ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

DESARROLLO DEL SISTEMA WEB MÉDICO–ADMINISTRATIVO, "HEALTH AND WELFARE", PARA CENTROS MÉDICOS

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO EN ANÁLISIS DE SISTEMAS INFORMÁTICOS

DIEGO GABRIEL YUPANGUI SANTAMARÍA

dygs@hotmail.es

DIRECTOR: Msc. ROSA NAVARRETE

dicc@epn.edu.ec

Quito, enero 2010

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por DIEGO GABRIEL YUPANGUI SANTAMARÍA, bajo mi supervisión.

_____

Msc. ROSA NAVARRETE
DIRECTORA DE PROYECTO

DECLARACIÓN

Yo, DIEGO GABRIEL YUPANGUI SANTAMARÍA, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

.

_____

Diego Gabriel Yupangui Santamaría

AGRADECIMIENTOS

El presente trabajo va dirigido con una expresión de gratitud y amor por todo el apoyo, comprensión y amor incondicional de mi Madre Consuelo y Padre Gustavo, a quienes les agradezco de todo corazón por estar junto a mí siempre.

Agradezco a Dios por iluminar mi camino, ser mi amigo incondicional y mi guía día a día.

Y a mi querida POLI que en sus aulas aprendí el significado de ser un buen profesional y un hombre de bien para la sociedad.

Diego

DEDICATORIA

El presente trabajo está dedicado a mis padres por darme la fortaleza frente a las adversidades y no dejarme derrotar. Les quiero mucho, gracias por ayudarme a ser quien soy.

El presente trabajo también es el reflejo de mi trabajo, esfuerzo y dedicación que he puesto durante los años de estudio

Diego

TABLA DE CONTENIDO

CAPITULO 1. INTRODUCCIÓN.....1
1.1 ÁMBITO DEL PROBLEMA.....1
1.1.1 PLANTEAMIENTO DEL PROBLEMA.....1
1.2 OBJETIVOS.....1
1.2.1 OBJETIVO GENERAL.....1
1.2.2 OBJETIVOS ESPECÍFICOS.....1
1.3 ALCANCE.....1
1.4 JUSTIFICACIÓN.....2
CAPITULO 2. MARCO TEÓRICO..... 3
2.1 INGENIERÍA DE SOFTWARE.....3
2.1.1 PROCESO DE INGENIERÍA DE SOFTWARE.....3
2.1.2 CARACTERÍSTICAS DEL PROCESO.....3
2.1.3 DIFERENCIAS ENTRE INGENIEÍA DE SOFTWARE E INGENIERÍA WEB.....4
2.2 INGENIERÍA WEB.....4
2.2.1 EL PROCESO DE INGENIERÍA WEB 5
2.2.2 CONTROL Y GARANTÍA DE LA CALIDAD..... 6
2.2.3 CONTROL DE LA CONFIGURACIÓN 6
2.2.4 LA GESTIÓN DEL PROCESO 7
2.3 ARQUITECTURA MULTINIVEL (WEB) 8
2.3.1 APLICACIONES MULTINIVEL 8
2.4 HERRAMIENTAS DE DESARROLLO 10
2.5 ESTANDARES DE DISEÑO WEB 16
2.6 DESARROLLAR CON ESTÁNDARES TIENE MUCHAS VENTAJAS..... 17
CAPITULO 3. ASPECTOS METODOLÓGICOS..... 19
3.1 ASPECTOS METODOLÓGICOS..... 19
3.1.1 Proceso de Ingeniería Web 19
3.1.2 PARADIGMA ESPIRAL ORIENTADO A LA WEB (Modelo del proceso IWeb) 19
3.2 METODOLOGÍA: OOHDM (Object Oriented Hypermedia Design Methodology) 27
3.2.1 Introducción OOHDM..... 27
3.2.2 Fases de la Metodología OOHDM 28
3.2.3 Ventajas de OOHDM 34
3.3 UML (Unified Modeling Language o Lenguaje Unificado de Modelamiento)..... 35

3.3.1 Introducción UML 35
3.3.2 Objetivos del UML 36
3.3.3 Arquitectura del UML 36
UML define cuatro tipos de relación en los Diagramas de Casos de Uso: 43
CAPITULO 4. CONCLUSIONES Y RECOMENDACIONES 76
4.1 CONCLUSIONES 76
4.2 RECOMENDACIONES 77
BIBLIOGRAFÍA 78
ANEXOS 80

CAPITULO 1. INTRODUCCIÓN

1.1 AMBITO DEL PROBLEMA

1.1.1 PLANTEAMIENTO DEL PROBLEMA

En el sector de la salud es muy importante el manejo administrativo de horarios, turnos, médicos y pacientes, a fin de garantizar un mejor servicio a la hora de atender requerimientos tanto de pacientes como de médicos.

Las deficiencias en este manejo ocasionan problemas de inconsistencia y pérdida de información en las fichas clínicas de los pacientes, falta de coordinación entre turnos médicos, lo que genera altos tiempos de espera de los pacientes y la consiguiente incomodidad.

Se propone la creación de un sistema en plataforma web, que facilite el manejo administrativo de pacientes y mejore los servicios que se prestan en el ámbito médico. Adicionalmente se conseguiría que el Centro Médico sea competitivo frente a sus similares.

1.2 OBJETIVOS

1.2.1 OBJETIVO GENERAL

Mejorar la gestión administrativa en los Centros Médicos a través del desarrollo y posterior implantación del sistema Web.

1.2.2 OBJETIVOS ESPECÍFICOS

- Facilitar el acceso a la información garantizando su seguridad a través de la creación de perfiles de usuarios.
- Desarrollar y trabajar sobre un módulo específico por fichas o historiales de pacientes, el cual permita llevar la información de todos los acontecimientos clínicos de los pacientes, e incluso permitirá

realizar un registro sencillo de pacientes mediante una interface dinámica e intuitiva.

- Desarrollar un módulo dedicado exclusivamente a los médicos del Centro Médico, el mismo que facilite establecer las actividades que los médicos podrán llevar a cabo, dentro de sus horarios y disposiciones.
- Desarrollar un módulo de turnos, que integre disponibilidad de médicos de manera que los pacientes posean una facilidad a la hora de su atención.

1.3 ALCANCE

El sistema llevará cada dependencia de administración central, organismo y unidad médica de manera que llegue a completar los requerimientos que satisfagan la administración de los Centros Médicos tales como control de turnos, datos de pacientes y disponibilidad médica.
El sistema no contempla ofrecer:

- Interfaz funcional como el sistema Contable-Financiero

1.4 JUSTIFICACIÓN

El desarrollo del sistema Web permitirá al sector de la salud emplear de manera exhaustiva y eficiente datos e información; con el fin de mejorar la gestión administrativa y cumplir los requerimientos que satisfagan la administración de los Centros Médicos.

CAPITULO 2. MARCO TEÓRICO

2.1 INGENIERÍA DE SOFTWARE

Es la rama de la ingeniería que se encarga del estudio de los principios y metodologías de la ciencia de la computación y las matemáticas para desarrollar y mantener software de calidad.

Fuente: *http://www.angelfire.com/scifi/jzavalar/apuntes/IngSoftware.html*

2.1.1 PROCESO DE LA INGENIERÍA DE SOFTWARE

Es el conjunto estructurado de actividades requeridas para desarrollar un sistema de software.

- Especificación.
- Diseño.
- Validación.
- Evolución.

Las actividades varían dependiendo de la organización y del tipo de sistema a desarrollarse.

Debe estar explícitamente modelado si va a ser bien administrado.

2.1.2 CARACTERÍSTICAS DEL PROCESO

- Entendible ¿Se encuentra el proceso bien definido y es entendible?
- Visible ¿El proceso es visible al exterior?
- Soportable ¿Puede el proceso ser soportado por herramientas CASE?
- Aceptable ¿El proceso es aceptado por aquellos involucrados en el?

- Confiable ¿Los errores del proceso son descubiertos antes de que se conviertan en errores del producto?
- Robusto ¿Puede continuar el proceso a pesar de problemas inesperados?
- Mantenible ¿Puede el proceso evolucionar para cumplir con los objetivos Organizacionales?
- Rapidez ¿Que tan rápido puede producirse el sistema?

2.1.3 DIFERENCIA DE LA INGENIERÍA WEB CON LA INGENIERÍA DE SOFTWARE

A modo de breve resumen enumeramos las siguientes diferencias:

Confluencia de disciplinas: Sistemas de Información, Ingeniería Software y Diseño Gráfico que requiere equipos multidisciplinares y polivalentes.

Ciclos de vida y tiempo de desarrollo muy cortos - Cambio continuo: Necesidad de soluciones que permitan flexibilidad y adaptación conforme el proyecto cambia.

Requisitos fuertes de Seguridad, Rendimiento y Usabilidad.

Fuente: *http://dis.unal.edu.co/~fgonza/courses/2003/ingSoft1/CAP1.pdf*

2.2 INGENIERÍA WEB

Es el proceso utilizado para crear, implantar y mantener aplicaciones y sistemas *Web* de alta calidad. Esta breve definición nos lleva a abordar un aspecto clave de cualquier proyecto como es determinar que tipo de proceso es más adecuado en función de las características del mismo.[1]

[1] Murugesan et al., promotores iniciales del establecimiento de la *Ingeniería Web* como nueva disciplina, dan la siguiente definición: *"Web Engineering is the establishment and use of sound scientific, engineering and management*

2.2.1 EL PROCESO DE INGENIERÍA WEB

Características como inmediatez y evolución y crecimiento continuos, nos llevan a un proceso incremental y evolutivo, que permite que el usuario se involucre activamente, facilitando el desarrollo de productos que se ajustan mucho a lo que éste busca y necesita.

Según Pressman, las actividades que formarían parte del marco de trabajo incluirían las tareas abajo enumeradas. Dichas tareas serían aplicables a cualquier aplicación W*eb,* independientemente del tamaño y complejidad de la misma.

Las actividades que forman parte del proceso son: **Fuente:** *Pressman, 813.*

- Formulación,
- Planificación análisis,
- Modelización,
- Generación de páginas,
- test y evaluación del cliente.

La **Formulación** identifica objetivos y establece el alcance de la primera entrega.
La **Planificación** genera la estimación del coste general del proyecto, la evaluación de riesgos y el calendario del desarrollo y fechas de entrega.

El **Análisis** especifica los requerimientos e identifica el contenido.

La **Modelización** se compone de dos secuencias paralelas de tareas. Una consiste en el diseño y producción del contenido que forma parte de la aplicación. La otra, en el diseño de la arquitectura, navegación e interfaz de usuario. Es importante destacar la importancia del diseño de la interfaz.

---

principles and disciplined and systematic approaches to the successful development, deployment and maintenance of high quality Web-based systems and applications." (Murugesan, 3)

Independientemente del valor del contenido y servicios prestados, una buena interfaz mejora la percepción que el usuario tiene de éstos.

En la **Generación de páginas** se integra contenido, arquitectura, navegación e interfaz para crear estática o dinámicamente el aspecto más visible de la aplicación, las páginas.

2.2.2 CONTROL Y GARANTÍA DE LA CALIDAD

Una de las tareas colaterales que forman parte del proceso es el Control y Garantía de la Calidad (CGC). Todas las actividades CGC de la ingeniería software tradicional como son: establecimiento y supervisión de estándares, revisiones técnicas formales, análisis, seguimiento y registro de informes, etc, son igualmente aplicables a la *Ingeniería Web*.

Fuente: *Olsina, 266*

Sin embargo, en la *Web* toman especial relevancia para valorar la calidad aspectos como: Usabilidad, Funcionabilidad, Fiabilidad, Seguridad, Eficiencia y Mantenibilidad.

2.2.3 CONTROL DE LA CONFIGURACIÓN

Establecer mecanismos adecuados de control de la configuración para la *Ingeniería Web* es uno de los mayores desafíos a los que esta nueva disciplina se enfrenta. La *Web* tiene características únicas que demandan estrategias y herramientas nuevas. Hay cuatro aspectos importantes a tener en cuenta en el desarrollo de tácticas de control de la configuración para la *Web* (S. Dart):

Contenido: Considerando la dinamicidad con la que el contenido se genera, es tarea compleja organizar racionalmente los objetos que forman la configuración y establecer mecanismos de control.

Personal: Cualquiera realiza cambios. Hay mucho personal no especializado que no reconoce la importancia que tiene el control del cambio.

Escalabilidad: Es común encontrar aplicaciones que de un día para otro crecen considerablemente. Sin embargo, las técnicas de control no escalan de forma adecuada.

Política: ¿Quién posee la información? ¿Quién asume la responsabilidad y coste de mantenerla?

2.2.4 LA GESTIÓN DEL PROCESO

En un proceso tan rápido como es el proceso de *Ingeniería Web*, donde los tiempos de desarrollo y los ciclos de vida de los productos son tan cortos, ¿merece la pena el esfuerzo requerido por la gestión? La respuesta es que dada su complejidad es imprescindible.

Entre los aspectos que añaden dificultad a la gestión destacamos:

- Alto porcentaje de contratación a terceros,
- El desarrollo incluye una gran variedad de personal técnico y no técnico trabajando en paralelo,
- El equipo de desarrollo debe dominar aspectos tan variopintos como, software basado en componentes, redes, diseño de arquitectura y navegación, diseño gráfico y de interfaces, lenguajes y estándares en Internet, test de aplicaciones *Web*, etc, lo que hace que el proceso de búsqueda y contratación de personal sea arduo.

2.3 ARQUITECTURA MULTINIVEL (WEB)

2.3.1 APLICACIONES MULTINIVEL

Al hablar del desarrollo de aplicaciones Web resulta adecuado presentarlas dentro de las aplicaciones multinivel. Los sistemas típicos cliente/servidor pertenecen a la categoría de las aplicaciones de dos niveles. La aplicación reside en el cliente mientras que la base de datos se encuentra en el servidor. En este tipo de aplicaciones el peso del cálculo recae en el cliente, mientras que el servidor hace la parte menos pesada, y eso que los clientes suelen ser máquinas menos potentes que los servidores.

Además, está el problema de la actualización y el mantenimiento de las aplicaciones, ya que las modificaciones a la misma han de ser trasladada a todos los clientes. Para solucionar estos problemas se ha desarrollado el concepto de arquitecturas de tres niveles: interfaz de presentación, lógica de la aplicación y los datos.

La capa intermedia es el código que el usuario invoca para recuperar los datos deseados.

La capa de presentación recibe los datos y los formatea para mostrarlos adecuadamente. Esta división entre la capa de presentación y la de la lógica permite una gran flexibilidad a la hora de construir aplicaciones, ya que se pueden tener múltiples interfaces sin cambiar la lógica de la aplicación.

La tercera capa consiste en los datos que gestiona la aplicación. Estos datos pueden ser cualquier fuente de información como una base de datos o documentos XML.

Convertir un sistema de tres niveles a otro multinivel es fácil ya que consiste en extender la capa intermedia permitiendo que convivan múltiples aplicaciones en lugar de una sola

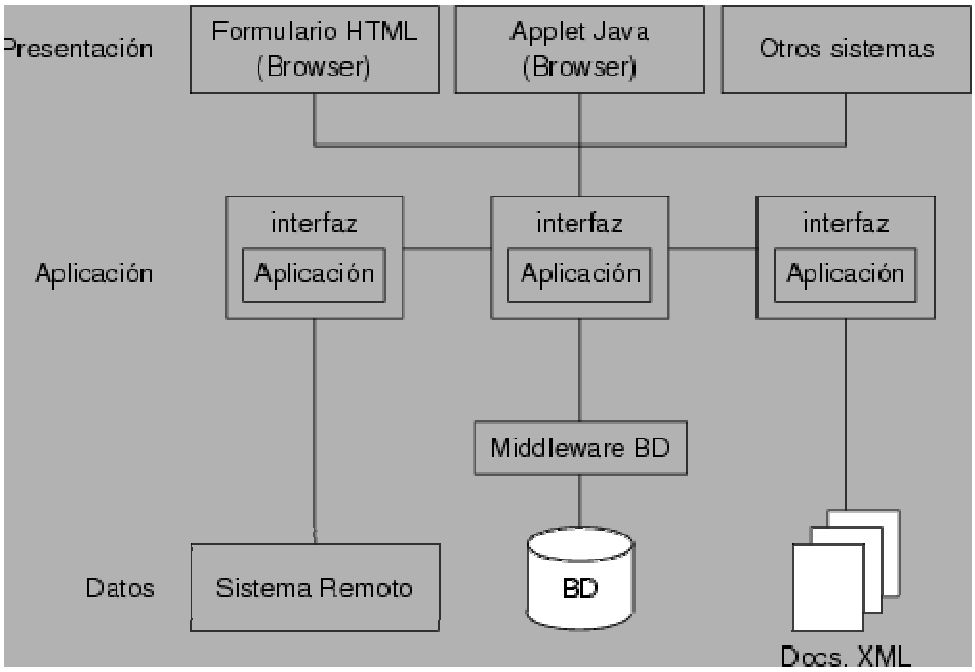


Figura 1: Arquitectura Multinivel.

Fuente: *http://www.infor.uva.es/~jvegas/cursos/buendia/pordocente/node21.html*

La arquitectura de las aplicaciones Web suelen presentar un esquema de tres niveles (véase la Figura 2.). El primer nivel consiste en la capa de presentación que incluye no sólo el navegador, sino también el servidor web que es el responsable de dar a los datos un formato adecuado. El segundo nivel está referido habitualmente a algún tipo de programa o *script*. Finalmente, el tercer nivel proporciona al segundo los datos necesarios para su ejecución.

