Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

DISEÑO DE APLICACIONES WEB DISTRIBUIDAS EN N CAPAS UTILIZANDO PATRONES

José Manuel Ruiz Juárez

Asesorado por el: Ing. Jorge Armín Mazariegos

Guatemala, marzo de 2006

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DISEÑO DE APLICACIONES WEB DISTRIBUIDAS EN N CAPAS UTILIZANDO PATRONES

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

JOSÉ MANUEL RUIZ JUÁREZ
ASESORADO POR EL: ING. JORGE ARMÍN MAZARIEGOS

AL CONFERÍRSELE EL TÍTULO DE
INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, MARZO DE 2006

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	
VOCAL II	Lic. Amahán Sánchez Álvarez
VOCAL III	Ing. Julio David Galicia Celada
VOCAL IV	Br. Kenneth Issur Estrada Ruiz
VOCAL V	Br. Elisa Yazminda Vides Leiva
SECRETARIA	Inga. Marcia Ivonne Véliz Vargas

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Herbert René Miranda Barrios
EXAMINADOR	Ing. Luis Alberto Vettorazzi España
EXAMINADOR	Ing. Franklin Antonio Barrientos Luna
EXAMINADOR	Ing. Otto Amilcar Rodríguez Acosta
SECRETARIA	Inga. Gilda Marina Castellanos de Illescas

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DISEÑO DE APLICACIONES WEB DISTRIBUIDAS EN N CAPAS UTILIZANDO PATRONES,

tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha junio de 2004.

José Manuel Ruiz Juárez

AGRADECIMIENTOS A:

DIOS	Por darme la vida y vida en abundancia llena de bendiciones.
Mi esposa	Dámaris gracias por estar siempre apoyandome en todo lo que emprendo
Mis padres	Porque sin ellos no hubiera sido posible alcanzar esta meta, siempre me han dado su amor y comprensión; y apoyado en mi educación.
Mis hermanos	Luisfer y Patty gracias y sigamos adelante!
Mis compañeros de universidad	Jorge, Heber, Edgar y Maco gracias por compartir tantos momentos en la U.
Mis catedráticos	Por haber contribuido a mi formación profesional.
La universidad	Gracias Universidad de San Carlos por haberme dado la oportunidad de convertirme un profesional. Pondré lo que este a mi alcance para poner tu nombre en alto.

DEDICATORIA A:

Jesucristo	Por haberme escogido y acompañarme en todo momento de mi vida.
Mi esposa, Dámaris	Mi amor logré alcanzar esta meta y te la dedico.
Mi padre, José Luis Ruiz	Misión cumplida papá. Gracias por estar en muchos proyectos que he emprendido.
Mi madre, Estelita	Gracias por apoyarme y aconsejarme en todas las etapas de mi vida. Dios te bendiga!
Mis hermanos, Luis Fernando y Silvia Patricia	A ustedes también les dedico este trabajo.
Mi tía, Flory	Por su apoyo a lo largo de toda mi vida.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

GLOSARIO

ActiveX	Pequeños programas diseñados para ejecutarse en aplicaciones Web. Son construidos con tecnología Microsoft.
Aplicación *Web*	Sistema de software construido para ser accedido por un navegador de páginas o *browser*.
Applet	Pequeño programa diseñado para ser ejecutado en otras aplicaciones.
Broker	En el contexto de aplicaciones distribuidas es un "corredor de componentes" ubicándolos en la red.
Capa de una aplicación	Parte independiente y especializada de una aplicación que provee cierta funcionalidad.
Caso de uso	Artefacto producido dentro de la metodología RUP para encontrar requerimientos y tener una visión general de una aplicación o proceso.
Código fuente	Conjunto de archivos escritos en un lenguaje de programación listos para ser compilados.
CORBA (*Component Object Request Broker Architecture*)	Modelo de componentes distribuidos definido por Object Group.

DCOM (*Distributed Component)*	Protocolo de comunicación para componentes distribuidos definido por Microsoft ®.
Escalabilidad	Capacidad de un sistema para aumentar el número de clientes sin afectar su desempeño y poder agregar servidores de una forma transparente.
Hardware	Componentes físicos de una computadora.
HTTP (*Hyper-Text Transport Protocol)*	Protocolo de transporte de hipertexto. Es el protocolo de comunicación generalmente utilizado desde un *browser* para acceder las páginas de una aplicación Web.
HTTPS (Hyper-Text Transpor Protocol Secured over Secure Socket Layer)	Protocolo HTTP sobre el protocolo de seguridad SSL (*Secure Socket Layer*, Capa de comunicacón segura) que encripta y desencripta las páginas Web que son solicitadas desde un navegador.
Intranet	Red interna o de área local donde pueden estar accesibles aplicaciones.
Java	Lenguaje de programación diseñado por la empresa *Sun Microsystems* en 1995 especialmente para aplicaciones Web.
Java RMI (*Java Remote Method Invocation*)	Protocolo de comunicación para componentes distribuidos definido por Sun Microsystems.

Lenguaje de programación	Conjunto de caracteres, símbolos y reglas utilizado para escribir las instrucciones a una computadora.
Mainframe	Sistema de cómputo grande y muy poderoso usado para tareas computacionales intensas.
Navegador	Programa de aplicación utilizado para ubicar y visualizar páginas Web.
Página Web	Unidad de información accesible desde un navegador.
Plataforma	Conjunto de elementos que pueden ser de *hardware* o *software* que sirven de base para la construcción de aplicaciones.
Protocolo	Conjunto de reglas semánticas y sintácticas que determinan el comportamiento de unidades funcionales en la comunicación.
Proxy	Programa o aplicación que sirve como intermediario para conectar a un cliente y un servidor.
RUP (Rational Unified Process)	Proceso unificado de Rational. Describe una metodología para el desarrollo de proyectos de *software*.
Script	Serie de instrucciones que son ejecutadas en orden y en tiempo de ejecución.

Servidor *Web*	Computadora que procesa solicitudes de páginas Web.
Sistema operativo	*Software* encargado de administrar los recursos de una computadora a las otras aplicaciones.
SOAP (*Simple Object Access Protocol*)	Protocolo de comunicación entre objetos distribuidos, el cual define una estructura de mensajería por XML
Software	Programa o conjunto de programas con que se alimenta una computadora para que funcione.
UNIX	Sistema operativo diseñado por Laboratorios Bell, escrito en el año de 1969 y fue importante para el desarrollo de aplicaciones Web.
URL (*Uniform Resource Locator*)	Dirección global de documentos y otras fuentes en Internet.
Web Service	Aplicación disponible generalmente a través de los protocolos HTTP y SOAP.
XML (*Extensible Markup Language*)	Lenguaje de marcado extensible utilizado para definir estructuras para mensajes, configuraciones, describir información, etc.

RESUMEN

Este trabajo de graduación presenta una serie de consideraciones de los aspectos más relevantes de las aplicaciones Web distribuidas en N capas haciendo uso de los patrones de diseño. Se empieza con los conceptos básicos de las aplicaciones Web y las capas de una aplicación, luego se expone la administración de proyectos de aplicaciones Web para luego concluir con los conceptos de patrones y algunos ejemplos.

El primer capítulo, describe la evolución que ha pasado para llegar a las aplicaciones Web distribuidas y la arquitectura orientada a servicios. Se hace énfasis en la evolución que han tenido las arquitecturas y modelos de programación hasta llegar a los más modernos. Se analizan las aplicaciones desde el punto de vista de las capas que la forman y se termina analizando la convergencia de las aplicaciones distribuidas y las aplicaciones Web.

El segundo capítulo, describe un marco de trabajo básico para la administración de proyectos enfocados a aplicaciones Web. Esto es importante conocer porque un determinado patrón puede ir orientado a roles dentro de un equipo de desarrollo. También se enumeran cada uno de los artefactos que los integrantes del equipo deben hacer.

El tercer capítulo introduce el concepto de patrones de diseño, como se pueden agrupar y tener marcos de referencia para facilitar su utilización. Se da una guía de cómo deben estar formados y como utilizarlos. Además se hace énfasis en la utilización de patrones para aplicaciones Web y como asociarlos a los roles de un equipo de trabajo.

El cuarto capítulo, tiene algunos ejemplos de patrones de la arquitectura de las aplicaciones Web. Los patrones de arquitectura pretenden dar un concepto general de cómo están formadas las aplicaciones. Se describe el contexto, el problema, la solución, las consecuencias y los roles a los cuales les podría ser útil el patrón.

El último capítulo, tiene algunos ejemplos de patrones de diseño enfocados a las aplicaciones Web. Se describen los que se consideran más importantes para empezar a desarrollar una aplicación. Cada patrón tiene descrito el contexto para aplicarlo, el problema que resuelve, las consecuencias de aplicarlo y enumera cada uno de los roles a los cuales va enfocado.

OBJETIVOS

General

Desarrollar un trabajo que ayude a comprender de la mejor forma las aplicaciones Web haciendo uso del concepto de patrón de diseño.

Específicos

1. Mostrar las características de una aplicación Web y cómo llego a tenerlas luego de la constante evolución de las aplicaciones.

2. Describir a las aplicaciones Web distribuidas en N capas utilizando patrones para su utilización en aplicaciones y facilitar su comprensión.

3. Exponer los conceptos de patrones de diseño para utilizarlos como herramientas que apoyen el diseño de aplicaciones Web.

4. Identificar a cada uno de los roles que participan en el desarrollo de aplicaciones Web y cómo pueden relacionarse a los patrones.

5. Exponer ejemplos de patrones de arquitectura y de diseño para utilizarlos como base para entender mejor los conceptos y aplicarlos.

INTRODUCCIÓN

Las aplicaciones Web son la tendencia actual para el desarrollo de aplicaciones y existen distribuidas en N capas. Algunos podrán decir ¿por qué N capas? y no solamente tres: presentación, lógica y datos. La respuesta es que en las aplicaciones distribuidas cada una de las capas mencionadas anteriormente puede subdividirse en otras.

La utilización de patrones de diseño ayuda a identificar de mejor forma las capas que forman la aplicación y entender varios conceptos. Tienen como propósito exponer un diseño ya probado muchas veces, y están documentados de tal forma que sea fácil su comprensión utilizando el UML como lenguaje de modelado.

Es importante identificar las capas a nivel de arquitectura y a nivel de la aplicación y como se relacionan entre sí. Los patrones de arquitectura y diseño respectivamente documentan estos conceptos dentro de un contexto. Es siempre importante aplicar el razonamiento y la experiencia para las decisiones de diseño ya que cada problema puede tener una solución particular.

Para entender bien los conceptos de las aplicaciones Web es necesario conocer la evolución de la arquitectura de las aplicaciones y los modelos de programación. También es importante conocer su ciclo de vida para saber administrarlas e identificar los integrantes del equipo para el desarrollo de una aplicación Web. Estos conceptos luego se relacionan con el uso de patrones de diseño.

Este trabajo presenta como pueden utilizarse los patrones como un apoyo al diseño de aplicaciones Web describiendo los conceptos básicos de aplicaciones Web, administración de proyectos, definición de patrones y finalizando con algunos ejemplos de patrones de arquitectura y diseño. Estos ejemplos también pueden servir para entender como están compuestas las aplicaciones Web distribuidas en N capas.

1. EVOLUCIÓN DE LA ARQUITECTURA DE LAS APLICACIONES DE *SOFTWARE*

1.1. Evolución en el desarrollo de las aplicaciones de *software*

La tendencia en el desarrollo de aplicaciones ha evolucionado desde el desarrollo en sistemas centralizados de *mainframes* hasta las aplicaciones Web distribuidas en N capas. La evolución se ha dado por el desarrollo de nuevas plataformas de *software* y *hardware* para poder satisfacer de mejor forma las necesidades de las empresas.

Las plataformas de *software* están constituidas por sistemas operativos, protocolos de comunicación, bases de datos, servidores de aplicaciones especializadas, etc. y las de *hardware* incluyen el desarrollo de nuevas tecnologías para computadoras (*mainframes*, servidores, computadoras personales), tecnologías de comunicación, tecnologías de almacenamiento de información, etc.

Todo este nuevo *software* y *hardware* ha contribuido para el desarrollo de aplicaciones Web que reúnen un conjunto de elementos que han creado nuevos paradigmas en el diseño de la arquitectura e implementación de aplicaciones de *software*. Por lo tanto, es importante conocer y clasificar cada una de las partes que forman estas aplicaciones según sus características para poder construirlas de la mejor forma.

La principal característica de una aplicación Web es que se accede por un navegador para Internet (también llamado *browser*) solamente teniendo la

dirección URL (*Uniform Resource Locator*, Localizador Uniforme de Recursos) de la aplicación. Generalmente estará compuesta por páginas con hiper-texto donde el usuario podrá "navegar" por medio de enlaces a varias páginas, y hacer solicitudes al servidor Web creando un nuevo concepto de cliente / servidor.

El concepto de capas de una aplicación es una forma de dividir las partes de una aplicación: la parte con la que interactúa el usuario (capa de presentación), el proceso de la información (capa de lógica de la aplicación) y el almacenamiento de dicha información (capa de datos). Cada capa puede considerarse como independiente y bien especializada en realizar cierto trabajo asignado.

Por ejemplo, si se tiene una aplicación ya construida la cual es necesario tenerla disponible como una aplicación Web deben construirse las páginas Web (capa de presentación) y que estas con cierta lógica (capa de lógica de la aplicación) puedan tener acceso a la información almacenada en la aplicación antigua (capa de datos). En este caso la información almacenada de la aplicación antigua se convierte en la capa de datos de la nueva aplicación Web a construir.

En este capítulo se estudiarán las arquitecturas que han surgido a lo largo de la historia del desarrollo de software, enfocándose principalmente a mostrar las ventajas del desarrollo de aplicaciones distribuidas y comprender los nuevos conceptos de las aplicaciones Web distribuidas en N capas.

1.1.1. Historia de arquitecturas tecnológicas

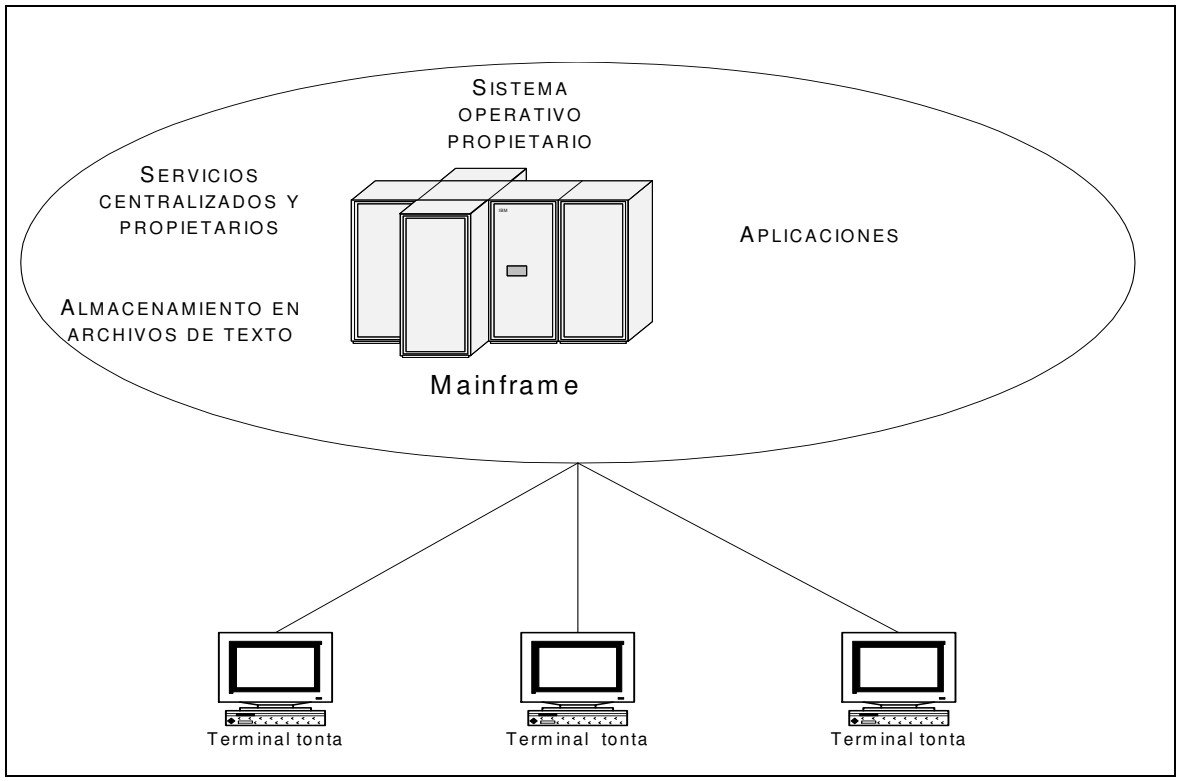
A lo largo de la historia del desarrollo de aplicaciones han aparecido varias arquitecturas, siendo las más recientes las que resuelven problemas que han tenido las anteriores. Debe tenerse en cuenta que el aparecimiento de una nueva arquitectura no significa que deba suplantar obligatoriamente a otra más antigua. Cada arquitectura tiene sus ventajas y desventajas, y en algunos casos es muy costoso trasladarse de una arquitectura a otra.

1.1.1.1 Arquitectura centralizada de *mainframe*

Esta arquitectura requiere un computador central con una alta capacidad de procesamiento llamado *mainframe* al cual se conectan terminales. Estas terminales se les llama “terminales tontas” ya que no tienen capacidad de procesamiento sino simplemente son pantallas de tipo texto donde puede interactuarse con las aplicaciones del *mainframe.*

Existen varias empresas que manejan esta arquitectura debido a que es muy difícil y costoso migrar las aplicaciones a nuevas arquitecturas y, además, son bastante seguras y de muy buen rendimiento. Actualmente, se están implementando plataformas para aplicaciones distribuidas sobre esta arquitectura y así aprovechar las ventajas que tienen. La arquitectura puede observarse en esquema de la figura 1.

Figura 1. Arquitectura centralizada de *mainframe*



a. Características principales

- Sistemas propietarios.
- Sistemas con una alta capacidad de procesamiento y manejo de recursos.
- Lenguajes de programación de 3ra. generación.

b. Ventajas

- Administración centralizada.
- Tienen computadores con gran capacidad para poder administrar gran cantidad de recursos (*mainframes)*.
- Alto rendimiento.

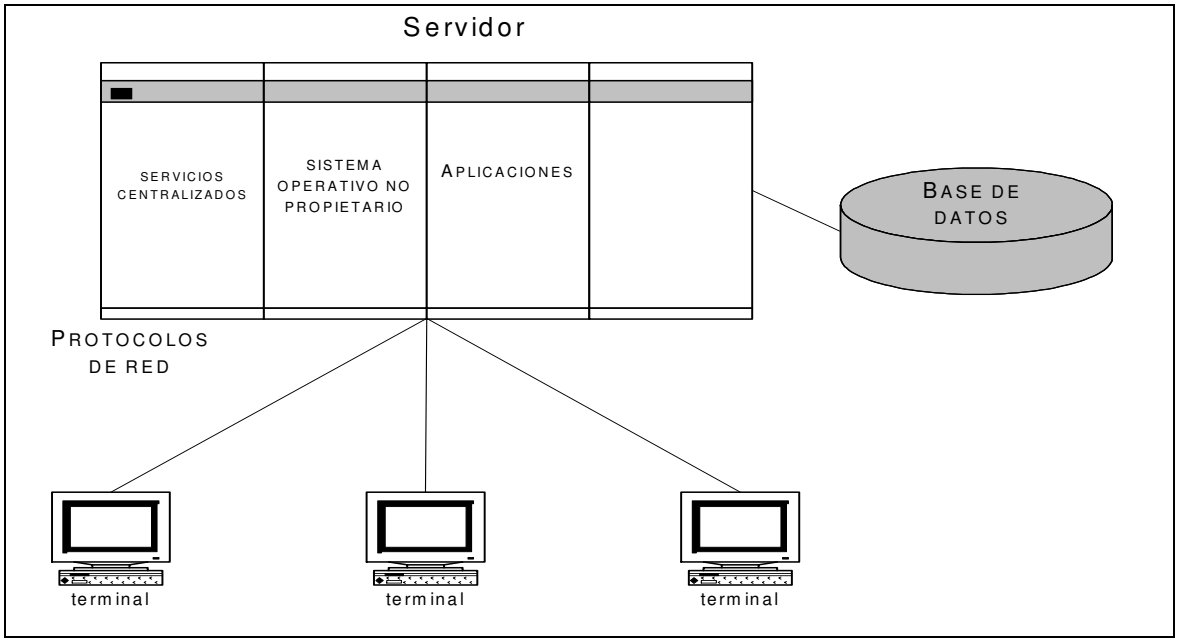
c. Desventajas

- Requieren personal altamente capacitado.
- Son sistemas cerrados.
- Utilizan técnicas de programación antiguas.
- Ambiente no gráfico.
- Costo elevado en compra, soporte, mantenimiento y crecimiento.

1.1.1.2 Arquitectura centralizada con sistemas abiertos

Los sistemas abiertos surgieron después del sistema centralizado. Esta arquitectura todavía es centralizada pero ya no es propietaria. Esto significa que la plataforma de *software* y *hardware* está basada en estándares abiertos y no es específica para un fabricante. Surgieron sistemas operativos como UNIX manejando el concepto de servidor y *terminal*. La figura 2 muestra el esquema de la arquitectura centralizada con sistemas abiertos.

Figura 2. Arquitectura centralizada con sistemas abiertos



a. Características

- No son sistemas propietarios.
- Surgen las bases de datos relacionales.
- Dominan aún las bases de datos jerárquicas.
- Lenguajes de programación de 3ra. y 4ta. generación.

b. Ventajas

- Se empieza a manejar el concepto de bases de datos relacionales.
- La plataforma de *software* no es propietaria, esta incluye: sistemas operativos, manejadores de bases de datos, protocolos de comunicación, etc.
- La plataforma de *hardware* están disponibles para diversas plataformas de *software*: procesadores, dispositivos de almacenamiento, etc. .
- Soporte de múltiples lenguajes de programación.
- Interconexión entre sistemas operativos a través de protocolos de red estándares.

c. Desventajas

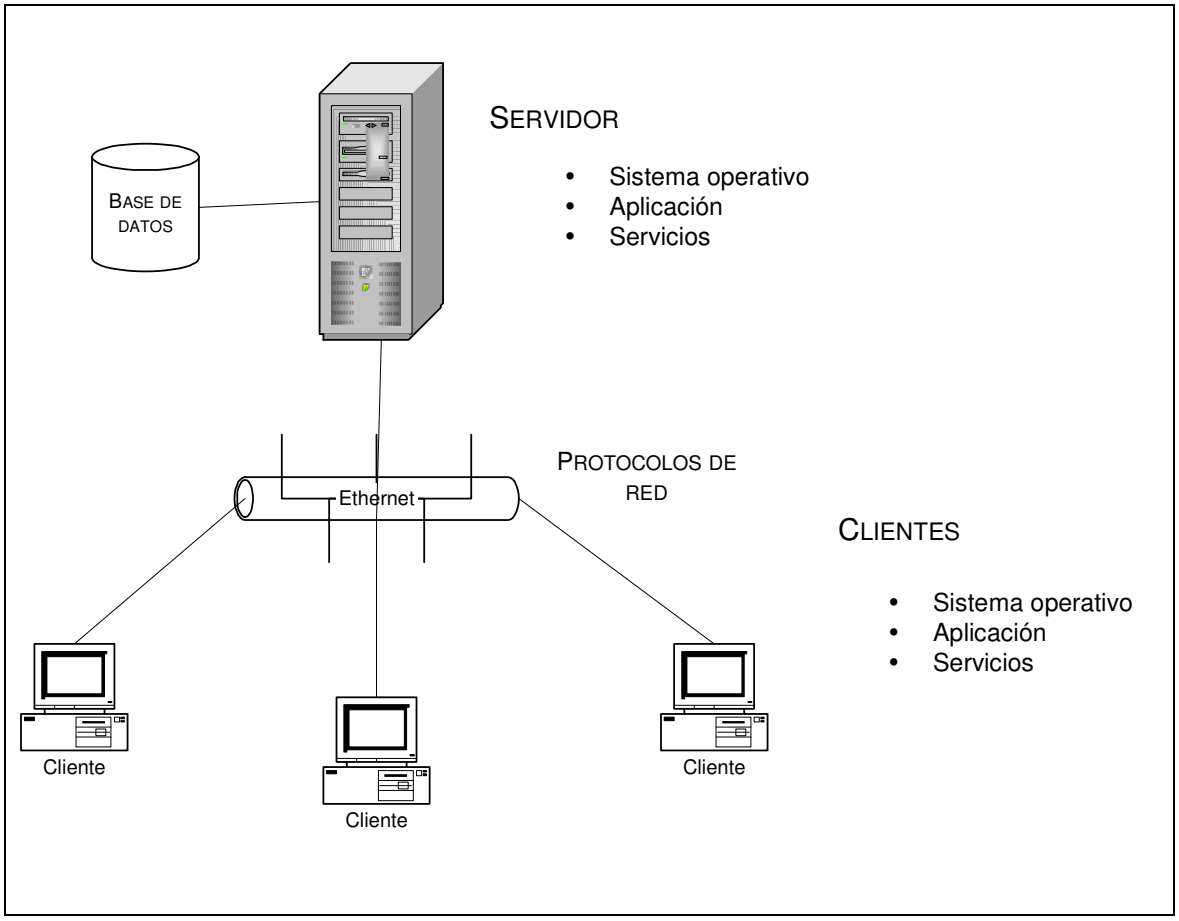
- El ambiente no es gráfico.
- Se requieren servidores de alta capacidad y rendimiento.
- Lenguajes de programación de 3ra. generación.

1.1.1.3 Arquitectura cliente / servidor

La arquitectura cliente / servidor surge con el invento de la computadora personal o PC (*Personal Computer,* computadora personal). Los sistemas operativos para las PC tienen características que hacen que tenga

independencia de procesamiento y maneje un entorno gráfico para el usuario. La figura 3 muestra como está constituida la arquitectura de cliente / servidor.

Figura 3. Arquitectura cliente / servidor



a. Características

- Procesamiento en el servidor y en cliente por el particionamiento de la aplicación.
- Base de datos relacionales.
- Conexiones de red a través de protocolos estándar.
- Herramientas de lenguajes programación de 4ta. generación.

b. Ventajas

- Lenguajes de programación gráficos orientados a objetos y eventos.
- Desarrollo de aplicaciones más rápido.
- Ambiente gráfico.

c. Desventajas

- Instalación en todas las computadoras clientes.
- Número limitado de conexiones de computadoras clientes.
- Necesidad de mayor cantidad de recursos en computadoras cliente.

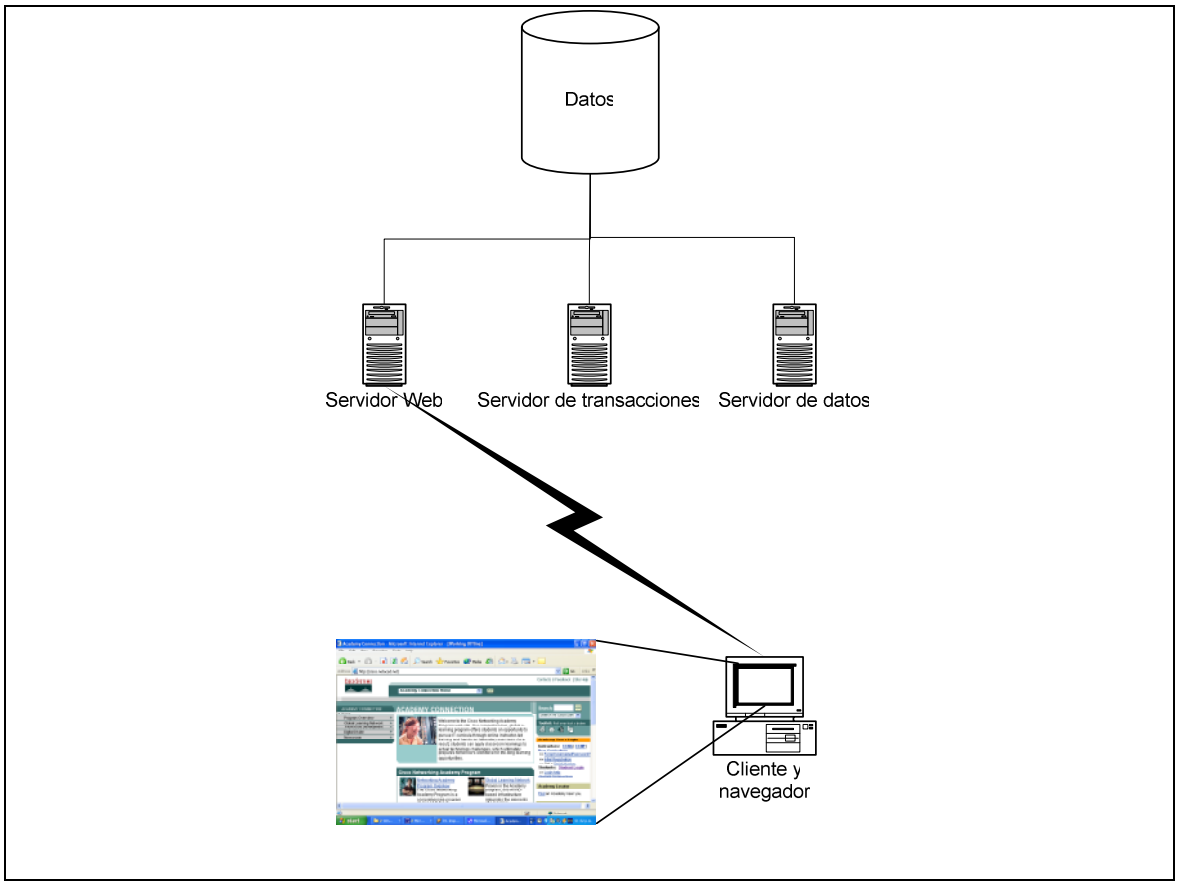
1.1.1.4 Arquitectura cliente/ servidor de N capas

Las aplicaciones distribuidas son un conjunto de elementos que interactúan entre sí en varios niveles o capas. La figura 4 muestra un diagrama general de la arquitectura de las aplicaciones Web distribuidas en N capas.

