UNIVERSIDAD NACIONAL DE INGENIERÍA
RECINTO UNIVERSITARIO SIMÓN BOLÍVAR
UNI - RUSB

Ingeniería de Software II

Análisis y Diseño Orientado a Objetos

DOCENTE: Magda Luna.

INTEGRANTES:
Abraham Galeano C.
Reynaldo Porras G.
Osman Gutiérrez S.

GRUPO: 5T2-Co.
Grupo de Trabajo: #4

Managua, Miércoles 18 de Mayo del 2005.

Índice

Introducción

En nuestro mundo se encuentran un sin número de objetos, estos objetos existen como entidades hechas por el hombre, negocios y productos que se usan en la vida diaria. Todos estos objetos pueden ser clasificados, descritos, organizados, combinados, manipulados y creados.

La idea básica de la programación orientada a objetos se basa en 8 principios, que se muestran para un mejor entendimiento de la metodología:

- **Clases:** Una clase es una categoría de objetos similares. Los objetos se agrupan en clases.

- **Herencia:** Las clases pueden tener hijos, esto es, una clase puede ser creada a partir de otra clase. La clase original, o madre, es llamada "clase base". La clase hija es llamada "clase derivada". Una clase derivada puede ser creada en forma tal que herede todos los atributos y comportamientos de la clase base.

- **Objetos:** Un objeto es una representación en una computadora de alguna cosa o evento del mundo real.

- **Encapsulación:** Típicamente, la información dentro de un objeto esta encapsulada por su comportamiento. Esto significa que un objeto mantiene datos acerca de cosas del mundo real a las que representa en un sentido verdadero.

- **Atributo:** dato asociado a un objeto.

- **Mensajes:** Se puede enviar información de un objeto a otro.

- **Método:** proceso que realiza un objeto cuando recibe un mensaje.

- **Polimorfismo:** El término polimorfismo se refiere a comportamientos alternos entre clases derivadas relacionadas. Cuando varias clases heredan atributos y comportamientos, pude haber casos en donde el comportamiento de una clase derivada debe ser diferente del de su clase base o de sus clases derivadas parientes.

Un enfoque orientado a objetos, dependiendo de la naturaleza del software a desarrollar, puede facilitar la elaboración de la aplicación, debido a que los objetos describen de forma abstracta a los elementos del mundo en que vivimos.

Los beneficios de la tecnología orientada a objetos se fortalecen si se usa antes y durante el proceso de desarrollo del software una metodología de análisis y diseño orientada a objetos. Para obtener los mejores resultados se deben considerar el análisis de requisitos orientados a objetos, diseño orientado a objetos, análisis del dominio orientado a objetos, entre otros.

En el análisis orientado a objetos se desarrollan una serie de modelos que describen el software de computadora a trabajar para satisfacer un conjunto requisitos

definidos por el cliente. El modelo del análisis orientado a objetos ilustra información, funcionamiento y comportamiento.

El diseño orientado a objetos transforma el modelo del análisis en un modelo de diseño que sirve como anteproyecto para la construcción d software.

Existen diferentes metodologías orientadas al análisis y diseño orientado a objetos, entre estas se encuentran:

- El método de Booch: este método abarca un micro proceso de desarrollo y un macro proceso de desarrollo tanto para el análisis como para el diseño. El nivel micro define un conjunto de tareas de análisis que se reaplican en cada etapa en el macro proceso. El micro proceso identifica clase y objetos, define relaciones entre clases y objetos y realizan una serie de refinamientos para elaborar el modelo del análisis. El macro proceso, en el diseño, engloba una actividad de planificación arquitectónica, que agrupa objetos similares en particiones arquitectónicas separadas, capas de objetos por nivel de abstracción, identifica situaciones relevantes, crea un prototipo de diseño y valida el prototipo aplicándolo a situaciones de uso.

- El método de Rumbaugh: este método mejor conocido como OMT, se utiliza para el análisis, diseño del sistema y diseño a nivel de objetos. La de análisis crea tres modelos: el modelo de objetos, el modelo dinámico y el modelo funcional. El diseño se divide en dos actividades diseño de sistemas y diseño de objetos.

- El método de Jacobson: también llamado OOSE (que en español significa ingeniería del Software Orientada a Objetos), es una versión simplificada de Objectory, un método patentado, también desarrollado por Jacobson. Este método, en el análisis, se diferencia de los otros por la importancia que da al caso de uso. En principio, el modelo idealizado del análisis se adapta para acoplarse al ambiente del mundo real. Después los objetos de diseño primarios, llamados bloques, son creados y catalogados como bloques de interfaz, bloques de entidades y bloques de control. La comunicación entre bloques durante la ejecución se define y los bloques se organizan en subsistemas.

En este documento se abordará el análisis y diseño orientado a objetos empleando la metodología de Coad y Yourdon la cual es sencilla de aprender, a continuación se detalla los pasos que se siguen en esta metodología.

Análisis Orientado a Objetos

El enfoque de Coad y Yourdon para el análisis orientado a objetos esta basado en cinco capas. Esas cinco capas consisten de capa clase /objeto, capa de estructura, capa de atributos, capa de servicios, y capa de tema. Estas capas dan mayor poder a la representación de la complejidad del análisis y el diseño en sistemas flexibles.

1. **Capa Clase Objeto**: Esta capa del análisis y diseño indica las clases y objetos.
2. **Capa de Estructura**: Esta capa captura diversas estructuras de clases y objetos, como las relaciones uno a muchos.
3. **Capa de Atributos**: Esta capa detalla los atributos de las clases.
4. **Capa de Servicios**: Esta capa indica los mensajes y comportamientos de los objetos.
5. **Capa de Tema**: Esta capa divide el diseño en unidades de implementación o asignaciones de equipos.