Una aplicación Web típica recogerá datos del usuario (primer nivel), los enviará al servidor, que ejecutará un programa (segundo y tercer nivel) y cuyo resultado será formateado y presentado al usuario en el navegador (primer nivel otra vez).

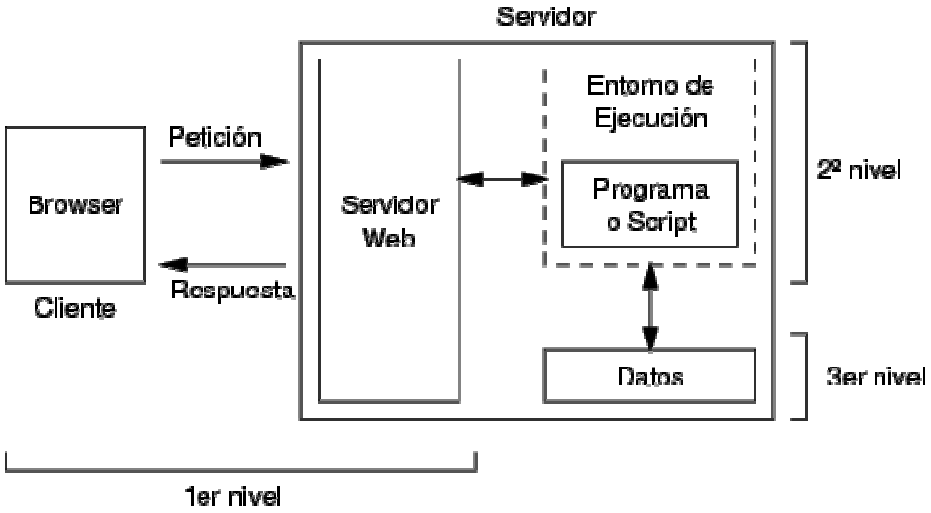


Figura 2: Arquitectura Web de tres niveles.

Fuente: *http://www.infor.uva.es/~jvegas/cursos/buendia/pordocente/node21.html*

Lamentablemente, el uso de toda esta tecnología pasa por el dominio de técnicas de programación y de acceso a bases de datos, condición esta que no se puede presuponer en un curso de divulgación como éste. Así, nos vamos a restringir al uso de herramientas básicas a la hora de la construcción de un portal docente.

2.4 HERRAMIENTAS DE DESARROLLO

Nivel		Lenguaje
Cliente		HTML JavaScript
Servidor	Servidor Web	Apache
	Lenguaje de scripts	PHP
Servidor de Datos	Base de Datos	MySQL

2.4.1 LENGUAJE HTML

Este lenguaje estructura documentos. La mayoría de los documentos tienen estructuras comunes (títulos, parrafos, listas...) que van a ser definidas por este lenguaje mediante tags. Cualquier cosa que no sea una tag es parte del documento mismo.

Este lenguaje no describe la apariencia del diseño de un documento sino que ofrece a cada plataforma que le de formato según su capacidad y la de su navegador (tamaño de la pantalla, fuentes que tiene instaladas...). Por ello y para no fustrarnos, no debemos diseñar los documentos basándonos en como lucen en nuestro navegador sino que debemos centrarnos en proporcionar un contenido claro y bien estructurado que resulte fácil de leer y entender.

Ventajas de HTML

HTML tiene dos ventajas que lo hacen prácticamente imprescindibles a la hora de diseñar una presentación web y son:

- Su compatibilidad y su facilidad de aprendizaje debido al reducido número de tags que usa.
  http://www.webestilo.com/html/cap2a.phtml

2.4.2 JAVASCRIPT

JavaScript es un lenguaje interpretado, al igual que VisualBasic, Perl, TCL... sin embargo, posee una característica que lo hace especialmente idóneo para trabajar en Web, usado principalmente en "clientes web", ya que son los navegadores que utilizamos para viajar por ella los que interpretan (y por tanto ejecutan) los programas escritos en JavaScript. De esta forma, podemos enviar documentos a través de la Web que incorporan el código fuente de un programa, convirtiéndose de esta forma en documentos dinámicos, y dejando de ser simples fuentes de información estáticas.

Los programas en JavaScript no son la primera forma que conoce la Web para transformar información, dado que el uso de CGIs está ampliamente difundido. La diferencia básica que existe entre un programa CGI y uno escrito en JavaScript es que el CGI se ejecuta en el servidor de páginas Web mientras que el programa en Javascript se ejecuta en el cliente (es decir, en el navegador). Por regla general, el CGI necesita unos datos de entrada (que normalmente se proporcionan mediante un formulario), los procesa y emite un resultado en forma de documento HTML. Esto implica tres transacciones en la red.

La primera carga la página del formulario, la segunda envía los datos al servidor, y la tercera recibe la nueva página que ha generado el CGI. Por el contrario, los programas escritos en JavaScript se ejecutan en el navegador del cliente, sin necesidad de que intervenga el servidor. De esta forma, una sola transacción basta para cargar la página en la que se encuentra tanto el formulario, para los datos de entrada, como el programa en JavaScript que proporciona los resultados.

Fuente:*http://geneura.ugr.es/~victor/cursillos/curso_javascript_basico/jsb_intro.html*

Las dos principales características de JavaScript son:

- Es un lenguaje basado en objetos (es decir, el paradigma de programación es básicamente el de la programación dirigida a objetos, pero con menos restricciones),

- JavaScript es además un lenguaje orientado a eventos, debido por supuesto al tipo de entornos en los que se utiliza (Windows y sistemas X-Windows). Esto implica que gran parte de la programación en JavaScript se centra en describir objetos (con sus variables de instancia y métodos de "clase") y escribir funciones que respondan a movimientos del ratón, pulsación de teclas, apertura y cerrado de ventanas o carga de una página, entre otros eventos.

FUENTE:*http://geneura.ugr.es/~victor/cursillos/curso_javascript_basico/jsb_intro.html*

2.4.3 WAMP

Es la abreviación de Windows, Apache, Mysql y PHP y es un proyecto desarrollado por franceses. Este paquete instala la versión 2.2 de Apache, PHP5, MySQL 5.0.27, PhpMyAdmin y SQLitemanager en tu computadora.

Una de las ventajas de este paquete es la ofrecer la versión de PHP5 en el instalador y sus constantes actualizaciones. La instalación es bastante sencilla y destaca que los servicios de apache y de mysql se instalarán en Windows bajo los nombres de wampapache y wampmysqld para no interferir con otras instalaciones previas de estos servidores.

FUENTE: *http://www.en.wampserver.com/presentation.php*

2.4.4 APACHE WEB SERVER

Es un servidor web libre, es decir, el encargado de construir y devolver las páginas web que solicitan los navegadores. Su nombre procede de "a patchy server", por ser una versión "parcheada" en 1995 de uno de los primeros servidores web, el NCSA HTTPD, y actualmente corre en muy diversas plataformas (Unices, Windows, etc.). Debido a su licencia libre pero no copyleft, existen también versiones propietarias de Apache, aunque es desarrollado y mantenido por la comunidad del software libre a través de la Fundacion Apache.

2.4.5 PHP

(Pre-procesador Hipertexto) Es un lenguaje de programación implantado (enraízado) en HTML HTML-embebed scripting lenguaje. La mayoría de su sintaxis está prestada de los lenguajes de programación C, Java y Perl, con la inclusión de algunos rasgos únicos de PHP. La meta del lenguaje es permitir a los que desarrollan sitios Web escribir rápidamente páginas generadas dinámicamente.

El lenguaje PHP es un lenguaje de programación de estilo clásico, con esto quiero decir que es un lenguaje de programación con variables, sentencias condicionales, bucles, funciones.... No es un lenguaje de marcas como podría ser HTML, XML o WML. Está más cercano a JavaScript o a C, para aquellos que conocen estos lenguajes. Pero a diferencia de Java o JavaScript que se ejecutan en el navegador, PHP se ejecuta en el servidor, por eso nos permite acceder a los recursos que tenga el servidor como por ejemplo podría ser una base de datos.

El programa PHP es ejecutado en el servidor y el resultado enviado al navegador. El resultado es normalmente una página HTML pero igualmente podría ser una pagina WML.

Figura 3: Funcionamiento de PHP

Fuente: *http://www.webestilo.com/php/php00.phtml*

Al ser PHP un lenguaje que se ejecuta en el servidor no es necesario que su navegador lo soporte, es independiente del navegador, pero sin embargo para que sus páginas PHP funcionen, el servidor donde están alojadas debe soportar PHP.

Fuente: *http://www.webestilo.com/php/php00.phtml*

2.4.6 PHPMYADMIN

PhpMyAdmin es una utilidad que nos sirve para inteactuar con una base de datos de forma muy sencilla y desde una interfaz web. Nos sirve por ejemplo para crear bases de datos, tablas, borrar o modificar dados, añadir registros, hacer copias de seguridad, etc. Es una aplicación tan útil que casi todos los hosting con MySQL disponen de ella, por ello se analizará su instalación. Además, vamos a usarlo para crear los usuarios MySQL para así poder utilizar las bases de datos de forma segura. Al ser una aplicación escrita en PHP, necesita de Apache y MySQL para poder funcionar. *http://www.phpmyadmin.net*

2.4.7 MySQL

Es el servidor de bases de datos relacionales de fuente abierta más popular en el mundo. Su arquitectura lo hace extremadamente rápido y fácil de adaptar.

MySQL es un gestor de base de datos sencillo de usar y increíblemente rápido. También es uno de los motores de base de datos más usados en Internet, la principal razón de esto es que es gratis para aplicaciones no comerciales.
Las características principales de MySQL son:

- **Es un gestor de base de datos**. Una base de datos es un conjunto de datos y un gestor de base de datos es una aplicación capaz de manejar este conjunto de datos de manera eficiente y cómoda.

- **Es una base de datos relacional**. Una base de datos relacional es un conjunto de datos que están almacenados en tablas entre las cuales se establecen unas relaciones para manejar los datos de una forma eficiente y segura. Para usar y gestionar una base de datos relacional se usa el lenguaje estándar de programación SQL.

- **Es Open Source**. El código fuente de MySQL se puede descargar y está accesible a cualquiera, por otra parte, usa la licencia GPL para aplicaciones no comerciales.

- **Es una base de datos muy rápida**, segura y fácil de usar. Gracias a la colaboración de muchos usuarios, la base de datos se ha ido mejorando optimizándose en velocidad. Por eso es una de las bases de datos más usadas en Internet.

- **Existe una gran cantidad de software que la usa**.
  http://www.webestilo.com/mysql/intro.phtml

2.4.8 MACROMEDIA DREAMWEAVER CS3

Es una aplicación de desarrollo web WYSIWYG (What You See Is What You Get), creado por Macromedia. Es el programa de este tipo más utilizado en el sector del diseño y la programación web, por sus funcionalidades, su integración con otras herramientas como Macromedia Flash y, recientemente, por su soporte de los estándares del World Wide Web Consortium.

2.5 ESTANDARES DE DISEÑO WEB

En un principio la construcción de un sitio web era una tarea más o menos simple. Constaba de unas cuantas páginas enlazadas entre sí de forma sencilla, y cada una de ellas estaba formada por un código HTML básico, unas pocas imágenes y poco más.

Pero con el paso del tiempo las exigencias del mercado hicieron aparecer más y más lenguajes de programación basados en los protocolos TCP/IP, y especialmente en el HTTP, a la par que se introdujeron en el mundo web las tiendas virtuales y la banca electrónica, demandando sitios web capaces de poder operar con datos, con transacciones y con una interminable serie de nuevas aplicaciones concebidas para estos propósitos.

Esta circunstancia creó una situación inestable, en la que muchos sitios quedaban rápidamente obsoletos debido a las continuas innovaciones en navegadores y dispositivos.

Con los estándares Web podemos diseñar y generar sofisticados sitios de gran belleza garantizando que funcionen en el futuro. Por estándares nos referimos a lenguajes estructurales como XHTML y XML, lenguajes de presentación como CSS, modelos de objeto como el DOM del W3C y lenguajes de secuencia de comandos como ECMAScript.

2.6 DESARROLLAR CON ESTÁNDARES TIENE MUCHAS VENTAJAS

En general, la mayoría de los beneficios de usar estándares web parten de la premisa de separación de contenido, en XHTML, y presentación, en CSS. Entre algunas, de las muchas ventajas, podemos nombrar que los estándares:

- **Optimizan un sitio para motores de búsqueda (SEO).** Desarrollar con estándares produce código XHTML limpio y semántico. Puesto que la mayoría de los buscadores trabajan indexando el contenido de un sitio, éstos tienden a priorizar a aquellos sitios con menor cantidad de código basura o no estándar en el sitio.

- **Producen un sitio fácil de mantener.** El código resultante de un sitio con estándares es simple y se puede dividir en secciones, reduciendo la dependencia de un solo desarrollador y facilitando la comunicación entre varios grupos de trabajo en una empresa de desarrollo web (programación, maquetado, diseño, etc.).

- **Reducen el tamaño del sitio.** Debido a la eliminación de elementos HTML que formatean visualmente una página, agrupando toda la parte visual en CSS, el tamaño de un sitio se reduce drásticamente. Esto nos

garantiza que el sitio será cargado rápidamente aún en conexiones lentas, como celulares.

- **Son un paso necesario para la accesibilidad.** A nivel dispositivo, los estándares aseguran que el contenido de nuestro sitio estará disponible para cualquier dispositivo con conexión a Internet. A nivel personas, implica tener en cuenta que nuestro sitio puede estar siendo accedido por personas con discapacidades, ya sean físicas (discapacidad visual, motriz, etc.) o de entorno o (sin mouse, con pantallas demasiado chicas, etc.).

- **Otorgan mayor flexibilidad al desarrollador web.** Esto permite a los desarrolladores, por un lado, ocuparse solamente de la parte estructural del sitio (XHTML) y por otro lado, a los diseñadores nos da la versatilidad suficiente como para cambiar cualquier aspecto del diseño de un sitio. Como ejemplo, Wired Magazine (http://www.wired.com) cambia el esquema de colores de su sitio todas las semanas, y para hacerlo modifica solamente una línea de código en su XHTML.

FUENTE: *http://www.16-bits.com.ar/archivos/hacia-un-nuevo-modelo-para-construir-la-web/*

CAPITULO 3. ASPECTOS METODOLÓGICOS

3.1 ASPECTOS METODOLÓGICOS

3.1.1 PROCESO DE INGENIERÍA WEB

Es claramente incremental y evolutivo.

Por la naturaleza intensiva, tendremos:

- Amplia y diversa población de usuarios (obtención y modelado de requisitos)
- Arquitectura altamente especializada (exigencias en el diseño)

3.1.2 PARADIGMA ESPIRAL ORIENTADO A LA WEB (MODELO DEL PROCESO IWEB)

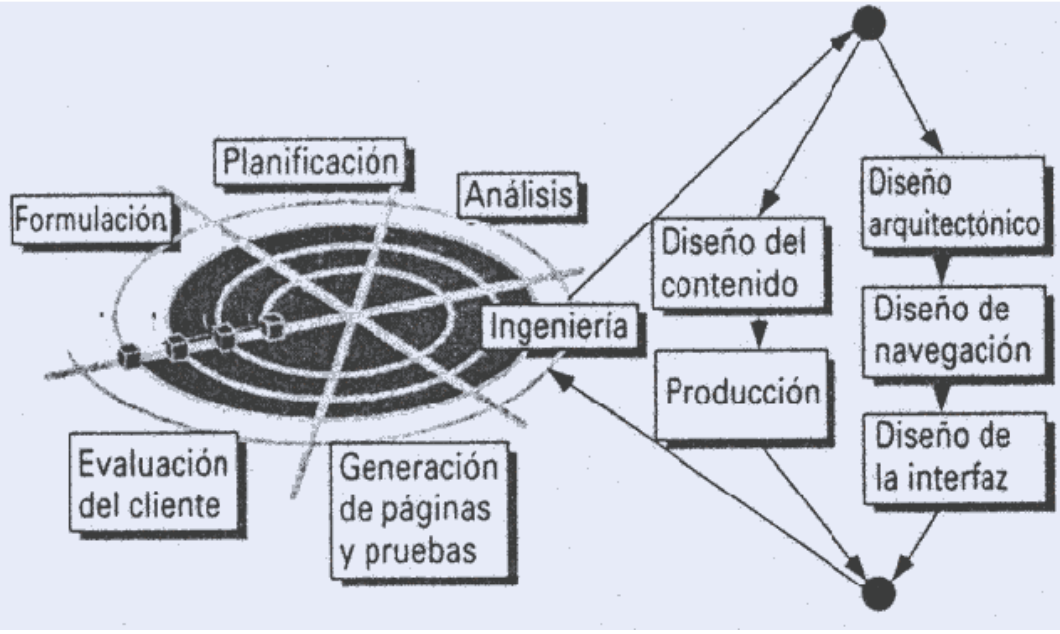


Figura 4: El modelo del proceso IWeb

Fuente: *Pressman, 525*

3.1.2.1 Formulación

Para hacer una correcta formulación, debemos preguntarnos, entre otras cosas:

- Por qué y para que hacer la WebApp?
- Por qué es necesaria?
- Quién la va a usar?

