71

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DESARROLLO E IMPLEMENTACION DE UN SISTEMA DE COMPUTO PARA EL MANEJO DE LA INFORMACION DE LA SUBDIRECCION DE DESARROLLO PROFESIONAL DE PEMEX Y DE LA SECCION DE INGENIERIA PETROLERA DEPFI-UNAM

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERA EN COMPUTACION

P R E S E N T A

MA. GUADALUPE MENDOZA JUAREZ

Directora de Tesis:
M. en A. Carmen Maldonado Susano

OCTUBRE, 2002

Agradecimientos

A la Universidad Nacional Autónoma de México y principalmente a la Facultad de Ingeniería por haberme abierto sus puertas y brindado una formación profesional. Un agradecimiento especial a los profesores de la Facultad de Ingeniería por haberme dado una parte importante de su tiempo y sus conocimientos.

A Dios
Por permitirme estar aquí

A Carmen Maldonado Susano
Por la amistad, valiosos consejos, y tiempo que me ha proporcionado

A mis padres
Juana y Manuel
Por su ejemplo y Amor

A mis hermanos y familiares
Miguel, Ramón, Carmen y Manuel, Martín, José y Aurea, Reyna, Jessica, Lupita, Alex y J. Armando.
Por su apoyo incondicional

A Luis Torres Arriaga
Por su paciencia, respeto, y ejemplo que siempre me ha demostrado y cuya amistad es invaluable.

A mis amiguitos
Lucesita Jiménez Silva, Rene Trujillo, Silvia Julian Sánchez, Leonardo el "Maestro", Gustavo I. Molina, Ma. Leonor Saucedo y Edith Figueroa Nolasco
Por su compañerismo, valiosa amistad e incondicional apoyo.

¡Muchas Gracias por haber estado siempre junto a mi!
Cada uno de ustedes ocupan un lugar importante en mi corazón.

OBJETIVO

Desarrollar un sistema de cómputo para el manejo de la información de las publicaciones que generan los investigadores de la Subdireccion de Desarrollo Profesional y de la UNAM, logrando con ésto mantener el oportuno y fácil acceso a la información, así como la actualización y la vinculación entre estas áreas del conocimiento.

ÍNDICE

CAPÍTULO 1

DEFINICIÓN DEL SISTEMA

1.1 Introducción

El presente trabajo está enfocado al desarrollo de un sistema que permita el manejo de la información que se genera en la Subdirección de Desarrollo Profesional de PEMEX y de la Sección de Ingeniería Petrolera, empleándose algunas de las metodologías más recientes del desarrollo de sistemas.

La Subdirección de Desarrollo Profesional de PEMEX es un organismo que se encarga de apoyar al personal de PEMEX en su crecimiento profesional, para obtener así un alto desempeño en sus funciones laborales.

La Sección de Ingeniería Petrolera, tiene como objetivo apoyar a estudiantes y profesores con material y equipo, para propiciar de esta forma un mejor desempeño académico.

Actualmente, el proceso que se realiza para el manejo de información entre la Sección de Ingeniería Petrolera y la Subdirección de Desarrollo Profesional de PEMEX es una tarea tediosa y poco segura, provocando así gran pérdida de tiempo en el área de investigación.

Con el desarrollo e implementación del sistema se pretende agilizar el intercambio de información entre estas dependencias, así como la reducción de costos.

Para la realización de este trabajo, se hizo una investigación sobre el tipo de tecnología más adecuado para satisfacer los requerimientos del problema. Concluyendo así el capítulo 1.

En el capítulo 2, se realiza un análisis del problema, presentando los requerimientos del sistema de software.

En el capítulo 3, se presenta un diseño del sistema en general.

En el capítulo 4, se generan las interfaces del sistema y el manual de usuario.

1.2 Planteamiento del Problema

Actualmente, los alumnos y personal del área de Ingeniería Petrolera utilizan, como parte del apoyo en su desempeño académico y laboral, los artículos realizados sobre Sistemas de Producción Petrolera. Estos artículos se elaboran en las diferentes regiones petroleras de la República Mexicana. A continuación se listan estas regiones.

Región Norte que incluye los centros: Veracruz, *Veracruz.*; Reynosa, *Tamaulipas.*; Altamira, *Tamaulipa* y Poza Rica, *Veracruz.*

Región Sur que cuenta con tan solo: Villahermosa, *Tabasaco.*

Región Marina Noreste que cuenta con: Ciudad del Carmen, *Campeche.*

Región Marina Suroeste que está en: Ciudad del Carmen, *Campeche.*

Sede México, D.F que incluye a: PEP(*Perforación y Exploración*).Planeación, IMP (*Instituto Mexicano del Petróleo*), INFOTEC, PEP.STDP y la Sección de Ingeniería Petrolera de la Facultad de Ingeniería de la UNAM.

Una vez elaborados los artículos, el proceso que se sigue para que el personal del área de Ingeniería Petrolera tenga acceso a esta información es el siguiente:

- Digitalizar los artículos que se elaboren en cada dependencia, para su almacenamiento.
- Investigar dónde puede estar el artículo que se desea.
- Llamar por teléfono a la dependencia que cuenta con el o los artículos y hacer la solicitud.
- Si la información digitalizada no ocupa mucho espacio, se podrá enviar por correo electrónico, de lo contrario, se solicitan los servicios de mensajería para su envío. Cuando la distancia por recorrer es muy grande, los costos de mensajería son altos y el riesgo de que la información se pierda y demore en su entrega, es también grande.
- Este proceso se realiza constantemente por el personal del área de Ingeniería Petrolera y la Subdirección de Desarrollo Profesional. Creándose una labor tediosa, de altos costos, poco segura y demasiado lenta en su entrega, provocando así gran pérdida de tiempo en el área de investigación.

Por esas razones, entre otras se propone este trabajo, para evitar las pérdidas y tener un sistema que nos permita el fácil acceso a la información, bajos costos, rapidez y eficiencia; y sobre todo la vinculación entre la UNAM y PEMEX.

1.3 Historia de las Redes

Conforme las computadoras se van acoplando a nuestra vida diaria, cada vez las usamos más para resolver nuestros problemas. Una sola computadora resulta muy valiosa por su capacidad para procesar información sin necesidad de influencia externa. Las computadoras se emplean para realizar con eficiencia diversas tareas:

- Procesamiento de texto.
- Administración de base de datos.
- Contabilidad y análisis financiero.
- Diseño gráfico asistido por computadora (CAD) y muchas cosas más.

Sin embargo, consideremos las capacidades adicionales que estarían disponibles si se conectaran unas con otras y desplegaran la información contenida en ellas. La puerta se abriría hacia un mundo mucho mayor de información. Una red hace posible todo esto y más.

Las *redes* constan de dos o más computadoras conectadas entre sí y permiten compartir recursos e información. La información por compartir suele consistir en archivos y datos. Los recursos son los dispositivos o las áreas de almacenamiento de datos de una computadora.

Para comprender mejor lo que son las redes y la forma en que mejoran nuestra capacidad para manejar más eficientemente las computadoras, es importante conocer la manera en que aquéllas han evolucionado hasta llegar a lo que son en la actualidad.

1.3.1 Primeras computadoras

Hace veinte años las tarjetas perforadas constituyeron uno de los medios para alimentar a las computadoras con información para procesamiento. Uno se sentaba ante una perforadora y creaba con un teclado un conjunto de tarjetas con perforaciones rectangulares. Cada tarjeta representaba una línea de código o de datos del programa, la que después alimentaría a la computadora para que ésta la procesara. En su momento, la información contenida en las tarjetas perforadas la leía un sistema de macrocomputadora, se procesaba y se imprimían los resultados. El proceso de leer información y procesarla como un todo se conoce actualmente como *procesamiento por lotes*.

La siguiente gran mejora en la alimentación a la macrocomputadora, de datos para procesamiento, fue el uso de terminales tontas. Así, en vez de sentarse ante una perforadora y producir tarjetas perforadas, uno se sentaba ante una *terminal "tonta"* (una pantalla y un teclado conectado a la macrocomputadora) y tecleaba la información como se observa en la figura1.3.1.1.

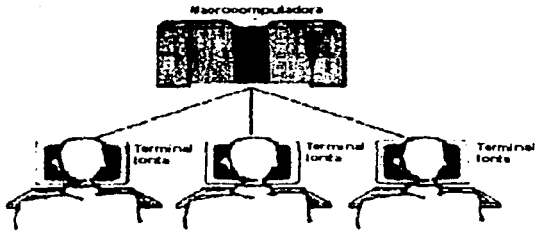


Figura. 1.3.1.1. Terminales "tontas" conectadas directamente a una macrocomputadora.

Una *terminale "tonta"* es aquella que no realiza procesamiento alguno en la terminal misma, sino que se utiliza para enviar datos a la computadora anfitriona por medio del teclado y para recibirlos lo hace por medio de la pantalla. A diferencia de una terminal inteligente, que cuenta con un procesador programable que le permite efectuar operaciones de computadora diferentes a los de entrada y salida.

Una *macrocomputadora* es aquélla computadora grande empleada para aceptar y producir grandes cantidades de datos, sin que necesite realizar cálculos elaborados con cualquiera de esos datos.

Una *computadora anfitriona* es aquélla a la que están conectadas las terminales tontas. Las computadoras anfitrionas pueden ser macrocomputadoras o computadoras pequeñas.

En los años sesenta comenzó a florecer un nuevo tipo de servicio de red comercial conocido como tiempo compartido. El *tiempo compartido* permitió que se instalaran las terminales en lugares geográficamente aislados de la computadora anfitriona, en locales de negocios o en centros de cómputo específicos, desde donde podrían servir para tener acceso a los recursos de cómputo de la computadora anfitriona. Las terminales tontas se conectaban a la computadora anfitriona por medio de líneas telefónicas alquiladas como se muestra en la figura 1.3.1.2. La computadora anfitriona asignaba y distribuía su tiempo entre las diferentes terminales que solicitaban su servicio. Cuando se enviaba un trabajo de procesamiento por lotes a la computadora central, ésta lo pasaba a la macrocomputadora, para su verdadero procesamiento. Cuando la macrocomputadora terminaba el procesamiento del trabajo, la computadora anfitriona intermedia guardaba los resultados y los enviaba de regreso a la terminal "tonta" o a alguna impresora. Solían usarse varias computadoras anfitrionas pequeñas para proporcionar el acceso de las terminales "tontas" a una sola macrocomputadora.

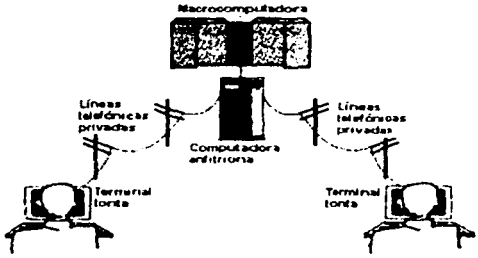


Figura 1.3.1.2. Acceso a una macrocomputadora mediante líneas telefónicas privadas

En esta etapa se utilizaron dos tipos de redes para permitir la comunicación entre las computadoras. La conexión de las terminales "tontas" con las computadoras anfitrionas se consideró como una red. La conexión entre estas últimas y la macrocomputadora era otra red.

Durante este mismo periodo se dieron muchos más avances, los que hicieron todavía más fácil el acceso y el uso de los recursos de cómputo de una macrocomputadora y los de las computadoras anfitrionas.

En vez del procesamiento por lotes estándar requerido anteriormente, se tuvo la capacidad para procesar información en tiempo real. El *procesamiento en tiempo real* permitió que los usuarios vieran el resultado de la información procesada en cuanto se tecleaba. El tiempo real es el tiempo de respuesta del sistema operativo a una petición, en un tiempo acotado.

Conforme se dispuso de más servicios de tiempo compartido, los usuarios de esos sistemas se encontraron en un predicamento interesante. Cada servicio tenía normalmente su propia terminal y requería una línea alquilada por separado para la conexión. También era necesario, por lo general, una terminal aparte, debido a que cada computadora anfitriona tenía su propio método para comunicarse con las terminales, pues no había un estándar.

El ASCII (American Standard Code for Information Interchange o Código Estándar Americano para Intercambio de Información) fue adoptado en 1964 por la Organización Norteamericana de Patrones. El ASCII permite que los 128 caracteres, incluidas las letras del alfabeto, los 10 dígitos numéricos (0-9), así como otros símbolos, puedan representarse en formato binario de 0 y 1. El formato binario para el juego de caracteres ASCII se sirve de una representación de 7 bits, que permite 128 combinaciones posibles.

Una vez establecido el ASCII como el método estándar para transmitir caracteres, se necesitó otro estándar para especificar la manera en que los datos serían transferidos por el cable. Se perfeccionó la norma RS-232C para especificar los voltajes y los parámetros eléctricos de comunicación empleados para conectar dispositivos.

La norma RS-232-C , corresponde a la tercera versión revisada de la norma original RS-232. Esta norma establece:

- Especificación mecánica: características físicas del conector
- Especificación funcional: indica los circuitos que están conectados al conector así como el significado de cada una de sus patillas
- Especificación eléctrica
- Especificación procedual: es el protocolo, es decir, el establecimiento de la secuencia legal de acciones.

Con la llegada de los estándares ASCII y RS-232C frecuentemente se utilizaba una sola terminal "tonta" para acceder a muchos tipos de computadoras anfitrionas y servicios. Se crearon y adoptaron estándares adicionales para especificar protocolos que debían aplicarse a la comunicación.

1.3.2 Introducción a las computadoras personales

En 1981, IBM introdujo la IBM PC y puso en escena lo que sería el futuro de la computación personal. Aunque ya habían sido introducidas varias computadoras personales unos cuantos años antes que la IBM PC, fue ésta la que tuvo la primicia de lo que iba a ser una revolución en la computación. Las computadoras personales (*PC*) proporcionaron la capacidad de cómputo en una sola unidad para una sola persona. En vez de tener una terminal "tonta" conectada a una computadora anfitriona, a partir de ese momento se tuvo una computadora independiente puesta en el escritorio. Por supuesto, la computadora personal no era tan poderosa como las macrocomputadoras o minicomputadoras, pero tampoco era necesario, ya que sólo tenía que dar servicio a las necesidades de un solo usuario. La minicomputadora es una computadora muy poderosa capaz de manejar los experimentos científicos, de empresas y universidades grandes.

Aunque la computadora personal era capaz de ejecutar programas y procesar datos sin la intervención de otra computadora, todavía existía la necesidad de tener acceso a otros sistemas de cómputo. Se diseñó un software que permitió que la computadora personal reemplazara a la terminal tonta para conectarse con una computadora anfitriona, ya fuera mediante un módem o por conexión directa. Sin embargo, la computadora personal no resultó una terminal "tonta" y, por lo tanto, fue capaz de reducir los costos generados por la conexión a una computadora anfitriona mediante servicios de tiempo compartido.

La computadora personal servía para introducir los datos, que normalmente se tecleaban en una terminal "tonta" mientras estaba en línea con un servicio de tiempo compartido, lo cual ahorraba los gastos de la conexión a la línea telefónica y a la computadora anfitriona. Después de teclear los datos en la computadora personal y guardados en un archivo, se podía hacer la conexión a la computadora central y transferirle a ésta esos datos con mayor velocidad que si fueran tecleados directamente en una terminal "tonta". La computadora personal también sirvió entonces para capturar los datos que se enviaban de la computadora anfitriona a un archivo (proceso también llamado bajar datos), y que luego podrían utilizarse en la computadora personal o enviarse a otra computadora como se muestra en la figura 1.3.2.1.

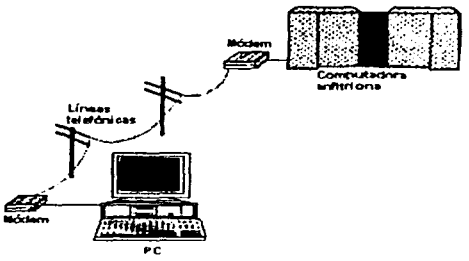


Figura.1.3.2.1. Uso de una PC con un módem y un software de comunicaciones para tener acceso a una computadora anfitriona.

Un módem, es un periférico del ordenador que sirve para permitir la conexión entre dos o más ordenadores a través de una línea telefónica normal. Puesto que las líneas telefónicas no soportan el formato de datos que los ordenadores utilizan para comunicarse entre ellos (por ejemplo, a través de una red), estos datos tienen que ser convertidos a un formato soportado por la línea. De ahí el nombre del aparato, que no es más que el anagrama de las funciones. MOdulador/DEModulador; ya que, no solo tiene que codificar lo que nosotros queremos enviar al ordenador que está al otro lado de la línea, si no que debe decodificar a un formato que nuestro ordenador pueda entender todo lo que recibe a través de la línea telefónica.

1.3.3 La necesidad de la conectividad

A la computadora personal se le añadieron capacidades de almacenamiento adicional y de procesamiento, con lo que cada vez se hizo más pequeña la diferencia entre la computadora personal y las macrocomputadoras. De hecho, en algunos casos esto dio como resultado que la potencia de la computadora personal superara a la de las minicomputadoras o macrocomputadoras que entonces se utilizaban. Además las minicomputadoras y macrocomputadoras tenían un alto costo de adquisición y mantenimiento, mientras que la computadora personal podía comprarse por poco dinero y los costos de mantenimiento eran casi inexistentes.

Se podían comprar y usar varias computadoras personales por mucho menos del costo de un contrato de mantenimiento estándar para una macrocomputadora o minicomputadora. Sin embargo, las computadoras personales eran todavía unas computadoras aisladas, pues la tecnología para conectarlas en red estaba aún por desarrollarse. Aunque las computadoras personales se podían conectar a macrocomputadoras o minicomputadoras, funcionando como terminales "tontas" por medio de un software de comunicaciones, no había forma de que se comunicaran entre ellas.

1.3.4 La red de área local

La *red de área local* (LAN) nació con los beneficios de conectar las computadoras personales o las microcomputadoras (categoría que incluye a las computadoras más pequeñas, que consisten en un microprocesador, elementos de almacenamiento de entrada y salida) con el fin de compartir información. Mucho antes de que fuera considerada factible la idea de que las computadoras personales reemplazaran a las macrocomputadoras o las minicomputadoras, comenzaron a aparecer las primeras redes de área local (*LAN*) de computadoras personales.

Una red de área local (*LAN*) es un sistema de comunicaciones de alta velocidad que conecta microcomputadoras o computadoras personales que se encuentren cercanas, por lo general dentro del mismo edificio. Una red de área local (*LAN*) consta de hardware y software de red y sirve para conectar las computadoras personales que están aisladas. Una red de área local (*LAN*) da la posibilidad de que las computadoras personales compartan entre ellas programas, información y recursos, como unidades de disco, directorios e impresoras.

El proceso de incorporar una computadora personal o microcomputadora a una LAN consiste en la instalación de una *tarjeta de interfaz de red* (NIC) en cada computadora. Las NIC de cada computadora se conectan con un cable especial para red. El último paso para implantar una LAN es cargar en cada computadora personal un software conocido como sistema operativo de red (NOS; *Network Operating System*). El NOS trabaja con el software del sistema operativo de la computadora y permite que el software de aplicación (el procesador de palabra, las bases de datos, las hojas de cálculo, los paquetes de contabilidad, etc.) que se esté ejecutando en la computadora, se comunique a través de la red con otras computadoras; también permite que se configuren los nodos de la red para que ejecuten funciones que se desean. Por ejemplo, el NOS (*Network Operating System*) permite configurar una o más computadoras de la red para que compartan recursos —como las unidades de disco y las impresoras— con otras computadoras. Es posible configurar computadoras para que no tengan la capacidad de compartir sus propios recursos o tener acceso a los recursos que las otras comparten.

1.3.5 La red de área amplia

Las LAN pueden conectarse para formar una red de área amplia (WAN; *Wide Area Network*). Las redes de área local (*LAN*) sirven para conectar a las computadoras personales cercanas, aunque las WAN no están geográficamente limitadas en tamaño. Para conectar a las redes de área local (*LAN*), las WAN suelen necesitar un hardware especial (equipos de puestas de enlace o gateways), así como líneas telefónicas proporcionadas por una compañía telefónica. Las WAN también pueden usar hardware y software especializado para incluir mini y macrocomputadoras como elementos de la red. El hardware para crear una WAN también llega a incluir enlaces de satélite, fibras ópticas, aparatos de rayos infrarrojos y de láser.

La red de computadoras que comprende a Internet está conectada para formar una WAN.

1.3.6 Servidores y estaciones de trabajo

La función de los nodos de la red la determina la manera en que se configura cada uno cuando se instala por primera vez en la red. Al nivel más elemental, un nodo de red puede configurase como servidor o como estación de trabajo. La estación de trabajo es la computadora capaz de aprovechar los recursos de otras computadoras. Una estación de trabajo no comparte sus propios recursos con otras computadoras y, por lo tanto, los demás nodos no pueden usar recurso alguno de ella. En cambio, un servidor es la computadora capaz de compartir sus recursos con otras computadoras. Los recursos pueden incluir impresoras, unidades de disco, unidades de CD-ROM, directorios en disco duro y hasta archivos individuales como se observa en la figura.1.3.6.1.

Hay dos tipos de servidores, los dedicados y los no dedicados. La cantidad y el tipo de servidores de una red depende de la flexibilidad del NOS que se haya seleccionado y de la manera en que se haya escogido la configuración de las computadoras de la red.

- Un *servidor no dedicado* es aquel que opera como estación de trabajo también. Es posible operar un servidor no dedicado y usarlo como estación de trabajo, compartiendo al mismo tiempo sus recursos con otras computadoras.
- Un *servidor dedicado* es un servidor que no puede ejecutar trabajo alguno aparte del requerido para compartir sus recursos con los nodos de la red. A diferencia de los servidores no dedicados, los servidores dedicados no pueden usarse como estaciones de trabajo. Un servidor dedicado maneja por lo general una versión del NOS que optima la velocidad a la que se intercambian los datos entre el servidor y los otros nodos de la red. Puesto que el servidor dedicado se usa solamente para las tareas relacionadas con la red, se elimina sobrecarga adicional (que sería requerida si el servidor también actuara como estación de trabajo) y esto da por resultado un mejor rendimiento.

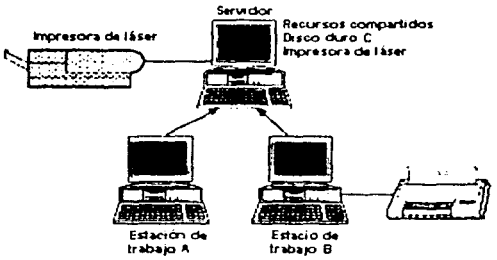


Figura.1.3.6.1. El acceso permitido a los servidores desde las estaciones de trabajo.

1.3.7 Topologías de red

Los nodos de red necesitan estar conectados para comunicarse. A la forma en que están conectados los nodos se le llama *topología*. Una topología tiene dos tipos: uno físico y otro lógico.

La topología física es la disposición física real o actual de la red, la manera en que los nodos están conectados unos con otros. La topología lógica es el método que se usa para comunicarse con los demás nodos, la ruta que toman los datos de la red entre los diferentes nodos de la red. Las topologías física y lógica pueden ser iguales o diferentes.

Las tres topologías estándar son:

- Bus
- Estrella
- Anillo.

También hay combinaciones de más de una topología. Por ejemplo, una topología de árbol es la combinación de una topología de bus y una de estrella, como se muestra en la figura 1.3.7.1.

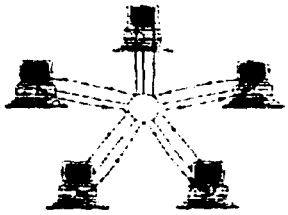


Figura 1.3.7.1. Topología de árbol.

En una *topología de bus*, cada computadora está conectada a un segmento común de cable de red, como se muestra en la figura 1.3.7.2. El segmento de red se coloca como un bus lineal, es decir, un cable largo que va de un extremo a otro de la red, y al cual se conecta cada nodo de la red.

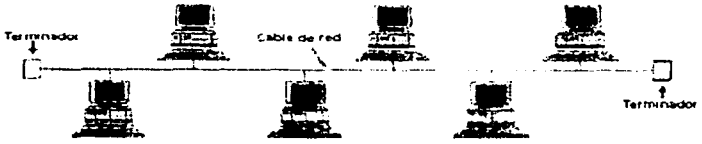


Figura 1.3.7.2. Topología de bus.

En una *topologia de estrella*, cada computadora está conectada a un concentrador ubicado centralmente. El concentrador es un dispositivo con varios puertos, donde se conecta, por medio de un conector, el cable de red; como se observa en la figura 1.3.7.3.

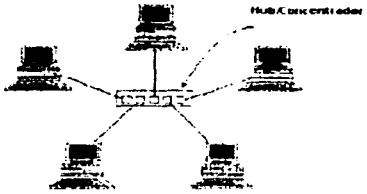


Figura 1.3.7.3. Topologia en estrella

En una *topologia de anillo*, cada computadora se conecta en forma de anillo a la red. Las topologías de anillo casi siempre son lógicas con topología física de estrella. La figura 1.3.7.4 muestra la forma en la que fluyen los datos en una topología de anillo lógica conectada a una topología física de estrella. La topología física muestra que cada computadora se conecta a un dispositivo central y parece una estrella como se observa en la figura 1.3.7.5. La ruta seguida por los datos de una computadora a otra ilustra que la topología lógica es de anillo.

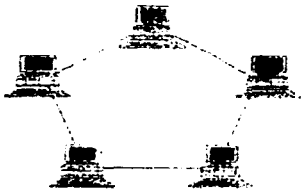


Figura 1.3.7.4. Topología en anillo

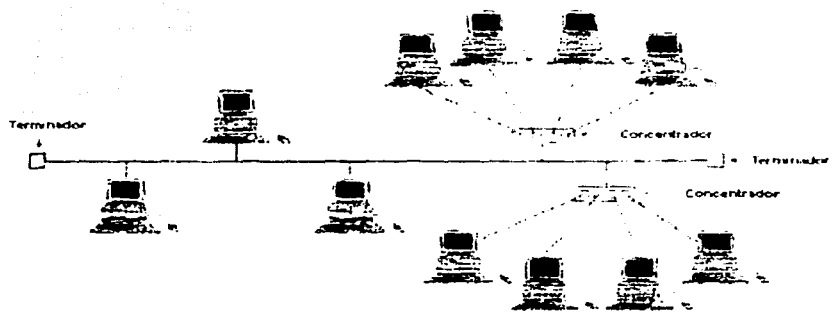


Figura. 1.3.7.5. Topología física de estrella con topología lógica de anillo

1.3.8 Estándares de redes

Las redes están compuestas por muchos componentes físicos diferentes que deben trabajar juntos para crear una red funcional. Los componentes que comprenden las partes de hardware de la red incluyen tarjetas adaptadoras de red, cables, conectores, concentradores y computadoras.

Se han creado estándares que definen la forma de conectar componentes de hardware en las redes y protocolos de uso cuando se establecen comunicaciones por red. Todos los datos que fluyen por el cable de red deben ir en secuencia y distinguirse, para que los diversos nodos puedan asegurarse de que los datos debidos lleguen al lugar pretendido.

Un protocolo es un juego de reglas que definen la forma en que deben efectuarse las comunicaciones de las redes, incluyendo el formato, la temporización, la secuencia y la revisión y la corrección de errores.

Un estándar son las especificaciónes de la red adoptadas e incluyen guías y reglas que se refieren al tipo de componentes que deben usarse, a la manera de conectar los componentes, así como a los protocolos de comunicación que hay que emplear.

Los tres estándares más populares que se utilizan en las redes son: Ethernet, ARCnet y Token Ring. Ethernet y Token Ring son estándares respaldados por el Institute of Electrical and Electronic Engineers (IEEE). ARCnet es un estandar de la industria que ha llegado a ser recientemente uno de los estándares del Instituto National de Estándares Americanos (ANSI).

La gran variedad de arquitecturas, protocolos y modos de conexión y comunicaciones existentes, obligaron a la industria de las comunicaciones a estandarizar mediante un grupo de normas internacionales conocido como el modelo OSI (*Open Systems Interconection*),

puesto en operación por la Organización Internacional de Estándares (*International Standars Organization*, ISO), que consiste en la implantación de los protocolos de comunicación entre computadoras en siete capas, cada una con funciones bien definidas, pero relacionadas entre sí. La figura 1.3.8.1 muestra un diagrama de las siete capas del Modelo OSI.

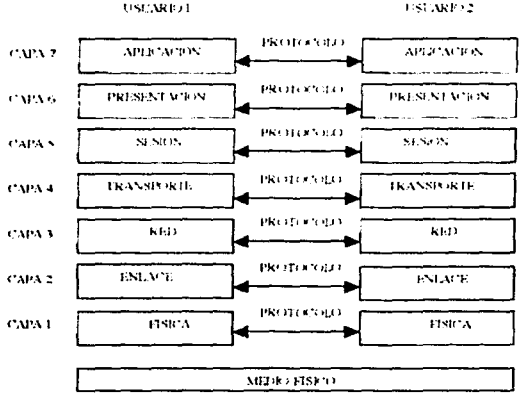


Figura 1.3.8.1 Diagrama de capas del Modelo OSI

Las capas funcionan como una pirámide; es decir, las más altas están basadas en las funciones de sus antecesoras. Grosso modo, como puede verse en la tabla 1.3.8.2, la función específica de cada capa es:

Capa 7	Define las reglas de comunicación entre aplicaciones y la red (correo electrónico, transferencia de archivos, bases de datos entre otros).
Capa 6	Organiza la transferencia de datos entre sistemas para representarlos y codificarlos de una manera uniforme.
Capa 5	Se encarga de la coordinación de las comunicaciones en forma ordenada y del control de la sesión.
Capa 4	Verifica la validez de la integridad de la transmisión utilizando algoritmos de corrección de errores.
Capa 3	Define la ruta entre las unidades de emisión y recepción de datos, haciendo las veces de conmutador.
Capa 2	Se responsabiliza de la integridad de la transmisión de datos entre nodos.
Capa 1	Configura las características físicas necesarias para la transmisión y recepción de datos en forma de bits.

Tabla 1.3.8.2 Descripción de las capas del Modelo OSI

1.4 TCP/IP (*Transmission Control Protocol*)/(*Internet Protocol*) EL PROTOCOLO DE INTERNET

Es un protocolo de comunicaciones basado en software usado en redes. Aunque el nombre TCP/IP implica que el alcance completo del producto es una combinación de dos protocolos —Protocolo de Control de Transmisión y Protocolo Internet—, el término TCP/IP no se refiere a una entidad única que combina dos protocolos, sino a un conjunto más grande de programas de software que proporciona servicios de red como registros remotos, transferencias de archivos remotas y correo electrónico. El TCP/IP proporciona un método para transferir información de una máquina a otra. Un protocolo de comunicaciones debe manejar los errores en la transmisión, administrar el enrutamiento y envío de datos y controlar la transmisión real por medio del uso de señales de estado predeterminadas. TCP/IP logra todo esto.

La figura1.4.1 muestra los elementos básicos de la familia de protocolos TCP/IP y en la tabla 1.4.1 se muestra una pequeña descripción de los mismos. Como se observa, TCP/IP no está comprendido en las dos capas inferiores del modelo OSI (vinculación de datos y física), sino que comienza en la capa de red, donde reside el protocolo Internet (IP). En la capa de transporte intervienen el TCP (*Transmission Control Protocol*)) y el UDP (*User Datagrama Protocol*). Las utilerías y protocolos que forman el resto del conjunto TCP/IP se crean encima de éstos al usar las capas TCP o UDP e IP para su sistema de comunicaciones.

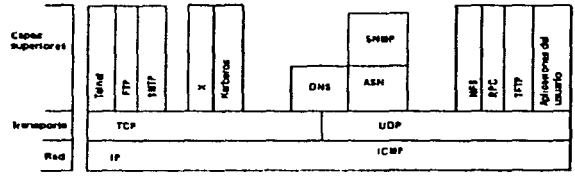


Figura 1.4.1 Las capas OSI.

Telnet	Registro remoto
FTP	Protocolo de transferencia de archivos
SMT	Protocolo simple de transferencia de correo
X	Sistema X Windows
Kerberos	Seguridad
DNS	Sistema de nombre de dominio
ASN	Notación de sintaxis abstracta
SNMP	Protocolo simple de administración de redes
NFS	Servidor de archivos de red
RCP	Llamadas de procedimiento remoto
TFTP	Protocolo trivial de transferencia de archivos
TCP	Protocolo de control de transmisiones
UDP	Protocolo de datagrama de usuario
IP	Protocolo Internet

Tabla 1.4.1. La colección TCP/IP

La figura 1.4.1 muestra que algunos de los protocolos de la capa superior dependen del TCP (como Telnet y FTP), mientras que algunos dependen del UDP (*User Datagrama Protocol*) (como TFTP y RCP). La mayor parte de los protocolos TCP/IP de las capas superiores usan sólo uno de los dos protocolos de transporte (TCP o UDP), aunque unos cuantos, incluyendo DNS (Domain Name Service), puede usar ambos.

TCP/IP es dependiente del concepto de cliente y servidor. Esto no tiene que ver con un servidor de archivos al que se tiene acceso mediante una estación de trabajo o PC sin disco. El término *cliente servidor* tiene un significado sencillo en TCP/IP: cualquier dispositivo que inicie comunicaciones es el cliente y el dispositivo que responda es el servidor. El servidor está respondiendo a las solicitudes del cliente.

1.4.1 Una síntesis rápida de los componentes TCP/IP

Para entender los papeles de los muchos componentes de la familia de protocolos TCP/IP es útil saber lo que se puede hacer en una red TCP/IP. La siguiente lista no es exclusiva, pero menciona las aplicaciones de usuario primarias que proporciona el TCP/IP.

Telnet

El programa Telnet proporciona una capacidad de registro remoto. Esto permite al usuario de una máquina registrarse en otra y actuar como si estuviera directamente enfrente de la segunda máquina. La conexión puede estar en cualquier parte de la red local o en otra red en cualquier parte del mundo, siempre que el usuario tenga autorización para registrarse en el sistema remoto.

Telnet se puede usar cuando se necesitan ejecutar acciones en una máquina a través del país. Esto no se hace con frecuencia, excepto en un contexto de LAN o WAN, pero unos cuantos sistemas accesibles por medio de Internet permite sesiones Telnet, mientras los usuarios juguetean con una aplicación o sistemas operativos nuevos.

FTP *(File Transfer Protocol)*

El File Transfer Protocol (FTP) permite a un usuario copiar un archivo de un sistema a otro sistema. En realidad el usuario no se registra como un usuario completo en la máquina a la que desea tener acceso, como con telnet, en su lugar usa el programa FTP para permitir el acceso. Una vez más son necesarias las autorizaciones correctas para proporcionar acceso a los archivos.

Una vez que se ha establecido la conexión con una máquina remota, FTP permite copiar uno o más archivos a la máquina de uno. El término *transferir* implica que el archivo se mueve de un sistema a otro, pero el original no se afecta. Los archivos se copian. FTP es un servicio usado en forma amplia en Internet, así como en muchas LAN y WAN grandes.

SMTP *(Simple Mail Transfer Protocol)*

El Simple Mail Transfer Protocol (SMTP) se usa para transferir correo electrónico. SMTP es transparente por completo para el usuario. SMTP se conecta con máquinas remotas y transfiere mensajes de correo de modo parecido a como FTP transfiere archivos. Los usuarios casi nunca se dan cuenta del trabajo de SMTP y pocos administradores de sistemas tienen que preocuparse de él. SMTP sobre todo, es un protocolo libre de problemas y muy usado.

Kerberos

Kerberos es un protocolo de seguridad soportado en forma muy amplia. Kerberos usa una aplicación especial llamada servidor de autentificación para validar las contraseñas y los esquemas de encriptación. Kerberos es uno de los sistemas de encriptación más seguros que se usan en comunicaciones y es muy común en UNIX.

DNS *(Domain Name Service)*

Domain Name Services (DNS) permite a una computadora con un nombre común convertirse en una dirección de red especial. Por ejemplo, otra máquina no puede tener acceso a una PC llamada Darkstar en la misma red (o en cualquier otra red conectada), a menos que esté disponible algún método que revise el nombre de la máquina local y reemplace el nombre con la dirección del hardware de la máquina. DNS proporciona una conversión del nombre local común a la dirección física única, de la conexión de red del dispositivo.

Simple Network Management Protocol

Simple Network Protocol (SNMP) proporciona mensajes de estado y reporta problemas a través de una red hacia un administrador. SNMP usa el User Datagrama Protocol (UDP) como un mecanismo de transporte. SNMP emplea términos ligeramente diferentes del TCP/IP, al trabajar con administradores y agentes en lugar de clientes y servidores (aunque en esencia significa lo mismo). Un agente proporciona información sobre un dispositivo, mientras que un administrador se comunica a través de la red con los agentes.

Network File System

Network File System (NFS) es un conjunto de protocolos desarrollados por Sun Microsystems para permitir a múltiples máquinas tener acceso a los directorios de cada una de las otras, de manera transparente. Logran esto al usar un esquema de sistema de archivo distribuido. Los sistemas NFS son comunes en ambientes corporativos grandes, en especial aquellos que usan estaciones de trabajo UNIX.

Remote Procedure Call

El protocolo Remote Procedure Call (RCP) es un conjunto de funciones que permite a una aplicación comunicarse con otra máquina (el servidor). Atiende funciones de programación, código remoto y variables predefinidas para soportar la computación distribuida.

Trivial File Transfer Protocol

Trivial File Transfer protocol (TFTP) es un protocolo de transferencia de archivos muy sencillo que carece de seguridad. Usa el UDP como transporte. TFTP ejecuta las mismas tareas que el FTP, pero usa un protocolo de transporte diferente.

Transmission Control Protocol

Transmission Control Protocol (la parte TCP del TCP/IP) es un protocolo de comunicaciones que proporciona una transferecia confiable de datos. Es responsable de ensamblar los datos pasados de aplicaciones de capas superiores hacia paquetes estándar y asegurar que los datos se transfieren en forma correcta.

User Datagram Protocol

User Datagram Protocol (UDP) es un protocolo orientado hacia la ausencia de conexión, lo que significa que no entiende la retransmisión de datagramas (unidad básica de información utilizada con TCP/IP) a diferencia del TCP, el cual está orientado hacia la conexión. UDP no es muy confiable pero tiene propósitos especializados. Si las aplicaciones que usan el UDP tienen incorporada la revisión de la confiabilidad se superan los defectos del UDP.

Internet Protocol

Internet Protocol (IP) es responsable de mover los paquetes de datos ensamblados, ya sea por el TCP o el UDP a través de las redes. Usa un conjunto de direcciones únicas para cada dispositivo en la red, a fin de determinar el enrutamiento y los destinos.

Internet Control Message Protocol

Internet Control Message Protocol (ICMP) es responsable de revisar y generar mensajes sobre el estado de dispositivos en una red. Puede usarse para informar a otros dispositivos de una falla en una máquina particular.

Por lo general, ICMP e IP funcionan juntos.

1.4.2 Protocolo Internet

El protocolo internet (IP) es un protocolo primario del modelo OSI, así como una parte integral del TCP/IP (como sugiere el nombre). Aunque la palabra "internet" aparece en el nombre del protocolo, no está restringido para uso con Internet. Es cierto que todas las máquinas en Internet pueden usar o entender el IP, pero el IP también se puede usar en redes dedicadas que no tienen relación en absoluto con internet. El IP define un protocolo, no una conexión. En efecto, el IP es una elección muy buena para cualquier red que necesite un protocolo eficiente para comunicaciones máquina a máquina.

Sus tareas principales son direccionar los datagramas (unidad básica de información utilizada con TCP/IP) de información entre computadoras y manejar el proceso de fragmentación de estos datagramas. El IP es responsable del enrutamiento de un datagrama, determinando a dónde será enviado y concibiendo una ruta alternativa en caso de problemas.

El IP no tiene nada que ver con el control o la confiabilidad del flujo: no tiene capacidad inherente para verificar que un mensaje enviado se reciba en forma correcta. El IP no tiene una suma de verificación para el contenido de datos de un datagrama, sólo para la información del encabezado. Las tareas de verificación y control del flujo se dejan a otros componentes en el modelo de capas. El IP puede hacer una suposición de cuál es la mejor ruta para mover un datagrama al siguiente nodo a lo largo de una ruta, pero no verifica de manera inherente que la ruta elegida sea la más rápida o la más eficiente. Parte del sistema IP define cómo manejan los gateways (equipos de compuerta de enlace) los datagramas, cómo y cuándo deben producir mensajes de error y cómo recuperarse de problemas que podrían surgir.

El IP es sin conexión, lo que significa que no se preocupa por los nodos por donde pasa un datagrama a lo largo de la ruta o incluso, en cuáles máquinas empieza y termina el datagrama.

1.4.3 Protocolo de Control de Transmisión (TCP)

El protocolo de Control de Transmisión proporciona una cantidad considerable de servicios a la capa IP y a las capas superiores. De mayor importancia, proporciona un protocolo orientado hacia la conexión para las capas superiores, que permite a una aplicación estar segura de que un datagrama enviado por la red fue recibido integro. En este papel, el TCP actúa como un protocolo de validación del mensaje que proporciona comunicaciones confiables. Si un datagrama se altera o pierde, por lo general el TCP maneja la retransmisión, en lugar de las aplicaciones de las capas superiores.

El TCP maneja el flujo de datagramas desde las capas superiores hasta la capa IP, así como los datagramas que llegan desde la capa IP hacia los protocolos de niveles superiores. El TCP tiene que asegurar que las prioridades y la seguridad se respeten de manera apropiada. El TCP debe ser capaz de manejar la terminación de una aplicación superior, que estaba esperando la llegada de datagramas, al igual que las fallas en las capas inferiores. El TCP también debe mantener una tabla de estado de todas las series de datos que entran y salen de la capa TCP. El aislamiento de todos estos servicios en una capa separada permite que las aplicaciones se diseñen sin importar el control del flujo o la confiabilidad del mensaje. Sin la capa TCP, cada aplicación tendría que aplicar los servicios por sí misma, lo cual es un desperdicio de recursos.

Debido a que el TCP es un protocolo orientado hacia la conexión responsable para asegurar la transferencia de un datagrama, desde la máquina fuente hasta la máquina de destino (comunicaciones de extremo a extremo), el TCP debe recibir mensajes de comunicaciones desde la máquina de destino para acusar recibo del datagrama. El término circuito virtual por lo general se usa para referirse a las comunicaciones entre las dos máquinas en los extremos, la mayor parte de las cuales son sólo mensajes de acuse de recibo (ya sea de confirmación del recibo o de una falla en el código) y números de secuencia de datagrama.

Beneficios del TCP/IP

TCP/IP se ha convertido en el estándar de Internet y de las redes en general debido a los beneficios que ofrece:

- Es independiente del hardware subyacente y se ejecuta en Ethernet y Token Ring, e incluso en las redes telefónicas para el acceso telefónico a redes.
- Es independiente del tipo de sistema informático utilizado. Transmite datos entre cualquier ordenador, desde pequeños PC hasta grandes mainframes.
- Es dirigible, por lo que las diferentes partes de un mensaje pueden enviarse por caminos específicos, reduciendo el tránsito en otras partes de la red.
- Es fiable, eficiente y funciona con poca sobrecarga de datos.
- Es un estándar abierto, totalmente disponible y que es totalmente independiente de cualquier fabricante de software o hardware.