Análisis de Clases y Objetos

Objeto: Es una abstracción de algo en un dominio de un problema que refleja las capacidades de un sistema para llevar información acerca de ello, interactuar con ello o ambas cosas. Una encapsulación de valores de atributos y sus servicios exclusivos.

Clase: Una descripción de uno o más objetos con un conjunto de atributos y servicios uniformes, incluyendo una descripción de cómo crear nuevos objetos en la clase.

Clase y Objeto: Un término que se refiere tanto a la clase como a los objetos que ocurren en la clase.

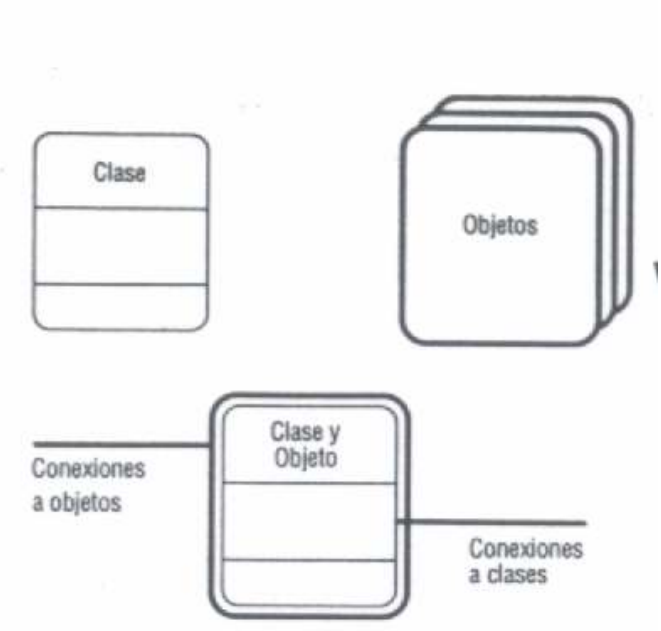
En el análisis de clases y Objetos se identifican las clases, los objetos y las clases y objetos candidatos para la aplicación a desarrollar. Las clases, los objetos y las clases y objetos se pueden obtener haciendo un análisis gramatical de la descripción del problema, subrayando términos que podrían ser representados con objetos dentro del sistema. Dentro del análisis gramatical de la descripción del problema, los objetos se pueden manifestar de la siguiente manera:

- Entidades externas: (otros sistemas, dispositivos, gente) que producen o consumen información a ser utilizada en el sistema.
- Cosas: (informes, visualizaciones, cartas, señales) forman parte del dominio de información del problema.
- Ocurrencias o Sucesos: (una transferencia de una propiedad o la terminación de una serie de movimientos de un robot) ocurren en el contexto de operación del sistema.

- Papeles: (gestor, ingeniero, vendedor) son jugados por personas que interactúan con el sistema.
- Unidades organizativas: (división, grupo, equipo) son relevantes para la aplicación.
- Lugares: (sala de facturación, muelle de descarga) establecen el contexto del problema y el funcionamiento general del sistema.
- Estructuras: (sensores, vehículos de cuatro ruedas, computadoras) definen clases de objetos.

Analizando gramaticalmente el problema los objetos potenciales corresponderían con los nombres (sustantivos) de la narración. No son objetos los nombres procedimentales imperativos, por ejemplo: *inversión de imagen*, la imagen sería un objeto, inversión un procedimiento del objeto imagen (una operación sobre la imagen).

Para representar las clases, los objetos y las clases objetos, se utiliza la siguiente notación:



Las clases son representadas por cuadros rectangulares redondeados divididos en tres partes. El nombre de la clase se muestra en la división superior del cuadro. Las otras dos divisiones se usan para las capas de atributos y servicios. Cuando una clase aparece sin objetos, puede ser solamente una clase base, ya que la única razón para que tal clase exista, es que sea un medio para agrupar atributos y servicios que serán heredados por varias otras clases.

Los objetos que tienen ocurrencia de una clase son representados por un cuadro sombreado rodeado por la clase. Debido a que los objetos tienen ocurrencias de una clase, no es posible que objetos existan independientemente de su clase.

Coad y Yourdon proporcionan la notación clase y objeto para distinguir gráficamente entre estructuras y mensajes que están orientados hacia la clase (tales

como el mensaje “crear una nueva ocurrencia de un objeto”) de estructuras y mensajes orientados al objeto (“tal como “Quitar motor””).

A continuación se muestran criterios que se pueden usar para determinar si se justifica una nueva clase de objetos:

