléfono. Estos datos son importantes para el materialista, ya que puede mandar su camioneta por material con alguno de los proveedores o para hacer el pedido por teléfono en algún momento dado.

Las salidas de la nota de compra y entradas a los materiales, indican el material que ingreso al almacén de materiales, por supuesto son importantes para actualizar el inventario del almacén así como para comprobar que dicho material que entro es el que el propietario pidió.

Las salidas hacia la caja, indican la fecha en que entrego el material y el importe total que tiene que pagar al proveedor en efectivo.

Por último, como no se ha aclarado como ingresa el dinero de las notas que son por cobrar o que se pago solo una parte y se quedo a deber otra, tuve que crear una nueva clase que pudiera realizar esa tarea, eso se verá mas adelante en el diseño del sistema.

Hasta aquí, he concluido con el análisis, cabe señalar que hice varias iteraciones para poder construir esta parte, aunque no se si sean las necesarias, por la importancia que tiene esta fase en el desarrollo de software.

4.2 Diseño del Sistema

La segunda parte del ciclo de vida del software es el diseño, parte fundamental para crear software de calidad. Al principio del capitulo indique lo extenso y complejo que puede llegar a ser este sistema, por los diferentes procesos que existen en una casa de materiales, aunque ésta sea pequeña. Es por ello que solo decidí realizar el sistema para el control de inventario.

Lo primero que se hace es descomponer el sistema en subsistemas, en este caso uno de los subsistemas es el inventario. Sin embargo este subsistema se puede dividir en otros más pequeños para reducir la complejidad del mismo. Dentro de estos subsistemas están las notas de compra, las notas de venta y el almacén de materiales partes fundamentales ya antes mencionadas en el análisis. Cada una de éstas partes es un pequeño subsistema independiente y a su vez cada una de ellas se relaciona con otras partes.

Analicemos en detalle las siguientes figuras de las pantallas propuestas para el diseño de interfaz gráfica de usuario del sistema.