1.5 Internet

La red *Internet* es el resultado de comunicar miles de redes de computadoras entre sí. Permite conectar diferentes tipos de redes, que pueden ser de área local o de área extensa, utilizando protocolos como TCP-IP, que identifican los datos aunque procedan de diferentes tipos de equipos (PC's, Macintosh, Amiga) y usen sistemas operativos anteriormente incompatibles como UNIX, MS-DOS, OS/2, System 7, XENIX, etc.

Usando una PC o una terminal en el hogar, en la escuela o en el trabajo, es posible tener acceso a cientos de miles de computadoras alrededor de todo el mundo.

1.5.1 Historia de Internet

La red *Internet* tiene su origen en una antigua red de comunicaciones desarrollada por la Agencia de Proyectos Avanzados de Investigación (ARPA), del Ministerio de Defensa de Estados Unidos, la cual puso en marcha en 1969 un sistema de comunicaciones conocido como ARPANET y restringido al uso interno del Ministerio. En la década de los 70 y principio de los 80 otros países comenzaron a desarrollar sus propias redes de comunicación, como la red Teletel/Minitel en Francia, o Eunet en toda Europa.

ARPANET fue una colección de computadoras que interconectaban muchos servidores de terminales. La preocupación era que una guerra nuclear pudiera cortar totalmente las comunicaciones, así que las vías para conectar redes tenían que ser flexibles. Los creadores de este sistema tuvieron el cuidado de desarrollar reglas voluntarias que cubrieran todos los aspectos de este sistema. Se hicieron estándares para la creación de direcciones y para los protocolos de comunicaciones. Esta idea incluye enviar mensajes *empaquetados* (packet) en una especie de envoltura. El mensaje es puesto en un paquete IP (Internet Protocol) y es enviado por la computadora fuente. La computadora fuente es responsable de asegurarse que

el mensaje llegue a su destino. La red no tiene esta responsabilidad. Si alguna ruta no está disponible, se puede seleccionar otra.

Ligada a ARPANET se creó en la década de los 80 la red de la *National Science Foundation* (NSFNET) red que unía las principales instituciones científicas de los Estados Unidos mediante cinco grandes superordenadores. En 1990 la red ARPANET dejó de existir, creándose en 1991 la *Commercial Internet exchange Association*, que se hizo cargo de la administración de lo que fue ARPANET y ya es internet. El organismo que rige hoy la red Internet, la Internet Society, aparece en 1992, año en el que el Centro Europeo de Investigación Nuclear (CERN) puso en marcha la World Wide Web (WWW), también conocida como «telaraña mundial», que supuso una auténtica revolución en el mundo de la comunicación por Red. A partir de 1994 la red Internet se convirtió en lo que hoy día conocemos: una red mundial para compartir información en tiempo real.

1.5.2 Historia del Web

El World Wide Web (WWW) es un sistema distribuido de información basado en el concepto de hipertexto. Los documentos guardados en el sistema Web pueden ser de varios tipos. El tipo más habitual son los documentos de hipertexto con formato conforme con el lenguaje de marcas de hipertexto (*HyperText Markup Language*, HTML), que está basado a su vez en el lenguaje normalizado de marcas generalizado (*Standard Generalized Markup Language*, SGML). Los documentos HTML contienen texto, especificaciones de fuentes y otras instrucciones de formato. Los enlaces con otros documentos (sean locales o remotos) pueden asociarse con zonas del texto. También se puede hacer referencia a las imágenes con las instrucciones de formato adecuadas (como los enlaces a los archivos de imágenes).

Los usuarios de los sistemas Web ven texto con formato junto con imágenes, en vez del texto sin formato con instrucciones de formato. El texto con formato con las imágenes es mucho más atractivo visualmente que el texto únicamente. Además, los documentos mostrados son documentos de hipertexto: con un explorador adecuado los usuarios pueden pulsar en las zonas que tengan enlaces asociados y se mostrarán los documentos a los que apunten esos enlaces. Por tanto, la interfaz de hipertexto con el sistema Web forma una interfaz para exploración poderosa y visualmente atractiva. Más recientemente, los exploradores Web han comenzado a incluir un lenguaje de programación denominado *Java* que permite la existencia de documentos que contengan programas que se ejecuten en la computadora del usuario. Por tanto, los documentos ahora pueden *ser activos*, y no meramente *pasivos*.

1.5.3 Localizadores universales de recursos

Los sistemas de hipertexto deben poder guardar los punteros a los documentos. En el sistema Web, la funcionalidad de los punteros la proporciona los Localizadores Universales de Recursos (*Universal Resource Locator*, URLs). A continuación se ofrece un ejemplo de URL:

http://www.bell-labs.com/topic/book/db-book

La primera parte del URL indica la manera en que hay que generalizar el acceso al documento: «http» indica que se tiene acceso al documento utilizando el protocolo para transferencia de hipertexto (*HyperText Transfer protocol*, HTTP), que es un protocolo para transferencia de documentos HTML. La segunda parte indica un nombre único en Internet de la máquina. El resto del URL es el nombre del camino hasta el archivo en la máquina.

Por lo tanto, los URL proporcionan un nombre global único para cada documento al que se puede tener acceso desde los sistemas Web. Dado que los URL son legibles por las personas, éstas pueden utilizarlos directamente para tener acceso a los documentos deseados, en lugar de desplazarse por un camino desde una ubicación predefinida. En el sistema Web, cualquier usuario puede crear documentos y puede facilitar los URL a cualquier otro. De hecho, como consecuencia, el sistema Web es casi anárquico, no hay una autoridad central que controle el sistema Web. Muchos usuarios actuales de Internet disponen de *páginas iniciales* en el sistema Web; estas páginas suelen contener información sobre la vida personal y profesional de los usuarios.

1.5.4 Servidores Web

HTTP proporciona potentes prestaciones, más allá de la mera transferencia de los documentos, como el cifrado de seguridad. El nombre del documento del URL puede identificar un programa ejecutable que, al ejecutarse, genere un documento de este tipo, ejecuta el programa y devuelve el documento HTML que se genera. Además, el cliente Web puede transferir otros argumentos con el nombre del documento que se transferirán como argumentos cuando el programa se ejecute. Por tanto, el documento generado depende de los argumentos transferidos.

Como consecuencia de estas prestaciones, los servidores Web pueden servir sin dificultades como interfaz para gran variedad de servicios de información. Para instalar un nuevo servicio en el sistema Web los usuarios sólo necesitan crear e instalar un programa ejecutable que proporcione este servicio.

Como se puede ver, un simple clic en un hipervínculo inicia una muy importante serie de acciones donde participa no sólo su software de navegador Web, sino también un servidor Web en alguna parte de Internet. En la figura 1.5.4.1 se muestra esta secuencia de sucesos.

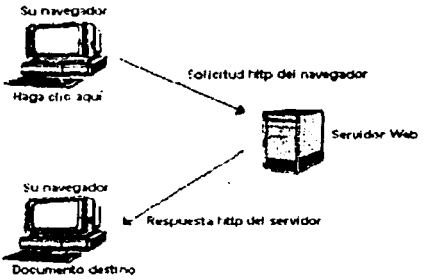


Figura 1.5.4.1 Comunicación de navegador/servidor Web por medio de HTTP.

1.5.5 Lenguajes de visualización

Los sistemas informáticos actuales no proporcionan sólo una interfaz sencilla basada en los caracteres, sino que también permiten gráficos, varias fuentes, color, etc., cuyas opciones pueden mejorar en gran medida la calidad de la información mostrada. Sin embargo, hay gran variedad de normas para la visualización gráfica y los diferentes tipos de computadoras tienen interfaces diferentes. Es difícil escribir programas que sean directamente compatibles con varias interfaces de visualización.

Los lenguajes de marcas de texto, como el lenguaje normalizado de marcas generalizado (*Standard Generalized Markup Language, SGML*), se desarrollaron para cubrir el hueco existente entre el texto sin formato y los lenguajes para la descripción de las páginas o para el formato del texto. Los lenguajes de marca proporcionan órdenes de formato sencillas, como los separadores de párrafos y las listas numeradas o con viñetas, así como cierto grado de control sobre las fuentes que aparecen en la pantalla. Por tanto, proporcionan una interfaz de salida normalizada para los programas.

Los lenguajes de marca de texto resultan especialmente importantes para los sistemas de información distribuidos de propósito general, dado que la información que debe mostrarse debe tener un formato agradable y no ser meramente texto sin formato. Una manera alternativa de implantar este formato es que cada servicio de información ejecute un programa de propósito específico en la computadora cliente, lo cual resulta obviamente inconveniente para los usuarios que desean tener acceso a muchos servicios de información.

1.5.6 Lenguaje de marca de hipertexto

El SGML proporciona una gramática para la especificación de los formatos de los documentos basada en anotaciones de marcas normalizadas. HTML es un lenguaje de visualización de hipertexto de propósito general basado en SGML. Especifica un formato de

los documentos que permite órdenes para el formato del texto, así como órdenes para enlaces de hipertexto y órdenes para la exhibición de imágenes.

El HTML consiste propiamente en un archivo de texto con códigos que especifican en cada parte de él, si se trata de *texto, gráficos o sonido*. Además se resaltan palabras o partes del texto para desde ahí, realizar "saltos" hacia otra parte del mismo archivo, hacia otra página o incluso hasta la página del Web de otra computadora remota. Algunos de estos enlaces de hipertexto pueden activar un sonido o voz, un video o mostrar una imagen, en un proceso conocido como *hipermedia*.

El principio del HyperText Markup Languaje fue un gran avance, pero no significaba más que poner al alcance del cliente, los servicios de texto y gráficas sobre una interfaz tipo terminal o sea en modo texto. Nunca se consideró la posibilidad de sacar todo el provecho a la página del WWW hasta que el servicio se popularizó en las universidades y centros de investigación, en donde se desarrollaron los primeros programas de *navegación en el Web*.

Para poder visualizar las páginas Web y navegar dentro de la Red, es necesario disponer de un programa cliente denominado navegador, el cual se conecta al servidor Web (ordenador de nuestro proveedor de acceso a Internet) con el fin de recibir la información contenida en las páginas Web; es decir dicho programa lee el contenido de esas páginas Web y las presenta en la pantalla del usuario que lo solicitó.

En el Centro Nacional de Aplicaciones de Supercómputo (National Center for Supercomputing Applications, NCSA) de la Universidad de Illinois, Marc Andreessen trabajó a principios de 1993 en un proyecto cuyo propósito era leer las páginas del Web que estaban en formato HTML, pero no en modo texto, sino en forma gráfica y con capacidades de hipermedia. El producto fue MOSAIC. Por las siglas de este centro de investigación, se conoció a esta primera versión como *NCSA Mosaic*. A estose tipos de programas se les conoce como visualizadores (Browsers) u hojeadores del Web, ya que a las interfaces de los clientes y servidores de estos servicios se les conoce como páginas del Web.

Con el paso del tiempo, la red *Internet* se va haciendo más difícil de definir. Apenas hace unos años, la red *Internet* era como todas las redes de computadoras que usaban el protocolo IP. Hoy, muchas redes que no usan el protocolo IP se pueden conectar a las redes IP usando lo que llamamos *Gateways* o puentes. El crecimiento tan acelerado de la red sobrepasó rápidamente todos los pronósticos, convirtiéndose a la fecha en la red de redes. Desde 1993 Internet deja de ser la red de instituciones gubernamentales y universidades para convertirse en la red pública más grande del mundo. Han proliferado los servicios de conexión como Prodigy, CompuServe y America Online en Estados Unidos; Internet de México, y Datanet en México y algunos más en otros países.

Hay infinidad de maneras de ingresar a la gran red. Las universidades y tecnológicos proporcionan una clave a sus investigadores para que puedan estar al tanto de los avances de la tecnología de los más importantes centros de estudios del mundo.
En caso de no tener alguna de las facilidades mencionadas, puede contratarse un servicio de conexión con alguna compañía dedicada a las comunicaciones.

1.5.7 Tipos de conexiones en Internet

Hay cuatro formas básicas de conectarse a Internet y unas cuantas variaciones de éstas:

1. *Conexión permanente o dedicada*

Una *conexión permanente* significa que se trata de una computadora conectada directamente a una red TCP/IP (Transmission Control Protocol/Internet Protocol) que forma parte de Internet. A este tipo de conexion recurren por lo general organizaciones grandes; por ejemplo, universidades, grupos de escuelas y compañías. El proveedor de servicios pone un ruteador o encaminador en una oficina de la organización y contrata una línea telefónica que sirva para conectar ese ruteador con la computadora del proveedor de servicios (la computadora *anfitriona*). Ya que se tiene una línea alquilada, se encuentran permanentemente conectados con Internet. No hay que llamar por teléfono cada vez que se quiera tener acceso a la computadora del proveedor de servicios. Este tipo de servicio es muy caro.

2. *Conexiones directas por conmutación telefónica*

Las *conexiones directas por conmutación telefónica* se conocen generalmente como *SLIP* (Serial Line Internet Protocol), *CSLIP* (SLIP comprimido) o conexión *PPP* (Point-to-Point Protocol).

Se trata de un servicio de "conmutación telefónica", por lo que necesitará instalar un módem en su computadora y marcar el número de teléfono que le haya proporcionado el proveedor de servicios. Una vez conectado a la computadora de éste e iniciada la sesión, aparte de la velocidad, no va a notar ninguna diferencia entre una cuenta SLIP y una cuenta dedicada (se podrán transferir archivos hacia su computadora y de esta hacia otras, exactamente como si se tratara de una computadora anfitriona), de hecho, se le identificara en la red en calidad de anfitriona, como se observa en la figura 1.5.7.1.

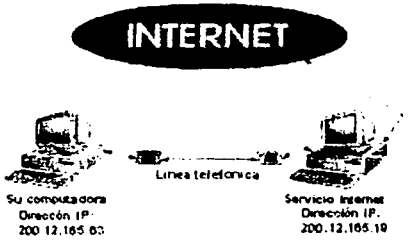


Figura 1.5.7.1. Una de las maneras mas sencillas y costeables de entrar a Internet es a través de una conexión SLIP o PPP

3. *Conexión de terminal por conmutación telefónica*

En este tipo de conexión se tiene que llamar por teléfono a la computadora del proveedor de servicios. Y, una vez conectado, la computadora funcionará como si fuera una terminal.

Este arreglo difiere de la conexión permanente o la directa por conmutación telefónica, porque, la computadora que hace la llamada telefónica, no aparecerá como anfitriona de la red; simplemente será una terminal de la computadora del proveedor de servicios. Todos los programas que se ejecuten se realizarán en la computadora del proveedor de servicios. Esto implica que podrá transferir archivos a través de Internet desde y hacia la computadora del *proveedor de servicios*, pero no directamente de la computadora.

4. *Conexiones de correo*

Internet cuenta con varias conexiones de correo. Si, por ejemplo, tiene cuenta con CompuServe o America Online, entonces ya dispone de una conexión de correo con Internet. Otra forma de conexión de correo electrónico es aquella en la que uno se conecta a Internet de la misma manera que en una terminal por conmutación telefónica directa pero sólo se permite entrar al sistema de correo. Finalmente, lo que tendrá que hacer es conseguir una conexión UUCP. Se trata de una conexión de correo que utiliza software destinado a este propósito (en vez de un programa de comunicaciones para toda finalidad). Lo único que podrá hacer es mandar y recibir correo y mensajes del grupo de noticias

1.5.8 Servicios que permite Internet

Cuando un usuario contrata con un proveedor de acceso a Internet una cuenta de Internet, lo que está haciendo es contratar una conexión a la Red a través de su línea telefónica y además contratar una serie de servicios, entre los que destacan los siguientes que se muestran en la tabla 1.5.8.1.

Servicio	¿Qué hace?
Correo electrónico	Envía mensajes y archivos asociados entre usuarios.
Servicio de acceso remoto	Permite que los usuarios tengan acceso a los ordenadores de la red desde puestos remotos.
Grupos de noticias	En Internet existen las noticias USENET, más de 10,000 "tablones de anuncios" sobre temas de discusión e intercambio de información.
Listas de distribución	A diferencia de los grupos de noticias, a los que uno debe acudir, una lista de distribución envía cada uno de los mensajes a cada uno de los suscriptores
Transferencia de archivos	Envía archivos de un ordenador a otro.
Conferencia por voz y video	Utilizando hardware adicional, los usuarios pueden mantener conferencias en tiempo real incluso con los que están muy, muy lejos.

Tabla 1.5.8.1. Servicios de Internet

1.6 Intranet

Se utiliza el término *intranet* para referirse a la manera en que una organización aprovecha World Wide Web y la tecnología relacionada con Internet para llevar a cabo su trabajo esencial: el de ayudar a producir los bienes o servicios para los cuales está destinada la organización.

En la fiebre por entrar a Web, la mayoría de las organizaciones piensa en términos de poner alguna información disponible para personas ajenas a la organización. Muchas compañías han instalado servidores Web y los hacen accesibles a través de Internet con la idea de poner información corporativa a disposición de otros o vender cosas en Web. Sin embargo, es interesante saber que el objetivo inicial de los pioneros de Web en el CERN, en Ginebra, era crear un medio para que los científicos del CERN pudieran compartir información con mayor facilidad. De este modo, la primera Web fue, de hecho, una *intranet*, que se diseñó con el objetivo de distribuir información dentro de una organización, para su propio personal. Sin restar importancia al probado valor para los negocios que tienen los servicios de Word Wide Web al poner la información a disposición de compañías y organizaciones externas.

1.6.1 Modelos organizacionales

En épocas anteriores, cuando los departamentos de manejo de sistemas de información tenían un monopolio de la información y procesamiento de datos, hubiera sido fácil asignar la responsabilidad de la configuración y diseño de una intranet. En la actualidad, Web está basada en la computación distribuida, y el departamento no puede controlar la PC o la estación de trabajo de cada uno.

Con base en el enfoque filosófico que se elija en la asignación de responsabilidades para la *intranet*, hay varios modelos que se pueden seguir. Aquí hay tres:

- *Modelo centralizado* con un solo servidor Web administrado por una organización específica de la compañía y un proceso formal para el desarrollo e instalación de nuevos servicios.
- *Modelo descentralizado* donde todos son libres de configurar un servidor Web y colocar recursos de su elección en él.
- *Modelos mixtos* con elementos de los modelos anteriores.

1.6.2 Seguridad en la intranet

Aunque existen múltiples opciones de seguridad para las intranet, sólo existen dos principios fundamentales sobre la seguridad en una intranet:

- Primero, *mantener alejados a los extraños.*
- Segundo, *restringir el acceso de los trabajadores internos a la información pública* y *limitar el acceso a toda información importante.*

La instalación de una seguridad confiable exige la planificación, el mantenimiento y la comprobación efectiva. La seguridad fiable nunca se basa en una solución con un único nivel, ni en una en la que todos los usuarios tengan el mismo nivel de autoridad. La seguridad real debe instalarse a lo largo de varias capas de un sistema y tener diferentes niveles de acceso para los usuarios. El esquema de seguridad más habitual y efectivo es el modelo de círculo concéntrico mostrado en la figura 1.6.2.1. La fuerza de este esquema reside en sus sucesivas capas, con una protección cada vez mayor cuando los usuarios intentan acceder a un sistema deben presentar sus permisos para atravesar las capas de seguridad adicionales, cada vez más restrictivas. Como una cebolla, los usuarios deben pasar por cada capa de seguridad para tener acceso a la siguiente. En las capas más profundas del sistema las restricciones son más estrictas y la obtención de la autorización de acceso a cada nivel se vuelve cada vez más difícil. En un entorno seguro y efectivo, un usuario debe atravesar con éxito tres o más capas antes de poder entrar a cualquier archivo o aplicación.

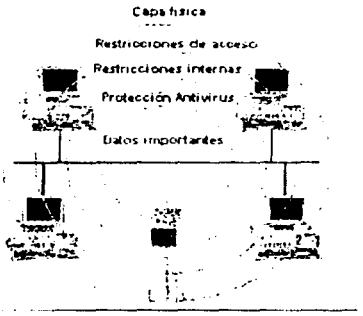


Figura 1.6.2.1. El modelo de seguridad de círculo concentrico

Restricciones físicas

El nivel de protección inicial se centra en la restricción de acceso a los dispositivos físicos (servidores y estaciones de trabajo) que configuran la red. Los componentes que suelen encontrarse en la capa de seguridad física son: edificios cerrados, cámaras de seguridad, guardias de seguridad, candados en los ordenadores, entre otros.

Restricciones de acceso

La siguiente capa de seguridad restringe los privilegios de acceso que se aplican tanto a los ordenadores localizados en el lugar de trabajo como al acceso desde el exterior mediante líneas de telecomunicaciones o conexiones de Internet. La restricción de acceso es la función de seguridad que requiere una contraseña y un nombre de usuario válido para permitir el acceso a un usuario.

Restricciones internas

Hay dos esquemas básicos de restricción interna: basado en objetos/recursos y basado en usuario. El método basado en objeto/recurso implica la configuración de restricciones de acceso en todos los objetos, archivos, directorios, ordenadores y periféricos para cada uno de los usuarios. No se trata sólo de una tarea larga, tediosa y difícil, si no que es también un método de seguridad bastante pobre para las grandes redes, ya que se necesita un nombre de usuario y una contraseña cada vez que se tiene acceso a un objeto o a un recurso.

Protección antivirus

La última capa de seguridad de la intranet deberá ser la protección antivirus. En realidad se trata de una barrera doble. Protege la intranet de la infección no intencionada y de ataques lanzados intencionalmente. El propósito de esta capa de seguridad no es impedir el acceso no deseado a sus archivos. En su lugar, protege la integridad y la existencia de sus datos, un nivel de protección mucho más importante y básico. La capa de seguridad final proporciona el último y más importante nivel de protección para sus archivos.

No todos los datos son creados iguales

Cada parte de los datos que residen en su intranet es diferente, tanto en su autor, su propósito, su aplicación de origen y su importancia y valor para su organización. Por ello no se debería tratar igual a todos los datos.
Se deberán definir los niveles de importancia, valor y privacidad de todos los objetos de datos de la intranet. Al definir e implantar estos niveles se mejora la seguridad de su intranet y se comunica a los usuarios la importancia de proteger los datos críticos.

La gran diferencia entre una intranet y una red tradicional es el ***acceso***. El acceso es la capacidad de un usuario de la organización para interactuar con recursos de la intranet así como recursos externos de Internet o de otro servicio de información en línea. El acceso también es la capacidad de que personas de fuera de la organización interactúen con recursos de su intranet. Esto significa que las intranet ofrecen muchas más rutas de acceso a los recursos que las redes tradicionales, por lo que los aspectos relacionados con la seguridad de estas rutas adicionales deben ser tratados convenientemente.

Es importante comprender que el objetivo principal del diseño de una intranet no es la seguridad, ya que las intranet son un cruce entre las redes tradicionales y la tecnología de Internet. El objetivo de la intranet es establecer una *comunicación confiable y permitir que los usuarios remotos compartan datos*. Aunque existen medidas de seguridad efectivas instaladas en la intranet, se trata de dispositivos adicionales que no son inherentes a los protocolos o a la construcción de Internet.

En la instalación de la intranet es importante aprovechar todos los beneficios posibles de las redes tradicionales y de Internet, al tiempo que se eliminan o excluyen tantos inconvenientes como sea posible.

1.7 Conceptos básicos de la tecnología Orientada a Objetos

Debido a que el presente sistema se desarrolló con la tecnología Orientada a Objetos, se muestran a continuación algunas de las ideas fundamentales que subyacen en esta tecnología.

- Objetos y Clases
- Métodos
- Herencia
- Encapsulado
- Polimorfismo

¿Qué es un Objeto?

Las personas nos formamos conceptos desde temprana edad. Cada concepto es una idea particular o una comprensión de nuestro mundo. Los conceptos adquiridos nos permiten sentir y razonar acerca de las cosas en el mundo. A estas cosas a las que se aplican nuestros conceptos se llaman *objetos*. Un *objeto* podría ser real o abstracto.

Un *objeto* tiene estado, comportamiento e identidad; la estructura y comportamiento de objetos similares están definidos en su clase común.

- **Estado**: El *estado* de un objeto abarca todas las propiedades (normalmente estáticas) del mismo más los valores actuales (normalmente dinámicos) de cada una de esas propiedades. Una propiedad es una característica inherente o distintiva, un rasgo o cualidad que contribuye a hacer que un objeto sea ese objeto y no otro.
- **Comportamiento**. Ningún objeto existe de forma aislada. En vez de eso, los objetos reciben acciones, y ellos mismos actúan sobre otros objetos. Así puede decirse que el *comportamiento es cómo actúa y reacciona un objeto*, en términos de sus cambios de estado y paso de mensajes.
- **Identidad**. La *identidad* es aquella propiedad de un objeto que lo distingue de todos los demás objetos.

¿Qué es una clase?

Las *clases* son los *bloques de construcción* más importantes de cualquier sistema orientado a objetos. Una *clase* es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones, comportamiento y semántica. La *clase* especifica la estructura de datos de cada uno de sus objetos y las operaciones que se utilizan para tener acceso a los objetos. Entiéndase por operación como un tipo de servicio solicitado
Las *clases* se pueden utilizar para capturar el vocabulario del sistema que se está desarrollando. Estas *clases* pueden incluir abstracciones que formen parte del "dominio del problema", así como *clases* que constituyan una implantación. Se puede utilizar *clases* para representar cosas que sean software, hardware o puramente conceptuales.

Dominio del problema

El *dominio* es el área de conocimiento o actividad caracterizada por un conjunto de conceptos y terminología comprendidos por los practicantes de ese dominio. El dominio del problema es el contexto sobre el que se define un problema —generalmente un problema que debe ser "resuelto" por un sistema. El *dominio del problema* es comprendido, por lo general, por el cliente del sistema.

Métodos

Los *métodos* especifican la forma en que se controlan los datos de un objeto. Los *métodos* en un tipo de objeto sólo hacen referencia a las estructuras de datos de ese tipo de objeto.

No deben tener acceso directo a las estructuras de datos de otros objetos. Para utilizar la estructura de datos de otro objeto deben enviar un mensaje a éste. El tipo de objeto empaca juntos los tipos de datos y los *métodos*. Un objeto es entonces una cosa cuyas propiedades están representadas por tipo de datos y su comportamiento por *métodos*.

Encapsulado

El empaque conjunto de datos y métodos se llama *encapsulado*. El objeto esconde sus datos de los demás y permite el acceso a los datos mediante sus propios métodos. Esto recibe el nombre de *ocultación de la información*. El *encapsulado* evita la corrupción de los datos de un objeto. Si todos los programas pudieran tener acceso a los datos en cualquier forma que quisieran los usuarios, los datos se podrían corromper o utilizar de mala manera. El *encapsulado* protege los datos del uso arbitrario y no pretendido.

El *encapsulado* oculta los detalles de su implantación interna a los usuarios de un objeto. Los usuarios se dan cuenta de las operaciones que pueden solicitar del objeto pero desconocen los detalles de cómo se lleva a cabo la operación. Todos los detalles específicos de los datos del objeto y la codificación de sus relaciones están fuera del alcance del usuario.

El *encapsulado* es importante porque separa el comportamiento del objeto de su implantación. Esto permite la modificación de la implantación de un objeto sin que se tengan que modificar las aplicaciones que lo utilizan.

Herencia

Con frecuencia, las clases de objetos suelen tener puntos en común. Estos puntos comunes pueden manifestarse en atributos, métodos o relaciones. Siempre que entre las clases de objetos existan puntos en común de importancia podrá pensarse en crear una generalización de la clase de objetos.

La generalización es algo más que una mera relación de asociación, ya que implica la noción de *herencia*. Las clases de objetos que son especializaciones de otras clases de objeto heredan automáticamente los atributos y los métodos de su clase de generalización. Los tipos de herencia son:

- La herencia simple; Es aquélla en la que una clase puede heredar la estructura de datos y operaciones de una clase base.
- La *herencia múltiple*; Se da cuando una clase puede heredar la estructura de datos y operaciones de más de una clase base.

Polimorfismo

El *polimormismo* es la capacidad de dos objetos diferentes para comportarse de forma distinta cuando reciben el mismo mensaje. Cada objeto reacciona de distinta manera dependiendo de la información suministrada, y es capaz de interpretar el contexto de la información que ha sido introducida. Verdaderamente, el concepto de polimorfismo refleja el mundo real, donde se reciben mensajes idénticos todos los días y se reacciona de manera diferente; por ejemplo, un mensaje "+" a un objeto ENTERO significaría suma, mientras que para un objeto STRING significaría concatenación ("pegar" strings uno seguido al otro).

1.7.1 ¿Por qué Orientada a Objetos?

El objetivo de esta tecnología es obtener un software más consistente, robusto y reutilizable, más fácil de verificar, mantener, refinar y extender. El paradigma orientado a objetos representa un paso más en la dirección de acercar el lenguaje de las soluciones informáticas al lenguaje en que se plantean los problemas.

Actualmente, el enfoque orientado a objetos forma parte de la tendencia principal para el desarrollo de software, simplemente porque ha demostrado ser válido en la construcción de sistemas en toda clase de dominios de problemas, abarcando todo el abanico de tamaños y complejidades. Más aún, la mayoría de los lenguajes actuales, sistemas operativos y herramientas son orientados a objetos de alguna manera, lo que ofrece más motivos para ver el mundo en términos de objetos. El desarrollo orientado a objetos proporciona la base fundamental para ensamblar sistemas a partir de componentes.

1.7.2 Beneficios de la Tecnología Orientada a Objetos

Reutilización. La *reutilización* de clases existentes que han sido probadas en proyectos anteriores, conduce a la elaboración de sistemas de mayor calidad que satisfacen mejor los requisitos de negocios y contienen menos errores.

Herencia. La *herencia* hace posible definir sistemas más flexibles, más fáciles de extender y menos costosos de mantener.

Paso de mensajes. El paso de *mensajes* significa que las descripciones de interfaces entre módulos y sistemas externos se vuelven mucho más sencillas y sirven de apoyo para el desarrollo de GUI *(Interfase User Graphics)* y de sistemas cliente servidor.

Ocultamiento. El *ocultamiento* de información ayuda a construir sistemas seguros. La orientación de objetos es una herramienta clave para enfrentarse a la complejidad del software. Los sistemas orientados a objetos soportan mejor el paso de pequeña a gran escala. El *encapsulamiento* sirve de ayuda para el aumento de escala a medida que va creciendo el tamaño y complejidad del proyecto.

El diseñador piensa en términos del comportamiento de objeto y no en detalles. El encapsulado oculta los detalles y hace que las clases complejas sean fáciles de utilizar. Las clases son como cajas negras; el investigador utiliza la caja negra y no ve hacia el interior de ésta. Sólo debe entender el comportamiento de la caja negra y cómo comunicarse con ella.

Confiabilidad. Es probable que el software construido a partir de clases ya probadas tengan menos fallas que el software elaborado a partir de cero.
Integridad. Las estructuras de datos sólo se pueden utilizar con métodos específicos. Esto tiene particular importancia en los sistemas cliente-despachador y los sistemas distribuidos, en los que usuarios desconocidos podrían intentar el acceso al sistema.

Programación más sencilla. Los programas se conjuntan a partir de piezas pequeñas, cada una de las cuales, en general, se puede crear fácilmente. El programador crea un método para una clase a la vez. El método cambia el estado de los objetos en formas que suelen ser sencillas cuando se les considera en sí mismas.

Mantenimiento más sencillo. El programador encargado del mantenimiento cambia un método de la clase a la vez. Cada clase efectúa sus funciones independientemente de las demás. Los problemas de evolución y mantenimiento del sistema se ven paliados por la fuerte descomposición que surge del encapsulamiento y de unas interfaces uniformes de los objetos.

Modelo más realista. El análisis orientado a objetos modela la empresa o área de aplicación de manera que sea lo más cercana posible a la realidad de lo que se logra con el análisis convencional. El análisis se traduce de manera directa en el diseño y la implantación. En las técnicas convencionales, el paradigma cambia cuando pasamos del análisis al diseño y del diseño a la programación. Con la técnica orientada a objetos, el análisis, diseño e implantación utilizan el mismo paradigma y lo afina de manera sucesiva.
El reparto de tareas tiene una base natural y debería ser más sencillo. El análisis y diseño, al descomponer un dominio en objetos correspondientes a una visión orientada a la solución de sus contrapartidas del mundo real, suele ser más natural que la descomposición funcional por refinamiento progresivo.

Mejor comunicación entre los profesionales de los sistemas de información y los empresarios. Los empresarios comprenden más fácilmente el paradigma orientado a objetos. Piensan en términos de eventos, objetos y políticas empresariales que describen el comportamiento de los objetos. La metodología orientada a objetos apoya una mejor comprensión, ya que los usuarios finales y los creadores de software comparten un modelo común.

Independencia de diseño. Las clases están diseñadas para ser independientes del ambiente de plataforma, hardware y software. Utilizan solicitudes y respuestas con formato estándar. Esto

les permite ser utilizadas en múltiples sistemas operativos, controladores de bases de datos, controladores de redes, las interfaces usuaria gráficas, etc. El creador de software no tiene que preocuparse por el ambiente o esperar a que éste se especifique.

1.7.3 Desventajas de las Técnicas Orientadas a Objetos.

Los beneficios que anteriormente se pusieron de manifiesto no son universales y en la aproximación orientada a objetos existe un cierto número de problemas ocultos. Entre éstos se cuentan:

- Unos límites de tiempo muy estrictos pueden significar que no se llegue a proporcionar la reutilización.
- Es preciso desarrollar las bibliotecas de objetos y hay que diseminar el conocimiento acerca de ellas.
- La disciplina se está desarrollando muy deprisa y los productos actuales se pueden ver sustituidos por otros muy rápidamente. Por ejemplo, el apoyo pleno para la herencia múltiple todavía no está disponible en ningún sistema único.
- Cuando se utiliza la herencia y otras estructuras semánticas es necesario imponer unos controles estrictos, so pena de que la reutilización se vea comprometida.
- Los asuntos relacionados con la persistencia, concurrencia y rendimiento esperan, todavía, los beneficios del consenso.
- Es importante la topología del paso de mensajes, o resultaría muy posible construir malos sistemas orientados a objetos.
- La escritura de componentes reutilizables puede costar mucho más que lo indicado por estimaciones convencionales.
- La administración de bibliotecas de componentes es difícil y costosa.
- Es necesario el cambio de cultura y a todo el mundo le sienta mal.

1.8 Metodologías de Diseño de Software

Un *método* es un proceso disciplinado para generar un conjunto de modelos que describen varios aspectos de un sistema de software de desarrollo, utilizando alguna notación bien definida.

Una *metodología* es una colección de métodos aplicados a lo largo del ciclo de vida del desarrollo del software y unificado por alguna aproximación general o filosófica. Los métodos son importantes por varias razones. En primer lugar, inculcan una disciplina en el desarrollo de sistemas de software complejos. Definen los productos que sirven como vehículo común para la comunicación entre los miembros de un equipo de desarrollo. Además, los métodos definen los hitos que necesita la dirección para medir el progreso y gestionar el riesgo.

Los métodos han evolucionado como respuesta a la complejidad creciente de los sistemas de software. Los principios de la informática, simplemente no se escribían programas grandes, porque la capacidad de las máquinas era muy limitada. Las restricciones dominantes en la

construcción de sistemas se debía sobre todo al hardware: las máquinas tenían poca memoria principal, los programas tenían que enfrentarse a latencias considerables en dispositivos de almacenamiento secundario como tambores magnéticos y los procesadores tenían tiempos de ciclo medidos en cientos de microsegundos. En los años sesenta la economía de la computación comenzó a cambiar de manera dramática a causa del descenso de los costes del hardware y el aumento de las capacidades de los computadores. Como consecuencia, resultaba más apetecible y finalmente más económico automatizar más y más aplicaciones de creciente complejidad. Los lenguajes de programación de alto nivel entraron en escena como herramientas importantes. Tales lenguajes mejoraron la productividad del desarrollador individual y del equipo de desarrollo en su conjunto, lo que irónicamente presionó a los informáticos a crear sistemas aún más complejos.

Durante los sesenta y los setenta se propusieron muchos métodos de diseño para enfrentarse a esta complejidad en aumento. El más influyente fue el diseño estructurado descendente, también conocido como diseño compuesto. Este método fue influido directamente por la topología de lenguajes de alto nivel tradicionales, como FORTRAN y COBOL. En estos lenguajes, la unidad fundamental de descomposición es el subprograma, y el programa resultante toma la forma de un árbol en el que los subprogramas realizan su función llamando a otros subprogramas. Éste es exactamente el enfoque del diseño estructurado descendente: se aplica descomposición algorítmica para fragmentar un problema grande en pasos más pequeños.

Desde los sesenta y los setenta han aparecido computadores de capacidades muchísimo mayores. La programación estructurada parece derrumbarse cuando las aplicaciones superan alrededor de 100,000 líneas de código. Más recientemente, se han propuesto docenas de métodos de diseño, muchos de los cuales se inventaron para resolver las deficiencias detectadas en el diseño estructurado descendente.

La mayoría pueden catalogarse como pertenecientes a uno de los tres tipos siguientes:

- Diseño estructurado descendente.
- Diseño dirigido por los datos.
- Diseño orientado a objetos.

El **diseño estructurado descendente.** Este método aplica descomposición algorítmica. El diseño estructurado no contempla los problemas de abstracción de datos y ocultación de información, ni proporciona medios adecuados para tratar la concurrencia. El diseño estructurado no responde bien ante el cambio de tamaño cuando se aplica a sistemas extremadamente complejos, y es muy inapropiado para su uso con lenguajes de programación basados en objetos y orientados a objetos.

El **diseño dirigido por los datos.** En este método, la estructura de un sistema de software se deduce de la correspondencia entre las entradas y salidas del sistema. Al igual que en el diseño estructurado, el diseño dirigido por los datos se ha aplicado con éxito a una serie de dominios complejos, particularmente sistemas de gestión de la información que implican relaciones directas entre las entradas y las salidas del sistema, pero que no requieren demasiada atención hacia eventos en los que el tiempo sea crítico.

El **diseño orientado a objetos.** Su concepto subyacente es que se deberían modelar los sistemas de software como colecciones de objetos que cooperan, tratando los objetos individuales como instancias de una clase que está dentro de una jerarquía de clases. El diseño orientado a objetos refleja directamente la topología de lenguajes de programación , como Smalltalk, Object Pascal, C++, Common Lisp Object System (CLOS), Ada y Java

A continuación se muestran una lista de los métodos más conocidos de análisis Orientado a Objetos, enumerados por orden aproximado de aparición. Aún cuando la descripción se centra en el componente de análisis de estos métodos. Obsérvese que la mayoría de ellos incluye también componentes relacionados con el diseño o incluso con la implantación.

El método de **Coad-Yourdon** surgió inicialmente de un esfuerzo por trasladar la orientación a objetos las ideas procedentes del análisis estructurado. Contiene cinco frases: búsqueda de clases y objetos, partiendo del dominio de la aplicación y analizando responsabilidades del sistema; identificación de estructuras buscando las relaciones de generalización-especialización y de todo-parte; búsqueda de "sujetos" (grupos de clases objetos); definición de atributos; definición de servicios.

El método **OMT** (Técnica de Modelado de Objetos) combina conceptos de la tecnología de objetos con otros pertenecientes al modelo de entidad-relación. El método incluye un modelo estático, basado en los conceptos de clase, atributo, operación, relación y agregación, y un modelo dinámico basado en diagramas evento-estado que describe de forma abstracta el comportamiento que se pretenda tenga el sistema.

El método de **Booch** emplea un modelo lógico (estructura de clases y objetos) y uno físico (arquitectura de módulos y de procesos), que incluye componentes tanto estáticos como dinámicos y se basa en numerosos símbolos gráficos. Está destinado a ser integrado en el UML.

El método **OOSE** (Ingeniería de Software orientada a Objetos), conocido también como el método de Jacobson u Objectory que es el nombre de la herramienta que originalmente la proporcionaba, se basa en la utilización de casos de uso: modelo de objetos del dominio, modelo de análisis (los casos de uso estructurados mediante el análisis), modelo de diseño, modelo de implantación y modelo de comprobación.

Elementos de los métodos de diseño del software

La comunidad de la ingeniería del software ha desarrollado docenas de métodos de diseño diferentes que a grandes rasgos pueden clasificarse en las tres categorías, ya mencionadas anteriormente. A pesar de sus diferencias, todos estos métodos tienen elementos en común.

Específicamente, cada uno incluye lo siguiente:

- **Notación**: el lenguaje para expresar cada modelo.
- **Proceso**: las actividades que encaminan a la construcción ordenada de los modelos del sistema.

- **Herramientas**: los artefactos que eliminan el tedio de construir el modelo y reafirman las reglas sobre los propios modelos, de forma que sea posible relevar errores e inconsistencias.

1.9 Proceso Unificado de desarrollo de Software

Un *proceso* de ingeniería de software es una definición del conjunto completo de actividades necesarias para transformar los requisitos de usuario en un producto software.

La palabra *requisito* se utiliza con un sentido general, refiriéndose a "necesidades". Al principio, estas necesidades no necesariamente se entienden en su totalidad. Para capturar estos requisitos, o necesidades, de una forma más completa, tenemos que comprender con mayor amplitud el negocio de los clientes y el entorno en que trabajan sus usuarios.

Un *proceso* define quién está haciendo qué, cuándo, y cómo alcanzar un determinado objetivo. En la ingeniería de software el objetivo es construir un producto de software o mejorar uno existente. Un *proceso* efectivo proporciona normas para el desarrollo eficiente de software de calidad. Captura y presenta las mejores prácticas que el estado actual de tecnología permite. En consecuencia reduce el riesgo y hace el proyecto más predecible.

1.9.1 Proceso Unificado

El *proceso unificado*, como su nombre lo refleja, es la unificación que ha tenido lugar en muchas dimensiones: unificación de técnicas de desarrollo, a través del Lenguaje Unificado de Modelado, y unificación del trabajo de muchos metodologistas. No se pretende tratar de identificarlas todas; sin embargo, se describirá la influencia sobre el producto de los métodos de diseño orientado a objetos de Ericsson y de Rational.

El método de Ericsson

Ericeson modelaba el sistema entero como un conjunto de bloques interconectados (en UML, se les conoce como "subsistemas" y se implantan mediante "componentes"). Después, ensamblaba los bloques de más bajo nivel en subsistemas de más alto nivel, para hacer el sistema más manejable: identificaban los bloques estudiando los casos de negocio —hoy conocidos como "casos de uso", —previamente especificados. Para cada caso de uso, identificaban los bloques que deberían cooperar para realizarlo. Con el conocimiento de las responsabilidades de cada bloque, preparaban su especificación. Sus actividades de diseño producían un conjunto de diagramas de bloques estáticos con sus interfaces, agrupados en subsistemas.