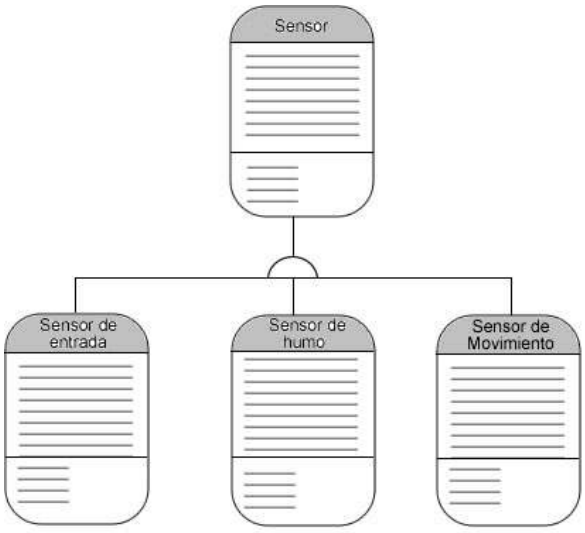
1. Hay necesidad de recordar el objeto. Esto es que si el objeto puede ser descrito en un sentido definido y sus atributos son relevantes para el problema.
2. Hay una necesidad de determinados comportamientos del objeto. Aunque un objeto no tenga atributos, hay servicios que debe proporcionar o estados de objeto que deben ser llamados.
3. Usualmente un objeto tendrá varios atributos. Los objetos que tienen solamente uno o dos atributos sugieren diseños sobre analizados.
4. Usualmente una clase tendrá más de una instancia de objetos, a menos que sea una clase base.
5. Usualmente los atributos tendrán siempre un valor significativo para cada objeto de la clase. Los objetos que producen un valor nulo para un atributo, o para los que no es aplicable un atributo, por lo general implican una estructura generalización-especialización.
6. Usualmente los servicios siempre se comportarán en la misma forma para todos los objetos de una clase. Los servicios que varían dramáticamente para algunos objetos de una clase o que regresan sin realizar acción para algunos objetos también sugieren una estructura generalización-especialización.
7. Los objetos deben implementar requerimientos que son derivados del problema y no de la tecnología de solución. La parte de análisis del proyecto Orientado a Objetos no debe ser dependiente de una tecnología de implementación particular. Los objetos que atienden tales detalles técnicos no deben aparecer sino hasta muy tarde en la etapa de diseño. Los dependientes de la tecnología sugieren que el proceso de análisis tiene fallas.
8. Los objetos no deben duplicar atributos o servicios que pueden ser derivados de otros objetos en el sistema.

Análisis de Estructuras

Hay dos tipos de estructuras que deben ser impuestas en las clases y objetos. Estas son las estructuras generalización-especialización (conocida como Gen-Spec) y la estructura Completo-parte.

1. Estructuras Generalización-Especialización. La herencia se crea con las estructuras Generalización-Especialización. Estas relaciones entre clases son a veces llamadas relaciones de clasificación, subtipo o ISA. Las estructuras Generalización-Especialización son indicadas por un

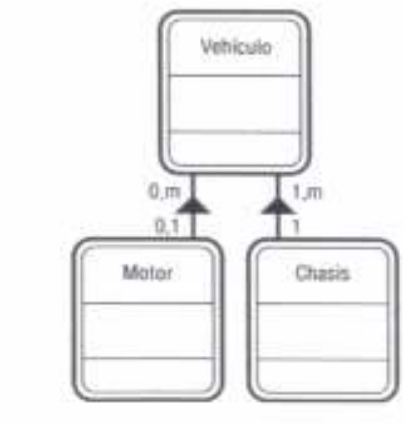
semicírculo con su lado redondo hacia la clase generalizada. Estas estructuras siempre conectan clases a clases. Son típicamente de forma jerárquica. A continuación se muestra un ejemplo de Gen-Spec, en donde la clase sensor se divide en otros tipos de sensores con particulares propias a cada uno de ellos.



Notación de Estructura de Clasificación

2. Estructuras Todo-Partes. Estas estructuras indican conjuntos diferentes de objetos que componen otro objeto completo. Tales relaciones entre objetos son a veces llamadas relaciones de ensambles, agregaciones o TIENEUN. Las estructuras Completo-parte son indicadas por un triángulo que apunta hacia el objeto "Completo". Estas estructuras conectan siempre objeto con objeto.

Las estructuras Todo-partes también tienen cardinalidad, representada por cero a uno, cero a mucho, uno a muchos y muchos a muchos. La cardinalidad es igual a la usada en el modelo entidad relación.



Análisis de Atributos

Aquí la idea básica de un atributo es la misma, es decir, que definen las propiedades de un objeto de datos. Sin embargo, tres nuevas ideas son relevantes desde la perspectiva orientada a objeto. Primero, los atributos son siempre más propensos al cambio que las clases. Si una estructura o un conjunto de clases parecen estar amontonándose, debido a que un objeto está cambiando de clase a clase, tal vez la Clase y Objeto en cuestión debiera simplemente ser un conjunto de atributos en otra clase. Segundo, los atributos deben ser mantenidos tan alto como sea posible en as estructuras Generalización-Especialización. Esta restricción reduce la programación y el mantenimiento, debido a que un cambio hecho en un objeto General será automáticamente heredado por todos los objetos especializados. Tercero, las asociaciones o relaciones entre los objetos deben ser detalladas como conexiones de ocurrencia, en vez de llaves foráneas.

Conexiones de Ocurrencia. En vez de amontonar el paquete de diseño con los detalles de llaves primarias y llaves foráneas, no se especifican los atributos de llave primaria. Por consecuencia, las referencias entre los objetos, como las asociaciones y relaciones, se indican por una solo línea entre los objetos con la misma notación de cardinalidad usada en las estructuras Todo-partes. Las conexiones de ocurrencia siempre suceden entre los objetos y no entre las clases.

Plantilla de Especificación Preliminar. Con la introducción de los atributos es necesario detalles de análisis adicionales para dar soporte al diagrama de capas. En esta etapa estos detalles toman en cuenta solamente descripciones de atributos y sus valores. Coad y Yourdon recomiendan una plantilla de especificación que proporcione un esquema similar al diccionario de datos.

Análisis de servicios

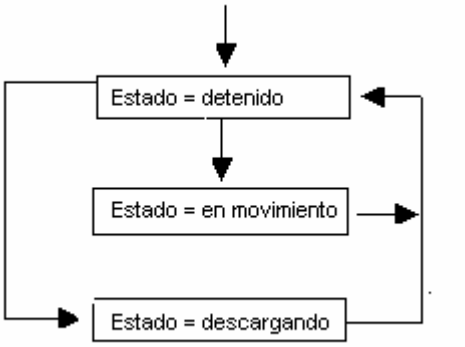
Los servicios pueden ser llamados también métodos o procedimientos y son una parte importante de los objetos así como los son sus atributos. Debido a que los servicios involucran frecuentemente cambios en el estado de un objeto, son comúnmente analizados y diseñados usando diagramas de estado.