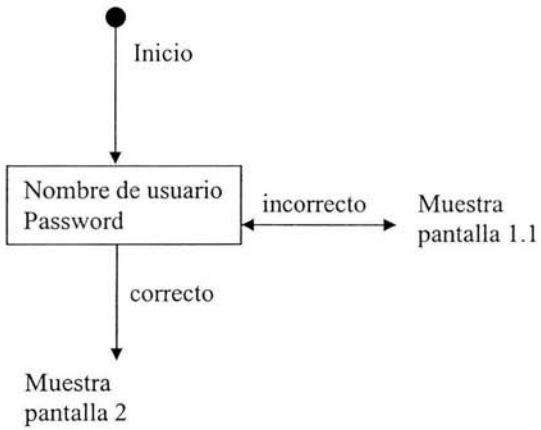


Figura 4.2.1

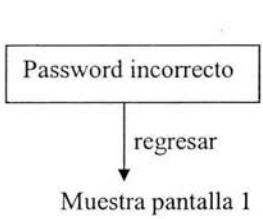


Figura 4.2.2

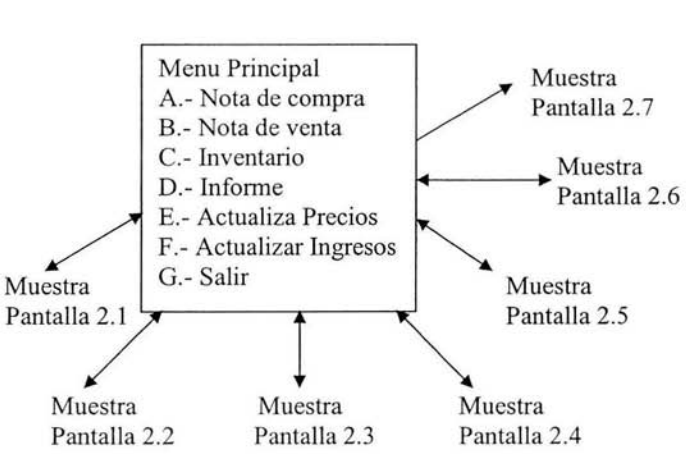


Figura 4.2.3

Pantalla 2.1

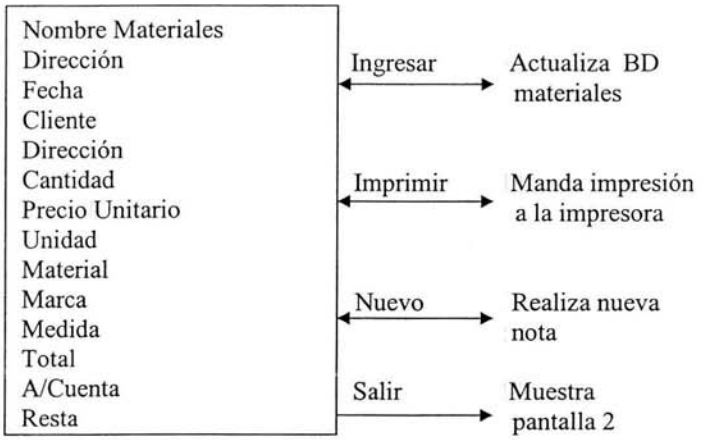


Figura 4.2.4

Este es el diseño de interfaz gráfica de usuario desde su construcción hasta la finalización de la misma para la creación de la nota de venta.

Veamos algunos los puntos que son importantes de acuerdo al diseño de sistemas.* Hago la aclaración de que no se llevan a cabo todos y cada uno de los pasos para realizar el diseño de sistemas ya que en este capitulo solo se muestra un ejemplo.

Siguiendo con las figuras 4.2.1, 4.2.2, 4.2.3 y 4.2.4 dentro del diseño de IGU (Interfaz Gráfica de Usuario), a continuación se muestra la concurrencia que existe entre ellas.

Pantalla 1 concurrencia con Pantalla 1.1
Pantalla 1.1 concurrencia con Pantalla 1
Pantalla 2 concurrencia con Pantalla 2.1, Pantalla 2.2, Pantalla 2.3 y Pantalla 2.4
Pantalla 2.1 concurrencia con Pantalla 2
Pantalla 2.2 concurrencia con Pantalla 2
Pantalla 2.3 concurrencia con Pantalla 2
Pantalla 2.4 concurrencia con Pantalla 2

Ahora se hará el diseño que sirve para construir la nota de compra, en base a las entradas y salidas generadas por este concepto.

* Ver el Capitulo 3.2 .1 Diseño de Sistemas Orientado a Objetos.

Pantalla 2.2

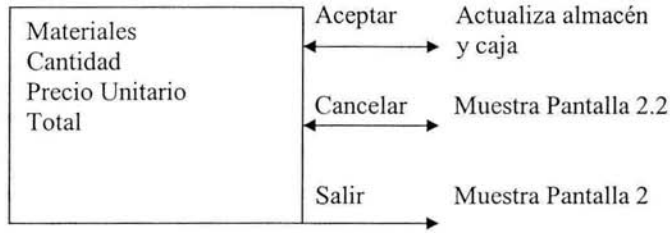


Figura 4.2.5

En la figura anterior, tal vez se pregunten ¿Dónde esta el nombre del proveedor? y ¿la dirección, teléfono? Etc. Es por el hecho de que el control se lleva manualmente, es decir se almacenan las notas que entrega el proveedor, sin embargo como estas notas afectan al almacén ya que actualizan el material, es necesario una interfaz que lleve ese control. Si el materialista quiere llevar un registro de las notas de cada proveedor se puede computarizar, pero ese no es el objetivo ni el problema que se quiere resolver.

Pantalla 2.2 concurrencia con pantalla 2 y 2.2

La figura 4.2.6 muestra el reporte de la cantidad de material que se tiene en existencia.