El primer producto del trabajo de las actividades de diseño era una descripción de arquitectura software. Se basaba en la comprensión de los requisitos más críticos y describía brevemente cada bloque y su agrupación en subsistemas.

El método Rational

Muchos libros, artículos y documentos internos detallan los desarrollos de Rational desde 1981, pero quizá las dos contribuciones más importantes al proceso fueron los énfasis en la arquitectura y en el desarrollo iterativo.

1.9.2 Pasos del Proceso de Unificado

El presente trabajo se construyó, bajo el proceso unificado de desarrollo. Por lo tanto a continuación se hace una descripción del mismo.

En la figura 1.9.2.1 se puede observar en un nivel alto, los pasos principales del proceso unificado, en la realización de una aplicación.

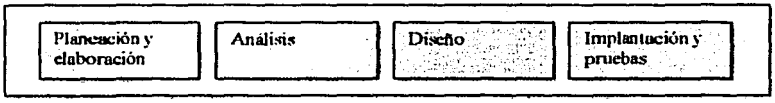


Figura 1.9.2.1. Pasos en el proceso de desarrollo de un sistema

A continuación se describen brevemente cada uno de estos pasos:

1. **Planeación y elaboración**: es una etapa del sistema que captura los requisitos y el dominio del problema. Aquí hay que *planear*, definir los requerimientos, construir prototipos, etcétera.
2. **Análisis**: durante el *análisis*, se estudian los requisitos que se describieron en la captura de requisitos, refinándolos y estructurándolos. El objetivo de hacerlo es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que nos ayude a estructurar el sistema entero.
3. **Diseño**: es la etapa del sistema que describe cómo se implementará el sistema, en un nivel lógico sobre código real. En el *diseño*, las decisiones estratégicas y tácticas se toman para resolver los requisitos funcionales y de calidad requeridos de un sistema.
4. **Implantación y pruebas**: Es aquella fase de un sistema que describe su funcionamiento en un medio ejecutable (tal como un lenguaje de programación, una base de datos o algún hardware digital). Para la *implantación* es preciso hacer que las decisiones tácticas de bajo nivel adapten el diseño al medio concreto de la *implantación* y además hay que soslayar sus limitaciones (todos los lenguajes tendrán limitaciones arbitrarias). Sin embargo, si el diseño está bien hecho entonces las decisiones de implantación serán locales y ninguna afectará a grandes segmentos del sistema.

1.10 El Lenguaje Unificado de Modelado

El lenguaje Unificado de Modelado (UML, *Unified Modeling Language*) es un lenguaje gráfico para visualizar, especificar, construir y documentar los sistemas con gran cantidad de

software. UML proporciona una forma estándar de escribir los planos de un sistema, cubriendo tanto las cosas conceptuales, tales como procesos del negocio y funciones del sistema, como las cosas concretas, tales como las clases escritas en un lenguaje de programación específico, esquemas de bases de datos y componentes de software reutilizables.

1.10.1 ¿Por qué modelamos?

Un *modelo* es la simplificación de la realidad, creada para comprender mejor el sistema que se está creando; abstracción semánticamente cerrada de un sistema. El modelado capta los aspectos importantes de lo que estamos modelando, desde cierto punto de vista, y simplifica u omite el resto.

Una empresa de software con éxito es aquella que produce de una manera consistente software de calidad que satisface las necesidades de sus usuarios. Una empresa que puede desarrollar este software de forma predecible y puntual, con un uso eficiente y efectivo de recursos, tanto humanos como materiales, tiene un negocio sostenible.

El modelado es una parte central de todas las actividades que conducen a la producción de buen software. Construimos modelos para comunicar la estructura deseada y el comportamiento de nuestro sistema. Construimos modelos para visualizar y controlar la arquitectura del sistema. Construimos modelos para comprender mejor el sistema que estamos construyendo, muchas veces descubriendo oportunidades para la simplificación y la reutilización. Construimos modelos para controlar el riesgo.

Objetivos de modelado

Construimos modelos para comprender mejor el sistema que estamos desarrollando. A través del modelado conseguimos cuatro objetivos:

- Los modelos nos ayudan a visualizar cómo es o queremos que sea un sistema.
- Los modelos nos permiten especificar la estructura o el comportamiento de un sistema.
- Los modelos nos proporcionan plantillas que nos guían en la construcción de un sistema.
- Los modelos documentan las decisiones que hemos logrado.

1.10.2 Principios del modelado

El uso del modelado tiene una historia interesante en todas las disciplinas de ingeniería. Esa experiencia sugiere cuatro principios básicos de modelado.

Primero:

La elección de qué modelos crear tiene una profunda influencia sobre cómo se acomete un problema y cómo se da forma a una solución. Cada visión del mundo conduce a un tipo de sistema diferente, con diferentes costes y beneficios.

Segundo:

Todo modelo puede ser expresado a diferentes niveles de precisión. Los mejores tipos de modelos son aquellos que permiten elegir el grado de detalle, dependiendo de quién está viendo el sistema y por qué necesita verlo.

Tercero:

Los mejores modelos están ligados a la realidad. Es mejor tener modelos que tengan una clara conexión con la realidad, y donde esta conexión sea débil, saber exactamente cómo se apartan esos modelos del mundo real. Todos los modelos simplifican la realidad; el aspecto más importante consiste está en asegurarse de que las simplificaciones no enmascaren ningún detalle importante.

En el software, el talón de Aquiles de las técnicas de análisis estructurado es el hecho de que hay una desconexión básica entre el modelo de análisis y el modelo de diseño del sistema. No poder salvar este abismo hace que el sistema concebido y el sistema construido diverjan con el paso del tiempo.

Cuarto:

Un único modelo no es suficiente. Cualquier sistema no trivial se aborda mejor a través de un pequeño conjunto de modelos casi independientes. La expresión clave aquí es "casi independientes". En este contexto significa tener modelos que podamos construir y estudiar separadamente, pero aún así están interrelacionados.

1.10.3 Modelo Orientado a Objetos

En el software hay varias formas de enfocar un modelo. Las dos formas más comunes son la perspectiva orientada a objetos y la perspectiva algoritmica.
La visión tradicional del desarrollo de software toma una perspectiva algoritmica. En este enfoque, el bloque principal de construcción de todo software es el procedimiento o función. Esta visión conduce a los desarrolladores a centrarse en cuestiones de control y de descomposición de algoritmos grandes en otros más pequeños. No hay nada inherentemente malo en este punto de vista, salvo que tiende a producir sistemas frágiles. Cuándo los requisitos cambian y el sistema crece, los sistemas construidos con un enfoque algoritmico se vuelven difíciles de mantener.

La visión actual de desarrollo de software toma una perspectiva orientada a objetos. En este enfoque, el principal bloque de construcción de todos los sistemas del software es el objeto o clase.

1.10.4 Diagramas

Cuando se modela algo, se crea una simplificación de la realidad para comprender mejor el sistema que se está desarrollando. Con UML (*Unified Modeling Language*) se construyen modelos a partir de bloques de construcción básicos.

Los *diagramas* son los medios para ver estos bloques de construcción. Un diagrama es una presentación gráfica de un conjunto de elementos que la mayoría de las veces se dibujan como grafo conexo de nodos (elementos) y arcos (relaciones). Los diagramas se utilizan para visualizar un sistema desde diferentes perspectivas. Como ningún sistema puede ser comprendido completamente desde una única perspectiva, UML (*Unified Modeling Language*) define varios diagramas que permiten centrarse en diferentes aspectos del sistema independientemente.
Cuando se modelan sistemas reales, sea cual sea el dominio del problema, muchas veces se dibujan los mismos tipos de diagramas, porque representan vistas comunes de modelos comunes. Normalmente, las partes estáticas de un sistema se representarán mediante uno de los cuatro diagramas siguientes:

1. Diagrama de clases.
2. Diagrama de objetos.
3. Diagrama de componentes.
4. Diagrama de despliegue.

A menudo se emplearán cinco diagramas adicionales para ver las partes dinámicas de un sistema:

1. Diagramas de casos de uso.
2. Diagramas de secuencia
3. Diagramas de colaboración.
4. Diagramas de estados.
5. Diagramas de actividades

CAPÍTULO 2

PLANEACIÓN, ELABORACIÓN Y ANÁLISIS DEL SISTEMA

2.1 Fase de Planeación y Elaboración

La *fase de planeación y elaboración* consiste en la especificación de los requerimientos o requisitos del sistema.

Un *requisito* es una condición o capacidad que debe cumplir un sistema.

El propósito fundamental del flujo de trabajo de los requisitos es guiar el desarrollo hacia el sistema correcto. Esto se consigue mediante una descripción de los requisitos del sistema (es decir las condiciones o capacidades que el sistema debe cumplir) suficientemente buena como para que pueda llegarse a un acuerdo entre el cliente (incluyendo a los usuarios) y los desarrolladores sobre qué debe y qué no debe hacer el sistema.

Los requisitos del sistema se clasifican de la siguiente manera:

- **Requerimientos funcionales**: Lo que debe hacer el sistema.
- **Requerimientos no funcionales**: Son las restricciones físicas del sistema.

A continuación en la figura 2.1.1 se muestra la gráfica de los pasos del proceso unificado de desarrollo de un sistema, donde se puede observar la *fase de planeación y elaboración*, dentro del proceso de desarrollo.

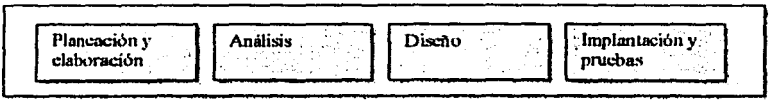


Figura 2.1.1. Pasos en el proceso unificado del desarrollo de un sistema.

2.1.1 Artefactos de la fase de Planeación y Elaboración

La siguiente figura 2.1.1.1, muestra las dependencias de los artefactos durante la fase de planeación y elaboración. Los que están señalados con una flecha, corresponden a los artefactos que se deberán construir en esta fase.

Un *Artefacto* es una pieza de información que es utilizada o producida por un proceso de desarrollo de Software.

A continuación se da una descripción breve de los artefactos que se deberán construir en la fase de planeación y elaboración.

- **Plan**: Programa, recursos, presupuesto, etcétera.
- **Informe preliminar de investigación**: Motivos, opciones, necesidades de la empresa.
- **Especificación de requerimientos Funcionales y no Funcionales**: Declaración de los requerimientos.
- **Glosario**: Diccionario (nombres de conceptos, por ejemplo) y toda información afín, como las restricciones y las reglas.
- **Prototipos**. Sistema de prototipos cuyo fin es facilitar la comprensión del problema, los problemas de alto riesgo y los requerimientos.
- **Casos de uso**. Descripción gráfica de todos los casos y de sus relaciones.

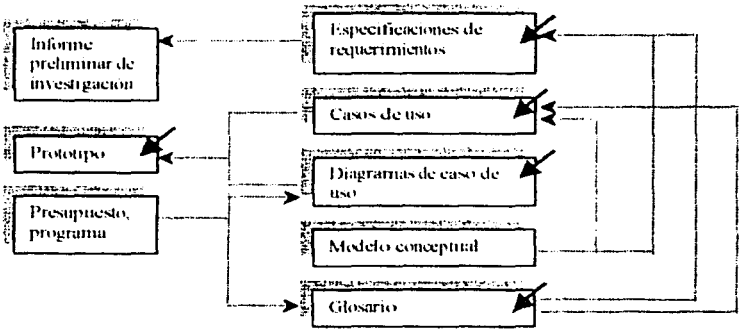


Figura 2.1.1.1. Artefactos en la fase de planeación, elaboración y análisis

En la tabla 2.1.1.1. se muestra los documentos utilizados en la etapa de planeación y elaboración del sistema.

Tabla 2.1.1.1. Documentos utilizados en la etapa de planeación y elaboración..

Planeación y elaboración	Documentos relacionados
Especificación de Requerimientos funcionales	Casos de uso
Descripción de las interfaces	Prototipo gráfico
Especificación de Requerimientos no Funcionales	Documentación

2.1.2 Especificaciones de requerimientos funcionales del sistema

Un *requerimiento funcional* es un requisito que especifica una acción que debe ser capaz de realizar el sistema, sin considerar restricciones físicas; es un requisito que especifica el comportamiento de entrada/salida de un sistema. Se recomienda elaborar los siguientes documentos en la fase de requerimientos funcionales:

- Panorama general
- Clientes
- Metas
- Funciones del sistema

Presentación general

Este proyecto tiene por objeto crear un sistema para el manejo de la información de las publicaciones que generan los investigadores de la Subdirección de Desarrollo Profesional de PEMEX.

Clientes

- Investigadores
- Alumnos de la Maestría de Ingeniería Petrolera.
- Profesores de la Maestría de Ingeniería Petrolera.

Metas

- Reducir considerablemente los costos de mensajería.
- Reducir el riesgo de la pérdida de información.
- Reducir el tiempo de entrega de la información.

Funciones del sistema

Las funciones del sistema son las que éste tendrá que realizar. Hay que identificarlas y listarlas en grupos cohesivos y lógicos.

Categorías de las funciones

Las funciones han de clasificarse fin de establecer prioridades entre ellas e identificar las que de lo contrario pasarían inadvertidas (pero que consumen tiempo y otros recursos). Las categorías de las funciones se muestran la tabla 2.1.2.1.

Tabla 2.1.2.1 Categorías de las funciones y su significado

Categoría de la función	Significado
Evidente	Debe realizarse y el usuario debería saber que se ha realizado.
Oculta	Debe realizarse, aunque no es visible para los usuarios. Esto se aplica a muchos servicios técnicos subyacentes, como guardar información en un mecanismo persistente de almacenamiento. Las funciones ocultas a menudo se omiten (erróneamente) durante el proceso de obtención de los requerimientos.
Superflua	Opcionales; su inclusión no repercute significativamente en el costo ni en otras funciones

A continuación se listan las funciones obtenidas para el manejo de la Información de la Subdirección de Desarrollo profesional de PEMEX y de la Sección de Ingeniería Petrolera de la DEPFI-UNAM.

En la tabla 2.1.2.2 se muestra la función: Consulta de Información

Tabla 2.1.2.2. Muestra los pasos a seguir para consultar un artículo publicado, de la base de datos.

#Referencia	Función	Categoría
R1.1	Realiza una búsqueda de información.	Oculta
R1.2	Muestra los datos de clasificación para los artículos.	Evidente

En la tabla 2.1.2.3 se muestra la función: Agregar nueva información

Tabla 2.1.2.3 Muestra los pasos a seguir para agregar un nuevo artículo, en la base de datos.

#Referencia	Función	Categoría
R2.1	Captura la información que clasifica a un artículo. Esto se realiza introduciendo cada uno de sus datos.	Evidente
R2.2	Ofrece un mecanismo de almacenamiento para toda la información capturada.	Oculta

En la tabla 2.1.2.4 se muestra la función: Borrar información

Tabla 2.1.2.4. Muestra los pasos a seguir para eliminar un artículo publicado, de la base de datos.

#Referencia	Función	Categoría
R3.1	Realiza la búsqueda de un artículo.	Oculta
R3.2	Muestra la información del artículo a eliminar.	Evidente
R3.3	Borra los datos del artículo del sistema.	Oculta

En la tabla 2.1.2.5 se muestra la función: Modificar los datos

Tabla 2.1.2.5. Muestra los pasos a seguir para modificar un artículo, de la base de datos

#Referencia	Función	Categoría
R4.1	Realiza una búsqueda de un artículo.	Oculta
R4.2	Muestra la información de un artículo a modificar.	Evidente
R4.3	Almacena los cambios realizados al artículo.	Oculta

Las funciones y los atributos del sistema son los documentos mínimos de los requerimientos, de modo que se necesitan otros artefactos importantes para atenuar el riesgo y entender el problema, a saber:

Requerimientos y equipos de enlace: Lista de los involucrados en la especificación de las funciones y atributos del sistema, en la realización de entrevistas, de pruebas de negocios y de otras actividades.

- Administrador del sistema.
- Investigadores.
- Analista y desarrollador.

Grupos afectados. Los que reciben el impacto del desarrollo o aplicación del sistema.

- Alumnos de la Maestría de Ingeniería Petrolera.
- Profesores de la Maestría de Ingeniería Petrolera.
- Investigadores.

Riesgos. Las cosas que pueden ocasionar el fracaso o retraso.

- Políticas de la empresa. Que no se permita la publicación en Internet de los artículos elaborados.
- Que la presentación de las interfaces no sea lo suficientemente atractiva a los usuarios.
- Poca participación de los usuarios con los analistas y diseñadores.

Glosario. Definición de los términos pertinentes.

El glosario o diccionario modelo (semejante a un diccionario de datos) incluye y define todos los términos que requieren explicación para mejorar la comunicación y aminorar el riesgo de malos entendidos.

Un significado uniforme y compartido resulta extremadamente importante durante el desarrollo de las aplicaciones, sobre todo cuando muchos miembros del equipo intervienen en el proyecto.

2.1.2.1 Casos de uso

El *caso de uso* es un documento narrativo que describe la secuencia de eventos de un actor (agente externo) que utiliza un sistema para completar un proceso. Los casos de uso son historias o casos de utilización de un sistema; no son exactamente los requerimientos ni las especificaciones funcionales, sino que ejemplifican e incluyen tácitamente los requerimientos en las historias que narran. Existen varios motivos por los cuales los casos de uso son buenos:

Las dos razones fundamentales son:

1. Proporcionan un medio sistemático e intuitivo de capturar los requisitos funcionales centrándose en el valor añadido para el usuario.
2. Dirigen todo el proceso de desarrollo debido a que la mayoría de las actividades como el análisis, diseño y prueba se llevan a cabo partiendo de los casos de uso. El diseño y la prueba pueden también planificarse y coordinarse en términos de casos de uso. Esta característica es aún más evidente cuando la arquitectura se ha estabilizado en el proyecto, después del primer conjunto de iteraciones.

Diagramas de los casos de Uso

Un *diagrama de caso de uso* explica gráficamente un conjunto de casos de uso de un sistema, los actores y la relación entre éstos y los casos de uso. Estos últimos se muestran en óvalos y los actores son figuras estilizadas. Hay líneas de comunicaciones entre los casos y los actores; las flechas indican el flujo de la información o el estímulo.

El diagrama tiene por objeto ofrecer una clase de diagrama contextual que nos permite conocer rápidamente los actores externos de un sistema y las formas básicas en que lo utilizan.

Actores

El actor es una identidad externa del sistema que de alguna manera participa en la historia del caso de uso, por lo regular estimula el sistema con eventos de entrada o recibe algo de él. Los actores están representados por el papel que desempeñan en el caso, gráficamente se representan como se muestra en la figura 2.1.2.1.1.

Figura 2.1.2.1.1. Ícono del lenguaje UML que representa un actor de casos de uso.

Los sistemas y sus fronteras

Un caso de uso describe la interacción con un "sistema". Las fronteras ordinarias del sistema son:

- La frontera hardware/software de un dispositivo o sistema de cómputo
- El departamento de una organización
- La organización entera

La figura 2.1.2.1.2 muestra gráficamente la fronteras del sistema.

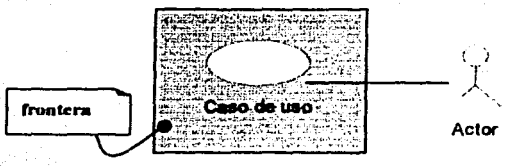


Figura 2.1.2.1.2. Frontera de un caso de uso.

A continuación se muestra el diagrama general de los casos de uso para el sistema para el manejo de la información de la Subdirección de Desarrollo Profesional de PEMEX y de la Sección de Ingeniería Petrolera de la DEPFI-UNAM

Diagrama General de Casos de Uso del Sistema

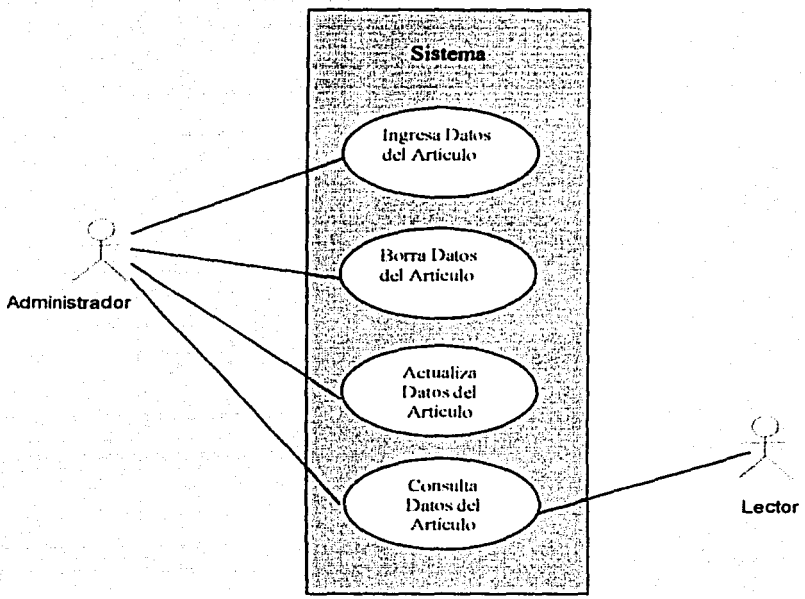


Figura 2.1.2.1.3. Diagrama general de Casos de Uso.

Caso de uso. Consultar un artículo por Título.

Actores.	Lector.
Propósito.	Consulta de los títulos de los artículos publicados por la Sección de Ingeniería Petrolera. Gráficamente este caso de uso se representa como se observa en la figura 2.1.2.1.4.
Tipo.	Primario y esencial.
Referencias	Funciones: R1.1, R1.2

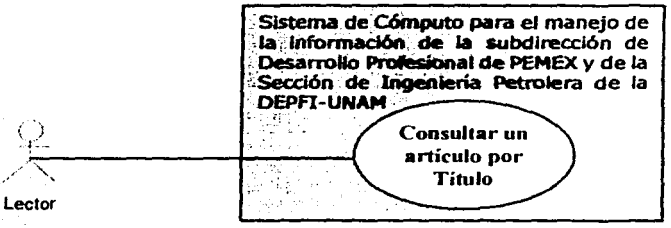


Figura 2.1.2.1.4. Diagrama de Caso de Uso. Consultar un artículo por Título

Descripción. Este caso de uso lo inicia el lector, con la finalidad de consultar los artículos generados por la Sección de Ingeniería Petrolera.

Flujo:

Curso normal de los eventos	
Acción del actor	**Respuesta del sistema**
1.Este caso de uso comienza cuando un lector utiliza el sistema para realizar una consulta del contenido de los artículos publicados. El lector no requiere de una contraseña para entrar al sistema.	
2.Introduce el nombre del artículo deseado.	
	3.Realiza la búsqueda de información por nombre (en los datos que tiene almacenados sobre los artículos publicados).
	4.Muestra el nombre, autor y contenido del artículo solicitado.
5.Termina la consulta	

Cursos alternos.

Línea 2. Introducción de identificador inválido. Indica que no pudo realizar la búsqueda.

Caso de uso. Agregar un nuevo artículo.
Actores. Administrador.
Propósito. Agregar las nuevas publicaciones de los artículos de investigación a la base de datos del sistema. Gráficamente este caso de uso se representa como se observa en la figura 2.1.2.1.5.
Tipo. Primario y esencial.
Referencias. R2.1, R2.2

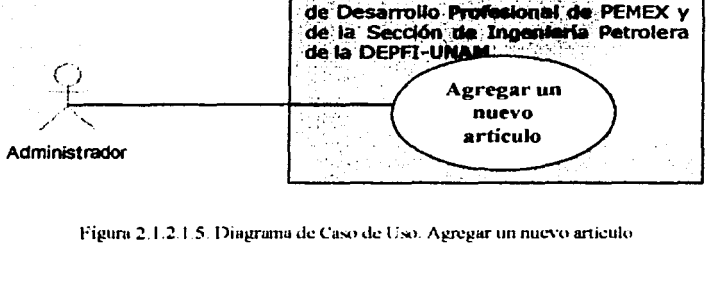


Figura 2.1.2.1.5. Diagrama de Caso de Uso. Agregar un nuevo artículo

Descripción. Este caso de uso lo inicia el administrador para agregar un artículo previamente publicado.

Flujo:

Curso normal de los eventos	
Acción del actor	**Respuesta del sistema**
1.Este caso de uso comienza cuando el administrador utiliza el sistema para guardar los datos de los nuevos artículos publicados.	
2.Introduce los datos del artículo nuevo a almacenar.	
3.Al terminar, guarda la información.	
	4.Guarda la información capturada.
	5.Limpia la pantalla para una nueva captura.

Cursos alternos.
Linea 2. Se introducen datos inválidos. Indica un error
Linea 3. Que no se introduzcan datos obligatorios. Pide capturar datos.
Linea 3. Si el artículo ya existe en el sistema de almacenamiento. Indica un error

Caso de uso. Eliminar un artículo del sistema.

Actores. Administrador.
Propósito. Eliminar la información, almacenada en la base de datos del sistema, de las publicaciones de investigación Gráficamente este caso de uso se representa como se observa en la figura 2.1.2.1.6.
Tipo. Secundario.
Referencias. R3.1, R3.2, R3.3

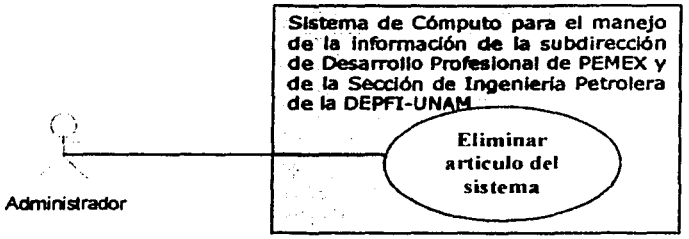


Figura 2.1.2.1.6. Diagrama de Caso de Uso. Eliminar un artículo del sistema

Descripción. Este caso de uso lo inicia el administrador para eliminar un articulo previamente seleccionado.

Flujo:

Curso normal de los eventos	
Acción del actor	**Respuesta del sistema**
1.Este caso de uso comienza cuando el administrador utiliza el sistema para eliminar los datos de un artículo almacenado en el sistema.	
2.Introduce la clasificación del articulo que desea eliminar.	
	3.Realiza la búsqueda de información por clasificación (en los datos que tiene almacenados sobre los articulos).
	4.Muestra la información del artículo solicitado en pantalla.
5.Borra la información desplegada en pantalla.	
	6.Guarda estos cambios en la base de datos.
	7.Limpia la pantalla.

Cursos alternos.
Linea 2. Introducción de identificador inválido. Indica que no pudo realizar la búsqueda.

Caso de uso. Actualizar un artículo.

Actores. Administrador.

Propósito. Actualizar la información almacenada de las publicaciones de investigación Gráficamente este caso de uso se representa como se observa en la figura 2.1.2.1.7.

Tipo. Secundario.

Referencias. R4.1, R4.2, R4.3

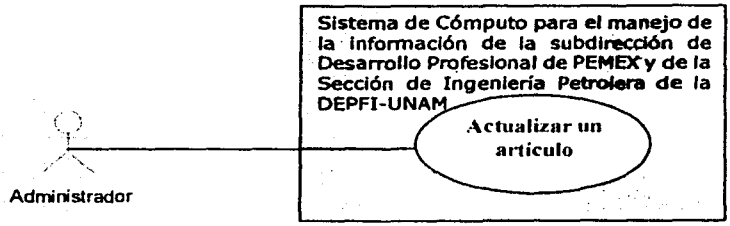


Figura 2.1.2.1.7. Diagrama de Caso de Uso. Actualizar un artículo.

Descripción. Este caso de uso lo inicia el administrador para eliminar un artículo previamente seleccionado.

Flujo:

Curso normal de los eventos **Acción del actor**	**Respuesta del sistema**
1.Este caso de uso comienza cuando el administrador utiliza el sistema para realizar algunas modificaciones a los datos almacenados sobre algún artículo publicado.	
2.Introduce la clasificación del artículo que desea borrar.	
	3.Realiza la búsqueda de información por clasificación (en los datos que tiene almacenados de los artículos publicados).
	4.Muestra la información en pantalla.
6. Introduce la nueva información y guarda los cambios.	
	7.Guarda las modificaciones de los datos.
	8.Limpia la pantalla.

Cursos alternos.

Línea 2. Introducción de identificador inválido. Indica que no pudo realizar la búsqueda.

Línea 6. Introducción de datos inválidos. Indica un error.

2.1.2.2 Construcción del Prototipo de la Interfaz

Es importante que los usuarios estén envueltos en las descripciones de las interfaces detalladas. Por consiguiente estas descripciones deben hacerse en una fase temprana. La interfaz tiene que capturar la vista lógica del usuario del sistema, porque el interés principal es la consistencia de esta vista lógica y la conducta real del sistema. Aquí las metas son:

- ✓ Especificar los requisitos del cliente
- ✓ Adaptar los puntos de vista y términos de acuerdo con los usuarios.
- ✓ La participación de los usuarios debe ser activa en este modelo.

El problema fundamental con que se enfrenta el usuario que trata de definir un nuevo sistema de software es que resulta muy difícil valorar cómo afectará a su trabajo la existencia de tal sistema. Para sistemas nuevos, en especial si son grandes y complejos, quizá sea imposible obtener una definición de requisitos consistente, completa y válida antes de construir y poner en uso el sistema.

Esto ha conducido a las sugerencias de que se debe adoptar un enfoque evolutivo en el desarrollo de sistemas. Ello implica presentar al usuario un sistema que se sabe incompleto y después modificar y ampliar ese sistema a medida que se van evidenciando los requisitos reales del usuario. Una manera de lograr esto es mediante la construcción de prototipos, haciendo que uno de los prototipos del sistema se transforme mediante una serie de pasos en el sistema final entregado.

Hay también la alternativa de tomar una decisión deliberada para construir un prototipo "desechable". Este es un enfoque muy utilizado en el desarrollo de hardware, donde se construye un prototipo para identificar los problemas iniciales y, después de experimentar, se formula una especificación mejorada. A continuación, se desecha el prototipo y se construye un sistema de calidad de producción. Este enfoque también es aplicable al desarrollo del software.

Las ventajas de utilizar un sistema prototipo durante el análisis de requisitos y la fase de definición del ciclo de vida del software pueden resumirse como sigue:

- Los malentendidos entre los desarrolladores del software y los usuarios pueden identificarse a medida que se demuestran las funciones del sistema.
- Pueden detectarse los servicios que le faltan al usuario.
- Se puede identificar y redefinir los servicios del usuario difíciles de utilizar o confusos.
- El personal de desarrollo del software puede encontrar los requisitos incompletos e inconsistentes durante el desarrollo del prototipo.
- Se dispone con rapidez de un sistema de trabajo, aunque limitado, para demostrar la viabilidad y utilidad de la aplicación a la administración.

En la figura 2.1.2.2.1 se muestra el prototipo de la pantalla para realizar la consulta de artículos ya sea por autor o título; para realizar actualizaciones o para agregar un nuevo

articulo se elaboran los prototipos mostrados en la figura 2.1.2.2.2 y en la figura 2.1.2.2.2.4. Finalmente en la figura 2.1.2.2.3 y figura 2.1.2.2.5 se muestran los catálogos de autores y paises.

Figura 2.1.2.2.1. Prototipo de pantalla para la consulta de los articulos de las publicaciones PEMEX.

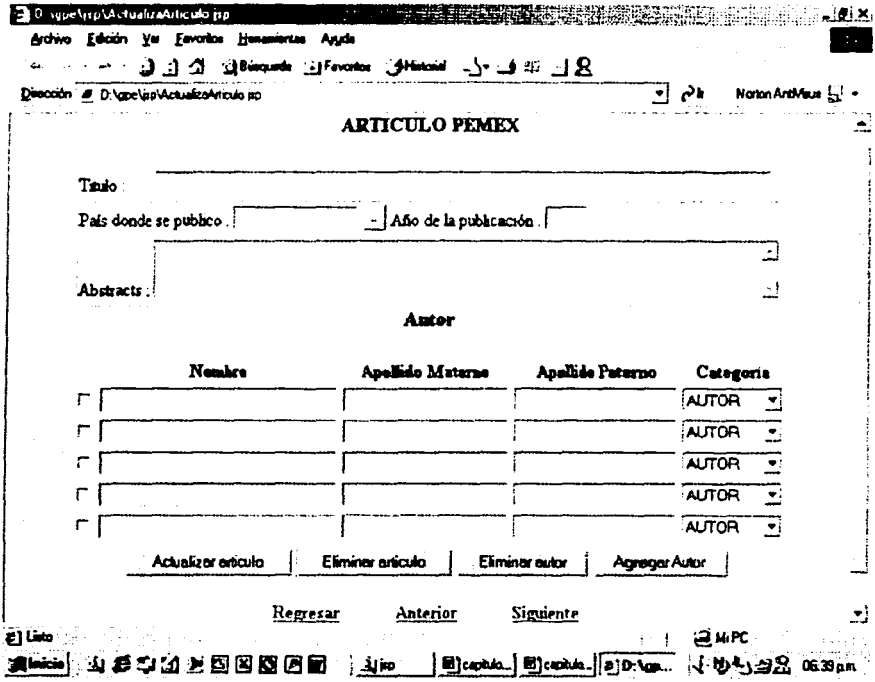


Figura 2.1.2.2.2. Prototipo de pantalla para la modificación de los artículos de las publicaciones PEMEX.

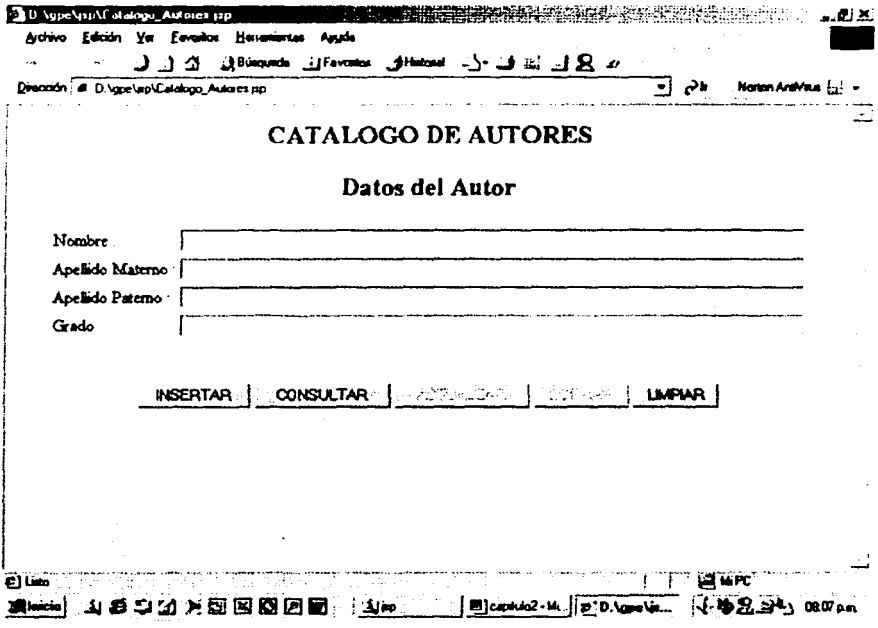


Figura 2.1.2.2.3. Prototipo de pantalla para el catálogo de autores de los artículos publicados.

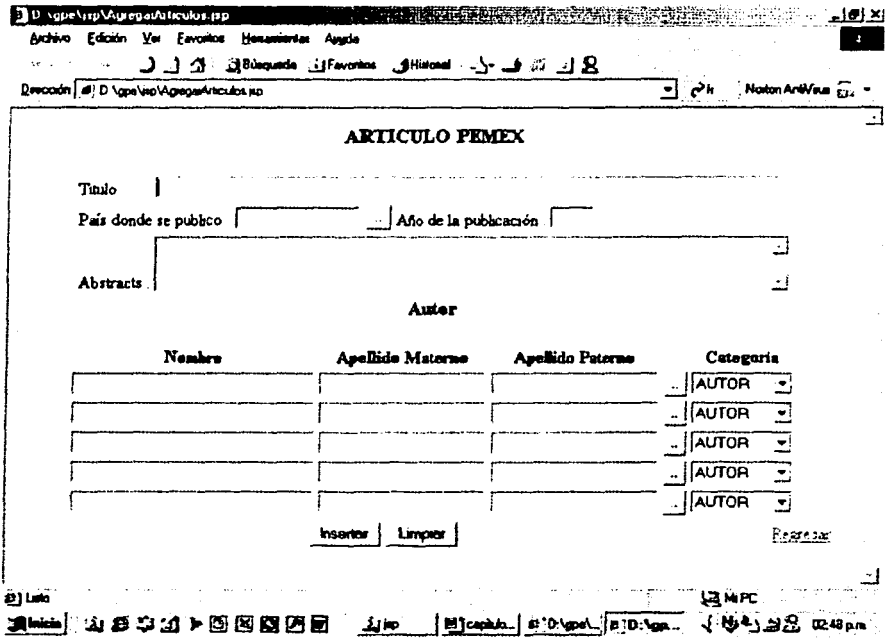


Figura 2.1.2.2.4. Prototipo de pantalla para el registro de los artículos de las publicaciones PEMEX.

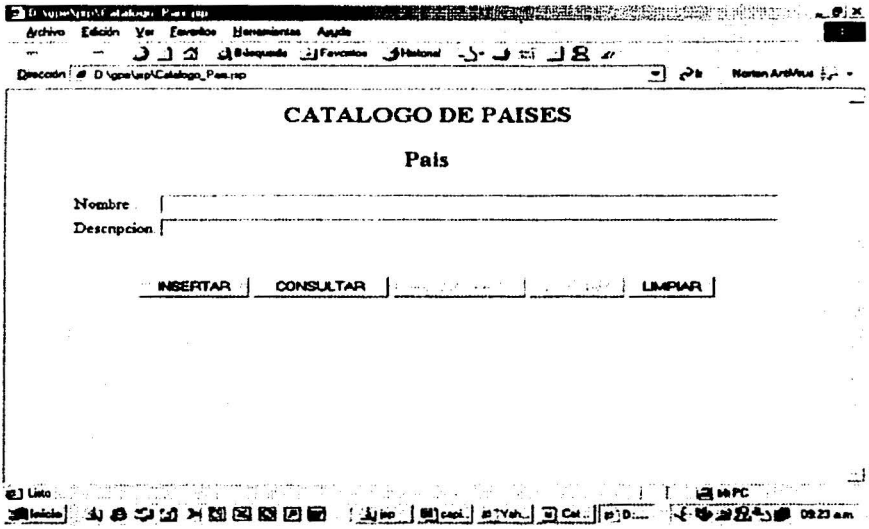


Figura 2.1.2.2.5. Prototipo de pantalla para el catálogo de países de los artículos.

2.1.3 Especificaciones de requerimientos no funcionales del sistema

Los *requerimientos no funcionales* son requisitos que especifican propiedades del sistema, como restricciones del entorno o de la implantación, rendimiento, dependencias de la plataforma, facilidad de mantenimiento, extensibilidad, y confiabilidad. La *confiabilidad* hace referencia a características como la disponibilidad, exactitud, tiempo medio entre fallos, defectos por miles de líneas de código y defectos por clase. Un requisito de rendimiento impone condiciones sobre los requisitos funcionales como la velocidad, rendimiento, tiempo de respuesta y uso de memoria. A continuación se listan algunos de estos requisitos.

Restricciones de diseño e implantación

- *Restricciones de diseño*: limita el diseño de un sistema, como lo hacen las restricciones de extensibilidad y mantenibilidad o las restricciones relativas a la reutilización de sistemas heredados o partes esenciales de los mismos.
- *Restricción de implantación*: especifica o limita la condición o construcción de un sistema. Son ejemplos los estándares requeridos, las normas de codificación, los lenguajes de programación, políticas para la integridad de la base de datos, limitaciones de recurso y entornos operativos.

Requerimientos del Sistema

- *Requisito de interfaz (Software)*: especifica la interfaz con un elemento externo con el cual debe interactuar el sistema, o que establece restricciones condicionantes en formatos, tiempos u otros factores de relevancia en esa interacción.
- *Requisitos físicos (Hardware)*: especifica una característica física que debe poseer un sistema, como su material, forma, tamaño o peso. Por ejemplo, puede utilizarse este tipo de requisito para representar requisitos hardware como las configuraciones físicas de red requeridas.

Requerimientos especiales

- Seguridad
- Portabilidad
- Desempeño
- Facilidad de aprendizaje
- Mantenimiento

2.1.3.1 Restricciones de diseño e implantación

Se necesitará para la construcción del sistema lo siguiente:

1. Lenguaje de programación.
2. Un sistema para el almacenamiento y administración de la base de datos.
3. Servidor de páginas Web.
4. Herramienta para la creación de la documentación del sistema.

2.1.3.1.1 Lenguaje de programación

Java

Java es un lenguaje de programación orientado al objeto desarrollado por Sun Microsystems. Su sintaxis es una derivación directa de la de C++. Los únicos cambios que se han hecho son eliminar ciertas características "conflictivas" de C++ como los punteros. Java no es un lenguaje destinado a la creación de páginas web. Aunque su utilización en el interior de páginas HTML sea uno de los usos más conocidos de Java.

Características de Java

A continuación se listan las características más importantes de Java

- Independencia de la plataforma hardware
- Compatibilidad
- Lenguaje de programación orientado a objetos
- Robustez
- Multienhebramiento
- Applets

Independencia de la plataforma hardware

Lo que hace a Java un lenguaje tan atractivo frente a cualquier otro lenguaje de programación es que es independiente del hardware o plataforma, tanto a nivel de código fuente como a nivel binario. A nivel de código fuente, las estructuras de datos primitivas de Java tienen los mismos tamaños para cualquier plataforma de desarrollo.

Los ficheros binarios de programas Java. también son independientes y pueden ejecutarse sobre distintos sistemas sin necesidad de recompilar el código fuente. Esto es posible gracias a que los ficheros binarios de Java se adaptan al formato que Sun denomina "binary bytecode format". Los bytecodes son un conjunto de instrucciones de bajo nivel muy similares al código máquina, pero sin ser específico de ninguna arquitectura de procesadores.

Los programas en Java pueden ejecutarse en cualquiera de las siguientes plataformas, sin necesidad de hacer cambios:

- Windows/95 y /NT
- Power/Mac
- Unix (Solaris, Silicon Graphics,..)

Lenguaje de programación orientado al objeto.

Java es un lenguaje de programación orientada al objeto (por tanto soporta tres caracteristicas de este tipo de programación encapsulado, herencia y polimorfismo). Java se basa en C++, con una sintaxis similar, pero que está diseñado para evitar las características más problemáticas del C++ (no existe aritmética de punteros en Java, los strings y los arrays se consideran objetos y la gestión de la memoria es automática), lo que hace más fácil la programación en Java. Java incluye un conjunto de librerías de clases para obtener los tipos de datos básicos, procedimientos de entrada/salida, comunicaciones a través de red, lleva integrados protocolos de Internet (TCP/IP, HTTP y FTP) y funciones para desarrollar interfaces de usuario.