Como puede observarse pueden existir varios servidores. Si tomamos en cuenta que un servidor presta servicios y que un cliente es quien consume estos servicios, entonces las aplicaciones distribuidas son un cliente / servidor múltiple porque existen varios servidores y varios clientes.

Las aplicaciones distribuidas están compuestas por servidores especializados: bases de datos, monitores transaccionales, servidores Web, etc. Lo interesante de esta arquitectura es como cada uno de estos servidores establecen protocolos de comunicación para que puedan funcionar en conjunto.

Figura 4. Arquitectura cliente / servidor de 3 capas



a. Ventajas de las aplicaciones distribuidas

- Escalabilidad de las aplicaciones: permiten distribuir la carga por medio de varios servidores, lo que además permite hacer un balanceo de carga.

- Simplifican el acceso a sistemas heterogéneos: existen servicios especializados que hacen posible la comunicación e integración de aplicaciones.

- Desarrollo de aplicaciones Web: están orientadas al desarrollo de aplicaciones sobre Internet.

- Soporte de varios lenguajes de programación: en la arquitectura de las aplicaciones distribuidas cada componente que la forma puede ser creado independientemente de los otros.

- Facilidad de administración de la aplicación: los servicios están centralizados facilitando el desarrollo y mantenimiento de la aplicación.

- Administración eficiente de recursos: están orientadas a administrar los recursos de forma que las aplicaciones puedan ser escalables.

- Implementación de seguridad: como la aplicación se comunica por medio de la red la seguridad es un aspecto muy importante y puede ser configurada según las necesidades de la aplicación.

b. Desventajas de las aplicaciones distribuidas

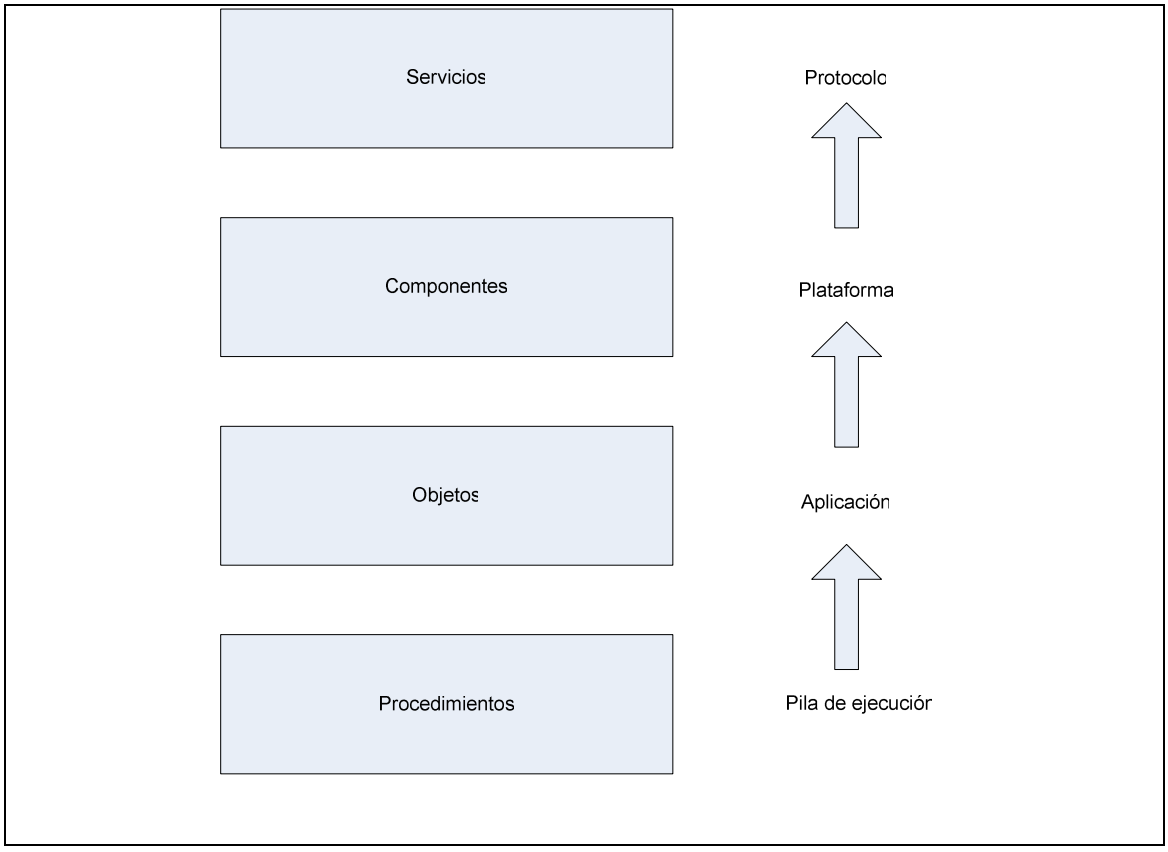
- Conocimiento en diversas tecnologías: requiere que el equipo de desarrollo de la aplicación tenga conocimientos en varias ramas. Por ejemplo, la integración de una arquitectura *mainframe* y una aplicación Web requiere de un experto en el desarrollo de aplicaciones *mainframe*, un experto en la conexión para la integración entre la aplicación Web y el sistema *mainframe* y, además, un experto en herramientas de desarrollo de aplicaciones Web.

- Evaluación de varias plataformas: se necesitan realizar evaluaciones en cada caso para determinar cual es la mejor plataforma para aplicaciones distribuidas entrando en un proceso de investigación para conocer a profundidad los detalles y escoger la mejor opción según las necesidades que se tengan.

1.1.2. Modelos de programación

Los modelos de programación han evolucionado desde la programación secuencial hasta la arquitectura orientada a servicios. En cada etapa de la evolución se ha agregado más funcionalidad a las aplicaciones para que puedan comunicarse unas con otras. Inicia desde los procedimientos que pueden comunicarse entre sí por medio de la pila de ejecución, los objetos que pueden comunicarse dentro de la misma aplicación, los componentes dentro de la misma plataforma y los servicios que comunican aplicaciones por medio de un protocolo establecido en un contrato. En la figura 5 se describe gráficamente esta evolución. Luego se presentan cada uno de los modelos de programación.

Figura 5. Evolución de los modelos de programación



1.1.2.1 Programación secuencial

Este modelo de programación consiste en ejecutar una serie de instrucciones una tras otra en un solo bloque y sin una estructura definida. El concepto de subrutinas no se encuentra implementado lo que imposibilita la utilización de variables locales. Tiene problemas cuando las aplicaciones aumentan de tamaño y complejidad porque es necesario controlar el flujo de la ejecución del programa a través de bifurcaciones repitiendo bloques de código que provoca sea difícil de entender. Ejemplos de lenguajes que utilizan este tipo de programación son: Fortran, Cobol y primeras versiones de *Basic*.

1.1.2.2 Programación estructurada

Facilita el diseño de la aplicación por medio de la utilización de subrutinas y manejo de variables locales por lo que la aplicación puede dividirse en bloques llamados procedimientos o funciones siendo éstos la principal base de la construcción de aplicaciones. Está caracterizada especialmente por utilizar:

- Bloques anidados
- Procedimientos
- Recursión

Si la aplicación es realizada por un grupo de programadores no deben preocuparse por conflictos en los nombres para las variables locales dentro de un bloque. Se introduce el concepto de reutilización de código a través de los procedimientos que pueden ser llamados desde otras partes del programa. Existe un cambio en las estructuras de datos dando lugar a tipos de datos definidos por el programador los cuales son los llamados tipos de datos abstractos que serían la base de la programación orientada a objetos.

La programación estructurada resuelve muchos problemas que tiene la programación secuencial, pero deja sin resolver varios como el nombramiento de variables globales y el acoplamiento de módulos (conjunto de procedimientos) realizados por diferentes personas dentro de un equipo. Esta programación surgió entre mediados y finales de los años sesenta y aún sigue utilizándose. Ejemplos de lenguajes de programación de este modelo son: C y Pascal.

1.1.2.3 Programación orientada a objetos

Se basa en realizar un análisis del problema desde el punto de vista del mundo real que esta formado por varios objetos que tienen relaciones entre sí. La solución está basada en la abstracción de un conjunto de objetos y sus relaciones dentro de un contexto para luego ser implantados en un lenguaje de programación.

En la programación orientada a objetos un objeto se define como un conjunto de datos que determinan su estado y un conjunto de operaciones que pueden cambiarlo. El término de "orientado a objetos" fue utilizado originalmente para distinguir a aquellos lenguajes "basados en objetos". Un lenguaje basado en objetos soporta:

- Encapsulación de información
- Abstracción de datos
- Paso de mensajes

Un lenguaje orientado a objetos además de los anteriores, también implementa el concepto de herencia que es la capacidad de poder crear otros objetos a partir de otro.

A través de la programación orientada a objetos se logra reutilizar mejor el código fuente. La comunicación entre estos objetos se logra mediante envío de mensaje a través interfaces que publican la funcionalidad de un objeto.

Aunque la programación orientada a objetos resolvió muchos de los problemas que se encontraban en la programación estructurada, todavía tiene algunas deficiencias como que los objetos son solamente reutilizables a nivel de código fuente porque no existe definido un estándar de comunicación con objetos construidos en otras plataformas o lenguajes de programación.

La compatibilidad a nivel de código fuente enfrenta también el problema de versiones porque cuando un objeto es actualizado las aplicaciones que lo utilicen deberán volver a compilarse y a distribuirse en los clientes. Ejemplos de lenguajes de programación de este modelo son: Ada, Delphi, Object Pascal, C++, lenguajes “punto net” de Microsoft y Java.

1.1.2.4 Programación orientada a eventos

Este tipo de programación vino a complementar a la programación orientada a objetos porque los eventos son operaciones especiales de un objeto. Un evento es algo que ocurre que cambia el valor de los datos produciendo una acción y, con esto, el objeto actúa más parecido a la vida real. El evento debe estar definido en el objeto para que lo reconozca.

Las operaciones especiales están constantemente esperando un cambio en el estado del objeto (atributos) para que en caso de que ocurra un evento se ejecute una acción. Los datos especiales del objeto sirven para detectar un estado, ya que estos cambian cuando ocurre algún evento.

El flujo de ejecución del programa está definido por los eventos definidos para los objetos de la aplicación por lo que no lleva un orden específico. Esto permite mayor interacción del usuario con la aplicación. Ejemplos de lenguajes de programación de este modelo son: Java, Delphi, Microsoft Visual Basic, y lenguajes "punto net" de Microsoft.

1.1.2.5 Programación con lenguajes de marcado

Este tipo de lenguajes es utilizado para desarrollar aplicaciones Web. El HTML (*Hypertext Markup* Language, Lenguaje de Marcado de Hipertexto) es el que se utiliza en la mayoría de aplicaciones Web para la creación de las páginas Web. También se utiliza el XML (*Extensible Markup Language*, Lenguaje de marcado extensible) para múltiples propósitos de estructurar información.

Los lenguajes de marcado consisten en un conjunto de etiquetas en donde cada etiqueta puede concebirse como un objeto porque tiene sus atributos y eventos asociados a ellas. El HTML y XML se derivan del SGML (*Standard Generalized Markup Language*, Lenguaje Estandarizado y Generalizado de Marcado) que se definió como estándar internacional para el marcado de documentos.

Una página HTML es un archivo de texto común pero la apariencia es controlada por las etiquetas que se encuentran dentro del texto y son interpretadas por un navegador el cual las presenta en forma gráfica. Por ejemplo, un texto puede ser marcado por etiquetas para indicar que es un vínculo a otra página (hipervínculo), que el texto va subrayado, que es un botón, que tiene un color, etc.

Las etiquetas de HTML deben encerrar al texto por lo que deben colocarse en pares para marcar el texto. La primera aplica las características de la etiqueta y la segunda indica el fin. Las páginas HTML tienen una estructura definida que incluye un encabezado y un cuerpo.

El XML es una nueva especificación que permite a los diseñadores de páginas Web crear etiquetas personalizadas y obtener mucha funcionalidad que no proporciona el HTML. Es utilizado para describir meta-contenidos respecto a documentos o recursos, publicación e intercambio de información con bases de datos y en el intercambio de mensajes entre organizaciones o aplicaciones.

1.1.2.6 Componentes distribuidos

Los objetos pueden evolucionar a una forma de estar distribuidos en la red por medio de una plataforma, llamándose "componentes". Se les llama así porque formarán parte de una aplicación pero a la vez son independientes. Para que puedan comunicarse a través de la red son necesarios los protocolos de comunicación definidos por una plataforma.

Existen varios protocolos para componentes, los mas conocidos son: DCOM (*Distributed Components*, Componentes Distribuidos), CORBA (*Component Object Request Broker Architecture*, Arquitectura de Despachador de Componentes) y Java RMI (*Java Remote Method Invocation*, Método de Invocación remota de Java) que permiten la independencia de lenguaje de programación y comunicación a través de una red.

Estos protocolos para componentes presentaron ciertos problemas de compatiblidad entre componentes de distintas plataformas. Luego se definió SOAP (*Simple Object Access Protocol*, Protocolo de Acceso Simple a Objetos)

que utiliza XML para definir el formato de los mensajes y HTTP (*Hyper-Text Transport Protocol*, Protocolo de transporte de hipertexto) como protocolo de comunicación.

Actualmente, a los componentes distribuidos que utilizan SOAP y http como protocolos de comunicación se les llama Web Services. Estos definen una interfaz la cual puede ser consultada para saber que funcionalidad tiene el componente llamada WSDL (*Web Service Description Language*, Lenguaje de Descripción del *Web Service*) y permite la comunicación entre aplicaciones construidas sobre plataformas diferentes.

1.1.2.7 Programación orientada a servicios

Basada en la arquitectura orientada a servicios también conocida como SOA (*Services Oriented Architecture*, Arquitectura Orientada a Servicios). No reemplaza a la programación orientada a objetos y a los componentes distribuidos, sino los complementa introduciendo mejoras. SOA no ve a la aplicación desde la perspectiva de objetos relacionados entre sí, sino de que está formada por un conjunto de servicios autónomos.

Un servicio es una aplicación que puede interactuar con otras por medio de mensajes sobre un protocolo definido. Debe ser autónomo, con límites bien definidos, y tiene que dar a conocer su contrato y esquema de comunicación. Cuando las aplicaciones se conciben como servicios reflejan mejor los procesos y relaciones del mundo real. El contrato define las reglas de comunicación, rendimiento del servicio, seguridad y otros aspectos que describen el servicio.

Pueden implantarse como *Web Services* o componentes distribuidos pero son vistos desde el mundo exterior como una aplicación independiente que provee un servicio, y para accederlo debe establecerse un contrato definido por el mismo servicio.

1.1.3. Aplicaciones divididas en capas

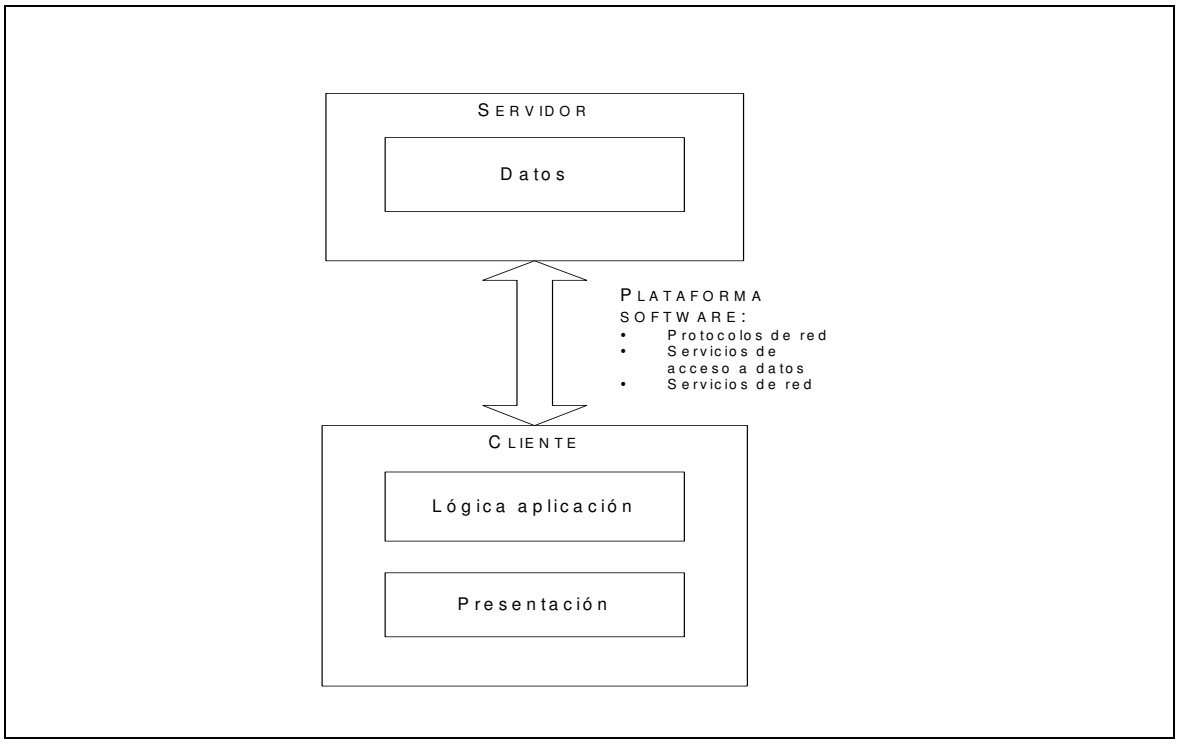
Las aplicaciones pueden dividirse en varias partes y pueden estar distribuidas en la red en computadoras separadas. Cada capa es independiente una de la otra y da cierta funcionalidad a la aplicación. El modelo típico es el de 3 capas que divide la aplicación en: presentación, lógica y datos.

1. **Capa de presentación:** está constituido por la interfaz que se presenta al usuario para que interactúe con la aplicación.
2. **Capa de lógica de la aplicación:** administra la lógica de la aplicación. Se realizan validaciones y se interactúa con el repositorio de datos.
3. **Capa de datos:** es donde se almacena la información que administra la aplicación.

1.1.3.1 Análisis de la arquitectura cliente / servidor de 2 capas

Comúnmente es llamada solamente arquitectura cliente / servidor. Esta arquitectura tiene su aparición con el surgimiento de las computadoras personales y de las redes de área local (LAN, *Local Area Network*). En este tipo de arquitectura se agrupan los elementos de presentación y lógica de la aplicación en los clientes y el elemento de datos en un servidor formando entonces dos capas. La figura 6 muestra el esquema de esta arquitectura.

Figura 6. Cliente/ servidor de 2 capas



a. Ventajas de la arquitectura cliente / servidor de 2 capas

- La información está centralizada: en la base de datos que se encuentra en el servidor por lo que cualquier mantenimiento es en un solo lugar.

- Se comparten datos a través de la red: las redes facilitan la comunicación entre el cliente y el servidor para acceder la base de datos.

- Garantiza consistencia en el acceso a los datos: como los datos están centralizados es más fácil administrar la consistencia en su acceso.

- Se simplifica la generación de reportes: los reportes y consultas son más simples de realizar debido a la base de datos centralizada.

b. Desventajas de la arquitectura cliente / servidor

- Compra de licencias para aumentar número de conexiones: generalmente se compran licencias por el número de conexiones que se tendrán a una base de datos.

- Dificultad de escalabilidad: la escalabilidad es limitada por el número de conexiones que permitan los recursos del servidor de la base de datos.

- Difícil mantenimiento de actualizaciones: ya sea por nuevas necesidades o para corregir errores la instalación de actualizaciones deben realizarse en cada uno de los clientes.

- Dificultad en reutilización de código: la reutilización de *software* es dificultosa debido a que los módulos están configurados para acceder una base de datos.

- Aplicación ligada a una base de datos: el cliente debe poseer un servicio de acceso a datos que es configurado para conectarse a una base de datos específica, por lo que si se cambia a otra base de datos debe configurarse en cada cliente.

- Rendimiento deficiente en la red: se vuelve deficiente por la cantidad de datos que es transferida a través de ésta porque se tiene una conexión permanente.

1.1.3.2 Cliente / Servidor de 2 capas con procedimientos almacenados

El mayor beneficio de la arquitectura cliente / servidor de dos capas es la información centralizada en una base de datos. Aprovechando esa ventaja puede trasladarse la lógica de la aplicación a la base de datos con los procedimientos almacenados, pero surge otro problema: la lógica de la aplicación es totalmente dependiente de la base de datos. El esquema de esta modalidad de cliente / servidor se muestra en la figura 7.

a. Ventajas

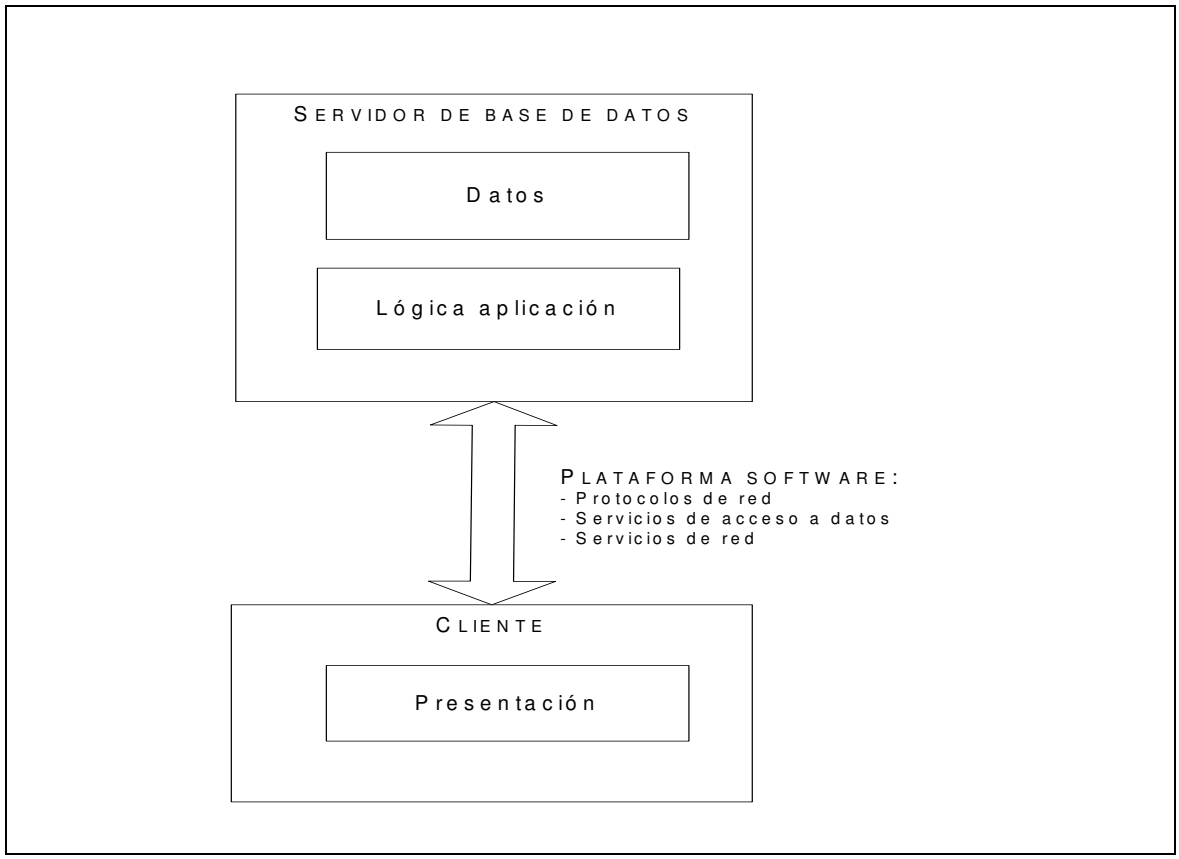
- Buen rendimiento de la aplicación: evita el tráfico en la red debido a que el procedimiento almacenado devuelve ya la información procesada desde la base de datos. Por ejemplo, si se necesitaba el cálculo de un valor promedio debía de enviarse toda la información al cliente para que realizara el cálculo, en cambio el procedimiento almacenado puede calcular el valor sin necesidad de enviar todos los datos al cliente.

- Administración de seguridad: pueden configurarse los permisos a los procedimientos almacenados para restringir el acceso a usuarios.

- Permite escalabilidad: realizando procedimientos almacenados especializados que permitan procesar los datos de la manera más eficiente.

- Facilita el mantenimiento de actualizaciones: el mantenimiento de cambios en la lógica puede realizarse en el código del procedimiento almacenado y ser transparente para los clientes permitiendo encapsular funcionalidad.

Figura 7. Cliente / servidor de 2 capas con procedimientos almacenados



b. Desventajas

- El lenguaje de procedimientos almacenados es limitado: no es rico en instrucciones como un lenguaje de programación.

- Conexiones limitadas a la base de datos: existe mayor problema cuando procedimientos almacenados tienen períodos largos de ejecución.

- Dependencia a una base de datos: los procedimientos almacenados están directamente ligados a una base de datos en particular.

1.2. Comparación entre la arquitectura cliente / servidor y la arquitectura cliente / servidor de N capas

La principal diferencia está en como la aplicación se encuentra distribuida entre el cliente y el servidor. Los clientes grandes son la modalidad más común de cliente / servidor donde la mayor parte de la aplicación se ejecuta en lado del cliente. Los servidores grandes son más fáciles de administrar y distribuir en la red porque la mayor parte del código de la aplicación se ejecuta en el servidor y el cliente únicamente aporta el elemento de presentación.

En cada caso, servidores grandes o clientes grandes, tienen sus propios usos, y lo que pretende la arquitectura de cliente / servidor de N capas es buscar un equilibrio que aproveche todas las ventajas de la arquitectura cliente / servidor, especialmente en las aplicaciones Web. Puede observarse en la Tabla I como estará distribuida la carga.

Tabla I. Tamaño de clientes y servidores según el número de capas

	2 capas	*N capas*
Servidor	Grandes o medianos	Grandes
Cliente	Grandes	Livianos

Que tan grande sea un cliente o un servidor está definido por la división de los elementos de la aplicación: presentación, lógica y datos. En la arquitectura cliente / servidor de 2 capas la lógica de la aplicación se encuentra junto al elemento de presentación o al elemento de datos en los procedimientos almacenados. En la arquitectura cliente / servidor de N capas la lógica de la aplicación ocupa una capa intermedia donde se encuentra la mayoría de servicios. Esto significa que la lógica de la aplicación fue trasladada del cliente a múltiples servidores en una capa intermedia.

La tabla II muestra una comparación entre el cliente / de N capas y un cliente / servidor de 2 capas.