Pantalla 2.3

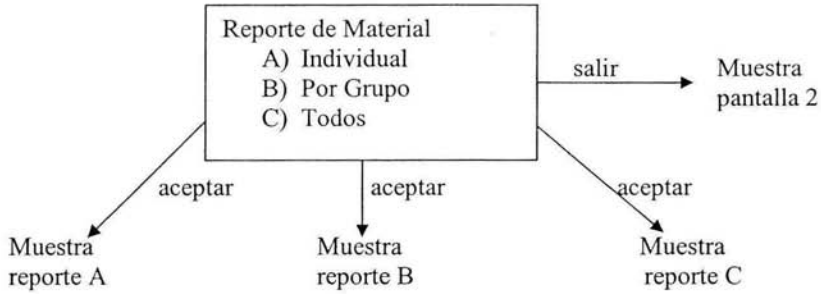


Figura 4.2.6

Este reporte es parte de la solución del problema ya que el usuario puede revisar la existencia de cualquier material en cualquier momento.

Pantalla 2.3 concurrencia pantalla 2.3 y 2.

La siguiente pantalla (figura 4.2.7) muestra el informe de costos e ingresos generados diariamente.

Pantalla 2.4

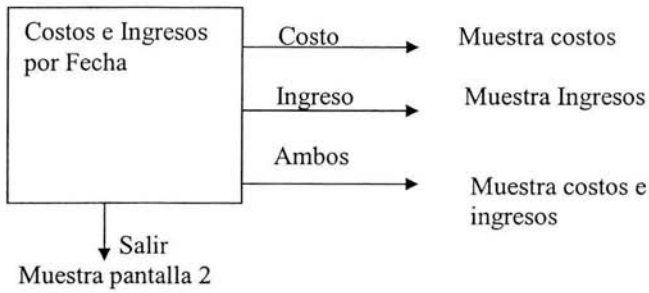


Figura 4.2.7

Pantalla 2.4 concurrencia 2.4 y 2

Continuando con las pantallas de IGU, la siguiente es la actualización de precio de los productos (Figura 4.2.8).

Pantalla 2.5

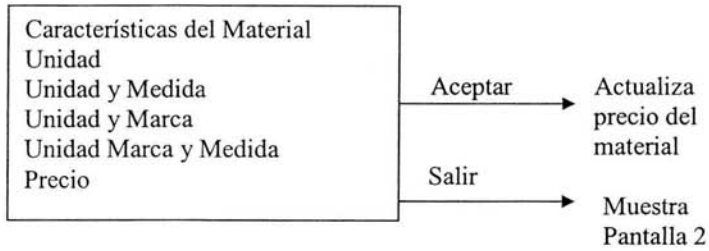


Figura 4.2.8

Pantalla 2.5 concurrencia 2.5 y 2

Por último tenemos la pantalla 2.6 Que al final del análisis se menciono. Esta pantalla se realizó, para hacer la actualización de ingresos de las notas que quedaron a deber o que se cobran una vez enviado el material. La cual quedaría como se muestra en la figura 4.2.9

Pantalla 2.6

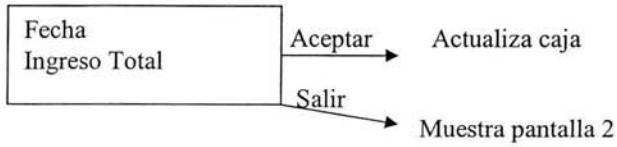


Figura 4.2.9

Pantalla 2.7

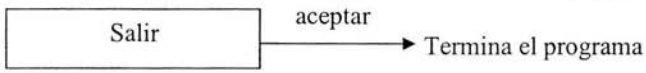


Figura 4.2.10

Este es el diseño propuesto para resolver el problema que se formulo en el análisis. Ahora se mostrarán los requisitos propuestos de Software y Hardware para poder realizar el software requerido. A partir de este momento ya es mucho más fácil poder programar y además con base al modelo desarrollado durante el análisis y el diseño se podrá desarrollar un software confiable y de calidad.