En C++ se suele trabajar con librerías dinámicas (DLL's) que obligan a recompilar la aplicación cuando se retocan las funciones que se encuentran en su interior. Este inconveniente es resuelto por Java mediante un interfaz especifico llamado RTTI(Run Time Type Identification) que define la interacción entre objetos excluyendo variables de instancias o implantación de métodos. Las clases en Java tienen una representación en el intérprete que permite a los programadores interrogar por el tipo de clase y enlazar dinámicamente la clase con el resultado de la búsqueda.

Robustez

Java se puede considerar un lenguaje robusto. A diferencia de C++, con el que resulta sumamente fácil tener que reinicializar el ordenador por culpa de algún error de programación. Java obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error. Java soporta punteros, pero no así la aritmética propia que éstos tienen en C++. Java siempre verifica los índices al entrar a un arreglo. Se implantan arrays auténticos, en vez de listas enlazadas de punteros. De esta manera, se consigue evitar el problema de corrupción de memoria resultante de punteros que señalan a zonas equivocadas. Java posee un recolector de basura que administra automáticamente la memoria. Es el recolector el que determina cuándo se puede liberar el espacio ocupado por un objeto cuestión que no puede realizar el programador.

Multienhebramiento

Java puede trabajar además con sistemas operativos de alto nivel que soporten multienhebramiento (multithreading). De esta forma, un programa Java puede tener más de una hebra en ejecución. Por ejemplo, podría realizar un cálculo largo en una hebra, mientras otras hebras interactúan con el usuario. Así los usuarios no tienen que dejar de trabajar mientras los programas Java completan las operaciones más largas.

Applets y aplicaciones Java

Los programas Java se dividen en dos grupos: Applets y aplicaciones propiamente dichas. Los Applets son programas Java que se encuentran en un servidor de páginas Web y son ejecutados por un navegador Web en el ordenador del cliente. Si el cliente no posee un navegador que soporte Java, no podrá ejecutar los applets.

Las aplicaciones son programas independientes y más generales escritos en lenguaje Java. Estas aplicaciones no necesitan de un navegador para ejecutarse y de hecho, se puede utilizar Java para escribir un programa como lo haríamos con C o Pascal. Para ejecutar estos programas se debe utilizar el intérprete de Java. Por ejemplo, el navegador de Web creado por Sun, HotJava (naturalmente, como su nombre indica es un navegador que soporta Java) es una aplicación escrita íntegramente en lenguaje Java.

Un mismo programa Java puede ser un Applet o una aplicación o ambas cosas a la vez, dependiendo de cómo se escriba el programa.

2.1.3.1.2 Un sistema para el almacenamiento y administración de las bases de datos

Acces

Acces un sistema de gestión de bases de datos relacionales de Microsoft Corporation, integrado dentro del paquete de aplicaciones de Office.

Algunas de las funciones que realiza *Acces* son las siguientes:

- ✓ Crear Bases de Datos
- ✓ Crear y diseñar tabla
- ✓ Introducir datos
- ✓ Ordenar y filtrar registros
- ✓ Buscar información
- ✓ Crear consultas, utilizar criterios y calcular totales
- ✓ Crear y dar formato a nuevos formularios
- ✓ Manejar distintos controles disponibles: etiquetas, cuadros de texto, botones, imágenes, etc.
- ✓ Crear y diseñar distintos tipos de informes
- ✓ Ordenar y agrupar la información
- ✓ Automatizar distintas tareas de Acces mediante la utilización de macros.

Se trabajará con esta herramienta en la elaboración del presente sistema, ya que además de ser muy fácil de utilizar y la información que se almacenará es pequeña; también se cuenta con licencia para la misma.

2.1.3.1.3 Servidor Web

¿Qué es un Servidor Web?

Para ver y examinar páginas Web, lo único que se necesita es un navegador Web. Para publicar páginas Web, en la mayoría de los casos necesitará un *servidor Web*.

Un *servidor Web* es un **programa** que se encuentra en una máquina de Internet, a la espera de que un navegador se conecte con él y le solicite un archivo. Cuando le llega la solicitud por el cable, el servidor acepta la conexión, localiza el archivo y lo envía al navegador y después cierra la conexión. Entonces el navegador le da formato a la información que recibe del servidor. Los servidores y los navegadores Web se comunican mediante HTTP (*Protocolo de Transferencia de Hipertexto*) "lenguaje" creado específicamente para la solicitud y transferencia de documentos de hipertexto a través de Web.

Por el lado del servidor, muchos navegadores pueden conectarse al mismo servidor para solicitar la misma información. El servidor Web es responsable de manejar toda esas solicitudes.

Los servidores Web hacen más que solo depositar archivos, también son responsables de:

- **Tipos de archivo y medios.** Los servidores son responsables de indicarle al navegador el tipo de contenido del archivo. Un servidor Web se puede configurar para que envíe diferentes clases de medios y para que maneje diversos tipos de archivos y extensión.
- **Administración de archivos y de medios.** El servidor es responsable de la administración de archivos, que consiste básicamente en determinar dónde se encuentran un archivo y a dónde se envía.
- **Se encarga de manejar la entrada de formularios** y de vincular formularios y navegadores con programas que corran en el servidor, como bases de datos.
- **Procesamiento de scripts CGI (Common Gateway Interface), programas y formularios.** Una de las tareas más interesantes (y más complejas) que puede realizar un servidor es ejecutar programas externos en la máquina, basándose en la entrada que le dan los lectores desde sus navegadores. A estos programas por lo general se les llama scripts de CGI (*Common Gateway Interface*) y son la base para crear formularios interactivos. Los scripts CGI (*Common Gateway Interface*) también se usan para conectar un servidor Web con una base de datos o algún otro sistema de información del lado del servidor.
- **Procesamiento de archivos del lado del servidor.** Algunos servidores pueden procesar archivos antes de enviárselos a los navegadores. Por ejemplo, los contadores de acceso que se ven en las páginas Web, o las modificaciones que se hacen a los archivos al vuelo según el navegador solicitante también en la ejecución de pequeñas partes del código script.
- **Autentificación y seguridad.** La autentificación es la capacidad de proteger archivos y directorios en el servidor Web, ya que para ver los archivos se exige que el lector escriba su nombre y su contraseña.

Al igual que sucede con los navegadores existen numerosos servidores para diferentes plataformas, cada uno con sus propias caracteristicas y en un rango de costos que va desde el gratuito hasta el muy costoso.

JavaServer Pages

La tecnología *JavaServer pages* (JSP) está basada en el lenguaje de programación Java y encaminada a facilitar el desarrollo de los cada vez más complicados sitios Web.

La incorporación de contenido dinámico en un sitio Web siempre lleva consigo algún tipo de programación para indicar cómo debe generarse ese contenido dinámico. Una de las metas en la creación de contenido dinámico es minimizar la necesidad de programación y, en último caso, separar la programación de la presentación del contenido. Combinando este objetivo con el uso de Java y la utilización de etiquetas, la tecnología *JavaServer Pages* (JSP) es el resultado creado por *Sun Microsystems*.

La tarea de la generación de contenido dinámico debe ser separada en dos partes, para facilitar la programación y reducir en lo posible el coste de creación y mantenimiento. Las dos partes que intervienen en la generación de contenidos dinámicos son entonces:

- La *lógica de negocio*, creación de contenidos que controla la relación entre la entrada, los algoritmos y la salida.

- La *lógica de presentación*, presentación de contenidos o diseño gráfico que determina la forma en que se va a presentar la información al usuario. La lógica de presentación puede ser manejada a través de la tecnología JSP (*JavaServer Pages*).

La tecnología JSP (*JavaServer Pages*) es un producto hibrido, porque por un lado soporta código embebido en sus páginas, al igual que ASP (*Active Server Pages*) o PHP (*Personal Home Page*) pero por otro, también permite el uso de etiquetas que interactúan con objetos Java en el servidor.

Con este modelo hibrido la tecnología JSP (*JavaServer Pages*) proporciona muchas ventajas. Los desarrolladores pueden ofrecer etiquetas personalizadas que los diseñadores de páginas pueden utilizar mediante sintaxis semejante a las etiquetas HTML que ya conoce: Como el motor JSP (*JavaServer Pages*) es capaz de compilar la página JSP (*JavaServer Pages*) bajo demanda, el autor de la página puede realizar actualizaciones fácilmente. Las páginas JSP (*JavaServer Pages*) pueden proporcionar acceso a componentes JavaBeans que encapsulan la lógica de negocio, o programación, acceso a datos externos, etc. Estos componentes, una vez escritos, son transportables entre plataformas y servidores. La reutilización de los componentes ya existentes acelera el desarrollo de nuevas aplicaciones.

Los diseñadores de páginas Web pueden modificar y editar la parte estática de la página tantas veces como deseen, sin afectar a la lógica de la aplicación.

Ejecución de páginas JSP (*JavaServer Pages*)

Una página JSP (*JavaServer Pages*) no es más que otra forma de ver un servlet. El concepto inherente en un archivo JSP (*JavaServer Pages*) es permitir ver un servlet Java como una página HTML. Esto elimina la desagradable lista de sentencias print() que normalmente lleva el código Java del servlet. Es decir, una página JSP (*JavaServer Pages*) trata de permitir que se pueda incluir código java dentro de una página HTML normal.

Los servlet son como pequeñas aplicaciones para añadir funcionalidad interactiva a los servidores Web. El modelo de programación de los servlets es semejante al utilizado por los scripts CGI, pero a diferencia de éstos no necesita lanzar un nuevo proceso por cada petición http, sino que todos los servlets asociados a un servidor Web se ejecutan en proceso único, siendo la máquina virtual java la que crea una tarea específica para atender a cada petición.

Una página JSP (*JavaServer Pages*) es preprocesada en un archivo .java, que luego es compilado para generar un archivo .class. Ésta es la innovación que proporciona la tecnología JavaServer Pages y que la diferencia de otras semejantes, por ejemplo, una página Active Server Page (ASP) de Microsoft, se compila en memoria, no en un archivo separado. La figura 2.1.3.1.3.1 muestra el flujo de la conversión de una pagina JSP (*JavaServer Pages*).

Figura 2.1.3.1.3.1

Cuando un servidor Web recibe la petición de una página JSP (*JavaServer Pages*), éste lanza el motor JSP (*JavaServer Pages*), que se ejecuta en el mismo proceso que ese servidor Web. El motor JSP (*JavaServer Pages*) comprueba si la página es nueva o ha cambiado, en cuyo caso realiza el proceso de traslación de la página y luego compila el resultado. El proceso de traslación es la parte principal del funcionamiento de la tecnología JSP (*JavaServer Pages*) y consiste en la conversión de la página JSP (*JavaServer Pages*) en un servlet Java. Este servlet es compilado mediante el compilador Java estándar y ejecutado utilizando el API (*Application Programming Interfaces*) estándar Java.

El proceso de traslación es el que alenta la ejecución de las páginas JSP (*JavaServer Pages*); sin embargo, una vez que la página JSP (*JavaServer Pages*) ha sido convertida a servlet y compilada, su ejecución es tan rápida como si su origen hubiese sido un servlet normal.

La figura 2.1.3.1.3.2 muestra los componentes involucrados en el procesamiento de la petición http realizada por el cliente de una página JSP (*JavaServer Pages*):

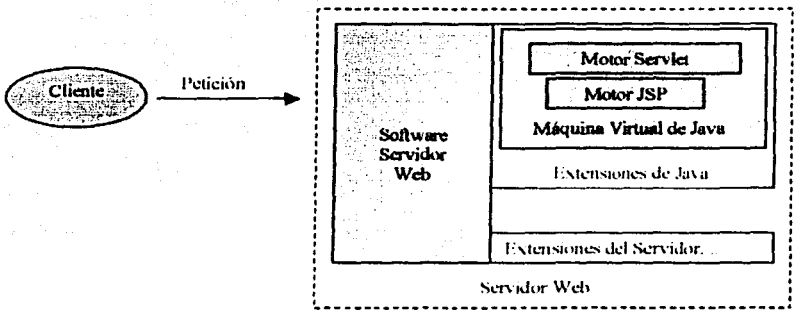


Figura 2.1.3.1.3.2

En la figura 2.1.3.1.3.2 se muestra en más detalle el proceso de la petición http. Todo el tratamiento de la petición http que se hace en el servidor web hasta que se devuelve la respuesta al cliente, se resume en cuatro pasos:

1. El motor JSP (*JavaServer pages*) analiza la página solicitada y crea un fichero de código fuente Java correspondiente al servlet.
2. El servlet generado se compila para obtener el archivo .class, que pasa al control del motor servlet que lo ejecuta del mismo modo que si se tratase de cualquier otro servlet.
3. El motor servlet carga la clase del servlet generado para ejecutarlo.
4. El servlet se ejecuta y devuelve su respuesta al solicitante.

La figura 2.1.3.1.3.3 muestra el flujo completo desde la petición de la página JSP hasta la obtención de la respuesta detallada:

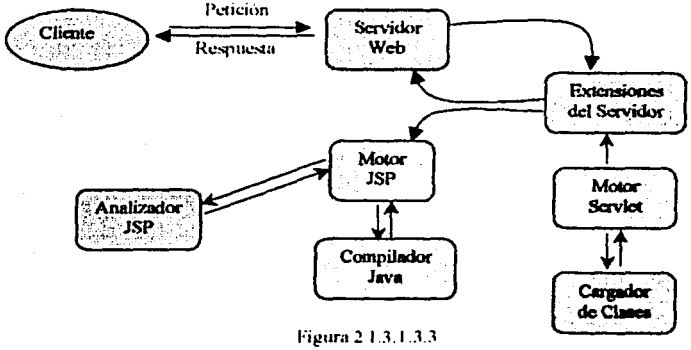


Figura 2.1.3.1.3.3

El proceso es mucho más eficiente de lo que puede deducirse observando la figura anterior. El análisis de la página y la traslación a servlet solamente ocurre una vez, la primera ocasión en que se realiza la solicitud de la página o cuando se modifica ésta. La carga de la clase en el motor servlet también ocurre una sola vez desde la última ocasión en que se haya arrancado el motor servlet. Después de esto, el servlet ya está disponible durante toda la vida de la máquina virtual Java. Incluso para mejorar la eficiencia del último paso, hay algunos servidores Web que proporcionan un mecanismo de "caché" de página que reduce el coste de ejecutar la petición y mejorar el rendimiento de la aplicación; es decir, mediante este mecanismo, la generación de la respuesta puede realizarse una única vez o bien solamente su contenido dinámico haya sido variado.

En junio de 1999, Sun Microsystem y la Apache Software Foundation anunciaron conjuntamente el proyecto Jakarta, cuyo objetivo es la implantación en código abierto de las especificaciones Servlet y JSP (*JavaServer pages*) sirviendo de plataformas de referencia para las tecnologías.

Jakarta-Tomcat

El servidor *Jakarta-Tomcat*, es uno de los proyectos de código abierto liberado por Apache Software Foundation. El servidor *Tomcat* es una aplicación web basada en Java creada para ejecutar servlets y páginas JSP, siendo la implantación oficial de referencia de las especificaciones Servlet 2.3 y JavaServer Pages 1.2. Este servidor presenta una gran estabilidad en los servicios que ofrece y al permitirme trabajar con Java, se podrá utilizar para la implantación de la presente aplicación.

2.1.3.1.4 Herramienta utilizada para documentar el sistema

Rational Rose

Es una herramienta para la generación de código y documentación de los sistemas orientados a objetos. Algunas de sus principales características:

- **Modelado de componentes.** Rose soporta la notación nativa de UML/COM (*Unified Modeling Language*)/(*Common Objet Modeling*). De tal forma que los usuarios podrán modelar sus componentes e interfaces fácilmente.
- **Soporta múltiples lenguajes.** Rose soporta multiples lenguajes en un área única de trabajo. Tales lenguajes son: C++, Java y Visual Basic.
- **Otras características.** Permite el uso de etiquetas para ponerle nombre a los íconos de modelado. Soporta una gran cantidad de estereotipos.

2.1.3.2 Requerimientos del Sistema

Para que el sistema trabaje de manera adecuada es necesario que se garanticen al menos las siguientes condiciones externas.

2.1.3.2.1 Hardware

- **Servidor**

  El Servidor Web corre en cualquiera de los siguientes sistemas operativos.
  - Un sistema operativo: WindowsNT/2000, Unix (Solaris, Silicon Graphics)
  - Procesador mayor a 140 MHz.
  - Conexión a la red.
  - Memoria de al menos 128 MB.
  - Espacio libre en disco de al menos 60 MB (para la base de datos, el servidor Web y el kit de desarrollo java).

- **Cliente**

  Los programas en Java pueden ejecutarse en cualquiera de las siguientes plataformas, sin necesidad de hacer cambios:
  - Un sistema operativo: Windows/95/98/2000, NT, Power/Mac, Unix (Solaris, Silicon Graphics)
  - Procesador mayor a 140 MHz.
  - Conexión a la red.
  - Memoria de al menos 128 MB.
  - Un módem con una velocidad de 9600 bauds (se recomienda de 14400 bauds) o una dirección de red con salida a Internet.

2.1.3.2.2 Software

- **Servidor**
  - Base de datos : Acces 2000
  - Servidor Web: Jakarta-Tomcat 4.0
  - Lenguaje de programación: Kit de desarrollo JDK1.2

- **Cliente**
  - Navegador: Netscape 2.0 o posterior, Internet Explorer 3.0 o posterior.

Comunicaciones

- Soporte para redes basadas en el protocolo de comunicaciones TCP/IP (*Transmisión Control Protocol/Internet Protocol).*

2.2 Fase de Análisis del sistema

El propósito fundamental del *análisis* es estudiar los requisitos con mayor profundidad, pero con la gran diferencia (comparado con la captura de requisitos) de que puede utilizarse *el lenguaje de los desarrolladores* para describir los resultados.

El lenguaje que utilizaremos en el análisis se basa en un modelo de objetos conceptual. Este modelo presenta las diversas categorías de las cosas en el dominio del problema; no sólo los papeles de las personas sino también todas las cosas de interés. El modelo conceptual nos ayuda a refinar los requisitos y nos permite razonar sobre los aspectos internos del sistema, incluidos sus recursos compartidos internos. De hecho, un recurso interno puede representarse como un objeto en el modelo de análisis.

Analizar los requisitos en la forma de un modelo conceptual es importante por varios motivos:

- ✓ Un modelo conceptual ofrece una especificación más precisa de los requisitos que la que tenemos como resultado de la captura de requisitos, incluyendo al modelo de casos de uso.
- ✓ Un modelo conceptual se describe utilizando el lenguaje de los desarrolladores, y puede por tanto introducir un mayor formalismo y ser utilizado para razonar sobre los funcionamientos internos del sistema.
- ✓ Un modelo conceptual estructura los requisitos de un modo que facilita su comprensión, su preparación, su modificación y en general, su mantenimiento.
- ✓ Un modelo conceptual puede considerarse como una primera aproximación al modelo de diseño (aunque es un modelo por sí mismo) y es por tanto una entrada fundamental cuando se da forma al sistema en el diseño y en la implantación. Esto se debe a que debería ser mantenible el sistema en su conjunto y no sólo la descripción de sus requisitos.

A continuación en la figura 2.2.1. se muestra la gráfica de los pasos del proceso unificado de desarrollo de un sistema, donde se puede observar la *fase de análisis*, dentro del proceso de desarrollo.

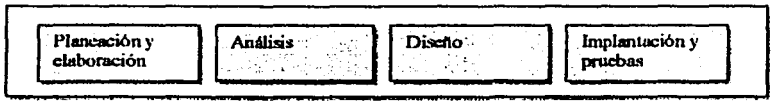


Figura 2.2.1 Pasos en el proceso unificado del desarrollo de un sistema.

2.2.1 Artefactos de la fase de Análisis

La siguiente figura 2.2.1.1, muestra las dependencias de los artefactos durante la fase de análisis del sistema. Los que están señalados con una flecha, corresponden a los artefactos que se deberán construir en la fase de análisis.

A continuación se da una descripción breve de los artefactos que se deberán construir en la fase de análisis.

Bosquejo del modelo conceptual. Modelo conceptual preliminar cuya finalidad es facilitar el conocimiento del vocabulario del dominio, especialmente en su relación con los casos de uso y con las especificaciones de los requerimientos.

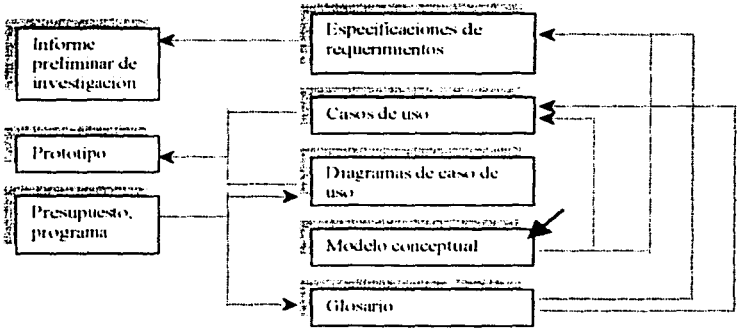


Figura 2.2.1.1. Pasos en el proceso unificado del desarrollo de un sistema.

En la tabla 2.2.1.1 se muestran los documentos utilizados en la etapa de análisis.

Tabla 2.2.1.1. Documentos utilizados en la etapa de análisis.

Planeación, elaboración.	Documentos relacionados
Análisis del dominio	Modelo conceptual
Análisis del comportamiento de los sistemas	Modelo de secuencias

2.2.2 Construcción de un Modelo Conceptual

El paso esencial de un análisis o investigación orientada a objetos es descomponer el problema en conceptos u objetos individuales: las cosas que sabemos. Un *modelo conceptual* es una representación de conceptos en un dominio del problema (o contexto del sistema).

El Modelo Conceptual puede mostrarnos:

- Conceptos
- Asociaciones entre conceptos
- Atributos de conceptos

Concepto

En términos informales el concepto es una idea, cosa u objeto. En un lenguaje más formal, podemos considerarlo a partir de su símbolo, intensión y extensión.

- *Símbolo.* Palabra o imágenes que representan un concepto
- *Intensión.* La definición del concepto
- *Extensión.* El conjunto de ejemplos a los que se aplica el concepto

Obtención de conceptos a partir de una lista de categorías de conceptos

La creación de un modelo conceptual se comienza preparando una lista de conceptos idóneos. Contiene muchas categorías comunes que vale la pena tener en cuenta, sin que importe el orden de importancia.

En la siguiente tabla se muestra la lista de conceptos del dominio del problema, del manejo de la información de las publicaciones.

Tabla 2.2.2.1. Muestra las categorías de conceptos empleadas en el sistema actual.

Categoría del concepto	**Ejemplo**
Objetos físicos o tangibles	Computadora
Especificaciones, diseño o descripciones de cosas.	Descripción de los artículos
Lugares	Centros de cómputo
Transacciones	Consultar, agregar, eliminar, modificar
Línea o renglón de elemento de transacción	Artículos
Papel de la persona	Lector, administrador
Otros sistemas de cómputo o electromecánicos externos al sistema	Navegador de internet
Conceptos de nombres abstractos	Navegar, intranet
Organizaciones	Departamento de Ingeniería Petrolera
Eventos	Consulta, agrega, borra, cambia
Procesos (a menudo no están representados como conceptos, pero pueden estarlo)	Consulta tema de artículo, agrega artículo, elimina artículo, cambia datos de artículos
Reglas y políticas	Políticas de Publicaciones
Registros de trabajo, de contrato de asuntos legales	Contrato de Empleo
Manuales, libros	Manual de usuario

Asociaciones

La *asociación* es una relación entre dos conceptos que indican alguna conexión significativa e interesante entre ellos, como se muestra en la figura 2.2.2.2.. Las asociaciones que valen la pena mencionar suelen incluir el conocimiento de una relación que ha de preservarse durante algún tiempo: puede tratarse de milisegundos o años según el contexto. En la figura se puede observar la notación de las asociaciones en el UML (*Unified Modeling Language*).

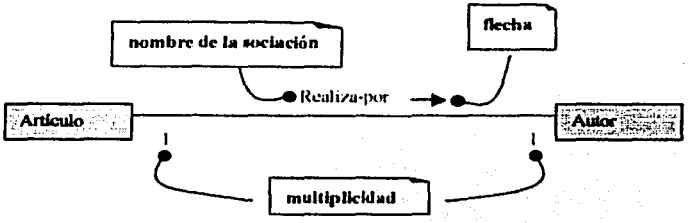


Figura 2.2.2.2. Notación de las asociaciones en el lenguaje UML

La flecha de dirección de la lectura no tiene un valor semántico tan sólo es una ayuda para leer el diagrama.

Papeles

A los extremos de una asociación se les nombra *papeles*. Éstos pueden tener:

- *Nombre*. Se asigna nombre a una asociación basándose en el formato NombredeTipo-FraseNominal-NombredeTipo. Los nombres de tipo de las asociaciones comienzan con una mayúscula, e indican una acción.
- *Expresión de multiplicidad*. La multiplicidad define cuántas instancias de un tipo A pueden asociarse con una instancia del tipo B en determinado momento. En la figura 2.2.2.3 se pueden observar las expresiones de multiplicidad.
- *Navegabilidad*. Indica la posibilidad de navegar unidireccionalmente en una asociación, desde los objetos fuente hasta la clase destino.

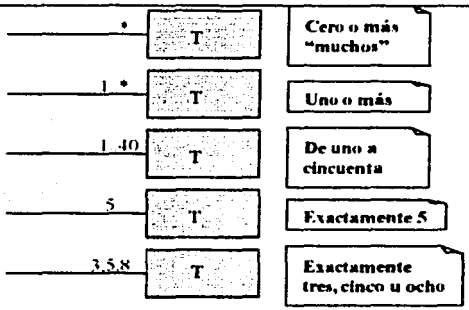


Figura 2.2.2.3. Valores de la multiplicidad

Agregación de los atributos

Es necesario identificar los atributos de los conceptos que se necesitan para satisfacer los requerimientos de información de los casos de uso en cuestión.

Modelo Conceptual para el manejo de la información de la Subdirección de Desarrollo Profesional de PEMEX y de la Sección de Ingeniería Petrolera de la DEPFI-UNAM.

El modelo conceptual como el que se muestra en la figura 2.2.2.4 es una representación de cosas reales, no de componentes del software. Cualquier información concerniente a los atributos ha de interpretarse dentro del contexto de entidades del mundo real. El modelo del conceptual, contiene lo que se ha entendido sobre el negocio y sirve de punto de partida para las clases clave del dominio.

Modelo Conceptual del sistema

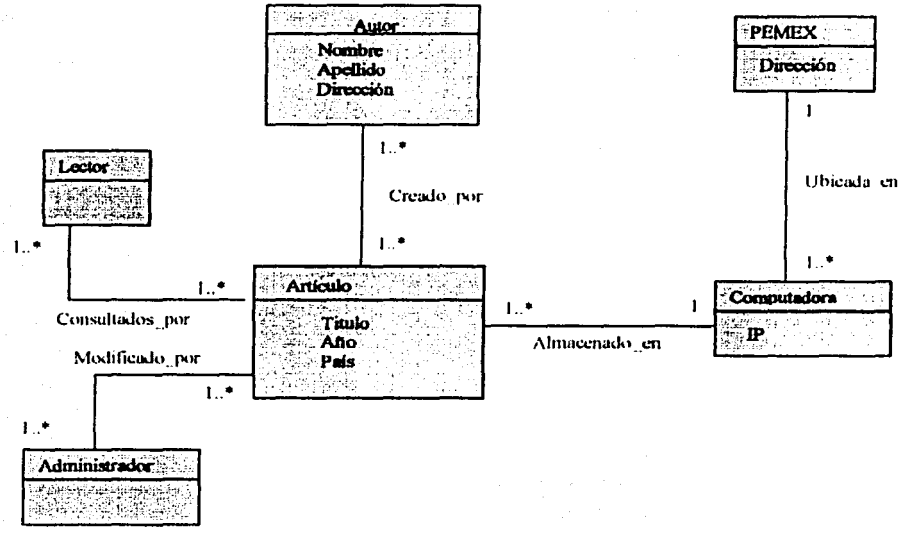


Figura 2.2.2.4. Valores de la multiplicidad

2.2.3 Comportamiento de los sistemas

El *diagrama de la secuencia* de un sistema muestra gráficamente los eventos que fluyen de los actores al sistema. La creación de los diagramas de la secuencia de un sistema forma parte de la investigación para conocer el sistema; se incluye, pues, dentro del análisis. Su creación depende de la formulación previa de los casos de uso.

El *evento* de un sistema es un hecho de entrada que un actor produce en un sistema.

Para identificar los eventos de un sistema es necesario tener una idea clara de lo que se quiere al escoger su frontera. Por lo que respecta al desarrollo del software, la frontera de un sistema suele seleccionarse para que sea el sistema de software (y, posiblemente, del hardware). dentro de este contexto, un evento del sistema es un hecho externo que estimula directamente el software.

El UML (*Unified Modeling Language*) ofrece una notación con los diagramas de la secuencia que muestran gráficamente los eventos que pasan de los actores al sistema.

A continuación se muestran los diagramas de secuencia para el manejo de la información de la Subdirección de Desarrollo Profesional de PEMEX y de la Sección de Ingeniería Petrolera de la DEPFI-UNAM.

En la figura 2.2.3.1 se muestra el diagrama de secuencia que ilustra las operaciones que se deben realizar entre el actor y el sistema para ingresar un nuevo artículo.

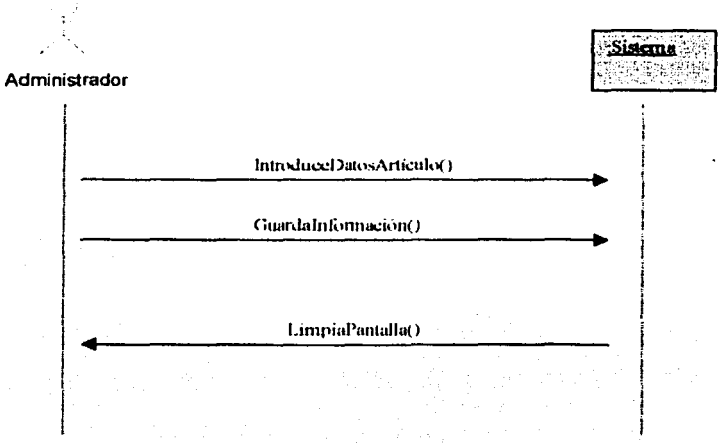


Figura 2.2.3.1. Diagrama de la secuencia del sistema al agregar un artículo nuevo.

En la figura 2.2.3.2 se muestra el diagrama de secuencia que ilustra las operaciones que se deben realizar entre el actor y el sistema para borra un artículo.

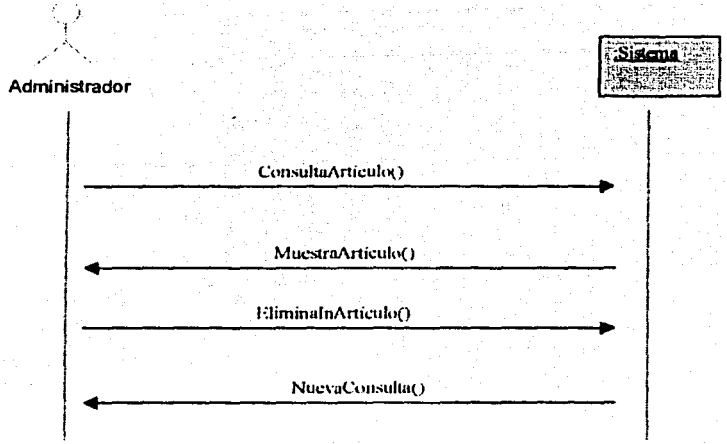


Figura 2.2.3.2. Diagrama de la secuencia del sistema al eliminar un artículo.

En la figura 2.2.3.3 se muestra el diagrama de secuencia que ilustra las operaciones que se deben realizar entre el actor y el sistema para actualizar un artículo.

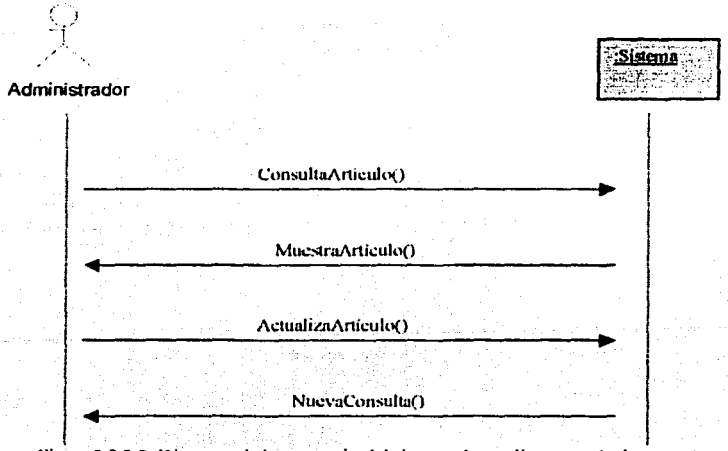


Figura 2.2.3.3. Diagrama de la secuencia del sistema al actualizar un artículo.

En la figura 2.2.3.4 se muestra el diagrama de secuencia que ilustra las operaciones que se deben realizar entre el actor y el sistema para consultar un artículo.

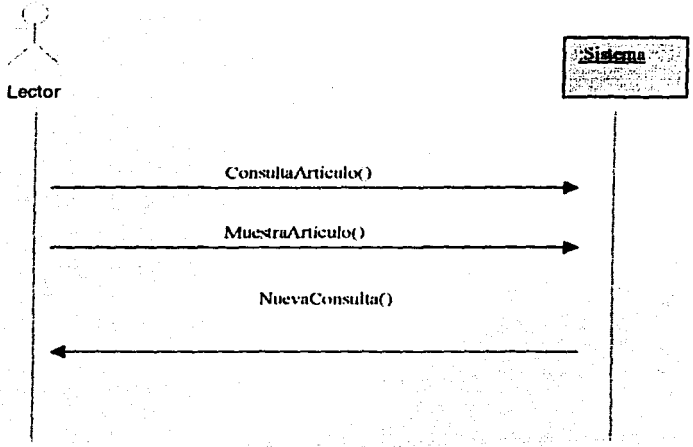


Figura 2.2.3.4. Diagrama de la secuencia del sistema al consultar un artículo.

2.3 Plan de pruebas del Sistema

En la última fase del ciclo de desarrollo del sistema, se inicia un plan de trabajo de pruebas las cuales verifican que el sistema proporciona de verdad la funcionalidad descrita en los casos de uso y que satisface los requisitos del sistema. Un *caso de prueba* define una colección de entradas, condiciones de ejecución y resultados. Muchos de los *casos de prueba* se pueden obtener directamente de los casos de uso y por tanto tenemos una independencia de traza entre el caso de prueba y el caso de uso correspondiente. Esto significa que se verificará que el sistema puede hacer lo que los usuarios quieren que haga, es decir, que ejecute los casos de uso. Cualquiera que haya probado software en el pasado en realidad ha probado casos de uso. Lo novedoso y diferente del proceso de desarrollo que aquí se sigue es que la prueba puede planificarse al principio del ciclo de desarrollo. Tan pronto como se hayan capturado los casos de uso, es posible especificar los *casos de prueba* ("pruebas de caja negra") y determinar el orden en el cual realizarlos, integrarlos y probarlos. Más adelante, según se vayan realizando los casos de uso en el diseño, pueden detallarse las pruebas de los casos de uso (pruebas de "caja blanca"). Cada forma de ejecutar un caso de uso —es decir, cada camino a través de la realización de un caso de uso— es un caso de prueba candidato

Las pruebas de caja negra se refiere a las pruebas que se llevan sobre la interfaz del software. Los *casos de prueba* pretenden demostrar que las funciones del software son operativas que la entrada se acepta de forma adecuada y que se produce una salida correcta, así como la integridad de la información externa (por ejemplo: archivos de datos) se mantiene. Una prueba de la caja negra examina algunos aspectos del modelo fundamental del sistema sin tener mucho en cuenta la estructura lógica del software.

La prueba es conjunto de actividades que se pueden planificar por adelantado y llevarse a cabo sistemáticamente con la finalidad de asegurar que los objetivos principales del sistema sean alcanzados.

Las pruebas se llevarán a cabo en un orden determinado por la forma en que se crea la información y se utiliza. Definiremos dos categorías:

1. Funcionalidades que generan información.
2. Funcionalidades que utilizan información.

CASO DE PRUEBA. Registro de los Datos de un Artículo por Internet.

Fundamentos. Es necesario contar con los artículos de las publicaciones PEMEX antes de poder usarlos en otros casos de uso. El sistema ofrece la posibilidad de guardar la información relacionada con los artículos.

Precondiciones. Entrar mediante un navegador de internet a la página de Mantenimiento de los Artículos de PEMEX.

ENTRADA	RESULTADOS ESPERADOS	CASOS DE USO
Seleccionar del Menú Principal, la opción Registra Artículo.	El sistema desplegará la página del agrega artículos.	Agrega un artículo nuevo
Llenar los datos del Artículo en la plantilla presentada. • Correctos • Incorrectos	El sistema verifica la información para: • Realizar la inserción • No realizar la inserción y mostrará el mensaje de error	Agrega un artículo nuevo
Se regresa a la página principal y se selecciona la opción de consulta, para aplicar el caso de prueba de consulta de información del Artículo para ver los cambios realizados.		Consulta de un Artículo por Título.

CASO DE PRUEBA. Consulta de Información de Artículos por Internet.

Fundamento. La información almacenada sobre los artículos publicados, es valiosa siempre y cuando tenga la posibilidad de ser accedida de manera fácil y sencilla por parte de la comunidad. Se debe garantizar que los mecanismos de búsqueda funcionen con base en lo propuesto.

Precondiciones: Entrar mediante un navegador de internet a la publicación mensual del Boletín y seleccionar la opción de Artículos.

ENTRADAS	RESULTADO ESPERADO	CASOS DE USO
Seleccionar Consulta de Artículo PEMEX del menú.	Vista de la página Consulta de Artículos.	
Seleccionar Consulta de Artículo bajo el criterio de Título. • Seleccionar Título.	Vista de lista de Títulos. Vista del Título seleccionado.	Consulta de un Artículo.
Seleccionar Consulta de Artículos bajo el criterio de Autor. • Seleccionar Autor.	Vista de lista de Autores. Vista del Autor seleccionado.	Consulta de un Artículo.

CASO DE PRUEBA: Modificación de los Datos de un Artículo por Internet.

Fundamento. El sistema permite la modificación de la información correspondiente a los artículos almacenados.

Precondiciones. Entrar mediante un navegador de internet a la página de Mantenimiento de los Articulos de PEMEX.

ENTRADAS	RESULTADOS ESPERADOS	CASOS DE USO
Carga la página para la Modificación de los Datos.	El sistema desplegará la ventana de consulta de un artículo.	Actualizar un artículo.
Realizar la consulta de los datos, ya sea por titulo del artículo o por nombre de autor.	El sistema desplegará la información concerniente a la consulta realizada.	Consulta un artículo
Modificar los datos deseados y confirmar la operación. • Datos correctos. • Datos incorrectos	El sistema: • Actualizará la información • No actualizará la información y mostrará el mensaje de error.	Actualizará un artículo.
Se regresa a la página principal y se selecciona la opción de consulta, para aplicar el caso de prueba de consulta de información del Artículo para ver los cambios realizados.		Consulta de un Artículo.

CASO DE PRUEBA. Eliminación de los Datos de un Artículo por Internet.

Fundamento. El sistema permite la eliminación de la información correspondiente a los artículos almacenados.

Precondiciones. Entrar mediante un navegador de internet a la página de Mantenimiento de los Artículos de PEMEX.

ENTRADAS	RESULTADOS ESPERADOS	CASOS DE USO
Carga la página para la Modificación de los Datos.	El sistema desplegará la ventana de consulta de un artículo.	Actualizar un artículo.
Realizar la consulta de los datos, ya sea por título del artículo o por nombre de autor.	El sistema desplegará la información concerniente a la consulta realizada.	Consulta un artículo.
Eliminar los datos deseados y confirmar la operación. • Datos correctos.	El sistema: • Elimina la información. • No elimina la información y mostrará el mensaje de error.	Eliminación de un artículo.
Se regresa a la página principal y se selecciona la opción de consulta, para aplicar el caso de prueba de consulta de información del Artículo para ver los cambios realizados.		Consulta de un Artículo.

CAPÍTULO 3

DISEÑO DEL SISTEMA

3.1 Introducción

En el *diseño* modelamos el sistema y encontramos su forma para que soporte todos los requisitos. Una "entrada esencial" en el diseño es el resultado del análisis, esto es, el *modelo conceptual*. El modelo conceptual proporciona una comprensión detallada de los requisitos. Y lo que es más importante, impone una estructura del sistema que debemos esforzarnos por conservar lo más fielmente posible cuando demos forma al sistema.

En concreto, los propósitos del diseño son:

- ✓ Adquirir una comprensión en profundidad de los aspectos relacionados con los requisitos no funcionales.
- ✓ Crear una entrada apropiada y un punto de partida para actividades de implantación subsiguientes capturando los requisitos o subsistemas individuales, interfaces y clases.
- ✓ Ser capaces de descomponer los trabajos de implantación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo, teniendo en cuenta la posible concurrencia. Esto resulta útil en los casos en que la descomposición no pueda ser hecha basándose en los resultados de la captura de requisitos (incluyendo el modelo de casos de uso) o análisis (incluyendo el modelo conceptual) Un ejemplo podrían ser aquellos casos en los que la implementación de estos resultados no es directa.
- ✓ Ser capaces de visualizar y reflexionar sobre el diseño utilizando una notación común.
- ✓ Crear una abstracción sin costuras de la implantación del sistema, en el sentido de que la implantación es un refinamiento directo del diseño que rellena lo existente sin cambiar la estructura. Esto permite la utilización de tecnologías como la generación de código y la ingeniería de "ida y vuelta" entre el diseño y la implantación. Esto contribuye a una arquitectura estable y sólida y a crear un plano del modelo de implantación. Más tarde, durante la fase de construcción, cuando la arquitectura es estable y los requisitos están bien entendidos, el centro de atención se desplaza a la implantación.

A continuación en la figura 3.1.1 se muestra la gráfica de los pasos del proceso unificado de desarrollo de un sistema, donde se puede observar la *fase de diseño*, dentro del proceso de desarrollo.

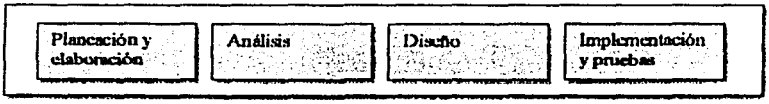


Figura 3.1.1. Pasos en el proceso unificado del desarrollo de un sistema

3.2 Artefactos de la fase de Diseño

La siguiente figura 3.2.1, muestra las dependencias de los artefactos durante la fase de diseño y construcción. Los que están señalados con una flecha, corresponden a los artefactos que se deberán construir en la fase de diseño.

A continuación se da una descripción breve de los artefactos que se deberán construir en la fase de diseño.

