UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

INGENIERIA EN COMPUTACION

TESIS:

Desarrollo de una aplicación de foros de discusión por medio de las tecnologías ASP y JSP y comparación de ambas tecnologías y sus plataformas subyacentes

AUTOR:

Gonzalo Eduardo Santisbón Rodríguez

ASESOR:

Juan Gastaldi Pérez

MAYO DEL 2003

I N D I C E

INTRODUCCION.-

El desarrollo de aplicaciones Intranet-Internet ha sido un área que a últimas fechas ha experimentado un crecimiento notable, pese a la mala situación financiera originada en la generación de expectativas exageradas, junto con la ambición desmedida alrededor del fenómeno dotcom (punto com); sin embargo, la creciente importancia de las tecnologías alrededor de Internet no ha experimentado un riesgo real. Cultural, tecnológica y económicamente, Internet ha ido adquiriendo cada vez más importancia y no existen indicios de que eso vaya a cambiar a corto o mediano plazo sino al contrario. Probablemente las curvas de crecimiento por venir no sean similares al crecimiento explosivo del que fuimos testigos. Seguramente los recursos económicos invertidos en la Internet serán ahora condicionados a un control mucho más estricto basado en expectativas de utilidades más realistas.

Durante este crecimiento explosivo de la Internet resultaba lógico que las tecnologías subyacentes experimentaran también un desarrollo explosivo. En poco tiempo pasamos de páginas y otros contenidos estáticos a aplicaciones interactivas y la tendencia actual de remplazar la interfase de las aplicaciones tradicionales a una interfaz Web. Dado que la necesidad de generar contenido dinámico se remonta casi a los mismos orígenes de Internet, han sido muchas las tecnologías que han intentado solventar esta necesidad. Sin embargo a nuestro entender, la primera que brindo una solución efectiva y suficientemente completa son los guiones CGI[1].

CGI establece una especificación acerca de cómo comunicar un servidor Web con alguna aplicación propia llamada aplicación CGI. Una aplicación CGI es aquélla que cumple con la especificación CGI al momento de recibir y enviar información al servidor Web, dichas aplicaciones pueden ser escritas en una gran variedad de lenguajes y conceptualmente son externas al servidor Web. Un ejemplo típico de una solución CGI sería el uso de formularios HTML a los que el servidor Web le pasaba el contenido capturado por el usuario a una aplicación CGI externa la cual procesa dicha información y al finalizar genera una respuesta o confirmación la cual es pasada al servidor Web quien a su vez la pasa al navegador en el cliente a través de HTTP.

En sus orígenes la tecnología CGI sufría de importantes problemas de desempeño ya que al ser las aplicaciones CGI externas al servidor Web, se requería lanzar un nuevo proceso (en el servidor donde reside la

[1] Common Gateway Interface

aplicación CGI) para responder a cada interacción de cada usuario. Debido a estos problemas de desempeño surgieron otras tecnologías

Tales como Fast CGI (CGI rápido) en donde los procesos CGI no se reiniciaban para cada petición sino que se mantenían corriendo y permitían ser reutilizados. Adicionalmente se desarrollaron extensiones a los propios servidores Web gracias a la implementación de API's específicos a la interacción con los mismos, dichos API's permitían una ejecución de servicios mucho más eficientes aunque con la desventaja de ser propietarias de algún servidor Web en particular. Tal es el caso de ISAPI y NSAPI.

Paralelamente a las tecnologías del lado del servidor, del lado del cliente también se desarrollaron y evolucionaron tecnologías abocadas a la producción de contenido dinámico basadas en estándares relacionados o dependientes del estándar HTML. Tales como JavaSript, Hojas de estilo en cascada, etc. Las cuales en su conjunto se conocen como HTML dinámico (DHTML). Sin embargo la funcionalidad que se puede alcanzar con este tipo de tecnologías es bastante limitada. Adicionalmente dadas las diferencias en el soporte al DHTML entre los navegadores del momento, la producción de interfaces de usuario basadas en DHTML se convertía en una labor ardua sin brindar la certeza de producir aplicaciones realmente portables entre clientes (o específicamente entre diversos navegadores o versiones de los mismos).

Sin embargo, la principal desventaja de desarrollar una aplicación Web cuya funcionalidad se apoye principalmente en tecnologías del lado del cliente como la que comprenden el DHTML, la constituye la endeble seguridad que proporcionan las mismas. Probablemente los riesgos a la seguridad que supone el responsabilizar de la funcionalidad de un sistema a dichas tecnologías serían aceptables en escenarios en donde la necesidad de generación de contenido dinámico (generalmente de carácter público) es decididamente mayor que la de implementar ciertas reglas de negocio de una aplicación empresarial típica. Definitivamente cuando las reglas de negocio implementadas en una aplicación tienen un carácter crítico para la integridad de otros procesos del negocio relacionados con dicha aplicación, el implementar dicha funcionalidad en tecnologías del lado del cliente como DHTML no resulta una opción recomendable o incluso viable.

Estas limitaciones, tanto en las tecnologías del lado del servidor (CGI, ISAPI, NSAPI) como en aquellas del lado del cliente (DHTML) al no satisfacer adecuadamente la necesidad de contar con un medio o esquema de desarrollo de aplicaciones Internet flexible y sin la inseguridad asociada a DHTML, promovieron el surgimiento de

tecnologías más flexibles del lado del servidor, en especial surgieron tecnologías basadas en plantillas que definieron un conjunto adicional de Tags similares a los de HTML pero estos permitían invocar un proceso, servicio o recurso del Servidor Web de una manera sencilla. Dichos Tags adicionales son traducidos a CGI por el propio servidor Web liberando a los programadores o desarrolladores de contenido de la ardua tarea de lidiar directamente con CGI o con API's del servidor.

Adicionalmente se integraron varios lenguajes interpretados o de guión (a los que se les conoce como Scripting Languages) que permiten programar reglas de negocio relativamente complejas insertadas en un solo archivo que incluye tanto el contenido HTML, los Tags adicionales y el propio código del lenguaje interpretado. La primera implementación exitosa de dichas herramientas y tecnologías fue la plataforma ASP de Microsoft.

Posteriormente con el creciente éxito de Java como lenguaje universalmente portable y en función del interés de Sun y otras empresas de convertir a Java en una alternativa viable para aplicaciones empresariales (J2EE) se desarrollaron inicialmente el API de servlets que permitían de una manera sencilla construir componentes que intercepten las peticiones a un servidor Web y respondan a las mismas, posteriormente se desarrolló también la tecnología JSP que presenta grandes similitudes con ASP.

Pero si bien estas nuevas tecnologías verdaderamente han facilitado la labor que involucra el desarrollo de aplicaciones Intranet-Internet, no todo es positivo respecto a las mismas, uno de los principales problemas lo constituye el hecho de que ambas tecnologías no están regidas por estándares universales (como por ejemplo el HTML), sino que los promulgadores de ambas tecnologías han definido sus propios estándares, los cuales han tenido mayor o menor aceptación y adopción por parte de la industria pero al mismo tiempo han dividido a la misma entre los que están a favor de uno u otro estándar.

Debemos reconocer que las dos alternativas han tenido éxito en la industria de La Tecnología de la Información y naturalmente estas dan muestra de la fuerte competencia entre dos arquitecturas también rivales: Por un lado está la plataforma que incluye las tecnologías ActiveX, COM+, ADO, ASP, MTS y Windows NT/2000/XP y por el otro la Plataforma que incluye las tecnologías Java, JavaBeans, Enterprise JavaBeans, JDBC, JSP, y Servlets.

Dada la amplitud de ambas Plataformas, para las empresas de desarrollo de software es difícil no inclinarse por alguna de las dos debido al costo y esfuerzo que supone el intentar abarcar simultáneamente a las mismas. Este fenómeno ha sido una de las causas principales de la marcada división en la industria que mencionamos con anterioridad.

Además, la enconada competencia en la que los propulsores de ambas tecnologías se han engarzado, hace difícil el encontrar información no sesgada que permitiera a una empresa realizar una evaluación sobre cual plataforma encaja mejor ante los requerimientos de algún proyecto de desarrollo. Lamentablemente han terminando imponiéndose las fuerzas de la mercadotecnia a las cualidades tecnológicas que cada plataforma pueda tener.

Pese a esto último, en la práctica (y sorprendentemente) no es tan difícil trasladar y aplicar los conocimientos y recursos adquiridos por una empresa de una plataforma a la otra. El escenario que enfrente algún proyecto de software en donde se tengan los recursos humanos experimentados en una tecnología pero con el requisito de utilizar a la otra, no es tan sombrío. Pues afortunadamente ambas tienen muchas bases comunes. Por lo que el proceso de migración de conocimientos o experiencia, no es un obstáculo insalvable.

Dentro del conjunto de bases comunes en primer lugar tenemos a la arquitectura de diseño basada en componentes, que ambas plataformas persiguen. Gracias a esto se promueve la separación entre servicios enfocados a la presentación, visualización e interacción con el usuario, de los servicios que encapsulan las reglas de negocio del sistema, y los servicios que se especializan en la persistencia de la información.

El desarrollo basado en componentes facilita además el reemplazo de los mismos y reduce la dependencia entre las diversas partes que conforman un sistema. Si a esto le añadimos que, si bien no con igual vehemencia, ambas plataformas promueven la orientación a objetos. Obtenemos una mayor separación o desacoplamiento entre los múltiples servicios de la aplicación. Y sobre todo una menor dependencia con alguna tecnología propietaria de alguna de las plataformas.