Las respuestas serán muy generales, y no entraran en detalles. Podemos clasificar las metas específicas en:

- **Metas Informativas:** Definen los objetivos sobre el contenido e información que se dará al usuario.
- **Metas Aplicables**: Son los servicios o tareas que puede realizar la WebApp.

Después de las metas, haremos el Perfil del Usuario, determinando las principales características de los potenciales navegadores y clientes.

Más adelante, se hace la Afirmación del Ámbito, con la que vemos la posible integración con sistemas ya existentes, como pueden ser bases de datos.

3.1.2.2 Planificación

Estimación del coste global del proyecto, riesgos, etapas y subetapas.

3.1.2.3 Análisis

Identifica los datos y requisitos funcionales y de comportamiento para la WebApp. Durante la IWeb, se realizan 4 tipos de análisis:

- **Análisis del contenido:** Se puede utilizar el modelado de datos, y en esta etapa se identifica todo el contenido que se va a proporcionar. (texto, gráficos, imágenes, video y sonido)
- **Análisis de la interacción:** Se realizan casos prácticos y sus casos de uso para la descripción detallada de la interacción usuario-WebApp.
- **Análisis funcional:** Se detallan las funciones y operaciones de procesamiento adicionales que se aplicaran en el contenido de la WebApp
- **Análisis de la configuración:** Se detalla y describe el lugar donde va a residir la App. (Intranet, Internet o Extranet). También se tiene que identificar la infraestructura de los componentes y el grado de utilización de la base de datos para generar el contenido.

En todo caso es recomendable hacer un documento que recoja la información de todo el proceso de análisis y que será revisado y modificado para hacer otro documento que pasarle a los diseñadores de la WebApp. En el caso de una App grande no es recomendable hacer un documento muy extenso, porque los requisitos estarán cambiando continuamente, y quedaría obsoleto antes de terminarlo.

3.1.2.4 Ingeniería

Dos tareas paralelas:

- Diseño del contenido y producción: echas por personal NO técnico. Recopilación de información, medios audiovisuales, a integrar en la App.
- Diseño arquitectónico, de navegación y del interfaz: hecho por técnicos

3.1.2.5 Diseño

La característica de inmediatez obliga a que los diseños se hagan rápidamente y a que sean evolucionables. Muchas veces la rapidez o precipitación en el diseño nos cierra puertas a la evolución de la aplicación.

Los ingenieros Web, trabajan bajo los siguientes elementos técnicos:

- ***Principios y métodos de diseño***: Facilitaran la adaptación, pruebas, mejoras y uso.
    - Modularidad eficaz (cohesión alta y acoplamiento bajo)
    - Elaboración paso a paso
    - Diseño orientado a objetos y diagramas UML
- ***Reglas de oro:*** que se han ido construyendo desde los inicios de Internet
- ***Configuraciones de diseño***: Aplicables a los elementos funcionales y a los documentos, gráficos y estética general.
- ***Plantillas:*** Dotan de una estructura similar cada elemento, configuración de diseño, o documento a utilizar dentro de la WebApp. Se hace posible pasando como parámetros a esa plantilla, los datos relevantes, que darán cuerpo al esquema.

3.1.2.5.1 Diseño Arquitectónico

Se encarga de la definición de la estructura global hipermedia y en la aplicación de las configuraciones de diseño y plantillas. Dicha estructura depende de las metas establecidas, del contenido y de la filosofía de navegación. Típicamente hay:

- **Estructuras lineales:** cuando es predecible la sucesión de interacciones. Por ejemplo en la entrada, y validación de datos, hay una estructura lineal. También existen lineales con flujo opcional, y lineal con desviaciones.
- **Estructuras reticulares:** Solo si el contenido de la Web puede ser organizado en dos o más dimensiones. Para ellos el contenido debe ser

muy regular. Por ejemplo, marcas de electrodomésticos y tipos de electrodomésticos.

- **Estructuras jerárquicas:** Son las más comunes. En las jerarquías de software tradicionales se fomentan el flujo de control solo a lo largo de las ramas verticales. En una WebApp se pueden enlazar por hipertexto ramas verticales de la misma estructura. Es el “Acoplamiento”.
- **Estructura en red (o de web pura):** Es como la arquitectura “en evolución” de los sistemas OO. Se enlaza todo con todo. Da mucha flexibilidad de navegación, aunque a veces es confusa para el usuario.

Es común combinar varias de las estructuras, dando lugar a estructuras híbridas.

Los **patrones de diseño** pueden aplicarse en el nivel de componente (cuando se requiere la funcionalidad del proceso de datos), jerárquico, y de navegación (que tratan sobre como el usuario podrá moverse por el contenido de la aplicación) Entre estos últimos, están:

- Ciclo: Se devuelve al usuario al nodo de contenido visitado anteriormente.
- Anillo de Web: Se enlazan páginas de un mismo tema.
- Contorno: Cuando varios ciclos inciden en otro
- Contrapunto: durante la narración se añaden comentarios de hipertexto.
- Mundo de espejo: Varias narraciones desde puntos de vista distintos
- Tamiz: Se presentan opciones que el usuario va eligiendo, hasta llegar a un punto que el mismo habrá provocado con sus decisiones.
- Vecindario: Marco de navegación uniforme por todas las páginas web

3.1.2.5.2 Diseño de navegación

Una vez establecida la arquitectura se define la ruta que permitirá acceder al contenido y a los servicios. Se deberá identificar una **semántica** para según qué usuarios definir una **sintaxis** (mecánica) para la navegación.

Se tendrán, habitualmente, varios papeles: visitante, cliente, cliente registrado, cliente privilegiado, administrador, etc. La semántica para cada rol será distinta.

El diseñador crea una USN (Unidad Semántica de Navegación) para cada meta asociada a cada rol de usuario. Cada USN tiene unas “formas de navegación” (WoN) para que cada usuario llegue a cada meta que se proponga.

Entre las opciones de enlaces (texto, iconos, botones, interruptores, metáforas gráficas, etc.) deberemos elegir la que más se adecuen al interfaz de nuestra web.

Desde el punto de vista de los buscadores hoy por hoy es mejor un enlace texto con la palabra con la que nos gustaría dotar de importancia a la página web enlazada que cualquier otra cosa.

Sin embargo, desde nuestra visión de diseño, los botones, imágenes e iconos que usemos deberán tener un aspecto clickable. Los enlaces de texto deberán tener un color característico, diferenciador del resto del documento.

También se harán necesarias ayudas a la navegación por el sitio: una vista de esquema, un mapa web, tabla de contenidos, mecanismos de búsqueda y servicios dinámicos de ayuda.

3.1.2.5.3 Diseño de la interfaz

Además de las consideraciones de diseño de interfaces de cualquier otro software, en WebApps es necesario considerar nuevos factores, todos ellos, bastante subjetivos.

Algunas sugerencias muy generalizadas son:

- Los errores de servidor deben ser mínimos. El usuario tiene poca paciencia, y generalmente muchos otros recursos en la Web.

- No se debe obligar a hacer leer grandes cantidades de texto, sobre todo si estamos en alguna de las secciones de Ayuda de nuestra App.
- Evitar poner “En construcción”. Crea expectativas decepcionantes.
- Evitar el scroll. Un usuario poco experto “no sabe que existe el scroll”. Todo lo que se le pueda dar en un “pantallazo” será mejor entendido por la mayoría.
- Los menús de navegación estarán disponibles en todas las páginas. Las funciones de navegación no deberán depender del navegador que se esté usando.
- La estética nunca deberá sustituir la funcionalidad.
- Las opciones de navegación y el resto de funcionalidades deberán ser obvias.

3.1.2.6 Generación de páginas

Se adecua al diseño arquitectónico, de navegación y de interfaz, el contenido provisto para sacar las páginas HTML, XML, etc. Es en esta fase donde se integra la WebApp con el software intermedio (CORBA, DCOM, JavaBeans).

3.1.2.7 Pruebas

Son el proceso de ejercitar el software con el fin de encontrar y corregir los errores. En las WebApps, es un reto, debido a la variedad de navegadores, sistemas operativos, plataformas hardware y protocolos de comunicación.

Las estrategias y tácticas a seguir son:

1. El modelo de contenido es revisado para descubrir errores: similar a un corrector ortográfico.
2. El modelo de diseño es revisado para descubrir errores de navegación: Se revisan los posibles errores 404 de navegación, y vemos si cada enlace lleva a la correspondiente USN de la meta del rol de usuario a la que pertenece.

3. Se aplican pruebas de unidad a los componentes de proceso seleccionado y las páginas Web: en muchos casos la unidad comprobable más pequeña es la propia página web. Muchas veces no es posible o practico comprobar elementos más pequeños como formularios, objetos, mapas de imágenes, etc.

4. Se construye la arquitectura y se realizan las pruebas de integración: La estrategia para la prueba de integración depende de la arquitectura que se haya elegido. En estructuras jerárquicas lineales, reticulares o sencillas, es muy similar a como se integran los módulos del software convencional. En jerarquías mezcladas o arquitecturas de red, es similar a los sistemas OO.

5. La WebApp ensamblada se prueba para conseguir un a funcionalidad global y un contenido: Se hace una prueba de acciones visibles y de salidas reconocibles para el usuario.

6. Se implementa la WebApp en una variedad de configuraciones diferentes de entornos y comprobar así la compatibilidad con cada configuración: Se lleva hace una matriz de referencias cruzadas con sistemas operativos, plataformas de hardware, navegadores y protocolos de comunicación. Se hacen pruebas para cubrir los errores asociados con todas y cada una de las configuraciones posibles.

7. La WebApp se comprueba con una población de usuarios finales controlada y monitorizada: Se hacen grupos de usuarios según los posibles roles, se hace un uso intensivo y se evalúan los resultados, para ver errores de contenido y navegación, usabilidad, compatibilidad, fiabilidad y rendimiento.

3.1.2.8 Evaluación del cliente

No es la última fase. Es una fase a ejecutar cada vez que se termina alguna de las anteriores. Los cambios se hacen efectivos por el flujo incremental del proceso.

3.2 METODOLOGÍA: OOHDM (OBJECT ORIENTED HYPERMEDIA DESIGN METHODOLOGY)

3.2.1 INTRODUCCIÓN OOHDM

Object Oriented Hypermedia Design Methodology (OOHDM, Método de Diseño Hipermedia Orientado a Objetos), propuesto por Schwabe y Rossi (1998). OOHDM tiene por objetivo simplificar y a la vez hacer más eficaz el diseño de aplicaciones hipermedia.

Las metodologías tradicionales de ingeniería de software, o las metodologías para sistemas de desarrollo de información, no contienen una buena abstracción capaz de facilitar la tarea de especificar aplicaciones hipermedia. El tamaño, la complejidad y el número de aplicaciones crecen en forma acelerada en la actualidad, por lo cual una metodología de diseño sistemática es necesaria para disminuir la complejidad y admitir evolución y reusabilidad.

Producir aplicaciones en las cuales el usuario pueda aprovechar el potencial del paradigma de la navegación de sitios web, mientras ejecuta transacciones sobre bases de información, es una tarea muy difícil de lograr. En primer lugar, la navegación posee algunos problemas. Una estructura de navegación robusta es una de las claves del éxito en las aplicaciones hipermedia. Si el usuario entiende dónde puede ir y cómo llegar al lugar deseado, es una buena señal de que la aplicación ha sido bien diseñada.

Construir la interfaz de una aplicación web es también una tarea compleja; no sólo se necesita especificar cuáles son los objetos de la interfaz que deberían

ser implementados, sino también la manera en la cual estos objetos interactuarán con el resto de la aplicación.

En hipermedia existen requerimientos que deben ser satisfechos en un entorno de desarrollo unificado (framework). Por un lado, la navegación y el comportamiento funcional de la aplicación deberían ser integrados. Por otro lado, durante el proceso de diseño se debería poder desacoplar las decisiones de diseño relacionadas con la estructura navegacional de la aplicación, de aquellas relacionadas con el modelo del dominio.

OOHDM propone el desarrollo de aplicaciones hipermedia a través de un proceso compuesto por cuatro etapas: diseño conceptual, diseño navegacional, diseño de interfaces abstractas e implementación.

3.2.2 FASES DE LA METODOLOGÍA OOHDM

Esta metodología plantea el diseño de una aplicación de este tipo a través de cinco fases que se desarrollan de un modo iterativo.

Estas fases son:

- Determinación de Requerimientos.
- Diseño Conceptual.
- Diseño Navegacional.
- Diseño de Interfaz Abstracta.
- Implementación.

3.2.2.1 Determinación de Requerimientos

Ésta es una de las fases más importantes, debido a que es aquí donde se realiza la recogida de datos, para ello se deben de proporcionar las respuestas a las siguientes interrogantes:

- ¿Cuáles son los tópicos principales que serán atendidos?
- ¿Cómo los tópicos están relacionados entre sí?

- ¿Qué categoría de usuarios serán atendidos?
- ¿Cuáles son las tareas principales que serán abordadas?
- ¿Qué tareas corresponden a qué categoría de usuarios?
- ¿Los recursos disponibles son competitivos con la información levantada?

3.2.2.2 Diseño Conceptual

Durante esta actividad se lleva a cabo, según Koch (2002) un esquema conceptual representado por los objetos del dominio, las relaciones y colaboraciones existentes establecidas entre ellos.

El esquema de las clases consiste en un conjunto de clases conectadas por relaciones. Los objetos son instancias de las clases. Las clases son usadas durante el diseño navegacional para derivar nodos, y las relaciones que son usadas para construir enlaces. Como es de costumbre en modelos orientados a objetos, las clases son descritas por un conjunto de atributos y métodos (implementando el comportamiento de las clases), siendo aún, organizadas en jerarquías (parte-de y es uno/a). En la Figura 5 se puede observar un ejemplo de cómo se representa un diagrama de clases.

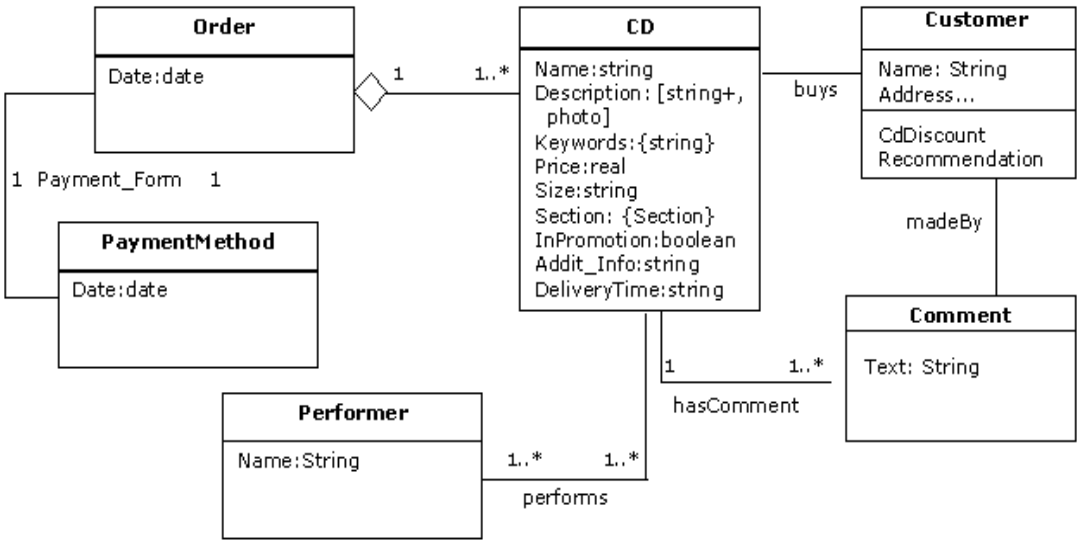


Figura 5: Modelo Conceptual para una Tienda de CD's.

Fuente: *Designing Personalized Web Applications*

http://www10.org/cdrom/papers/395/index.html

3.2.2.3 Diseño Navegacional

La primera generación de aplicaciones web fue pensada para realizar navegación a través del espacio de información, utilizando un simple modelo de datos de hipermedia.
Según Koch (2002) El diseño de navegación es expresado en dos esquemas: el esquema de clases navegacionales y el esquema de contextos navegacionales, los cuales se definen a continuación:

- ***Esquema de Clases Navegacionales***: establece las posibles vistas del hiperdocumento a través de unos tipos predefinidos de clases, llamadas navegacionales como son los nodos, los enlaces y otras clases que representan estructuras o formas alternativas de acceso a los nodos, como los índices y los recorridos guiados (Koch, ob. cit), ver Figura 6.