El análisis de servicios consiste de tres actividades:

- **Análisis del estado del objeto**

Existen atributos en cada objeto que pueden afectar su cambio de estado, es importante por tanto en este análisis encontrar esos atributos. Es recomendable preguntarse en estos casos ¿Cambiará el comportamiento del objeto cuando cambie el valor de este atributo? Si ningún atributo logra causar tal efecto en el objeto, pueden haber ciertas condiciones que hagan que el objeto cambie su comportamiento, en estos casos podría ser útil añadir un atributo de "estado". Por ejemplo si analizando un carro de tren que tomará y dejará pasajeros, se sabe que el tren debe comportarse diferente en reacción a un mensaje "descargar pasajeros", dependiendo de si el tren está detenido en

una plataforma de estación o está en movimiento. Para tal objeto tren con una variable de estado, se puede representar un diagrama que se vería así:

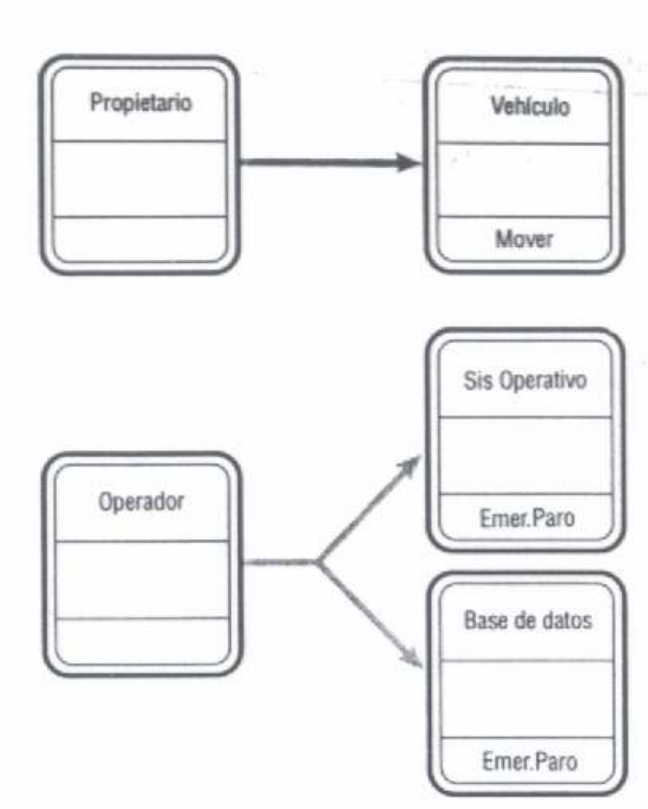


La flecha en la parte superior del cuadro Estado = detenido muestra que el estado inicial cuando el objeto es creado es siempre detenido. Las otras flechas muestran cambios de estado posibles, por ejemplo, de *detenido* a *en movimiento* o de *detenido* a *descargando*. No hay forma para cambiar estados de *en movimiento* a *descargando*. Esto previene lógicamente que el objeto descargue pasajeros mientras está en movimiento. Los diagramas de estado son añadidos conforme se necesita a las plantillas de especificación para documentar tales atributos de estado.

- **Especificación de servicio**

Los servicios pueden ser de dos tipos, simples o compuestos. Los servicios simples involucran muy pocas condiciones u operaciones, y frecuentemente se aplican a cada Clase y Objeto en un sistema. Estos incluyen servicios tales como crear objetos, almacenar objeto, recuperar objeto, conectar objeto (hacer una ocurrencia de conexión), y acceder objeto (obtener o poner los valores de los atributos) y borrar objeto. Los servicios simples son implícitos, a veces especificados una sola vez en el diseño y nunca vueltos a mencionar. Algunas veces incluso estos servicios no se mencionan en el diseño por ser muy obvios por lo que se les deja a los programadores que los construyan cuando los necesiten durante la implementación.

Los servicios complejos en cambio involucran ciclos, muchas operaciones o condiciones compuestas. Estos servicios por lo general se aplican a una Clase y Objeto. Los servicios complejos pueden ocasionar servicios “compañeros” o “privados” que son similares a los módulos de subrutina. Los servicios privados son subrutinas internas, de las cuales solo el objeto mismo sabe y puede activar. Los servicios compañeros son subrutinas usadas por servicios complejos, y también pueden ser activadas como servicios distintos por mensajes de otros objetos del sistema. Los nombres de tales servicios aparecen en la sección inferior de los cuadros de clases. La figura a continuación muestra tres servicios, Mover en el objeto Vehículo, Emer.Paro en el objeto Sis Operativo y Emer.Paro en el objeto Base de Datos. Se pueden usar diferentes herramientas para representar estos servicios, tales como diagramas de flujo de programas, diagramas Warnier-Orr, tablas de decisiones o español estructurado.



- **Especificación de mensajes**

Los mensajes detallan la manera en que el comportamiento de un objeto puede activar el comportamiento de otro objeto, esto con el fin de documentar la dependencia de un proceso sobre otro proceso en un objeto diferente. Los mensajes existen solamente para comunicarse entre servicios, y ocasionan flujo de control y flujo de datos.