Tabla II. Comparación cliente/servidor de 2 capas con cliente/servidor de N capas

Característica de aplicación	**Cliente / servidor de 2 capas**	**Cliente / servidor Web de N capas**
Número de clientes por aplicación	menos de 100	Millones
Número de servidores por aplicación	1 o 2	De uno en adelante, sin límite.
Geografía	Basada en campus	Global
Interacciones servidor a servidor	No	Sí
Plataformas de software	Servidor de base de datos y procedimientos almacenados.	Servidor web, servidor de componentes distribuidos, Servidor de colas. etc.
Arquitectura cliente / servidor	2 capas	N capas
Contenido de multimedia	Bajo	Alto
Administración de servidor	Compleja (más lógica manejada en el cliente)	Menos compleja (la aplicación es administrada centralmente)
Seguridad	Baja (es configurada a nivel de datos)	Alta (puede ser configurada a nivel de componente)

1.3. Internet y las aplicaciones distribuidas

La arquitectura de las aplicaciones distribuidas es parecida en varios aspectos a la arquitectura centralizada de *mainframe*. Los navegadores para Internet son como las terminales de los *mainframes*, accediendo una aplicación que se encuentra centralizada en un solo lugar. Pero las aplicaciones distribuidas han acumulado varias ventajas de cada uno de las arquitecturas predecesoras.

Las aplicaciones Web son centralizadas como las aplicaciones de *mainframe y* la arquitectura de servicios que la componen son estándares abiertos como en los sistemas abiertos centralizados. Además, existen múltiples capas siendo actuando como servidores y otras como clientes, como en la arquitectura cliente / servidor. Se puede decir entonces que las aplicaciones distribuidas tienen características similares a arquitecturas anteriores pero con mejoras sustanciales.

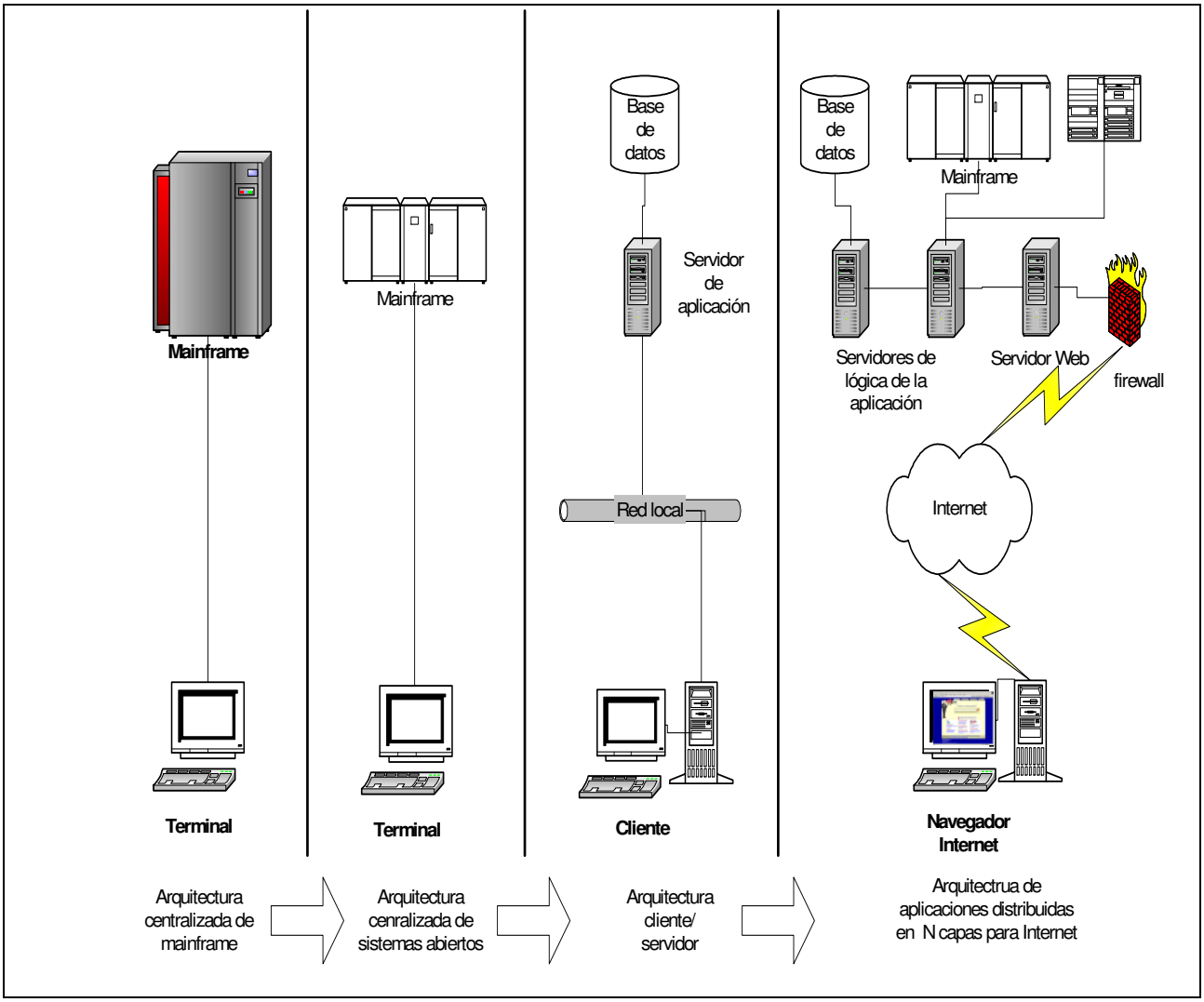
Las aplicaciones Web se iniciaron con los lenguajes de marcado para la construcción de páginas Web. Al inicio solo se tenían hipervínculos entre páginas y no existía ningún procesamiento de lógica. Cuando empezaron a utilizarse con el manejo de lógica de la aplicación todo el procesamiento se realizaba en las páginas Web y cuando existían muchos usuarios conectados al mismo tiempo el rendimiento no era bueno.

Para lograr hacer que las aplicaciones Web fueran robustas y que estuvieran preparadas para una gran cantidad de usuarios haciendo solicitudes constantemente, se empezaron a crear los ambientes donde podrían ser ejecutadas a los cuales se les ha llamado plataformas de aplicaciones distribuidas, y son estas las que permiten un entorno en el cual pueden construirse aplicaciones empresariales. La figura 8 muestra como han evolucionados las aplicaciones hasta llegar a las aplicaciones Web.

El futuro de las aplicaciones web está orientado al concepto de Web Semántica, la cual consiste en que las computadoras podrán “entender” el contenido de las páginas Web y no solamente procesarlas. Esto podrá darse definiendo un lenguaje común que pueda describir el contenido de las aplicaciones y comunicarlo el cual se está construyendo por medio de XML y RDF (*Resource Description Framework*, Marco de trabajo de descripción de recursos).

En los siguientes capítulos se describe principalmente los patrones para el diseño de una aplicación Web y una introducción de como administrar proyectos de este tipo. Un patrón es un modelo a seguir obtenido de la experiencia y diseños probados bajo ciertas condiciones que ayudarán a hacer mejores diseños, no empezar desde cero y entender mejor los conceptos de las aplicaciones Web distribuidas en N capas.

Figura 8. Evolución hacia las aplicaciones Web distribuidas en N capas



2. ADMINISTRACIÓN DE PROYECTOS DE APLICACIONES WEB DISTRIBUIDAS EN N CAPAS

En el capítulo anterior se mostró la evolución de las aplicaciones hasta llegar a las aplicaciones Web las cuales reunen las mejores características de las arquitecturas predecesoras. En este capítulo se estudia como se desarrollan estas aplicaciones bajo un marco de trabajo que también trata de reunir las mejores prácticas.

A continuación se presenta el ciclo de vida para las aplicaciones Web y como debe administrarse un equipo para su desarrollo desde el análisis hasta la instalación final de la aplicación. Esto es importante para conocer los distintos roles que deben considerarse en el equipo de trabajo de un proyecto de aplicación Web.

2.1. Análisis y diseño para aplicaciones Web

Generalmente las aplicaciones Web son aplicaciones empresariales que integran aplicaciones y requieren hacer un desarrollo bastante rápido y de alta calidad. Para cumplir estas demandas se necesitan equipos más grandes con roles especializados trabajando sobre un proceso de desarrollo.

Un proceso es un marco de trabajo para desarrollar una aplicación de *software* el cual da una guía de las actividades, documentos, roles, criterios de monitoreo y control del proyecto para hacer equipos de desarrollo productivos.

El proceso de desarrollo de aplicaciones que se describe ne este capítulo es el llamado RUP (*Rational Unified* Process, Proceso Unificado de Rational).

Un proceso recibe una entrada de insumos para realizar una serie de acciones para alcanzar una meta o producto determinado. El RUP es un proceso que nos define cómo tomar las entradas, que acciones realizar y quién realizará dichas acciones para producir un sistema nuevo.

El RUP es un proceso iterativo que puede constar de varios ciclos. Cada ciclo consta de: toma de requerimientos, análisis, diseño, implementación, pruebas, instalación y un período de evaluación. Es iterativo porque en cada ciclo el proceso es repetido y refinado hasta que cumpla con los requerimientos del sistema, y entonces es instalado. Este proceso de desarrollo del software tiene cuatro aspectos principales que se describen en la Tabla III.

Tabla III. Elementos del RUP para el desarrollo de aplicaciones

Flujo de trabajo	Provee una guía acerca del orden de las actividades del equipo
Integrantes del equipo	Describe a los encargados de elaborar los artefactos
Actividades	Provee una dirección de las tareas de cada integrante y del equipo como un todo
Artefactos	Ofrece un criterio de monitoreo y medida del avance de los productos del proyecto y sus actividades.

Un flujo de trabajo es un conjunto de actividades: toma de requerimientos, análisis, diseño, implementación, pruebas e instalación que producen resultados tangibles u observables. Cada flujo de trabajo necesita integrantes del equipo para completar las actividades y artefactos.

Entonces, el flujo de trabajo define el conjunto de actividades en el cual deben estar involucrados los integrantes del equipo para producir salidas o productos llamados artefactos. Un artefacto es una pieza de información persistente que es producida por los integrantes del equipo durante el proceso. Los artefactos pueden ser: modelos, código fuente, y documentos. Los artefactos pueden sufrir cambios durante el proceso para lo cual se lleva una historia de su evolución dentro de una administración de control de versiones.

El RUP tiene tres características principales:

- Guiado por casos de uso
- Se basa en una arquitectura céntrica
- Es iterativo e incremental

2.2. Desarrollo de Aplicaciones Web utilizando el RUP

Las aplicaciones Web tienen un ciclo de vida corto, lo que significa que una vez terminada no pasará mucho tiempo para que se le agreguen nuevas opciones, se realicen cambios de funcionalidad o simplemente cambie en su aspecto (colores, imágenes, etc.). En este punto el RUP es un proceso de desarrollo adecuado para aplicaciones Web.

RUP es una metodología para el desarrollo de aplicaciones que está compuesta por varios modelos. Un modelo es la representación abstracta de la aplicación que se va a construir, que se está construyendo o que se construyó. En el contexto de RUP, un modelo es un conjunto de artefactos cada uno expresando una vista del sistema. Cada integrante del equipo en el proceso de desarrollo usa o contribuye a uno o varios modelos.

La construcción de un modelo abstracto del sistema total ayuda a comprender mejor la complejidad del sistema y puede ser utilizado para responder cualquier duda que se de en el transcurso del proceso. Por ejemplo, un buen modelo nos puede indicar cuales componentes están asociados a un caso de uso y cuales son sus capacidades o alcances. También nos podrá ayudar a diagnosticar que impacto tendrá un cambio de requerimientos en el sistema.

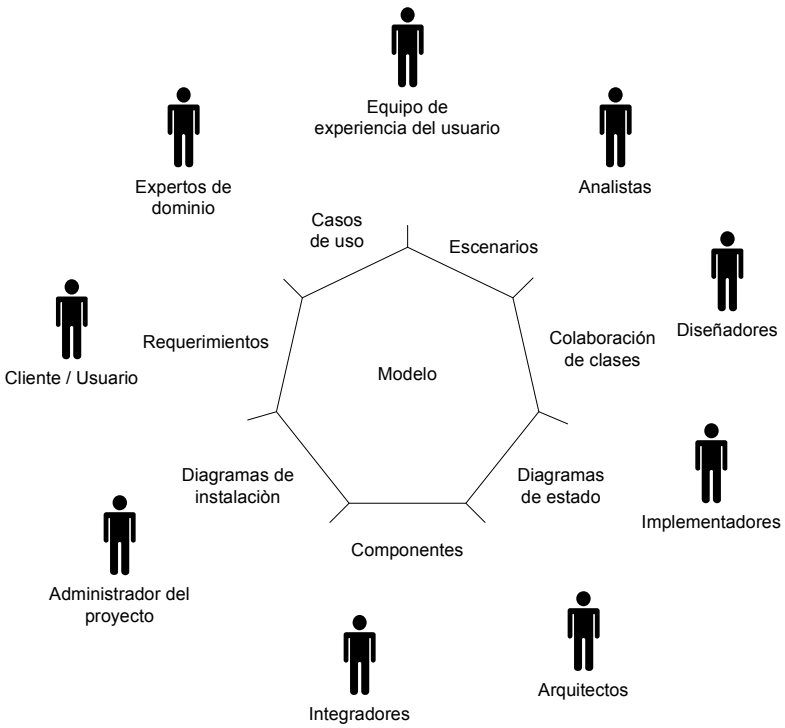
La utilización de esta debe adaptarse a la realidad de la empresa en recursos (tiempo y personal), pero no olvidando el objetivo de entregar aplicaciones de calidad a tiempo, dentro del presupuesto y que satisfaga las necesidades reales del cliente.

El proceso de desarrollo deberá entonces producir un modelo del sistema total compuesto por varios artefactos y los integrantes del equipo responsables de ellos, ya sea para su elaboración o su uso. En la figura 9 se muestra como de que se compone este modelo.

Los elementos a tomar en cuenta en las aplicaciones Web son:

- Interfaz en el cliente
- Lógica en el cliente
- Navegación entre páginas
- Componentes de la lógica de la aplicación
- Servicios de comunicación e integración
- Páginas Web del servidor

Figura 9. El modelo como artefacto central del proceso de desarrollo



Fuente: Building Web Applications with UML, pg, 110

El RUP nos indica las mejores prácticas que se deben llevar a cabo para que un proyecto de aplicación Web tenga éxito. Estas son mostradas en la tabla IV.

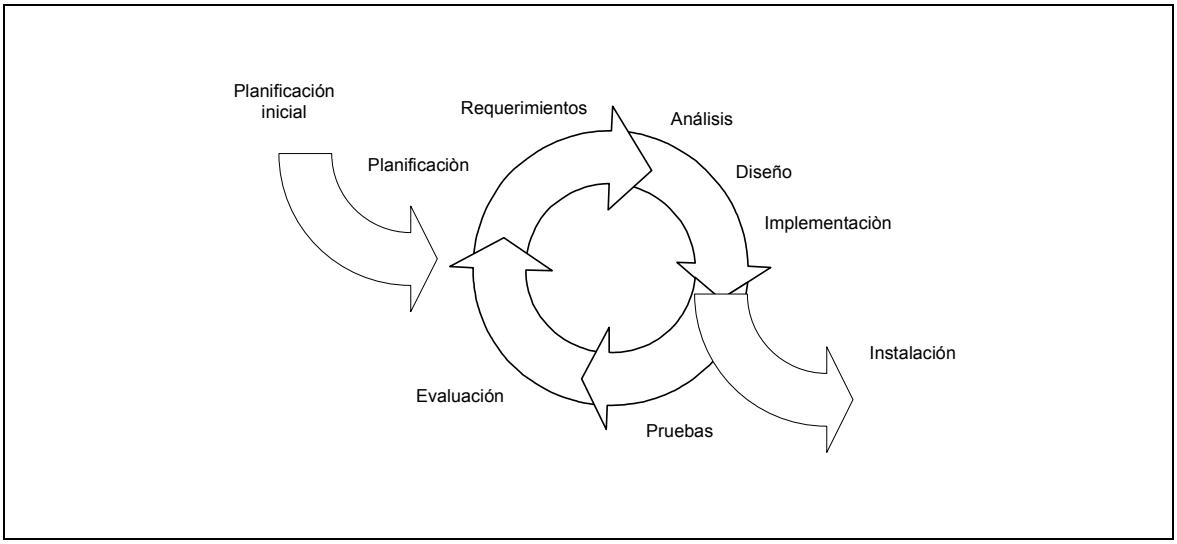
Tabla IV. Mejores prácticas para el desarrollo de aplicaciones Web

- Desarrollo iterativo
- Administración de requerimientos
- Utilización de arquitecturas basadas en componentes
- Utilización de modelos visuales
- Verificación de la calidad
- Administración de cambios

2.2.1. Desarrollo iterativo

El ciclo de vida que consta de análisis, diseño, implementación, instalación y pruebas se repetirá varias veces para obtener la versión final de la aplicación estando siempre presentes a lo largo del tiempo. Como se mencionó anteriormente las aplicaciones Web tienen un ciclo de vida corto y como para su instalación no implica ir a cada computadora a instalarla nuevamente se hacen más factibles las iteraciones. La figura 10 muestra el desarrollo iterativo.

Figura 10. Proceso iterativo



Fuente: Philippe Kuchten, *The Rational Unified Process: An Introduction* 2ed. (Boston, MA:Addison-Wesley, 2000, p.7)

2.2.2. Administración de requerimientos

Un requerimiento es una característica que el sistema debe cumplir. La administración de requerimientos es un proceso para organizar y documentar los requerimientos. Además, sirve para establecer acuerdos entre el cliente y usuarios, y el equipo del proyecto con respecto a requerimientos cambiantes.

Los requerimientos que deben administrarse son sobre la interfaz, la lógica de la aplicación y la información que desea almacenarse. También debe tomarse en cuenta tiempos de respuesta ya que estos varían de estar en un servidor local de desarrollo a estar en Internet, y son muy importantes en aplicaciones críticas.

2.2.3. Utilización de arquitecturas basadas en componentes

Para las aplicaciones Web es recomendable utilizar arquitecturas basadas en componentes distribuidos. Esta arquitectura tiene las ventajas mostradas en la Tabla V.

Tabla V. Ventajas de la arquitectura basada en componentes

Reutilización	Es mucho más fácil la reutilización de código a través de componentes.
Funcionalidad	En los componentes puede modularizarse mejor la funcionalidad.
Rendimiento	Un componente ya estará compilado y se ejecuta en una plataforma que optimizará su rendimiento.
Escalabilidad	Se tienen características especiales que hacen muy escalable a las aplicaciones.

Los componentes estarán distribuidos según sus funciones en varias capas:

- Componentes de la capa de presentación
- Componentes de la capa de lógica de la aplicación
- Componentes de la capa de datos

En la capa de presentación están los componentes que tienen la función de mostrar una interfaz al usuario para que interactúe con la aplicación. En la capa de la lógica de la aplicación se encuentran los componentes que tienen las reglas del negocio y el acceso a datos. La capa de datos la conforma la fuente de datos.

Cada capa debe ser independiente una de la otra. La capa de la lógica de la aplicación no debería sufrir cambios si se modifica la capa de presentación o la capa de datos. Por ejemplo, se podría tener una interfaz estilo una aplicación cliente / servidor o una interfaz a través de un navegador y ambas utilizando los mismos componentes de la lógica de la aplicación y la misma capa de datos.

2.2.4. Utilización de modelos visuales

Estos modelos son muy útiles en el diseño. El objetivo es tener un lenguaje en común que tenga un balance entre lo técnico y lo concerniente al negocio de la empresa para mostrar el diseño a todo el equipo del proyecto. El RUP utiliza al UML (*Unified Model Language*, Lenguaje Unificado de Modelado) para realizar los modelos visuales.

Actualmente el UML es el estándar en la industria del *software* y está compuesto por diversos elementos gráficos que se combinan con ciertas reglas para formar diagramas. Cada diagrama presenta una perspectiva de un sistema y a estos se le llama modelos. Un modelo de UML describe que hace un sistema, pero no como debe implementarse. Su principal objetivo es dejar muy claro lo que el sistema hace o hará cuando se implemente.

El UML tiene una extensión llamada WAE (*Web Application Extensión*, Extensión para Aplicaciones Web) que define específicamente un modelo para

aplicaciones Web. Los modelos recomendados para el desarrollo de una aplicación Web se enumenan en la Tabla VI.

Tabla VI. Modelos recomendados para una aplicación Web

• Modelo de administración del proyecto • Modelo de dominio • Modelo de caso de uso • Modelo de análisis/diseño • Modelo de implementación • Modelo de procesos • Modelo de seguridad • Modelo de experiencia del usuario / interface de usuario

2.2.5. Verificación de la calidad

La calidad se verifica realizando pruebas en cada iteración del proceso y conforme el sistema va creciendo en opciones implementadas las pruebas también aumentan. Es importante realizar pruebas en cada iteración y no al final de la etapa de implementación; porque cuando un problema se detecta tempranamente es mucho más barato resolverlo que encontrarlo al casi finalizar el proyecto. La calidad de una aplicación tiene varias dimensiones presentadas en la Tabla VII.

Tabla VII. Dimensiones de la verificación de la calidad

Tipo de dimensión	Descripción
Funcionalidad	Que cumpla con los requerimientos deseados.
Confiabilidad	Que la aplicación tenga la confianza de sus usuarios y el cliente.
Rendimiento	Que pueda tenerse control acerca de la carga y tiempos de ejecución.

2.2.6. Administración de los cambios

Como existen varios artefactos producidos en cada iteración hechos por varios integrantes de todo el equipo es necesario llevar el control de los cambios de todos los artefactos. En cada iteración los cambios son solicitados principalmente por el cliente o usuarios a través de nuevos requerimientos, por lo que deben evaluarse para evitar salirse del alcance inicial definido en el proyecto. En general, todo artefacto producido debe tener un número de versión según vaya cambiando por el avance o nuevos requerimientos.

2.3. Integrantes del equipo para el desarrollo de una aplicación Web

Como se mencionó anteriormente para la elaboración de los artefactos del proyecto es necesario que existan varios roles tomados por los integrantes del equipo. La cantidad de integrantes dependerá del tamaño y tiempo del proyecto. Lo que no debe olvidarse son las funciones que cumple cada rol para asignarlas. A continuación se da una breve descripción de cada rol, indicando sus funciones y con quién interactúa.

- **Cliente** / **usuario:** es la parte interesada en el desarrollo de la aplicación. Contribuye con los integrantes del equipo de desarrollo del proyecto para la elaboración de los requerimientos y no son responsables de ningún artefacto. Estos son los clientes, ejecutivos de la compañía, inversionistas y usuarios. También pueden ser los dueños de procesos de negocios que van a ser trasladados a una aplicación. Todos participan activamente en la evolución y entrega del sistema.

- **Administrador de proyecto:** encargado de coordinar a todo el equipo de desarrollo. Tendrá la comunicación con el cliente/usuario, y todo el equipo. Constantemente minimizará los riesgos del proyecto.

- **Arquitecto:** dirige y coordina actividades técnicas como análisis y diseño de la arquitectura, determinación de concurrencia distribución, estructuración de los modelos y priorización de casos de uso. Es responsable del contenido de los modelos y de definir las guías de diseño, guías de programación y el documento de la arquitectura. Define la arquitectura general del sistema.

- **Analista:** sus principales funciones son comprender y representar en modelos los requerimientos del cliente/usuario y comunicarlos al resto del equipo. Dependiendo de la naturaleza y el tamaño de la aplicación pueden existir varios analistas especializados en un área: en el negocio, en diseño de arquitectura, en diseño de la base de datos o en la que sea necesaria.

- **Diseñador:** a partir de los artefactos generados en el diseño empieza a generar modelos más concretos orientados a la implementación de la aplicación. Muchas veces el rol de analista y diseñador puede ser uno solo.

- **Desarrollador:** encargado de programar la aplicación en un lenguaje de programación produciendo componentes y páginas Web.

- **Integrador:** su tarea es unir los componentes implementados por los desarrolladores para formar la aplicación completa.

- **Probador:** tiene a su cargo realizar el plan de pruebas, generar *scripts*, y documentar los resultados.

- **Instalador:** debe conocer la arquitectura y la aplicación para poder generar instaladores y configurar la aplicación.

- **Experto de dominio:** lo forman las personas que tienen experiencia y conocimiento en el contexto del negocio donde se realizará la aplicación. Ayudan a definir conceptos y procesos acerca del negocio. Su principal artefacto producido es el glosario del dominio en el que se definen conceptos claves.

- **Experto en experiencia del usuario:** principalmente se encargan del diseño de la interfaz que tendrá la aplicación, no solamente evaluando colores y gráficas sino más bien su funcionalidad.

2.4. Artefactos producidos en el desarrollo de una aplicación Web

A continuación se presentan los principales artefactos elaborados en cada fase del proceso de desarrollo.

2.4.1. Administración del proyecto

Los principales artefactos de la administración del proyecto son el plan del proyecto, el plan de iteraciones y la administración de cambios del plan. El plan del proyecto y el plan de iteraciones tienen el calendario para el desarrollo de la aplicación e incluyen cosas como el personal, responsables, y fechas principales. El administrador del proyecto debe trabajar con el arquitecto acerca de los detalles técnicos para programar el calendario de la manera más real. En la Tabla VIII pueden verse los artefactos y los roles responsables.

Tabla VIII. Artefactos en la administración del proyecto

Rol	Artefacto
Administrador del proyecto	Plan del proyecto Plan de iteraciones Plan de administración de cambios en el plan
Administrador del proyecto Arquitecto	Plan de administración de iteraciones del plan

2.4.2. Definición del dominio

El modelo de dominio es realizado en UML y contiene el contexto general del negocio definiendo términos del contexto a través de un glosario. El glosario contiene definiciones de términos claves y conceptos que se utilizarán en las reuniones para definir reglas del negocio. El glosario debe estar accesible a todo el equipo. La Tabla IX muestra los artefactos producidos en la definición del dominio.

Tabla IX. Artefactos en la definición del dominio

Rol	Artefactos
Analista Expertos de dominio	Modelo de dominio Glosario

2.4.3. Definición de requerimientos

Un requerimiento es una declaración de lo que el sistema debe hacer. Los artefactos serán modelos de lo que se espera haga el sistema. Los roles encargados de producir artefactos pueden verse en la Tabla X.

Tabla X. Artefactos en la toma de requerimientos

Rol	Artefactos
Analista Cliente / Usuario (solo como fuente de información)	Visión
Arquitecto	Documento de guía de la interfaz del usuario (ahora es llamado guía de experiencia del usuario)
Administrador del proyecto Arquitecto Analista	Modelo de casos de usos Diagrama de secuencias Diagrama de actividades
Analista	Especificación de requerimientos suplementarios del negocio (políticas de seguridad y características) y requerimientos de arquitectura (mecanismos de seguridad, rendimiento y disponibilidad)

2.4.4. Análisis

Se elaboran las bases para realizar el diseño. Los artefactos producidos en esta fase pueden verse en la Tabla XI.

Tabla XI. Artefactos en el análisis

Rol	Artefactos
Arquitecto	Documento de la arquitectura Prototipo de la arquitectura
Analista	Realización de casos de uso Diagramas de secuencias Diagramas de estado Análisis de clases
Experto en experiencia del usuario	Diagramas de colaboración

2.4.5. Diseño

En esta fase se empieza a concretar la implementación de la aplicación por medio de modelos más completos según puede observarse n la Tabla XII.

Tabla XII. Artefactos en el diseño

Rol	Artefactos
Arquitecto	Vista de procesos
Diseñador	Diseño de lógica de páginas Diseño de clases Diseño de componentes
Experiencia del usuario Diseñador	Diseño de páginas Web

2.4.6. Implementación

En esta fase se toman los artefactos del diseño para que a partir de ellos se construya la aplicación por medio de herramientas de desarrollo produciendo componentes y páginas Web. En la Tabla XIII pueden observarse los artefactos y los roles involucrados en esta fase.

Tabla XIII. Artefactos en la implementación

Rol	Artefactos
Desarrollador	Código fuente Componentes compilados
Desarrollador Experto en experiencia del usuario	Páginas Web

2.4.7. Pruebas

Esta fase se enfoca en la evaluación de los artefactos ejecutables del sistema. Sus artefactos se muestran en la Tabla XIV.

Tabla XIV. Artefactos en las pruebas

Rol	Artefactos
Probador	Plan de pruebas Procedimientos de pruebas *Scripts* de pruebas
Arquitecto Desarrollador	Resultados de pruebas

2.4.8. Instalación

La instalación de la aplicación debe estar documentada por medio de instrucciones y *scripts* de instalación para cuando se realicen las primeras versiones sea fácil instalarlas. Es importante incluir los tópicos de seguridad y configuraciones. Los artefactos para esta fase son los mostrados en la Tabla XV.