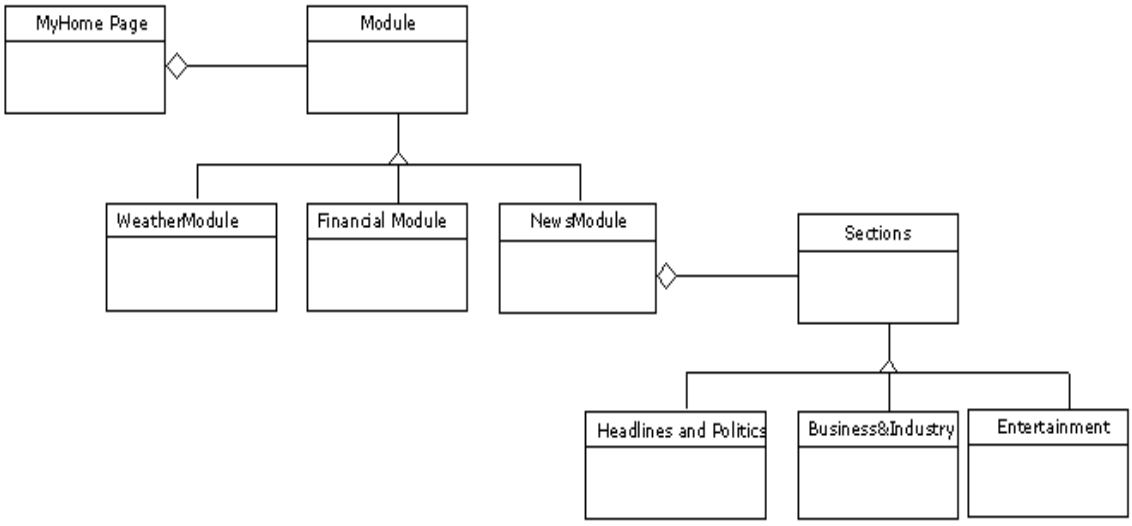


Figura 6: Esquema básico navegacional para my.yahoo.com.
Fuente: *Designing Personalized Web Applications*
http://www10.org/cdrom/papers/395/index.html

- ***Esquema de Contexto Navegacional***: es el que permite la estructuración del hiperespacio de navegación en sub-espacios para los que se indica la información que será mostrada al usuario y los enlaces que estarán disponibles cuando se accede a un objeto (nodo) en un contexto determinado (Koch, ob. cit). Sánchez (s.f.) comenta con respecto a esta

fase, “es la fase en que diseñamos la aplicación teniendo en cuenta los usuarios a los que va dirigida y los objetivos de la misma”, en pocas palabras, es la fase en que se plantea la manera de cómo será la navegación del usuario en el hiperdocumento.
Las tareas que se ejecutan son las siguientes:

- Se reorganiza la información representada en el modelo conceptual.
- Se estructura la vista de navegación sobre el modelo conceptual.

Una innovación de OOHDM es que los objetos sobre los cuales navega el usuario no son objetos conceptuales, sino otro tipo de objetos que se construyen a partir de uno o más objetos conceptuales, lo cual implica a su vez que el usuario navegue a través de enlaces, muchos de los cuales no se pueden derivarse directamente en relaciones conceptuales.
Este modelo implementa un conjunto de datos predefinidos, los cuales se describen a continuación:

- **Nodos:** son contenedores de información, éstos se definen como vistas orientadas a objetos de las clases conceptuales. Los nodos se pueden definir combinando atributos de clases relacionadas en el esquema conceptual (Sánchez, ob. cit).
- **Enlaces:** son los que identifican las relaciones implementadas en el esquema conceptual. Las clases de los enlaces especifican sus atributos, comportamiento y los objetos fuentes del mismo. Estos representan las posibles formas de comenzar la navegación (Sánchez, ob. cit).
- **Estructuras de Acceso:** Las estructuras de acceso actúan como índices o diccionarios que permiten al usuario encontrar de forma rápida y eficiente la información deseada. Los menús, los índices o las guías de ruta son ejemplos de estas estructuras. Las estructuras de acceso también se modelan como clases, compuestas por un conjunto de referencias a objetos que son accesibles desde ella y una serie de criterios de clasificación de las mismas.

- **Contexto Navegacional:** Para diseñar bien una aplicación hipermedia, hay que prever los caminos que el usuario puede seguir, así es como únicamente se podrá evitar información redundante o que el usuario se pierda en la navegación. En OOHDM un contexto navegacional está compuesto por un conjunto de nodos, de enlaces de clases de contexto y de otros contextos navegacionales. Estos son introducidos desde clases de navegación (enlaces, nodos o estructuras de acceso), pudiendo ser definidas por extensión o de forma implícita.
- **Clase de Contexto:** Es otra clase especial que sirve para complementar la definición de una clase de navegación. Por ejemplo, sirve para indicar qué información está accesible desde un enlace y desde dónde se puede llegar a él.

3.2.2.4 Diseño de Interfaz Abstracta

Una vez que las estructuras navegacionales son definidas, se deben especificar los aspectos de interfaz. Según Schwabe, Rossi y Simone (s.f.) esto significa definir la forma en la cual los objetos navegacionales pueden aparecer, cómo los objetos de interfaz activarán la navegación y el resto de la funcionalidad de la aplicación.

El modelo de interfaz ADVs (Vista de Datos Abstractos), especifica la organización y comportamiento de la interfaz, pero la apariencia física real o de los atributos, y la disposición de las propiedades de las ADVs (Vista de Datos Abstractos), en la pantalla real son hechas en la fase de implementación (Schwabe y otros, ob. cit.).

En la Figura 7 se puede observar un ejemplo de diagramas de configuración, y en la Figura 8 se pueden apreciar el diagrama de eventos que ocurre sobre el mismo.

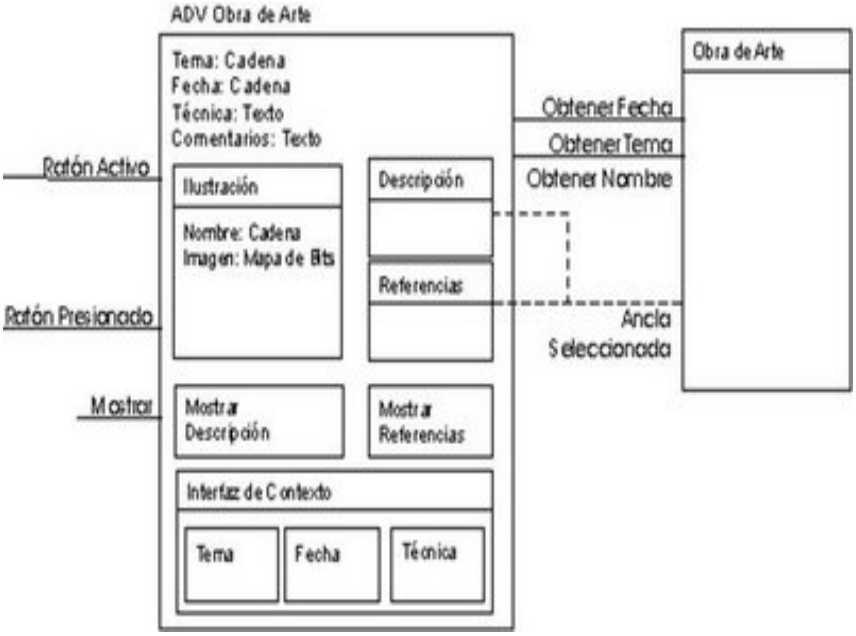


Figura 7: Diagrama de Configuración para los nodos ADV.

Fuente: *OOHDM's Design Process* *http://www-di.inf.puc-rio.br/schwabe/HT96WWW/section3.html*

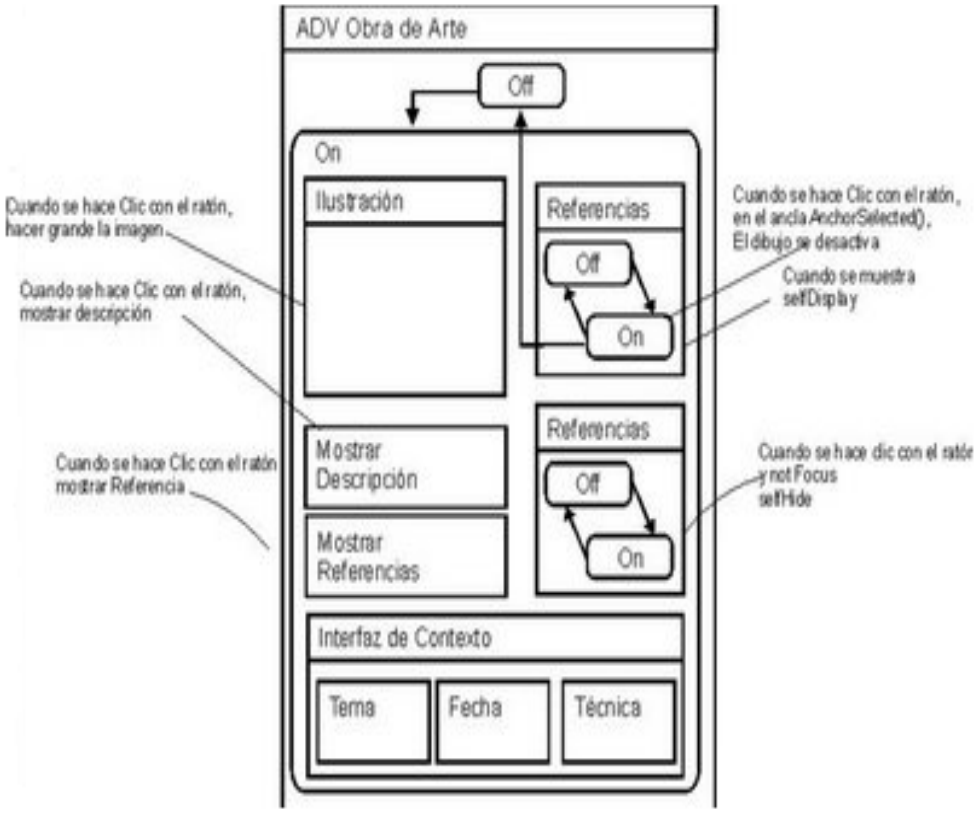


Figura 8: Diagrama de Configuración para los nodos ADV.

Fuente: *OOHDM's Design Process* http://www-di.inf.puc-rio.br/schwabe/HT96WWW/section3.html

3.2.2.5 Implementación

En esta fase, el diseñador debe implementar el diseño. Según Schwabe y Rossi. (1998), hasta ahora todos los modelos fueron construidos en forma independiente de la plataforma de implementación; en esta fase es tenido en cuenta el entorno particular en el cual se va a correr la aplicación.

Al llegar a esta fase, el primer paso que debe realizar el diseñador es definir los ítems de información que son parte del dominio del problema. Debe identificar también, cómo son organizados los ítems de acuerdo con el perfil del usuario y su tarea; decidir qué interfaz debería ver y cómo debería comportarse.

A fin de implementar todo en un entorno web, el diseñador debe decidir además qué información debe ser almacenada. Es de especial importancia el hacer notar que hoy en día, hay muchos y varios ambientes de implementación, con características distintas. Es claro, por ejemplo, que no se puede usar el mismo conjunto de líneas de acción en la traducción de un proyecto OOHDM para un documento HTML que para un programa en Macromedia Flash.

3.2.3 VENTAJAS DE OOHDM

De acuerdo con Silva y Mercerat (2001) OOHDM como metodología de desarrollo de aplicaciones de hipermedia, proporciona ventajas como:

- La recuperación de la información puede realizarse sin problemas.
- Se pueden crear enlaces entre nodos cualesquiera.
- La modularidad y la consistencia se potencian.
- Marco idóneo para la autoría en colaboración.
- Soporte a diferentes modos de acceso a la información.

En la actualidad, el desarrollo de software empleando patrones de proyecto, se encuentra en crecimiento según Gamma, Helm, Johnson y Vlissides (1995), sin embargo, su potencial se encuentra inexplorado en el campo de hipermedios,

especialmente a la hora de describir las arquitecturas para la navegación e interface en aplicaciones de hipermedia.

OOHDM según Schwabe y Rossi (1998) se torna diferente y superior a otras metodologías de desarrollo de aplicaciones de hipermedia al ofrecer la ventaja de patrones de proyecto poderosos como primitivas para la construcción del modelo navegacional de una aplicación hipermedia.

Puede decirse con base en todas las ventajas antes mencionadas que OOHDM toma en cuenta las crecientes necesidades de analistas y programadores de aplicaciones hipermedia, y se presenta como una técnica ideal de desarrollo para la producción de aplicaciones evolutivas de alta calidad.

3.3 UML (UNIFIED MODELING LANGUAGE O LENGUAJE UNIFICADO DE MODELAMIENTO)

3.3.1 INTRODUCCIÓN UML

Es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Se usa para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas a construir.

UML capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Un sistema se modela como una colección de objetos discretos que interactúan para realizar un trabajo que finalmente beneficia a un usuario externo.

El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar.

UML no es un lenguaje de programación. Las herramientas pueden ofrecer generadores de código de UML para una gran variedad de lenguaje de

programación, así como construir modelos por ingeniería inversa a partir de programas existentes.

Es un lenguaje de propósito general para el modelado orientado a objetos. UML es también un lenguaje de modelamiento visual que permite una abstracción del sistema y sus componentes.

3.3.2 OBJETIVOS DEL UML

- UML es un lenguaje de modelado de propósito general que pueden usar todos los modeladores. No tiene propietario y está basado en el común acuerdo de gran parte de la comunidad informática.

- UML no pretende ser un método de desarrollo completo. No incluye un proceso de desarrollo paso a paso. UML incluye todos los conceptos que se consideran necesarios para utilizar un proceso moderno iterativo, basado en construir una sólida arquitectura para resolver requisitos dirigidos por casos de uso.

- Ser tan simple como sea posible pero manteniendo la capacidad de modelar toda la gama de sistemas que se necesita construir. UML necesita ser lo suficientemente expresivo para manejar todos los conceptos que se originan en un sistema moderno, tales como la concurrencia y distribución, así como también los mecanismos de la ingeniería de software, como son la encapsulación y componentes.

- Debe ser un lenguaje universal, como cualquier lenguaje de propósito general.

3.3.3 ARQUITECTURA DEL UML

Arquitectura de cuatro capas, definida a fin de cumplir con la especificación Meta Object Facility del OMG:

- *Meta-metamodelo*: define el lenguaje para especificar metamodelos.
- *Metamodelo*: define el lenguaje para especificar modelos.
- *Modelo*: define el lenguaje para describir un dominio de información.

UML está compuesto por los siguientes diagramas:

Área	**Vista**	**Diagramas**	**Conceptos Principales**
Estructural	Vista Estática	Diagrama de Clases	Clase, asociación, generalización, dependencia, realización, interfaz.
	Vista de Casos de Uso	Diagramas de Casos de Uso	Caso de Uso, Actor, asociación, extensión, generalización.
	Vista de Implementación	Diagramas de Componentes	Componente, interfaz, dependencia, realización.
	Vista de Despliegue	Diagramas de Despliegue	Nodo, componente, dependencia, localización.
Dinámica	Vista de Estados de máquina	Diagramas de Estados	Estado, evento, transición, acción.
	Vista de actividad	Diagramas de Actividad	Estado, actividad, transición, determinación, división, unión.
	Vista de interacción	Diagramas de Secuencia	Interacción, objeto, mensaje, activación.
		Diagramas de Colaboración	Colaboración, interacción, rol de colaboración, mensaje.
Administración o Gestión de modelo	Vista de Gestión de modelo	Diagramas de Clases	Paquete, subsistema, modelo.

Extensión de UML	Todas	Todos	Restricción, estereotipo, valores, etiquetados.

3.3.3.1 Diagramas de Clases

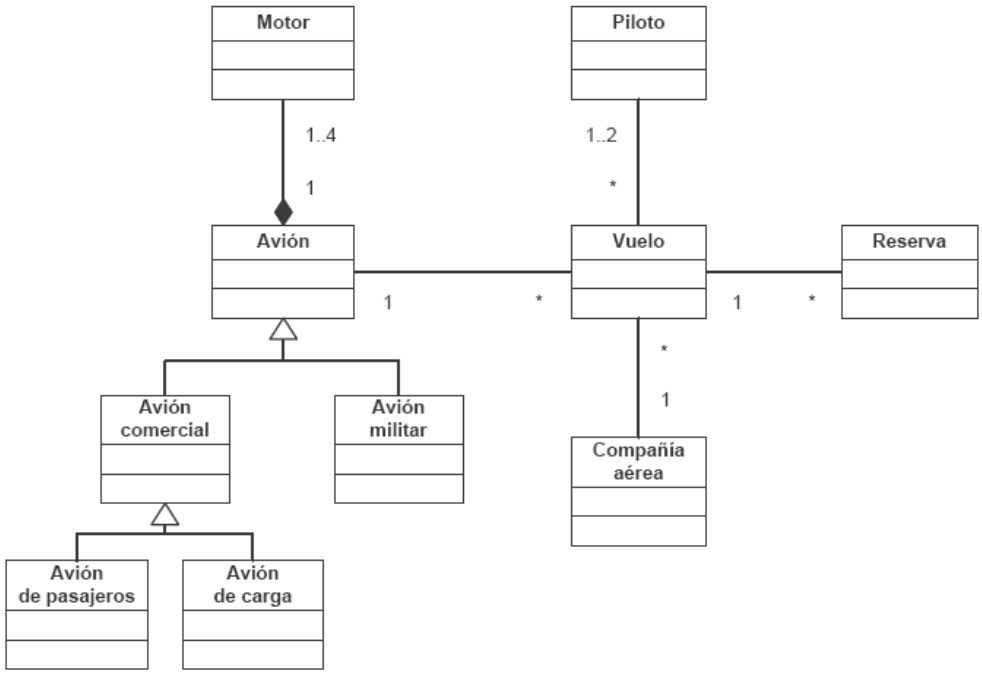


Figura 9: Diagrama de Clases.
Fuente: UML *http://www.creangel.com/uml/diagramas.*

Un diagrama de clases es una descripción visual de los posibles sistemas. Un diagrama de clases y un diagrama de objetos son las alternativas de representación de modelos de objetos, aunque los diagramas de clases prevalezcan más que los de objetos. Normalmente se puede construir un diagrama de clases y ocasionalmente uno de objetos para ilustrar las estructuras de datos más complejas.