Software Propuesto:

- Sistemas Operativos Windows 9x.
- Lenguaje de Programación Java 2 versiones 1.2, 1.3, ó 1.4
- Microsoft Access 2000

Hardware Propuesto:

- Procesador Pentium
- Memoria RAM 32 MB
- Disco Duro 2HB
- Impresora
- Unidad de Disckets de 3.5"

Estas estimaciones propuestas tanto de software como de hardware son mínimas, es decir, cualquier equipo con características mayores o superiores a las mostradas serán mejores para el software.

A continuación se muestra la parte del programa del ejemplo de la nota de venta.

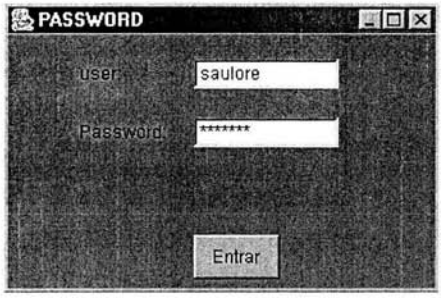


Figura 4.2.11 Pantalla inicial de la aplicación.

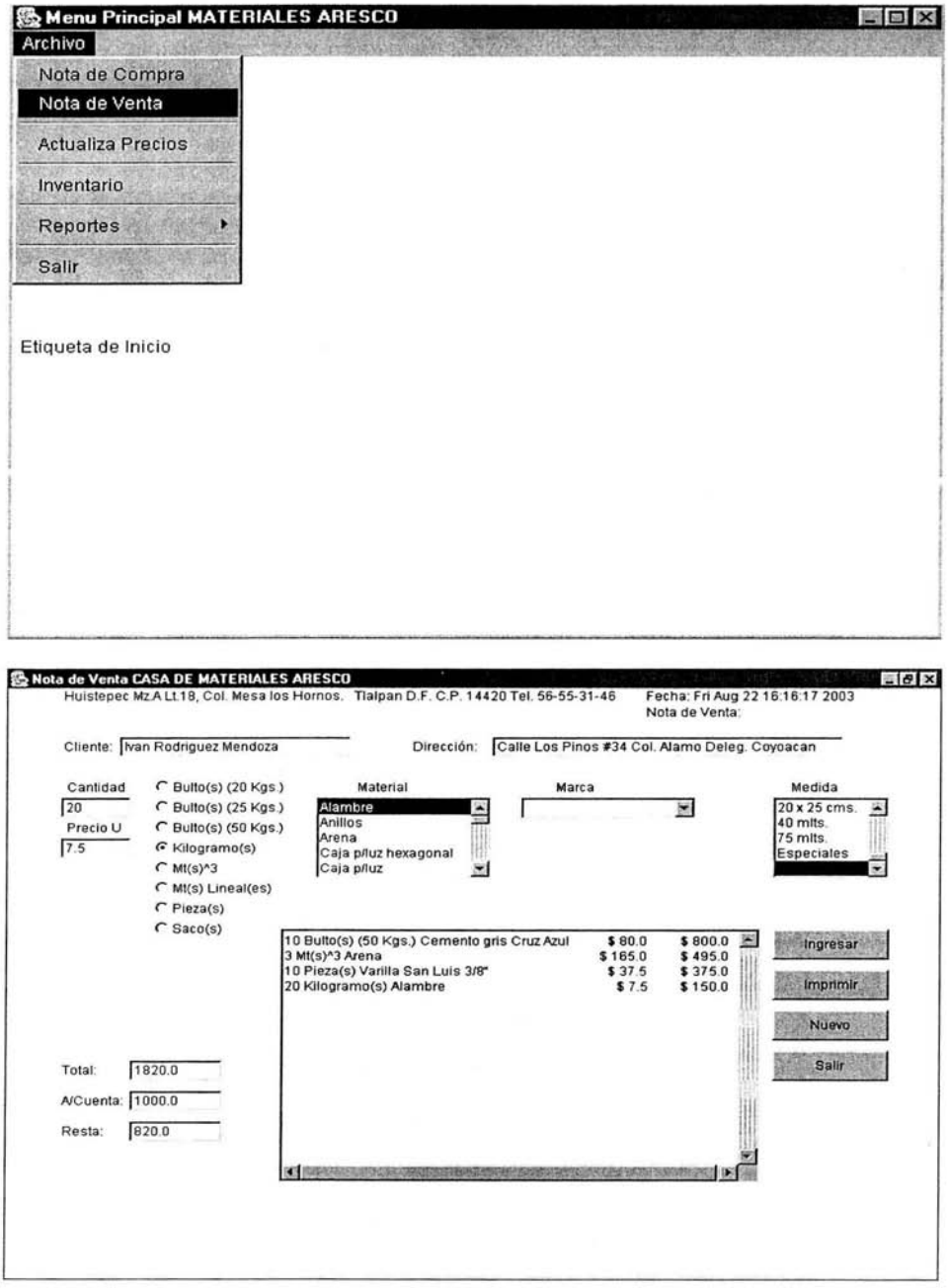


Figura 4.2.12 y 4.2.13. Pantalla del menu Principal (arriba) y nota de venta (abajo).