Tabla XV. Artefactos en la instalación

Rol	Artefactos
Instalador	Plan de instalación Archivos de instalación Componentes compilados Páginas Web *Script* de base de datos Estilos de páginas

3. DISEÑO DE APLICACIONES WEB UTILIZANDO OBJETOS Y PATRONES

Las aplicaciones Web se mencionó en el capítulo 2, necesitan un marco de trabajo que garantize sus beneficios por ello en este capítulo se explora la metodología orientada a objetos y la utilización de patrones. Un patrón define una solución reutilizable de un problema bajo cierto contexto, y la puede describir utilizando la metodología orientada a objetos y UML orientándola a uno o varios roles del equipo del proyecto.

La metodología orientada tiene los conceptos de clases, abstracción, herencia, polimorfismo, encapsulamiento y relaciones permiten modelar sistemas de manera muy parecida al mundo real, y fomenta el desarrollo basado en componentes; de manera que primero se genera un sistema mediante un conjunto de objetos y sus relaciones para luego preocuparse por la implementación.

Los patrones se relacionan con la metodología orientada a objetos en el sentido que permiten organizar clases en estructuras comunes y comprobadas, para poder adaptarlas y aumentar su reutilización y pueden orientarse a un rol del equipo del proyecto.

La fórmula para el éxito de un proyecto es que todos los involucrados deben estar claros en lo que se va a realizar. El equipo de desarrollo del proyecto debe entender que es lo que se necesita a lo largo del proceso de desarrollo y es donde el UML se convierte en una herramienta muy valiosa para los patrones de diseño.

Los patrones de diseño permiten realizar implementaciones de aplicaciones tomando la experiencia de arquitectos y desarrolladores de aplicaciones que en un momento se enfrentaron a problemas similares y se encuentran bien documentados utilizando UML. El propósito de los patrones es hacer más fácil la reutilización de diseños y arquitecturas que han sido implementados exitosamente. Algunos de estos se describen en el capítulo 4 y 5.

La utilización de los patrones es una innovación para la creación de aplicaciones desde su arquitectura hasta la implementación en una plataforma específica. Como se dijo anteriormente uno de los principales objetivos de los patrones es la reutilización del *software*, pero también del diseño y arquitectura de la aplicación.

A continuación se define el concepto de los patrones de diseño y las partes que los componen. Luego se plantean los diferentes enfoques que puede tener un patrón y se explica como se van a estudiar los patrones para las aplicaciones Web distribuidas en N capas y los roles que deben exister dentro del equipo del proyecto.

3.1. Definición de patrón de diseño

¨Cada patrón describe a un problema, el cual ocurre varias veces en un entorno, y describe el núcleo de la solución, de tal forma que puede utilizarse un millón de veces más, sin resolverlo nuevamente¨. Esta definición fue dada por Christopher Alexander en su libro "*A Pattern Language*"en 1977, y se refería a patrones en la construcción de edificios, pero aplica igual a la construcción de aplicaciones de *software*.

En el ámbito de desarrollo de aplicaciones de *software* un patrón de diseño es una descripción para un problema común de arquitectura, diseño o implementación en un contexto dado y una solución recomendada que tendrá ciertas consecuencias al aplicarlo. Pero para poder aplicarlos es necesario que se encuentren bien documentados. En la Tabla XVI se muestran los principales elementos de un patrón.

Tabla XVI. Elementos de un patrón

Nombre del patrón	Se debe dar un nombre a un problema de diseño.
El problema	Describe cuando aplicar el patrón.
La solución	Describe los elementos de como hacer el diseño, sus relaciones, responsabilidades y colaboraciones con otros patrones.
Las consecuencias	Son los resultados de aplicar el patrón.

3.1.1. Nombre del patrón

El nombre del patrón debe describir un problema de diseño, sus soluciones y consecuencias en unas pocas palabras. Si se nombra inmediatamente a un patrón se incrementa el vocabulario del diseño y se obtendrá un alto nivel de abstracción. Se deben escoger un buen nombre que lo describa para que pueda ser identificado por otras personas.

3.1.2. El problema

Se describe cuando aplicar el patrón. Se debe explicar el problema y el contexto donde se puede dar. Algunas veces el problema podrá incluir una lista de condiciones que deben darse para poder aplicar el patrón.

3.1.3. La solución

Se describen los elementos de como hacer el diseño, sus relaciones, responsabilidades y colaboraciones. La solución no describe un diseño o implementación concreta y en particular, porque un patrón es como una plantilla que puede ser aplicada en muchas situaciones diferentes. Sin embargo, los patrones dan una descripción abstracta del diseño de un problema y como un conjunto de acciones pueden resolverlo.

3.1.4. Las consecuencias

En esta parte se describen los resultados de aplicar el patrón. Aunque las consecuencias muchas veces no son expresadas cuando se describen las decisiones de diseño son críticas para la evaluación de alternativas de diseños y conocer los costos y beneficios que se tendrán al aplicar el patrón. Podrían darse al escoger una plataforma de desarrollo o un lenguaje de programación. Los impactos que podrían darse son en flexibilidad, extensibilidad o portabilidad. Por ello listando las consecuencias explícitamente ayuda a comprenderlas y evaluarlas.

3.2. Descripción de los patrones de diseño

El nivel de abstracción de un patrón es relativo, y podría ser difícil encontrar uno que se podría aplicar, por ello es que debe documentarse de manera que pueda entenderse fácilmente, y esto se puede hacer incluyendo algunos de los siguientes aspectos según sea la naturaleza del patrón a describir.

3.2.1. Nombre del patrón y clasificación

El nombre del patrón debe poder identificar su esencia como tal, y describirlo de una forma resumida.

3.2.2. Alcances

Se debe indicar en forma resumida que hace el patrón, cual es su lógica, si resuelve algún caso especial o en que problema se enfoca principalmente.

3.2.3. Otros nombres del patrón

Pueden darse otros nombres al patrón si se considera que existen varios y que ayudaría a poder identificarlos mejor.

3.2.4. Motivación

Es interesante colocar un escenario que ilustre un problema de diseño y como el patrón puede resolverlo. Un escenario puede ayudar a comprender mejor la abstracción del diseño.

3.2.5. Aplicación

Es importante describir cuales son las situaciones en las que el patrón puede ser aplicado, y ejemplos de como reconocer estas situaciones.

3.2.6. Estructura

Una representación gráfica del patrón utilizando una notación basada en UML es muy útil. Según se considere necesario se pueden incluir distintos diagramas para poder especificar mejor la estructura.

3.2.7. Participantes

Se deben identificar cada uno de los elementos que interactúan dentro del patrón, y que responsabilidades tienen.

3.2.8. Colaboraciones

Debe especificarse cómo los participantes colaboran entre sí para llevar a cabo sus responsabilidades.

3.2.9. Consecuencias

Se pueden describir los resultados de utilizar el patrón y las consecuencias que produce el patrón para cumplir con sus objetivos.

3.2.10. Implementación

Se incluyen riesgos, sugerencias o técnicas que podrían apoyar la implementación del patrón.

3.2.11. Código de ejemplo

Es enriquecedor si puede dar a conocer fragmentos de códigos de como puede implementarse el patrón en algún lenguaje de programación o plataforma de *software* específica.

3.2.12. Usos conocidos

Ejemplos de patrones encontrados en sistemas reales, y podrían incluirse dos ejemplos de sistemas en contextos diferentes.

3.2.13. Patrones relacionados

Un patrón puede estar relacionado con otro dentro de un sistema, por lo que es importante indicar los patrones que están relacionados o que deberían utilizarse.

3.3. Organización de los patrones

Los patrones pueden organizarse desde varios puntos de vista:

- Por su nivel de abstracción: arquitectura, diseño e implementación.
- Desde la perspectiva de un aspecto de la aplicación

También cada patrón se relaciona con otros patrones formando agrupaciones de patrones que se enfocan a una solución general, formando:

- Agrupaciones de patrones: varios patrones relacionados con un tópico específico.
- Marcos de trabajo de patrones: varias agrupaciones de patrones definidos dentro de un contexto para una solución general.

Teniendo clara la organización de los patrones es fácil poder reutilizarlos en distintos problemas. Es útil organizarlos para llevar una secuencia adecuada para su aplicación en un problema.

3.3.1. Niveles de abstracción

La agrupación de patrones facilita la administración de estos, por ejemplo, si se está buscando un patrón para el diseño de la capa de presentación se buscará dentro de los patrones de presentación Web y se analizarán como están relacionados los patrones dentro de este grupo. Pero además, se deben tener varios niveles de abstracción porque la persona que esté buscando el patrón puede ser un desarrollador de aplicaciones o un arquitecto de la aplicación.

El desarrollador de sistemas estará más interesado en aspectos de implementación, pero el arquitecto en como se definirá la comunicación entre las capa de presentación y la capa de lógica de la aplicación. Cómo se puede observar los patrones pueden dividirse según el nivel de abstracción que se necesite, permitiendo ir de lo más general a lo más específico como se muestra en la Tabla XVII.

Tabla XVII. Niveles de abstracción de un patrón

Nivel de abstracción	Descripción
Patrones de arquitectura	Definen los patrones a un más alto nivel de abstracción dando una solución general.
Patrones de diseño	Incluyen patrones más especializados acerca del diseño de la aplicación enfocándose a soluciones específicas.
Patrones de implementación	Se refieren a dar soluciones a una plataforma de desarrollo específica.

3.3.1.1 Patrones de arquitectura

La definición de arquitectura tiene muchos sentidos según el ámbito, pero la arquitectura de las aplicaciones de *software* se refiere a la identificación de los elementos más importantes del sistema y las relaciones entre ellos para obtener una visión global del sistema. La arquitectura ayudará a entender el sistema, a organizar la instalación de la aplicación y a poder tener conocimiento de las acciones a tomar si la aplicación crecerá.

La definición de la arquitectura de una aplicación es una decisión muy importante que debe tomarse regularmente al inicio de un proyecto, y los patrones de arquitectura describen la solución ante determinada situación, pero también describen las consecuencias que se tendrán si se aplica.

El patrón enumerará los principales aspectos y sus relaciones para entender la arquitectura, pero deben examinarse los requerimientos definidos para la aplicación para tomar una decisión correcta. Un patrón de arquitectura define fundamentalmente la estructura de la organización de una aplicación dando un conjunto de subsistemas predefinidos, especificando sus responsabilidades e incluyendo reglas y guías para la organización de las relaciones entre ellos.

Los patrones de arquitectura definen como los principales elementos de la aplicación estarán organizados, las relaciones entre ellos y cómo será instalada la aplicación en ellos. Dan un nivel de abstracción bastante general pero define las bases de como la aplicación será construida desde el punto de vista de la plataforma de *software* y *hardware*.

3.3.1.2 Patrones de diseño

Luego de tener una visión global de la aplicación por medio de los patrones de arquitectura el siguiente nivel de abstracción que tienen los patrones es el de diseño. En estos patrones se analizan los componentes que formarán la aplicación tomando en cuenta los componentes de *software* a construir, la plataforma de *software* y el equipo de *hardware* a utilizar.

Un patrón de diseño provee un esquema para especificar los subsistemas o componentes de una aplicación, las relaciones entre ellos y como se comunican en un contexto determinado.

3.3.1.3 Patrones de implementación

Este conjunto de patrones son más detallados y muchas veces orientados a lenguajes de programación o plataformas de *software* específicas. Muchos de estos patrones son propuestos por empresas que proveen las plataformas de desarrollo de las aplicaciones.

3.3.2. Perspectivas de los aspectos de la aplicación

Una aplicación de *software* tiene varios aspectos que deben tomarse en cuenta para que pueda funcionar. Primero debe existir la aplicación que se va a construir pero los aspectos a considerar son que debe instalarse sobre una plataforma de *software* y una infraestructura de *hardware*. En la Tabla XVIII se muestran los aspectos de una aplicación.

Tabla XVIII. Aspectos de una aplicación

Aplicación	*Software* que se implementará: diseño de clases, implementación de componentes, etc.
Plataforma del *software*	Es el *software* sobre el cual se construye la aplicación. Por ejemplo: bases de datos, sistemas operativos, servidores de páginas Web, monitores transaccionales, etc.
Infraestructura de *hardware*	Es el *hardware* necesario para que la aplicación pueda ser instalada y ejecutada
Instalación de la aplicación	Se refiere a como debe instalarse la aplicación en la plataforma de *software* y la infraestructura de *hardware*

Los patrones también pueden enfocarse a un aspecto de la aplicación sirviendo como guía para desarrollarla en un contexto bien definido. Además, estos aspectos ayudan a delimitar el contexto, por ejemplo, para las aplicaciones Web distribuidas en N capas se puede definir el tipo de base de datos, la plataforma de *software* necesaria y como debería ir instalada.

3.3.3. Agrupaciones de patrones

Los patrones pueden agruparse y estar relacionados entre sí por un tema común. Por ejemplo si se tiene el grupo de patrones de la Presentación Web se tendrán todos los patrones que son útiles para ese tema. Las agrupaciones de patrones no indican como aplicarlos sino solamente los reúnen.

3.3.4. Marco de trabajo de patrones

Un marco de trabajo de patrones es un conjunto de patrones clasificados, con aspectos de la aplicación definidos y con diferentes niveles de abstracción referidos a la naturaleza de un problema más general pero delimitado por restricciones y en el cual se define un lenguaje común.

3.3.4.1 Restricciones de un marco de trabajo de patrones

Las restricciones están dadas por los aspectos de la solución y el nivel de abstracción. Entonces se podría referir a un patrón de la plataforma de *software* al nivel de arquitectura o al nivel de diseño. Pero en el nivel de arquitectura deben definirse las restricciones de acuerdo a la solución para que los siguientes niveles de abstracción sean acordes a la arquitectura definida.

Para el caso de las aplicaciones Web distribuidas en N capas deben definirse los patrones de arquitectura para cada uno de los aspectos de la aplicación y así poder ir bajando de nivel hasta llegar al nivel de implementación. Las restricciones ayudarán a enfocarse en la arquitectura que se va a desarrollar.

3.3.4.2 Patrones base

Estos patrones sirven de guía para desarrollar otros más específicos y delimitan el alcance del marco de trabajo. Se encuentran como los patrones raíces dentro de cada restricción y ayudan a clasificarlos. Regularmente están orientados en la arquitectura.

3.4. Patrones para una aplicación Web distribuida en N capas

Una aplicación Web distribuida en N capas debe ser una aplicación robusta y escalable, soportar una gran cantidad de usuarios, estar disponible casi todo el tiempo y poder ser accedida desde Internet. Para esto se requieren múltiples servidores, conexiones a datos remotos y servicios especiales por lo que es muy necesaria la definición de una arquitectura y diseño debido a la complejidad que tienen.

La arquitectura de este tipo de aplicación se divide en capas que deben diseñarse según las necesidades de la aplicación. La razón de las capas entre otras cosas, es dividir la aplicación para facilitar su crecimiento e implementación, pero se involucra complejidad que ya ha sido identificada en patrones.

El conocimiento y experiencia adquirida al implementar una aplicación es plasmado en un patrón que puede utilizarse para resolver otro problema similar, es como hacer una copia del conocimiento de arquitectos y desarrolladores de aplicaciones. Cada empresa tiene sus propios requerimientos por lo que el patrón no será exacto pero guiará hacia la solución particular.

Entonces, una aplicación es la combinación de patrones, ya que cada uno describe y resuelve efectivamente un problema separando particularidades que pudiese tener la aplicación. Un patrón debe de dar una solución general y no específica para que pueda aplicarse en varios casos.

Los patrones pueden tener varios niveles de abstracción dependiendo de lo especializado que sea para resolver ciertos problemas. Esto facilita la comprensión del patrón porque va de lo general a lo más específico, por lo que si se comprende lo general será más fácil comprender lo específico.

3.4.1. Agrupación de patrones para aplicaciones Web

Una aplicación Web distribuida en N capas tiene las agrupaciones mostradas en la Tabla XIX. Estas agrupaciones permiten tener una visualización general de cómo encontrar un patrón respondiendo a la pregunta de un problema bastante general en el contexto de las aplicaciones Web.

Tabla XIX. Agrupaciones de patrones para una aplicación Web distribuida en N capas

Agrupaciones de patrones	Problema
Presentación Web	¿Cómo crear la capa de presentación para aplicaciones Web?
Distribución de la aplicación	¿Cómo dividir la aplicación en capas e instalaras en la infraestructura de *hardware*?
Componentes distribuidos	¿Cómo comunicar componentes que se encuentren en diferentes computadoras?
Servicios para aplicaciones distribuidas	¿Qué servicio de aplicaciones distribuidas se debe utilizar en determinada aplicación?
Servicios de aplicación	¿Cómo acceder servicios proveídos por otras aplicaciones? y ¿Cómo exponer los servicios que se realicen para que estén disponibles para otras aplicaciones?

3.4.2. Definición del marco de trabajo de patrones

Cómo se mencionó anteriormente un marco de trabajo de patrones tiene ciertos aspectos de la aplicación que se deben definir para delimitar los patrones a utilizar, y analizarlos en varios niveles de abstracción: arquitectura, diseño e implementación. A continuación se describen las restricciones y los pattrones base de un marco de trabajo para una aplicación Web.

3.4.2.1 Restricciones

- Aplicación: para una aplicación Web se tienen las siguientes restricciones:
  1. El diseño de la aplicación será orientado a objetos.
  2. La aplicación estará distribuida lógicamente en N capas.

- Plataforma de *software*: esta también restringe un marco de trabajo, y para una aplicación Web se tiene:
  1. Para el almacenamiento de la información se considera una base de datos relacional utilizando transacciones OLTP (*On Line Transaction Processing*, Proceso de transacciones en lìnea*)*.
  2. Se consideran los protocolos de componentes distribuidos.
  3. Se consideran los servicios para aplicaciones distribuidas.

- Infraestructura de *hardware*: la infraestructura de red y servidores deberán soportar la distribución de la aplicación en N capas, para que pueda ser una aplicación robusta y escalable.

- Instalación de la aplicación: no tiene restricciones propias, sino dependerá del las restricciones de la plataforma de *software* y la infraestructura de *hardware*.

3.4.2.2 Patrones base

Las restricciones del marco de trabajo tienen suficiente información para empezar a describir una solución para aplicaciones Web utilizando patrones. Los patrones se describirán desde el punto de vista de los niveles de abstracción como es mostrado en la Tabla XX.

Tabla XX. Patrones base de las aplicaciones Web

Nivel de abstracción	Patrones base
Arquitectura	• Aplicación distribuida en N capas • Componentes distribuidos
Diseño	• Capa de presentación • Capa de lógica de la aplicación • Capa de datos
Implementación	• Plataforma de *software* de Microsoft .NET • Plataforma de *software* de CORBA • Plataforma de *software* J2EE

3.5. UML como apoyo al diseño por patrones

El UML es utilizado como lenguaje de modelado para describir los patrones. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se utiliza para entender, diseñar, configurar, mantener y controlar la información sobre tales sistemas. No es un lenguaje de programación pero ciertas herramientas que lo implementan pueden ofrecer generadores de código para una gran variedad de lenguajes de programación.

Es un lenguaje de modelado de propósito general y no una descripción de un proceso de desarrollo detallado. Se pretende que sea utilizado como lenguaje de modelado subyacente a la mayoría de procesos de desarrollo existentes o de nueva creación, pero está especialmente pensado para apoyar un estilo de desarrollo iterativo e incremental.

Actualmente existen implementaciones como Microsoft® Visio, Rational Rose, SELECT Enterprise y Visual UML entre otras que siguiendo un proceso de desarrollo de aplicaciones pueden modelar y generar código en lenguajes de programación o hacer ingeniería inversa.

3.6. Perspectivas de los patrones de arquitectura y roles

Existen definiciones de cómo los patrones pueden mapearse a las perspectivas de la arquitectura y los roles. Esto facilita a cada rol encontrar que patrones puede aplicar según su función en el desarrollo de la aplicación. La que se describe a continuación está siendo propuesta por Microsoft y está en una etapa de propuesta inicial. Se le llama "Tabla de Organización del Espacio de la Arquitectura Empresarial" y permite organizar y clasificar los patrones, además de descubrir que áreas no están documentadas.

Esta tabla fue construida sobre cuatro trabajos existentes: Marco de trabajo de la Arquitectura Empresarial de Zachman (*Zachman Framework for Enterprise Architecture*), IEEE 471 (Estándar de descripción de arquitecturas), Información de la arquitectura Empresarial de Consultores Andersen (*Andersen Consulting's Enterprise Information Architecture*), y el desarrollo guiado por pruebas (*test-driven development*).

Dentro de esta tabla se mapean los artefactos que corresponden a cada rol y a cada perspectiva de la arquitectura. La tabla de organización divide la arquitectura empresarial en cinco perspectivas:

1. Arquitectura del negocio
2. Arquitectura de integración
3. Arquitectura de la aplicación
4. Arquitectura operacional
5. Arquitectura de desarrollo

3.6.1. Arquitectura del negocio

Provee una base para otras perspectivas de la arquitectura. Las aplicaciones Web empresariales existen para darle un valor agregado a la empresa y deben estar alineadas según los objetivos del negocio. Sin una arquitectura del negocio bien definida el desarrollo de la aplicación puede ser improvisada y reactiva lo cual puede terminar en el fracaso de la aplicación. Los roles involucrados en esta perspectiva son:

- Gerente general de la empresa (CEO, *Chief Enerprise Officer*)
- Administrador general
- Dueño del proceso
- Trabajador del proceso

3.6.2. Arquitectura de integración

Abarca la integración de sistemas dentro de la empresa y fuera de ella con otras empresas. Los roles involucrados son:

- Arquitecto empresarial
- Diseñador
- Desarrollador

3.6.3. Arquitectura de la aplicación

Describe todos los elementos necesarios de *software* necesarios para poder ejecutar la aplicación como bases de datos, servidores Web, servidores de

aplicación, redes, componentes, lógica de la aplicación, y la aplicación como tal. Los roles que se incluyen dentro de esta perspectiva son:

- Arquitecto empresarial
- Arquitecto
- Diseñador
- Desarrollador

3.6.4. Arquitectura operacional

Define las operaciones cuando el sistema se encuentre en producción para mantenerlo estable, seguro, escalable y administrado. Contiene elementos relacionados a eventualidades del sistema, administración del rendimiento, administración de usuarios, copias de seguridad, monitoreo y mejora de rendimiento. Los roles involucrados son:

- Arquitecto
- Ingeniero de sistemas

3.6.5. Arquitectura de desarrollo

Trata de cómo implementar las arquitecturas anteriores, de cómo construir y mantener una aplicación de una forma sistemática y eficiente. Está compuesta por herramientas de diseño y desarrollo, monitoreo, pruebas, bitácoras y otras. Los roles involucrados son:

- Administrador de la configuración
- Instalador

- Probador
- Desarrollador

3.6.6. Interrogativas del espacio de la arquitectura

Las perspectivas de la arquitectura y los roles proveen cierta clasificación de artefactos que debe ser producidos, pero puede incluirse más granularidad por medio de interrogativas que están relacionadas con las iniciativas y propuestas dentro del negocio de la empresa. Estas son mostradas en la Tabla XXII.

Tabla XXI. Interrogativas del espacio de la arquitectura

Propósito (¿Por qué?)	Debe especificar cual es la razón para la decisión de arquitectura hecha en respuesta a una iniciativa.
Datos (¿Qué?)	Qué información es requerida o será producida como resultado de haber tomado una decisión o iniciativa.
Función (¿Cómo?)	Cómo las decisiones o iniciativas de arquitectura podrán realizarse
Calendario (¿Cuándo?)	Cuándo tendrán lugar eventos o serán exigidos los productos de la iniciativa.
Red (¿Dónde?)	Dónde se ubicarán los productos de la iniciativa tomada.
Personal (¿Quién?)	Quiénes se beneficiarán o serán afectados por la iniciativa.
Control (Evaluar)	Cómo se medirá que el propósito de la iniciativa ha sido alcanzado.

3.6.7. Combinación de perspectivas, roles e interrogativas

La tabla de organización queda como se muestra en la figura 11. Cada Intersección de rol e interrogativa puede tener artefactos que pueden ser guiados por patrones.

Figura 11. Tabla de Organización del Espacio de la Arquitectura Empresarial

		Propósito	Datos	Función	Tiempo	Red	Personal	Scorecard
Arquitectura del negocio	CEO							
	Administrador Gral.							
	Dueño de proceso							
	Trabajador del proceso							
Arquitectura de Integración	Arquitecto empresarial							
	Diseñador							
	Desarrollador							
Arquitectura de Aplicación	Arquitecto empresarial							
	Arquitecto							
	Diseñador							
	Desarrollador							
Arquitectura operacional	Arquitecto de sistemas							
	Ingeniero de sistemas							
Arquitectura de dedarrollo	Ing.Adm.Configuración							
	Integrador							

4. EJEMPLOS DE PATRONES DE ARQUITECTURA

En el capítulo 3 se introducen los conceptos de patrones. En este capítulo se muestran algunos ejemplos de los patrones de arquitectura considerados como base para los patrones de diseño y también para comprender de mejor manera la arquitectura de las aplicaciones Web.

Los patrones de arquitectura dan un punto de vista de cómo a nivel general puede realizarse una aplicación considerando un conjunto de servicios que pueden ser utilizados según el contexto donde se desarrolle. Se inicia cada patrón describiendo su contexto, el problema y se termina dando una solución y consecuencias de aplicar el patrón.

4.1. Elementos de la arquitectura de una aplicación distribuida

4.1.1. Contexto

La arquitectura de una aplicación está compuesta por elementos básicos sobre los cuales se construirá y ejecutará la aplicación. La decisión sobre la arquitectura dependerá del análisis, diseño e implementación los cuáles implícitamente indicarán cierta arquitectura a utilizar. Por ejemplo, si se necesita comunicar con un sistema heredado se necesitará tener servicios de comunicación con dicho sistema o si es requerida alta disponibilidad debe incluirse soluciones de *clusters* (grupos de servidores).

4.1.2. Problema

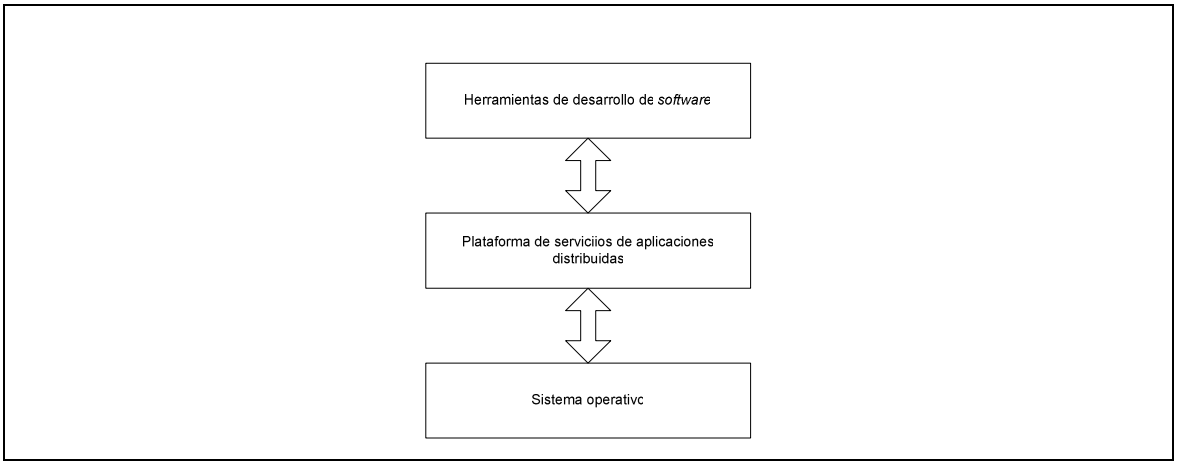
¿Cuáles son los elementos básicos que deben tomarse en cuenta para definir la arquitectura de una aplicación Web distribuida en N capas?

4.1.3. Solución

Los aspectos de la arquitectura para construir una aplicación son: la herramienta para el desarrollo del *software*, la plataforma de servicios para aplicaciones distribuidas y el sistema operativo. Cada uno de estos elementos deben ser compatibles e independientes entre sí. También deben ser capaces de comunicarse entre sí como se muestra en la Figura 12.

La herramienta para el desarrollo del *software* la constituye principalmente el lenguaje de programación que generará los ejecutables de la aplicación (componentes, páginas Web, etc.) que deberán poder utilizar los servicios para aplicaciones distribuidas que estarán instalados sobre un sistema operativo.