- **Diagramas de interacción.** Se trata de un término genérico que se aplica a varios tipos de diagramas que hacen hincapié en las interacciones entre objetos.
- **Diagramas de clase de diseño.** Es un diagrama que muestra un conjunto de clases, interfaces y las relaciones entre éstos; los diagramas de clase muestran el diseño de un sistema desde un punto estático; un diagrama que muestra una colección de elementos estáticos.
- **Diagramas de paquete de arquitectura.** Es un diagrama que muestra un conjunto de paquetes. Un paquete es un mecanismo de propósito general para organizar elementos en grupos.
- **Esquema de base de datos.** Diagrama entidad relación, para modelar las bases de datos del sistema.

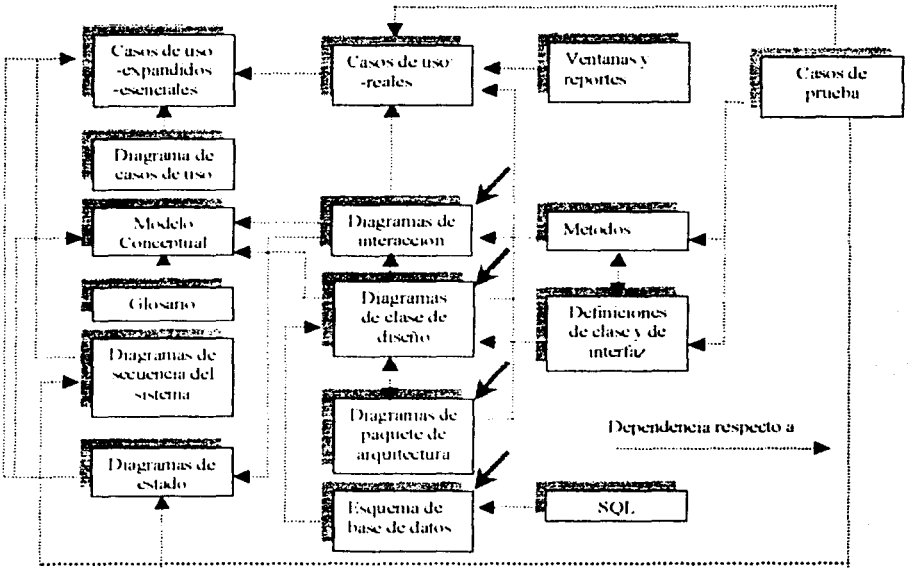


Figura 3.2.1. Dependencias de los artefactos obtenidos durante la fase de diseño e implantación.

En la tabla 3.2.1 se muestran los documentos utilizados en la fase de diseño.

Tabla 3.2.1. Documentos utilizados en la etapa de diseño.

Diseño	Documentos relacionados
Identificar la arquitectura del sistema	Diagrama de arquitectura del sistema.
Diseño de clases	Diagrama de clases con el detalle de la implantación.
Diseño de paquetes	Diagrama de paquetes de arquitectura con el detalle de la implantación.
Diseño de flujos de interacción	Diagrama de interacción con el detalle de las operaciones más importantes del sistema.
Modelado Entidad /Relación	Diagrama entidad/relación

3.3 Arquitectura

¿Por qué es necesaria una Arquitectura?

La arquitectura describe los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. Un sistema software grande y complejo requiere una arquitectura para que los desarrolladores puedan progresar hasta tener una visión común. Se necesita una arquitectura para:

- Comprender el sistema.
- Organizar el desarrollo
- Fomentar la reutilización
- Hacer evolucionar el sistema.

Comenzamos determinando un diseño de alto nivel para la arquitectura, a modo de una arquitectura en capas.

3.3.1 Arquitectura de Tres Capas

Una arquitectura común de los sistemas de información que abarca una interfaz para el usuario y el almacenamiento persistente de datos se conoce con el nombre de *arquitectura de tres capas*. He aquí una descripción clásica de las capas, como la que se muestra en la figura 3.3.1.1.

1. **Presentación.** Ventanas, informes, etcétera.
2. **Lógica de aplicaciones.** Tareas y reglas que rigen el proceso.
3. **Almacenamiento.** Mecanismo de almacenamiento persistente.

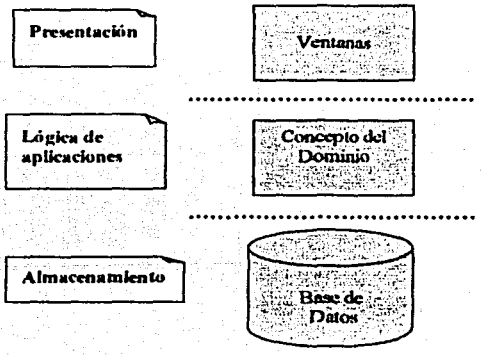


Figura 3.3.1.1. Arquitectura de tres capas

Capa de presentación

El nivel cliente que en ocasiones se denomina nivel de presentación, es donde se muestra la información al usuario y en donde se aceptan los datos de entrada para su procesamiento.

Capa lógica de aplicaciones

El nivel de aplicaciones es donde se lleva a cabo todo el procesamiento, de acuerdo con la lógica implementada por el sistema. Aquí es donde se aplican las reglas del negocio, donde se comprueba la integridad de los datos y donde se realiza el procesamiento complejo dictado por las necesidades del sistema. Este nivel constituye el motor del enfoque de tres niveles.

Capa almacenamiento

El nivel de datos que con frecuencia se denomina módulo de servicio, desempeña un papel importante en el almacenamiento de la información para satisfacer las solicitudes de los dos niveles anteriores. En numerosas ocasiones, aunque no en todas, es una base de datos relacional optimada especificamente para recibir datos del nivel de aplicaciones y devolver esa información de nuevo a dicho nivel, a medida que los usuarios interactúan con los sistemas.

La calidad tan especial de la arquitectura de tres capas consiste en aislar la lógica de la aplicación y en convertirla en una capa intermedia bien definida y lógica del software. En la capa de presentación se realiza relativamente poco procesamiento de la aplicación; las ventanas envían a la capa intermedia peticiones de trabajo. Y éste se comunica con la capa de almacenamiento del extremo posterior.

3.3.2 Arquitectura de Dos Capas

La arquitectura de tres capas contrasta con el diseño de dos capas, donde —por ejemplo— colocamos la lógica de aplicaciones dentro de las definiciones de ventana que leen y escriben directamente en una base de datos; no hay una capa intermedia que separe la lógica. Una de sus desventajas es la imposibilidad de representar la lógica en componentes aislados, lo cual impide reutilizar el software. No es posible distribuir la lógica de aplicaciones en una computadora diferente.

3.3.3 Arquitectura Multicapas Orientadas a Objetos

Una arquitectura multicapas que se adecue a los sistemas de información orientados a objetos incluye la división de las responsabilidades que encontramos en la arquitectura clásica de tres capas. Las responsabilidades se asignan a los objetos de software

En un diseño orientado a objetos, la capa de la lógica de aplicaciones se divide en otras menos densas. Esta arquitectura organiza a partir de las clases de software, como se

muestra en la figura 3.3.3.1. La capa de la lógica de aplicaciones está constituida por las siguientes capas.

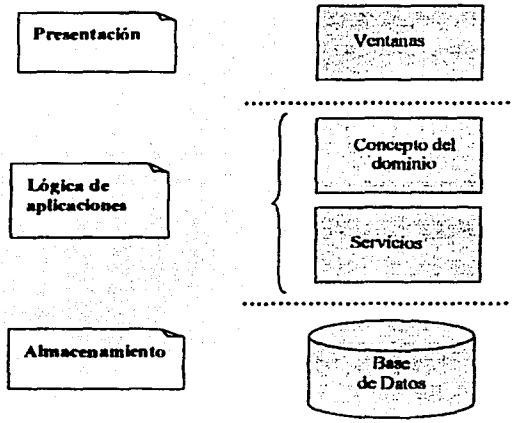


Figura 3.3.3.1. Descomposición de la capa de lógica de aplicaciones en estratos o subcapas más detalladas.

Cuando vemos la arquitectura desde la perspectiva de una descomposición más detallada, podemos prescindir de la designación "arquitectura de tres capas" hablar en cambio de *arquitectura multicapas*; en ellas está implícita la capa intermedia de la lógica de aplicaciones.

Es posible agregar más capas y descomponer ulteriormente las ya existentes. Así, la capa de Servicios puede dividirse en servicios de alto y bajo nivel (por ejemplo, generación de informes frente a entrada/salida).

Motivos para utilizar la arquitectura multicapas

Entre los motivos por los cuales se recurre a la arquitectura multicapas se cuentan los siguientes:

- Aislamiento de la lógica de aplicaciones en componentes independientes susceptibles de reutilizarse después en otros sistemas.
- Distribución de las capas en varios nodos físicos de cómputo y en varios procesos. Esto puede mejorar el desempeño, la coordinación y el compartir la información en un sistema de cliente-servidor.
- Asignación de los diseñadores para que construyan determinadas capas; por ejemplo, un equipo que trabaje exclusivamente en la capa de presentación. Y así se brinda soporte a los conocimientos especializados en las habilidades de desarrollo y también a la capacidad de realizar actividades simultáneas en equipo.

3.4 Paquetes

Un paquete es un mecanismo de propósito general para organizar elementos en grupos. Los paquetes ayudan a organizar los elementos en los modelos con el fin de comprenderlos más fácilmente. Los paquetes también permiten controlar el acceso a sus contenidos para controlar las líneas de separación de la arquitectura del sistema.

UML (*Unified Modeling Language*) proporciona una representación gráfica de los paquetes, como se muestra en la Figura 3.4.1.

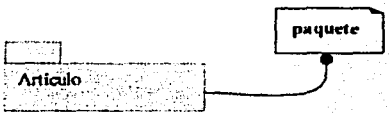


Figura 3.4.1. Representación de un paquete en Lenguaje UML.

Nombres

Cada paquete ha de tener un nombre que lo distinga de otros paquetes. Un *nombre* es una cadena de texto. El nombre solo se denomina *nombre simple*; un *nombre de camino* consta del nombre del paquete precedido por el nombre del paquete en el que se encuentra, si es el caso, como se muestra en la figura 3.4.2.

Figura 3.4.2. Paquete simple y extendido

Elementos contenidos

Un paquete puede contener otros elementos, incluyendo clases, interfaces, casos de uso, diagramas e incluso otros paquetes. La posesión es una relación compuesta, lo que significa que el elemento se declara en el paquete. Si el paquete se destruye, el elemento es destruido. Cada elemento pertenece exclusivamente a un único paquete. Gráficamente uno de los elementos que contiene un paquete se muestran como en la figura 3.4.3.

Un paquete forma un espacio de nombres, lo que quiere decir que los elementos de la misma categoria deben tener nombres únicos en el contexto de su paquete contenedor. Por ejemplo, no se puede tener dos clases llamadas *Articulo* dentro del mismo paquete, pero se puede tener una clase *Articulo* en el paquete P1 y otra clase (diferente) llamada *Articulo* en

el paquete P2. Las clases P1::Artículo y P2::Artículo son, de hecho, clases diferentes y se pueden distinguir por sus nombres de camino. Diferentes tipos de elementos pueden tener el mismo nombre.

Esta semántica de posesión convierte a los paquetes en un mecanismo importante para enfrentarse al crecimiento. Sin los paquetes, se acabaría con grandes modelos planos en los que todos los elementos deberían tener nombres únicos (una situación inmanejable), especialmente cuando se compran clases y otros elementos desarrollados por varios equipos. Los paquetes ayudan a controlar los elementos que componen un sistema mientras evolucionan a diferentes velocidades a lo largo del tiempo.

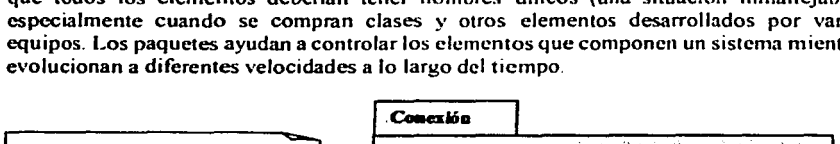


Figura 3.4.3. Elementos contenidos

Importación y exportación

Supongamos que tenemos dos clases llamadas A y B que se conocen mutuamente. Al ser compañeras, A puede ver a B y B puede ver a A, así que cada una depende de la otra. Dos clases tan sólo constituyen un sistema trivial, así que realmente no se necesita ningún tipo de empaquetamiento.

Ahora imaginemos que tenemos unos cientos de clases que se conocen entre ellas. No hay límites para la intricada red de relaciones que se puede establecer. Además, no hay forma de comprender un grupo de clases tan grande y desorganizado. Éste es un problema real para grandes sistemas (el acceso simple y no restringido no permite el crecimiento). Para estas situaciones se necesita algún tipo de empaquetamiento controlado para organizar las abstracciones.

Ahora imaginemos que en lugar de esto se coloca A en un paquete y B en otro, teniendo cada paquete conocimiento del otro. Supongamos también que A y B se declaran ambas como públicas, en sus respectivos paquetes. Esta es una situación muy diferente. Aunque A y B son ambas públicas, ninguna puede acceder a la otra porque sus paquetes contenedores forman un muro opaco. Sin embargo, si el paquete de A importa al paquete de B, A puede ahora ver a B, aunque B no puede ver a A. La importación concede un permiso de un solo sentido para que los elementos de un paquete se conecten a los elementos de otro. Al empaquetar las abstracciones en bloques significativos y luego controlar los accesos mediante la importación, se puede controlar la complejidad de tener un gran número de abstracciones.

Las partes públicas de un paquete son sus exportaciones. Las partes que exporta un paquete son sólo visibles al contenido de aquellos paquetes que lo importan explícitamente.

3.4.1 Diagramas de paquetes

Los paquetes nos permiten describir la arquitectura de un sistema con la notación UML (*Unified Modeling Language*). En la figura 3.4.1.1 se muestra un agrupamiento lógico mediante los diagramas de un paquete de UML (*Unified Modeling Language*). A este tipo de diagramas podemos llamarlo *diagrama de paquetes de la arquitectura*. A continuación se describe brevemente cada paquete:

1. **Paquete de Presentación o interfaz de usuario.** Aquí es donde se muestra la información al usuario.
2. **Paquete del Dominio.** Aquí es donde se representan los conceptos del dominio del problema.
3. **Paquete de Servicios.** Aquí se encuentran los servicios ofrecidos al cliente, por ejemplo: informes, interfaces de bases de datos, seguridad, comunicación entre procesos, creación de bibliotecas, acceso a base de datos y archivos, etc.

La línea punteada con flecha indica si el paquete conoce el acoplamiento con otro. Nótese en la figura 3.4.1.1 que el paquete de dominio no tiene dependencia con la capa de presentación.

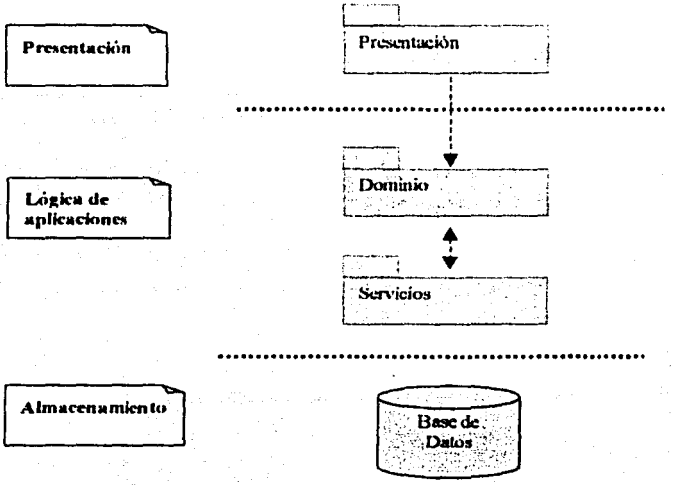
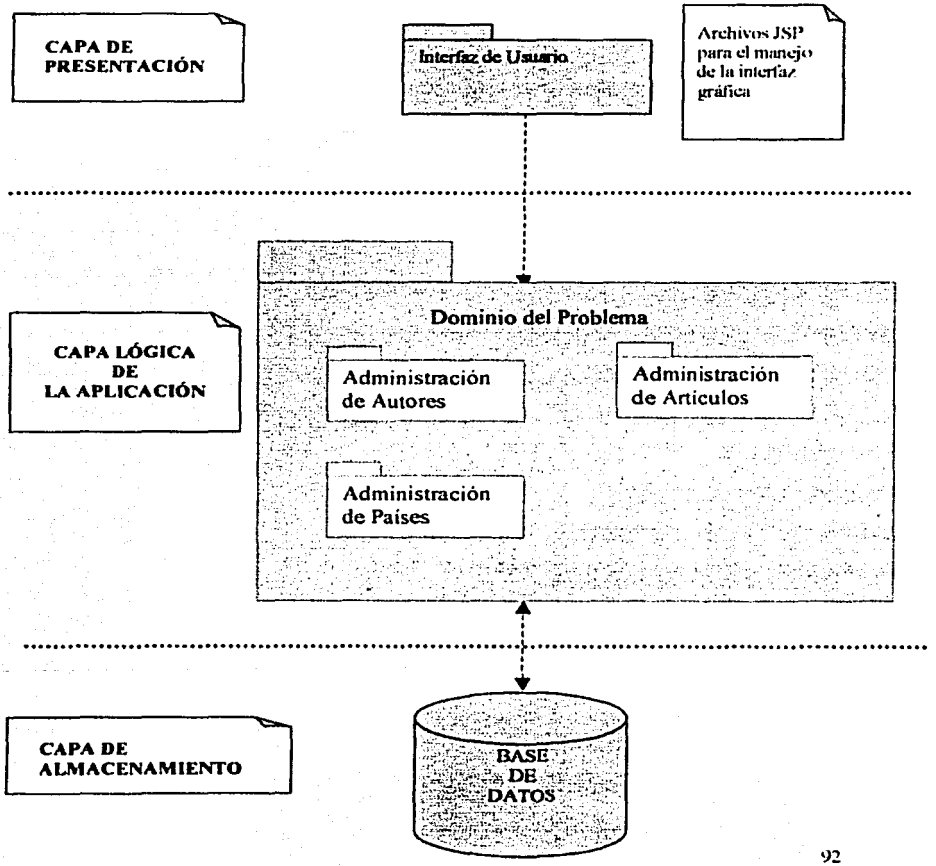


Figura 3.4.1.1. Unidades arquitectónicas en términos de los paquetes de UML.

En el diagrama 3.4.1.2, se muestran los paquetes obtenidos en el diseño del sistema para el manejo de las publicaciones de los artículos PEMEX.

Al observar el diagrama se puede distinguir una arquitectura de tres capas. Para la capa lógica de aplicaciones se diseñaron algunos paquetes que contienen las clases del dominio del problema. La capa de presentación muestra los archivos Java Server Pages que se diseñaron para el manejo de la interfaz gráfica.

3.4.1.2. Diagrama de paquetes para el sistema.



3.5 Clases

Una *clase* es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. Gráficamente una clase se representa como un rectángulo. En la figura 3.5.1 se pueden observar los elementos que integran una clase. Dentro de la estructura de una clase se pueden especificar:

- Nombre
- Atributos
- Operaciones
- Responsabilidades

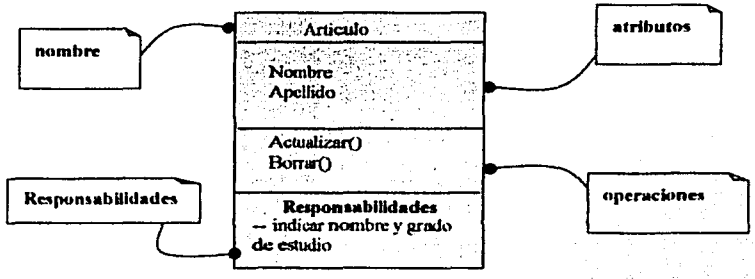


Figura 3.5.1. Clases

Nombres

Cada clase ha de tener un nombre que la distinga de otras clases. Un nombre es una cadena de texto. Ese nombre sólo se denomina nombre simple; un nombre de camino consta del nombre de la clase precedido por el nombre del paquete en el que se encuentra,

Atributos

Un atributo es una propiedad de una clase identificada con un nombre que describe un rango de valores que pueden tomar las instancias de la propiedad. Una clase puede tener cualquier número de atributos o no tener ninguno. Un atributo representa alguna propiedad del elemento que se está modelando que es compartida por todos los objetos de esa clase.

Operaciones

Una operación es la implantación de un servicio que puede ser requerido a cualquier objeto de la clase para que muestre un comportamiento. En otras palabras, una operación es una abstracción de algo que se puede hacer a un objeto y que es compartido por todos los objetos de la clase Una clase puede tener cualquier número de operaciones o ninguna. Gráficamente, las operaciones se listan en un compartimiento justo debajo de los atributos de la clase. Las operaciones se pueden representar mostrando sólo sus nombres

Responsabilidades

Una responsabilidad es un contrato o una obligación de una clase. A un nivel más abstracto, los atributos y operaciones son simplemente las características por medio de las cuales se llevan a cabo las responsabilidades de la clase.

3.5.1 Relaciones entre clases

Al realizar abstracciones, uno se da cuenta que muy pocas clases se encuentran aisladas. En vez de ello, la mayoría colaboran con otras de varias maneras. En el modelo orientado a objetos hay tres tipos de relaciones especialmente importantes:

a) Dependencias
b) Generalizaciones
c) Asociaciones

Dependencias

Una *dependencia* es una relación de uso que declara que un cambio en la especificación de un elemento, puede afectar a otro elemento que la utiliza, pero no necesariamente a la inversa. Gráficamente, una dependencia se representa como una línea discontinua dirigida hacia el elemento del cual se depende, como se observa en la figura 3.5.1.1. Las dependencias se usarán cuando se quiere indicar que un elemento utiliza a otro.

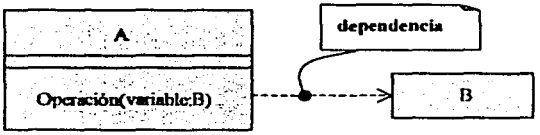


Figura 3.5.1.1. Dependencias

Generalizaciones

Una generalización es una relación entre un elemento general (llamado superclase o padre) y un caso más específico de ese elemento (llamado subclase o hijo). La generalización significa que los objetos hijos se pueden emplear en cualquier lugar que pueda aparecer el padre. Una clase hija hereda las propiedades de sus clases padres, especialmente sus atributos y operaciones. A menudo (no siempre) el hijo añade atributos y operaciones a los que hereda de sus padres. Una operación de un hijo con la misma signatura que una operación del padre redefine la operación del padre; esto se conoce como *polimorfismo*. Gráficamente, la generalización se presenta como una línea dirigida continua, con una gran punta de flecha vacía, apuntando al padre, como se muestra en la figura 3.5.1.2. Las generalizaciones se utilizarán cuando se quieran mostrar relaciones padre-hijo.

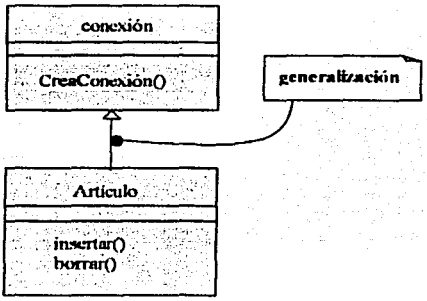


Figura 3.5.1.2. Generalización

Asociación

Una *asociación* es una relación estructural que especifica que los objetos de un elemento están conectados con los objetos de otro. Dada una asociación entre dos clases, se puede navegar desde un objeto de una clase hasta un objeto de la otra clase y viceversa. Es legal que ambos extremos de una asociación estén conectados a la misma clase. Esto significa que dado un objeto de la clase, se puede conectar con otros objetos de la misma clase. Aparte de esta forma básica, hay cuatro adornos que se aplican a las asociaciones.

a) Nombre.
b) Rol.
c) Multiplicidad.
d) Agregación.

a) Nombre

Una asociación puede tener un nombre que se utiliza para describir la naturaleza de la relación. Para que no haya ambigüedad en su significado, se puede dar una dirección al nombre por medio de una flecha que apunte en la dirección en la que se pretende que se lea el nombre, como se muestra en la figura 3.5.1.3.

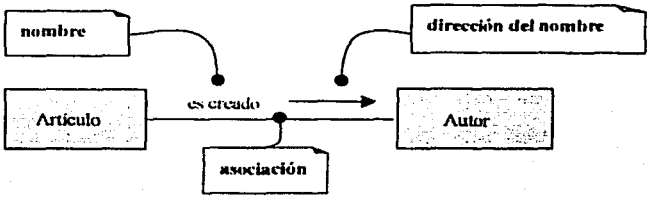


Figura 3.5.1.3. Nombre de asociaciones.

b) Rol

Cuando una clase participa en una asociación tiene un rol específico que juega en la asociación; un rol es simplemente la cara que la clase de un extremo de la asociación presenta a la clase del otro extremo. En la figura 3.5.1.4 puede observarse el rol para una clase Artículo

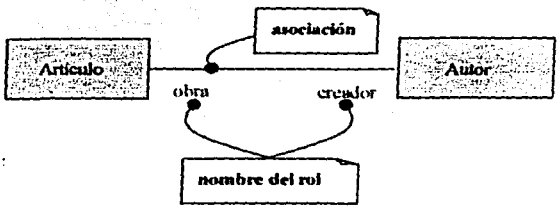


Figura 3.5.1.4. Roles

c) Multiplicidad

Una asociación representa una relación estructural entre objetos. Cuando se indica una multiplicidad en un extremo de una asociación, se está especificando que para cada objeto de la clase en el extremo opuesto debe haber tantos objetos como en este extremo. Se puede indicar una multiplicidad de exactamente uno (1), cero o uno (0..1), muchos (0..*), o uno o más (1..*). Incluso se puede indicar un número exacto. En la figura 3.5.1.5 puede observarse la multiplicidad de entre las clases Artículo y Autor.

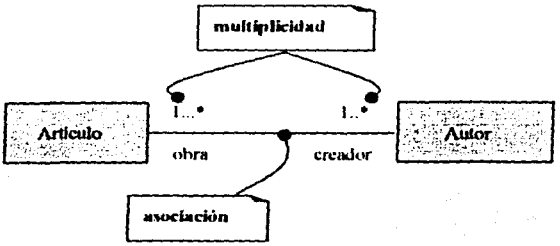


Figura 3.5.1.5 Multiplicidad

d) Agregación

Una asociación normal entre dos clases representa una relación estructural entre iguales, es decir, ambas clases están conceptualmente en el mismo nivel, sin ser alguna más importante que la otra. A veces, se desea modelar una relación "todo/parte", en la cual una clase representa una cosa grande (el "todo"), que consta de elementos más pequeños (las "partes"). Este tipo de relación se denomina agregación y se especifica añadiendo a una

asociación normal un rombo vacío en la parte del todo, como se muestra en la figura 3.5.1.6.

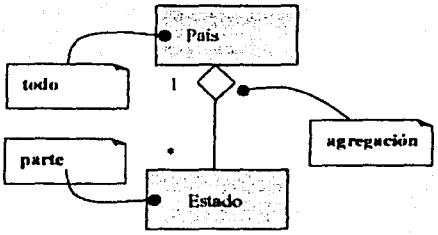


Figura 3.5.1.6. Agregación

3.5.2 Interfaces

Una *interfaz* es una colección de operaciones que se usa para especificar un servicio de una clase o de un componente.

Una interfaz bien estructurada proporciona una clara separación entre las vistas externa e interna de una abstracción, haciendo posible comprender y abordar una abstracción sin tener que sumergirse en los detalles de su implantación.

En el software es importante construir sistemas con una clara separación de intereses, de forma que al evolucionar el sistema, los cambios en una parte del sistema no se propaguen, afectando a otras partes del sistema. Una forma importante de lograr este grado de separación es especificar unas líneas de separación claras independientemente. Además, al elegir las interfaces apropiadas, se pueden tomar componentes estándar, bibliotecas para implantar esas interfaces, sin tener que construirlas uno mismo. Al ir descubriendo mejores implantaciones, se pueden reemplazar las viejas sin afectar a sus usuarios.

Nombre

Cada interfaz ha de tener un nombre que la distinga de otras interfaces. Un nombre es una cadena de texto. Ese nombre sólo se denomina nombre simple; un nombre de camino consta del nombre de la interfaz precedido por el nombre del paquete en el que se encuentra. Una interfaz puede dibujarse mostrando sólo su nombre, como se muestra en la figura 3.5.2.1.

Figura 3.5.2.1. Interfaz

Operaciones

Como una clase, una interfaz puede incluir cualquier número de operaciones.

Cuando se representa una interfaz en su forma normal como un círculo, por definición, se omite la visualización de estas operaciones. Sin embargo, si es importante para entender el modelado actual, se puede dibujar una interfaz como una clase estereotipada, listando sus operaciones en el compartimiento apropiado. Las operaciones se pueden representar sólo con su nombre o pueden extenderse para mostrar la signatura completa y otras propiedades, como se muestra en la figura 3.5.2.2.

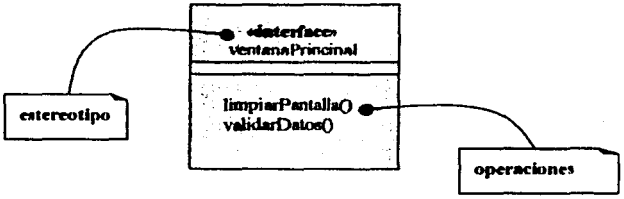


Figura 3.5.2.2. Operaciones

Relaciones

Al igual que una clase, una interfaz puede participar en relaciones de *generalización*, *asociación* y *dependencia*. Además, una interfaz puede participar en relaciones de *realización*. La realización es una relación semántica entre dos clasificadores en la que un clasificador especifica un contrato que el otro clasificador garantiza llevar a cabo. Gráficamente la realización se representa como se observa en la figura 3.5.2.3.

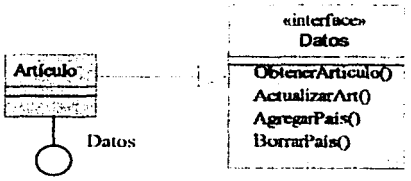


Figura 3.5.2.3. Relación de realización

3.5.3 Diagrama de Clases

Un *diagrama de clases* es un diagrama que muestra un conjunto de interfaces, colaboraciones y sus relaciones. Los diagramas de clases se utilizan para modelar la vista de diseño estática de un sistema. Esta vista soporta principalmente los requisitos

funcionales de un sistema, los servicios que el sistema debe proporcionar a sus usuarios finales. Gráficamente, un diagrama de clases es una colección de nodos y arcos, y describe gráficamente las especificaciones de las clases de software y de las interfaces (las de Java, por ejemplo) en una aplicación.

Los diagramas de clases contienen normalmente los siguientes elementos:

- Clases.
- Interfaces.
- Relaciones de dependencia, generalización y asociación.

Clases del diseño del sistema para el manejo de las publicaciones de los artículos PEMEX.

No existe un procedimiento inmediato que permita localizar las clases. Éstas suelen corresponder con sustantivos que hacen referencia al ámbito del sistema de información y que se encuentran en los documentos de las especificaciones de requisitos y los casos de uso.

Siguiendo la arquitectura de tres capas y partiendo del Modelo Conceptual obtenido en la fase de análisis, las clases se diseñaron de la siguiente manera:

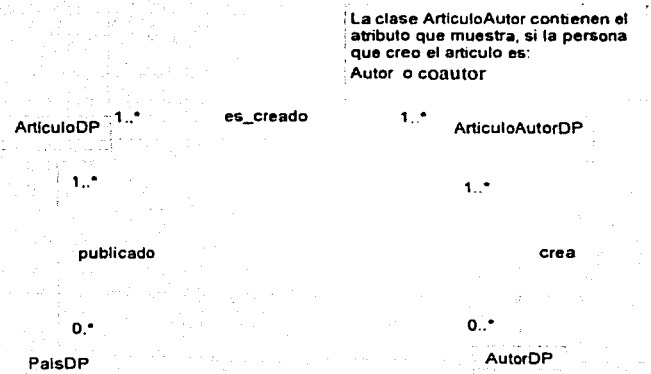
- *Clases del Dominio del problema.* Son aquellas clases que contienen únicamente la definición y declaración de sus atributos.
- *Clases de Interfaz de Usuario.* Son aquellas clases que contienen todos los métodos necesarios para llevar acabo la funcionalidad de la interfaces.
- *Clases de Manejo de datos.* Son aquellas clases que contienen los métodos para la manipulación de los datos(inserciones, actualizaciones, eliminación y consultas) en la base de datos.

La funcionalidad y creación de las interfaces, se diseñaron en los *Java Server Pages*, por tanto

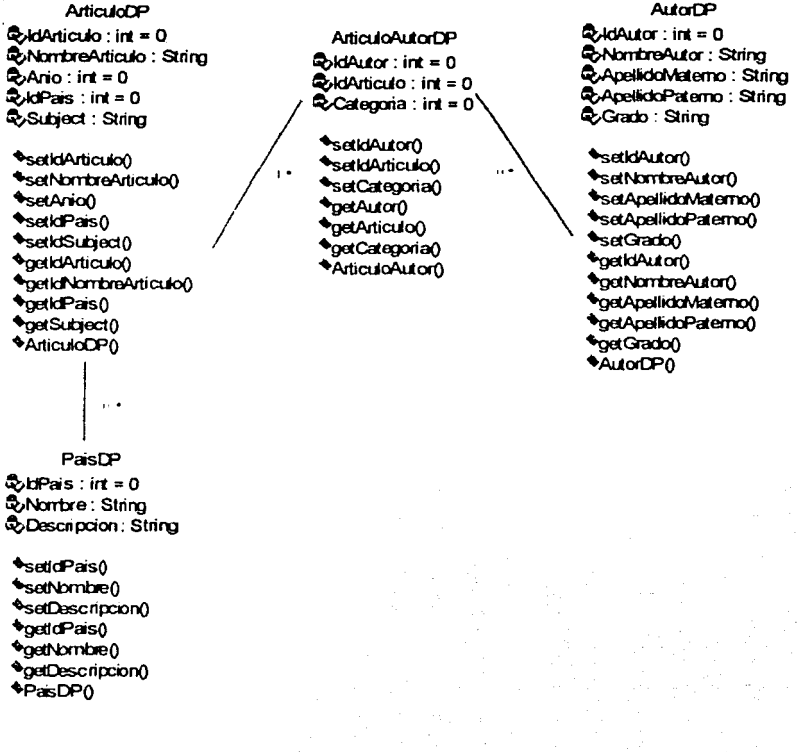
Para representar gráficamente las clases que se diseñaron para el sistema se crearon los siguientes diagramas:

- Diagrama general de clases. El cual se muestra en la figura 3.5.3.1
- Diagrama de clases del dominio del problema. Mostrado en la figura 3.5.3.2
- Diagrama de clases del manejo de datos. El cual se muestra en la figura 3.5.3.3
- Diagrama de interfaces de la capa de presentación. El cual se muestra en la figura 3.5.3.4

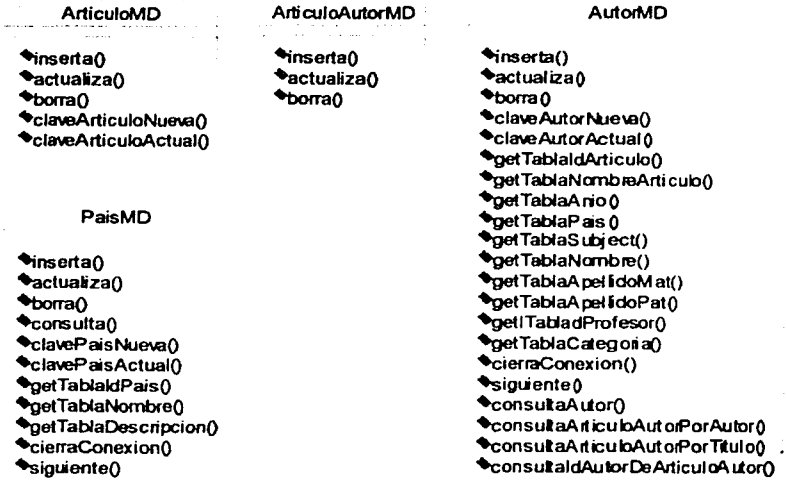
3.5.3.1 DIAGRAMA GENERAL DE CLASES



3.5.3.2 CLASES DEL DOMINIO DEL PROBLEMA

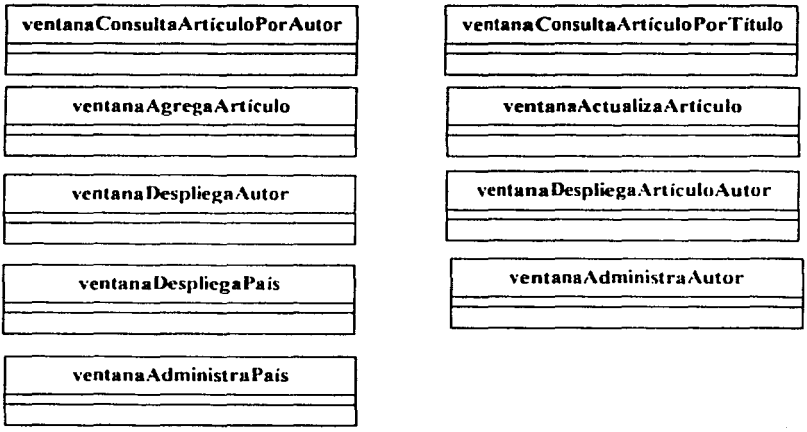


3.5.3.3 CLASES DE MANEJO DE DATOS



3.5.3.4. INTERFAZ DE USUARIO

Archivos Java Server Pages



3.6 Interacciones

Una *interacción* es un comportamiento que comprende un conjunto de mensajes intercambiables entre un conjunto de objetos dentro de un contexto para lograr un propósito.

UML (*Unified Modeling Lenguage*) proporciona una representación gráfica para los mensajes, como se muestra en la figura 3.6.1. Esta notación permite visualizar un mensaje de forma que permite destacar sus partes más importantes: nombre, parámetros (si tiene alguno) y secuencia. Gráficamente, un mensaje se representa como una linea dirigida y casi siempre incluye el nombre de su operación.

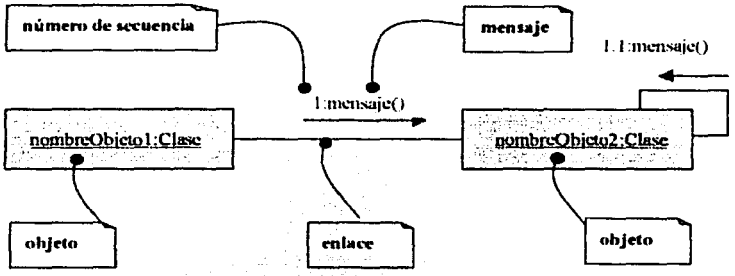


Figura 3.6.1. Mensajes, enlaces y secuenciamiento

Objetos

Un *objeto* es una entidad discreta, con límites bien definidos y con identidad, que encapsulan el estado y el comportamiento; se dice también de las instancias de una clase.

Enlaces

Un enlace es una conexión semántica entre objetos. Un enlace especifica un camino a lo largo del cual un objeto puede enviar un mensaje a otro objeto (o a si mismo). La mayoria de las veces es suficiente con especificar que existe ese camino.

Mensajes

Un mensaje es la especificación de una comunicación entre objetos transmite información, con la expectativa de que se desencadenará una actividad

Cuando se pasa un mensaje, la acción resultante es una instrucción ejecutable que constituye una abstracción de un procedimiento computacional. Una acción puede producir un cambio en el estado.

Secuenciación

Cuando un objeto envía un mensaje a otro (delegando alguna acción en el receptor) el objeto receptor puede, a su vez, enviar un mensaje a otro objeto, el cual puede enviar un mensaje a otro objeto diferente, etcétera. Este flujo de mensajes forma una secuencia.

3.6.1 Diagramas de Interacción

Los *diagramas de interacción* se utilizan para modelar los aspectos dinámicos de un sistema. Los diagramas de interacción pueden utilizarse para visualizar, especificar, construir y documentar la dinámica de una sociedad particular de objetos, o se pueden utilizar para modelar un flujo de control particular de un caso de uso.

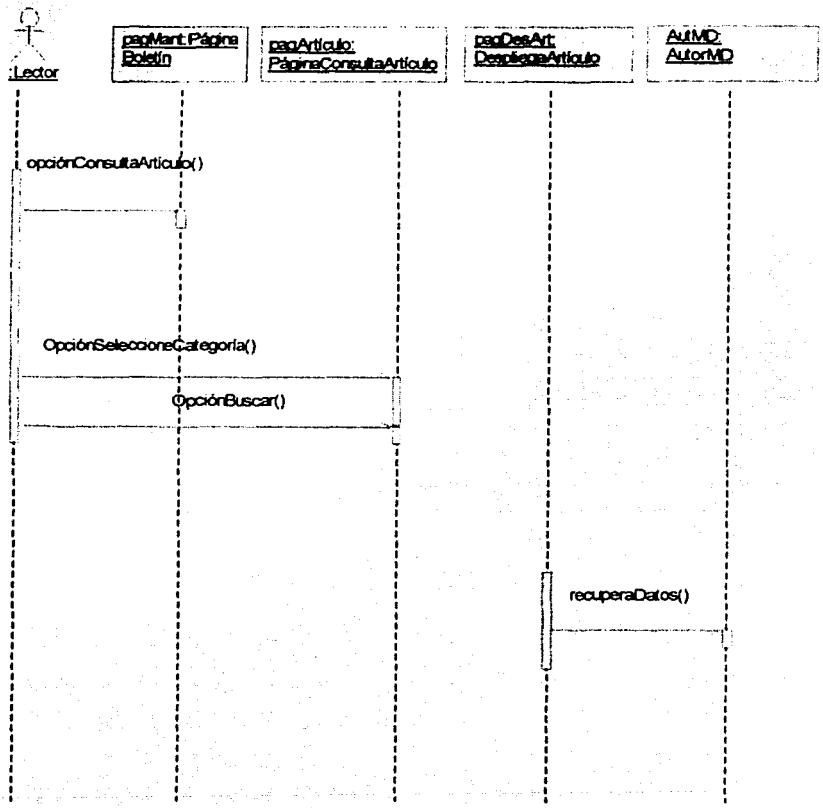
Los *diagramas de secuencia* y los *diagramas de colaboración* (ambos llamados diagramas de interacción) son dos de los cinco tipos de diagramas de UML que se utilizan para modelar los aspectos dinámicos de los sistemas. Un diagrama de interacción muestra una interacción que consiste en un conjunto de objetos y sus relaciones; incluyendo los mensajes que se pueden enviar entre ellos.

- Un *diagrama de secuencia* es un diagrama de interacción que destaca la ordenación temporal de los mensajes.
- Un *diagrama de colaboración* es un diagrama de interacción que destaca la organización estructural de los objetos que envían y reciben mensajes.

A continuación se muestran los *diagramas de secuencia* obtenidos en la fase de diseño del sistema para el manejo de las publicaciones de los artículos PEMEX. En estos diagramas de interacción se modelan los escenarios de los casos de uso obtenidos en la fase de planeación y elaboración del sistema.