Adicionalmente, un hecho fundamental del desarrollo de aplicaciones Intranet-Internet es que la parte de visualización será casi siempre a través de HTML. Por lo tanto se tiene un mayor grado de estandarización en dicha capa independientemente de la tecnología empleada. Normalmente será del lado de los componentes que

encapsulan las reglas de negocio y la persistencia, donde se tenga la mayor diferenciación entre estas dos plataformas[2].

Mediante este trabajo de tesis se pretende abordar la tarea de comparar ambas plataformas por medio de la construcción de una aplicación que sirva de caso práctico en este trabajo. Uno de los propósitos fundamentales de este trabajo de tesis es tratar de filtrar los dogmatismos y mercadotecnia inherentes a las plataformas intentando producir información no sesgada que auxilie en la evaluación de ambas plataformas. Además, se pretenden alcanzar una serie de metas a saber:

- Auxiliar la toma de decisiones cuando se tenga que escoger a alguna de estas dos tecnologías como plataforma de desarrollo de algún proyecto.
- Contrarrestar en la medida de lo posible las influencias de mercadotécnicas o dogmáticas que han creado este ambiente de rivalidad entre estas dos tecnologías en el fondo equivalentes.
- Esbozar algunas de las principales tareas necesarias y su dificultad intrínseca en caso de requerirse migrar una aplicación existente de una tecnología a la otra.

Como parte de este trabajo de tesis se abordará la tarea de desarrollar una aplicación de foros de discusión la cual servirá como caso práctico que nos permitirá mostrar los principales aspectos del desarrollo de una aplicación web típica usando ambas tecnologías y plataformas. En el desarrollo de este caso práctico buscamos a su vez alcanzar también una serie de metas a saber:

- Presentar algunos lineamientos básicos y patrones para minimizar el acoplamiento o dependencia a cualquiera de estas dos tecnologías por alguna aplicación que se desarrolle sobre la misma.
- Mostrar, los más que se pueda, los puntos fuertes y débiles que en la práctica cada tecnología presenta como plataforma de desarrollo de aplicaciones Internet/Intranet.
- Complementar los argumentos teóricos presentados en este trabajo de tesis, con argumentos que se desprendan directamente de este caso práctico.
- Tener un marco de referencia concreto de ambas plataformas, originado por nosotros mismos, que al abordar la comparación de

[2] Sin embargo un buen análisis y diseño orientados a objetos minimizarían las dependencias de las capas de dominio y persistencia a tecnologías propietarias.

las mismas nos permita invocar los resultados reales obtenidos en la construcción del caso práctico.

En el primer capítulo de la tesis se abordara pues, la fase de análisis para el desarrollo de nuestro caso práctico.

CAPÍTULO 1. ANALISIS

1.1 Introducción al Análisis.-

En el presente capítulo se abordará el análisis de nuestro proyecto de foros de discusión a través del web. Que como indicamos servirá como caso de estudio para realizar la comparación entre ASP y JSP, que es el objetivo de esta tesis.

Ya habíamos mencionado que uno de las metas es el elaborar un análisis genérico e independiente (hasta donde sea posible) de la plataforma de desarrollo e implantación, en este mismo sentido se determinó que el análisis sea orientado a objetos, con una notación estándar como es el UML y utilizando metodologías generalmente aceptadas de desarrollo de software.

Con esto en mente procederemos pues ha elaborar el análisis de nuestro proyecto. Generalmente los autores de UML y metodologías de orientación a objetos recomiendan que el análisis orientado a objetos (OOA) comience con la enumeración de los casos de uso y la elaboración de los diagramas correspondientes.

1.2 Casos de Uso Iniciales.-

La importancia que han adquirido los diagramas de casos de usos en las metodologías de desarrollo de software orientado a objetos radica principalmente en el hecho de que dichos diagramas son unos de los más sencillos de realizar y comprender (incluso para los usuarios finales) dentro conjunto de diagramas del UML y dado que con estos diagramas resulta relativamente sencillo plasmar los requerimientos a cubrir por el sistema, terminan siendo una herramienta visual importante para validar que el resto del análisis, el diseño y el desarrollo cumpla con los propósitos previamente establecidos.

Además de plasmar los requerimientos de un sistema, los casos de uso ayudan a fraccionar la funcionalidad del mismo en unidades más simples (similarmente al proceso top-down del análisis estructurado) dividiendo al sistema en Casos de Uso y los Actores que interactúan con los mismos.

Sin pretender realizar una exhaustiva exposición de los casos de uso en particular o del UML en general (prefiero en cambio remitir al que

le interese dicho aspecto a los libros de referencia de UML de los cuales incluyo algunos en la bibliografía) considero en cambio que vale la pena realizar una breve definición de los actores y los casos de uso.

Los actores representan un rol que se le otorga a cualquier posible usuario del sistema (pudiendo ser personas, otros sistemas u objetos externos) con la premisa de que no sean elementos propios o internos de la funcionalidad que se este estudiando. Los actores proveen de estímulos que disparan la acción del sistema o son depositarios de la respuesta del sistema a estos estímulos.

Los casos de uso describen el comportamiento del sistema al recibir determinado estimulo por un actor. Los casos de uso indican: El tipo de estimulo que el sistema recibe, los actores de los que se reciben los estímulos y los actores a los que se les remite determinada respuesta; adicionalmente los casos de uso pueden señalar las posibles excepciones al comportamiento normal del sistema.

En las etapas iniciales de análisis, los casos de uso se encuentran principalmente al estudiar los requerimientos del sistema así como de sesiones de trabajo ente analistas y usuarios. Posteriormente es común que la lista de casos de uso crezca con nuevos casos y que los casos de uso se refinen y probablemente se dividan en casos de uso más simples (también es posible que algunos casos de uso sea desechados por ser redundantes o innecesarios).

En nuestro proyecto de foros de discusión en particular tomando en cuenta los requerimientos, partiremos con la siguiente lista de casos de uso:

- Un usuario no-registrado se da de alta en el sitio
- Un usuario no-registrado se registra (identifica) convirtiéndose en usuario normal
- Un usuario modifica sus datos personales (perfil)
- Un usuario accede a un foro
- Un usuario se suscribe a un foro (establece preferencias de la suscripción)
- Un usuario cambia las preferencias de su suscripción
- Un usuario accede a un tópico
- Un usuario lee los tópicos de un foro
- Un usuario lee los mensajes de un tópico
- Un usuario lee el detalle de un mensaje

- Un usuario envía un mensaje en un tópico (responde a otro mensaje o directamente al tópico)
- Un usuario envía un mensaje en un tópico nuevo
- Un usuario busca dentro de los tópicos y mensajes sobre la base de sus atributos
- Un usuario navega entre los tópicos de un foro (siguiente, anterior)
- El moderador de un foro modifica un mensaje
- El moderador de un foro bloquea a un usuario
- El moderador de un foro elimina un tópico
- El moderador de un foro elimina un mensaje
- El moderador de un foro depura los mensajes inactivos de hace más de 3 meses
- El administrador del sitio designa a un usuario como moderador de un foro
- El administrador del sitio registra la información de los catálogos
- El administrador del sitio modifica la información de los catálogos
- El administrador del sitio agrega un foro
- El administrador del sitio elimina un foro
- El sistema envía por correo un resumen diario de los nuevos mensajes (dependiendo de los foros de interés de cada usuario y de sus preferencias)

En esta etapa inicial del análisis identificamos también a los siguientes actores principales:

- Los usuarios no-registrados (invitados)
- Los usuarios normales
- Los moderadores de un foro
- Los administradores del sitio

Procederemos ahora a mostrar los diagramas de casos de uso, para simplificar elaboraremos un diagrama por cada actor, esto teniendo en cuenta que un actor administrador puede fungir al mismo tiempo como usuario normal, de hecho nuestro conjunto de actores representa efectivamente una jerarquía en donde el usuario no-registrado (invitado) tiene el menor número de derechos o privilegios, que en este caso se traducen en un menor número de casos de uso a los que tiene acceso tanto para estimular como para recibir respuesta.