Figura 12. Elementos de la arquitectura de aplicaciones Web distribuidas



4.1.4. Consecuencias

Al definir los elementos básicos de la arquitectura de aplicaciones según los requerimientos y decisiones del equipo de trabajo se debe obtener una plataforma de *software* compatible sin tener problemas de elementos que no se consideraron en el transcurso del proyecto.

4.1.5. Roles

Para definir deben estar los líderes de los equipos de los siguientes roles:

- Administrador del proyecto
- Arquitecto
- Analista
- Desarrollador
- Integrador de aplicaciones

4.2. Arquitectura de las aplicaciones distribuidas en N capas

4.2.1. Contexto

Cuando una aplicación Web es distribuida en N capas es porque cada capa tiene la función de dar servicios específicos a las otras capas. Estos servicios son construidos sobre la plataforma de *software* sobre la cual se ejecutará la aplicación, y solamente necesitan una interfaz para que puedan ser utiizados.

Podría decirse que la existencia de los servicios contribuye a la separación de capas, ya que no existe un servicio centralizado para toda la aplicación sino una colaboración de servicios para lograr el funcionamiento de la aplicación.

4.2.2. Problema

¿Cuáles son los servicios para aplicaciones distribuidas? ¿Dónde se encuentran ubicados?

4.2.3. Solución

Los usuarios de las aplicaciones Web solamente necesitan de un navegador para accederla. El navegador se encarga de presentarla en una forma gráfica dando servicios a la capa de presentación. También se necesita de un sistema operativo para poder realizar las tareas de comunicación sobre la red.

En el servidor se necesita como base un sistema operativo que administre los recursos solicitados por la plataforma de aplicaciones distribuidas la cual tiene tres tipos de servicios generales como se muestra en la Tabla XXII. Cuando se hace referencia a un servidor no es exactamente una computadora, sino más bien la existencia de un proveedor de dicho servicio.

Tabla XXII. Tipos de servicios en la plataforma de aplicaciones distribuidas

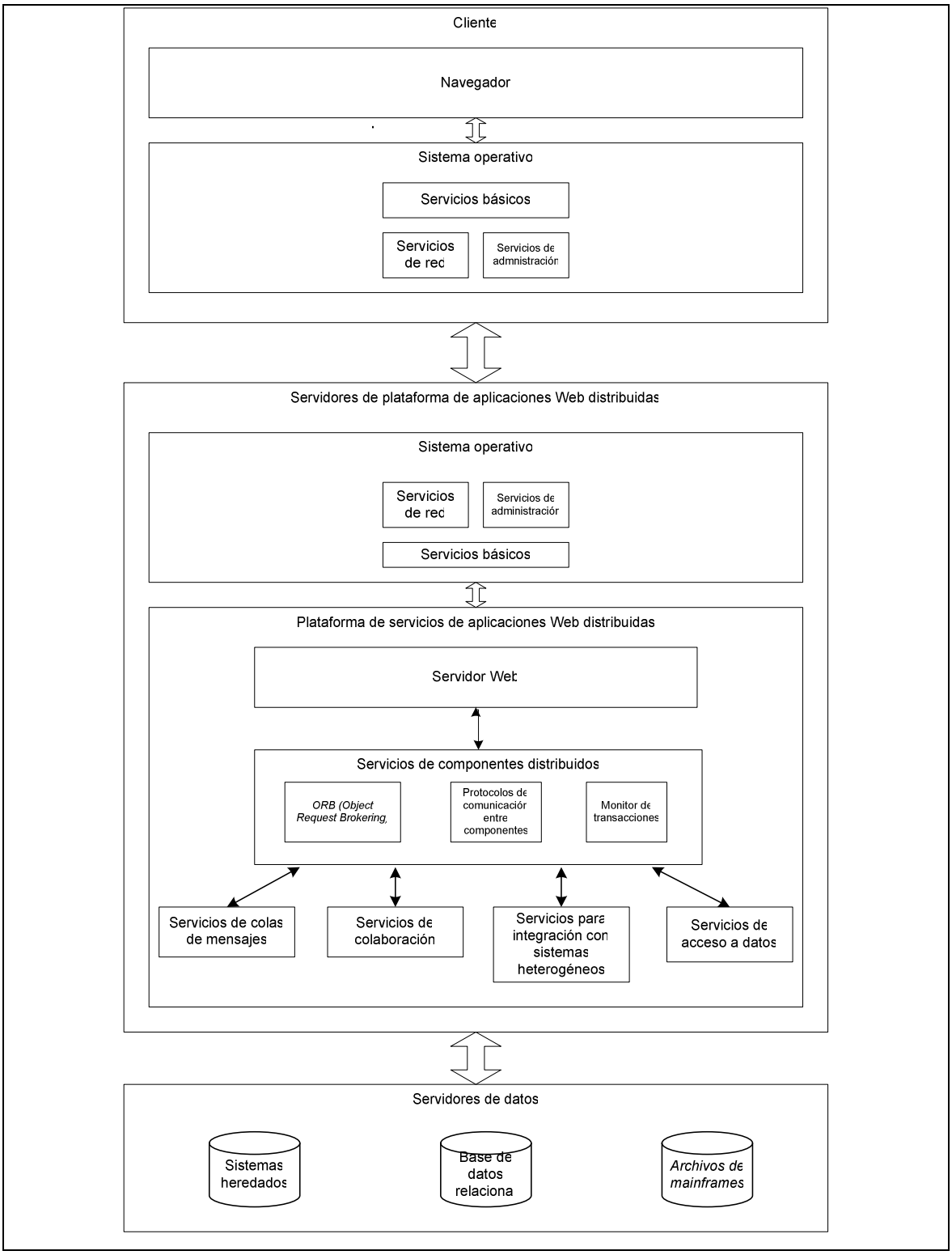
• Servicios Web • Servicios de aplicaciones distribuidas • Servicios de acceso a datos

Los servicios de la plataforma de aplicaciones distribuidas son los que permiten que la aplicación sea construida sobre ellos. Para las aplicaciones Web el servidor Web es el que procesa las solicitudes de las páginas, pero este se comunica con servicios de aplicaciones distribuidas como componentes, monitores transaccionales, etc. según la necesidad de la aplicación.

La tendencia actual de desarrollo de aplicaciones es implementar aplicaciones Web para hacer integraciones con aplicaciones ya existentes. Cuando se hacen estas integraciones es porque en las aplicaciones actuales no pueden crearse aplicaciones para Internet o no es su objetivo principal. Entonces, no se deben descartar estos sistemas heredados (llamados *legacy systems*) porque se puede realizar una comunicación con ellos y hacer que sean la fuente de datos.

Los elementos de la plataforma de aplicaciones distribuidas deben ser capaces de comunicarse. Por ejemplo, los componentes distribuidos de la aplicación pueden invocarse desde las páginas Web y tener acceso a la capa de datos por medio de algún servicio de base de datos o servicio de interconexión con sistemas heterogéneos, por ello es que se le llama una plataforma En la figura 13 pueden observarse los servicios involucrados en una aplicación Web distribuida en N capas.

Figura 13. Arquitectura de una aplicación distribuida



4.2.4. Consecuencias

Dependiendo de la complejidad de la aplicación distribuida y de los sistemas que integre así será el número de capas que se construyan en la aplicación. Como se puede observar el número de capas es variable dependiendo la naturaleza de la aplicación por ello es que se le llama de N capas.

Existe un retardo en la llamada a los servicios de cada capa y esto puede ocasionar problemas de rendimiento de la aplicación. Para contrarrestar esto debe tomarse en consideraciones en el diseño para no involucrar más servicios innecesariamente. La arquitectura de N capas ayuda a encapsular la complejidad de aplicaciones grandes, pero puede agregar complejidad para aplicaciones pequeñas.

4.2.5. Roles

Este patrón es útil para los siguientes roles:

- Arquitecto
- Desarrollador
- Integradores de aplicaciones

4.3. Plan de instalación para una aplicación Web distribuida

4.3.1. Contexto

Como se mencionó en el patrón de "Arquitectura de las aplicaciones distribuidas en N capas" existen varios servicios que colaboran con el funcionamiento de la aplicación y todos ellos forman la plataforma de *software*. Además, se encuentra la aplicación Web que básicamente estará compuesta de componentes de *software* y páginas Web. Todo esto debe instalarse en los servidores que forman la infraestructura de *hardware*.

Debe considerarse también que debe instalarse en las computadoras de los usuarios que accederán la aplicación. En las aplicaciones Web se trata de que el cliente sea liviano en el sentido de que no debe instalarse casi nada a parte de un sistema operativo, un navegador y acceso a Internet.

4.3.2. Problema

¿Cómo se instalará la aplicación y la plataforma de *software* en la infraestructura de *hardware*?

4.3.3. Solución

La aplicación debe ser instalada sobre servidores y ser accedida por los usuarios desde un navegador a través de la red.

La mayor dificultad de la instalación está en los servidores ya que se puede distribuir el procesamiento, según la carga o el tipo de servidor que se requiera. Existen varios tipos de servidores, según la función que prestan (vea la muestra en la tabla XXIII).

Tabla XXIII. Tipos de servidores, según su función

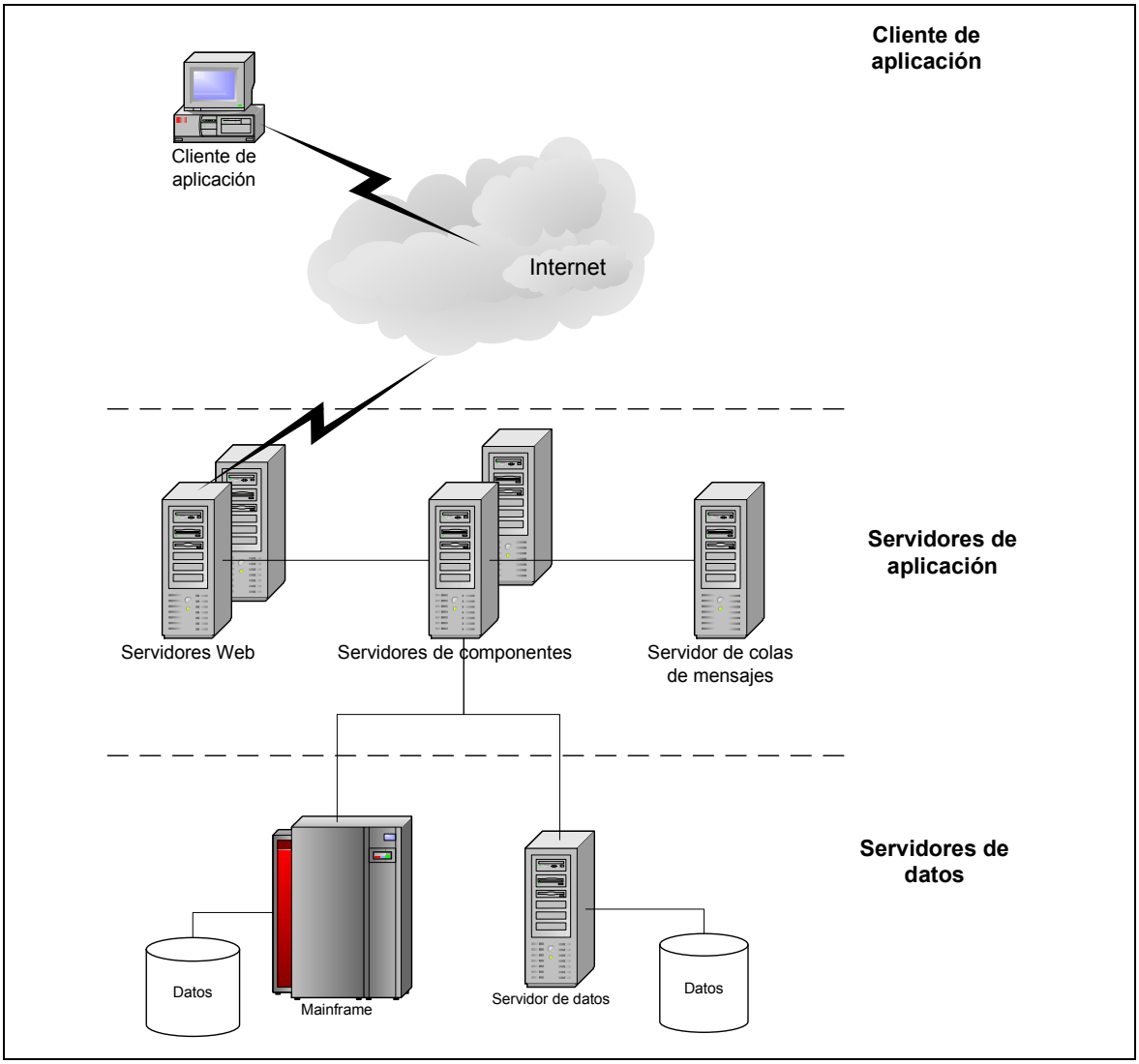
Servidores de plataforma de aplicaciones distribuidas	Se encuentran los servicios de las aplicaciones distribuidas. Por ejemplo: servicios de páginas Web, servicios de componentes distribuidos, monitores transaccionales, servicios de conexión con sistemas heterogéneos, etc.
Servidores de datos	Administradores de los datos que accederá la aplicación. Estos pueden ser varios si existen varias fuentes de datos para la aplicación. Solo actuarán como repositorios de información teniendo muy poca o ninguna lógica de la aplicación.

En las computadoras de los usuarios debe instalarse solamente un navegador y tener conexión por medio de la red al servidor de páginas Web. En los servidores se tendrá instalada la plataforma de *software* con la infraestructura de *hardware* que la conformarán principalmente los servidores y la red. La aplicación Web distribuida deberá instalarse básicamente las páginas Web, componentes de *software* y la base de datos.

Muchas veces solamente se tiene un servidor que realiza todas las funciones, y esto puede hacerse así dependiendo de la carga que se tenga. Cuando se tiene mucha carga y desea mantenerse un buen rendimiento es recomendable dividir la tarea en varios servidores según su función, y si es necesario tener varios servidores para una misma función (*clusters* de servidores, grupos de servidores). La instalación de la aplicación podría quedar como en la figura 14.

La decisión de tener *clusters* de servidores dependerá de los requerimiento de la aplicación por ejemplo: cantidad de usuarios conectados simultáneamente, alta disponibilidad de la aplicación o si es una aplicación crítica que requiere alto rendimiento.

Figura 14. Infraestructura y servicios de aplicaciones distribuidas



Después de instalar la plataforma del *software* es necesario instalar los componentes de la aplicación. De manera general la instalación de los componentes de la aplicación se muestra en la Tabla XXIV.

Tabla XXIV. Instalación de los componentes de la aplicación

Servidor Web (presentación)	Páginas Web (Componentes de inferface de usuario) Componentes de proceso de interface Servicios Web
Servidores de aplicación (lógica de la aplicación)	Componentes de lógica de la aplicación Componentes de acceso a datos
Servidor de datos	Base de datos

4.3.4. Consecuencias

La instalación de una aplicación Web con su plataforma e infraestructura necesaria requieren un equipo multidisciplinario ya que existen diferentes elementos a instalar donde se requieren ciertas habilidades específicas.

4.3.5. Roles

- Arquitecto
- Ingeniero de sistemas (Administrador de configuración)

4.4. Sistemas operativos para aplicaciones Web distribuidas

4.4.1. Contexto

La plataforma de aplicaciones distribuidas en el servidor debe ser instalada sobre un sistema operativo. El sistema operativo debe proveer servicios básicos y lo más importante es que sea compatible con la plataforma. También las computadoras clientes deben tener un sistema operativo.

4.4.2. Problema

¿Qué servicios básicos debe dar un sistema operativo a una aplicación distribuida?

4.4.3. Solución

El sistema operativo en los clientes no es relevante porque el cliente universal es el navegador que se vale de los servicios de red que preste el sistema operativo para comunicarse con la aplicación. A continuación, en la Tabla XXV se presentan ciertas características que debe tener un sistema operativo para el servidor.

Tabla XXV. Servicios de un sistema operativo

Servicios básicos	• Administración de recursos compartidos • Multitarea • Protocolos de comunicación • Soporte de redes • Seguridad • Administración eficiente de los recursos
Servicios extendidos	• Protocolos de comunicación especializados • Escalabilidad para aplicaciones • Mecanismos de alta disponibilidad

4.4.4. Consecuencias

La aplicación no debe ser dependiente del sistema operativo sino de su plataforma de *software*. El sistema operativo deberá encargarse de los solamente de los servicios básicos y poder trabajar en conjunto con la plataforma de *software*.

4.4.5. Roles

- Arquitecto
- Ingeniero de sistemas

4.5. Servidor Web

4.5.1. Contexto

El servicio de páginas Web es el principal elemento para la capa de presentación. Este es instalado en un servidor llamado Servidor Web, el cual administra las solicitudes de las páginas y servicios Web.

4.5.2. Problema

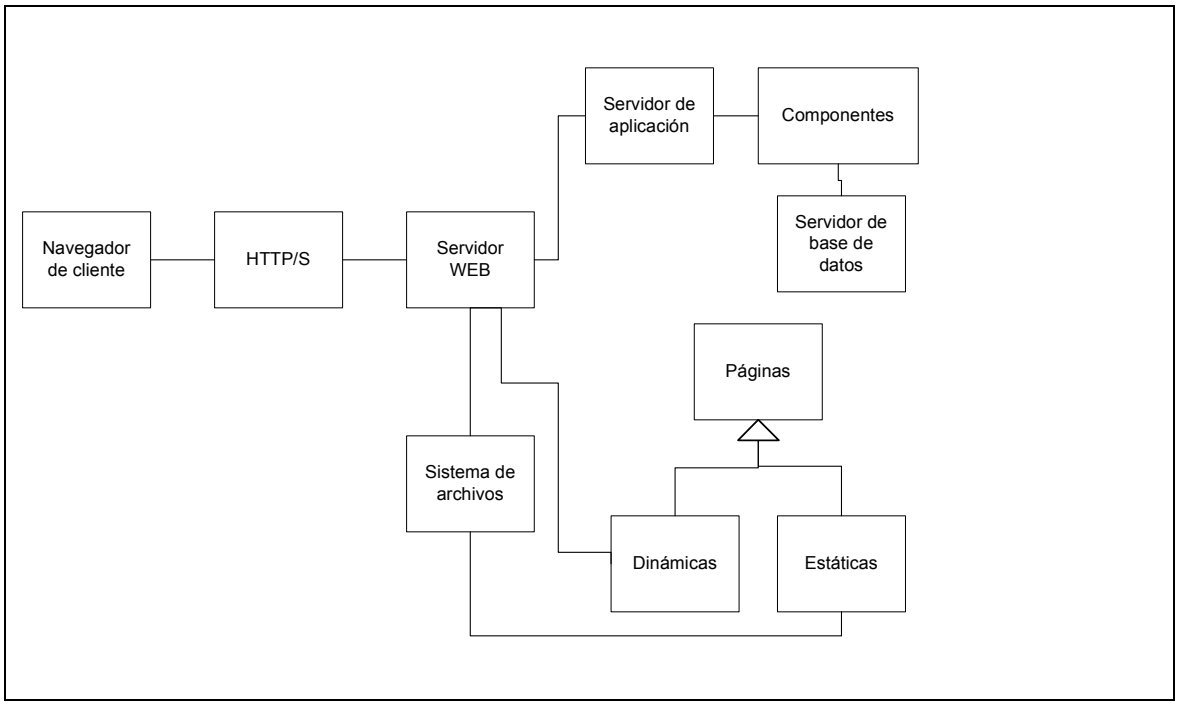
¿Cuáles funciones realiza el servidor Web?

4.5.3. Solución

Este servicio es el encargado de procesar las solicitudes realizadas desde un navegador por un cliente. Los servicios de páginas Web deben estar integrados con servicios como el de componentes distribuidos para procesar la página o servicio Web y devolver un resultado al navegador para que presente los resultados. Entonces, se tiene un cliente universal y un servidor que atiende las solicitudes devolviendo resultados en un formato estándar.

El servidor de páginas Web se encarga de atender las solicitudes de los clientes utilizando protocolos de comunicación como HTTP, HTTPS, FTP, SOAP, etc. También administra seguridad por medio de accesos a los archivos y del protocolo de seguridad SSL (*Socket Secure Layer*, Capa de conexión segura) que permite hacer conexiones seguras. Para las páginas Web existen varias tecnologías según el servidor de páginas Web que se utilice dentro de la plataforma. La figura 15 muestra el esquema general de un Servidor Web.

Figura 15. Servidor Web



4.5.4. Consecuencias

Los servicios de páginas Web permitirán construir las aplicaciones Web, que en combinación con los componentes distribuidos forman las aplicaciones Web distribuidas en N capas.

4.5.5. Roles

- Arquitecto
- Ingeniero de sistemas

4.6. Servidor de componentes distribuidos

4.6.1. Contexto

Los componentes distribuidos surgen porque se encuentran en distintos servidores debido a la división por capas de la aplicación y su instalación en la infraestructura de *hardware* en varios servidores. Pueden existir varios servidores Web y varios servidores de componentes o de aplicación. Los servidores Web tendrán las páginas de la aplicación y los servidores de aplicación los componentes de lógica de la aplicación y los componentes de acceso a datos.

4.6.2. Problema

¿Cómo están formados los servicios de componentes distribuidos? ¿Qué funciones tienen dentro de una aplicación distribuida?

4.6.3. Solución

Las funciones principales de los servicios de componentes distribuidos son proveer los medios necesarios para la comunicación entre componentes, la administración de los componentes y la integración con los demás servicios de la plataforma de *software*. Los servicios de componentes distribuidos están formados por los elementos mostrados en la Tabla XXVI.

Tabla XXVI. Servicios de componentes distribuidos

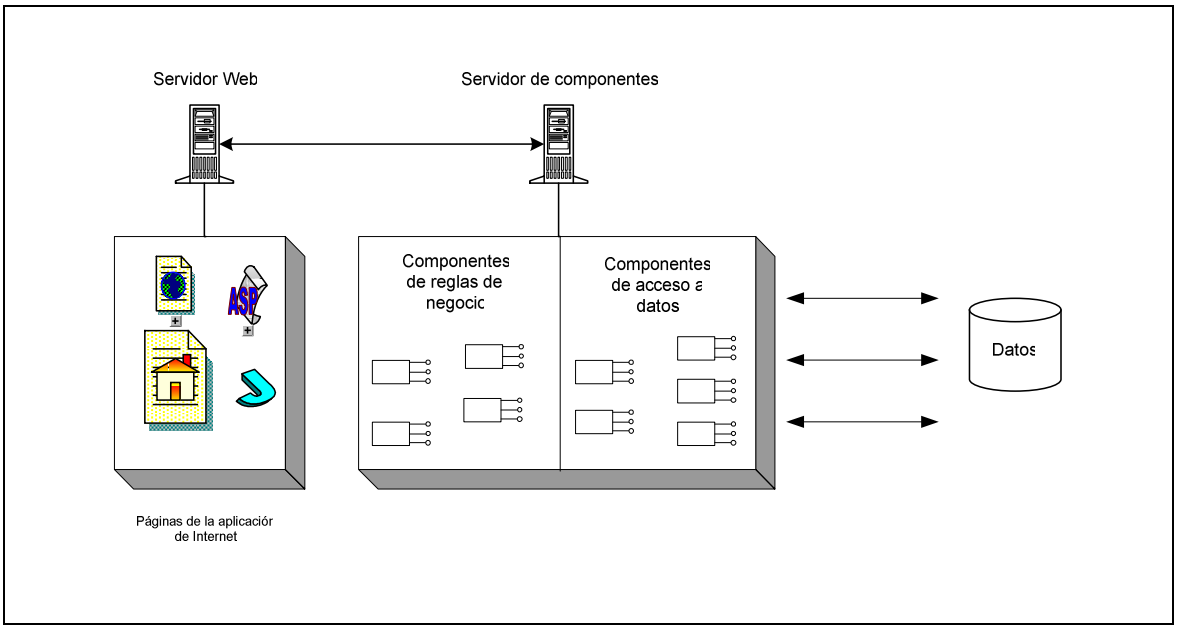
Administrador de solicitudes de componentes (ORB, *Object Request Brokering*)	Debe hacer posible la comunicación entre componentes a través de protocolos de comunicación establecidos.
Administrador de seguridad	Los componentes deben poder tener la opción de parametrizar sus niveles de seguridad.
Administrador de componentes	Deben permitir: crear, nombrar, almacenar, mover y eliminar los componentes. Además permiten el almacenamiento de información acerca de los componentes.
Monitores transaccionales de componentes	Es necesario entonces que existan monitores para guardar la integridad de la información, y coordinar transacciones en sistemas heterogéneos.
Interconexión con el servicio de páginas Web	Este servicio es vital para las aplicaciones distribuidas para Internet. Los componentes de alguna forma deben estar disponibles dentro de las páginas de la aplicación para Internet.

Cuando dos componentes se encuentran en distintos servidores físicos necesitan comunicación que es dada por los servicios de componentes distribuidos. Por ejemplo, si una página Web necesita ejecutar un componente de lógica de la aplicación, o si un componente de la lógica necesita comunicarse con un componente de acceso a datos en otro servidor es necesario tener un protocolo de comunicación. Se pueden dividir los protocolos en especializados y en los llamados Servicios Web.

Los servicios Web permiten la comunicación entre componentes utilizando XML y SOAP (*Simple Object Access Protocol*, Protocolo de Acceso Simple a Objetos) como protocolos principales haciendo que su implementación sea estándar. Los protocolos de comunicación especializados requieren mayor experiencia para su configuración y no son estándares entre distintas plataformas de *software*.

Los componentes pueden dividirse en dos grandes grupos: de lógica de la aplicación y de acceso a datos. Los primeros contienen las reglas del negocio y los segundos se encargan de acceder los datos almacenados que utilizará la aplicación. En la figura 16, puede observarse la funcionalidad de los componentes dentro de la plataforma de aplicaciones distribuidas.

Figura 16. Servidor de componentes distribuidos



4.6.4. Consecuencias

Una buena plataforma de *software* debe proveer los servicios para los componentes distribuidos ya que son fundamentales en este tipo de aplicaciones en el aspecto de escalabilidad y alta disponibilidad. Aplicando estos conceptos se estará creando una aplicación distribuida en la que los componentes serán independientes de su ubicación en la infraestructura de *hardware* o servidores.

4.6.5. Roles

- Arquitecto
- Desarrollador

4.7. Servidor de base de datos

4.7.1. Contexto

Los servicios de base de datos son los llamados DBMS (*Database Management Systems,* Administradores de Bases de datos) que dieron lugar a la arquitectura cliente / servidor de dos capas y que pueden manejar lógica de la aplicación. Proveen las herramientas para administrar las bases de datos y poder accederlas desde las aplicaciones.

4.7.2. Problema

¿Cuál es el rol que juegan las bases de datos en las aplicaciones distribuidas?

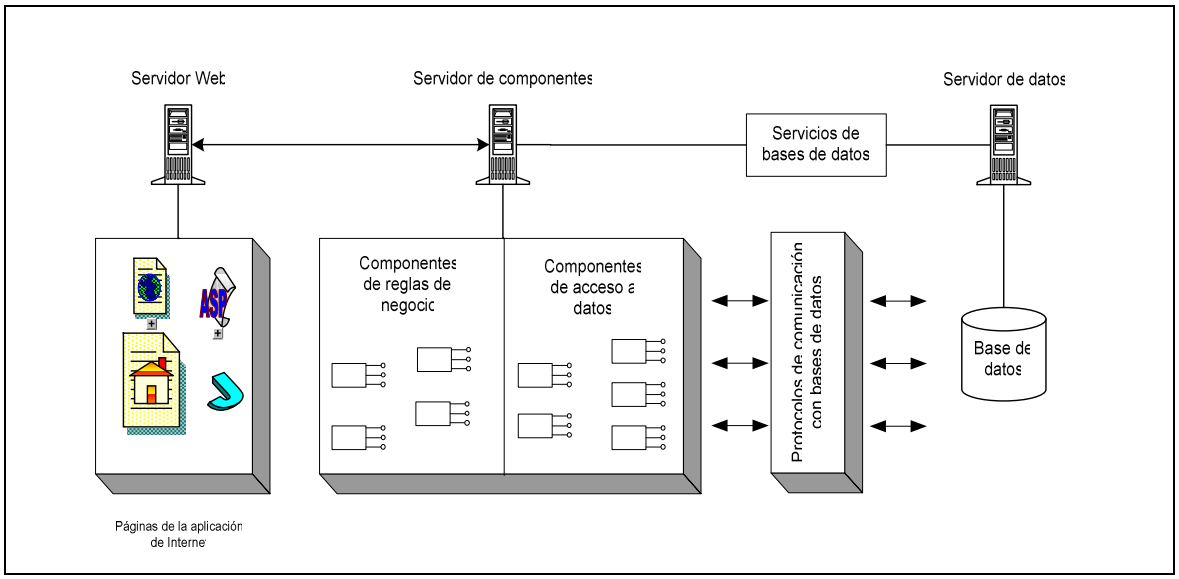
4.7.3. Solución

El servidor de base de datos debe tener los protocolos de comunicación para que los componentes de acceso a datos puedan comunicarse con la base de datos. No debe tener ninguna lógica de la aplicación sino solamente ser el repositorio de datos.