Un diagrama de clases presenta las clases del sistema con sus relaciones estructurales y de herencia. La definición de clase incluye definiciones para atributos y operaciones. El modelo de casos de uso aporta información para

establecer las clases, objetos, atributos y operaciones. El mundo real puede ser visto desde abstracciones diferentes (subjetividad).

Un diagrama de clases contiene íconos que representan las clases. Se pueden crear una o más diagramas que representan el nivel más alto de abstracción en el modelo e ir representando cada nivel con diagramas separados.

Mecanismos de abstracción:

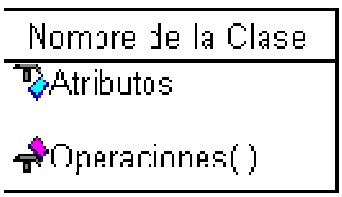
- Clasificación / Instanciación
- Composición / Descomposición
- Agrupación / Individualización
- Especialización / Generalización

La ***clasificación*** es uno de los mecanismos de abstracción más utilizados. La clase define el ámbito de definición de un conjunto de objetos, y cada objeto pertenece a una clase, Los objetos se crean por instanciación de las clases.

Una clase captura la estructura y comportamiento común de un conjunto de objetos. Una clase es una abstracción de ítemes del mundo real.

Una clase es una ícono que se representa como una caja, en OMT, la que se divide en tres partes, con el nombre de la clase en la parte superior, la lista de sus atributos en la segunda y la lista de sus operaciones o métodos en la última.

Cada clase se representa en un rectángulo con tres compartimientos:



Los atributos de una clase no deberían ser manipulables directamente por el resto de objetos. Por esta razón se crearon niveles de visibilidad para los elementos que son:

- (-) Privado : es el más fuerte. Esta parte es totalmente invisible (excepto para clases friends en terminología C++)
- (#) Los atributos/operaciones protegidos están visibles para las clases friends y para las clases derivadas de la original.
- (+) Los atributos/operaciones públicos son visibles a otras clases (cuando se trata de atributos se está transgrediendo el principio de encapsulación)

Relaciones entre clases:

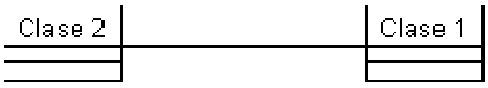
Los enlaces entre objetos pueden representarse entre las respectivas clases y sus formas de relación son:

- Asociación y Agregación (vista como un caso particular de asociación)
- Generalización/Especialización.

Las relaciones de Agregación y Generalización forman jerarquías de clases.

Asociación:

La asociación expresa una conexión bidireccional entre objetos. Una asociación es una abstracción de la relación existente en los enlaces entre los objetos. Puede determinarse por la especificación de multiplicidad (mínima...máxima)



- Uno y sólo uno
- 0..1 Cero o uno
- M..N Desde M hasta N (enteros naturales)
- * Cero o muchos
- 0..* Cero o muchos
- 1..* Uno o muchos (al menos uno)

Agregación:

La agregación representa una relación parte de entre objetos. En UML se proporciona una escasa caracterización de la agregación. Esta relación puede ser caracterizada con precisión determinando las relaciones de comportamiento y estructura que existen entre el objeto agregado y cada uno de sus objetos componentes. Muestra que el objeto agregado está físicamente construido a partir de otro objeto, o que lógicamente lo contiene.

Una agregación se podría caracterizar según:

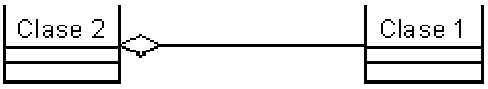
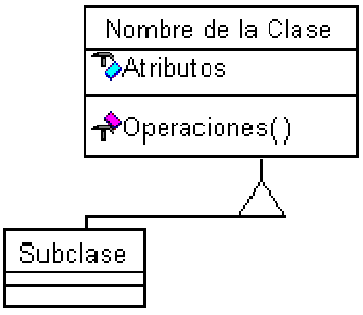


Diagrama de Clases y Diagramas de Objetos pertenecen a dos vistas complementarias del modelo. Un Diagrama de Clases muestra la abstracción de una parte del dominio. Un Diagrama de Objetos representa una situación concreta del dominio. Las clases abstractas no son instanciadas.

Generalización:

La Generalización consiste en factorizar las propiedades comunes de un conjunto de clases en una clase más general.



Los nombres usados: clase padre - clase hija. Otros nombres: superclase - subclase, clase base - clase derivada. Las subclases heredan propiedades de sus clases padre, es decir, atributos y operaciones (y asociaciones) de la clase padre están disponibles en sus clases hijas.

La *Generalización y Especialización* son equivalentes en cuanto al resultado: la *jerarquía y herencia* establecidas. Generalización y Especialización no son operaciones reflexivas ni simétricas pero sí transitivas. La especialización es una técnica muy eficaz para la extensión y reutilización. La noción de clase está próxima a la de conjunto. Dada una clase, podemos ver el conjunto relativo a las instancias que posee o bien relativo a las propiedades de la clase. Generalización y especialización expresan relaciones de inclusión entre conjuntos.

3.3.3.2 **Diagramas de Caso de Uso**

Casos de Uso es una técnica para capturar información de cómo un sistema o negocio trabaja, o de cómo se desea que trabaje. No pertenece estrictamente al enfoque orientado a objeto, es una técnica para captura de requisitos.

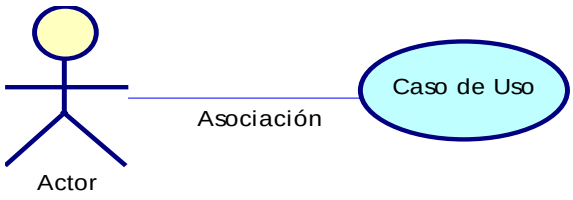


Figura 10: Diagrama de Casos de Uso.
Fuente: UML *http://www.creangel.com/uml/diagramas.*

- Los Casos de Uso (Ivar Jacobson) describen bajo la forma de acciones y reacciones el comportamiento de un sistema desde el p.d.v. del usuario.
- Permiten definir los límites del sistema y las relaciones entre el sistema y el entorno.
- Los Casos de Uso son descripciones de la funcionalidad del sistema independientes de la implementación.
- Comparación con respecto a los Diagramas de Flujo de Datos del Enfoque Estructurado.

- Los Casos de Uso cubren la carencia existente en métodos previos (OMT, Booch) en cuanto a la determinación de requisitos.
- Los Casos de Uso particionan el conjunto de necesidades atendiendo a la categoría de usuarios que participan en el mismo.
- Están basados en el lenguaje natural, es decir, es accesible por los usuarios.

Actores

- ***Principales:*** personas que usan el sistema.
- ***Secundarios:*** personas que mantienen o administran el sistema.
- ***Material externo:*** dispositivos materiales imprescindibles que forman parte del ámbito de la aplicación y deben ser utilizados.
- ***Otros sistemas:*** sistemas con los que el sistema interactúa.

Los Casos de Uso se determinan observando y precisando, actor por actor, las secuencias de interacción, los escenarios, desde el punto de vista del usuario. Los casos de uso intervienen durante todo el ciclo de vida. El proceso de desarrollo estará dirigido por los casos de uso. Un escenario es una instancia de un caso de uso.

UML define cuatro tipos de relación en los Diagramas de Casos de Uso:

- ***Comunicación***
- ***Inclusión:*** una instancia del Caso de Uso origen incluye también el comportamiento descrito por el Caso de Uso destino. «include» reemplazó al denominado «uses»
- ***Extensión:*** el Caso de Uso origen extiende el comportamiento del Caso de Uso destino. «extend»
- ***Herencia:*** el Caso de Uso origen hereda la especificación del Caso de Uso destino y posiblemente la modifica y/o amplía.

3.3.3.3 Diagramas de Componentes

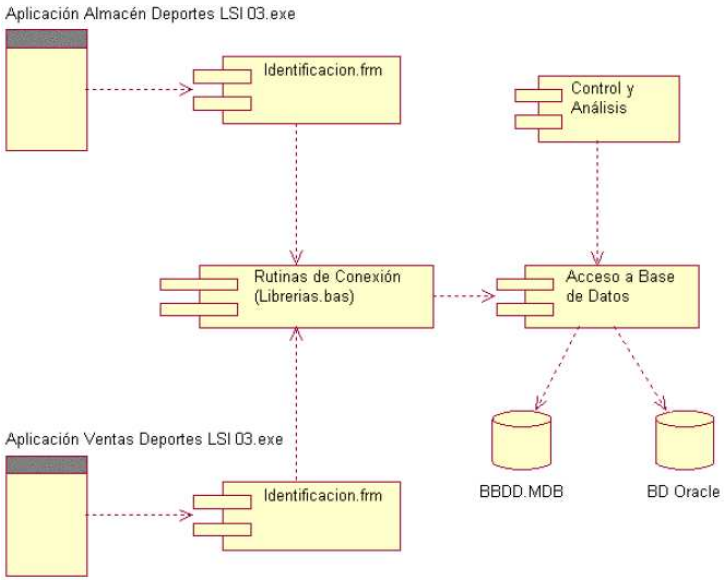


Figura 11: Diagrama de Componentes.
Fuente: UML *http://www.creangel.com/uml/diagramas.*

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Los componentes representan todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas. Pueden ser simples archivos, paquetes de Ada, bibliotecas cargadas dinámicamente, etc. Las relaciones de dependencia se utilizan en los diagramas de componentes para indicar que un componente utiliza los servicios ofrecidos por otro componente.

Un ***componente*** de sólo compilación es aquel que es significativo únicamente en tiempo de compilación. Un componente ejecutable es un programa ejecutable.
Un diagrama de componentes tiene sólo una versión con descriptores, no tiene versión con instancias. Para mostrar las instancias de los componentes se debe usar un diagrama de despliegue.

Un diagrama de componentes muestra clasificadores de componentes, las clases definidas en ellos, y las relaciones entre ellas. Los clasificadores de componentes también se pueden anidar dentro de otros clasificadores de componentes para mostrar relaciones de definición.
Un diagrama que contiene clasificadores de componentes y de nodo se puede utilizar para mostrar las dependencias del compilador, que se representa como flechas con líneas discontinuas (dependencias) de un componente cliente a un componente proveedor del que depende.

Los tipos de dependencias son específicos del lenguaje y se pueden representar como estereotipos de las dependencias. El diagrama también puede usarse para mostrar interfaces y las dependencias de llamada entre componentes, usando flechas con líneas discontinuas desde los componentes a las interfaces de otros componentes.

El diagrama de componente hace parte de la vista física de un sistema, la cual modela la estructura de implementación de la aplicación por sí misma, su organización en componentes y su despliegue en nodos de ejecución. Esta vista proporciona la oportunidad de establecer correspondencias entre las clases y los componentes de implementación y nodos. La vista de implementación se representa con los diagramas de componentes.

Componente

Es una parte física reemplazable de un sistema que empaqueta su implementación y es conforme a un conjunto de interfaces a las que proporciona su realización.

Algunos componentes tienen identidad y pueden poseer entidades físicas, que incluyen objetos en tiempo de ejecución, documentos, bases de datos, etc. Los componentes existentes en el dominio de la implementación son unidades físicas en los computadores que se pueden conectar con otros componentes, sustituir, trasladar, archivar, etc.

Los componentes tienen dos características: Empaquetan el código que implementa la funcionalidad de un sistema, y algunas de sus propias instancias de objetos que constituyen el estado del sistema. Los llamados últimos componentes de la identidad, porque sus instancias poseen identidad y estado.

Código

Un componente contiene el código para las clases de implementación y otros elementos. Un componente de código fuente es un paquete para el código fuente de las clases de implementación. Algunos lenguajes de programación distinguen archivos de declaración de los archivos de método, pero todos son componentes.

Un componente de código binario es un paquete para el código compilado. Una biblioteca del código binario es un componente. Cada tipo de componente contiene el código para las clases de implementación que realizan algunas clases e interfaces lógicas.

La relación de realización asocia un componente con las clases y las interfaces lógicas que implementan sus clases de implementación.

Las interfaces de un componente describen la funcionalidad que aporta. Cada operación de la interfaz debe hacer referencia eventualmente a un elemento de la implementación disponible en el componente

La **estructura estática**, ejecutable de una implementación de un sistema se puede representar como un conjunto interconectado de componentes.
Las dependencias entre componentes significan que los elementos de la implementación en un componente requieren los servicios de los elementos de implementación en otros componentes. Tal uso requiere que dichos elementos sean de visibilidad pública.

Identidad

Un componente de identidad tiene identidad y estado. Posee los objetos físicos que están situados en él. Puede tener atributos, relaciones de composición con los objetos poseídos, y asociaciones con otros componentes. Desde este punto de vista es una clase. Sin embargo la totalidad de su estado debe hacer referencia a las instancias que contiene.

Estructura

Un componente ofrece un conjunto de elementos de implementación, esto significa que el componente proporciona el código para los elementos. Un componente puede tener operaciones e interfaces. Un componente de identidad es un contenedor físico para las entidades físicas como bases de datos. Para proporcionar manejadores para sus elementos contenidos, puede tener atributos y asociaciones salientes, que deben ser implementadas por sus elementos de implementación. Este componente se representa con un rectángulo con dos rectángulos más pequeños que sobresalen en su lado izquierdo.

Las operaciones e interfaces disponibles para los objetos exteriores se pueden representar directamente en el símbolo de clase. Estos son su comportamiento como clase. Los contenidos del subsistema se representan en un diagrama separado. Las dependencias de un componente con otros componentes o elementos del modelo se representan usando líneas discontinuas con la punta de flecha hacia los elementos del proveedor. Sí un componente es la realización de una interfaz, se representa con un círculo unido al símbolo del componente por un segmento de línea.

3.3.3.4 Diagramas de Despliegue

Los Diagramas de Despliegue muestran la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos.

La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instacias de nodos conectados por enlaces de comunicación. Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria.

Los estereotipos permiten precisar la naturaleza del equipo:

- Dispositivos
- Procesadores
- Memoria

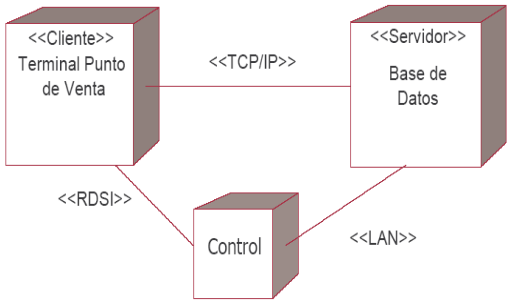


Figura 12: Diagrama de Despliegue.
Fuente: UML *http://www.creangel.com/uml/diagramas.*

Los nodos se interconectan mediante soportes bidireccionales que pueden a su vez estereotiparse. Esta vista permite determinar las consecuencias de la distribución y la asignación de recursos.

Las instancias de los nodos pueden contener instancias de ejecución, como instancias de componentes y objetos.
El modelo puede mostrar dependencias entre las instancias y sus interfaces, y también modelar la migración de entidades entre nodos u otros contenedores. Esta vista tiene una forma de descriptor y otra de instancia.

La forma de instancia muestra la localización de las instancias de los componentes específicos en instancias específicas del nodo como parte de una configuración del sistema.

La forma de descriptor muestra qué tipo de componentes pueden subsistir en qué tipos de nodos y qué tipo de nodos se pueden conectar, de forma similar a un diagrama de clases, esta forma es menos común que la primera.

3.3.3.5 Diagramas de Estado

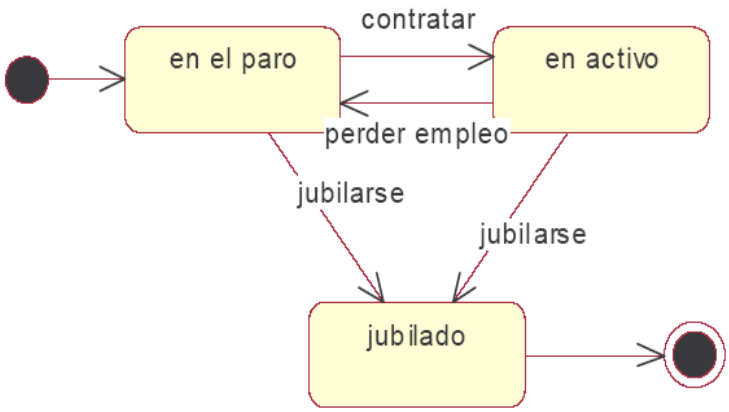


Figura 13: Diagrama de Estado.
Fuente: UML *http://www.creangel.com/uml/diagramas.*

Los Diagramas de Estado representan autómatas de estados finitos, desde el p.d.v. de los estados y las transiciones. Son útiles sólo para los objetos con un comportamiento significativo. Cada objeto está en un estado en cierto instante.

El estado está caracterizado parcialmente por los valores algunos de los atributos del objeto.

El estado en el que se encuentra un objeto determina su comportamiento. Cada objeto sigue el comportamiento descrito en el Diagrama de Estados asociado a su clase.

Los Diagramas de Estados y escenarios son complementarios, los Diagramas de Estados son autómatas jerárquicos que permiten expresar concurrencia, sincronización y jerarquías de objetos, son grafos dirigidos y deterministas. La transición entre estados es instantánea y se debe a la ocurrencia de un evento.