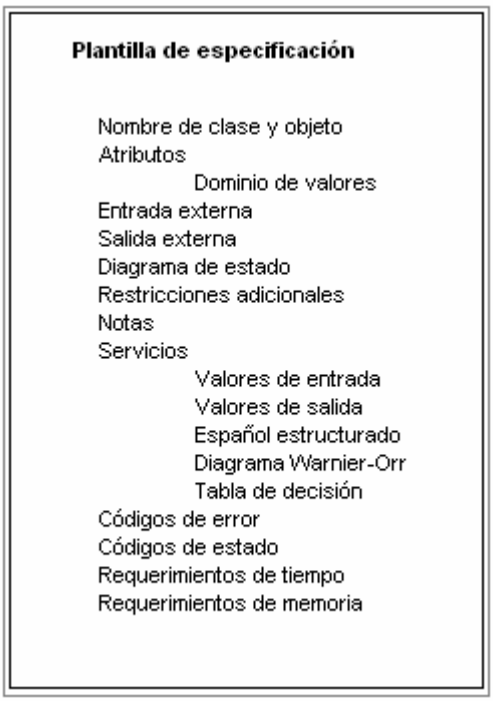
Los mensajes dirigidos a las clases, tales como crear objetos y borrar objeto, por lo general no son diagramados por que están implícitos en los servicios simples. Por tal razón los mensajes que sí se diagraman terminan siendo de objeto a objeto, y no de clase a clase o de objeto a clase. Los mensajes se representan por flechas direccionales como es el caso de las figuras del ejemplo anterior. Se puede notar en la parte inferior de la figura que un objeto le envía un mensaje a dos objetos, esto es lo que hace el objeto Operador, envía un solo mensaje que activa el comportamiento de paro de emergencia (Emer.Paro) en los objetos Base de Datos y Sis Operativo simultáneamente.

Debido a que los mensajes detallan servicios complejos, se alinean con el servicio emisor y el servicio receptor, sin embargo los servicios complejos son escasos en los sistemas y por consiguiente también los mensajes en el diagrama. De esta manera se resaltan las funciones realmente importantes del sistema. Los servicios complejos que no son activados por mensajes, tienden a ser activados por eventos temporales o interacción humana como es el caso de las clases sin mensajes de entrada.

Formato de plantilla de especificaciones

Las plantillas de especificación detallan todo lo relacionado con las clases y objetos, esta parte del análisis se inunda de detalles que pueden ser extensos, esto a causa de los niveles de complejidad en los diagramas de estado, español estructurado y diagrama de flujo. No hay un formato exacto que siempre deba de ser utilizar cuando se realiza esta plantilla, ésta puede variar de un análisis a otro.

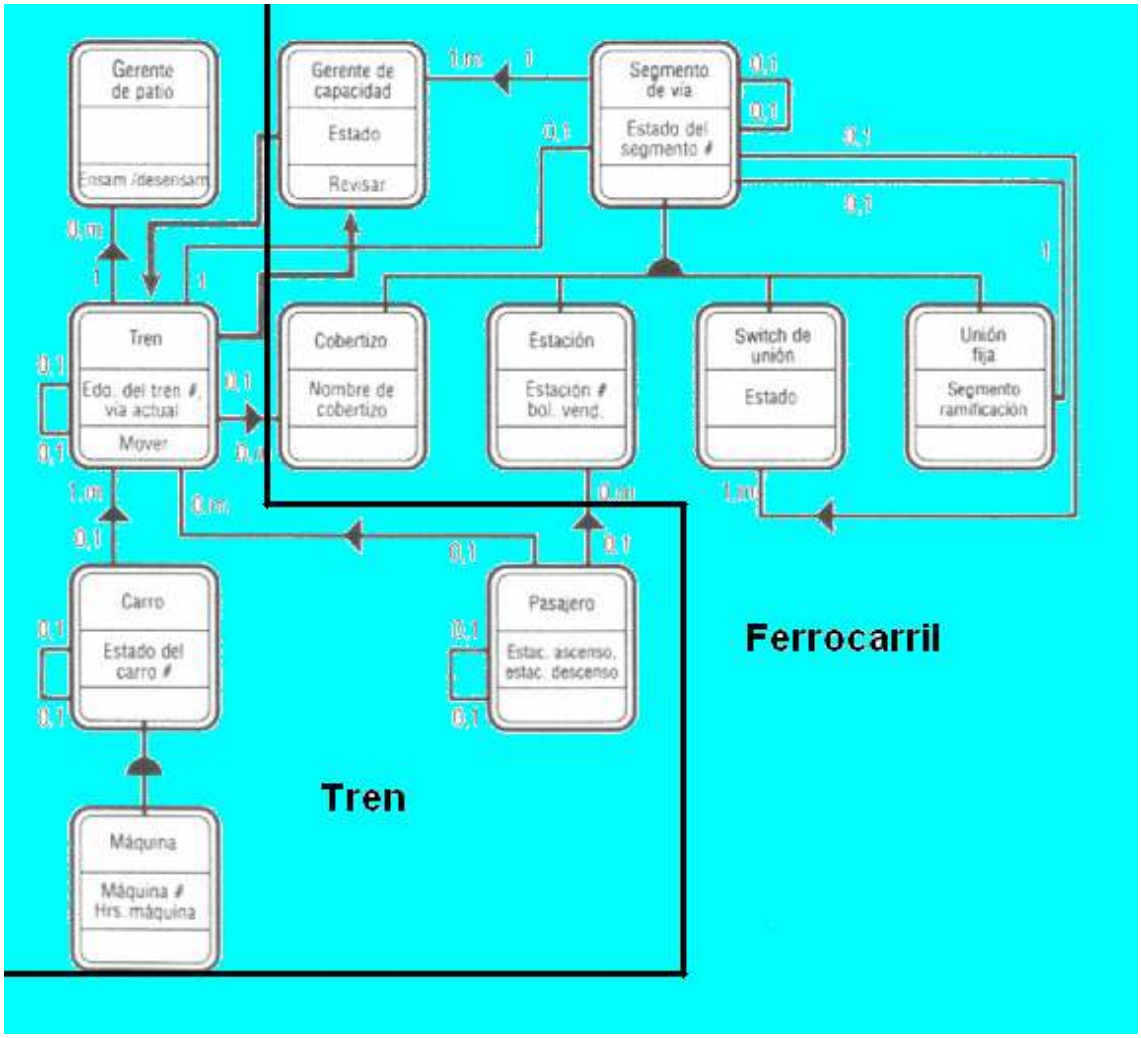
Un guión básico de plantilla se muestra a continuación:



Análisis de temas

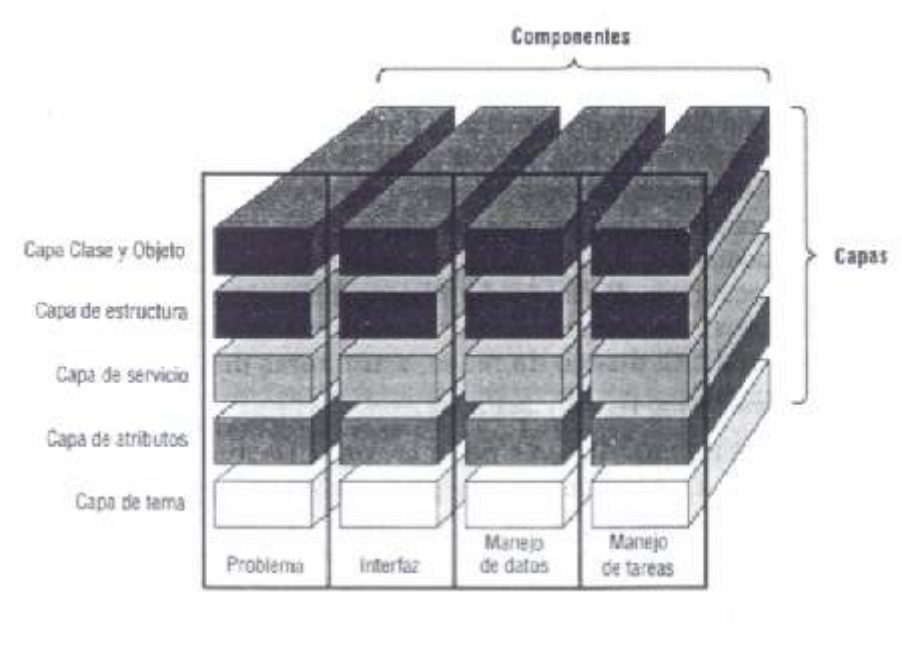
En el caso de sistemas muy grandes se puede usar una capa adicional en el paquete de diagrama en capas OO para organizar el trabajo de análisis, diseño e implementación. Esta capa permite subdividir una especificación compleja en unidades de trabajo lógicas, recomendable en proyectos grandes que involucran muchas clases.

Los temas son reasaltados trazando una línea de guiones ancha que indica las fronteras de un tema particular en el diagrama OO, anotando el nombre del tema en una esquina del cuadro del tema. Este nombre, por lo general es adquirido de la clase "propietaria", es decir será el mismo nombre que tiene la clase que está conectada centralmente con todas las clases y objetos en un espacio del tema.



Diseño Orientado a Objetos

El enfoque de Coad y Yourdon, plantea que el análisis es razonablemente independiente de la tecnología, en cambio el diseño viene a ser entonces cada vez más orientado hacia un lenguaje OO particular y a un ambiente de desarrollo. Las actividades de diseño OO están agrupadas en los cuatro componentes principales del sistema final: el componente del problema, el componente de interfaz humana, el componente de manejo de datos y el componente de manejo de tareas, todos ellos están expandido a lo largo de las cinco capas con que cuenta el diseño OO, mismas que se presentaron anteriormente en el análisis OO



Diseño del componente de dominio problema

El componente del dominio del problema (PDC) es el conjunto básico de objetos funcionales que llega de la etapa de análisis. Tales objetos directamente resuelven el problema que se pretende ser resuelto por el sistema que se está construyendo, lo que quiere decir que el diseño del PDC se termina en su mayor parte en la etapa de análisis, completándose ahora con la ejecución de tres actividades, las cuales son:

- **Diseño de reuso**

Se pueden añadir en esta etapa nuevas clases para reusar objetos que serán útiles más adelante. Es el caso de los paquetes comerciales de clase generalizada como las que contienen las organizaciones de programadores OO con experiencia, ellos por lo general poseen una biblioteca de clases desarrollada para los objetos. Estas bibliotecas y paquetes pueden contener clases que tienen atributos y servicios para objetos similares a los requeridos en el diseño del sistema a desarrollarse. Estas clases reusables pueden ser añadidas al diseño como clases bases en una estructura Gen-Spec.