Uno de los propósitos de las aplicaciones distribuidas es hacerlas independientes de la base de datos encapsulando su acceso en los componentes de acceso a datos. Esto permite la independencia entre la capa de la lógica de la aplicación y la capa de datos que es una de las premisas para las aplicaciones divididas en capas.

El rol que tiene una base de datos es proveer acceso a la información almacenada a los componentes de acceso a datos, así como garantizar su integridad. Toda la lógica de la aplicación debe estar considerada en los componentes de lógica y de acceso a datos. La figura 17 muestra el lugar que ocupan los servicios de datos dentro de la plataforma.

Figura 17. Servidor de bases de datos



4.7.4. Consecuencias

La separación de la capa de datos de la lógica de la aplicación permitirá independencia del producto de base de datos que se estará utilizando y facilitará la escalabilidad de la aplicación en el almacenamiento de información. También no se tendrán licencias de acceso por cada cliente de la aplicación sino serán dadas a los componentes de acceso a datos permitiendo la posibilidad ahorrar costos.

La independencia de la base de datos y la aplicación facilitará: la migración y la reutilización de los componentes en distintos productos de base de datos ya que la lógica de la aplicación debería ser la misma y los componentes de acceso a datos solamente deben tener el protocolo de acceso a la base de datos.

4.7.5. Roles

- Arquitecto
- Ingeniero de sistemas

4.8. Servicios de conexión a sistemas heterogéneos

4.8.1. Contexto

Cuando se dice que dos sistemas son heterogéneos es por que tienen distinto sistema operativo, utilizan diferentes plataformas de *software* o están además construidos sobre una infraestructura de *hardware* con una tecnología diferente no compatible. Las aplicaciones Web distribuidas en N capas propician este escenario porque existen aplicaciones que son incompatibles con la plataforma de *software* pero existen los medios para tener comunicación y que estos sistemas formen parte de la capa de datos. A esto se le llama interoperabilidad entre sistemas.

4.8.2. Problema

¿Cuándo son necesarios los servidores de conexión a sistemas heterogéneos?

4.8.3. Solución

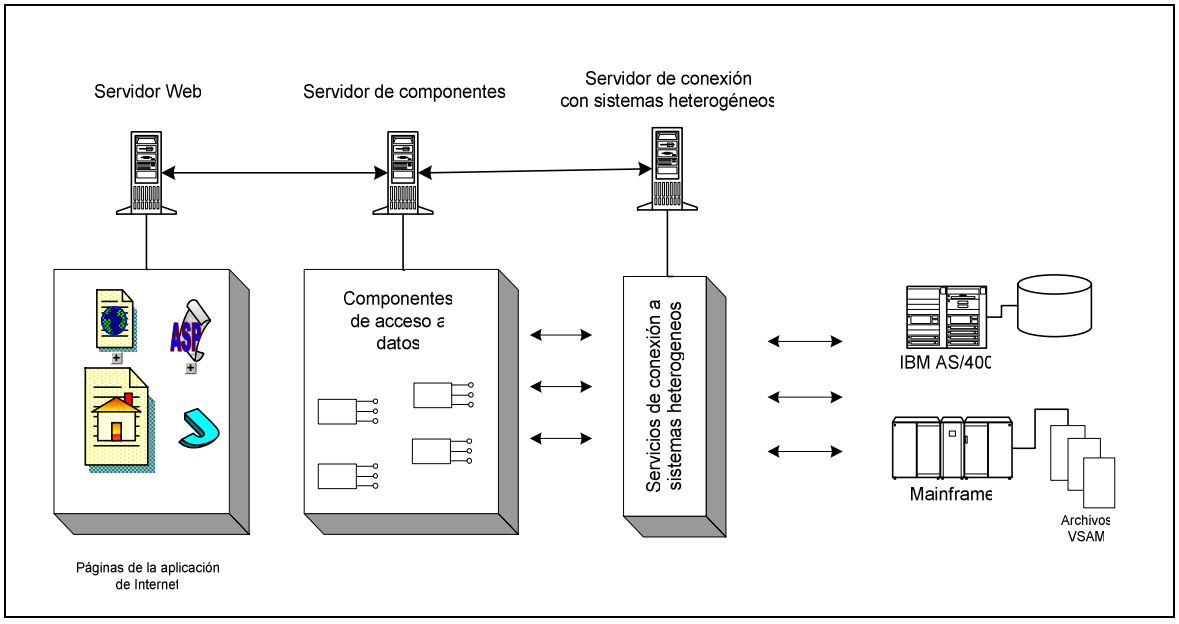
Son necesarios cuando se requiera acceder información que se encuentra almacenada en aplicaciones antiguas o que no puedan tener una comunicación directa por la plataforma de *software* utilizada, realizando una interoperabilidad entre sistemas.

Este tipo de servicios permite la integración entre varios sistemas heterogéneos produciendo una aplicación de N capas más compleja. Por ejemplo, si se toma el caso de una empresa que tiene un sistema *mainframe* antigua y se necesita una aplicación Web que tenga acceso a la información puede utilizarse el sistema antiguo como una fuente de datos.

La integración es una de las metas de las aplicaciones Web distribuidas y es una necesidad actual de las empresas poder integrar sus sistemas con otras empresas ya sea porque sean sus cliente, proveedores o socios.

En esencia los servicios de conexión a sistemas heterogéneos son protocolos de comunicación que permiten acceder la información guardada en ellos y que formen parte de la capa de datos. Esto puede verse en la figura 18.

Figura 18. Servicios de conexión a sistemas heterogéneos



4.8.4. Consecuencias

Al tener conexión con sistemas heterogéneos permiten acceder a la información de aplicaciones convirtiéndolas en parte de la capa de datos.

4.8.5. Roles

- Arquitecto
- Integrador de sistemas
- Ingeniero de sistemas

4.9. Servidores de colas de mensajes

4.9.1. Contexto

En la integración de aplicaciones las colas de mensajes permiten la comunicación entre sistemas de una forma asíncrona. Esto quiere decir que si una aplicación emisora de solicitudes envía un mensaje a una aplicación no se tendrá una respuesta en esa misma llamada sino la aplicación receptora envía la respuesta por otro canal en un tiempo no determinado.

4.9.2. Problema

¿Cuándo deben utilizarse servidores de colas de mensajes?

4.9.3. Solución

Los servicios de colas de mensajes permiten una comunicación asíncrona entre aplicaciones y también a sistemas heterogéneos. Estos servicios son muy importantes en el ambiente de aplicaciones distribuidas y en la integración de aplicaciones. Como los servicios deben ser independientes de la plataforma los servicios de colas de mensajes deben de poder integrar sistemas de distintas plataformas y establecer un canal de comunicación.

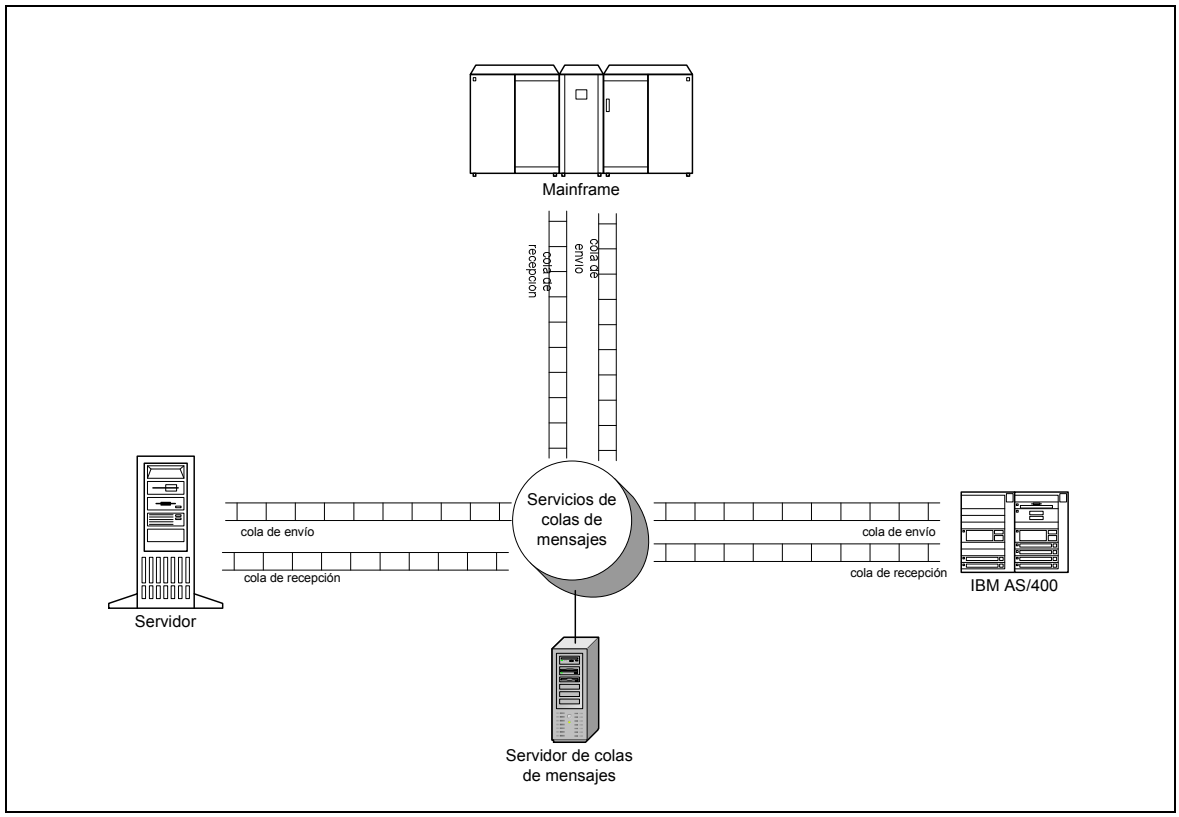
El objetivo de los servicios de colas de mensajes es que una cola sea el canal que permita la comunicación entre aplicaciones. Por ejemplo, las solicitudes o instrucciones que se necesitan sean atendidas por parte de otra aplicación se hace enviando mensajes en una cola y se reciben las respuestas en otra; para esto deben de definirse las colas en común entre las dos aplicaciones.

Estos servicios permiten que las aplicaciones puedan tolerar cierto nivel de espera de respuestas a solicitudes realizadas, contrario al modo síncrono de solicitudes. También permiten que la aplicación trabaje fuera de línea acumulando mensajes en las colas. Esto último hace posible que clientes nómadas realicen transacciones desconectados del servidor y cuando se realice la conexión todos los mensajes “encolados” se transmitan.

El proceso de los servicios de colas de mensajes para enviar mensajes a través de la red es poner el mensaje en una cola y otra aplicación obtenga mensajes de esa cola. La mayoría de productos de servicios de colas de mensajes proveen una interfaz o componentes para que pueda ser utilizada en múltiples sistemas operativos y herramientas de desarrollo.

También estos servicios incluyen niveles de tolerancia a fallas como mensajes persistentes que son almacenados en una base de datos y aseguran la llegada del mensaje haciéndolo una unidad transaccional. Los servicios de colas proveen una forma más fácil de conectar aplicaciones entre empresas y entre aplicaciones porque establecen una comunicación estándar independiente de las plataformas utilizando un protocolo de red estándar. Puede verse en la figura 19 como encajan dentro de la plataforma.

Figura 19. Servicios de colas de mensajes



4.9.3.1 Consecuencias

Las colas de mensajes permiten una comunicación asíncrona en las que existirá una cola de envío y otra cola de respuesta de solicitudes para cada

aplicación conectada. En aplicaciones complejas cuando se comunican varias aplicaciones es necesario un coordinados de mensajes para que pueda enviar los mensajes a la aplicación que corresponda.

4.9.4. Roles

- Arquitecto
- Integrador de sistemas
- Ingeniero de sistemas

4.10. Servicios de colaboración

4.10.1. Contexto

Existen aplicaciones implementadas y especializadas que se especializan en la administración de flujo de procesos. Son un conjunto de tecnologías que permiten implementar los procesos involucrados alrededor de actividades colaborativas en grupo. Estas tecnologías pueden separarse en: administración de documentos, *workflow* (flujo de trabajo), correo electrónico, y planificación de grupos de trabajo.

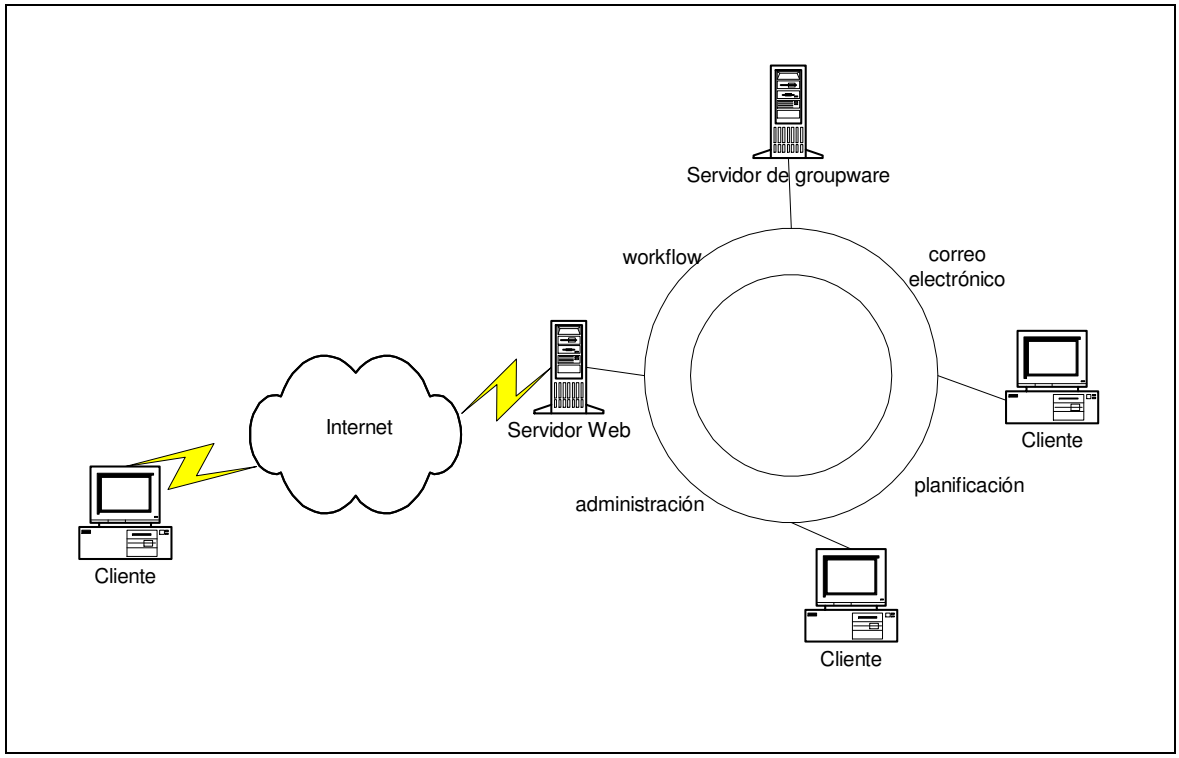
4.10.2. Problema

¿Cuándo integrar los servicios de colaboración para las aplicaciones Web?

4.10.3. Solución

Estos servicios se integran a las aplicaciones distribuidas por medio de componentes con interfaces para que puedan ser utilizados desde una aplicación. Se deben utilizar cuando se requiera reutilizar componentes implementados por lo servicios de colaboración. Uno de los servicios más importantes y utilizados es el de correo electrónico. En la figura 20 puede verse un esquema para los servicios de colaboración.

Figura 20. Servicios de colaboración



4.10.4. Consecuencias

Al utilizar los servicios de colaboración debe comprobarse la compatibilidad en las plataformas de *software* a utilizar y se debe de estar conciente de la especialidad que tienen las herramientas de colaboración, y no utilizarlas como un medio de comunicación entre aplicaciones.

4.10.5. Roles

- Arquitecto
- Ingeniero de sistemas

4.11. Capas de la aplicación

4.11.1. Contexto

Las aplicaciones Web se dividen en varias capas de componentes especializados para cumplir cierta funcionalidad. Cuando se desarrollan aplicaciones es importante identificar la función de cada componente para producir una aplicación bien estructurada que permita obtener los beneficios de la plataforma de aplicaciones distribuidas.

4.11.2. Problema

¿Cómo se dividen los componentes en las capas de una aplicación?

4.11.3. Solución

La capa de presentación está formada por las páginas Web, la lógica de la aplicación por componentes de *software* y la capa de datos por algún sistema heredado, un sistema de bases de datos o cualquier otra fuente de datos. Estos pueden distribuirse en la red sin ningún problema gracias a la capa de servicios de aplicaciones distribuidas. Para poder entender bien esto se debe partir de que toda aplicación consta de tres capas principales:

- Capa de presentación
- Capa de la lógica de la aplicación o reglas del negocio
- Capa de datos

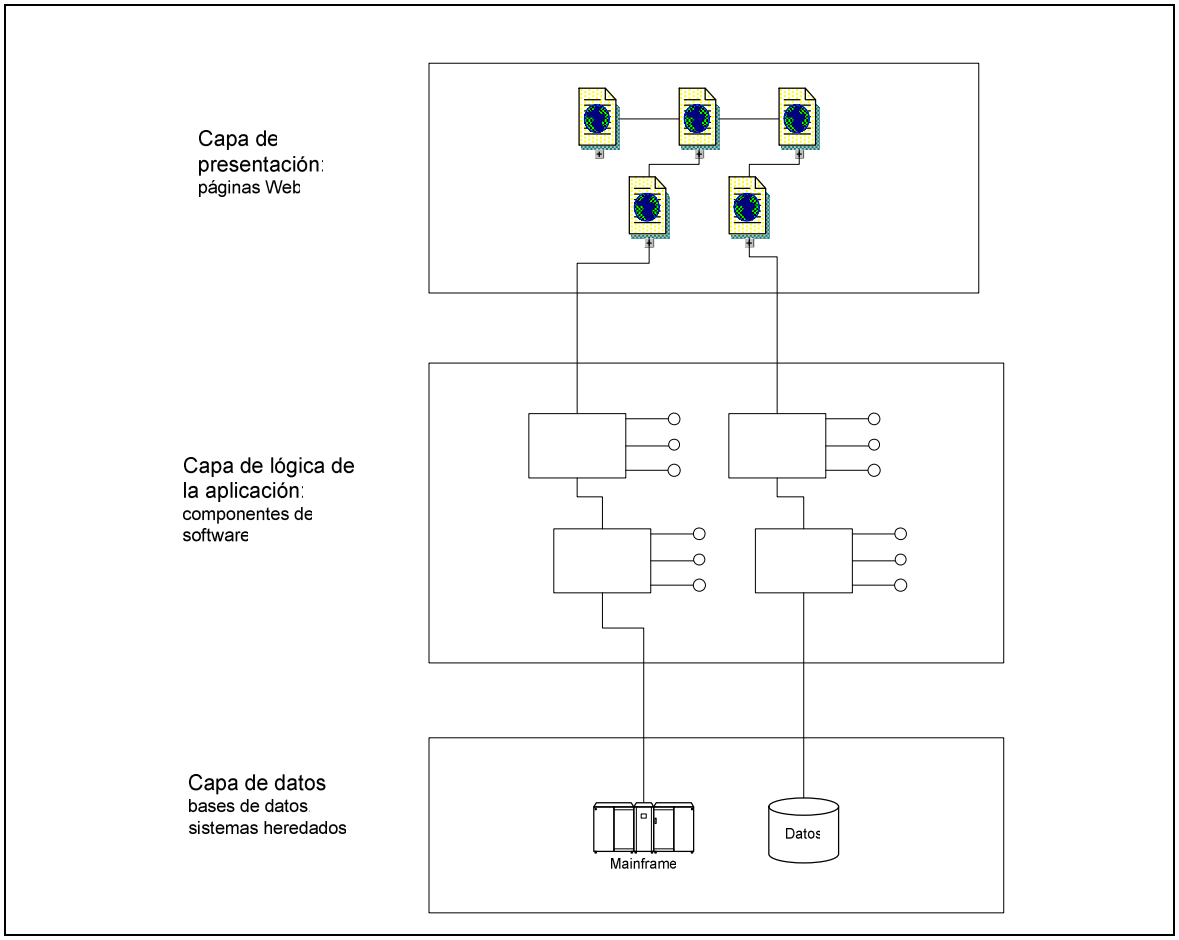
La capa de presentación debe ser independiente de los otros dos elementos y se muestra por medio de un navegador. La capa de lógica de la aplicación incluye los nuevos conceptos de las aplicaciones distribuidas dividiéndose en varias capas que prestan servicios específicos y que agrupan un conjunto de componentes. La capa de datos es accedido a través del elemento de lógica de la aplicación, el cual también debe ser independiente y el único servicio que presta es brindar acceso a los datos que utiliza la aplicación.

Cada capa está compuesta por componentes de software que interactúan entre sí a través de los servicios de aplicaciones distribuidas. Los componentes de la capa de lógica de la aplicación pueden estar separados tanto lógica como físicamente en varios servidores. El número de capas de la aplicación depende de la arquitectura de servicios que se utilicen como plataforma y de la naturaleza de la aplicación. En la figura 21 puede observarse como se dividen las capas.

4.11.3.1 La capa de datos

La capa de datos debe encargarse de proveer acceso a la información a la capa de la lógica de la aplicación independientemente de la tecnología utilizada para el almacenamiento. Puede estar formada por alguno de los siguientes elementos.

Figura 21. Capas de una aplicación Web



a. **Bases de datos: l**as bases de datos actualmente ofrecen implementaciones de seguridad y servicios de acceso a datos. Con la arquitectura cliente / servidor han tenido gran participación y muchas se están integrando para facilitar su acceso a través de aplicaciones Web. En el diseño de la arquitectura

de aplicaciones Web distribuidas ya no deben manejar ningún tipo de lógica que pueden implementarse en *triggers* o procedimientos almacenados.

b. ***Mainframes*:** estos son grandes sistemas capaces de almacenar una gran cantidad de información. En la actualidad son empresas bastante grandes los que utilizan esta arquitectura. La desventaja que tienen estos sistemas es que son propietarios. Se han creado varios servicios para el acceso de los datos a estos sistemas convirtiéndolos en fuente de datos especialmente para el acceso a la información por medio de aplicaciones Web.

c. **Sistemas heredados:** estos son sistemas antiguos que las empresas manejan y sólo servirán como fuente de información para las aplicaciones Web.

4.11.3.2 La capa de la lógica de la aplicación

Esta capa es donde se encuentran toda la lógica de la aplicación. Tiene la función de atender las solicitudes que se realizan a través de la capa de presentación, encapsular las reglas del negocio, definir el flujo del proceso del negocio y comunicar las solicitudes a la capa de datos.

Existen varios tipos de componentes distribuidos en esta capa según la función que desempeñen: procesos del negocio, reglas del negocio, administrar transacciones con el acceso a datos, formato de datos, servicios de conectividad con dispositivos y otros.

4.11.3.3 La capa de presentación

La forman las páginas Web que el usuario podrá ver y realizar solicitudes al Servidor Web por medio del navegador para Internet

4.11.4. Consecuencias

Cuando se identifican y agrupan cada uno de los componentes correctamente en las capas de la aplicación que corresponden según su funcionalidad permitirá tener aplicaciones que serán fáciles de mantener y aprovecharán todo el contexto de la plataforma de aplicaciones distribuidas.

4.11.5. Roles

- Arquitecto
- Desarrollador

5. EJEMPLOS DE PATRONES DE DISEÑO

Como se introdujo capítulo 3 los patrones de diseño se enfocan al funcionamiento de la aplicación: los componentes que la forman y la interacción entre ellos. A continuación se presentan algunos patrones orientados a las aplicaciones Web.

5.1. Aplicación Web con páginas estáticas

5.1.1. Contexto

Cuando las aplicaciones Web solamente solicitan páginas que no involucran ningún proceso adicional de lógica o acceso a datos se les llama páginas estáticas. Estas aplicaciones sirven cuando solamente se desea tener un sitio con información, multimedia y navegación entre páginas. Por lo regular forman los portales de sitios en Internet donde se presenta información que puede ser accedida por cualquier usuario de Internet.

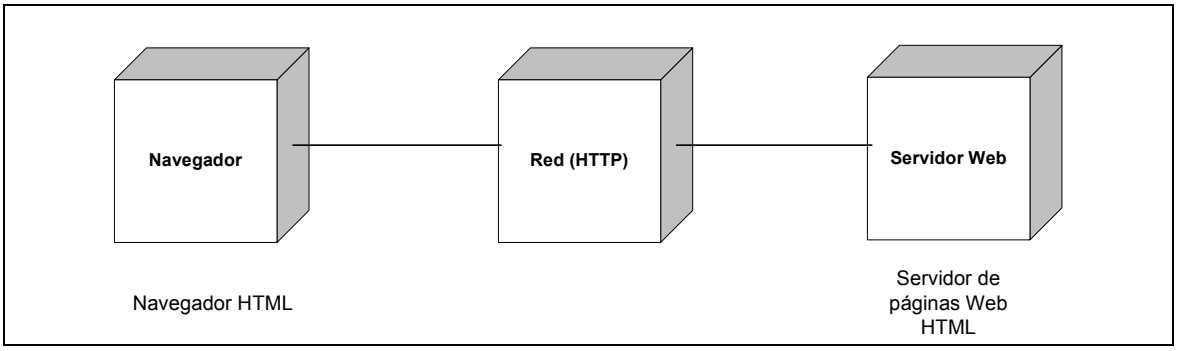
5.1.2. Problema

¿Qué elementos tiene una aplicación Web básica? ¿Para qué sirven las aplicaciones Web con páginas estáticas?

5.1.3. Solución

Los elementos de una aplicación Web básica son el navegador, el protocolo de comunicación HTTP a través de la red y el servidor Web. El servidor Web almacena las páginas que pueden ser solicitadas por el navegador y para ser desplegadas. Los elementos pueden verse en la figura 22.

Figura 22. Esquema de elementos de una aplicación Web básica



Se les llama páginas estáticas porque el contenido de la página siempre será el mismo. Estas aplicaciones no tienen componetes de capa de lógica de la aplicación y capa de datos sino solamente tienen la presentación que es definida por el HTML. El HTML es un lenguaje que especifica como un documento debe desplegarse en el navegador. Da características a la presentación de documentos como: color, tamaño, tipo de letra, etc. y además permite enlaces de navegación a otros documentos o páginas, despliegue de imágenes y utilización de multimedia.

Con páginas estáticas realmente se solicitan archivos que tienen un formato HTML y son transmitidos al navegador para que los despliegue. La única tarea del servidor es encontrar el archivo asociado a la página solicitada y enviarlo de respuesta.

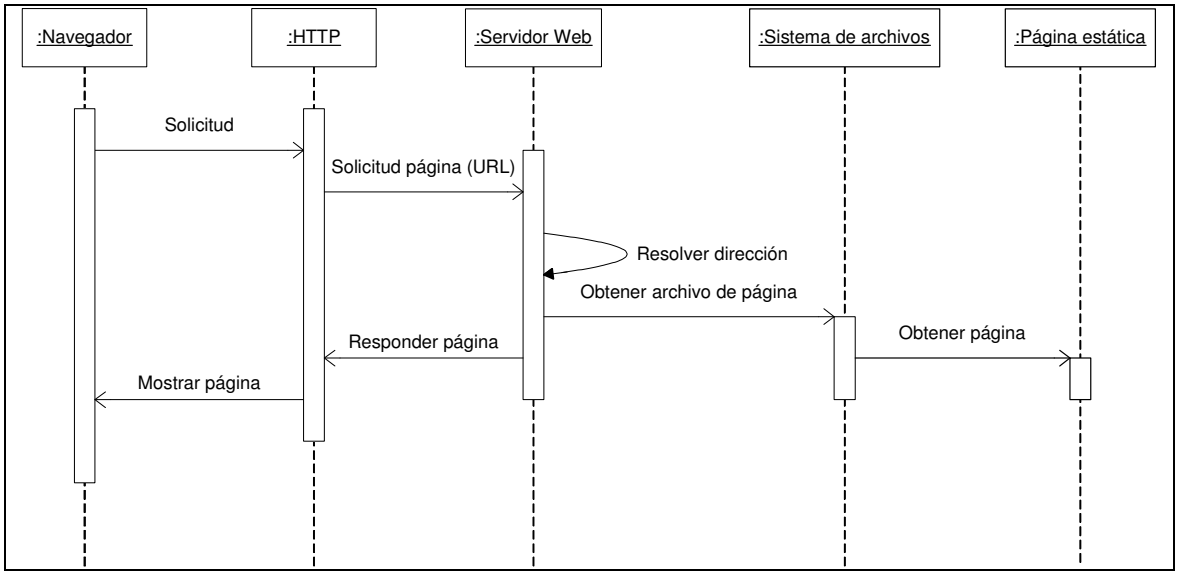
5.1.4. Consecuencias

Las aplicaciones Web con páginas estáticas no se tendrá procesamiento sino solamente la devolución de una página HTML. No tienen acceso a datos y ninguna lógica de aplicación en el servidor Web. Es recomendable utilizar estas aplicaciones cuando solamente se desea publicar contenido estático en un ambiente de navegación entre páginas.