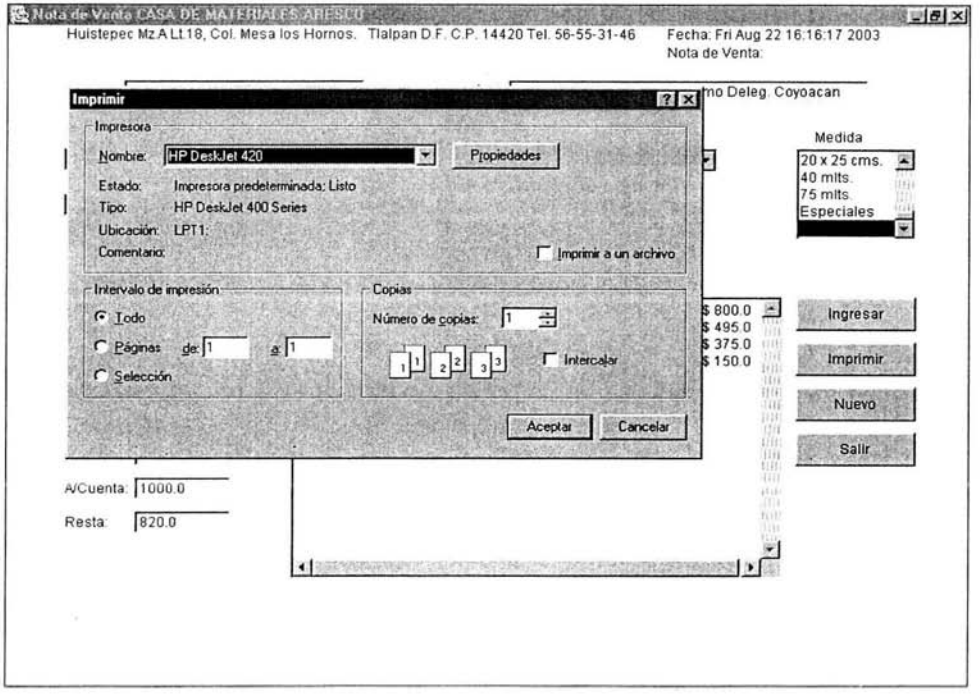


Figura 4.2.14 Impresión de la nota de venta.

Esta es la interfaz para la nota de venta. Al igual que esta interfaz se desarrollaron las otras propuestas en el diseño. De ahí la importancia del análisis y el diseño de software. Si se tiene un buen análisis y un buen diseño entonces tendrá una buena aplicación. Este software ya ha sido implementado y se están haciendo las pruebas correspondientes del mismo. Si usted quiere verificarlo, puede hacerlo ya que se anexa un disckt de 3 ½" con los archivos class para que usted los pueda compilar, además de que puede analizar todas las interfaces de usuario hechas para este sistema.