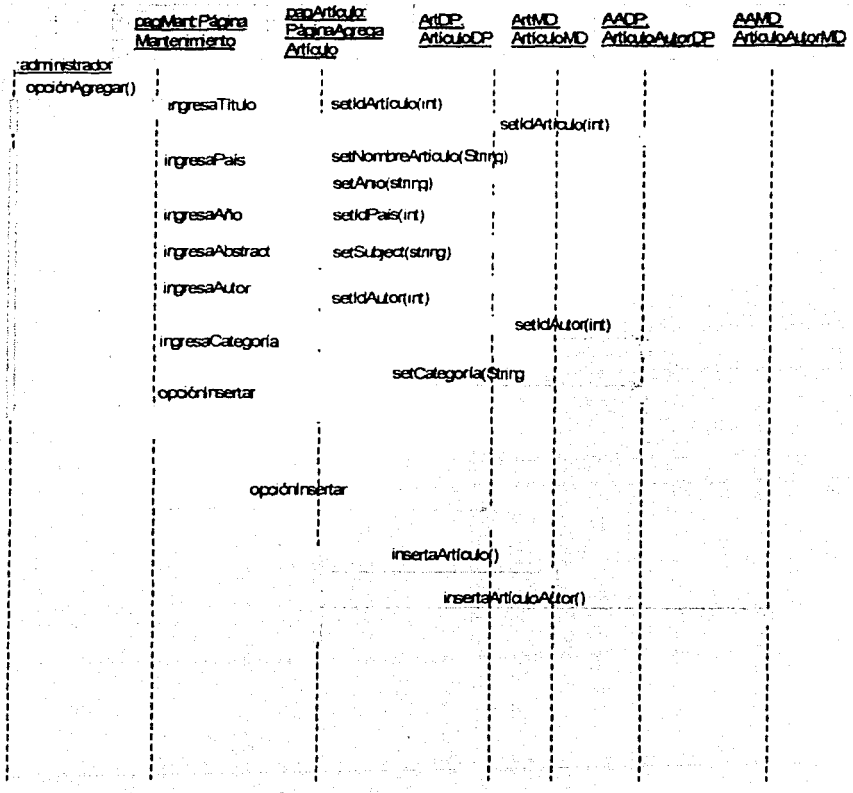
3.6.1.1. Caso de Uso. Consulta un Artículo
Referencia. R1.1, R1.2

Diagrama de secuencia:



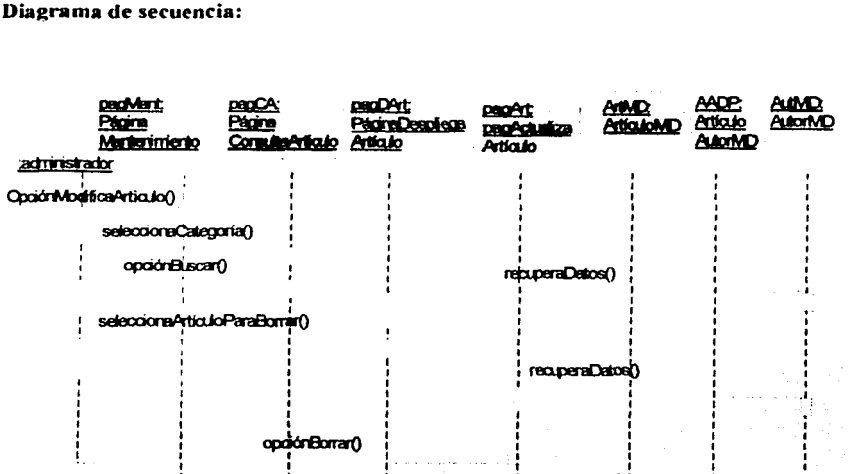
3.6.1.2 Caso de Uso. Agregar un Artículo Nuevo
Referencia. R2.1,R2.2

Diagrama de secuencia:



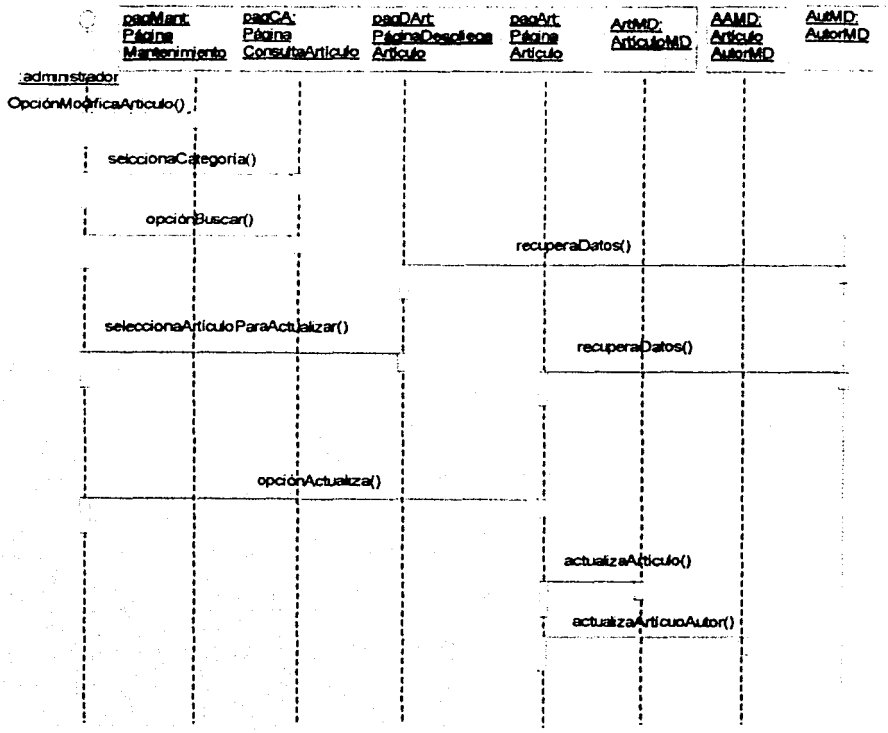
3.6.1.3 Caso de Uso. Eliminar un artículo del sistema
Referencia. R3.1, R3.2, R3.3

Diagrama de secuencia:



3.6.1.4 Caso de Uso. Actualiza un artículo del sistema
Referencia. R4.1, R4.2, R4.3

Diagrama de secuencia:



3.7 Bases de Datos Relacionales

La *base de datos* puede definirse como una colección de datos interrelacionados almacenados en conjunto sin redundancias perjudiciales o innecesarias; su finalidad es la de servir a una aplicación o más, de la mejor manera posible; los datos se almacenan de modo que resulten independientes de los programas que los usan; se emplean métodos bien determinados para incluir datos nuevos y para modificar o extraer los datos almacenados. Dícese que un sistema comprende una colección de base de datos cuando éstos son totalmente independientes desde el punto de vista estructural.

Objetivos de una organización de bases de datos

Un sistema de base de datos proporciona a la empresa un *control centralizado* de sus datos de operaciones, constituyendo uno de sus activos más valiosos. A continuación se consideran algunas ventajas de tener control centralizado de los datos.

1. Versatilidad para la representación de relaciones

Diferentes programadores requieren diferentes archivos lógicos. Estos archivos deben derivarse de la misma colección de datos. Existen diversas relaciones entre los rubros de datos almacenados. Algunas bases de datos comprenden un complejo entretejido de relaciones. El método de organización debe tener capacidad para representar estas relaciones y acomodar los posibles cambios en el futuro. El sistema de administración de datos debe tener a su vez capacidad para derivar, de datos y relaciones, los archivos lógicos que se presentan a un programador de aplicaciones y el almacenamiento físico de los datos.

2. Desempeño

Las bases de datos diseñadas para ser usadas por operadores de terminal deben asegurar un tiempo de respuesta adecuado para el diálogo entre el hombre y la terminal. Además, el sistema de bases de datos debe tener capacidad para manejar un adecuado caudal de transacciones. En los sistemas en que el volumen de tránsito es reducido, el caudal de transacciones (throughut) no tiene por que imponer muchas restricciones al diseño de la base de datos. En cambio, en las aerolíneas, el caudal de transacciones tiene una gran influencia sobre la organización del almacenamiento físico.

3. Costo mínimo

El costo por bit de almacenamiento está disminuyendo rápidamente gracias al progreso tecnológico, mientras que no ocurre lo mismo con el coste de la programación. Hay por lo tanto una necesidad creciente de mantener sencilla la programación de aplicación y las organizaciones lógicas de datos deben diseñarse con este objetivo.

4. Redundancia mínima

La redundancia es onerosa porque ocupa más espacio de almacén que el necesario y requiere múltiples operaciones de actualización. Debido a que diferentes copias de la misma información suelen hallarse en diferentes etapas de actualización, la redundancia da a menudo origen a respuestas incoherentes.

5. Capacidad de búsqueda

La capacidad para explorar una base de datos rápidamente y con diferentes criterios de búsqueda depende mucho de la organización física de los datos. Con algunas organizaciones los tiempos de búsqueda son excesivamente prolongados para poder considerar las respuestas como de tiempo real. Uno de los objetivos de la organización es, pues, el de lograr capacidad para la búsqueda rápida y flexible.

6. Integridad

Cuando una base de datos incluye información utilizada por muchos usuarios es importante que no puedan destruirse los datos almacenados ni las relaciones que existen entre los distintos ítems. Ocasionalmente se producirán fallos de hardware y diversos tipos de accidentes. El almacenamiento de los datos y los procedimientos de actualización e inserción deben asegurar que el sistema pueda recuperarse de estas contingencias sin daño para los datos. Toda instalación debe garantizar la integridad de la información que almacena.

Además de proteger los datos contra posibles problemas sistemáticos deben incluirse también procedimientos de revisión que aseguren que los valores de los datos se ajusten a ciertas reglas prescritas de antemano. Estas pruebas podrán hacerse verificando las relaciones que existen entre varios valores de datos.

7. Seguridad

La seguridad de los datos se refiere a la protección de éstos contra el acceso accidental o intencional por parte de personas no autorizadas y contra su indebida destrucción o alteración.

10. Afinación

La correcta afinación tiene dos requisitos. Primero, necesita la independencia física de los datos. Segundo, requiere medios para supervisar automáticamente el uso de la base de datos con el fin de que puedan hacerse los ajustes necesarios. En las futuras bases de datos se incorporarán posiblemente algunos medios para la afinación automática; por ejemplo, en lo que se refiere a la migración de datos. Se pretenda o no que la afinación sea automática, el sistema debe ser diseñado de modo que ella sea fácil.

11. Migración de datos

Algunos datos se usan con mucha frecuencia y otros solo raramente. Es deseable almacenar los datos de uso frecuente de manera que resulte fácil y rápido llegar a ellos. Los datos de uso ocasional se almacenarán, en cambio, de manera económica.

12. Simplicidad

Los medios que se utilizan para representar la vista general de los datos deben ser concebidos de manera simple y nitida. En muchos sistemas se utilizan punteros en la representación lógica con el fin de patentizar las relaciones que existen entre distintos items. Un problema que surge con el uso de punteros lógicos es que, a medida que se agregan más y más relaciones entre los items de datos, la colección de punteros resulta tan complicada que se hace difícil representar la vista lógica general de la base de datos con suficiente claridad. Aparecen punteros que indican punteros de punteros. En algunos casos, el abuso de los punteros múltiples en la representación de los datos para el usuario resulta engañoso. No se necesita, en realidad, tanta complejidad como la que se encuentra en ciertas estructuras lógicas de datos.

3.7.1 DBMS

Un programador aprende pronto que las estructuras de datos bien organizados simplifican la tarea de programación. De modo semejante, un diseñador de aplicaciones necesita principios de organización para manejar los grandes conjuntos de datos que deben ser coordinados en un grupo de programas. El diseñador de aplicaciones tiene que manejar numerosos problemas, incluyendo volúmenes de datos que se encuentran fuera de la capacidad de memoria, lenta operación de búsquedas no estructuradas, códigos para fines específicos personalizados para ciertos dispositivos de almacenamiento de información, asi como diversos mecanismos para representar relaciones entre elementos de datos. Estos problemas surgen por la complejidad de la computadora, no por lo intricado de la aplicación. Normalmente, la aplicación es suficiente desafío sin los problemas adicionales de soluciones computarizadas.

Un *sistema de administración de bases de datos* (DBMS) proporciona el método de organización necesario para el almacenamiento y recuperación flexibles de grandes cantidades de datos. El DBMS no elimina los problemas antes mencionados, sino que más bien los aísla en un paquete genérico que maneja estos problemas y presenta al diseñador una interfaz limpia. Con órdenes sencillas, la aplicación despacha objetos para que sean almacenados en al base de datos, dejando al DBMS que encuentre el correcto almacenamiento físico, que garantice el futuro acceso a través de varias rutas e integre los datos en relaciones apropiadas con otros elementos. Además, el DBMS maneja estas difíciles tareas de una manera genérica, de modo que el mismo DBMS puede servir en muchas aplicaciones.

3.7.2 Niveles de servicio de una base de datos

Hay tres niveles de servicios de base de datos.

- Nivel Físico.
- Nivel Conceptual.
- Nivel Externo.

1. Nivel físico

En el más bajo, más alejado de las entidades de aplicación, ciertos componentes físicos organizan y almacenan los datos en bruto (que no han sido procesados). Además del hardware, estos componentes incluyen estructuras de control que rastrean cuáles elementos de datos se encuentran en qué discos y en qué formato. Otras estructuras aparecen aquí también para acelerar la operación, como los búferes para retener datos que se emplean frecuentemente e informan para conducir búsquedas rápidas. La capa física por lo general tiene parámetros que pueden ser afinados para un funcionamiento óptimo bajo las pautas de acceso de una aplicación en particular

2. Nivel Conceptual

El asilamiento de estos detalles de almacenamiento en el nivel más bajo del DBMS proporcionan un cómodo separador para la siguiente capa más alta de abstracción, el nivel conceptual. Los objetos de aplicación existen a este nivel. Si el sistema operativo o el equipo básico cambia, las consecuencias se confinan a la interfaz entre la capa física y la capa conceptual inmediatamente encima de ella. En la capa central de la figura 3.7.2.1 se describe el entorno de aplicación completo en términos de las abstracciones soportadas por el DBMS, tales como tablas, objetos o reglas de inferencia. Aquí se encuentran las entidades de interés en las aplicaciones, junto con sus relaciones, restricciones y medidas de seguridad.

3. Nivel externo

En el nivel superior de la figura 3.7.3.2 representa vistas externas personalizadas que varían de acuerdo con la aplicación para diferentes usuarios.

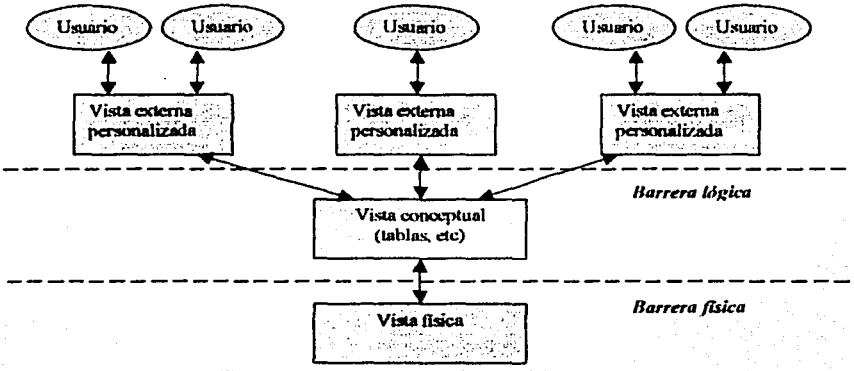


Figura 3.7.2.1. Independencia de datos física y lógica.

3.7.3 Modelo de datos Relacional

Cuando en el año 1970 el Dr. E.F. Codd propone un nuevo modelo de datos, los Sistemas gestores de bases de datos (SGBD) imperantes en el mercado, de tipo Codasyl y Jerárquico, no habían logrado superar el grave inconveniente que suponía la dependencia de las aplicaciones desarrolladas en ellos respecto a las estructuras de los datos.
A diferencia de estos modelos de datos basados en punteros físicos por los que tenía que navegar el programador a fin de recuperar y actualizar los datos, el Modelo Relacional (MR) se propone, como principal objetivo, aislar al usuario de las estructuras físicas de los datos, consiguiendo así la independencia de las aplicaciones respecto de los datos, finalidad perseguida desde los inicios de las bases de datos.

En el nuevo modelado, basado en la teoría matemática de las relaciones, los datos se estructuran lógicamente en forma de relaciones —tablas—. Esta formalización matemática convirtió rápidamente al modelo en una fuente fundamental de la investigación en bases de datos.
Los avances más importantes que el modelado de datos relacional incorpora respecto a los modelos de datos anteriores son:

- ✓ **Sencillez y uniformidad.** Los usuarios ven la base de datos relacional como una colección de tablas y al ser la tabla la estructura fundamental del modelo, éste goza de una gran uniformidad, lo que unido a unos lenguajes no navegacionales y muy orientado al usuario final da como resultado la sencillez de los sistemas relacionales.
- ✓ **Sólida fundamentación teórica.** Al estar el modelo definido con rigor matemático, el diseño y la evaluación del mismo pueden realizarse por métodos sistemáticos basados en abstracciones.

- ✓ **Independencia física/lógica.** Los lenguajes relacionales, al manipular conjuntos de registros, proporcionan una gran independencia respecto a la forma en la que los datos están almacenados.

La independencia física/lógica se refiere a tres aspectos:

1. **Independencia de la ordenación**; es decir, que el resultado obtenido en un acceso no dependa de cómo estén ordenados los datos físicamente.
2. **Independencia de la indexación**, separando los índices de los datos haciendo que la creación y mantenimiento sean manejados por el sistema.
3. **Independencia de los caminos de acceso**, haciendo que la navegación a través de los datos no tenga que estar previamente establecida consiguiendo así unas formas de acceso más flexibles.

3.7.3.1 Modelo Entidad/Relación

El *modelo entidad relación* describe con un alto nivel de abstracción la distribución de datos almacenados en un sistema. Existen dos elementos principales: las entidades y las relaciones.

Entidad

Es aquel objeto, real o abstracto, acerca del cual se desea almacenar información en la base de datos. La estructura genérica de un conjunto de entidades con las mismas características se denomina tipo de entidad.

Relación

Es una asociación o correspondencia existente entre una o varias entidades. Conceptualmente se pueden identificar tres clases de relaciones:

1. **Relación 1:1:** cada ocurrencia de una entidad se relaciona con una y sólo una ocurrencia de la otra entidad.
2. **Relación 1:N:** cada ocurrencia de una entidad puede estar relacionada con cero, una o varias ocurrencias de la otra entidad.
3. **Relación M:N:** cada ocurrencia de una entidad puede estar relacionada con cero, una o varias ocurrencias de la otra entidad y cada ocurrencia de la otra entidad puede corresponder a cero, una o varias ocurrencias de la primera.

Atributo

Es una propiedad o característica de un tipo de entidad. Se trata de la unidad básica de información que sirve para identificar o describir la entidad. Un atributo se define sobre un dominio. Cada tipo de entidad ha de tener un conjunto mínimo de atributos que identifiquen unívocamente cada ocurrencia del tipo de entidad. Este atributo se denomina identificador principal.

Llave Primaria

Es un identificador único para cada registro de la tabla.

Llave Candidata

Con frecuencia, en una entidad, es posible que más de un atributo pueda servir como clave primaria.

Llave Externa

Es una columna o grupo de columnas que se encuentra como llave primaria en alguna tabla y se encuentra en una o más tablas para relacionarse.

3.7.3.2 Normalización

La teoría de la *normalización* tiene por objetivo la eliminación de dependencias entre atributos que originen anomalías en la actualización de los datos y se proporcione una estructura más regular para la representación de las tablas, constituyendo el soporte para el diseño de bases de datos relacionales. Como resultado de la aplicación de esta técnica se obtiene un modelo lógico de datos normalizado.

La teoría de la normalización, como técnica formal para organizar los datos, ayuda a encontrar fallos y a corregirlos, evitando así introducir anomalías en las operaciones de manipulación de datos.

Se dice que una relación está en una determinada forma normal si satisface un cierto conjunto de restricciones sobre los atributos. Cuantos más restricciones existan, menor será el número de relaciones que las satisfagan, así, por ejemplo, una relación en tercera forma normal estará también en segunda y en primera forma normal.

Antes de definir las distintas formas normales se explican, muy brevemente, los siguientes conceptos necesarios para su comprensión.

a) Dependencia funcional.
b) Dependencia funcional completa.
c) Dependencia transitiva.

Dependencia funcional

Un atributo Y se dice depende funcionalmente de otro X si, y sólo si, a cada valor de X le corresponde un único valor de Y, lo que se expresa de la siguiente forma: X→Y (también se dice que X determina o implica Y).

X se denomina implicante o determinante e Y es el implicado.

Dependencia funcional completa

Un atributo Y tiene dependencia funcional completa respecto de otro X, si depende funcionalmente de él en su totalidad; es decir, no depende de ninguno de los posibles atributos que formen parte de X.

Dependencia transitiva

Un atributo depende transitivamente de otro si, y sólo si, depende de él a través de otro atributo. Así, Z depende transitivamente de X, si:

- $X \rightarrow Y$
- $Y \not\rightarrow X$
- $Y \rightarrow Z$

Se dice que X implica a través de Y.

Formas Normales

Una vez definidas las anteriores dependencias se pueden enunciar las siguientes formas normales:

1. Primera forma normal (1FN).
2. Segunda forma normal (2FN).
3. Tercera forma normal (3FN).
4. Cuarta forma normal.
5. Quinta forma normal.

1. Primera forma normal (1FN)

Una entidad está en 1FN si no tiene grupos repetitivos; es decir, un atributo sólo puede tomar un valor único de un dominio simple.

Una vez identificados los atributos que no dependen funcionalmente de la clave principal, se formará con ellos una nueva entidad y se eliminarán de la antigua. La clave principal de la nueva entidad estará formada por la concatenación de uno o varios de sus atributos más la clave principal de la antigua entidad.

2. Segunda forma (2FN)

Una entidad está en 2FN si está en 1FN y todos los atributos que no forman parte de las claves candidatas (atributos no principales) tienen dependencia funcional completa respecto de éstas es decir, no hay dependencias funcionales de atributos no principales respecto de una parte de las claves. Cada uno de los atributos de una entidad depende de toda la clave.

Una vez identificados los atributos que no dependen funcionalmente de toda la clave, sino sólo de parte de la misma, se formará con ellos una nueva entidad y se eliminarán de la

antigua. La clave principal de la nueva entidad estará formada por la parte de la antigua de la que depende funcionalmente.

3. Tercera forma normal (3FN)

Una entidad está en 3FN si está en 2FN y todos sus atributos no principales dependen directamente de la clave primaria es decir, no hay dependencias funcionales transitivas de atributos no principales respecto de las claves.
Una vez identificados los atributos que dependen de otro atributo distinto de la clave, se formará con ellos una nueva entidad y se eliminarán de la antigua. La clave principal de la nueva entidad será el atributo del cual depende. Este atributo en la entidad antigua, pasará a ser una clave externa.

4. Cuarta forma normal

En las relaciones varios-con-varios, entidades independientes no pueden ser almacenadas en la misma tabla.

5. Quinta forma normal

Existe otro nivel de normalización que se aplica a veces, pero es de hecho algo esotérico y en la mayoria de los casos no es necesario para obtener la mejor funcionalidad de nuestra estructura de datos o aplicación. Su principio sugiere:

La tabla original debe ser reconstruida desde las tablas resultantes en las cuales ha sido troceada.

Los beneficios de aplicar esta regla aseguran que no se haya creado ninguna columna extraña en las tablas y que la estructura de las tablas que se ha creado sea del tamaño justo que tiene que ser. Es buena práctica aplicar esta regla, pero a no ser que se esté tratando con una extensa estructura de datos probablemente no se necesite.

Objetivos de la Normalización

La normalización del modelo de datos deberá cumplir los siguientes objetivos

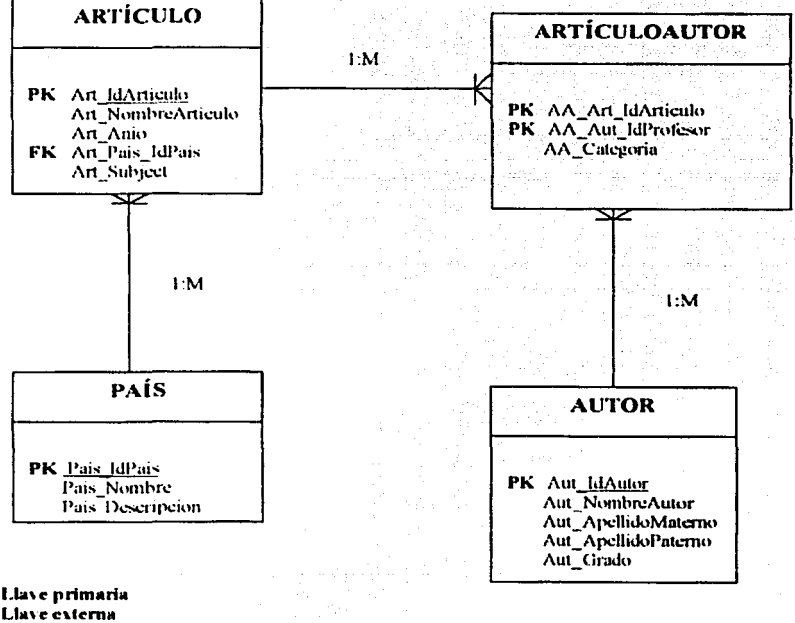
- ✓ Eliminar la redundancia y, por tanto, posibilidad de inconsistencias.
- ✓ Eliminar las ambigüedades.
- ✓ Evitar la pérdida de información.
- ✓ Evitar la pérdida de dependencias funcionales es decir, de ciertas restricciones de integridad que dan lugar a interdependencias entre los datos.
- ✓ Evitar la existencia de valores nulos (inaplicables).

Evitar la aparición, en la base de datos, de estados que no son válidos en el mundo real;

- **Anomalías de modificación**; ya que se podria, por ejemplo, cambiar el valor de un atributo en un renglón y por error no modificarlo en el resto de los renglones que le corresponden al mismo atributo, por lo que da lugar a inconsistencias.
- **Anomalías de inserción**; no cabe almacenar nueva información sobre una entidad en particular hasta que se establece su relación con otra entidad.
- **Anomalías de borrado**; ya que, al eliminar un renglón, se elimina toda la ocurrencia de la entidad.

A continuación se muestra el modelo entidad relación del diseño del sistema para el manejo de las publicaciones de los articulos PEMEX.

3.7.3.2.1. DIAGRAMA ENTIDAD/RELACION



PK: Llave primaria
FK: Llave externa

3.8 PLAN DE PRUEBAS DE INTEGRACIÓN

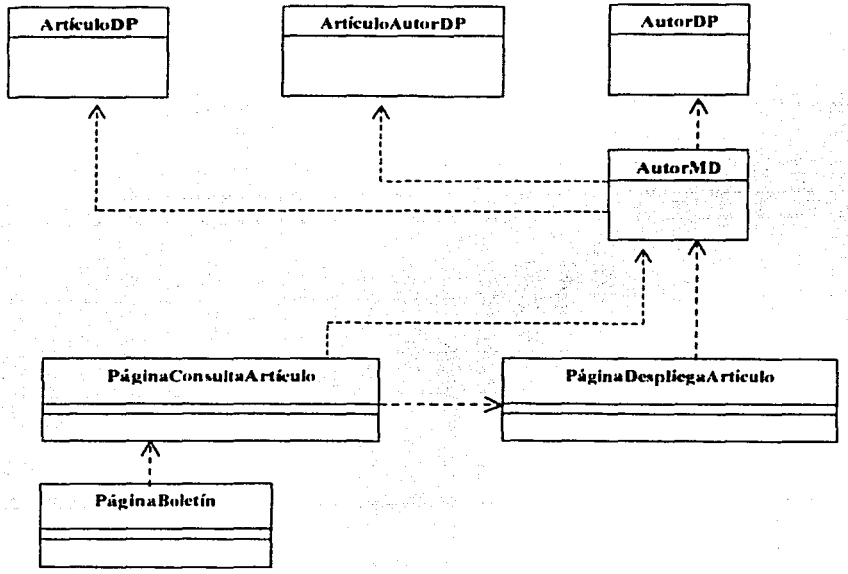
El objetivo principal de las pruebas de integración es detectar las fallas de interacción entre las distintas clases que componen el sistema.

Debido a que cada clase probada unitariamente se inserta de manera progresiva dentro de la estructura, las pruebas de integración son el mecanismo para comprobar el correcto ensamblaje del sistema completo. Al efectuar la integración de los módulos, se debe concentrar en la búsqueda de defectos tales como aquellos que puedan provocar excepciones arrojadas por los métodos.

A continuación se muestran los diagramas de clases dependientes, del sistema para el manejo de las publicaciones de los artículos PEMEX y cada uno de los métodos a probar.

3.8.1 CONSULTA DE DATOS

CLASES DEPENDIENTES



ArticuloDP

Depende de	Métodos a Probar
	Ninguno

Clase ArticuloAutorDP

Depende de	Métodos a Probar
	Ninguno

Clase AutorDP

Depende de	Métodos a Probar
	Ninguno

Clase AutorMD

Depende de	Métodos a Probar
AutorDP	AutorDP ▪ Métodos set[atributo]() ▪ Métodos get[atributo]()
ArticuloDP	ArticuloDP ▪ Métodos set[atributo]() ▪ Métodos get[atributo]()
ArticuloAutorDP	ArticuloAutorDP ▪ Métodos set[atributo]() ▪ Métodos get[atributo]()

PáginaConsultaArticulo

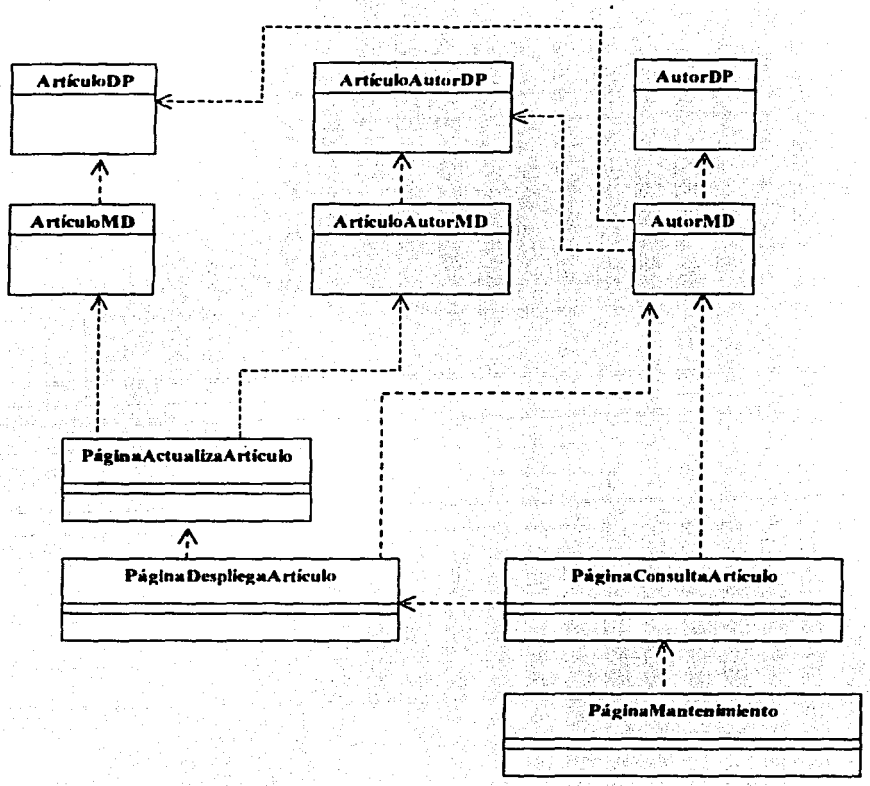
Depende de	Métodos a Probar
AutorMD	ArticuloMD ▪ ConsultaArticuloAutorPorAutor() ▪ ConsultaArticuloAutorPorTitulo()

PáginaDespliegaArtículo

Depende de	Métodos a Probar
PáginaConsultaArticulo	PáginaConsultaArticulo ▪ ObtieneTitulo() ▪ ObtieneAutor()
AutorMD	ArticuloMD ▪ ConsultaArticuloAutorPorAutor() ▪ ConsultaArticuloAutorPorTitulo() ▪ getTablaNombreArticulo() ▪ getTablaAnio() ▪ getTablaPais() ▪ getTablaSubject() ▪ getTablaNombre() ▪ getTablaApellidoMat() ▪ getTablaApellidoPat() ▪ getTablaCategoria()

3.8.2 MODIFICACIÓN DE DATOS DEL ARTÍCULO

CLASES DEPENDIENTES



Clase ArticuloDP

Depende de	Métodos a Probar
	Ninguno

Clase ArticuloAutorDP

Depende de	Métodos a Probar
	Ninguno

Clase AutorDP

Depende de	Métodos a Probar
	Ninguno

Clase ArticuloMD

Depende de	Métodos a Probar
ArticuloDP	ArticuloDP ▪ Métodos set[atributo]() ▪ Métodos get[atributo]()

Clase ArticuloAutorMD

Depende de	Métodos a Probar
ArticuloAutorDP	ArticuloAutorDP ▪ Métodos set[atributo]() ▪ Métodos get[atributo]()

Clase AutorMD

Depende de	Métodos a Probar
AutorDP	AutorDP ▪ Métodos set[atributo]() ▪ Métodos get[atributo]()
ArticuloDP	ArticuloDP ▪ Métodos set[atributo]() ▪ Métodos get[atributo]()
ArticuloAutorDP	ArticuloAutorDP ▪ Métodos set[atributo]() ▪ Métodos get[atributo]()

PáginaConsultaArticulo

Depende de	Métodos a Probar
AutorMD	ArticuloMD ▪ ConsultaArticuloAutorPorAutor() ▪ ConsultaArticuloAutorPorTitulo()

PáginaDespliegaArtículo

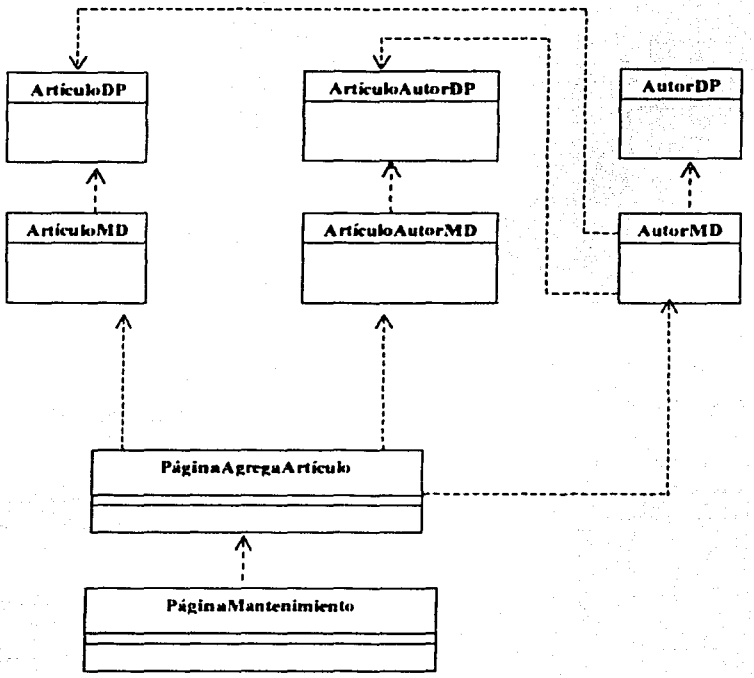
Depende de	Métodos a Probar
PáginaConsultaArticulo	PáginaConsultaArticulo ▪ ObtieneTitulo() ▪ ObtieneAutor()
AutorMD	ArticuloMD ▪ ConsultaArticuloAutorPorAutor() ▪ ConsultaArticuloAutorPorTitulo() ▪ getTablaNombreArticulo() ▪ getTablaAnio() ▪ getTablaPais() ▪ getTablaSubject() ▪ getTablaNombre() ▪ getTablaApellidoMat() ▪ getTablaApellidoPat() ▪ getTablaCategoria()

PáginaActualizaArtículo

Depende de	Métodos a Probar
PáginaDespliegaArticulo	PáginaDespliegaArticulo ▪ ObtieneIdArticulo()
ArticuloMD	ArticuloMD ▪ actualiza() ▪ borra()
ArticuloAutorMD	ArticuloAutor ▪ actualiza() ▪ borra()

3.8.3 INSERCIÓN DE DATOS DEL ARTÍCULO

CLASES DEPENDIENTES



Clase ArticuloDP

Depende de	Métodos a Probar
	Ninguno

Clase ArticuloAutorDP

Depende de	Métodos a Probar
	Ninguno

Clase AutorDP

Depende de	Métodos a Probar
	Ninguno

Clase ArticuloMD

Depende de	Métodos a Probar
ArticuloDP	ArticuloDP ▪ Métodos set[atributo]() ▪ Métodos get[atributo]()

Clase ArticuloAutorMD

Depende de	Métodos a Probar
ArticuloAutorDP	ArticuloAutorDP ▪ Métodos set[atributo]() ▪ Métodos get[atributo]()

Clase AutorMD

Depende de	Métodos a Probar
AutorDP	AutorDP ▪ Métodos set[atributo]() ▪ Métodos get[atributo]()
ArticuloDP	ArticuloDP ▪ Métodos set[atributo]() ▪ Métodos get[atributo]()
ArticuloAutorDP	ArticuloAutorDP ▪ Métodos set[atributo]() ▪ Métodos get[atributo]()

PáginaActualizaArtículo

Depende de	Métodos a Probar
AutorMD	AutorMD ▪ ConsultaAutor()
ArticuloMD	ArticuloMD ▪ inserta()
ArticuloAutorMD	ArticuloAutorMD ▪ claveArticuloNueva() ▪ claveArticuloActual() ▪ inserta()

CAPÍTULO 4

IMPLANTACIÓN Y PRUEBAS DEL SISTEMA

4.1 Introducción

En la *implantación* se empieza con el resultado del diseño para implantar el sistema en términos de componentes; es decir, ficheros de código fuente, scripts, ficheros de código binario, ejecutables y similares.

Afortunadamente, la mayor parte de la arquitectura del sistema es capturada durante el diseño, siendo el propósito principal de la implantación el desarrollar la arquitectura y el sistema como un todo. De forma más específica, los propósitos de la implantación son:

- ✓ Distribuir el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue.
- ✓ Implementar el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue
- ✓ Probar los componentes individualmente y a continuación integrarlos, compilándolos y enlazándolos en uno o más ejecutables, antes de ser enviados para ser integrados y llevar a cabo las comprobaciones del sistema.

A continuación en la figura 4.1.1 se muestra la gráfica de los pasos del proceso unificado de desarrollo de un sistema, donde se puede observar la *fase de implantación y pruebas*, dentro del proceso de desarrollo.

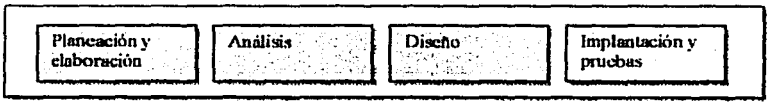


Figura 4.1.1. Pasos en el proceso unificado del desarrollo de un sistema.

4.2 Artefactos de la fase de Implantación y Pruebas del Sistema

La siguiente figura 4.2.1. muestra las dependencias de los artefactos durante la fase de implantación y pruebas. Los que están señalados con una flecha, corresponden a los artefactos que se deberán construir en la fase de implementación y pruebas.

A continuación se da una descripción breve de los artefactos.

- **Definición de clases y de interfaz.** Implantar las definiciones de clase y de interfaz.
- **Métodos.** Implantar los métodos.
- **Casos de Prueba.** Implantación del plan de pruebas de diseño (casos de pruebas). Descripción de los procedimientos y las métricas para evaluación de defecto.

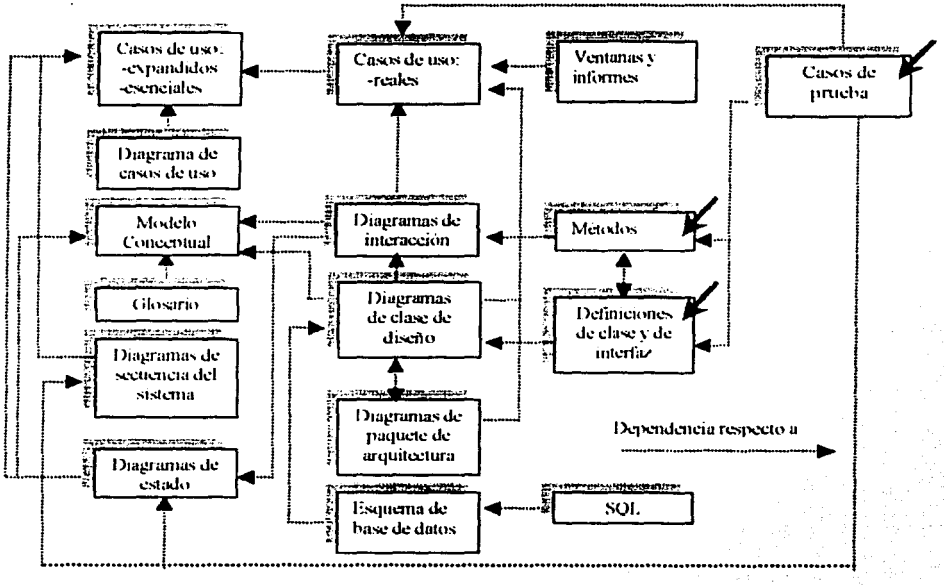


Figura 4.2.1. Dependencias de los artefactos obtenidos durante la fase de implantación y pruebas.

En la tabla 4.2.1 se muestran los documentos utilizados en la fase de implantación y pruebas.

Tabla 4.2.1 Documentos utilizados en la etapa de implantación y pruebas.

Implantación y pruebas	Documentos relacionados (documentos entregables)
Componentes.	Diagrama de Componentes
Configuración del sistema entregable.	Diagrama de Despliegue
Definición de estándares de programación.	Código fuente
Codificación y pruebas unitarias.	Soporte de pruebas unitarias
Pruebas de módulos y de sistemas.	Documentación del código

4.3 La programación y el proceso de desarrollo

Los artefactos del UML (*Unified Modeling Language*) creados en la fase de diseño —los diagramas de secuencia y los de la clase de diseño— servirán de entrada en el proceso de generación de código.

Si se quiere reducir el riesgo y aumentar la probabilidad de conseguir una aplicación adecuada, el desarrollo debería basarse en un suficiente modelado del análisis y diseño antes de iniciar la codificación. Ello no significa que durante la programación no tengan cabida los prototipos ni el diseño: las herramientas modernas del desarrollo ofrecen un excelente ambiente para examinar rápidamente métodos alternos y normalmente vale la pena dedicar poco o mucho tiempo al diseño por la codificación.