Veamos pues el diagrama con los casos de uso con los que interactúa el actor “invitado”:

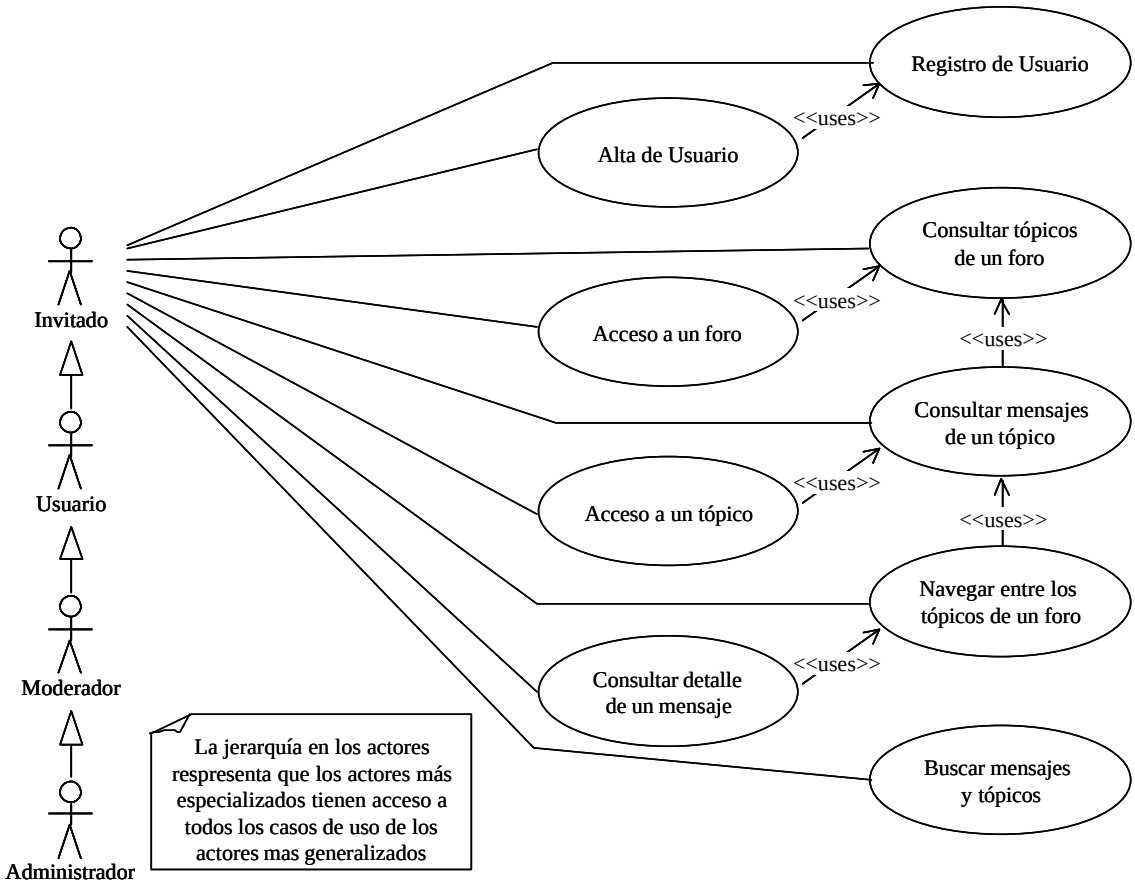


Diagrama 1.1
Casos de Uso del Actor Usuario No-Registrado (Invitado)

Como afirma la nota en el diagrama, los casos de uso de este actor son asequibles también para los Actores: Usuario, Moderador y Administrador. Exceptuando esa asociación jerárquica de pseudo-herencia y de algunas relaciones entre casos de uso, no se han querido introducir más *adornos*[3] pues, si se hiciera en etapas tan tempranas del análisis se corre el riesgo de sesgar el proyecto en una dirección específica (que podría ó no ser la mejor dirección).

Continuando con los diagramas de casos de uso tenemos ahora los casos con los que interactúa el actor usuario (esto es un usuario que ya se ha registrado).

[3] Término que en UML denota un indicador que clarifica la interpretación de un diagrama pero que va más allá del conjunto de elementos estándar básicos para un cierto tipo de diagrama UML.

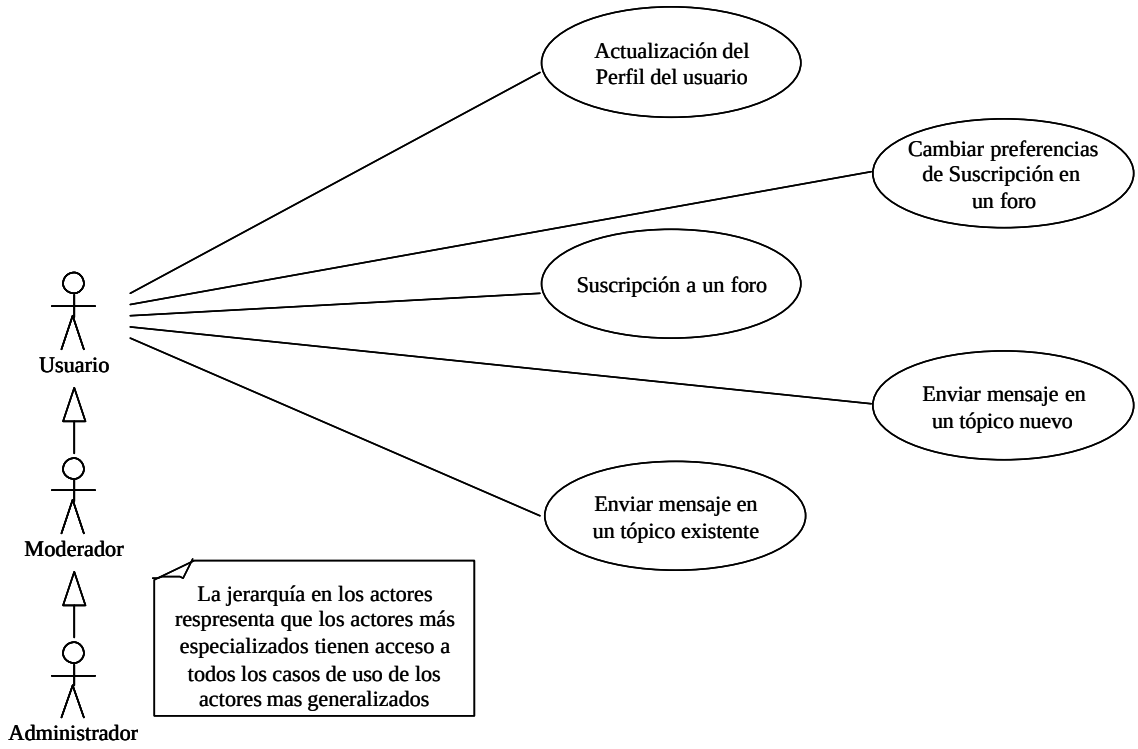


Diagrama 1.2
Casos de Uso del Actor Usuario

Como podemos ver, entre los diagramas 1.1 y 1.2 está plasmada la funcionalidad principal del sistema, esto es el envío de mensajes, la generación de tópicos y la consulta de los mismos.

En particular el caso de uso “Suscripción a un Foro” plantea un aspecto muy importante que es el que un usuario se suscribe a un foro y esto hace surgir la pregunta cual es la diferencia ente un usuario que se ha suscrito a un foro y otro que no lo ha hecho. Aun sin tener un modelo conceptual o un diagrama de clases o de entidad-relación que lo establezca sabemos que puede haber varios foros en el sitio dado que se menciono un caso de uso en el que el administrador del sitio agrega un foro, por lo tanto la suscripción a un foro no puede ser parte del *estado*[4] de algún objeto de tipo usuario[5] pues un usuario se podría suscribir a varios foros.

[4] El estado de un objeto lo compone el conjunto de los valores de todos sus atributos en un instante de tiempo dado.
[5] Esto por que los actores de los diagramas de casos de uso pueden ó no terminar siendo representados por clases durante el diseño y desarrollo del sistema.

Si bien todavía ni siquiera hemos empezado la búsqueda de abstracciones o conceptos en nuestro análisis, los descubrimientos de este tipo encontrados en las etapas iniciales de elaboración de casos de uso generalmente valen la pena ser documentados de alguna manera para tenerlos en cuenta en etapas posteriores del análisis o diseño de la aplicación.

Continuando con los diagramas de casos de uso tenemos ahora los casos con los que interactúa el actor moderador.

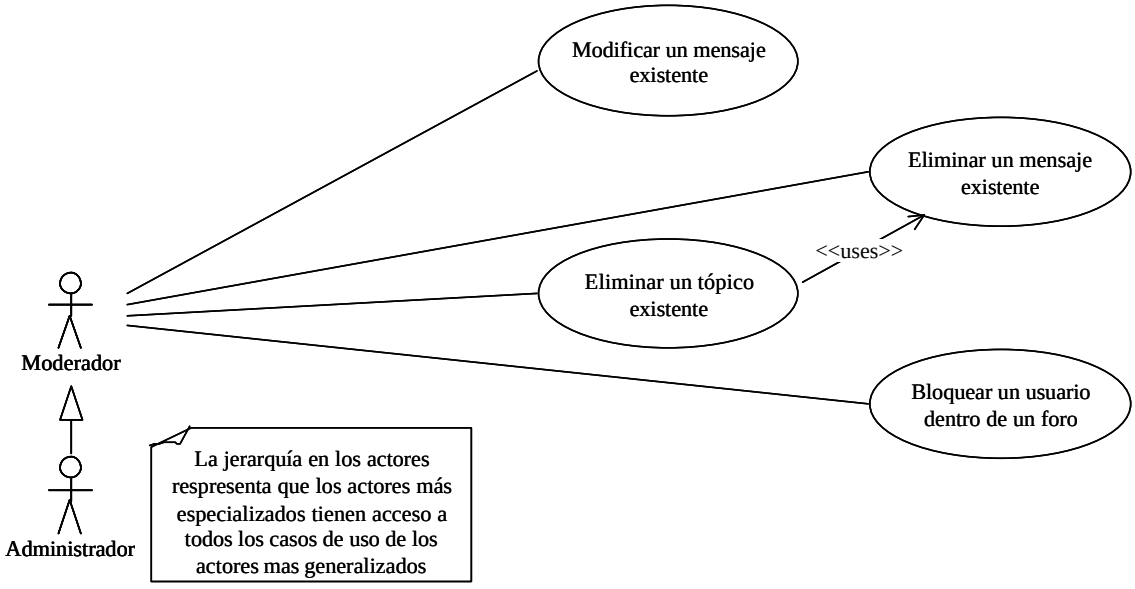


Diagrama 1.3
Casos de Uso del Actor Moderador

Durante la elaboración de los diagramas 1.1 y 1.3 se encontró que el caso de uso en el que el moderador de un foro depura los mensajes inactivos de hace más de 3 meses puede ser cubierto por medio de los casos de uso “Buscar mensajes y tópicos” y “Eliminar un tópico existente”, basta con que alguno de los criterios de búsqueda permita identificar cuanto tiempo ha estado inactivo un tópico, por ejemplo: La fecha en que se envió el último mensaje. El hecho de eliminar redundancia en los diagramas de uso es un fenómeno relativamente común y por supuesto benéfico para el análisis del mismo. Por otra parte si posteriormente se determina que existirá una clase tipo foro, tenemos un candidato a atributo para la misma (que sería la fecha del último mensaje).