Estado

Identifica un periodo de tiempo del objeto (no instantáneo) en el cual el objeto está esperando alguna operación, tiene cierto estado característico o puede recibir cierto tipo de estímulos. Se representa mediante un rectángulo con los bordes redondeados, que puede tener tres compartimientos: uno para el nombre, otro para el valor característico de los atributos del objeto en ese estado y otro para las acciones que se realizan al entrar, salir o estar en un estado (entry, exit o do, respectivamente).

Eventos

Es una ocurrencia que puede causar la transición de un estado a otro de un objeto. Esta ocurrencia puede ser una de varias cosas:

- Condición que toma el valor de verdadero o falso
- Recepción de una señal de otro objeto en el modelo
- Recepción de un mensaje
- Paso de cierto período de tiempo, después de entrar al estado o de cierta hora y fecha particular

El nombre de un evento tiene alcance dentro del paquete en el cual está definido, no es local a la clase que lo nombre.

Envío de mensajes

Además de mostrar y transición de estados por medio de eventos, puede representarse el momento en el cual se envían mensajes a otros objetos. Esto

se realiza mediante una línea punteada dirigida al diagrama de estados del objeto receptor del mensaje.

Transición simple

Una transición simple es una relación entre dos estados que indica que un objeto en el primer estado puede entrar al segundo estado y ejecutar ciertas operaciones, cuando un evento ocurre y si ciertas condiciones son satisfechas. Se representa como una línea sólida entre dos estados, que puede venir acompañada de un texto con el siguiente formato:

event-signature "[" guard-condition] "/" action-expression "^"send-clause

event-signature es la descripción del evento que da lugar la transición, guard-condition son las condiciones adicionales al evento necesarias para que la transición ocurra, **action-expression** es un mensaje al objeto o a otro objeto que se ejecuta como resultado de la transición y el cambio de estado y send-clause son acciones adicionales que se ejecutan con el cambio de estado, por ejemplo, el envío de eventos a otros paquetes o clases.

Transición interna

Es una transición que permanece en el mismo estado, en vez de involucrar dos estados distintos. Representa un evento que no causa cambio de estado. Se denota como una cadena adicional en el compartimiento de acciones del estado.

Acciones

Podemos especificar la solicitud de un servicio a otro objeto como consecuencia de la transición. Se puede especificar el ejecutar una acción como consecuencia de entrar, salir, estar en un estado, o por la ocurrencia de un evento.

Subestados

Un estado puede descomponerse en subestados, con transiciones entre ellos y conexiones al nivel superior. Las conexiones se ven al nivel inferior como estados de inicio o fin, los cuales se suponen conectados a las entradas y salidas del nivel inmediatamente superior.

Transacción Compleja

Una transición compleja relaciona tres o más estados en una transición de múltiples fuentes y/o múltiples destinos. Representa la subdivisión en threads del control del objeto o una sincronización. Se representa como una línea vertical de la cual salen o entran varias líneas de transición de estado.

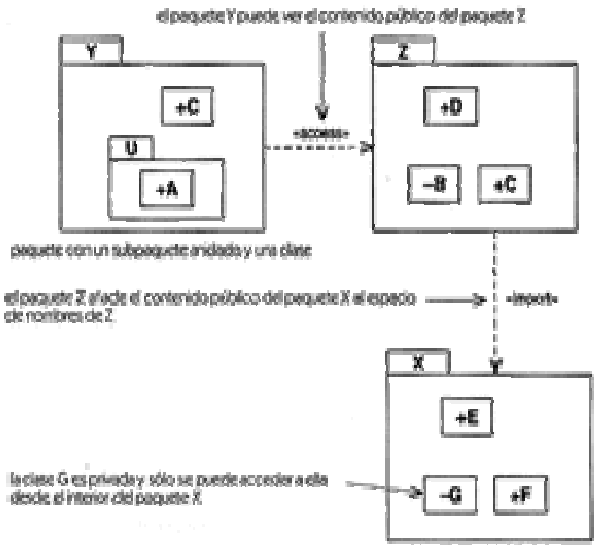
Transición a estados anidados

Una transición de hacia un estado complejo (descrito mediante estados anidados) significa la entrada al estado inicial del subdiagrama. Las transiciones que salen del estado complejo se entienden como transiciones desde cada uno de los subestados hacia afuera (a cualquier nivel de profundidad).

Transiciones temporizadas

- Las esperas son actividades que tienen asociada cierta duración.
- La actividad de espera se interrumpe cuando el evento esperado tiene lugar.
- Este evento desencadena una transición que permite salir del estado que alberga la actividad de espera. El flujo de control se transmite entonces a otro estado.

Paquetes



Cualquier sistema grande se debe dividir en unidades más pequeñas, de modo que las personas puedan trabajar con una cantidad de información limitada, a la vez y de modo que los equipos de trabajo no interfieran con el trabajo de los otros.

Un paquete es una parte de un modelo. Cada parte del modelo debe pertenecer a un paquete. Pero para ser funcional, la asignación debe seguir un cierto principio racional, tal como funcionalidad común, implementación relacionada y punto de vista común. UML no impone una regla para componer los paquetes.

Los paquetes ofrecen un mecanismo general para la organización de los modelos/subsistemas agrupando elementos de modelado. Cada paquete corresponde a un submodelo (subsistema) del modelo (sistema). Los paquetes son unidades de organización jerárquica de uso general de los modelos de UML. Pueden ser utilizados para el almacenamiento, el control de acceso, la gestión de la configuración y la construcción de bibliotecas que contengan fragmentos reutilizables del modelo.

Un paquete puede contener otros paquetes, sin límite de anidamiento pero cada elemento pertenece a (está definido en) sólo un paquete.

Los paquetes contienen elementos del modelo al más alto nivel, tales como clases y sus relaciones, máquinas de estado, diagramas de casos de uso, interacciones y colaboraciones; atributos, operaciones, estados, líneas de vida y mensajes están contenidos en otros elementos y no aparecen como contenido directo de los paquetes.

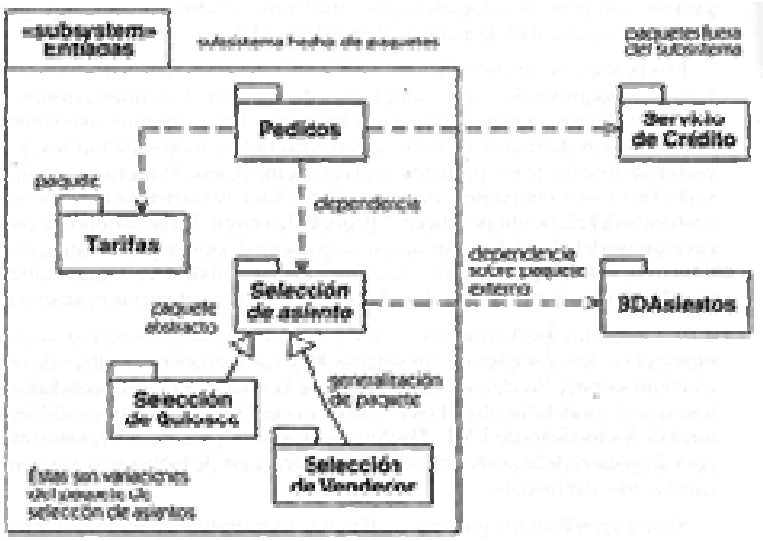
Dependencias en los paquetes

Las dependencias que se presentan entre elementos individuales, pero en un sistema de cualquier tamaño, deben ser vistas en un nivel más alto.

Las dependencias entre paquetes resumen dependencias entre los elementos internos a ellos, es decir, las dependencias del paquete son derivables a partir de las dependencias entre los elementos individuales.

La presencia de una dependencia entre paquetes implica que existe en un enfoque ascendente (una declaración de existencia), o que se permite que exista más adelante en un enfoque descendente (una restricción que limita cualquier otra relación), por lo menos un elemento de relación con el tipo de dependencia indicado entre elementos individuales dentro de los paquetes correspondientes.

Las dependencias múltiples del mismo tipo entre elementos individuales se agregan a una sola dependencia entre los paquetes que contienen los elementos. Si las dependencias entre elementos contienen estereotipos, éste puede ser omitido en la dependencia del paquete, para dar una sola dependencia de alto nivel.



Una clase de un paquete puede aparecer en otro paquete por la importación a través de una relación de dependencia entre paquetes. Todas las clases no son necesariamente visibles desde el exterior del paquete, es decir, un paquete encapsula a la vez que agrupa.

En general, un paquete no puede tener acceso al contenido de otro paquete. Los paquetes son opacos, a menos que sean abiertos por una dependencia de acceso o de importación. La dependencia de acceso indica que el contenido del paquete del proveedor puede aparecer en referencias efectuadas por los elementos del paquete cliente.

En general, un paquete puede ver solamente los elementos de otros paquetes que tienen visibilidad pública.

Los elementos con visibilidad protegida pueden ser vistos únicamente por los paquetes que son descendientes del paquete contenedor de dichos elementos.

Los elementos con visibilidad privada sólo son vistos por su paquete contenedor y anidados. La visibilidad también se aplica a las clases. El permiso de acceso y visibilidad son necesarios para hacer referencia a un elemento.

La dependencia de acceso no modifica el espacio de nombres del cliente no crea las referencias automáticamente, simplemente concede permiso para establecer referencias. La dependencia de importación se utiliza para agregar nombres al espacio de nombres del paquete del cliente como sinónimos de los caminos completos.

Los paquetes se dibujan como rectángulos con pestañas (similar al icono "carpeta"), las dependencias se muestran como flechas con líneas discontinuas. El operador "::" permite designar una clase definida en un contexto distinto del actual.

3.3.3.6 Diagrama de Actividades

El Diagrama de Actividad es una especialización del Diagrama de Estado, organizado respecto de las acciones y usado para especificar:

- Un método
- Un caso de uso
- Un proceso de negocio (Workflow)

Un **diagrama de actividad** muestra la realización de operaciones para conseguir un objetivo. Presentan una visión simplificada de lo que ocurre en un proceso, mostrando los pasos que se realizan, constituyéndose en uno de los diagramas que modelan los aspectos dinámicos del sistema.

Los diagramas de actividad son una extensión de los diagramas de estado. Los diagramas de estado **resaltan los estados** y muestran las actividades que dan lugar a cambios de estado, mientras que los diagramas de actividad **resaltan justamente las actividades.**

Los diagramas de actividad se pueden usar para modelar un Caso de Uso, o una clase, o un método complicado.

Un diagrama de actividades es provechoso para entender el comportamiento de alto nivel de la ejecución de un sistema, sin profundizar en los detalles internos de los mensajes. Los parámetros de entrada y salida de una acción se pueden mostrar usando las relaciones de flujo que conectan la acción y un estado de flujo de objeto.

Un grafo de actividades contiene estados de actividad que representa la ejecución de una secuencia en un procedimiento, o el funcionamiento de una actividad en un flujo de trabajo. En vez de esperar un evento, como en un estado de espera normal, un estado de actividad espera la terminación de su cómputo. Cuando la actividad termina, entonces la ejecución procede al siguiente estado de actividad dentro del diagrama. Una transición de terminación es activada en un diagrama de actividades cuando se completa la actividad precedente. Los estados de actividad no tienen transiciones con eventos explícitos, peor pueden ser abortados por transiciones en estados que los incluyen.

Un grafo de actividades puede contener también estados de acción, que son similares a los de actividad pero son atómicos y no permiten transiciones mientras están activos. Los estados de acción se deben utilizar para las operaciones cortas de mantenimiento.

Un diagrama de actividades puede contener bifurcaciones, así como divisiones de control en hilos concurrentes. Los hilos concurrentes representan actividades que se pueden realizar concurrentemente por los diversos objetos o personas. La concurrencia se representa a partir de la agregación, en la cual cada objeto tiene su propio hilo. Las actividades concurrentes se pueden realizar simultáneamente o en cualquier orden. Un diagrama de actividades es como un organigrama tradicional, excepto que permite el control de concurrencia además del control secuencial.

ELEMENTOS DEL DIAGRAMA DE ACTIVIDADES

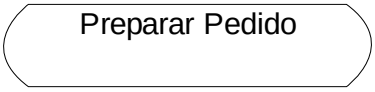
Básicamente un diagrama de actividades contiene:

- Estados de acción
- Estados de actividad
- Transiciones
- Bifurcaciones
- Calles
- Señales.
- Inicio y Terminación

ESTADO DE ACCIÓN

Un estado de actividad representa una actividad: un paso en el flujo de trabajo o la ejecución de una operación. Un grafo de actividades describe grupos secuenciales y concurrentes de actividades. Los grafos de actividades se muestran en diagramas de actividades. Las actividades se enlazan por transiciones automáticas. Cuando una actividad termina se desencadena el paso a la siguiente actividad.

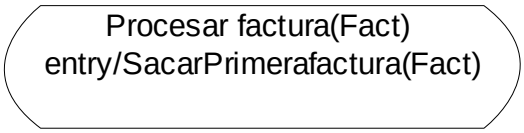
La representación es un rectángulo con las puntas redondeadas, en cuyo interior se representa una acción. La idea central es la siguiente: “Un estado que represente una acción es atómico, lo que significa que su ejecución se puede considerar instantánea y no puede ser interrumpida”



ESTADOS DE ACTIVIDAD

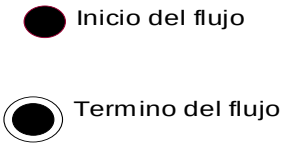
Un Estado de Actividad representa la realización de una o varias tareas que causa un cambio en el sistema. La representación es un rectángulo con las puntas redondeadas, un estado de actividad puede descomponerse en más sub-actividades representadas a través de otros diagramas de actividades. Además estos estados sí pueden ser interrumpidos y tardan un cierto tiempo en completarse.

En los estados de actividad podemos encontrar otros elementos adicionales como son: acciones de entrada (entry) y de salida (exit) del estado en cuestión.



INICIO Y TERMINACION

Un flujo de control debe empezar en algún sitio y terminar en otro al menos que se trate de un flujo infinito. El inicio se representa mediante un círculo completamente relleno. Mientras que el término se representa con un círculo relleno dentro de una circunferencia.

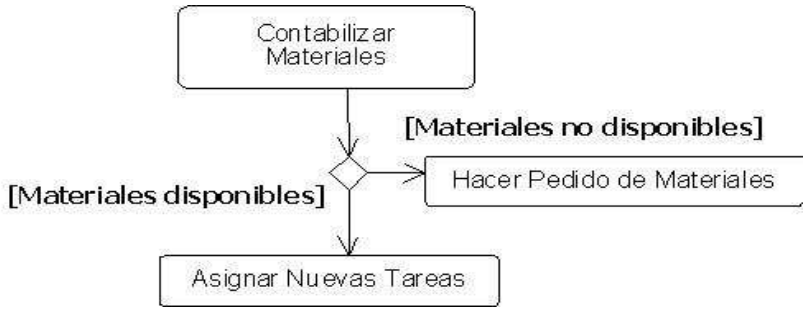


TRANSICIONES

Las transiciones reflejan el paso de un estado a otro, bien sea de actividad o de acción. Esta transición se produce como resultado de la finalización del estado del que parte la transición. Se representa mediante una línea dirigida que va de la actividad que finaliza hacia la actividad a ejecutar. Las flechas que indican la transición no deben ser etiquetadas pues son disparadas automáticamente por la finalización de la primera actividad. Como todo flujo de control debe empezar y terminar en algún momento, podemos indicar esto utilizando dos disparadores de inicio y fin.

BIFURCACIONES (Decisiones)

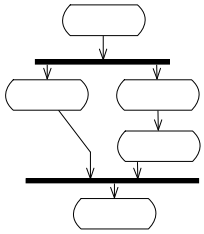
En los diagramas de actividad se pueden incluir caminos alternativos según se cumpla alguna condición. Estas condiciones se representan mediante un rombo al cual llega la transición de la actividad origen y del cual salen las múltiples transiciones a cada actividad destino, cada una con una condición diferente y sin un evento que las dispare. Se pueden incluir la palabra else que indica que ninguna de las expresiones de guarda es verdad. Una expresión de guarda es aquella que puede tomar un valor de verdadero o falso.
Las alternativas de la bifurcación han de ser excluyentes y contemplar todos los casos ya que de otro modo la ejecución quedaría interrumpida.



BARRAS DE SINCRONIZACION

Cuando modelamos flujos de procesos, especialmente en los procesos de negocio, nos encontramos con actividades que se pueden realizar simultáneamente. Estas actividades se pueden modelar con la ayuda de las *Barras de Sincronización*.