Pero la parte esencial de la aplicación, por ejemplo el modelo conceptual básico, las capas arquitectónicas, las principales asignaciones de responsabilidades y las interacciones más importantes de los objetos se determinan más satisfactoriamente en una investigación formal y en el proceso de diseño que "apresurándose a codificar". Esto último origina sistemas que son más difíciles de entender, ampliar y de darles mantenimiento, sin que brinden soporte a un proceso susceptible de repetirse eficazmente.

Dicho lo anterior, a menudo lo más conveniente es omitir la fase de diseño para realizar un poco de programación exploratoria con el fin de descubrir un diseño funcional y luego regresar a la fase formal de diseño.

La creación de código en un lenguaje orientado a objetos —Java y Smaltalk, entre otros— no forma parte del análisis ni del diseño orientado a objetos; es meta final en sí misma. Los artefactos producidos en la fase de diseño suministran suficiente información necesaria para generar el código y, como se verá más adelante, se trata de un proceso de traducción relativamente simple.

Una ventaja del análisis, del diseño y de la programación orientados a objetos —cuando se emplean junto con un proceso de desarrollo como el que ya se mencionó— es que ofrecen una guía completa de principio a fin para realizar la codificación a partir de los requerimientos. Los artefactos que se introducen en otros posteriores en una forma rastreable y útil culminarán finalmente en una aplicación funcional. Con ello no se quiere

decir que el camino sea fácil ni que se pueda conseguir mecánicamente, pues existen demasiadas variables. Pero la guía constituye un punto de partida para experimentar e intercambiar opiniones.

La toma de decisiones y el trabajo creativo se realizan en gran medida durante las fases de análisis y de diseño.

No obstante, en general la fase de programación no es un paso fácil de la generación del código, todo lo contrario. En realidad los resultados obtenidos durante el diseño son un primer paso incompleto; en la programación y en las pruebas se efectuará multitud de cambios y se descubrirán y resolverán problemas detallados.

Los artefactos del diseño, cuando están bien hechos, producen un núcleo flexible que crece con elegancia y fuerza para entender los nuevos problemas que surjan en la programación.

Mapeo de diseño para la codificación

Para implantar un lenguaje de programación orientado a objetos se requiere escribir código fuente para:

1. Definición de clase.
2. Definición de métodos.

1.Creación de las definiciones de clases a partir de los diagramas de clases del diseño.

Los diagramas de clases del diseño describen por lo menos el nombre de las clases, las superclases, las etiquetas de los métodos y los atributos simples de una clase. Esa información es suficiente para formular una definición básica de clase en un lenguaje orientado a objetos.

2. Creación de métodos a partir de los diagramas de secuencias.

Un diagrama de secuencia muestra los mensajes que se envían en respuesta a una llamada al método. La secuencia de mensajes se traduce a una serie de enunciados en la definición del método.

4.4 Componentes

Un *componente* es una parte física y reemplazable de un sistema que conforma con un conjunto de interfaces y proporciona la realización de esas interfaces. Gráficamente, un componente se representa como un rectángulo con pestañas.

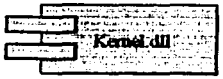


Figura 4.4.1. Componentes.

Los componentes se utilizan para modelar los elementos físicos que pueden hallarse en un nodo, tales como ejecutables, bibliotecas, tablas, archivos y documentos. Normalmente, un componente representa el empaquetamiento físico de elementos que por el contrario son lógicos, tales como clases, interfaces y colaboraciones. Los buenos componentes definen abstracciones precisas con interfaces bien definidas, permitiendo reemplazar fácilmente los componentes más viejos con otros más nuevos y compatibles.

Nombre

Cada componente debe tener un nombre que lo distinga del resto de componentes. Un nombre es una cadena de texto. Este nombre solo se denomina nombre simple; un nombre de camino consta del nombre del componente precedido del nombre del paquete en el que se encuentra. Normalmente, un componente se dibuja mostrando sólo su nombre.

Componentes y clases

En muchos sentidos, los componentes son como las clases: ambos tienen nombres; ambos pueden realizar un conjunto de interfaces; ambos pueden participar en relaciones de dependencia, generalización y asociación; ambos pueden anidarse; ambos pueden tener instancias; ambos pueden participar en interacciones; sin embargo, hay algunas diferencias significativas entre los componentes y las clases.

1. Las clases representan abstracciones lógicas; los componentes representan elementos físicos del mundo de los bits. Para decirlo brevemente, los componentes pueden estar en nodos, las clases no.
2. Los componentes representan el empaquetamiento físico de componentes que, por el contrario, son lógicos y se encuentran a distinto nivel de abstracción.
3. Las clases pueden tener atributos y operaciones directamente accesibles. En general, los componentes sólo tienen operaciones que sólo son alcanzables a través de sus interfaces.

La *primera diferencia* es la más importante. Durante el modelado de un sistema, la decisión de si se debería utilizar una clase o un componente es sencilla. Si el elemento que se va a modelar reside directamente en un nodo, debe usarse un componente; en otro caso, debe usarse una clase.

La *segunda diferencia* sugiere una relación entre las clases y los componentes: En particular, un componente es la implantación física de un conjunto de otros elementos lógicos, tales como clases. Como se muestra en la figura 4.4.1.2, la relación entre un componente y las clases que implantan puede representarse explícitamente mediante una relación de dependencia. La mayoría de las veces no será necesario representar estas relaciones gráficamente. Más bien, se mantendrá como una parte de la especificación del componente.

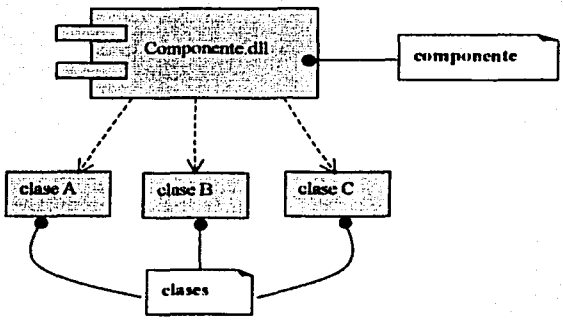


Figura 4.4.2. Componentes y clases

La tercera *diferencia* indica cómo las interfaces hacen de puente entre los componentes y las clases.

Componentes e interfaces

Una interfaz es una colección de operaciones que se utiliza para especificar un servicio de una clase o un componente. La relación entre componente e interfaz es importante. Todas las facilidades más conocidas de los sistemas operativos que están basadas en componentes (tales como COM+, CORBA y los Enterprise Java Beans) utilizan las interfaces como el pegamento que enlaza a los componentes entre si.

Como se indica en la figura 4.4.1.3, es posible mostrar la relación entre un componente y sus interfaces de dos formas. El primer estilo (y más frecuente) representa a la interfaz mediante un icono, contraída. El componente que la realiza se conecta a la interfaz con una relación de realización simplificada. La segunda forma representa a la interfaz expandida, quizá revelando sus operaciones. El componente que realiza la interfaz se conecta a ella con una relación de realización completa. En ambos casos, el componente que entra a los servicios del otro componente a través de la interfaz se conecta a ella con una relación de dependencia

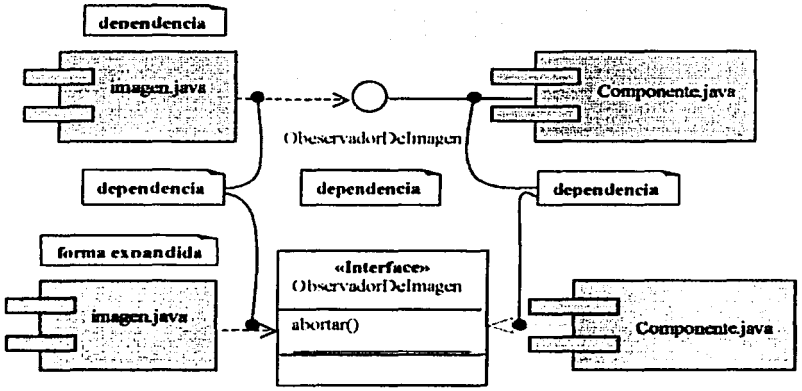


Figura 4.4.3. Componentes e interfaces

Organización de componentes

Los componentes se pueden organizar agrupándolos en paquetes de la misma forma en que se organizan las clases.

Los componentes también se pueden organizar especificando entre ellos relaciones de dependencia, generalización, asociación (incluyendo agregación) y realización.

4.4.1 Diagramas de Componentes

Los diagramas de componentes son uno de los dos tipos de diagramas que aparecen cuando se modelan los aspectos físicos de los sistemas orientados a objetos. Un diagrama de componentes muestra la organización y las dependencias entre un conjunto de componentes. Los diagramas de componentes se utilizan para modelar la vista de implementación estática de un sistema. Esto implica modelar las cosas físicas que residen en un nodo, tales como ejecutables, bibliotecas, tablas, archivos y documentos. Los diagramas de componentes son fundamentalmente diagramas de clases que se centran en los componentes de un sistema.

Los diagramas de componentes no sólo son importantes para visualizar, especificar y documentar sistemas basados en componentes, sino también para construir sistemas ejecutables mediante ingeniería directa e inversa.

Con UML (*Unified Modeling Language*) los diagramas de componentes se utilizan para visualizar los aspectos estáticos de estos componentes físicos y sus relaciones y para especificar sus detalles para la construcción, como se muestra en la figura 4.4.3.1.

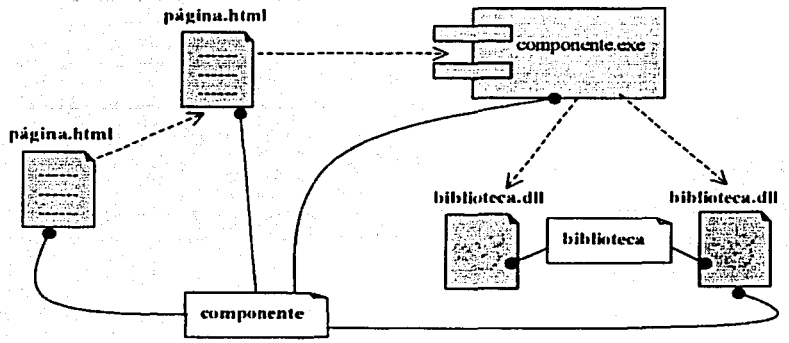


Figura 4.4.1.1. Un diagrama de componentes

Cuando se modela la vista de implantación estática de un sistema, normalmente se utilizarán los diagramas de componentes de una de las cuatro maneras siguientes:

1. ***Para modelar código fuente.***

Con la mayoría de los lenguajes de programación orientado a objetos actuales, el código se produce utilizando entornos integrados de desarrollo que almacenan el código fuente en archivos. Los diagramas de componentes se pueden utilizar para modelar la gestión de configuraciones de estos archivos, los cuales representan los componentes obtenidos como producto del trabajo.

2. ***Para modelar versiones ejecutables***

Una versión es un conjunto de artefactos relativamente consistente y completo que se entrega a un usuario interno o externo. En el contexto de los componentes, una versión se centra en las partes necesarias para entregar un sistema en ejecución. Cuando se modela una versión mediante diagramas de componentes, se están visualizando, especificando y documentando las decisiones acerca de las partes físicas que constituyen el software (es decir, sus componentes de despliegue).

3. ***Para modelar bases de datos físicas***

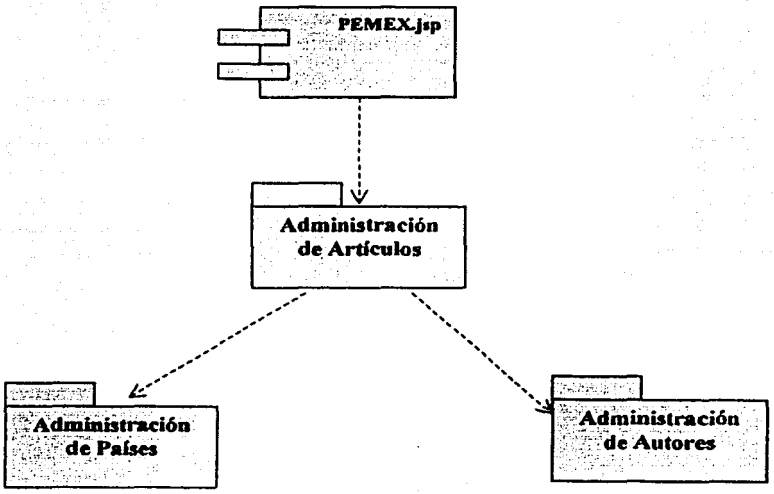
Una base de datos física puede ser vista como la realización concreta de un esquema y que pertenece al mundo de los bits. En efecto, los esquemas ofrecen una API para la información persistente; el modelo de una base de datos física representa el almacenamiento de esta información en las tablas de una base de datos relacional o las páginas de una base de datos orientada a objetos. Los diagramas de componentes se utilizan para representar éstos y otros tipos de bases de datos físicas.

4. Para modelar sistemas adaptables

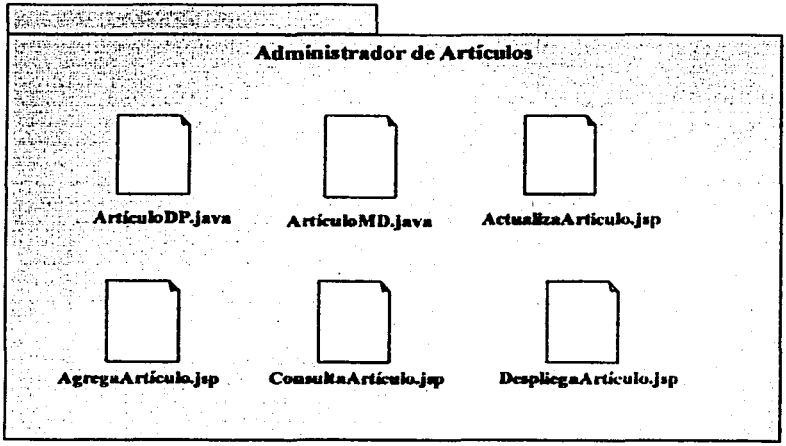
Algunos sistemas son bastante estáticos; sus componentes entran en escena, participan en la ejecución y desaparecen. Otros sistemas son más dinámicos e implican agentes móviles o componentes que migran con el propósito de equilibrar la carga o la recuperación de fallos. Los diagramas de componentes se utilizan, junto a algunos de los diagramas de UML, para modelar el comportamiento, con el fin de representar a estos tipos de sistemas.

A continuación se muestra el diagrama de componentes del sistema para el manejo de las publicaciones de los artículos PEMEX, siguiendo el modelo de archivos y documentos.

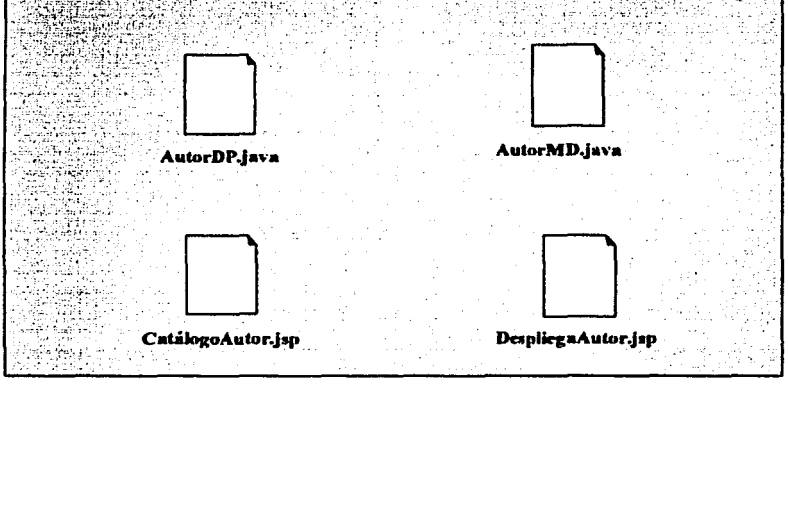
4.4.1.1 DIAGRAMA DE COMPONENTES DEL SISTEMA



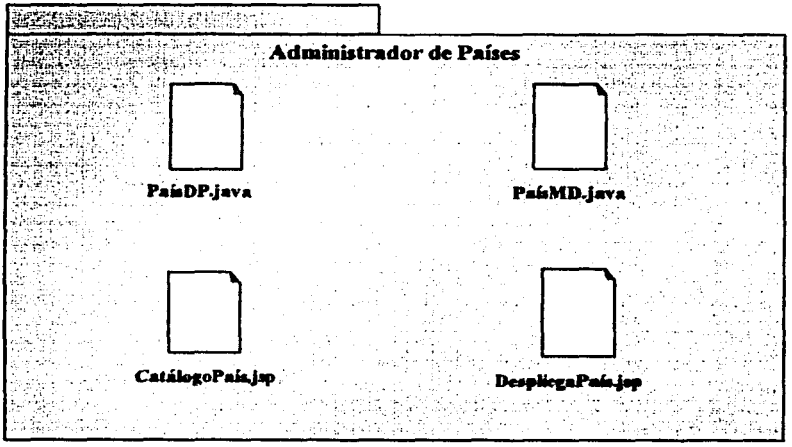
4.4.1.2 ADMINISTRACIÓN DE ARTÍCULOS



4.4.1.3 ADMINISTRACIÓN DE AUTORES



4.4.1.4 ADMINISTRACIÓN DE PAÍSES



4.4.2 Ingeniería directa e inversa

Hacer ingeniería directa e inversa con componentes es algo bastante directo, ya que los componentes son en sí mismos cosas físicas (ejecutables, bibliotecas, tablas, archivos y documentos) y, por lo tanto, son cosas cercanas al sistema en ejecución. Al hacer ingeniería directa con una clase o una colaboración, realmente se hace ingeniería directa hacia un componente que representa el código fuente, una biblioteca binaria o un ejecutable para esa clase. Así mismo, cuando se hace ingeniería inversa a partir de código fuente, bibliotecas binarias o ejecutables, realmente se hace ingeniería inversa hacia un componente o un conjunto de componentes que, a su vez, se corresponden con clases.

Una decisión que debe tomarse es la elección acerca de si al aplicar ingeniería directa (creación de código a partir de un modelo) a una clase, se obtendrá código fuente, una biblioteca en formato binario o un ejecutable. Los modelos lógicos se convertirán en código si se tiene interés en controlar la gestión de configuraciones de los archivos que son, posteriormente, manipulados por un entorno de desarrollo. Los modelos lógicos se convertirán directamente en bibliotecas binarias o ejecutables si se tiene interés por gestionar los componentes que se desplegarán en un sistema ejecutable. En algunos casos, se decidirá hacer ambas cosas.

Para hacer ingeniería directa con un diagrama de componentes:

- Hay que identificar las clases o colaboraciones que implanta cada componente.
- Hay que elegir el destino para cada componente. La elección se hará básicamente entre código fuente (una forma manipulable por herramientas de desarrollo) o una biblioteca binaria o un ejecutable (una forma que puede introducirse en un sistema ejecutable).
- Hay que utilizar herramientas para aplicar ingeniería directa a los modelos.

Construir un diagrama de componentes mediante ingeniería inversa (creación de un modelo a partir de código) no es un proceso perfecto ya que siempre hay una pérdida de información. A partir del código fuente, se pueden obtener las clases; esto es lo que se hace con más frecuencia. Cuando se haga ingeniería inversa desde el código hacia los componentes se revelarán dependencias de compilación entre esos archivos. Para las bibliotecas ejecutables, lo más que se pueda esperar es representar la biblioteca como un componente y descubrir sus interfaces a través de ingeniería inversa. Éste es el segundo uso más frecuente que se hace de los diagramas de componentes. De hecho, ésta es una forma útil de abordar un conjunto de nuevas bibliotecas que de otra forma estarían pobremente documentadas. Para los ejecutables, lo mejor que se puede esperar obtener es la representación del ejecutable como un componente y luego desensamblar su código (algo que rara vez será necesario, a menos que se trabaje en lenguaje ensamblador).

4.5 Despliegue

Un *nodo* es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional que, generalmente, tiene alguna memoria y, a menudo, capacidad de procesamiento. Gráficamente, un nodo se representa como un cubo.

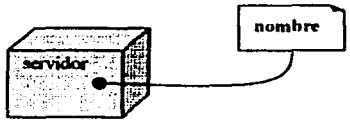


Figura 4.5.1 Nodos

Los nodos, al igual que los componentes, pertenecen al mundo material y son bloques de construcción importantes en el modelado de los aspectos físicos de un sistema. Los nodos se utilizan para modelar la topología del hardware sobre el que se ejecuta el sistema. Un nodo representa típicamente un *procesador* o un *dispositivo* sobre el que se pueden desplegar los componentes.

Cuando se diseña un sistema con gran cantidad de software, hay que considerar tanto su dimensión lógica como la física. En la parte lógica aparecen cosas como clases, interfaces, e interacciones. En la parte física se encuentran los componentes (que representan los empaquetamientos físicos de esos elementos lógicos) y los nodos (que representan el hardware sobre el que se despliega y ejecutan esos componentes).

Nombres

Cada nodo debe tener un nombre que lo distingue del resto de nodos. Un *nombre* es una cadena de texto. Este nombre solo se denomina *nombre simple*; un *nombre de camino* consta del nombre del nodo precedido del nombre del paquete en el que se encuentra. Normalmente, un nodo se dibuja mostrando sólo su nombre.

Nodos y componentes

En muchos aspectos, los nodos se parecen a los componentes: ambos tienen nombres; ambos pueden participar en relaciones de dependencia, generalización y asociación; ambos pueden anidarse; ambos pueden tener instancias; ambos pueden participar en interacciones. Sin embargo, hay algunas diferencias significativas entre los nodos y los componentes:

1. Los componentes son los elementos que participan en la ejecución de un sistema; los nodos son los elementos donde se ejecutan los componentes.
2. Los componentes representan el empaquetamiento físico de los elementos lógicos; los nodos representan el despliegue físico de componentes.

La primera diferencia es la más importante. Expresando con sencillez, en los nodos se ejecutan los componentes; los componentes son las cosas que se ejecutan en los nodos.

La segunda diferencia sugiere una relación entre clases, componentes y nodos. En particular, un componente es la materialización de un conjunto de elementos lógicos, tales como clases, y un nodo es la localización sobre la que se despliegan los componentes. Una clase puede ser implementada por uno o más componentes y, a su vez, un componente puede desplegarse sobre uno o más nodos. Como se muestra en la figura 4.5.1.2, la relación entre un nodo y el componente que despliega puede mostrarse explícitamente con una relación de dependencia. La mayoría de las veces no se necesitará visualizar estas relaciones gráficamente, pero se mantendrá como una parte de la especificación del nodo.

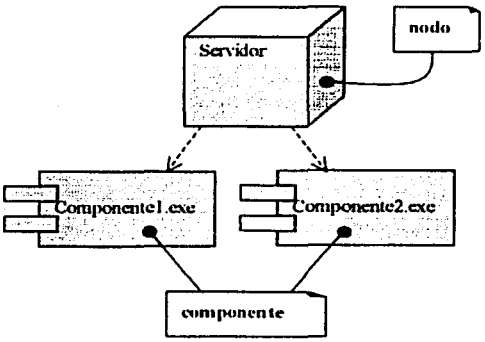


Figura 4.5.2. Nodos y componentes

Organización de nodos

El tipo más común de la relación entre nodos es la asociación. En este contexto, una asociación representa una conexión física entre nodos, como puede ser una conexión Ethernet, una línea en serie o un bus compartido, como se muestra en la figura 4.5.3. Se pueden utilizar asociaciones incluso para modelar conexiones indirectas, tales como un enlace por satélite entre procesadores distintos.

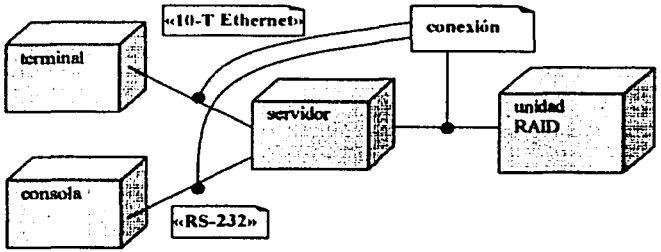


Figura 4.5.3. Conexiones.

Como los nodos son similares a las clases, se dispone de toda la potencia de las asociaciones. Esto significa que se pueden incluir roles, multiplicidad y restricciones.

4.5.1 Diagrama de Despliegue

Los *diagramas de despliegue* son uno de los dos tipos de diagramas que aparecen cuando se modelan los aspectos físicos de los sistemas orientados a objetos. Un diagrama de despliegue muestra la configuración de nodos que participan en la ejecución y de los componentes que residen en ellos

Los diagramas de despliegue se utilizan para modelar la vista de despliegue estática de un sistema. La mayoría de las veces, esto implica modelar la topología del hardware sobre el que se ejecuta el sistema. No sólo son importantes para visualizar, especificar y documentar sistemas empotrados, sistemas cliente/servidor y sistemas distribuidos, sino también para gestionar sistemas ejecutables mediante ingeniería directa e inversa.

Con UML (*Unified Modeling Language*), los diagramas de despliegue se utilizan para visualizar los aspectos estáticos de nodos físicos y sus relaciones y para especificar sus detalles para la construcción, como se muestra en la figura 4.5.1.

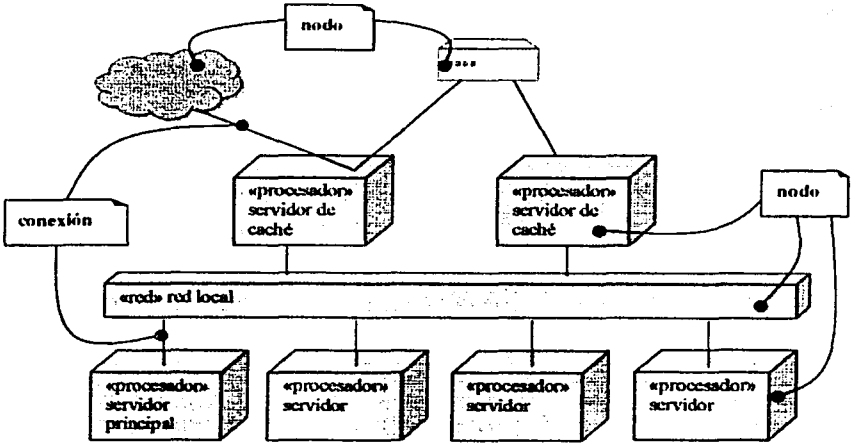


Figura 4.5.1.1. Un diagrama de despliegue.

Cuando se modela la vista de despliegue estática de un sistema, normalmente se utilizarán los diagramas de despliegue de una de las tres siguientes maneras.

1. Para modelar sistemas empotrados.

Un sistema empotrado es una colección de hardware con gran cantidad de software que interactúa con el mundo físico. Los sistemas empotrados involucran software que controla dispositivos como motores, actuadores y pantallas y que, a su vez, están controlados por estímulos externos tales como entradas de sensores, movimientos y cambios de temperatura. Los diagramas de despliegue se pueden utilizar para modelar los dispositivos y los procesadores que comprenden un sistema empotrado

2. Para modelar sistemas cliente/servidor.

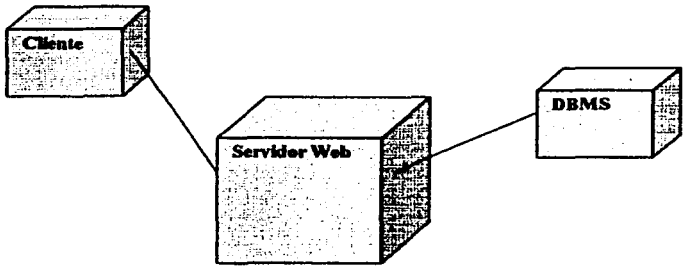
Un sistema cliente/servidor es una arquitectura muy extendida que se basa en hacer una clara separación de intereses entre la interfaz de usuario del sistema (que reside en el cliente) y los datos persistentes del sistema (que reside en el servidor). Los sistemas cliente/servidor son un extremo del espectro de los sistemas distribuidos y requieren tomar decisiones sobre la conectividad de red de los clientes a los servidores y sobre la distribución física de los componentes software del sistema a través de los nodos. La topología de tales sistemas se puede modelar mediante diagramas de despliegue.

3. Para modelar el sistema completamente distribuido.

En el otro extremo del espectro de los sistemas distribuidos se encuentran aquellos que son ampliamente, si no totalmente, distribuidos y que, normalmente, incluyen varios niveles de servidores. Tales sistemas contienen a menudo varias versiones de los componentes software, algunos de los cuales pueden incluso migrar de nodo en nodo. El diseño de tales sistemas requiere tomar decisiones que permitan un cambio continuo de la topología del sistema. Los diagramas de despliegue se pueden utilizar para visualizar la topología actual del sistema y la distribución de componentes, para razonar sobre el impacto de los cambios en esa topología.

A continuación se muestra el diagrama de instalación del sistema para el manejo de la información de los artículos publicados PEMEX.

4.5.1.1 DIAGRAMA DE INSTALACIÓN POR INTERNET.



4.6 Pruebas

La importancia de la *prueba* del software y sus implicaciones con la calidad del software no se pueden sobrevalorar.

La *prueba de software* es un elemento crítico para la garantía de calidad del software y representa un último repaso de las especificaciones del diseño y de la codificación.

La creciente aparición del software como un elemento más de muchos sistemas y la importancia de los "costes" asociados a un fallo del mismo están motivando la creación de pruebas minuciosas y bien planificadas. No es raro que una organización de desarrollo de software gaste el 40 por ciento del esfuerzo total de un proyecto en la prueba. ¡En casos extremos, la prueba del software para actividades críticas (por ejemplo, control aéreo, control de reactores nucleares) puede costar de tres a cinco veces más que el resto de los pasos de la ingeniería del software juntos!

Objetivos de la prueba

Los objetivos para realizar las pruebas son:

1. La prueba es un proceso de ejecución de un programa con la intención de descubrir un error.
2. Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
3. Una prueba tiene éxito si descubre un error no detectado hasta entonces.

Los objetivos anteriores suponen un cambio dramático del punto de vista. Nos quitan la idea que normalmente se tiene de que una prueba tiene éxito si no descubre errores. El objetivo es diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo. Si la prueba se lleva a cabo con éxito (de acuerdo con el objetivo anteriormente establecido) descubrirá errores en el software. La prueba no puede asegurar ausencia de defectos, sólo puede demostrar que existen defectos en el software.

Tipos de pruebas

Las diversas pruebas a que debe ser sometido un sistema deben ser realizadas tanto por el equipo de desarrolladores, como por los usuarios, equipos de operación y mantenimiento en la implantación, aceptación y mantenimiento del sistema de información. Los tipos de pruebas que deben realizarse son:

- Pruebas Unitarias.
- Pruebas d Integración.
- Pruebas del Sistema.
- Pruebas de Implantación.
- Pruebas de Aceptación.
- Pruebas de Regresión.

Pruebas Unitarias

Las pruebas unitarias tienen como objetivo la funcionalidad y estructura de cada componente individualmente una vez que ha sido codificado.

Las pruebas unitarias constituyen la prueba inicial de un sistema y las demás pruebas deben apoyarse sobre ellas. Existen dos enfoques principales para el diseño de casos de prueba:

- Enfoque estructural o de caja blanca.
- Enfoque funcional o de caja negra.

En enfoque que suele adoptarse para una prueba unitaria está claramente orientado al diseño de casos de caja blanca, aunque se complemente con caja negra. El hecho de incorporar casos de caja blanca se debe, por una parte, a que el tamaño del componente es apropiado para poder examinar toda la lógica y por otra, a que el tipo de defectos que se busca, coincide con los propios de la lógica detallada en los componentes. Los pasos para llevar a cabo las pruebas unitarias son los siguientes:

- Ejecutar todos los casos de prueba asociados a cada verificación establecida en el plan de pruebas, registrando su resultado. Los casos de prueba deben contemplar tanto las condiciones válidas y esperadas como las inválidas e inesperadas.
- Corregir los errores o defectos encontrados y repetir las pruebas que los detectaron. Si se considera necesario, debido a su implicación o importancia, se repetirán otros casos de prueba ya realizados con anterioridad.

La prueba unitaria se da por finalizada cuando se hayan realizado todas las verificaciones establecidas y no se encuentre defecto alguno, o bien se determine su suspensión.

Pruebas de Integración

El objetivo de las pruebas de integración es verificar el correcto ensamblaje entre los distintos componentes una vez que han sido probadas unitariamente con el fin de comprobar que interactúan correctamente a través de sus interfaces, tanto internas como externas, cubren la funcionalidad establecida y se ajustan a los requisitos no funcionales especificados en las verificaciones correspondientes.

En las pruebas de integración se examinan las interfaces entre grupos de componentes o subsistemas para asegurar que son llamados cuando es necesario y que los datos o mensajes que se transmiten son los requeridos.

Debido a que en las pruebas unitarias es necesario crear módulos auxiliares que simulen las acciones de los componentes invocados por el que se está probando y a que se han de crear componentes "conductores" para establecer las precondiciones necesarias, llamar al componente objeto de la prueba y examinar los resultados de la prueba, a menudo se combinan los tipos de prueba unitarias y de integración.
Los tipos fundamentales de integración son los siguientes:

- Integración incremental:
- Integración no incremental

Integración incremental

Se combina el siguiente componente que se debe probar con el conjunto de componentes que ya están probados y se va incrementando progresivamente el número de componentes a probar.

Integración no incremental

Se prueba cada componente por separado y posteriormente se integran todos de una vez realizando las pruebas pertinentes. Este tipo de integración se denomina también Big-Bang (gran explosión).

Pruebas del Sistema

Las pruebas del Sistema tienen como objetivo ejercitar profundamente el sistema comprobando la integración del sistema de información globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto del sistema de información con los que se comunica.

Son pruebas de integración del sistema de información completo y permiten probar el sistema en su conjunto y con otros sistemas con los que se relaciona para verificar que las especificaciones funcionales y técnicas se cumplen. Dan una visión muy similar a su comportamiento en el entorno de producción.

Pruebas de Implantación

El objetivo de las pruebas de implantación es comprobar el funcionamiento correcto del sistema integrado de hardware y software en el entorno de operación y permitir al usuario que, desde el punto de vista de operación, realice la aceptación del sistema una vez instalado en su entorno real y en base al cumplimiento de los requisitos no funcionales especificados.

Una vez que hayan sido realizadas las pruebas del sistema en el entorno de desarrollo, se llevan a cabo las verificaciones necesarias para asegurar que el sistema funcionará correctamente en el entorno de operación. Debe comprobarse que responde satisfactoriamente a los requisitos de rendimiento, seguridad, operación y coexistencia con el resto de los sistemas de la instalación para conseguir la aceptación del usuario de operación.

Las pruebas de seguridad van dirigidas a verificar que los mecanismos de protección incorporados al sistema cumplen su objetivo; las de rendimiento a asegurar que el sistema responde satisfactoriamente en los márgenes establecidos en cuanto tiempos de respuesta, de ejecución y de utilización de recursos, así como los volúmenes de espacio en disco y capacidad; por último con las pruebas de operación se comprueba que la planificación y

control de trabajos del sistema se realiza de acuerdo con los procedimientos establecidos, considerando la gestión y control de las comunicaciones y asegurando la disponibilidad de los distintos recursos.

Pruebas de Aceptación

El objetivo de las pruebas de aceptación es validar que un sistema cumple con el funcionamiento esperado y permitir al usuario de dicho sistema que determine su aceptación, desde el punto de vista de su funcionalidad y rendimiento.

Estas pruebas van dirigidas a comprobar que el sistema cumple los requisitos de funcionalidad esperados, recogidos en el catálogo de requisitos y en los criterios de aceptación del sistema de información y conseguir así la aceptación final del sistema por parte del usuario.

Pruebas de Regresión

El objetivo de las pruebas de regresión es eliminar el efecto onda; es decir, comprobar que los cambios sobre un componente de un sistema de información no introducen un comportamiento no deseado o errores adicionales en otros componentes no modificados.

Las pruebas de regresión se deben llevar a cabo cada vez que se hace un cambio en el sistema, tanto para corregir un error como para realizar una mejora. No es suficiente probar sólo los componentes modificados o añadidos, o las funciones que en ellos se realizan, sino que también es necesario controlar que las modificaciones no produzcan efectos negativos sobre él mismo u otros componentes.

CONCLUSIONES

El problema que se planteo al inicio del presente trabajo, ha sido resuelto con el desarrollo de un sistema de software. Proporcionando asi una solución óptima a la labor tediosa, de altos costos, poco segura, de gran pérdida de tiempo y demasiado lenta en su entrega, en el área de investigación.

Ya no será una labor tediosa, puesto que, no será necesario llamar por teléfono a la STDP (*Subdirección Tecnológica de Desarrollo Profesional*) de Villahermosa Tabasco, para pedir información sobre los articulos publicados. El sistema centrará toda su información en una base de datos almacenada en un servidor de PEMEX, y desde cualquier máquina que tenga conexión a la intranet de PEMEX podrá consultar la lista de todos los articulos que han sido publicados. Evidentemente que al tener acceso a la información desde cualquier máquina, los costos se reducen significativamente.

La manera en que se resolvió el problema, fue haciendo una revisión de la tecnologia actual de redes, para poder enlazar los distintos regionales de PEMEX. Actualmente PEMEX cuenta con conexión a internet, formando una intranet entre las distintas regiones, permitiendo asi una comunicación segura y rápida entre su personal.

Como ya se cuenta con la infraestructura tecnológica para lograr la comunicación entre todo el personal de PEMEX, la siguiente tarea consistió en buscar la tecnologia que soportaria el desarrollo de Sistemas para la Web.

Java fue el lenguaje que se utilizó para la codificación de los programas, ya que es un lenguaje orientado a objetos y por lo tanto cuenta con el soporte para programar los modelos obtenidos en el diseño del presente sistema. Se utilizó principalmente para la manipulación de los datos, es decir para agregar, consultar, modificar y eliminar información de la base de datos.

Además de ser un lenguaje orientado a objetos, una de sus principales caracteristicas es que es multiplataforma, permitiendo de esta manera el manejo de los datos, desde casi cualquier sistema operativo; que es uno de los requisitos necesarios para navegar en la Web.

Es necesario mencionar que todo el proceso que se llevo a cabo, para realizar el sistema, esta fundamentado, bajo una metodología de diseño de sistemas, que cubre todos los ciclos de vida del sistema; esto es para obtener un sistema flexible en su crecimiento, confiable y de fácil mantenimiento. Ésta metodologia ha sido una guia para el desarrolló del sistema, permitiendo reducir los costos económicos que se originarian si no se contara con la misma. Aplicando de esta forma la ingenieria en sistemas computacionales, en la solución de un problema, y logrando de este modo el bienestar del personal de PEMEX y de la división de Ingenieria Petrolera de la UNAM.

En el presente trabajo se desarrolló un sistema de software el cual cumplió con el objetivo planteado inicialmente, logrando eliminar las siguientes actividades:

- No será necesario hablar constantemente por teléfono a la STDP de Villahermosa Tabasco, para solicitar un articulo.

  Esto es debido a que la información sobre los articulos publicados se encuentra dispersa entre los regionales de PEMEX, la STDP de Villahermosa por cada llamada telefónica en la que le solicitan un articulo en particular, tiene que realizar una búsqueda exhaustiva en cada región, y si en ese momento se encuentra el personal que le puede dar informes hace el pedido del articulo, de otro modo tiene que esperar y volver a llamar.

- No será necesario mandar constantemente el mismo articulo solicitado (a diferentes correos electrónicos y a diferentes investigadores).

- No será necesario utilizar los servicios de mensajeria para enviar aquellos articulos que son muy grandes de tamaño, y no se pueden enviar por correo electrónico. Al desaparecer esta actividad se reduce significativamente la lentitud e inseguridad ocasionada en la entrega de los articulos al personal.

A demás de resolver los problemas antes citados, el sistema cumple con:

- Proporcionar seguridad a la información, por medio de claves de acceso.
- Estar dentro de los limites tolerables, para sistemas de consulta de información. Esto es, los tiempos de respuesta en la mayor parte de los casos dependerá del servidor Web.
- Será fácil de manejar, por cuanto se usará en su construcción elementos estándares de las aplicaciones, bajo interfaz gráfica.
- Considerando que la construcción del sistema estará basada completamente en la documentación que se generará, se estima que cualquier modificación se podrá ejecutar sin complicaciones facilitando así su mantenimiento.
- Debido a que el sistema que se desarrollo, se construyó con Java, se garantizará un alto porcentaje de compatibilidad en la ejecución del sistema en diferentes plataformas.
- Y principalmente se cumple con el objetivo de vincular a la UNAM con la infraestructura de PEMEX.

Así como se logró reducir costo, tiempo y con ello el oportuno y fácil acceso a la información.

BIBLIOGRAFÍA

Análisis y diseño orientado a objetos con Aplicaciones
Grady Booch
2ª Edición Addison-Wesley/Diaz de Santos

Análisis y diseño orientado a objetos
James Martin, James J. Odell
Prentice Hall

Construcción de software orientado a objetos
Bertrand Meyer
2ª Edición Prentice may

Métodos orientados a objetos
Ian Graham
2ª Edición Addison-Wesley/Diaz de Santos

El lenguaje unificado de modelado: Manual de referencia
James Rumbaugh, Ivar Jacobson, Grady Booch
Addison-Wesley/Diaz de Santos

El proceso unificado de desarrollo de software
James Rumbaugh, Ivar Jacobson, Grady Booch
Addison-Wesley/Diaz de Santos

El lenguaje unificado de modelado
James Rumbaugh, Ivar Jacobson, Grady Booch
Addison-Wesley/Diaz de Santos

UML y patrones: Introducción y diseño orientado a objetos
Graig Larman
Prentice Hill 2ª Edición

Ingenieria del software
Pressman
Mc Graw-Hill

Base de datos Modelos, lenguajes, diseño
James L.Jonson
Oxford

Fundamentos y modelos de Base de Datos
De Miguel, A. y Piattini, M

Diseño de Base de datos relacionales
De Miguel, A. y Piattini, M

Concepto y diseño de base de datos relacionales del Modelo E/R al modelo relacional
De Miguel, A. y Piattini, M
Ra-Ma

JavaServer Pages
Agustín Froufe
RA-MA

Multimedia e internet
Daniel Morata Sebastián
Paraíno

Los secretos de Intranet para UNIX y NT a tu alcance
Sharon Crawford, Charlie Russel
Anaya Multimedia, S.A

Aprendiendo TCP/IP en 14 días
Tim Parker
2ª Edición

Todo acerca de redes de computación
Kevin Stoltz
Prentice Hall
Edición en español

Internet paso a paso hacia la autopista de la información
Gonzalo Ferreira C.

Construya su propia Intranet
Tim Evans
Prentice Hall

Informática: Las computadoras en la sociedad
Radlow James
Mc Graw-Hill

Redes de ordenadores
Andrew S. Tanenbaum
Prentice Hall

Tecnología de interconectividad de redes
Merilee ford, H. Kim Lew, Steve Spainer
Prentice Hall

Redes de computadores, protocolos, normas e interfaces
Uyless Black
RA-MA

Redes para proceso distribuido
Jesús Gracia Tomás, Santiago Ferrand, Mario Piattini, Velthuis
RA-MA

GLOSARIO DE TÉRMINOS

Administrador. Es la persona encargada de ingresar los nuevos artículos publicados, así como de actualizar y borrar los ya existentes en la base de datos.