Continuando con los diagramas de casos de uso tenemos ahora los casos con los que interactúa el actor administrador.

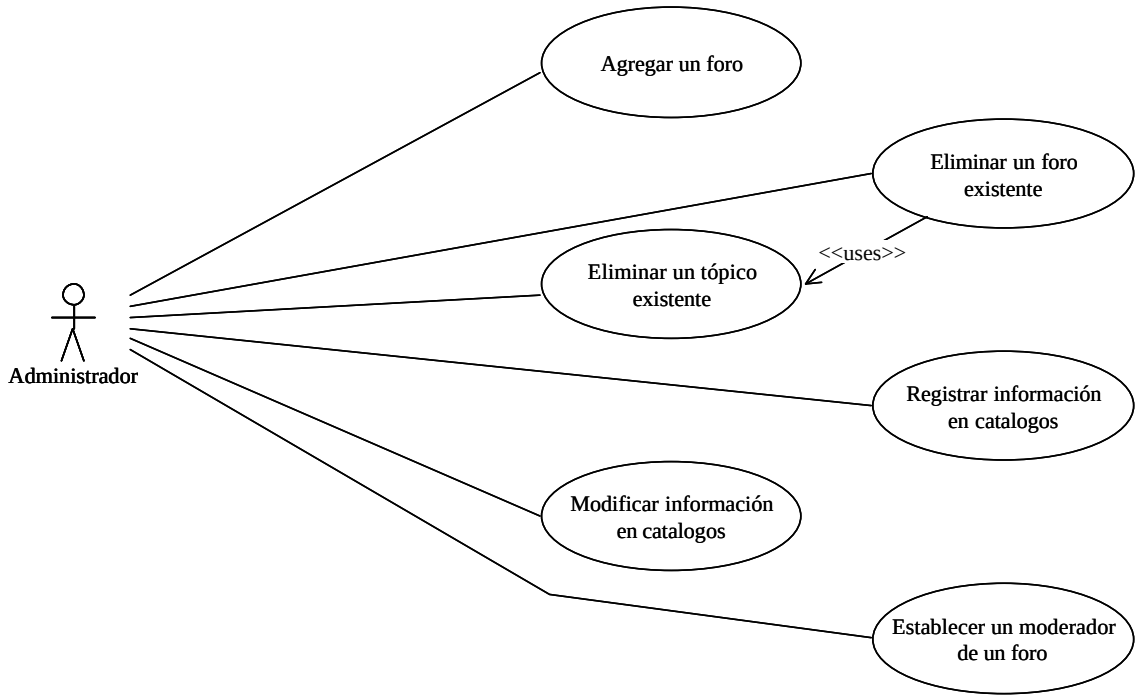


Diagrama 1.4
Casos de Uso del Actor Administrador

Simplemente como aclaración se decidió incluir el caso de uso “eliminar tópico existente” que ya habíamos puesto en el diagrama 1.3 debido a que el caso de uso “eliminar foro existente” lo utiliza.

Un caso de uso que no hemos puesto todavía en ningún diagrama es el de “El sistema envía por correo un resumen diario de los nuevos mensajes”, este caso de uso es interesante por que a primera vista parece indicar que es el sistema el iniciador del caso de uso. Si analizamos un poco esto podemos en cambio concluir que el actor que inicializa este caso de uso es aquél que es el responsable de ejecutar la aplicación. Esto es que, por ejemplo: Si es el sistema operativo el encargado de lanzar nuestra aplicación al momento de encender la computadora, entonces el actor iniciador puede considerarse tanto a la persona que encendió la computadora como al comando o tarea automatizada que ocasionó la ejecución de nuestra aplicación.

Siguiendo con este caso de uso podemos suponer que los actores que reciben el resultado de este caso de uso son las personas a las que

se les manda el resumen por correo electrónico, pero también es válido suponer que en vez de estas personas, es el servidor de correo electrónico el actor receptor. El diagrama de este caso de uso lo tenemos a continuación:

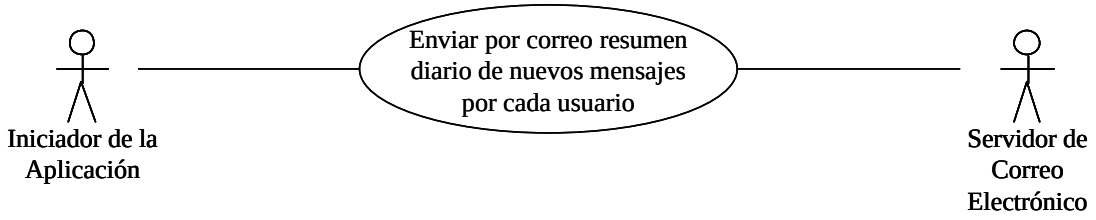


Diagrama 1.5
Caso de Uso: "Enviar por correo resumen diario de nuevos mensajes por cada usuario"

Los casos de uso han adquirido una importancia fundamental en las metodologías actuales de análisis, por ejemplo en el ámbito del UML constituyen una pieza fundamental del mismo y sin embargo estrictamente hablando los casos de uso **no son** orientados a objetos, pues no existe necesariamente una relación directa ni de los actores o los procesos descritos en los casos de uso con las clases que se construirán posteriormente en un sistema orientado a objetos.

Pese a esto y debido entre a otras cosas a que los casos de uso concretizan los requerimientos y funcionalidad de un sistema, son generalmente los casos de uso el producto inicial del análisis orientado a objetos[6] sin embargo, luego de contar con una lista de los casos de uso, la labor siguiente dentro del análisis orientado a objetos es la construcción del modelo conceptual. De hecho el modelo conceptual es al final de cuentas el producto principal del análisis orientado a objetos.

1.3 Análisis de la Estructura.-

1.3.1 Identificación de los conceptos principales.-

Después de plasmar los casos de uso iniciales o generales de un sistema, el siguiente paso consiste en la búsqueda e identificación de los conceptos o abstracciones clave del dominio del problema, las herramientas y técnicas para dicha labor son muchas y se hayan

[6] Sin embargo algunos autores prefieren situar la definición de los casos de uso en alguna etapa previa al análisis como por ejemplo la planeación o captura de requerimientos.

ampliamente documentadas en un buen número de libros, algunos de los cuales se presentan en la bibliografía. En general, la estrategia en la búsqueda de conceptos se basa principalmente en estudiar el dominio del problema o sistema real y aplicar técnicas diversas para resaltar las abstracciones del dominio clave. A continuación presentamos dos de las muchas técnicas válidas:

1.3.1.1 Obtención de conceptos entre los sustantivos de textos del dominio.-

Ésta es quizás la técnica más usada (incluso inconscientemente) en la búsqueda de conceptos y abstracciones de un dominio, sin embargo su empleo exige cierta precaución por que el lenguaje natural no carece de ambigüedades y excentricidades que pueden ser contagiadas al modelo conceptual resultante.

Hay que tener cuidado sobre todo en asegurarse que una abstracción candidata a formar parte del diccionario de conceptos sea efectivamente un elemento del sistema real y no un producto de la utilización de coloridas o extravagantes construcciones del lenguaje.

Otro riesgo importante, sobre todo cuando se analizan modelos de negocios es que los documentos o textos del dominio pueden formar parte de una estructura ya establecida que en si constituye un modelo previo que bien puede tener deficiencias, pero independientemente de esto último, se corre el riesgo de sesgar el análisis en la dirección que siguió el modelo específico del que los documentos forman parte.

Una buena fuente de descripciones textuales del dominio la constituyen los propios casos de uso (sobre todo si están en una versión expandida). Muchos de los sustantivos encontrados en la descripción de un caso de uso constituyen fuertes candidatos a ser conceptos o sino atributos de los mismos (como se verá posteriormente).

1.3.1.2 Revisión de una lista de categorías de conceptos.-

Los conceptos del modelo conceptual representan abstracciones clave en el dominio del problema, esto es que son objetos significativos en un sistema real. Sin embargo en un sistema real se pueden encontrar literalmente infinidad de objetos y resulta más o menos evidente que no todos los objetos de un sistema real son relevantes para ser tomados en cuenta en un modelo del mismo. Una de las técnicas para encontrar y separar a los objetos que si son relevantes constituye en clasificarlos en categorías que normalmente abarcan a los conceptos relevantes para un

modelo, la lista de estas categorías fundamentales se obtiene principalmente por experiencia personal o colectiva o basándose en documentación y patrones de diseño probados y exitosos.