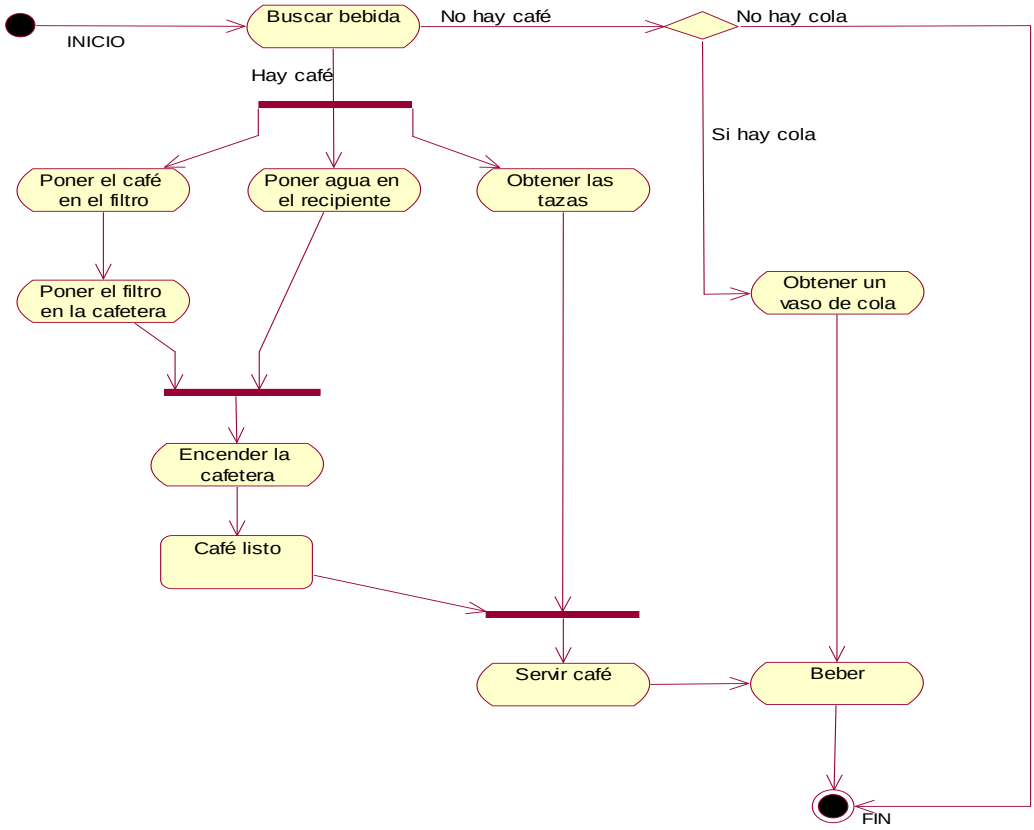
Una **barra de sincronización** se representa como una línea vertical u horizontal gruesa.



En este diagrama se muestra que el flujo de actividades se divide en dos, cada una de las cuales es independiente de la otra y se ejecuta por separado. Al terminar la ejecución de ambas el flujo se vuelve a unir mediante otra barra de sincronización.

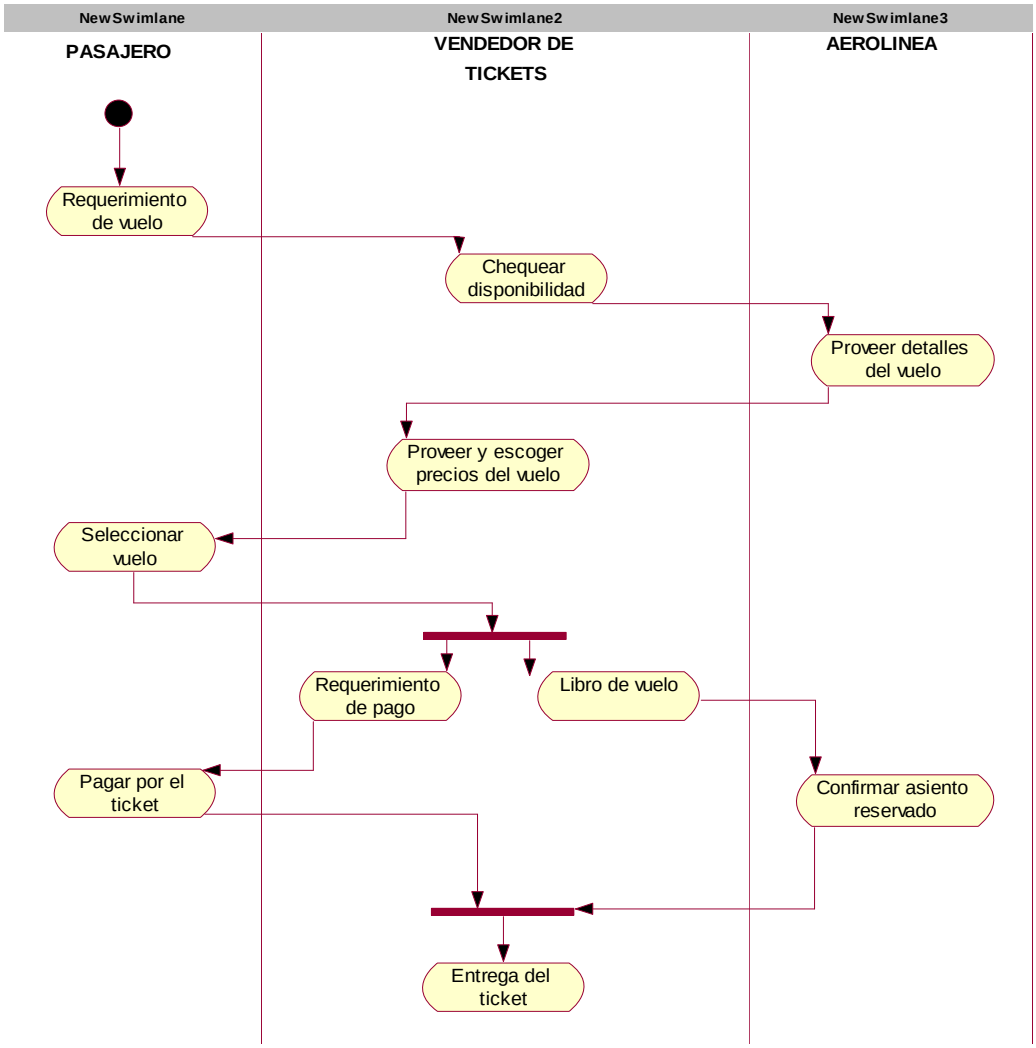
Las barras de sincronización muestran actividades concurrentes es decir que la ejecución de procesos es al mismo tiempo de forma paralela aunque en la realidad puede que no se ejecuten simultáneamente.

Ejemplo:



CALLES

Cuando se modelan flujos de trabajo de organizaciones, es especialmente útil dividir los estados de actividades en grupos, cada grupo tiene un nombre concreto y se denominan calles. Cada calle representa a la parte de la organización responsable de las actividades que aparecen en esa calle.



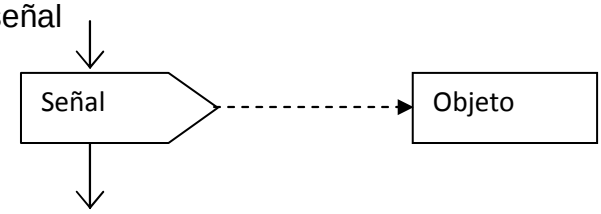
SEÑALES

Algunos objetos pueden enviar señales para la realización de alguna actividad. Estas señales pueden se de dos tipos: **señal de envió y señal de recibo,** aunque no son indispensables para la construcción de diagramas de actividades y pueden evitarse.

Señal de envió

El envió es una señal, desde nuestro diagrama de actividades hacia un objeto, puede ser mostrado como un pentágono convexo que luce como un rectángulo con un triangulo apuntando hacia fuera. El significado de esta señal es mostrada dentro del símbolo.

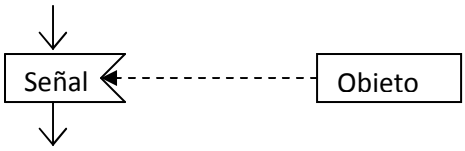
Esta señal se coloca dentro de dos transiciones una entrando desde una actividad y otra saliendo hacia otra actividad. Opcionalmente se puede dibujar una flecha de línea discontinua desde el pentágono convexo, hacia el objeto receptor de la señal



Señal de recibo

El recibo de una señal, por nuestro diagrama de actividades desde un objeto puede ser mostrado como un pentágono cóncavo que luce como un rectángulo con un triangular entrante en uno de sus lados. El significado de esta señal es mostrado dentro del símbolo.

Esta señal se coloca dentro de dos transiciones una entrando desde una actividad y la otra saliendo hacia otra actividad. Opcionalmente se puede dibujar una flecha de línea discontinua desde el objeto que envía la señal hacia el pentágono cóncavo



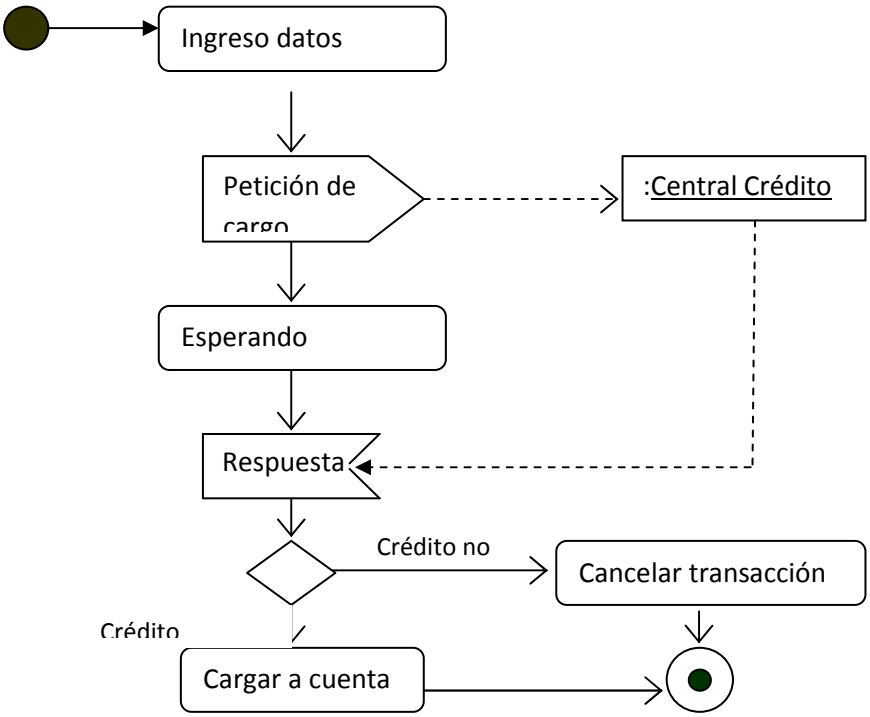
Ejemplo:

La siguiente descripción corresponde a un proceso típico de pago con tarjeta de crédito. Cuando se realizan pagos con tarjeta de crédito debe introducirse los datos de la tarjeta, luego se dispara una señal de petición de cargo, hacia la central de crédito mientras esto ocurre estaremos esperando a que la central de crédito nos envié la señal de autorizado, para finalmente hacer el cargo respectivo a la tarjeta.

Solución:

Observe el envió de la señal de **Petición de cargo** desde el proceso hacia el objeto **Central de Crédito**, la cual enviara a su vez la **Respuesta,** para luego

pasar a preguntar por el estado del crédito. Si ha sido aprobado se hará el cargo respectivo, sino cancelamos la transacción.



3.3.3.7 Diagramas de Interacción

Diagramas de Secuencia:

- Muestra la secuencia de mensajes entre objetos durante un escenario concreto
- Cada objeto viene dado por una barra vertical
- El tiempo transcurre de arriba abajo
- Cuando existe demora entre el envío y la atención se puede indicar usando una línea oblicua

Diagramas de Colaboración:

- Son útiles en la fase exploratoria para identificar objetos.
- La distribución de los objetos en el diagrama permite observar adecuadamente la interacción de un objeto con respecto de los demás

- La estructura estática viene dada por los enlaces; la dinámica por el envío de mensajes por los enlaces

Colaboración

Es una descripción de una colección de objetos que interactúan para implementar un cierto comportamiento dentro de un contexto. Describe una sociedad de objetos cooperantes unidos para realizar un cierto propósito. Una colaboración contiene ranuras que son rellenadas por los objetos y enlaces en tiempo de ejecución. Una ranura de colaboración se llama Rol porque describe el propósito de un objeto o un enlace dentro de la colaboración.

Un rol clasificador representa una descripción de los objetos que pueden participar en una ejecución de la colaboración, un rol de asociación representa una descripción de los enlaces que pueden participar en una ejecución de colaboración. Un rol de clasificador es una asociación que está limitada por tomar parte en la colaboración.
Las relaciones entre roles de clasificador y asociación dentro de una colaboración sólo tienen sentido en ese contexto. En general fuera de ese contexto no se aplican las mismas relaciones.

Una Colaboración tiene un aspecto *estructural* y un aspecto de *comportamiento:*

El aspecto estructural es similar a una vista estática: contiene un conjunto de roles y relaciones que definen el contexto para su comportamiento.

El comportamiento es el conjunto de mensajes intercambiados por los objetos ligados a los roles. Tal conjunto de mensajes en una colaboración se llama Interacción. Una colaboración puede incluir una o más interacciones.

Interacción

Es el conjunto de mensajes intercambiados por los roles de clasificador a través de los roles de asociación. Un mensaje es una comunicaión unidireccional entre dos objetos, un flujo de objeto con la información de un remitente a un receptor.

Un mensaje puede tener parámetros que transporten valores entre objetos. Un mensaje puede ser una señal (comunicación explícita entre objetos, con nombre y asíncrona) o una llamada (la invocación síncrona de una operación con un mecanismo para el control, que retorna posteriormente al remitente). Un patrón de intercambios de mensajes que se realizan para lograr un propósito específico es lo que se denomina una interacción.

Patrón

Un patrón es una colaboración parametrizada, junto con las pautas sobre cuándo utilizarlo. Un parámetro se puede sustituir por diversos valores, para producir distintas colaboraciones. Los parámetros señalan generalmente las ranuras para las clases. El uso de un patrón se representa como una elipse de línea discontinua conectada con cada una de las clases por una línea discontinua, que se etiqueta con el nombre del rol.

Diagramas de Secuencia

Diagrama que muestra las interacciones entre los objetos organizadas en una secuencia temporal. En particular muestra los objetos participantes en la interacción y la secuencia de mensajes intercambiados.

Representa una interacción, un conjunto de comunicaciones entre objetos organizadas visualmente por orden temporal. A diferencia de los diagramas de colaboración, los diagramas de secuencia incluyen secuencias temporales pero no incluyen las relaciones entre objetos. Pueden existir de forma de descriptor (describiendo todos los posibles escenarios) y en forma de instancia

(describiendo un escenario real). Dentro del conjunto de mensajes representados dispuestos en una secuencia temporal, cada rol en la secuencia se muestra como una línea de vida, es decir, una línea vertical que representa el rol durante cierto plazo de tiempo, con la ibnteracción completa. Los mensajes se muestran como flechas entre líneas de vida.

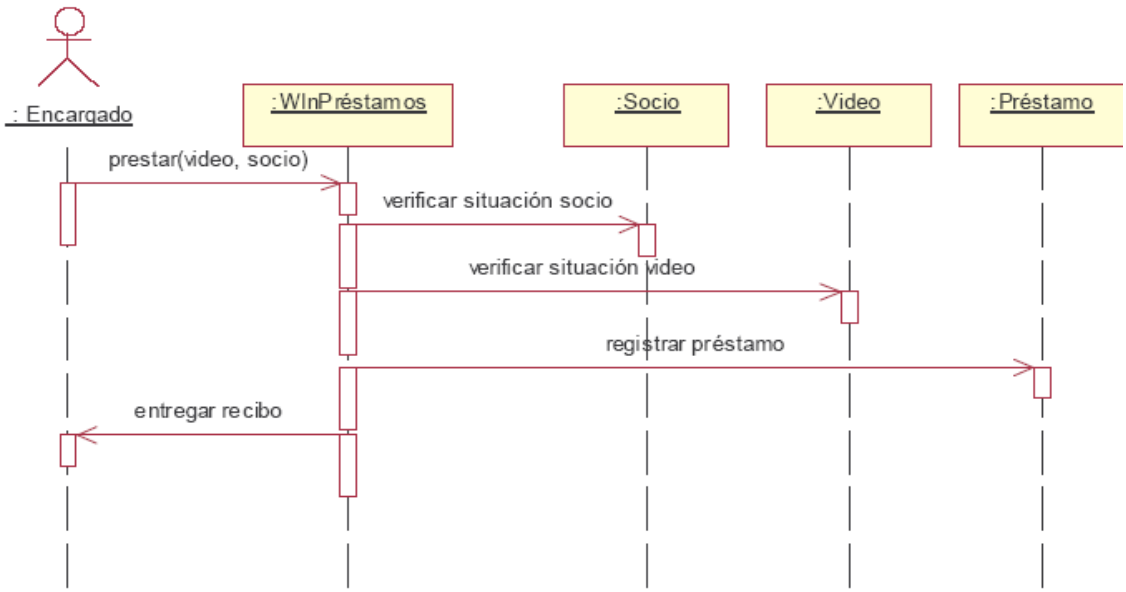
Un diagrama de secuencia puede mostrar un escenario, es decir, una historia individual de transacción. Un uso de un diagrama de secuencia es mostrar la secuencia del comportamiento de un caso de uso.

Un diálogo de secuencia posee dos dimensiones:

- *Vertical:* representa el tiempo,
- *Horizontal:* representa los objetos que participan en la interacción.

En general, el tiempo avanza hacia abajo dentro de la página (se pueden invertir los ejes si se desea). Con frecuencia sólo son importantes las secuencias de mensajes pero en aplicaciones de tiempo real el eje temporal puede ser una métrica.

La ordenación horizontal de los objetos no tiene ningún significado.