5.1.5. Esquemas

La figura 23 muestra como es procesada la solicitud de una página estática a un servidor Web. El navegador por medio del protocolo HTTP realiza la solicitud al Servidor Web, este hace una correspondencia entre la dirección de página enviada en URL y el archivo físico de la página, luego cuando es encontrada la página se envía al navegador.

Figura 23. Diagrama de secuencia de la solicitud de una página Web estática



5.1.6. Roles

- Arquitecto
- Diseñador
- Desarrollador

5.2. Aplicación Web con páginas dinámicas

5.2.1. Contexto

Cuando se requiere procesamiento de lógica y acceso a datos es necesario utilizar las páginas dinámicas. La lógica se puede encontrar incrustada dentro del HTML o en módulos compilados. Estas páginas requieren un proceso en el servidor Web en el que ejecutan la lógica de la página y generan una página con formato HTML lo cual permite la independencia entre el navegador y el servidor Web

5.2.2. Problema

¿Cómo desarrollar una página dinámica?

5.2.3. Solución

Existen tres formas de implementar páginas dinámicas, las cuales se describen a continuación.

5.2.3.1 Páginas con *scripts* interpretados

Un *script* de página Web es un conjunto de instrucciones escritas en un lenguaje de programación encerrado dentro de etiquetas especiales en HTML que ejecutan la lógica y acceso a datos. Los *scripts* son interpretados y ejecutados por el Servidor Web al momento de solicitar la página y dan como resultado una página HTML.

El desarrollo de las páginas con *scripts* interpretados es fácil, pero cuando la lógica a manejar es compleja el resultado es una página con un aspecto desordenado y difícil de darle mantenimiento. Otro aspecto que debe tomarse en cuenta es que las páginas son más lentas en su ejecución porque el Servidor Web debe interpretar y ejecutar cada línea del *script* que se encuentra en la página

5.2.3.2 Módulos compilados

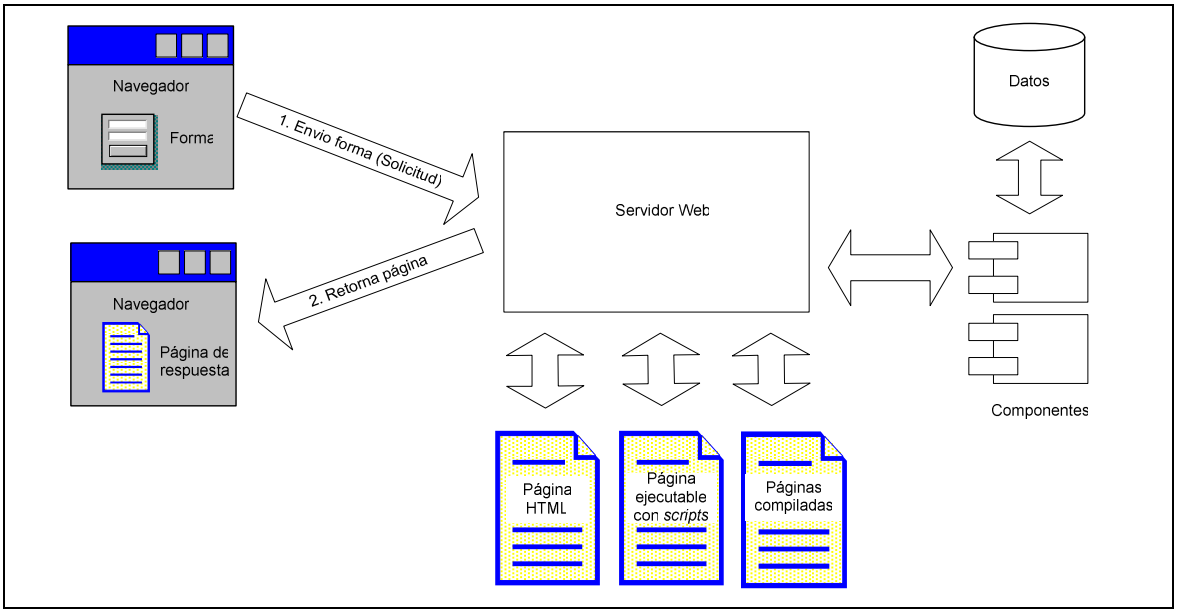
Los módulos compilados son cargados desde archivos binarios y ejecutados por el Servidor Web. Obtienen la información enviada desde el navegador por la página para procesarla y generar una página HTML que es enviada de vuelta al navegador. Tienen un mejor rendimiento porque ya se encuentran listos para ser ejecutados. La desventaja es que dentro del módulo se mezcla la lógica y presentación ocasionando el mismo problema que las páginas con *scripts*.

5.2.3.3 Páginas compiladas

Las páginas con *scripts* interpretados tienen incrustado un lenguaje de programación dentro del HTML, y los módulos compilados un lenguaje de

programación que genera HTML. Lo mejor es tener separada la presentación de la lógica de la aplicación y que ambas se encuentren compiladas. Un modelo general para una aplicación Web con páginas dinámicas quedaría como el mostrado en la figura 24.

Figura 24. Elementos de una Aplicación Web con páginas dinámicas

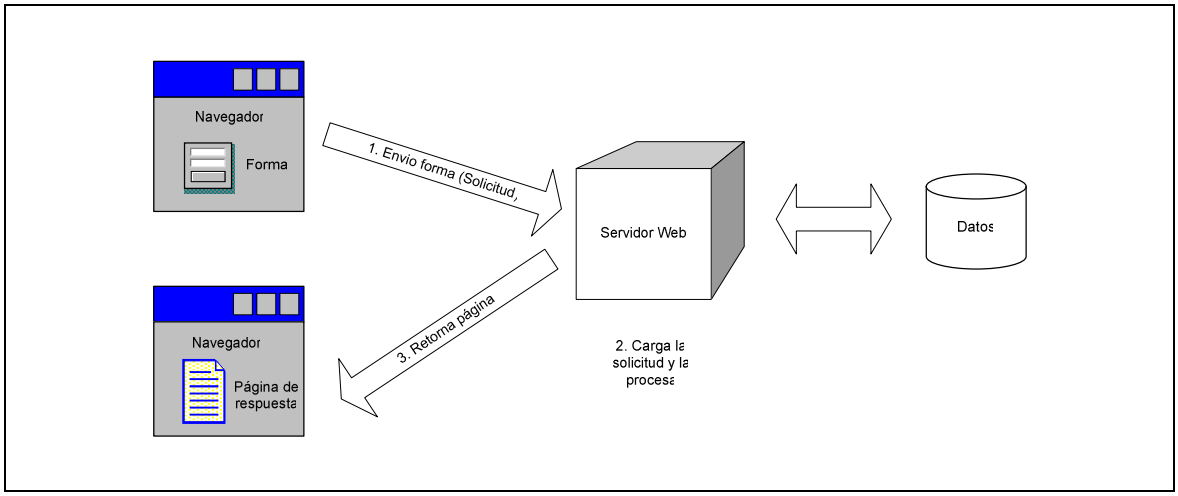


Para poder capturar la información ingresada por el usuario se utilizan formas. Una forma es el conjunto de campos que permiten a los usuarios ingresar y cambiar información. Cuando se envía el contenido de la forma al Servidor Web es procesada, y genera una respuesta. Las aplicaciones Web tienen páginas ejecutables que se procesan en el servidor como se muestra en la figura 25.

Las aplicaciones Web utilizan tecnologías para hacer su contenido dinámico y tener un enlace con la lógica de la aplicación. El interactuar con las reglas del negocio es lo que la hace una aplicación y la distingue de un sitio Web

informativo. Esto lo hace principalmente por medio de formas y páginas ejecutables que son ejecutadas por un servidor de aplicaciones.

Figura 25. Página Web ejecutable procesando una forma de entrada de datos



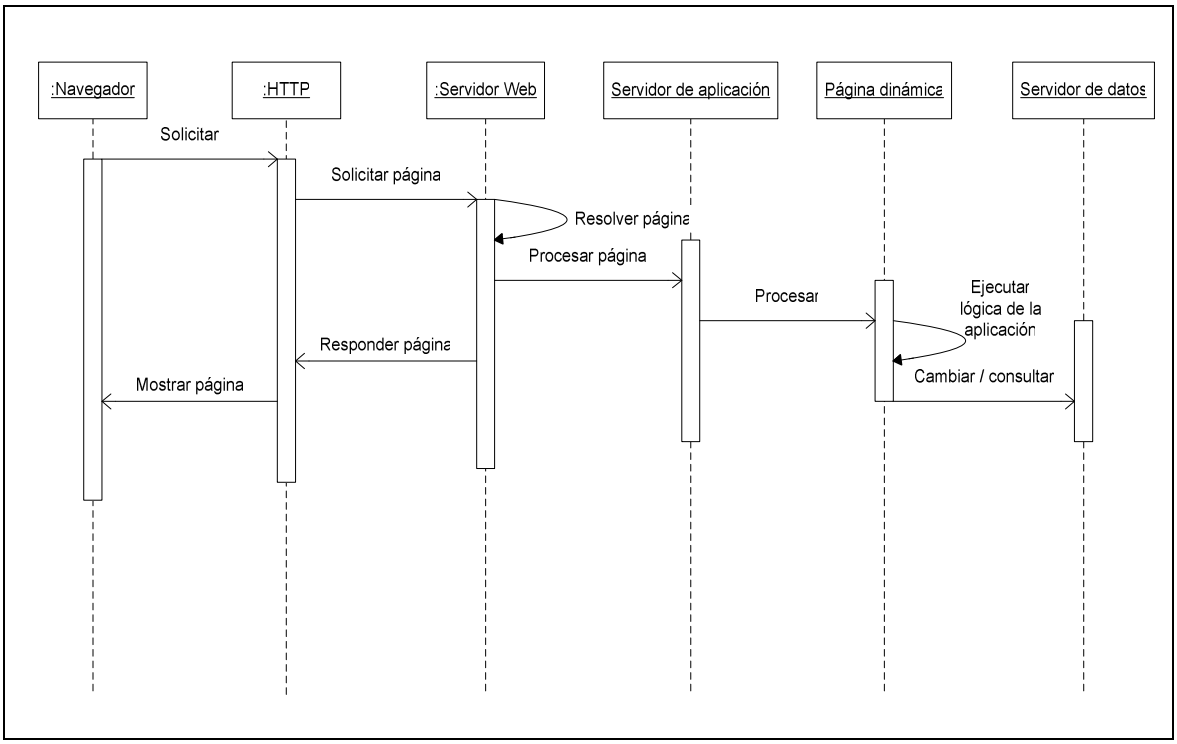
5.2.4. Consecuencias

La utilización de páginas dinámicas permite el procesamiento de lógica de aplicación en el servidor. Debe procurarse utilizar módulos o páginas compiladas porque las páginas con *scripts* interpretados no son tan eficientes y cuando una aplicación es accedida por gran cantidad de usuarios puede afectar el rendimiento. El mejor esquema es que las páginas ejecuten componentes compilados para procesar la lógica y las páginas tener como principal función mostrar e ingresar información por formas Web.

5.2.5. Esquemas

El procesamiento de una página Web dinámica puede verse en la figura 26.

Figura 26. Diagrama de secuencia de la solicitud de una página Web dinámica



5.2.6. Roles

- Arquitecto
- Diseñador
- Desarrollador

5.3. Aplicaciones distribuidas

5.3.1. Contexto

Las capas de una aplicación están formadas por componentes. Si se desea tener una aplicación con bastante carga y que tenga buen rendimiento es mejor especializar cada uno de los componentes asignándoles funciones según la capa a la que pertenezcan. Dentro del contexto de las aplicaciones distribuidas las páginas Web son consideradas también componentes de presentación las cuales invocarán a los componentes de lógica de la aplicación.

Un componente de lógica de la aplicación a su vez podrá ejecutar componentes de acceso a datos para realizar consultas o actualizaciones de información. Cuando se terminan de ejecutar los componentes envían un resultado al servidor Web el cual construye la página de respuesta al navegador.

5.3.2. Problema

¿Cómo debe construirse una aplicación Web distribuida?

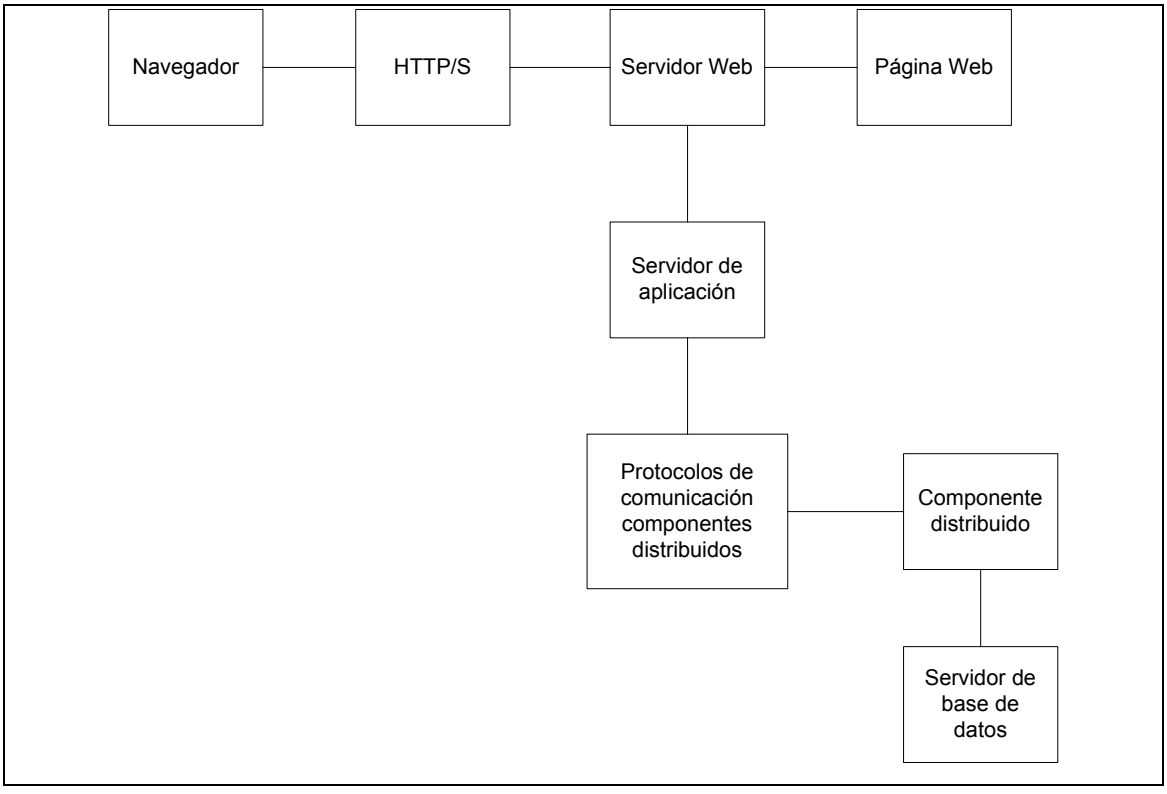
5.3.3. Solución

Para construir una aplicación distribuida deben separarse bien las capas de los componentes. Desde el navegador se debe solicitar una página dinámica al Servidor Web. La página debe ser lo más liviana posible, enfocándose a tareas

solamente de presentación e ingreso de datos, y a la llamada de componentes que tendrán la mayoría de la lógica de la aplicación.

Los componentes de lógica de la aplicación están ubicados en el servidor de la aplicación. Se comunican con el Servidor Web y entre sí por un protocolo de aplicaciones distribuidas (DCOM, CORBA, Java RMI) o por medio de Servicios Web. El objetivo de estos protocolos es permitir un acceso eficiente a componentes a través de la red porque pueden encontrarse en servidores físicamente diferentes. Los principales participantes de una aplicación Web distribuida se muestran en la figura 27.

Figura 27. Principales participantes en una aplicación distribuida



5.3.4. Consecuencias

El utilizar aplicaciones distribuidas los componentes estarán divididos en capas haciéndolos independientes y hacer posible distribuirlos en la red en varios servidores. Debe utilizarse un protocolo para comunicaición de componentes distribuidos el cual debe evaularse para su mejor desempeño en la red y en cuestiones de seguridad.

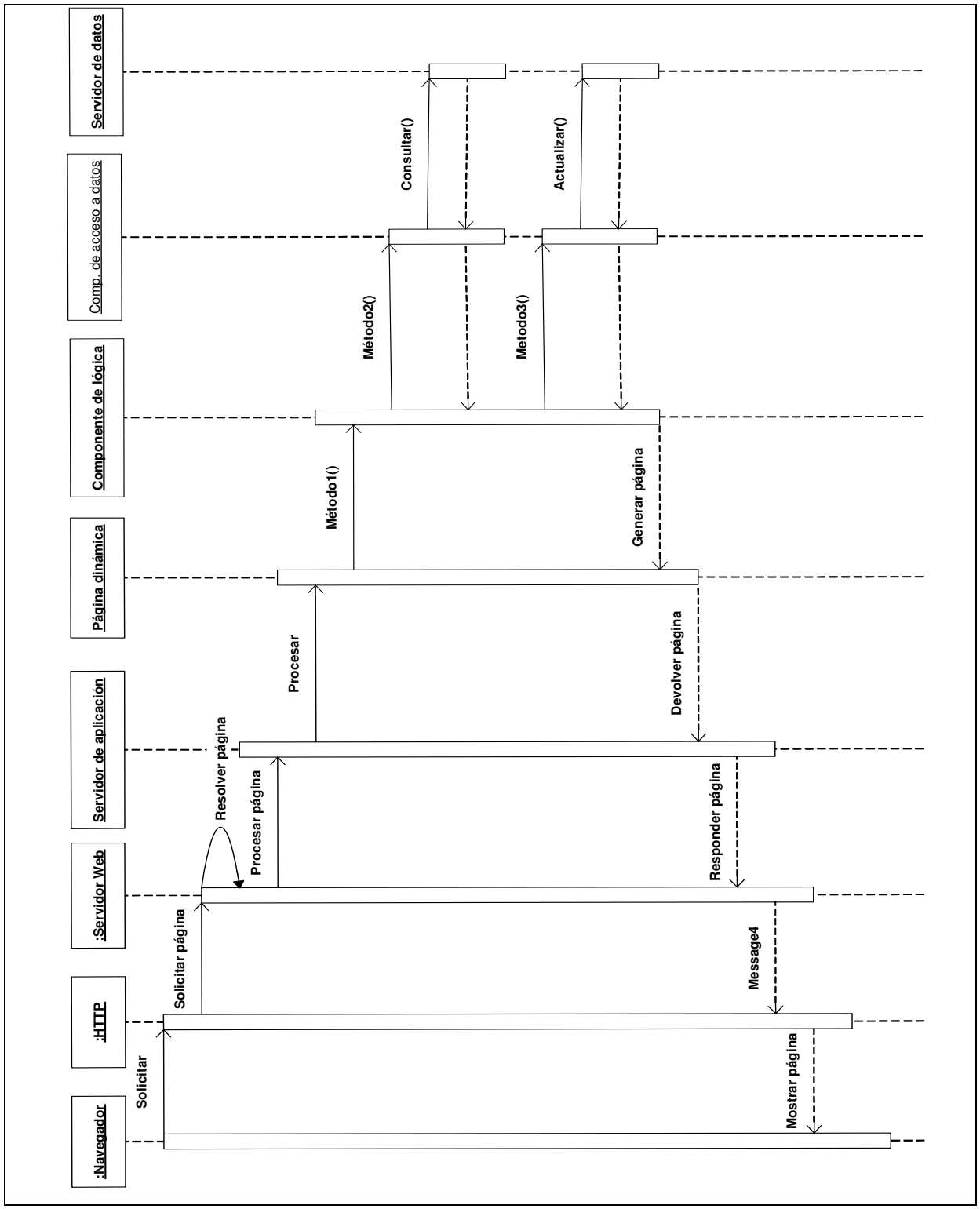
5.3.5. Esquemas

El procesamiento de una página Web que tiene llamada a componentes puede observarse en el diagrama de secuencias de la figura 28. Desde el navegador se solicita una página al Servidor Web. Dentro de la página se encuentra la llamada a un componente de lógica de la aplicación el cual utiliza componentes de acceso a datos realizando operaciones de actualización y consulta que devuelven resultados al componente de lógica, y éste a su vez a la página dinámica (que es considerada como otro componente). Luego esta devuelve el resultado al servidor de aplicación y este al servidor Web el cual construye la página de respuesta al navegador.

5.3.6. Roles

- Arquitecto
- Diseñador
- Desarrollador

Figura 28. Ejecución de una aplicación distribuida



5.4. Administración del estado

5.4.1. Contexto

Una de las principales dificultades en las aplicaciones Web es la administración del estado en el cliente, porque la conexión entre el cliente y el servidor Web no es continua. El estado es la persistencia de atributos de una conexión o sesión de un navegador.

5.4.2. Problema

¿Cómo administrar el estado en una aplicación Web?

5.4.3. Solución

El propósito de la administración del estado es hacer persistentes objetos o información manejada en la sesión de un navegador a una aplicación Web. Cuando el navegador se conecta a una aplicación Web se establece un canal de comunicación asíncrono y empieza una sesión en la que el servidor Web reconoce a cada uno de los navegadores que está realizando solicitudes. A continuación se presentan las formas más comunes de administrar el estado.

5.4.3.1 *Cookies*

Es una parte de información que un servidor Web puede solicitar al navegador almacene y la envie al servidor cada vez que el navegador hace

una solicitud posterior. Esta es una solución aceptada pero tiene sus inconvenientes en el área de la privacidad del usuario.

5.4.3.2 Parámetros en dirección de URL

Es otra forma de administrar la sesión. Consiste en enviar el estado en parámetros en la dirección URL de la página, para que sean recuperados en la siguiente página. Si estos se necesitarán en otra página deben enviarse nuevamente, si no se envían entonces se pierde el estado.

5.4.3.3 Sesiones

Una sesión está más ligada a la utilización de la aplicación Web por el usuario, permite que el estado se mantenga mientras el usuario interactúa con la aplicación durante un tiempo definido. Esta es utilizada cuando la aplicación introduce muchas páginas ejecutables, con muchas reglas del negocio y que se necesita mantener el estado del cliente a lo largo de toda la navegación de páginas. Este tipo es implementado por la mayoría de Servidores Web que administran páginas ejecutables, y puede ser almacenado de forma similar a las *cookies* o como parámetros de URL. Son llamadas “variables de sesión”.

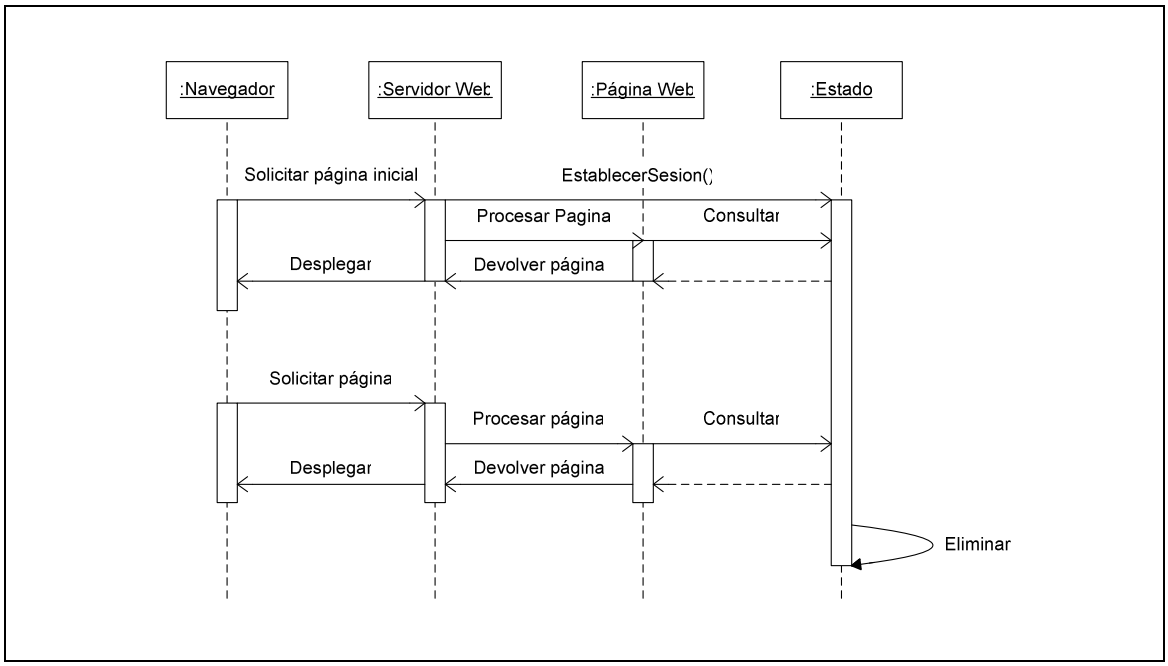
5.4.4. Consecuencias

La utilización de variables de sesión implica utilizar recursos de memoria en el servidor para que pueda recordarse el estado de la aplicación en cualquier momento. También debe considerarse el ciclo de vida o expiración del manejo del estado para que cuando el navegador se desconecte de la aplicación.

5.4.5. Esquema

En la figura 29 puede observarse como el objeto “Sesión” es mantenido durante la navegación del usuario de la aplicación a través de las páginas.

Figura 29. Administración del estado en una aplicación Web



5.4.6. Roles

- Diseñador
- Desarrollador

5.5. Lógica en los navegadores

5.5.1. Contexto

La evolución del HTML ha incorporando nuevas funcionalidades permitiendo que exista lógica en la página Web presentada por el navegador, pero también trae consigo la desventaja de la compatibilidad del HTML entre varios navegadores, y perder con ello la posibilidad de tener un cliente universal.

Debe tenerse claro que aunque el navegador posea cierta lógica es diferente a la lógica de la aplicación. Básicamente la lógica son validaciones de la información ingresada en formas, por ejemplo: datos de ingreso obligatorios, rangos de valores válidos, etc., y manipulación del contenido de la interfaz del usuario para hacerla más dinámica y evitar hacer solicitudes hasta el servidor.

5.5.2. Problema

¿Cómo se maneja la lógica en los navegadores?

5.5.3. Solución

Las tecnologías utilizadas para implementar la lógica en los navegadores, principalmente puede dividirse en dos grupos:

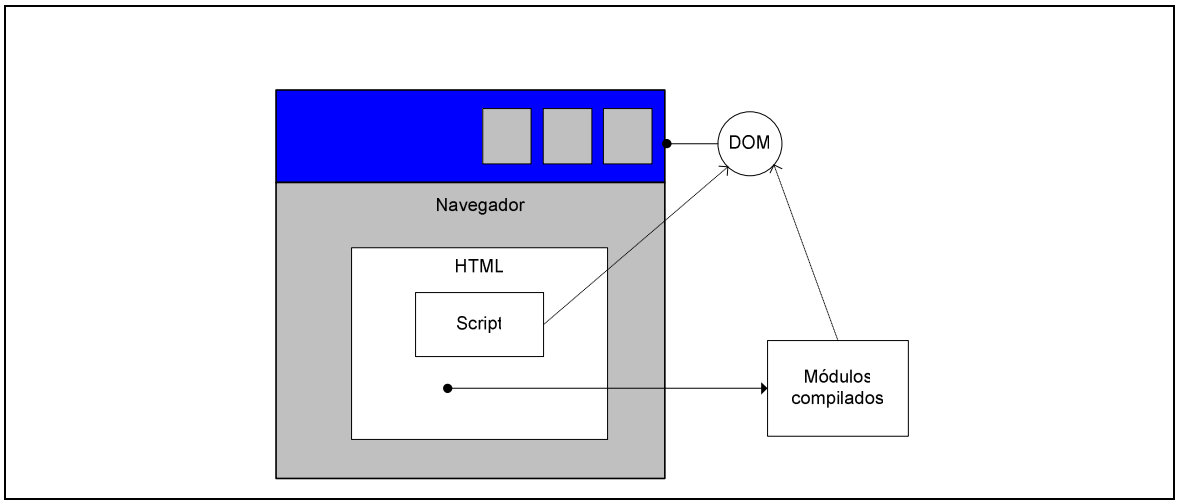
- Implementadas con *scripts*
- Componentes compilados

Estas tecnologías tienen las siguientes características:

- Están asociados a una página Web y pueden acceder y modificar su contenido.
- Automáticamente se instalan en el navegador cuando se necesitan ser utilizados

La lógica en los navegadores debe tomarse en cuenta en el diseño de la interfaz. En el navegador existe el objeto DOM (*Document Object Model*, Modelo de Objeto de Documento) que permite la lógica en los navegadores y se presenta en la figura 30. Según la versión que maneje el navegador así será la funcionalidad que tenga.