Artículos PEMEX. Son las publicaciones que los investigadores de PEMEX realizan sobre diversos temas de ingeniería petrolera.

Autor. Es la persona que realizó la investigación de un artículo.

Boletín. Es la publicación de la información relacionada con el departamento STDP y sus publicaciones, realizada mensualmente.

Coautor. Es la persona que participa con el autor de un artículo para realizar la investigación del mismo.

Lector. Son los investigadores, alumnos de la maestría de petrolera y académicos que pueden realizar consultas de los artículos publicados, que existen en el sistema.

Posgrado de Ingeniería Petrolera. Departamento de Ingeniería Petrolera.

Título. Es el nombre con el que se publica un artículo de PEMEX.

PEMEX. La siglas significan: Petróleos Mexicanos

STDP. Las siglas significan: Subdirección Tecnológica de Desarrollo Profesional.

Apendíce A

Manual de Usario

El manual de usuario es uno de los documentos finales que se entrega al cliente. En este documento se explica de forma detallada la manera en la que el usuario del sistema deberá interactuar con el mismo.

Objetivo

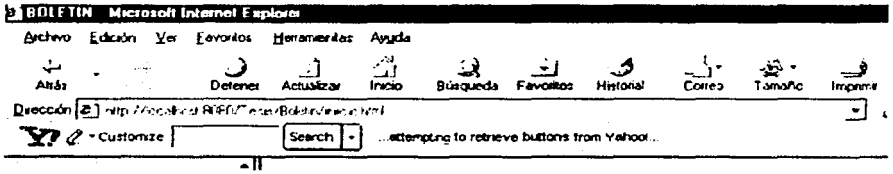
El presente manual contiene una descripción funcional del sistema para la administración y mantenimiento de los artículos públicados PEMEX, así como del boletín elaborado mensualmente por la Subdirección de Desarrollo Profesional de PEMEX.

El manual está dirigido a los alumnos y académicos de la sección de Ingeniería Petrólera, así como al personal de PEMEX y de la Subdirección de Desarrollo Profesional de PEMEX.

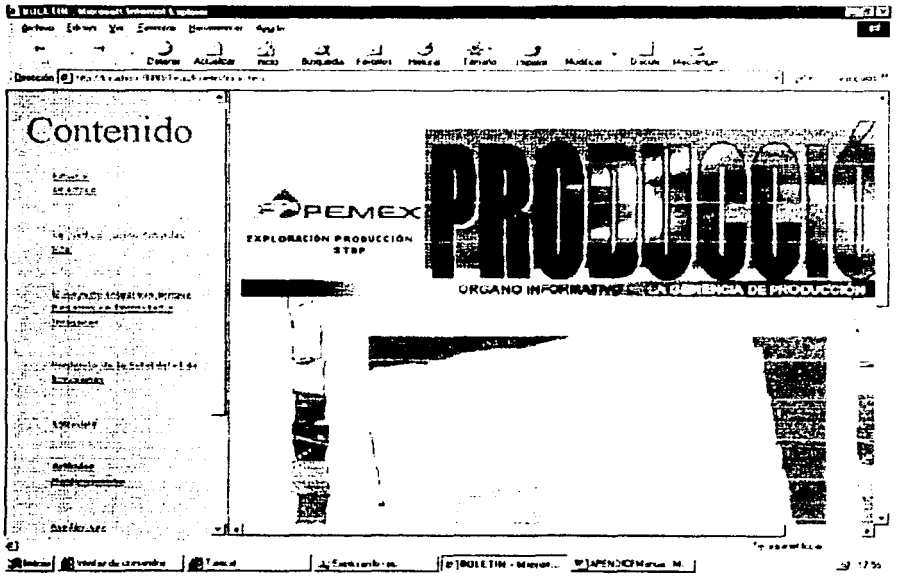
Boletín

Inicio de una sesión en Internet

Para entar a la información contenida en el boletín, hay que abrir cualquier navegador de internet y tecleear la siguiente dirección: http localhost:8080/Boletin/inicio.html, el localhost:8080 apunta al servidor, donde se encuentran almacenada la información



Una vez iniciada la sesión, el navegador de Internet desplegará la información contenida en el boletín



La información que muestra el boletín se encuentra distribuida de la siguiente manera:

a) Un frame izquierdo, donde está el índice o contenido del boletín.
b) Un frame central, donde se despliega la información relacionada con el índice del boletín.

Contenido

El índice o contenido del boletín está formado por los siguiente rubros:

- Editorial
- Rescate de experiencias
- Artículos a presentarse EXITEP
- Tecnología aplicable
- Entrevista del mes
- Artículos públicados
- Noticias
- Eventos y conferencias
- Buzón

Cada uno de estos rubros son páginas informativas de la Subdirección de Desarrollo Profesional PEMEX. Unicamente hay que dar un clic en la liga deseada para obtener la información.

En el caso de los "Artículos públicados", existen dos opciones:

a) Artículos.
b) Mantenimiento.

La liga de *Mantenimiento* llevará a una pantalla de *Login* para accesar a la parte de administración del sistema.

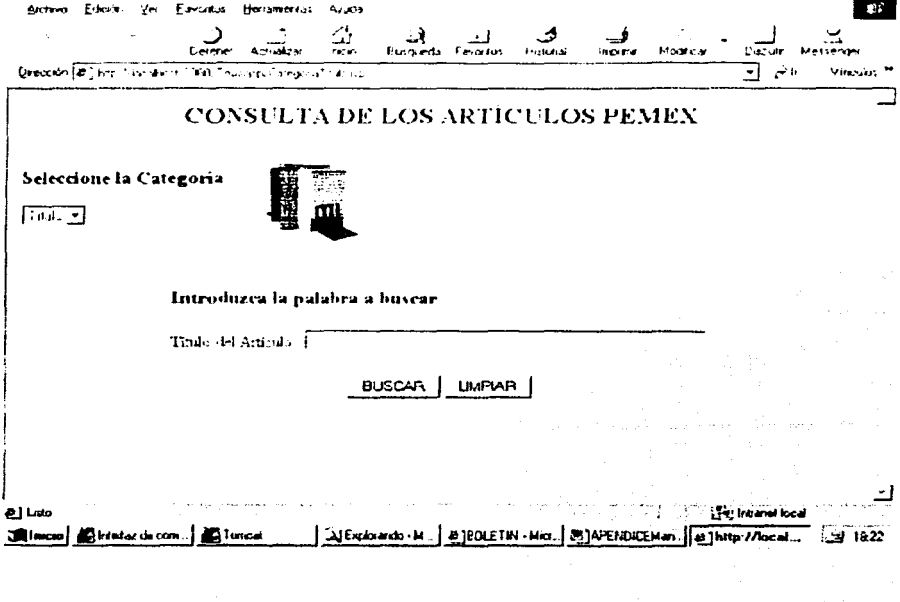
La liga *Artículos* llevará a una pantalla de *consulta*, la cual se explicará a continuación.

Consulta de los Artículos publicados

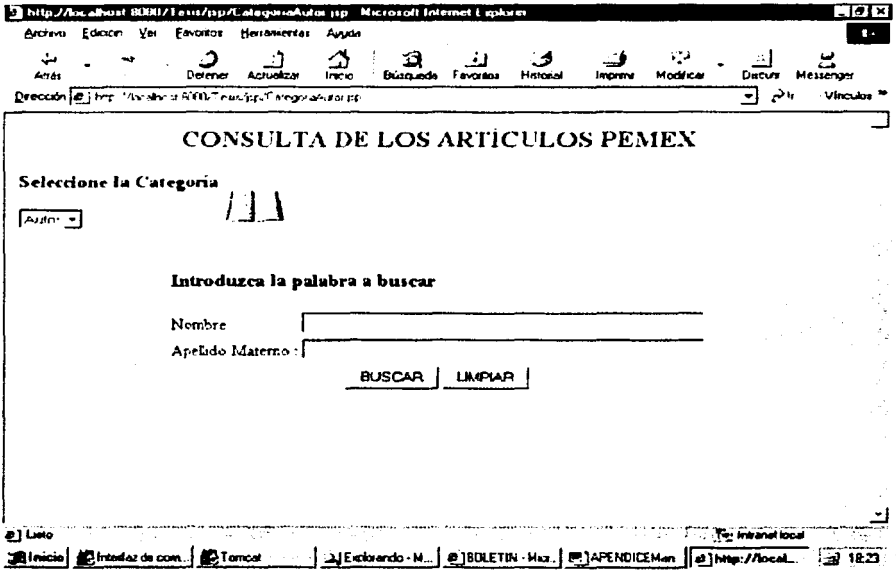
Una vez que se dio un click en la liga *Articulos* del rubro "Articulos publicados" del indice del boletin, aparecerá la pantalla de *Consulta de los Articulos Publicados*. Esta pantalla permite realizar consultas tanto por título, como por autor del articulo.

Para realizar una consulta primero hay que seleccionar la categoría.

Si selecciona la categoria de *título* aparecerá la pantalla de consulta, como la que se muestra a continuación



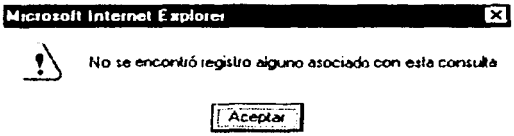
Si selecciona la categoria de *autor* aparecerá la pantalla de consulta, como la que se muestra a continuación.



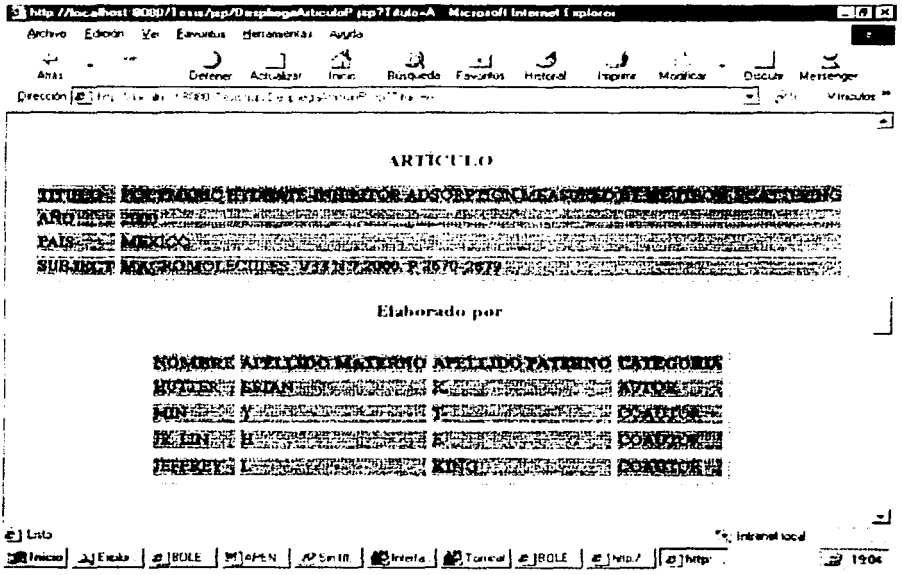
Para ambas categorías, al realizar una búsqueda de los artículos, es necesario introducir la palabra a buscar y oprimir el botón de *BUSCAR*.

Si no se introduce palabra alguna en las cajas de texto, el sistema enviará el siguiente mensaje.

Al introducir la palabra a buscar, si no existe información relacionada con la misma, el sistema enviará el siguiente mensaje.



En caso contrario, desplegará la información solicitada, como se muestra en la pantalla de *Resultado de la búsqueda* que a continuación se muestra



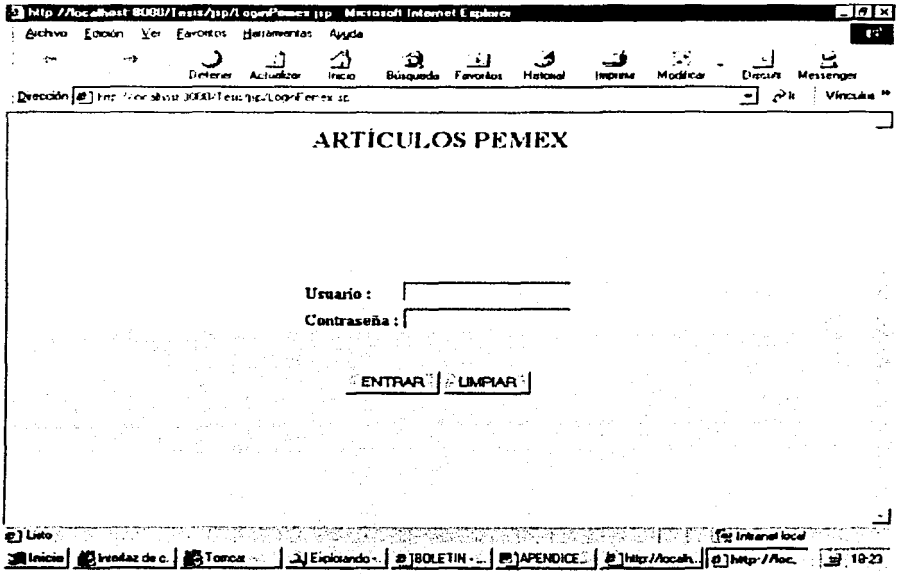
En esta pantalla se desplegarán los datos del articulo asi como los nombres de las personas que lo elaboraron.

La información está organizada en tablas, separando los articulos, de los autores.

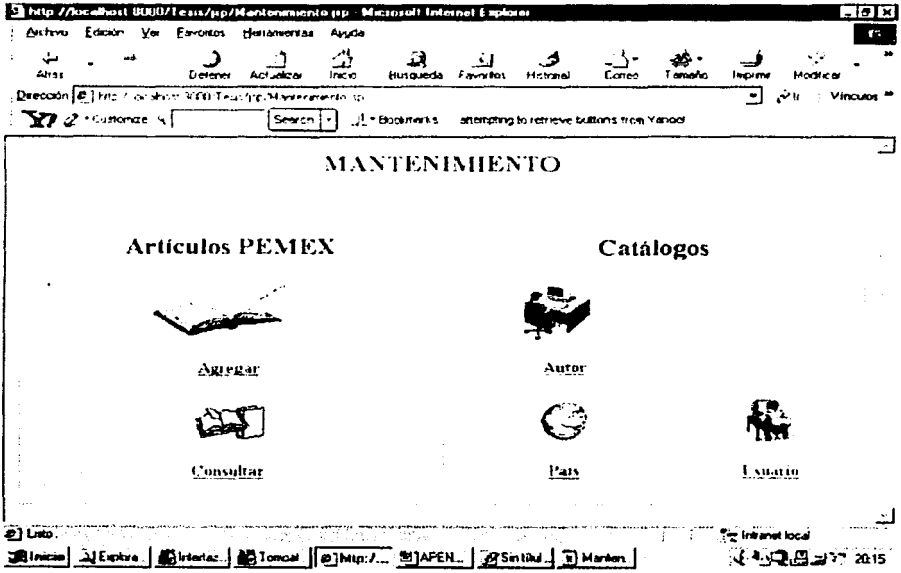
Para realizar una búsqueda en esta página, unicamente hay que desplazarse de arriba hacia abajo, utilizando la barra que está ubicada en la parte derecha de la pantalla.

MANTENIMIENTO DE LOS ARTÍCULOS PEMEX

Para el mantenimiento de los artículos almacenados en el sistema hay que seleccionar en el indice del boletin, dentro del rubro "Artículos publicados", la liga *Mantenimiento*. Y en seguida aparecerá la pantalla del *Logn*. Aquí es necesario introducir el **usuario** y **contraseña** correcto para poder ingresar al sistema.



Si el usuario y contraseña, son correctos se permitirá el acceso al sistema y aparecerá la pantalla de *Mantenimiento*



La pantalla de mantenimiento está integrada de la manera siguiente:

Para los Artículos PEMEX, se tienen dos opciones que son:

- Agregar
- Consultar

Para los Catálogos, el sistema cuenta con tres que son:

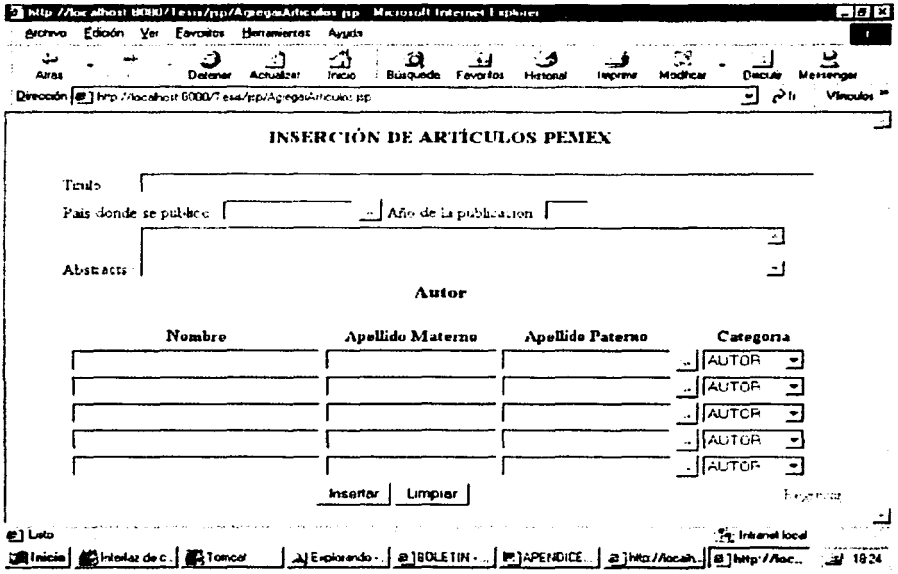
- Autor
- País
- Usuario

A continuación se describirá cada uno de estos puntos.

Artículos PEMEX

Agregar

Al seleccionar la opción *Agregar* de la pantalla *Mantenimiento*, aparecerá la pantalla de *Inserción de Artículos PEMEX*.

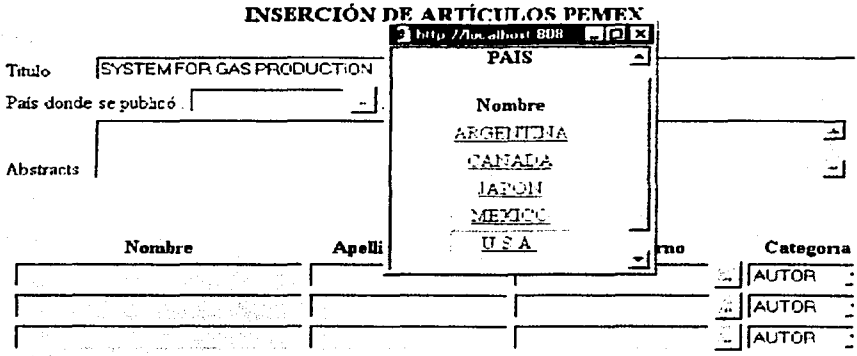


Esta pantalla de *inserción de artículos PEMEX* nos permite agregar nuevos artículos al sistema.

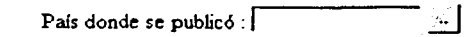
Para egregar un artículo nuevo hay que llenar cada uno de las cajas de texto que aparecen en la pantalla.

Para el caso de las cajas de texto que corresponden a los artículos hay que teclear la información solicitada, excepto para el país, aquí hay que darle un click al botón que se encuentra ubicado en la parte derecha de la caja, como se muestra en la figura.

Y entonces aparecerá una nueva ventana que desplegará una lista con los nombres de los países. Como a continuación se indica.

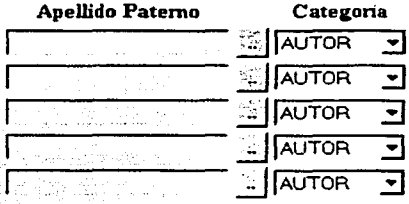


Aquí unicamente hay que darle un click a la información deseada y esta aparecerá en la caja de texto. Como se ilustra a continuación.



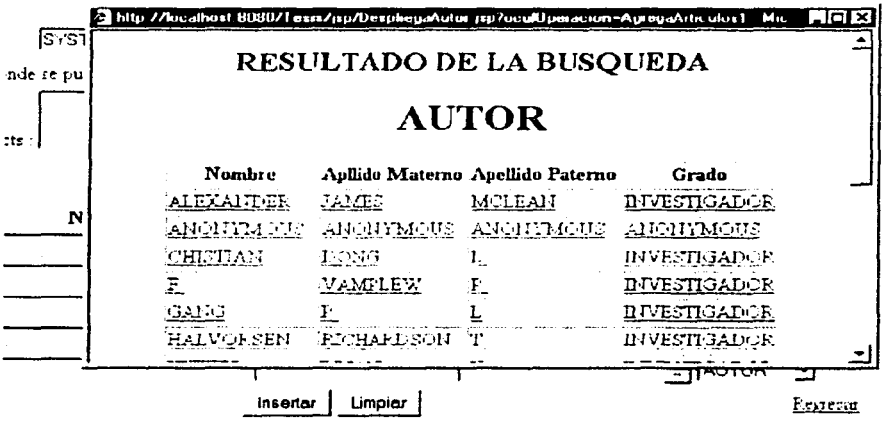
Una vez que se obtuvo la información solicitada con respecto a los datos de los articulos hay que llenar las cajas de texto del *Autor*, con los nombres de las personas que participaron en la elaboración del articulo.

Para llenar estas cajas de texto hay que darle un click al botón que está ubicado en la parte derecha de las cajas de texto que tienen la etiqueta de *Apellido Paterno*, como se muestra en la siguiente figura.



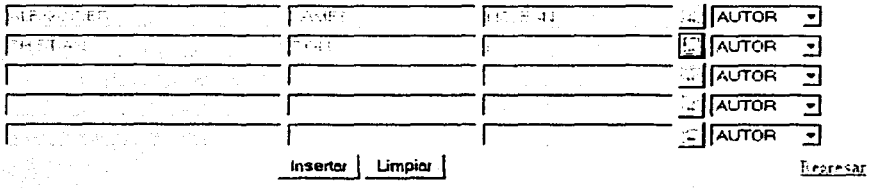
Y a continuación aparecerá una nueva ventana que muestra una tabla con los nombres, el apellido materno, el apellido paterno y grado, de cada una de las personas que han participado en la elaboración de los artículos

INSERCIÓN DE ARTÍCULOS PEMEX



En esta ventana unicamente hay que darle un click en cualquiera de los datos mostrados y entonces la ventana se cerrará, llenando las cajas de texto. Como se indica a continuación.

Autor



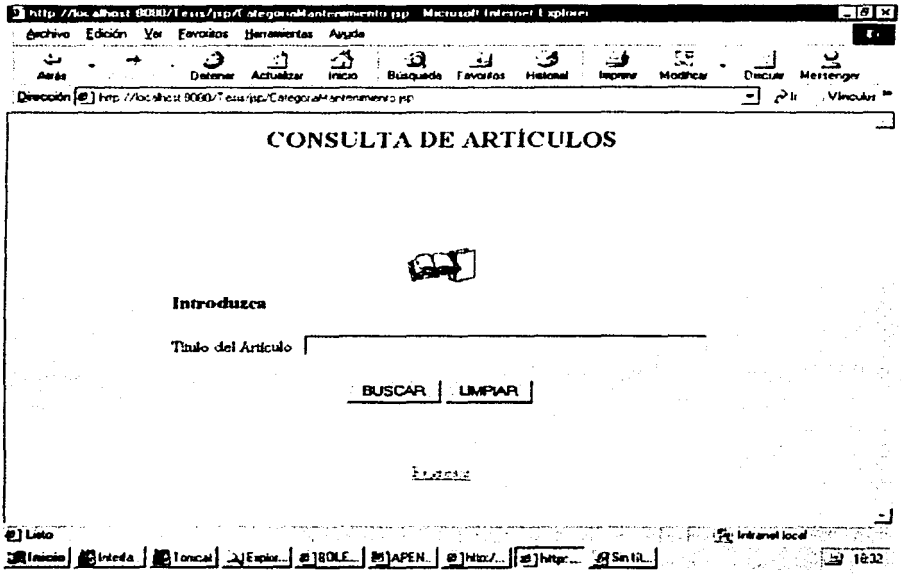
Como se puede observar en la figura anterior, para cada artículo se pueden insertar cinco autores a la vez.

Ya que se agregaron los primeros cinco autores es posible insertar otros cinco más, llenando nuevamente las cajas de texto para luego presionar el botón *Insertar*, y asi sucesivamente, hasta completar el número deseado de autores por insertar.

Artículos PEMEX

Consultar

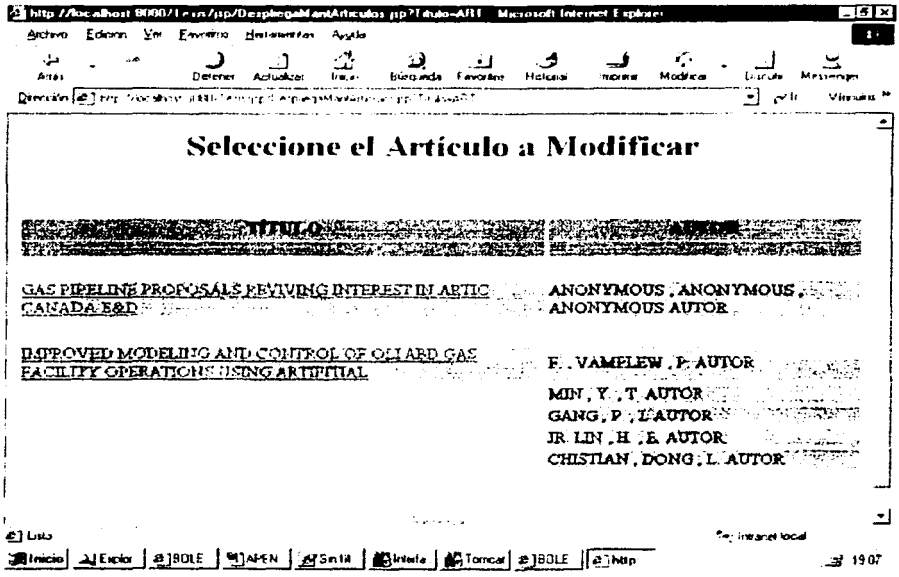
Al seleccionar la opción de *Consultar* de la pantalla de *Mantenimiento* aparecerá la pantalla de *Consulta de artículos*.



Esta pantalla me permite realizar consultas al sistema por titulo de articulo. En esta pantalla es necesario introducir la palabra deseada y oprimir el botón de *BUSCAR*.

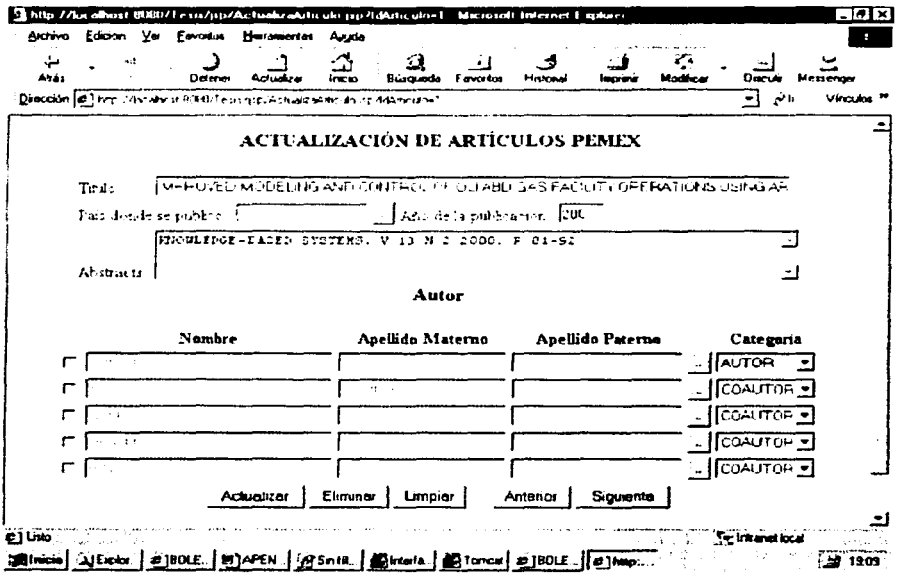
Si la palabra que se busca, no tiene información relacionada con ella, en la base de datos, el sistema mandará el siguiente mensaje.

Si la búsqueda fue exitosa mostrará la siguiente pantalla



En esta pantalla se muestra el titulo de cada artículo y los nombres de las personas que lo elaboraron.

La principal función de esta pantalla no es sólo de consulta, si no que al darle un clic al título de un artículo, nos lleva a una nueva pantalla, llamada *Actualización de artículos PEMEX*, como la que se muestra en la figura siguiente:



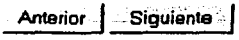
En esta ventana se despliega toda la información relacionada con un artículo, así como los nombres de todos las personas que participaron en la elaboración del mismo.

Aquí es posible actualizar lo datos de los artículos, así como eliminar o agregar autores.

Moverse entre registros

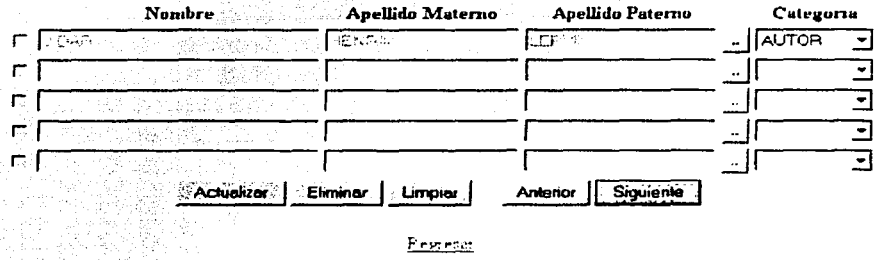
Como se observa en la pantalla anterior, únicamente es posible visualizar cinco autores a la vez, ¿Qué pasa si el artículo tiene relacionados más de cinco autores?

Para salvar esta dificultad, el sistema cuenta con dos botones que permiten al usuario desplazarse entre varios autores.



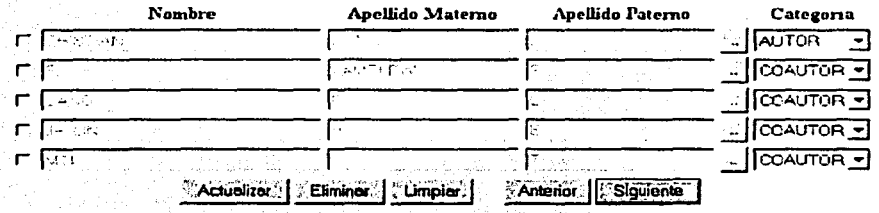
Una vez mostrados los primeros cinco autores, al presionar el botón de *Anterior*, el sistema mandará el siguiente mensaje:

Al presionar el botón *Siguiente*, el sistema mostrará los siguientes cinco autores.



Si no existen más autores que mostrar, al presionar el botón *Siguiente* el sistema muestra el siguiente mensaje:

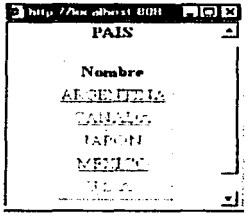
Para regresar a los cinco anteriores presionar el botón *Anterior* y a continuación aparecerán los cinco autores anteriores. Y así sucesivamente.



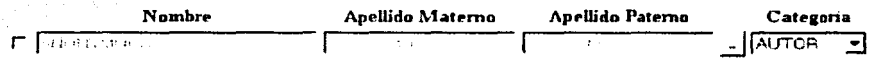
Acualizar

Para actualizar los datos de los artículos únicamente hay que modificar el texto deseado y darle un clic al botón de *Actualizar*.

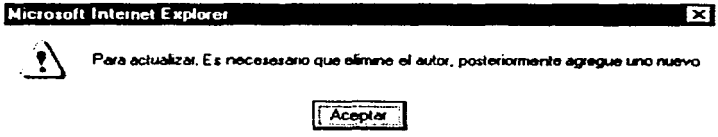
Para el caso de la caja de texto del *País* hay que darle un clic al botón adjunto y aparecerá una nueva ventana con los nombres de los países. Aquí hay que seleccionar el texto deseado y darle un clic para obtener la nueva información.



Para el caso de los autores, si se desea cambiar un autor por otro, el sistema no permite hacer el cambio directamente. Al darle un clic al botón que aparece adjunto a la caja de texto del *Apellido paterno*, como se muestra en la figura:



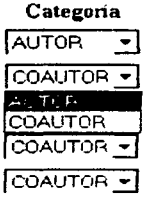
El sistema enviará un mensaje como el que a continuación se observa.



Para cambiar un autor por otro primero hay que eliminar el autor que se desea cambiar y luego agregar el nuevo autor.

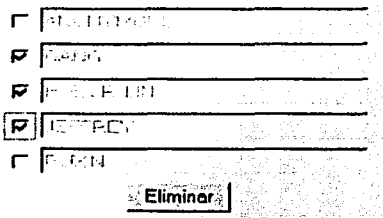
Una vez que se eliminó el autor y se agregó el nuevo autor, oprimir el botón *Actualizar*, para que los cambios realizados permanezcan en la base de datos.

También es posible cambiar la categoría del autor, aquí únicamente se selecciona la opción de Autor o Coautor y posteriormente se presiona el botón *Actualizar*.



Eliminar

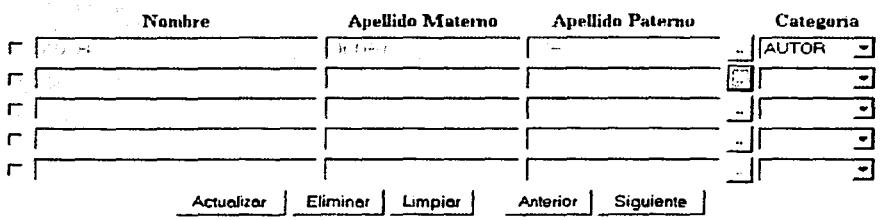
Para eliminar un autor hay que seleccionar los autores por eliminar. Para esto hay que dar un clic a la **caja** que aparece del lado izquierdo del nombre del autor y posteriormente presionando el botón *Eliminar*. Como se muestra en la siguiente figura.



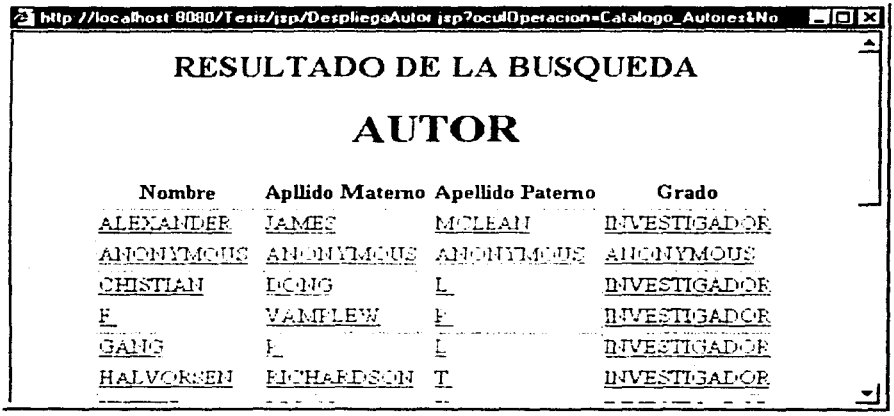
Si se desea eliminar el artículo completamente será necesario eliminar todos los autores relacionados con el artículo.

Agregar

Para agregar un nuevo autor presionar el botón que aparece en la parte derecha de la caja de texto *Apellido Paterno*, pero sólo en aquellas cajas que estén vacías.



Y a continuación aparecerá la siguiente ventana:

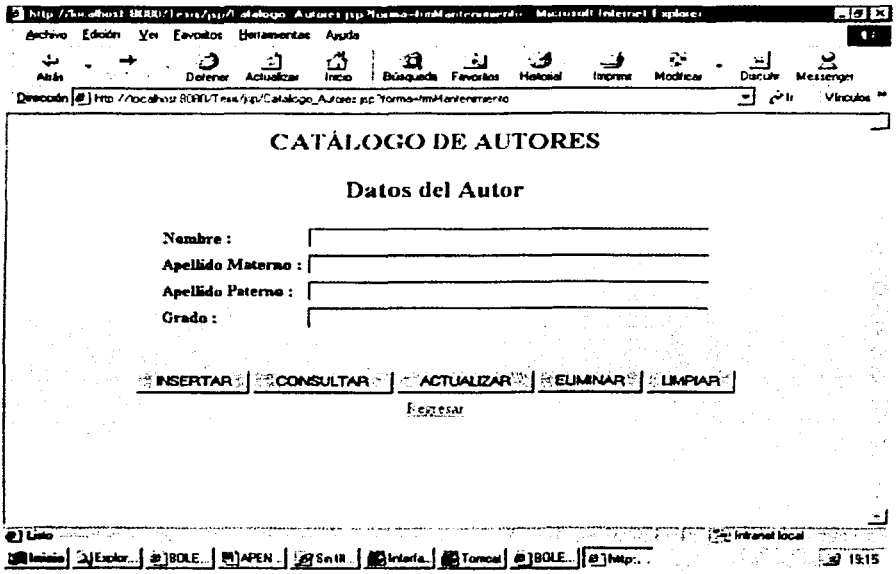


Aquí hay que seleccionar el nuevo autor pero éste no deberá estar ligado ya al artículo en cuestión es decir, el sistema no permitirá autores repetidos.

Posteriormente presionar el botón *Actualizar* para que el nuevo autor sea agregado al artículo.

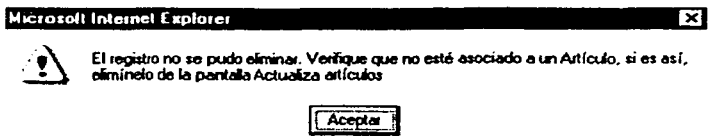
CATÁLOGO DE AUTORES

Al seleccionar la liga *Autor* de la pantalla de *Mantenimiento* se desplegará la pantalla de *Catálogo de Autores*.



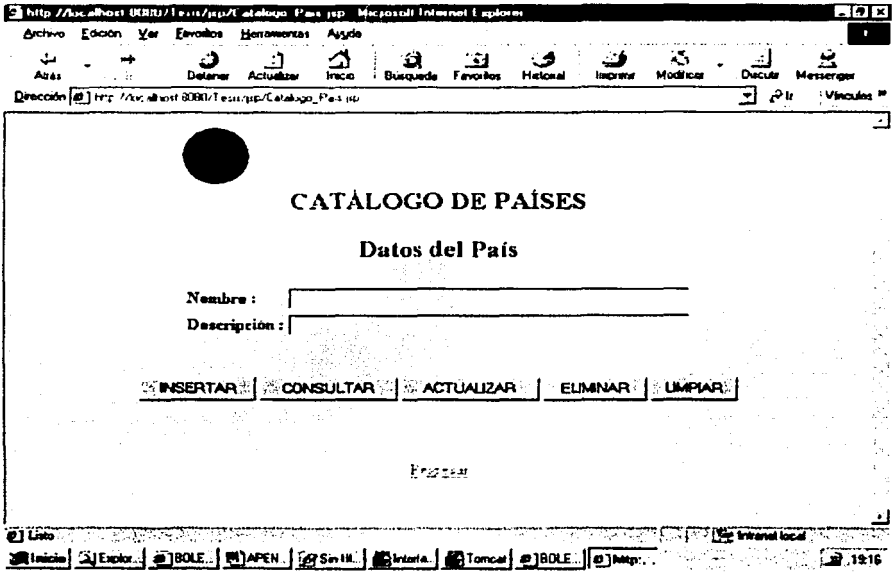
En esta pantalla se pueden agregar nuevos autores al sistema, así como eliminar y modificar los que actualmente existen.

En el caso de tratar de eliminar información que se encuentra ligada a un artículo, no se podrá realizar esta operación y el sistema enviará el siguiente mensaje.



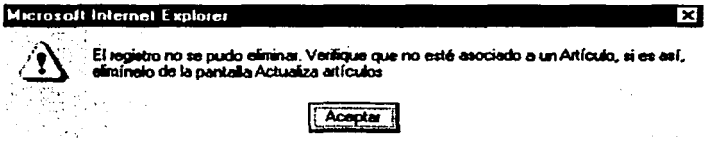
CATÁLOGO DE PAÍSES

Al seleccionar la liga *País* de la pantalla de *Mantenimiento* se desplegará la pantalla de *Catálogo de Países*



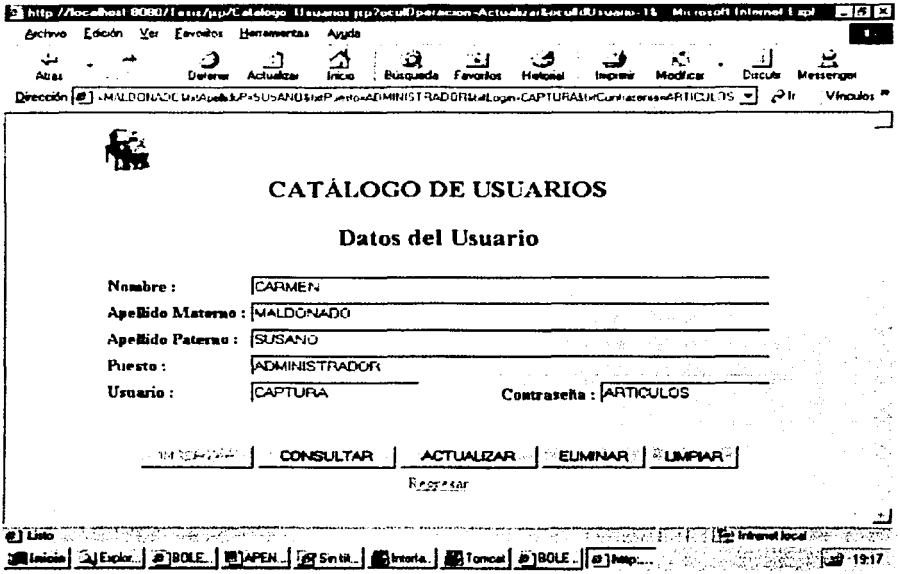
En esta pantalla se podrán agregar nuevos Países al sistema, así como eliminar y modificar los que actualmente existen.

En caso de querer eliminar un País que actualmente se encuentra ligado a un artículo, no se podrá eliminar y el sistema enviará el siguiente mensaje:



CATÁLOGO DE USUARIOS

Al seleccionar la liga *Usuario* de la pantalla de *Mantenimiento*, se desplegará la pantalla de *Catálogo de Usuarios*.



En esta pantalla se podrán agregar nuevos Usuarios al sistema, así como eliminar y modificar los que actualmente existen.

Esta pantalla sirve para controlar la seguridad proporcionada al mantenimiento del sistema de los Artículos publicados, ya que aquí se pueden agregar los nombres de las personas encargadas de administrar el sistema, así como de modificar al Usuario y su Contraseña, de los que ya existen en la base de datos.