Con esta implementación termina el desarrollo de éste pequeño software. Con él se pudo dar solución al problema planteado en el análisis, es decir este sistema lleva un buen control de inventario, obtiene los reportes de los gastos e ingresos que se realizan diariamente y otros adicionales, como son el dar de alta los precios en cualquier momento, realizar las notas de compra y de venta para automatizar el inventario sin mayor problema. Con ello, se puede afirmar que la metodología TMO es muy buena para la creación de diversas aplicaciones donde se utiliza el enfoque orientado a objetos.

CONCLUSIONES

Se ha comprobado que la TMO es una metodología buena, fácil de entender y confiable para trabajar con ella.

Con esta metodología se realizó un software para un pequeño negocio (una casa de materiales), y actualmente se esta trabajando en él, teniendo un buen desempeño y llevando un buen control de inventario, como se esperaba.

Muchas veces cuando somos estudiantes, nos molestamos porque no nos enseñan algún lenguaje de programación actual, como pueden ser Visual Basic, Java, Visual C++, etc. Y no le damos la importancia que tienen a las metodologías. La realidad es que la programación es una parte pequeña en el desarrollo de software, es por eso, que las metodologías tienen mucho mayor importancia, ya que a través de ellas se puede construir un software bien pensado, bien planeado y bien elaborado. Y para hacerlo, es necesario conocer como se construye el software de calidad. La ingeniería de software nos ayuda a comprender lo importante es cada paso en el ciclo de vida de un sistema, nos dice que elementos son indispensables y necesarios en la construcción del mismo.

Ahora existen reglas que se tienen que respetar para crear aplicaciones con estructuras fuertes, para hacer menos costoso el mantenimiento y más duraderos los sistemas.

Cabe señalar que una sola metodología no es suficiente para resolver problemas complejos. Existen muchas metodologías orientadas a objetos, no podemos regirnos por una sola, lo más indicado sería escoger una metodología adecuada al problema que se nos presenta y tomar de las otras, mejores ideas y asociarlas a un problema específico, de tal forma que sean esas metodologías complementarias a la metodología base.

Se puede crear software de utilidad mediante la TMO, aunque, no es tan sencillo, cuesta trabajo entenderla, pero una vez que se llega a comprender se pueden crear aplicaciones más rápido y en poco tiempo. Una vez creado el modelo del sistema se puede programar en Java o en C++ que son los lenguajes mas comerciales, pero se puede asegurar que la programación se llevaría igual de sencilla en cualquier otro lenguaje como Ada, Smalltalk o incluso lenguajes procedimentales, como Pascal o C.

Todos los lenguajes tienen sus ventajas y desventajas, así que lo importante es hacer un buen análisis y un buen diseño del sistema, conocerlo a fondo, saber como se llevan a cabo los procesos, analizar en que forma se podrían mejorar. La TMO nos da la pauta para hacerlo, la importancia de una buena aplicación, no es la interfaz gráfica o los efectos bonitos, sino resolver el problema que el cliente desea, porque un negocio puede crecer o quebrar si no se llevan adecuadamente los procesos.

BIBLIOGRAFIA

[1] Modelado y Diseño Orientados a Objetos
James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy y William Lorensen
Prentice Hall
1991

[2] Software Orientado a Objetos
Ann. L. Winblad, Samuel D. Edwards y David R. King
Addison-Wesley
1990

[3] Ingeniería de Software un Enfoque Práctico
Roger S. Pressman
McGraw-Hill
1998

[4] Software Engineering
Ian Sommerville
Addison-Wesley
1998

[5] Análisis y Diseño Detallado de Aplicaciones Informáticas y de Gestión
Mario G. Piattini, José A. Calvo-Manzano, Joaquín Cervera y Luis Fernández
Ra-ma
1996

[6] Análisis Estructurado
Edward Yourdon
Prentince Hall
1989

[7] Visual C++ 6.0 Manual de Referencia
Chris H. Pappas y William H. Murray
McGraw-Hill
1999

[8] Java 2 Black Book
Steven Holzner
Coriolis
2001

[9] Cómo Programar en Java
Deitel y Deitel
Prentince may
1998

[10] Select... SQL The Relational Database Language
Larry R. Newcomer
MacMillan Publishing Company
1992