Figura 30. Objeto DOM del navegador



La lógica en un navegador puede ser implementada a través de *scripts* en el navegador con objetos de *Java Script* y eventos de HTML utilizando al objeto DOM. También existen tecnologías especiales como componentes ActiveX y Java Applets que son utilizadas cuando se necesita una lógica más sofisticada. La descripción de cada uno de ellos puede verse en las siguientes tablas.

Tabla XXVII. Implementación de la lógica en los navegadores

Scripts en navegador	La tecnología más común para este tipo es Java Script, basado en el lenguaje Java. Es soportado por la mayoría de navegadores.
Objetos de JavaScript	Basados en la estructura de documentos llamada DOM (*Document Object Model,* Modelo de Objetos de Documento) de HTML.
Eventos HTML	También son basados en el DOM de HTML.

Tabla XXVIII. Implementación de la lógica en los navegadores con tecnologías especiales

Controles ActiveX	Componentes construidos con lenguajes de programación Microsoft. Solamente pueden ser ejecutados en sistemas Windows.
Java Applets	Construidos con Java. El navegador debe tener la característica de poder ejecutarlos.

La principal razón de la lógica en el navegador es hacer el mínimo de solicitudes al Servidor Web. Esto se logra por medio de verificaciones y validaciones en las formas de ingreso de datos en el navegador para evitar que los datos sean enviados al Servidor Web y la respuesta se demore.

5.5.4. Consecuencias

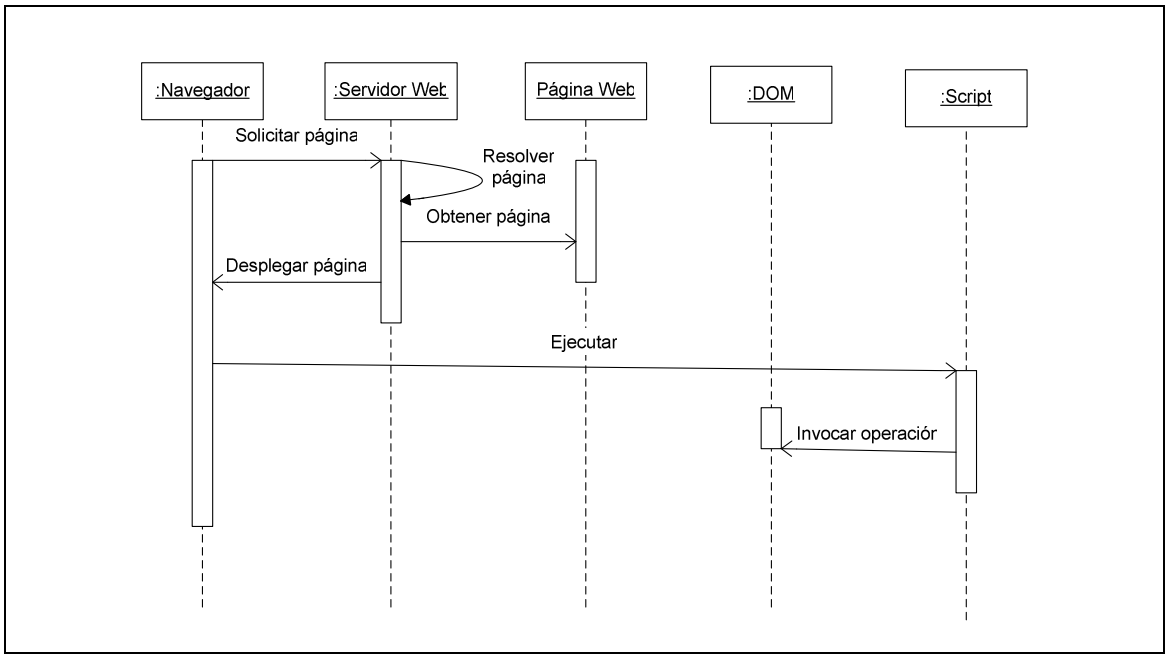
Cuando se utilizan validaciones en el navegador debe tomarse en cuenta que tipo de navegador utilizarán los usuarios. Puede utilizarse funcionalidad que se encuentre en la mayoría de navegadores o adaptarla para cada uno de los navegadores que soportará la aplicación.

Cuando la aplicación se encuentra en una intranet puede restringirse más fácilmente el uso de determinado navegador, pero si la aplicación estará disponible en Internet la utilización de lógica en el navegador debe implantarse con el cuidado de que sea compatible en los navegadores más utilizados.

5.5.5. Esquemas

En la figura 31 se muestra el diagrama de secuencias para la ejecución de la lógica en los navegadores.

Figura 31. Secuencia de ejecución de lógica en un navegador



5.5.6. Roles

- Diseñador y desarrollador

5.6. Tipos de componentes

5.6.1. Contexto

Los componentes deben agruparse según su capa y luego según la naturaleza de la función que realizan. En los componentes distribuidos se tiene casi toda la lógica de la aplicación y el acceso a los datos, y se acceden por medio de las páginas Web consideradas también como componenetes. Dentro de cada uno de estos grupos de componentes existen subtipos que realizan funciones más específicas dentro de cada capa y es importante identificarlos para tener una aplicación mejor modularizada.

5.6.2. Problema

¿Cuáles tipos de componentes pueden estar en una aplicación Web?

5.6.3. Solución

A continuación, se presentan los componentes agrupados por cada una de las 3 capas de una aplicación y luego se muestran los diferentes subtipos que generalmente se utiliza.

5.6.3.1 Componentes de presentación

Estos componentes son los que interactúan con el usuario. Estos deben hacer llamadas a los componentes de lógica de la aplicación. En aplicaciones para Internet los componentes de presentaciónson las páginas Web.

5.6.3.2 Componentes de la lógica de la aplicación

Los componentes de la lógica de la aplicación se realizan en módulos o bibliotecas que encapsulan determinada funcionalidad. Según la funcionalidad que tengan pueden clasificarse de la siguiente forma:

a. **Componentes de reglas del negocio o controladores:** son componentes que se encargan de realizar los procesos de funciones del negocio. Por ejemplo, podemos mencionar la operación de transferencia de fondos entre cuentas en un banco. En esta operación debe seguirse un proceso de debitar la cuenta origen, realizar el crédito a la cuenta destino y las acciones a tomar si existe algún problema en la transacción.

b. **Componentes de validaciones y cálculos:** son los encargados de proveer funcionalidad como cálculos matemáticos y validaciones lógicas del negocio. Por ejemplo, la validación del número de una tarjeta de crédito, cálculos de impuestos, etc.

c. **Componentes de acceso a datos:** proveen el acceso a la capa de datos. Se encargan solamente de proveer la información desde la capa de datos para que sean utilizados por los componentes de reglas del negocio. Tienen la conectividad con los datos y son capaces de comunicarse con otros sistemas e integrar información. Un ejemplo de una función de estos componentes podría ser el consultar el saldo de una cuenta.

d. **Componentes de formato de datos:** realizan tareas de formateo de datos provenientes de la capa de datos o que van hacia la capa de datos. Un ejemplo podría ser cuando cantidades de dinero se guardan con punto decimal y otras

no. Estos se utilizan bastante en la integración de sistemas para compatibilidad de tipo de datos.

e. **Componentes de conectividad con dispositivos:** son componentes que permiten la interacción con una tecnología en particular de *hardware*. Algunos ejemplos pueden ser: conectar a una aplicación con una máquina de fax necesaria para enviar estados de cuenta o integrarla con dispositivos para validación de seguridad.

f. **Componentes de utilitarios:** se encargan de realizar tareas que no van relacionados directamente con el negocio. Son bibliotecas de funcionalidad variada. Ejemplo de la funcionalidad es la encripción de información y generación de correlativos.

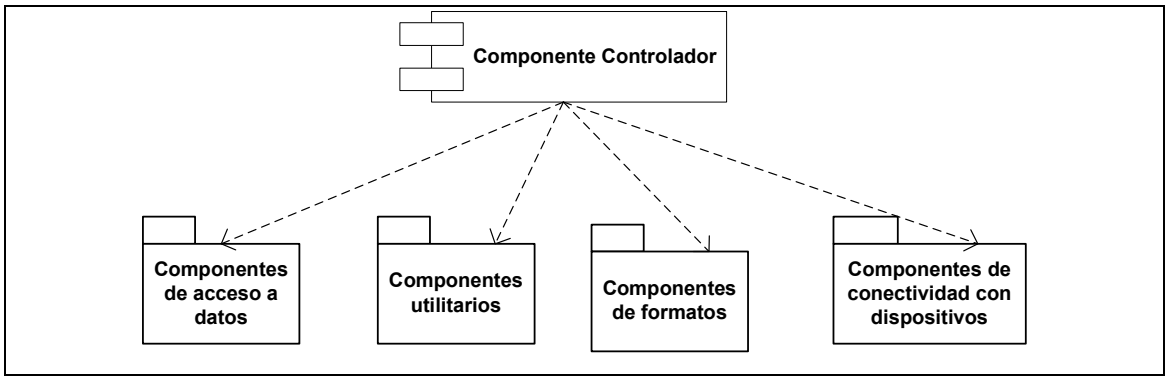
5.6.4. Consecuencias

Al dividir los componentes según su funcionalidad se logra una aplicación modularizada que permitirá mejor la reutilización de cada uno dentro de la misma aplicación y para reutilizarlos en otras. También se facilita el desarrollo de las aplicaciones desarrolladas por equipos grandes y la realización de pruebas modulares.

5.6.5. Esquemas

En la figura 32 se muestra el diagrama de como se relacionan los tipos de componentes de la lógica de la aplicación, en donde el orquestador es el componente de lógica de la aplicación que muchas veces es llamado “controlador”.

Figura 32. Tipos de componentes y sus dependencias



5.6.6. Roles

- Diseñador
- Desarrollador

5.7. Modelo, Control y Vista

5.7.1. Contexto

Desde las páginas Web se realizan solicitudes de actualización y consultas de los datos a los componentes de lógica de la aplicación. Toda la lógica de la aplicación podría solamente concentrarse en la página Web pensando en facilitar el desarrollo de la aplicación. Si se maneja de esta forma se dificultan los cambios a la interfaz de usuario y el poder realizar interfaces especializadas para dispositivos que podrían acceder la aplicación como PDAs (*Personal Digital Assistants*, Asistentes Personales Digitales) o teléfonos celulares.

La información solicitada a un componente de lógica de la aplicación puede desplegado de varias formas: un reporte de texto o una gráfica. También es importante decir que las habilidades que debe tener una persona para desarrollar la interfaz de la página Web son diferentes para el que se encarga de desarrollar la lógica de la aplicación.

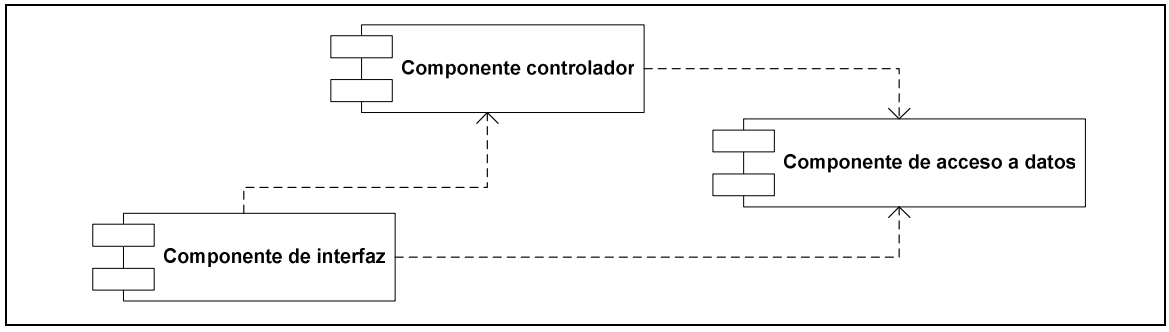
5.7.2. Problema

¿Cómo hacer independientes la interfaz de usuario de la lógica de la aplicación?

5.7.3. Solución

Se deben definir componentes enfocados a la separación de responsabilidades para que puedan ser extensibles y reutilizables. Así deben existir páginas Web que actúen como componentes de interfaz o vista, componentes de acceso a datos y los componentes controladores que orquestan a los anteriores. La figura 33 nos muestra las dependencias que deben existir entre cada uno de los componentes.

Figura 33. Dependencias de componentes en el patrón modelo, vista y controlador



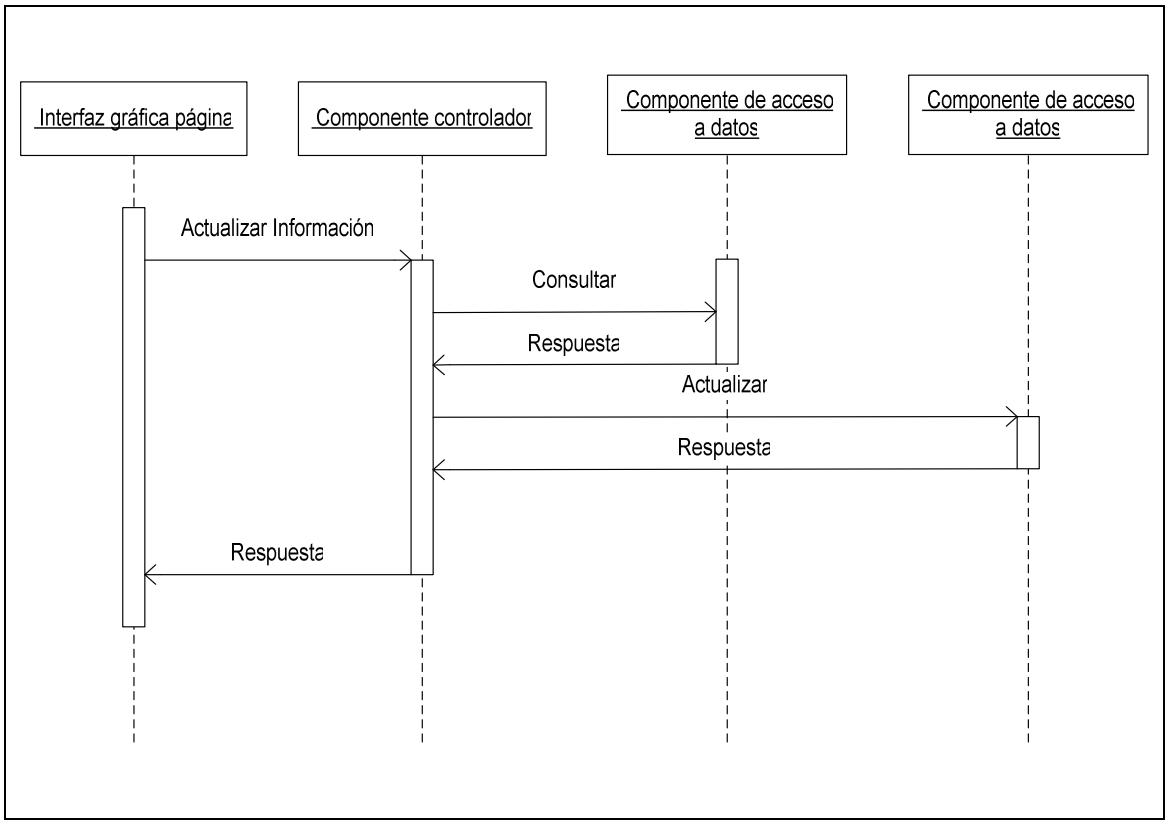
5.7.4. Consecuencias

Al separar cada uno de los componentes con responsabilidades bien definidas se obtiene una aplicación Web distribuida en N capas.

5.7.5. Esquemas

En la figura 34 se muestra el diagrama de secuencias de este patrón de diseño. Desde la página Web se realiza una solicitud de consulta y otra de actualización y la información es devuelta a la página Web. La página Web solamente se comunica con el componente controlador que tiene toda la lógica y este a su vez con componentes de acceso a datos.

Figura 34. Diagrama de secuencias del patrón modelo, vista y controlador



5.7.6. Roles

- Arquitecto
- Diseñador
- Desarrollador

5.8. *Broker*

5.8.1. Contexto

Un *broker es* un agente intermediario que hace transparente la ubicación de un componente en una red. Las aplicaciones distribuidas pueden ejecutarse con componentes que se encuentren en diferentes servidores en la red por las siguientes razones:

- Utilizar un *cluster* de servidores
- Por seguridad están en diferentes segmentos de la red
- Componentes de distintas plataformas de *software*
- Componentes o servicios de otras aplicaciones

En la implementación de aplicaciones distribuidas deben considerarse aspectos como la concurrencia y conexión entre distintas plataformas de *software* sobre conexiones de red poco confiables.

5.8.2. Problema

¿Cómo se puede implementar una aplicación distribuida sin preocuparse de los detalles de la comunicación entre componentes?

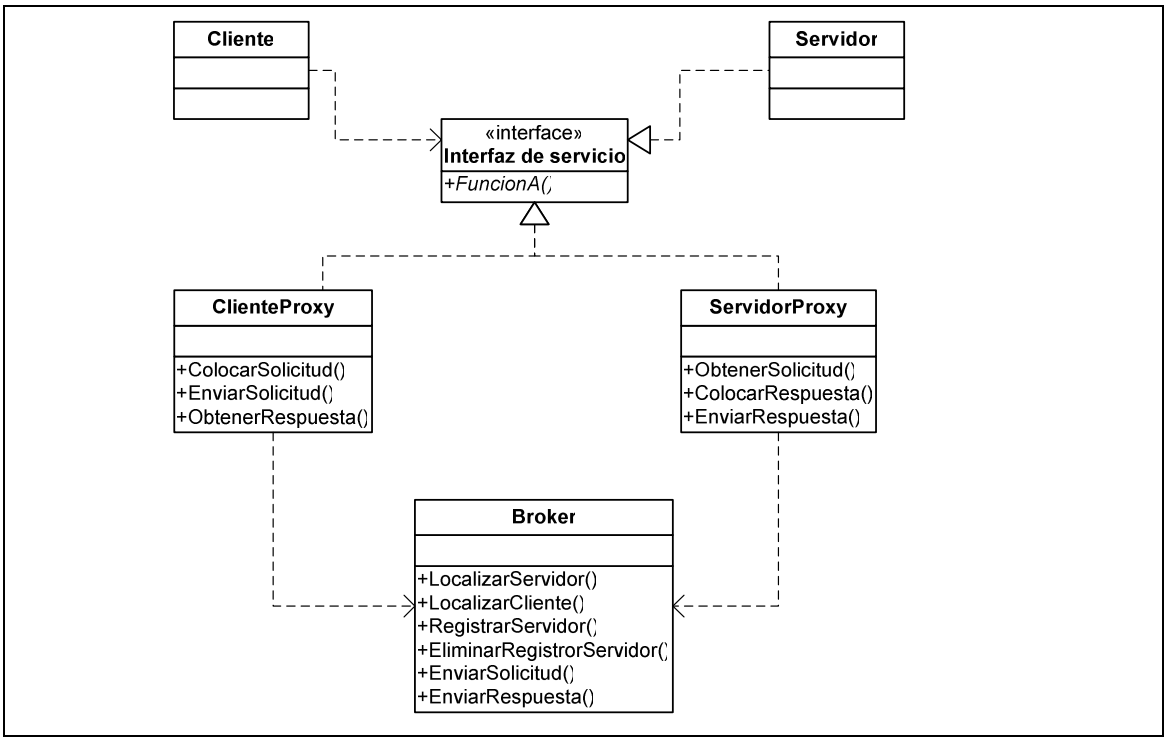
5.8.3. Solución

Un *broker* es generalmente implementado en las plataformas de aplicaciones distribuidas y oculta los detalles para la ubicación de los componentes en la red. Debe tener un diseño similar al presentado en la figura 35. El componente puede ser utilzado desde cualquier lugar accesible al servidor donde se encuentre.

El *broker* debe permitir crear una interfaz que hace creer al solicitante del componente (cliente) que está realizando una invocación local cuando realmente con la utilización del *broker* se realiza una llamada remota. Para esto el *broker* tiene un "*proxy* de componentes" en el cliente y en el servidor

La información para la localización del cliente y del servidor se encuentra en cada uno de los "*proxy* de componentes" y es utilizada en la invocación y envío de respuesta. El *broker* tiene funciones para la localización y búsqueda de componentes que generalmente son ocultadas al desarrollador. Además, los "*proxy* de componentes" tienen la información de las funciones y procedimientos que se encuentran disponibles en el componente.

Figura 35. Diagrama de clases de un b*roker*



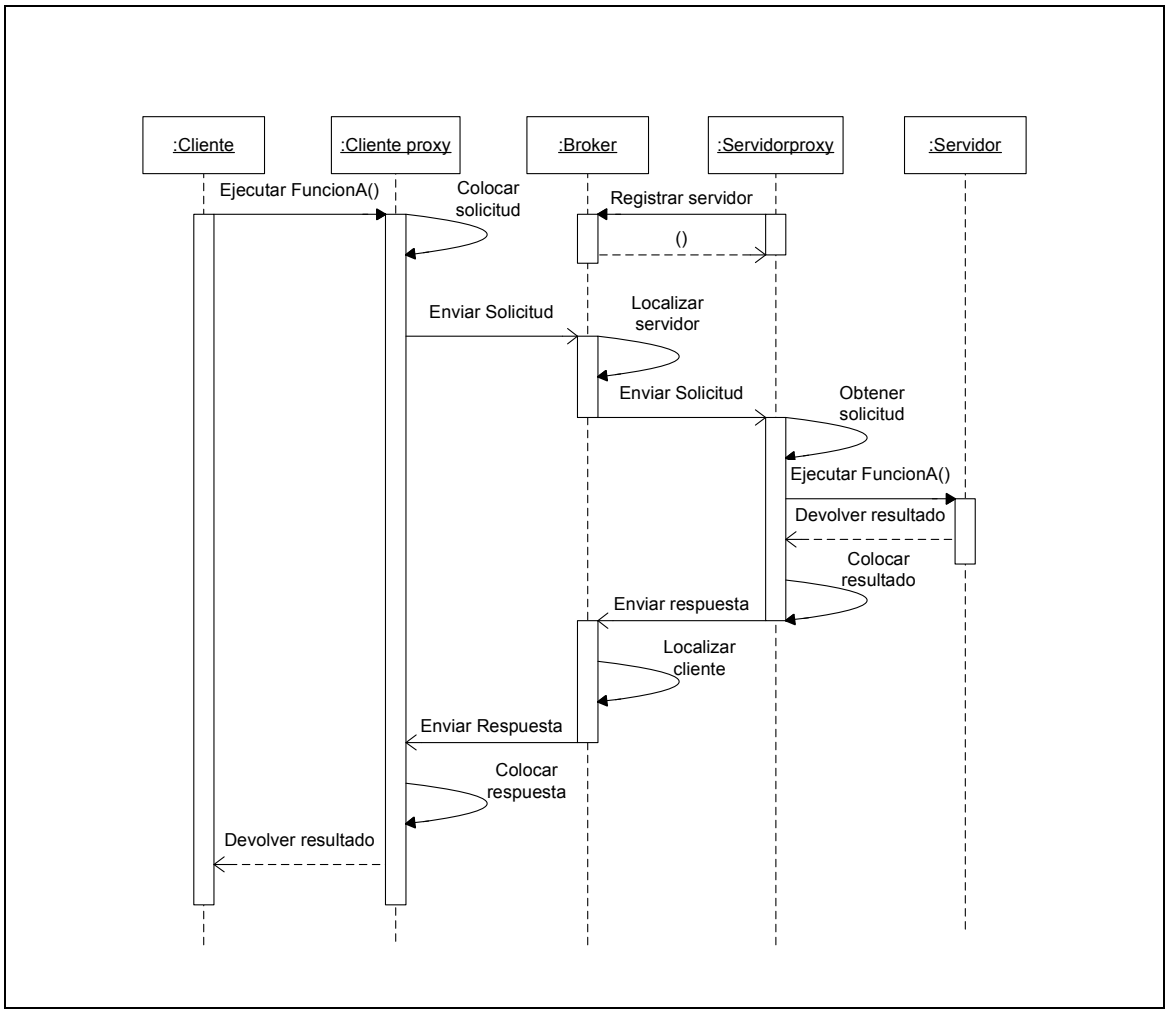
5.8.4. Consecuencias

Al utlizar un *broker* el desarrollador se olvida de los detalles de comunicación de componentes a través de la red, enfocándose a la solución de la aplicación.

5.8.5. Esquemas

En la figura 36, se muestra la secuencia de la invocación de la función de un componente remoto. Debe observarse que el *broker* actúa como intermediario entre los “componentes *proxy*”.

Figura 36. Diagrama de secuencia de comunicación entre componentes utilizando un *broker*



5.8.6. Roles

- Arquitecto
- Diseñador
- Desarrollador

CONCLUSIONES

1. El diseño por patrones apoya los conceptos de reutilización de los componentes de *software* para las aplicaciones Web distribuidas en N capas y están orientados a los distintos roles que existen en los equipos de trabajo de proyectos de este tipo de aplicaciones.

2. Los patrones de diseño ayudan a comprender los conceptos de las aplicaciones Web distribuidas en N capas y pueden ser utilizados como apoyo al diseño en aplicaciones que se construirán.

3. Los patrones de arquitectura y diseño son independientes de la plataforma y lenguaje de programación que se utilizará. Dan lineamientos generales que deben considerarse pero no se apegan a un producto en particular. Los patrones de implementación si se apegan a un producto específico existente en el mercado.

RECOMENDACIONES

1. Los ejemplos de patrones sirven como un modelo porque siempre es necesario aplicar el razonamiento y la experiencia en el momento de diseñar una aplicación.

2. Se sugiere diseñar aplicaciones Web de N capas para simplificarlas, modularizarlas y hacer reutilizables los componentes que la forman.

3. Los patrones de arquitectura y diseño ayudan a identificar y diseñar las capas de una aplicación. Debe empezarse por los de arquitectura y luego seguir con los de diseño porque así se va de lo más general a lo más específico.

4. La administración del ciclo de vida de las aplicaciones Web es muy importante, ya que es bastante corto y dinámico, por lo que se recomienda utilizar metodologías incrementales e iterativas apoyadas por patrones orientados a roles.

BIBLIOGRAFÍA

1. Conallen, Jim. **Building Web Applications with UML**. 2a Ed. U.S.A. Pearson Education, Inc. 2003.

2. **CORBA, DCOM y JAVA-RMI.** http://www.execpc.com/~gopalan/, 2005.

3. Gamma, Erich y otros. **Design Patterns: Elements of Reusable Object-Oriented Software**. 29a Ed. U.S.A. Addison Wesley . 2004.

4. Orfali, Robert y Harkey Dan. **Client/Server Programming with Java and CORBA**. 2a. Ed. U.S.A. John Wiley & Sons, Inc. 1999.

5. Orfali, Robert y otros. **Client/Server Survival Guide**. 23. Ed. U.S.A. John Wiley & Sons, Inc. 1998.

6. **Patrones y antipatrones.**http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/mtj_3317.asp, 2005.

7. Schmuller, Joseph. **Aprendiendo UML en 24 horas**. 1a. Ed. México. Pearson Educación de México, S.A. de C.V. 2000.

8. **Semantic Web.** http://www.semanticweb.org, 2005.

9. Trowbridge, David y otros. **Enterprise Solution Patterns using Microsoft .NET Version 2.0**. 1a. Ed. U.S.A. Microsoft Corp. 2003.