A continuación se presenta una lista sugerida de categorías de conceptos:

- Objetos Físicos o Tangibles
- Especificaciones, diseño o descripciones de cosas
- Lugares
- Transacciones
- Elementos de Transacciones
- Papel o rol de las personas
- Contenedores de cosas
- Cosas dentro de un contenedor
- Otros sistemas de computo o electromecánicos externos al sistema
- Conceptos de nombres abstractos
- Organizaciones
- Eventos
- Procesos (no es una regla general pero algunos procesos pueden ser indicativos de algún concepto subyacente)
- Reglas y políticas
- Catálogos
- Registros de finanzas o contables, contratos legales, etc.
- Instrumentos y servicios financieros
- Manuales, libros, etc.

1.3.1.3 Abstracciones candidatas a incluirse en el modelo conceptual.-

Basándonos en las dos técnicas anteriores y después de analizar las descripciones de nuestros casos de uso así como textos de requerimientos y otros documentos, separamos en este momento algunas de las abstracciones o conceptos más relevantes y que constituyen fuertes candidatos a integrar nuestro modelo conceptual, la lista de abstracciones es:

- Foro
- Tópico
- Mensaje
- Usuario Invitado
- Usuario
- Moderador

- Administrador
- Suscripción
- Resumen diario de mensajes
- Mensaje de correo electrónico
- Datos personales (perfil).

En esta lista se descartaron algunas abstracciones que podrían haber sido consideradas, por ejemplo Detalle de mensaje se pudo tomar como otro concepto, pero se consideró que sus características lo sitúan más como un atributo del concepto mensaje que como un concepto por si mismo, caso contrario es el de Datos personales que aunque si bien podría ser considerado como atributo del concepto usuario, también es posible definir un esquema en el que un usuario mantenga varios perfiles de si mismo, por lo que independientemente de la implementación que se realice posteriormente, se decidió para el análisis considerar a esta abstracción como concepto separado.

1.3.2 Identificación de las asociaciones entre conceptos.-

Una vez encontrado un conjunto de conceptos que en principio parezca suficiente para modelar el sistema real, el siguiente paso en el análisis orientado a objetos es el de encontrar las asociaciones entre estos conceptos, así como la distinción entre las asociaciones que son básicas al dominio de aquellas que sirven simplemente para brindar una comprensión más completa del mismo.

Una asociación es la relación entre dos conceptos resaltando una conexión significativa o interesante entre los mismos. El UML las define como las “relaciones estructurales entre los objetos de diversos tipos”.

Aunque es posible encontrar relaciones entre prácticamente cualquier concepto, en la práctica solo algunas relaciones resultan relevantes para la construcción del modelo conceptual, estas asociaciones trascendentes generalmente implican la percepción o conocimiento de alguno de los conceptos sobre el otro durante un intervalo de tiempo. En otras palabras estas asociaciones relevantes incluyen las relaciones que son útiles de ser recordadas al menos por un tiempo.

La representación de los conceptos y asociaciones del modelo conceptual puede plasmarse gráficamente en la notación de los diagramas de estructura estática del UML. Y en este sentido resulta útil

incluir si es posible la multiplicidad de las asociaciones. La mutilplicidad indica cuantas instancias de un concepto A pueden asociarse a una instancia de un concepto B, por ejemplo: Anteriormente habíamos descubierto que un usuario podría estar asociado a más de un foro (aunque también puede no estar asociado a ningún foro). Adicionalmente en estos diagramas puede ser útil incluir el nombre de la asociación así como la dirección de lectura del nombre.

Para encontrar asociaciones se pueden volver a utilizar algunas de las heurísticas empleadas en la búsqueda de conceptos, a continuación se presenta una lista de categorías de relación que puede ser aprovechado durante la búsqueda de asociaciones entre dos conceptos (en este caso los conceptos A y B):

- A es una parte física de B
- A es una parte lógica de B
- A está físicamente contenido en B
- A está contenido lógicamente en B
- A es una descripción de B
- A es un elemento de línea en una transacción o reporte B
- A se conoce/introduce/registra/presenta/captura en B
- A es miembro de B
- A en una subunidad organizacional de B
- A usa o dirige a B
- A se comunica con B
- A se relaciona con una transacción B
- A es una transacción relacionada con otra transacción B
- A está contiguo a B
- A es propiedad de B

Procedamos ahora a identificar las principales asociaciones en nuestro diccionario de conceptos:

En primer lugar tenemos que un Foro puede *contener* cero o muchos tópicos y es útil conocer a que foro *pertenece* un tópico por lo que ésta es nuestra primera asociación.

Similarmente un tópico puede contener[7] cero o muchos mensajes y es útil conocer a que tópico pertenece un mensaje.

[7] Algunos sistemas de foros de discusión identifican como tópico al primer mensaje de una hebra de discusión, por lo que no manejan el concepto de tópico. Otros sistemas si diferencian los tópicos de los mensajes y es posible introducir un tópico sin necesidad de un mensaje (solo el asunto del tópico) independientemente de cual alternativa se tome en el diseño se decidió durante el análisis mantener la postura más flexible ya que con esta se pueden desarrollar ambas soluciones.

Además del tópico a que pertenece, un mensaje puede ser la contestación de algún mensaje anterior por lo que otra asociación es la una instancia de mensaje con otra instancia de mensaje a la que se está contestando (ésta es una asociación de un concepto consigo mismo) un mensaje solo puede contestar a un solo mensaje pero muchos mensajes pueden contestar al mismo mensaje.

El usuario invitado (ó en general cualquier usuario) tiene una relación con los mensajes (ó foros ó tópicos) que está consultando en un momento dado, pero esta relación a nuestro juicio no merece ser conservada[8] por lo que no se incluye en nuestro modelo conceptual.

En cambio la relación de un usuario con los mensajes que él ha generado sí es una asociación que es provechoso conservar. Similarmente se considera a la relación entre el usuario y los tópicos que ha creado; el usuario puede ser autor de cero o muchos tópicos y mensajes.

Los moderadores y administradores tienen una asociación con todos los foros que ellos moderan (valga la redundancia). Cada moderador puede estar asociado con uno o muchos foros; de hecho es importante resaltar que un usuario es moderador solamente para los foros que tenga asociados para serlo. Por lo que debemos tener en cuenta que el rol de moderador no es permanente sino digamos contextual, esto es que depende del foro que se este considerando. En cambio el administrador es administrador en todo momento[9].

Habíamos mencionado que un usuario puede suscribirse a cero o muchos foros de discusión, sin embargo dado que introducimos el concepto suscripción la relación se dará entre el usuario y la suscripción así como entre la suscripción y el foro[10].

Otra asociación la encontramos entre usuario y datos personales, que como en algún momento dijimos, puede darse el caso que un usuario tenga más de un perfil; es esta etapa de análisis consideramos que un usuario debe tener al menos un perfil pudiendo llegar a tener

[8] Sin embargo este argumento no es absoluto, podría requerirse saber cuales son los mensajes que un usuario en particular ya consultó, para diferenciar o dar un trato especial a aquellos que ya fueron leídos ó a aquellos que no lo han sido, en nuestro caso no existe tal necesidad expresada en los requerimientos por lo que desde este momento descartamos esta posible asociación.

[9] Inclusive si no existe ninguna instancia de foro, por esto la multiplicidad del administrador con respecto a los foros es de cero o muchos.

[10] En este caso el concepto suscripción representa una asociación con multiplicidad de muchos a muchos. En los modelos de entidad-relación muchas veces se introducen entidades cuya única función es evitar las asociaciones muchos a muchos (los cuales no son directamente traducibles a una representación del modelo en, por ejemplo: una base de datos relacional)

muchos, sin embargo esta flexibilidad podría o no considerarse necesaria de conservar durante el análisis por lo que podría cambiarse después la multiplicidad de esta asociación.

Dado que el resumen diario depende de las preferencias de cada usuario existe una asociación entre usuario y resumen diario dado que por cada resumen diario se genera un mensaje de correo electrónico tenemos una asociación de uno a uno entre estos dos conceptos.

Finalmente y dado que el mensaje de correo electrónico va dirigido a algún usuario tenemos otra asociación en la que por cada usuario puede haber cero o muchos mensajes.

Con estas relaciones resulta oportuno plasmar nuestra primera versión del diagrama del modelo conceptual.