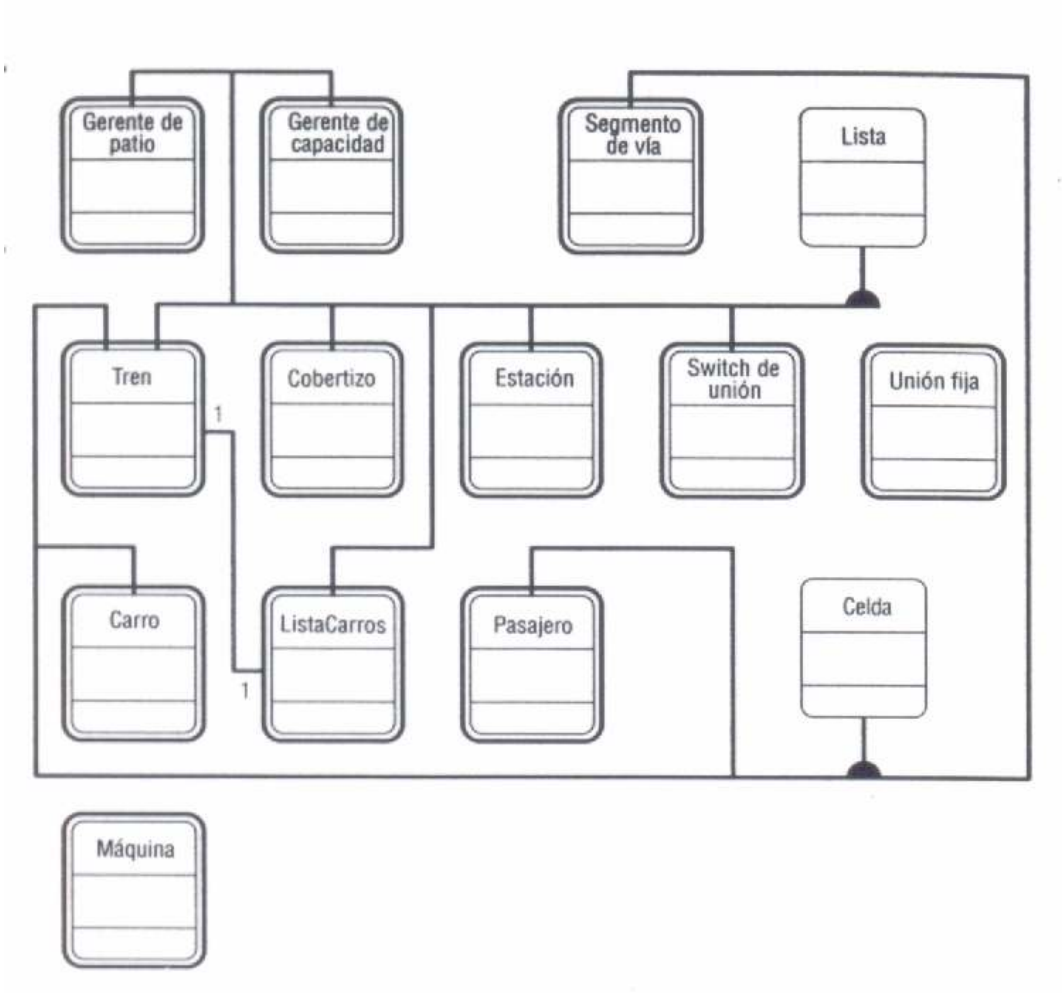
- **Estructura de Implementación**

Debido a la implementación en un lenguaje de programación en particular podría ser necesario que en el diseño se agreguen estructuras que pueden ser de agregación, o Gen-Spec, este último para permitir que varias clases de objetos compartan un protocolo o estructura de datos. Estas estructuras usan el concepto de herencia para hacer más fácil el enfoque de programación.

- **Acomodo al lenguaje**

En esta sección podemos corregir (si es necesario) el diseño para que las estructuras puedan ser construidas en el lenguaje de programación seleccionado, debido

a que estos lenguajes pueden tener diferentes patrones de herencia. Algunos lenguajes, por ejemplo, incluyen herencia múltiple (C++), otros solamente incluyen herencia simple (Java) y todavía otros que posiblemente no incluyen herencia. En los casos más restrictivos, los patrones de herencia del diseño deben ser modificados para permitir las capacidades del lenguaje de programación.



En la figura anterior se observa que el objeto Tren hereda de Listas y Celdas (clases propias de la biblioteca de C++), pero al no tener C++ forma de manejar dos estructuras Gen-Spec con Listas se tuvo que crear una segundo estructura llamada ListaCarrros y conectarlos uno a uno con el objeto Tren.

Diseño del Componente de Interfaz Humana

En esta actividad creamos los menús, reportes y pantallas interactivas que usarán las personas para trabajar con el sistema. Por lo general, se puede obtener ayuda en gran forma en clases de bibliotecas para el diseño de clases de Interfaz. Esta es un área donde la reusabilidad de las clases Orientado a Objetos ha probado ser muy efectiva. Las clases de bibliotecas generalmente proporcionan generalizaciones de menús, ventanas, control de tipo de letra, y utilerías de cortar y pegar.

Los prototipos son muy útiles durante el diseño de Interfaz para hacer más fácil la manera en que trabajarán las clases de biblioteca con los objetos del Dominio.

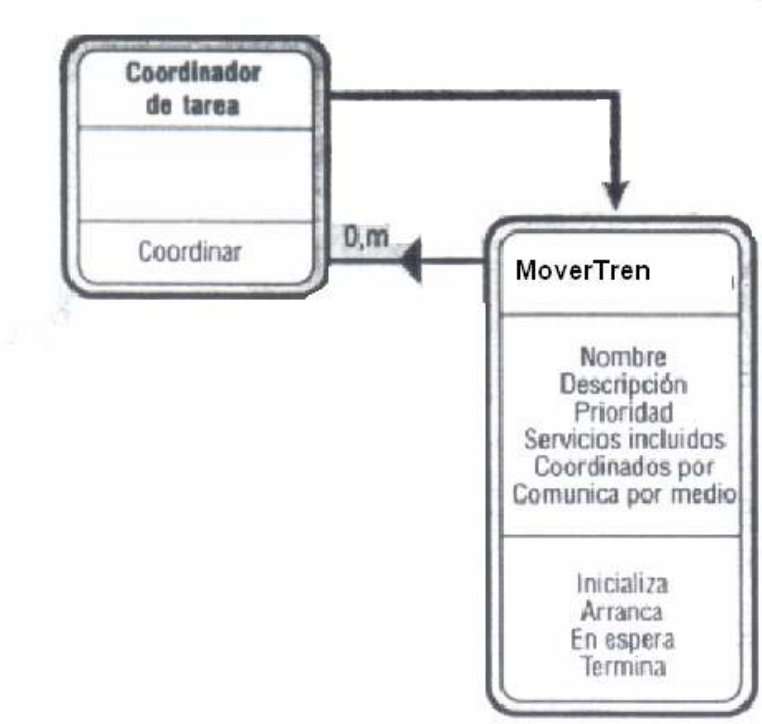
Por lo general, con la información obtenida en las entrevistas y casos de uso podemos recopilar información acerca de los perfiles de usuarios involucrados en el sistema y diseñar una interfaz correspondiente a su perfil. Con base a estos y otros perfiles, podemos seleccionar una interfaz