Cada objeto representa una columna distinta, se pone un símbolo de objeto al final de la flecha que representa el mensaje que ha creado el objeto; está situada en el punto vertical que denota el instante en que se crea el objeto. Esta se conoce como línea de división del objeto. Se pone una X grande en el punto en que deja de existir el objeto o en el punto en que el objeto se destruye a sí mismo. Para el periodo durante el cual esté activo el objeto, la línea de vida se amplía para ser una línea doble continua. Si el objeto se llama a sí mismo, entonces se superpone otra copia de la doble línea para mostrar la doble activación. El orden relativo de los objetos no tiene significado aún cuando resulta útil organizarlos de modo que se minimice la distancia de las flechas.

Cada mensaje se representa mediante una flecha horizontal que va desde la línea de vida del objeto que envió el mensaje hasta la línea de vida del objeto que ha recibido el mensaje. Si un mensaje requiere un cierto tiempo para llegar a su destino, entonces la flecha del mensaje se dibuja diagonalmente hacia abajo.

Para un flujo de objeto asíncrono entre objetos activos, los objetos se representan mediante líneas dobles continuas y los mensajes se representan como flechas. Se pueden enviar simultáneamente dos mensajes pero no se pueden recibir simultáneamente porque no sxe puede garantizar una recepción simultánea.
Las bifurcaciones se muestran partiendo la línea de vida del objeto. Cada bifurcación puede enviar y recibir mensajes. Eventualmente las líneas de vida del objeto tienen que fusionarse de nuevo.

Un diagrama de secuencia también se puede mostrar en forma de descriptor, en el cual los constituyentes son roles en lugar de objetos. Este diagrama muestra en el caso general, no una sola ejecución del mismo. Los diagramas del nivel de descriptores se dibujan sin subrayados porque los símbolos denotan roles y no objetos individuales.

3.3.3.8 Diagramas de Colaboración

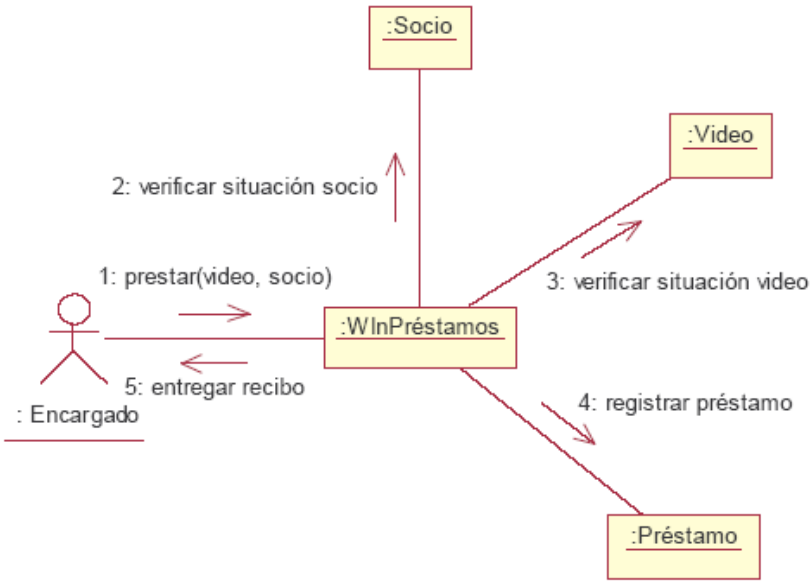


Figura 14: Diagrama de Colaboración.
Fuente: UML *http://www.creangel.com/uml/diagramas.*

Diagrama que muestra interacciones organizadas alrededor de los roles. A diferencia de los diagramas de secuencia, los diagramas de colaboración muestran explícitamente las relaciones de los roles. Por otra parte, un diagrama de colaboración no muestra el tiempo como una dimensión aparte, por lo que resulta necesario etiquetar con números de secuencia tanto la secuencia de mensajes como los hilos concurrentes.

Un diagrama de colaboración es también un diagrama de clases que contiene roles de clasificador y roles de asociación en lugar de sólo clasificadores y asociaciones.

Los roles de clasificador y los de asociación describen la configuración de los objetos y de los enlaces que pueden ocurrir cuando se ejecuta una instancia de la colaboración. Cuando se instancia una colaboración, los objetos están ligados a los roles de clasificador y los enlaces a los roles de asociación. El rol de asociación puede ser desempeñado por varios tipos de enlaces temporales,

tales como argumentos de procedimiento o variables locales del procedimiento. Los símbolos de enlace pueden llevar estereotipos para indicar enlaces temporales.

Un uso de un diagrama de colaboración es mostrar la implementación de una operación. La colaboración muestra los parámetros y las variables locales de la operación, así como asociaciones más permanentes. Cuando se implementa el comportamiento, la secuencia de los mensajes corresponde a la estructura de llamadas anidadas y el paso de señales del programa.

Un diagrama de secuencia muestra secuencias en el tiempo como dimensión geométrica, pero las relaciones son implícitas. Un diagrama de colaboración muestra relaciones entre roles geométricamente y relaciona los mensajes con las relaciones, pero las secuencias temporales están menos claras.

Aunque las colaboraciones muestran directamente la implementación de una operación, pueden también mostrar la realización de una clase entera. En este uso, muestran el contexto necesario para implementar todas las operaciones de una clase. Esto permite que el modelador vea los roles múltiples que los objetos pueden desempeñar en varias operaciones.

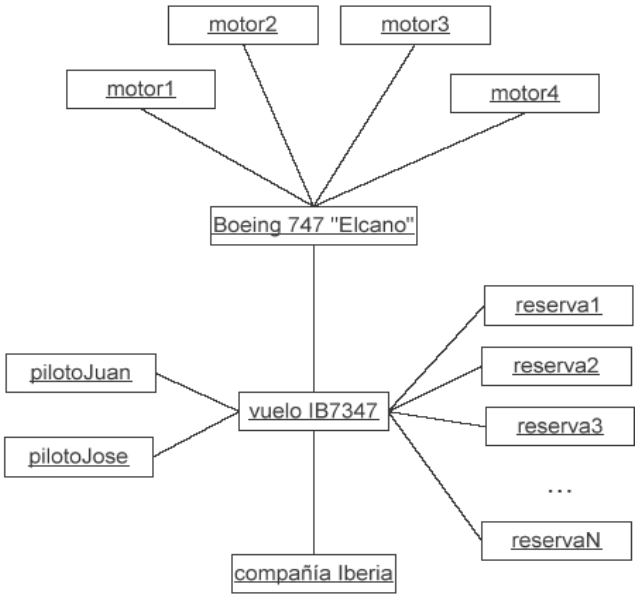
Mensajes

Los mensajes se muestran como flechas etiquetadas unidas a los enlaces. Cada mensaje tiene un número de secuencia, una lista opcional de mensajes precedentes, una condición opcional de guarda, un nombre y una lista de argumentos y un nombre de valor de retorno opcional.
El nombre de serie incluye el nombre (opcional) de un hilo. Todos los mensajes del mismo hilo se ordenan secuencialmente. Los mensajes de diversos hilos son concurrentes a menos que haya una dependencia secuencial explícita.

Flujos

Generalmente, un diagrama de colaboración contiene un símbolo para un objeto durante una operación completa. Sin embargo, a veces, un objeto contiene diferentes estados que se deban hacer explícitos. Por ejemplo, un objeto pudo cambiar de localización o sus asociaciones pudieron diferenciarse.
Los diferentes símbolos de objeto que representan un objeto se pueden conectar usando flujos "become" o "conversión". Un flujo "become" es una transición, a apartir de un estado de un objeto a otro. Se dibuja como una flecha de línea dsicontinua con el estereotipo "become" o "conversión" y puede ser etiquetado con un número de serie para mostrar cuando ocurre. Un flujo de conversión también se utiliza para mostrar la migración de un objeto a partir de una localización a otra distinta.

3.3.3.9 Diagramas de Objetos



15

Figura 15: Diagrama de Objetos.
Fuente: UML *http://www.creangel.com/uml/diagramas.*

Objeto es una entidad discreta con límites bien definidos y con identidad, es una unidad atómica que encapsula estado y comportamiento. La encapsulación en un objeto permite una alta cohesión y un bajo acoplamiento.

El Objeto es reconocido también como una instancia de la clase a la cual pertenece.

La encapsulación presenta tres ventajas básicas:

- Se protegen los datos de accesos indebidos
- El acoplamiento entre las clases se disminuye
- Favorece la modularidad y el mantenimiento

Un objeto se puede ver desde dos perspectivas relacionadas: como una entidad de un determinado instante de tiempo que posee un valor específico (Un objeto puede caracterizar una entidad física -coche-) y como un poseedor de identidad que tiene distintos valores a lo largo del tiempo (abstracta -ecuación matemática-).

Cada objeto posee su propia identidad exclusiva y se puede hacer referencia a él mediante una denominación exclusiva que permite accederle.
El Modelado de Objetos permite representar el ciclo de vida de los objetos a través de sus interacciones. En UML, un objeto se representa por un rectángulo con un nombre subrayado.

- Objeto = Identidad + Estado + Comportamiento
- El estado está representado por los valores de los atributos.
- Un atributo toma un valor en un dominio concreto.

La regla general para la notación de instancias consiste en utilizar el mismo símbolo geométrico que el descriptor. En la instancia se muestran los posibles valores pero las propiedades compartidas sólo se ponen de manifiesto en el descriptor. La notación canónica es un rectángulo con tres compartimientos. En el primero va el nombre del objeto, en el segundo sus atributos y en el tercero sus operaciones. Este último puede ser omitido si así se prefiere.

Oid (Object Identifier)

Cada objeto posee un oid. El oid establece la identidad del objeto y tiene las siguientes características:

- Constituye un identificador único y global para cada objeto dentro del sistema.
- Es determinado en el momento de la creación del objeto.
- Es independiente de la localización física del objeto, es decir, provee completa independencia de localización.
- Es independiente de las propiedades del objeto, lo cual implica independencia de valor y de estructura.
- No cambia durante toda la vida del objeto. Además, un oid no se reutiliza aunque el objeto deje de existir.

No se tiene ningún control sobre los oids y su manipulación resulta transparente. Sin embargo, es preciso contar con algún medio para hacer referencia a un objeto utilizando referencias del dominio (valores de atributos).

Características alrededor de un objeto:

Estado:

El estado evoluciona con el tiempo. Algunos atributos pueden ser constantes, el comportamiento agrupa las competencias de un objeto y describe las acciones y reacciones de ese objeto. Las operaciones de un objeto son consecuencia de un estímulo externo representado como mensaje enviado desde otro objeto.

Persistencia:

La persistencia de los objetos designa la capacidad de un objeto trascender en el espacio/tiempo, podremos después reconstruirlo, es decir, cogerlo de memoria secundaria para utilizarlo en la ejecución (materialización del objeto). Los lenguajes OO no proponen soporte adecuado para la persistencia, la cual

debería ser transparente, un objeto existe desde su creación hasta que se destruya.

Comunicación:

Un sistema informático puede verse como un conjunto de objetos autónomos y concurrentes que trabajan de manera coordinada en la consecución de un fin específico. El comportamiento global se basa pues en la comunicación entre los objetos que la componen.

Categorías de objetos:

- Activos o Pasivos
- Cliente -- Servidores, Agentes

- *Objeto Activo*: posee un hilo de ejecución (thread) propio y puede iniciar una actividad.
- *Objeto Pasivo*: no puede iniciar una actividad pero puede enviar estímulos una vez que se le solicita un servicio.
- *Cliente* es el objeto que solicita un servicio.
- *Servidor* es el objeto que provee el servicio solicitado.
- Los agentes reúnen las características de clientes y servidores. Son la base del mecanismo de delegación. Introducen indirección: un cliente puede comunicarse con un servidor que no conoce directamente.

Mensajes:

La unidad de comunicación entre objetos se llama mensaje. El mensaje es el soporte de una comunicación que vincula dinámicamente los objetos que fueron separados previamente en el proceso de descomposición. Adquiere toda su fuerza cuando se asocia al polimorfismo y al enlace dinámico. Un estímulo causará la invocación de una operación, la creación o destrucción de un objeto o la aparición de una señal. Un mensaje es la especificación de un estímulo.

Fuente: *http://www.creangel.com/uml/diagramas.php*

CAPITULO 4. CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

- El presente trabajo contribuye a los Centros Médicos en un mejor control de la información, de forma que se optimiza el tiempo tanto de pacientes como el de los médicos.

- La información se mantendrá en un nivel de privacidad, al que solo podrán acceder los usuarios del sistema.

- Las herramientas que se utilizaron para el desarrollo del sistema como flash, Adobe Dreamweaver CS3 son muy amigables en cuanto a la presentación que se da para el usuario permitiendo de esta forma obtener un producto de alta calidad y confiabilidad.

4.2 RECOMENDACIONES

- Para contar con un buen mantenimiento de la aplicación desarrollada debe existir una persona que cumpla las funciones de un administrador de base de datos y webmaster dentro del Centro Médico.

- La base de datos de la aplicación deberá ser respaldada al finalizar la jornada laboral, ya que al llevar un control exhaustivo sobre los pacientes demanda información diaria.

- Es necesario que a los nuevos usuarios del sistema tengan la capacitación conveniente para que el uso del sistema sea adecuado y la información no sea dañada.

- El responsable de administrar la base de datos deberá recibir capacitación para dicha actividad, para que no haya inconvenientes en el control de la información.

BIBLIOGRAFÍA

- Pressman Roger, Ingeniería de Software, MC. Graw-Hill, 2002.
- Pressman Roger, Ingeniería del software. Un enfoque práctico, Madrid, McGraw-Hill / Interamericana de España, 1997
- S. Murugesan, Y. Deshpande , S. Hansen, A. Ginige. “Web Engineering : A New Discipline for Development of Web - Based Systems.” Lecture Notes in Computer Science 2016 Springer, 2001.
- S. Dart, “Containing the Web Crisis Using Configuration Management,” Proc. 1st ICSE Workshop on Web Engineering, ACM, Los Angeles, May 1999.
- L. Olsina, G. Lafuente, G. Rossi. “Specifying Quality Characteristics and Attributes for Websites.” Lecture Notes in Computer Science 2016 Springer, 2001.
- Gamma, Y., Helm, R., Johnson R. y Vlissides, J. (1995). Design Patterns: Elements of reusable object-oriented software. USA: Addison Wesley.

Internet

- Booch G. 1998. Software Architecture and the UML. Presentación disponible en: http://www.rational.com/uml como arch.zip.
- German, D. (2003). The Object Oriented Hypermedia Design Method. [Documento en línea]. Disponible: http://www.telemidia.puc-rio.br/oohdm/oohdm.html.
- Koch, N. (2002). Ingeniería de Requisitos en Aplicaciones para la Web–Un estudio comparativo. [Documento en línea]. Disponible: http://www.lsi.us.es/docs/ informes/LSI-2002-4.pdf.
- Sánchez, M. (s. f.). Interfaz de Usuario en el Desarrollo de un Simulador de Conducción [Documento en línea] Disponible: http://www.cc3.net/sanchez/Sanchez_M.htm.
- Schwabe, D. y Rossi, G. (1998). Developing Hypermedia Applications using OOHDM [Documento en línea]. http://www.oohdm.inf.pucrio.br:8668/space/ pessoas+ligadas+ao+OOHDM/ExOOHDM.pdf.

- Schwabe, D., Rossi, G. y Simone, J. (s. f.). Systematic Hypermedia Application Design with OOHDM. [Documento en línea]. http://wwwx.cs.unc.edu /~barman/HT96/P52/section1.html.
- Silva, D. y Mercerat, B. (2001). Construyendo aplicaciones web con una metodología de diseño orientada a objetos. [Documento en línea]. Disponible: www.unab.edu.co/editorialunab/revistas/rcc/pdfs/r22_art_c. pdf.
- http://www.desarrolloweb.com/articulos/713.php?manual=27
- http://www.guiaweb.gob.cl/guia/capitulos/tres/accesorapido.htm
- http://www.hostito.com/es/faq/index.php
- http://www.sindominio.net/ayuda/glosario/
- http://es.wikipedia.org/wiki/Dreamweaver
- http://www.buscandohost.com/Glosario.asp
- http://www.16-bits.com.ar/archivos/hacia-un-nuevo-modelo-para-construir-la-web/
- http://www.informandote.com/jornadasingweb/programa.asp
- http://www.itlp.edu.mx/publica/tutoriales/analisis/24.htm
- http://www.ati.es/gt/LATIGOOO/Op96/Ponen6/atio6p06.html
- http://www.servidorti.uib.es/jbidi/jbidi2000/02_2000.pdf
- http://www.infor.uva.es/~jvegas/cursos/buendia/pordocente/node21.html
- http://www.informandote.com/jornadasIngWEB/articulos/jiw01.pdf
- http://www.rspa.com/spi/webe.html
- http://www.geocities.com/rcascos/
- http://gidis.ing.unlpam.edu.ar/personas/olsinal/olsinal.html
- http://www5.ulpgc.es/servidores/bib-inge/indices/647441.htm
- http://www.creangel.com/uml/home.php
- http://www.guiaweb.gob.cl/guia/capitulos/tres/accesorapido.htm
- http://www.servidorti.uib.es/jbidi/jbidi2000/02_2000.pdf
- http://www.creangel.com/uml/diagramas.php
- http://www.dcc.uchile.cl/~afierro/d_clases.html

ANEXOS