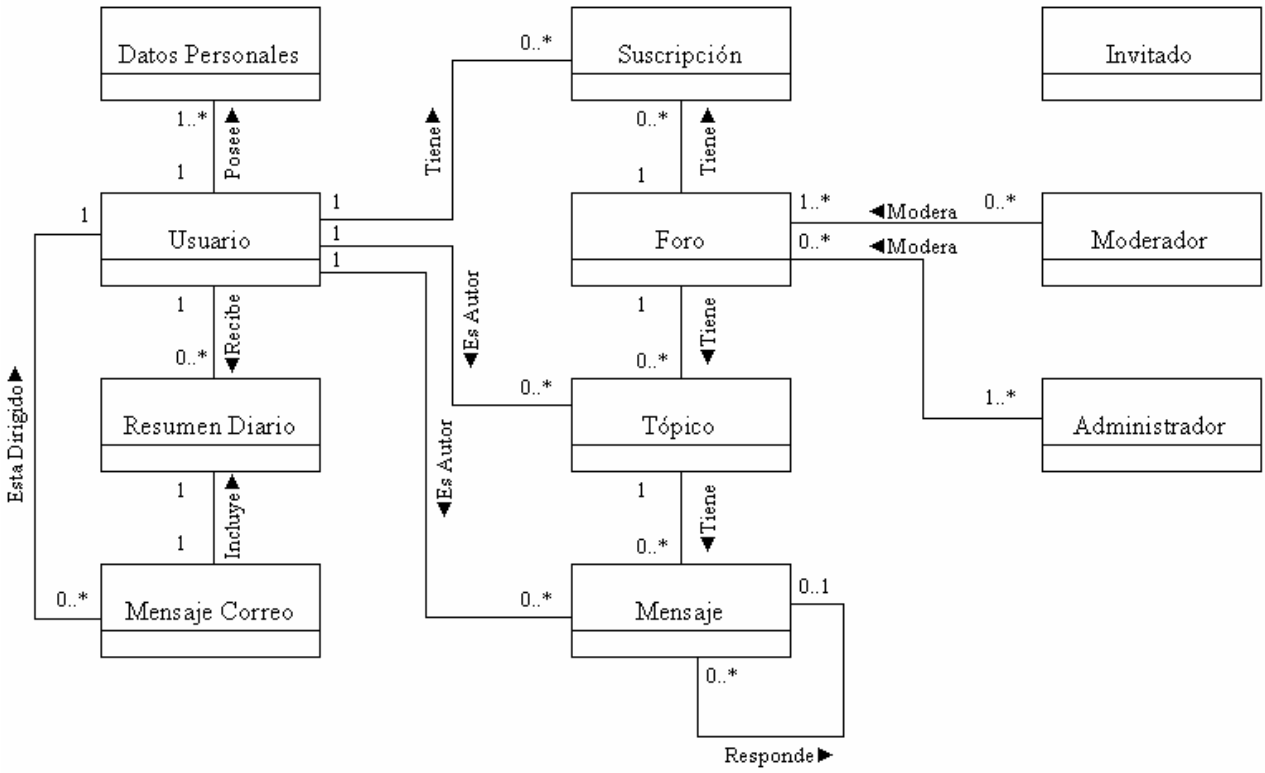


Diagrama 1.6
Modelo Conceptual versión 1.0

1.3.3 Identificación de los atributos de los conceptos.-

El siguiente paso en el proceso de elaboración y refinación del modelo conceptual consiste en la búsqueda y asignación de los atributos de los conceptos que hemos encontrado. Los atributos son propiedades o datos de los conceptos cuyo valor tiene un significado relevante en nuestro modelo, esto es que si bien los objetos reales pueden tener infinidad de atributos, solo nos deben interesar aquellos cuyo conocimiento incida en la lógica o el comportamiento del concepto en particular o del modelo en general.

En ocasiones, cuando un concepto está asociado con otro y se tiene conocimiento de esta asociación, puede inducir a confusiones en cuanto a querer representar dicha asociación como si fuera un atributo, pero esto es incorrecto[11]. Por lo general los atributos deberían estar restringidos solo a tipos primitivos de datos (booleano, fecha, número, cadena de texto, hora, etc.).

Como regla ningún atributo debe de ser utilizado en el modelo conceptual para relacionar dos conceptos, generalmente se incurre en este error pues se empiezan a contemplar cuestiones de diseño en el análisis, incluyendo por ejemplo llaves foráneas como atributos. Esto es incorrecto pues dado que existen muchas maneras de expresar una relación entre dos objetos, el análisis no debe de promulgar por alguna solución en particular sino posponer esta decisión hasta el momento de diseño[12]. En cambio si son válidos los atributos que determinan la multiplicidad de una relación (cantidad, monto, etc.).

Por otra parte los atributos que representan una llave primaria se recomienda evitarlos en los modelos conceptuales cuando representan una forma de implementación particular que no es forzosa en ese dominio, por ejemplo una llave primaría cuyo significado no sea otro que un valor (por ejemplo un número) sería recomendable no incluir en la elaboración del modelo conceptual, por el contrario si tenemos una clave o identificador que para este dominio es inmutable probablemente sería buena idea incluirla como atributo, sin embargo, debemos de reconocer que identificadores que en un momento pueden parecer perpetuos para un dominio con el tiempo terminan no siéndolo[13].

---

[11] El hecho que durante el diseño y la construcción, las asociaciones sean representadas comúnmente como atributos de tipos de datos abstractos (en especial clases) no debe influir que para en el modelo conceptual esta relación sea expresada como una asociación y no un atributo.
[12] Recordemos que el análisis es a final de cuentas el estudio del problema, no de la solución.
[13] Por ejemplo el RFC para identificar personas ahora esta siendo desplazado por el CURP.

Otra consideración importante es que hay varios niveles de importancia en los atributos, algunos podemos decir que son imprescindibles para representar a un sistema real, pero otros no lo son tanto, por ejemplo la dirección de correo electrónico (o simplemente email) de cada usuario es un atributo indispensable para cumplir con el caso de uso “El sistema envía por correo un resumen diario de los nuevos mensajes”, en cambio los apellidos del usuario no lo son tanto.

En una aplicación orientada a objetos las clases finales resultan tener los atributos que durante la elaboración del mismo se consideró que era importante conservar, sin embargo hasta la etapa del análisis en el modelo conceptual es común que el número de atributos por concepto sea mucho menor, esto es por que un buen modelo conceptual capta las abstracciones esenciales y la información indispensable para comprender el dominio, como es de suponerse hay algo de subjetividad en esta última oración pues para un equipo de análisis lo que es esencial puede diferir de lo que otro equipo considera.

Teniendo lo anterior expuesto en cuenta, nuestro modelo incluimos a nuestro modelo conceptual los atributos que consideramos esenciales en el contexto de nuestros requerimientos, como vemos en el siguiente diagrama del modelo:

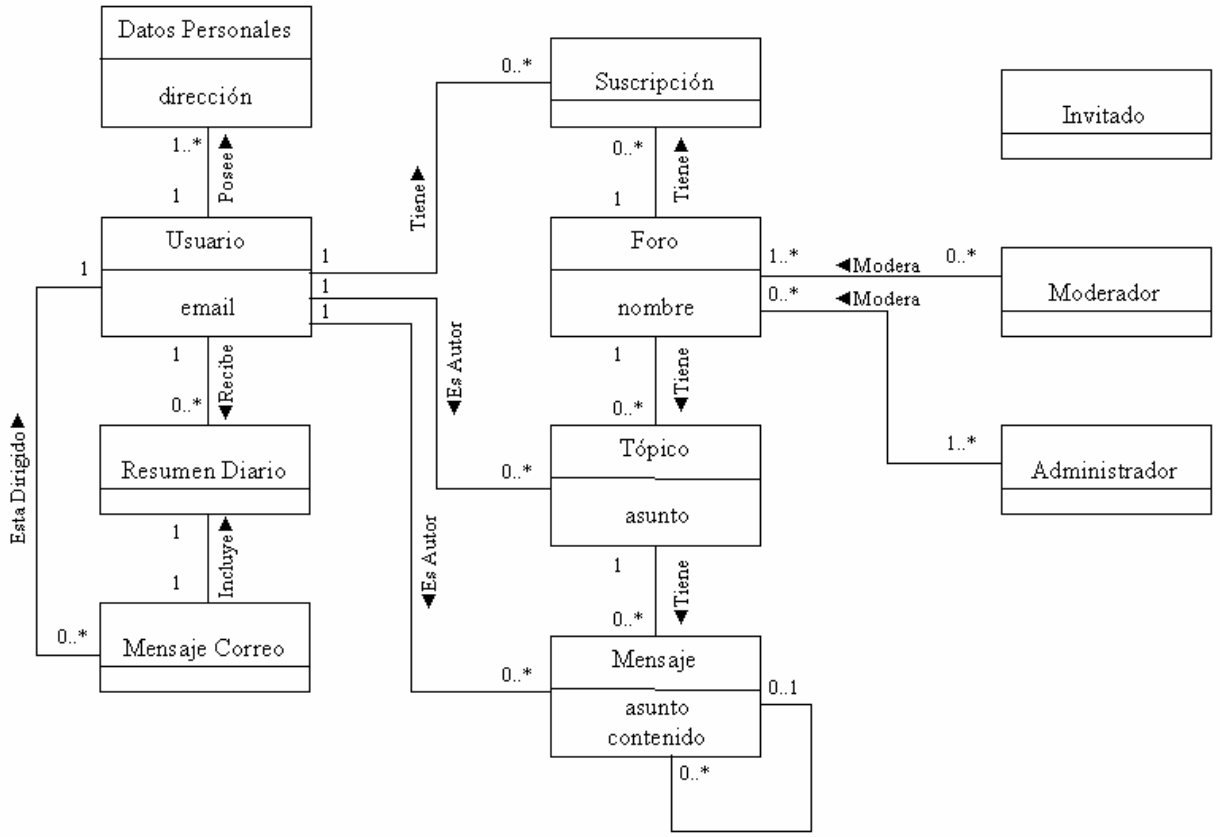


Diagrama 1.7
Modelo Conceptual versión 1.1

1.4 Análisis del Comportamiento.-

1.4.1 Diagramas de Secuencia del Sistema.-

El modelo conceptual es una poderosa herramienta que nos muestra gráficamente la estructura e interrelación de los componentes del sistema real, sin embargo el modelo no da ninguna indicación de la forma en que los casos de uso son resueltos por el sistema, no hemos visto hasta ahora ningún artefacto que nos describa cual es el comportamiento del sistema y que protocolo o secuencia de procesos se siguen durante un caso de uso. En este momento es cuando abordaremos este importante aspecto y una de las herramientas para hacerlo lo es los diagramas de secuencia del sistema.