Diseño de componentes de administración de tarea y datos

Ambos componentes están estrechamente relacionados con la tecnología de implementación. El manejo de tareas esta muy determinado por la configuración de hardware de computación, y el manejo de datos esta muy determinado por el software de sistema disponible cuando el sistema este de hecho en ejecución.

El componente de manejo de tareas es más importante cuando el sistema está ejecutándose en varios procesadores o computadoras. Una "tarea"es un conjunto de servicios relacionados que deben ejecutarse juntos (tal ves en el mismo procesador). Las tareas son activadas por el tiempo transcurrido o por un evento. Los objetos del manejo de tarea obedecen a activadores de tareas, asignación de procesadores y prioridades cuando son llamados los servicios.

Un ejemplo de componente de tareas es como se muestra a continuación, en él, el tema del componente de manejo de tareas se añade al paquete de diagrama de capas existente. Este componente es implementado luego creando objetos Tarea conforme son necesarios por el sistema.



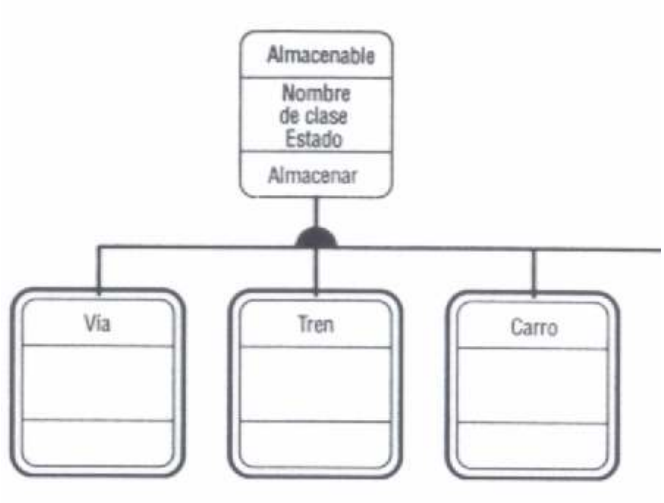
El componente de Manejo de Datos comprende, por lo general, clases y objetos necesarios para almacenar y recuperar a los otros objetos del sistema. El Componente Manejo de Datos varia considerablemente dependiendo de si la tecnología de tiempo de ejecución subyacente es una base de datos orientada a objetos, una base de datos relacional o un sistema de archivos "plano" ordinario. En un ambiente de Base de Datos relacional o de archivo plano el componente de manejo de datos debe proporcionar servicios de almacenamiento al sistema.

Hay tres formas diferentes para diseñar el Diagrama de Manejo de Datos:

1. Construir servicios de almacenamiento en cada Clase y Objetos en el diseño: Esto involucra, por lo general, una cantidad considerable de programación de diseño adicional.

2. Crear una Clase y Objeto, ServidorObjeto, que proporcione todos los servicios de Base de Datos: Involucra un complejo objeto que sepa cómo guardar o recuperar todos los objetos del sistema. Cualquier petición de almacenamiento se hace por medio de mensajes a este único objeto cuyo diseño podría ser como el que se muestra a continuación.

3. Crear una clase Almacenable: Es una combinación de los dos enfoques anteriores. Cada objeto del sistema que deba ser guardado o recuperado es conectado luego a una estructura Gen-Spec con la clase almacenable. La figura a continuación es un ejemplo de ésta.



Conclusiones

El análisis, diseño y programación orientada a objetos, ha sido desarrollado para responder a las necesidades de flexibilidad en los Sistema de información basados en computadora. La encapsulación, herencia y polimorfismo, tienen como objeto proporcionar sistemas complejos con mecanismos para un rápido, fácil y confiable mantenimiento y cambio de los programas. Aunque el desarrollo Orientado a Objeto típico involucra una fase de análisis y diseño más amplia, esta inversión se traduce en menores costos de operación de los sistemas que es probable que requiera una gran actividad de mantenimiento.

Existen varias metodologías orientadas a objetos, a pesar que tienen variantes entre ellas, todas trabajan con el mismo paradigma por tanto se basan en los mismo fundamentos de modelación de objetos. Este trabajo se enfoca en la técnica de Análisis y Diseño de Coad y Yourdon, por considerase sencilla al momento de aplicar para analistas con poca experiencia.

Las técnicas para el análisis y diseño Orientadas a Objetos todavía están en desarrollo, esto debido a que la programación misma todavía se encuentra en esta etapa. Por consiguiente han surgido tantas metodologías que tratan este modelo de programación, llegando a destacarse un enfoque de Modelación de Lenguaje Unificado (UML) como uno de los más prácticos y eficientes. Sin embargo, no existe una técnica que se haya definido como estándar para este paradigma.