Los diagramas de secuencia del sistema son elaborados utilizando la notación UML para los diagramas de secuencia pero con la particularidad que en vez de contemplar las secuencias entre un subconjunto de clases del sistema (o en nuestro caso conceptos), se

elabora considerando al sistema en su conjunto como una entidad completa (como una caja negra).

Los diagramas de secuencia desarrollan un escenario o trayectoria para un caso de uso[14], mostrando gráficamente los eventos disparados por los actores y los eventos que el sistema a su vez genera, ordenando la serie de eventos de acuerdo al instante de tiempo en que son generados. Los eventos del sistema mostrados pueden incluir parámetros cuando estos clarifiquen el comportamiento del sistema.

En nuestro proyecto de foros de discusión en particular la mayoría de los casos de uso no consisten más que en una petición por parte de algún actor (principalmente el usuario) y una respuesta por parte del sistema, esto es que hay pocas secuencias de eventos, esto se debe en parte al hecho de que los desarrollos para Internet-Intranet generalmente están muy poco orientados a mantener una continuidad en la interacción entre el usuario y la aplicación[15] y más orientados en cambio al esquema Request-Response en donde se solicita cierto contenido y la aplicación responde generando ya sea una página HTML u otro tipo de información.

Si bien hemos dicho y con razón que durante el análisis debemos tener cuidado de no sesgar nuestro modelo conceptual hacia ningún esbozo de solución en particular, por otra parte, es un hecho que los requerimientos de nuestro proyecto son cerrados en principio. Esto es que nuestra aplicación no está planeada de ser implementada en un entorno distinto al del HTML estándar, incluso el nombre, descriptivo es de “foros de discusión a través del web”, sin embargo el argumento principal es que el propósito de esta tesis es comparar y evaluar dos plataformas de desarrollo de contenido dinámico para Internet-Intranet por lo que tiene poco sentido mantener un modelo conceptual muy genérico y susceptible de ser aprovechado por soluciones adicionales al contexto de las páginas web[16].

Por lo tanto consideramos que no vale la pena describir secuencias de interacción demasiado elaboradas que seguramente en el diseño serían simplificadas y circunscritas al esquema Request-Response. Revisando nuestros casos de uso tenemos que la mayoría de los mismos

[14] Normalmente se desarrollan solo los cursos normales de los casos de uso o algunas variaciones que es importante documentar.
[15] Con sus excepciones por supuesto.
[16] Vale la pena considerar que los sistemas de foros de discusión históricamente han estado acoplados a las tecnologías de Internet desde los BBS, usenet, etc. Fuera del ámbito de Internet normalmente se dispone de herramientas más poderosas y eficientes para desarrollar sistemas de mensajería y colaboración para el intercambio de opiniones e información.

implican solamente que la petición del usuario y el sistema debería generar una página con el contenido solicitado, hay sin embargo algunos casos de uso que requieren un poco más de interacción entre el usuario y el sistema. Como ejemplo mostramos los diagramas de secuencia de cuatro de estos casos de uso:

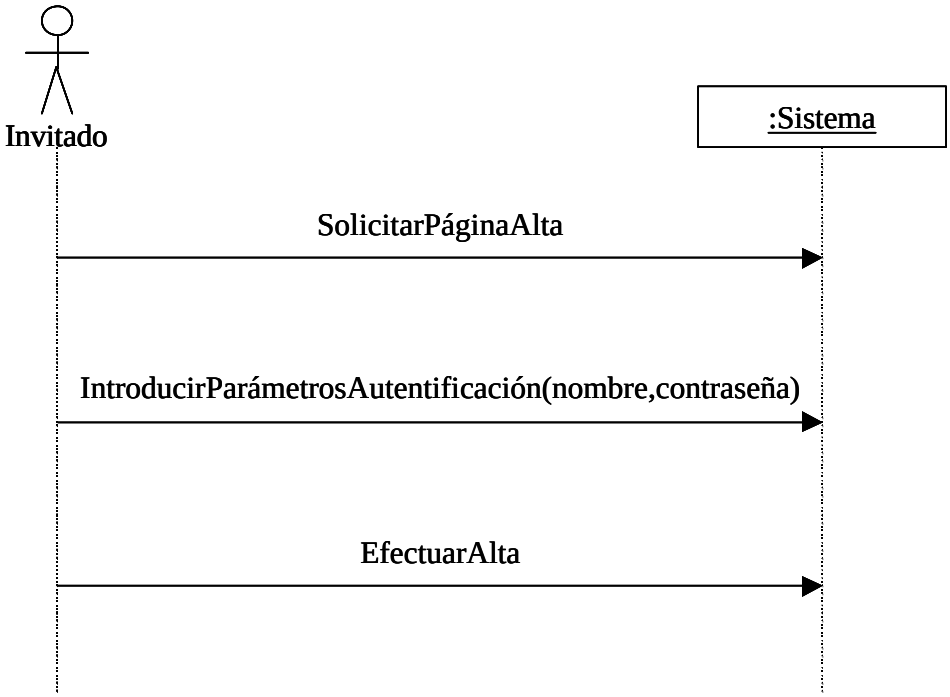


Diagrama 1.8
Diagrama de Secuencia del Caso de Uso: Un usuario no-registrado se da de alta en el sitio

Para nuestro proyecto de foros de discusión, cuando un usuario empieza a solicitar páginas de nuestra aplicación y si no se ha identificado todavía se le trata como un invitado, siendo invitado puede navegar a través de los foros consultando tópicos y mensajes. Sin embargo para realizar otro tipo de acción (digamos responder un mensaje) necesita estar registrado en el sistema. Si nunca lo había hecho debe de darse de alta en el mismo a través de la secuencia del diagrama anterior; por otra parte, si alguna vez se dio de alta, el sistema debe de conservar su información y por lo tanto lo único que debe hacer el usuario es: Registrarse indicando determinada información que sea suficiente para que el sistema lo identifique, como se muestra en el siguiente diagrama:

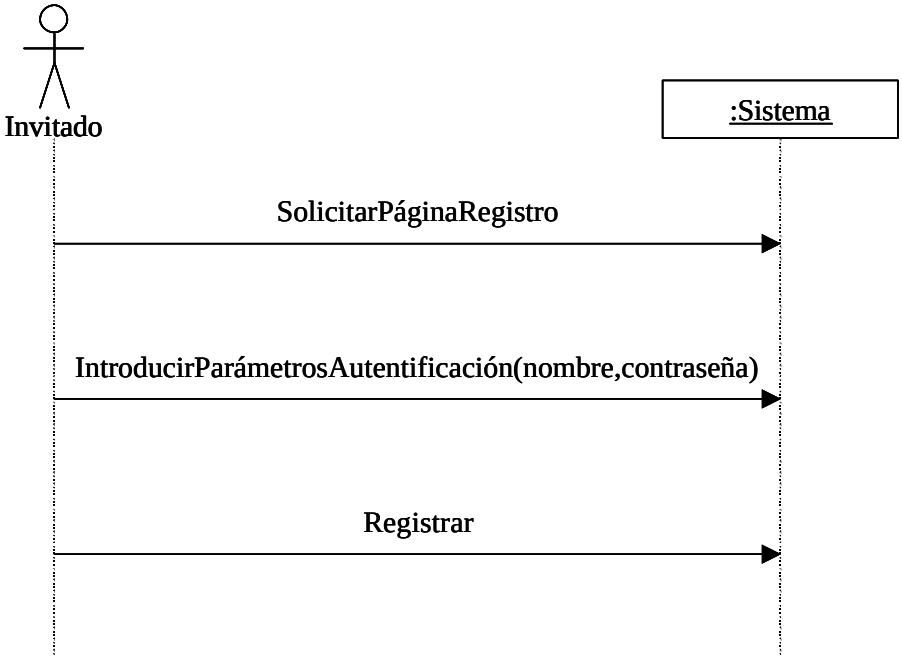


Diagrama 1.9
Diagrama de Secuencia del Caso de Uso: Un usuario no-registrado se registra (identifica)

Un usuario que ya se ha identificado (ya sea usuario normal, moderador de algún foro o administrador) puede cambiar sus datos personales (perfil) siguiendo la siguiente secuencia:

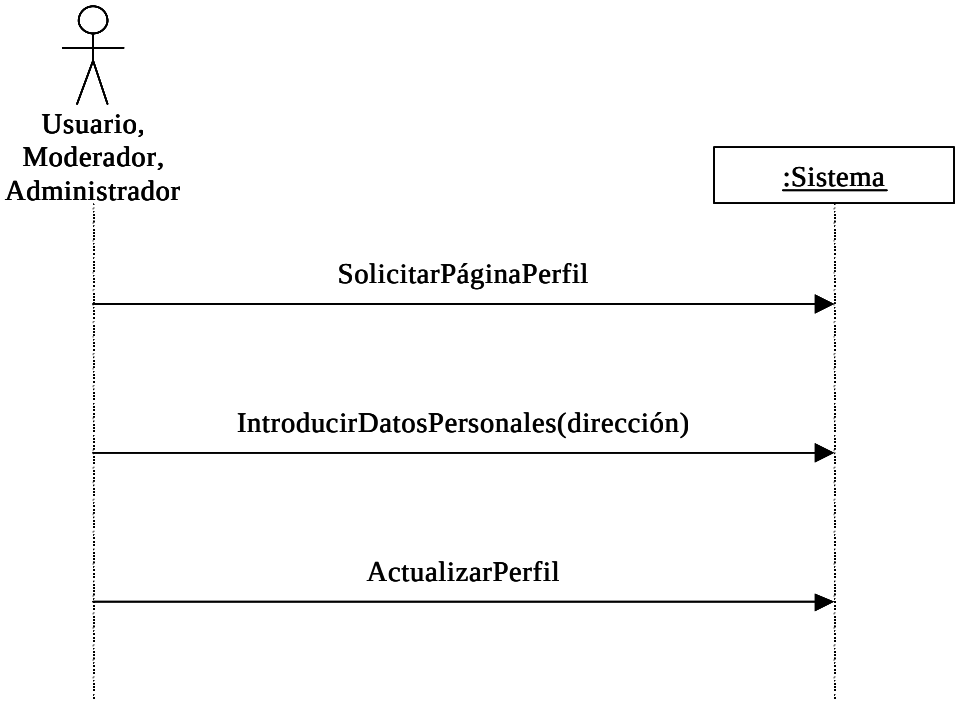


Diagrama 1.10
Diagrama de Secuencia del Caso de Uso: Un usuario modifica sus datos personales (perfil)

Todos estos diagramas de secuencia han sido muy similares, esto es que para desarrollar el caso de uso el sistema requiere de un paso intermedio para obtener una serie de parámetros. Para poner un ejemplo de una interacción más compleja podríamos tomar el caso de uso "El moderador de un foro depura los mensajes inactivos de hace más de 3 meses"[17], a continuación mostramos la secuencia de este caso de uso:

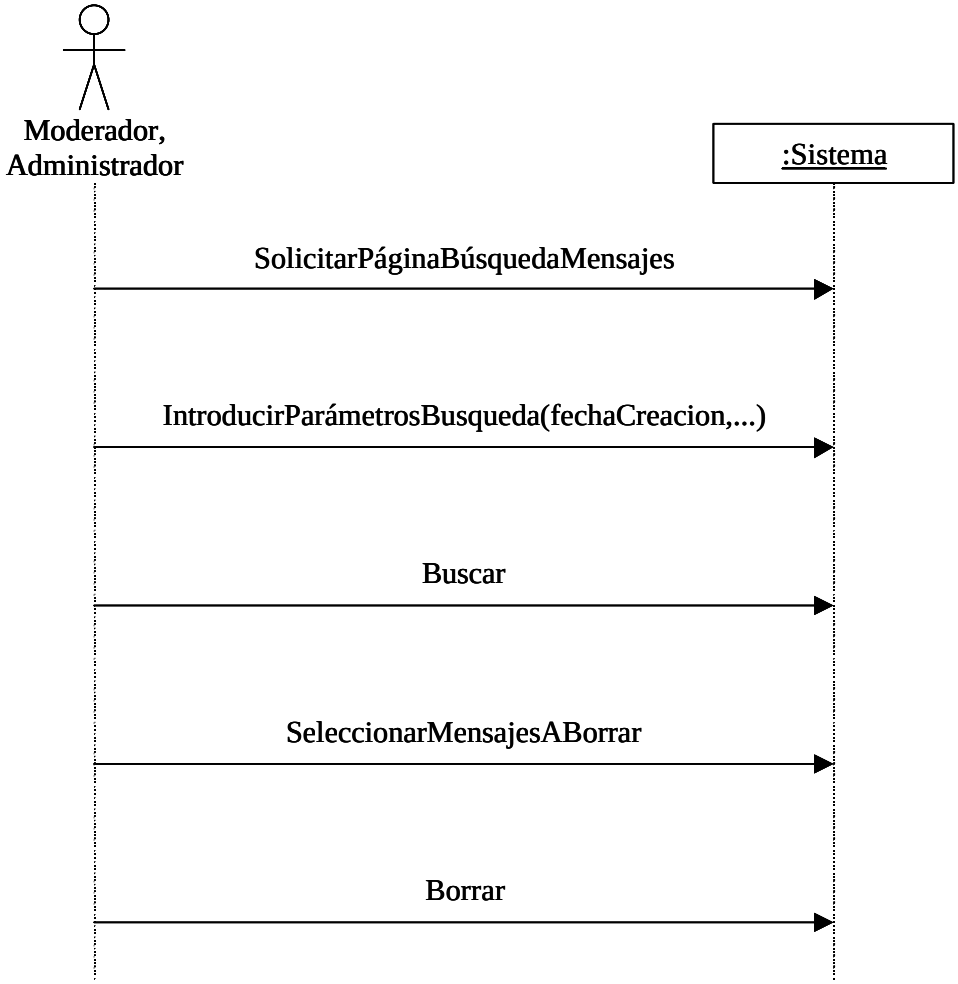


Diagrama 1.11
Diagrama de Secuencia del Caso de Uso: El moderador depura mensajes

Con estos ejemplos se ha pretendido mostrar el estilo de interacción entre los usuarios y el sistema basado en el esquema Request-Response; como se mencionó previamente, muchos de los casos de uso implican simplemente una solicitud por parte del usuario y la respuesta es una página con el contenido deseado, en otros casos de

[17] Que como ya mencionamos es la implementación de dos casos de uso: Uno para buscar mensajes de acuerdo a su fecha de creación (o bien podría ser de acuerdo a otra propiedad) y el Segundo que es el de borrar mensajes.

uso este mismo esquema se mantiene pero con la salvedad de que se tienen eventos intermedios en los cuales el usuario envía parámetros requeridos por el caso de uso. Con los diagramas incluidos, pensamos que es suficiente para ejemplificar las tareas requeridas en esta etapa del análisis del comportamiento.

Recapitulando lo realizado en este capítulo tenemos que hemos enumerado una lista de casos de uso que al parecer son suficientes para cumplir con nuestros requerimientos, además y principalmente hemos construido un modelo conceptual en el que son captadas las abstracciones clave de nuestro problema así como algunos de sus más fundamentales atributos; finalmente elaboramos los diagramas de secuencia que describen los escenarios normales de algunos de los casos de uso más representativos del sistema.

Con lo hasta ahora alcanzado consideramos que es suficiente para poder continuar con la elaboración del sistema ahora en su etapa de diseño, esto es que habíamos estudiado hasta este punto solo al problema (el “que”) y ahora empezaremos ha encontrar la solución (el “como”), sin embargo queremos mantener una metodología flexible en el sentido de que podamos durante la etapa de diseño evaluar, refinar e incluso redefinir algunos de los artefactos (casos de uso, modelo conceptual, etc.) elaborados en el análisis. Entonces, mas que concluir la etapa del análisis, debemos decir que empieza ahora la etapa del diseño.

CAPÍTULO 2. DISEÑO

2.1 Introducción al Diseño.-

En el presente capítulo se abordará el diseño de nuestro proyecto de foros de discusión a través del web. El propósito fundamental del diseño es encontrar una solución lógica apoyada en la comprensión adquirida durante el análisis del sistema real. Al igual que en el análisis, durante el diseño son elaborados una serie de documentos o artefactos que plasman la labor realizada y los conocimientos adquiridos en esta importante etapa del proceso de desarrollo de un sistema.

Así mismo, en este capítulo se refinará el modelo conceptual generado en la etapa del análisis. El modelo conceptual nos permitirá partir hacia la definición del diagrama de clases; se detallarán los casos de uso los cuales se traducen en casos de uso reales[18]. El siguiente paso será la obtención de las clases de diseño las cuales quedarán plasmadas en un diagrama de clases, el diagrama de clases representa el producto final del diseño de la estructura del sistema.

A diferencia del análisis, en el diseño sí son consideradas las características de la plataforma donde será implementada la solución. Sin embargo tenemos el propósito de realizar un diseño genérico hasta donde sea posible, dado que la construcción del sistema se realizará en dos plataformas (ASP y JSP), cuyas características no son enteramente equivalentes. Por lo tanto, el diseño del comportamiento del mismo, el cual normalmente es plasmado en diagramas de interacción y/o secuencia entre clases (los cuales se basan en los diagramas de secuencia del sistema), se abordará en capítulos posteriores; en los cuales se emprenderá la construcción y codificación en estas dos plataformas. Esto debido a que mucha de la funcionalidad de los casos de uso estará resuelta por clases muy acopladas a la plataforma de implantación (capa de persistencia, capa de visualización, etc.).

Además de las consideraciones que implicará la implantación en estas dos plataformas, deberemos considerar otro aspecto de igual importancia, que es la persistencia. Por requerimientos del proyecto, el almacenamiento de la información se realizará en una base de datos relacional, por lo que necesitaremos de un conjunto de clases que nos

[18] Se convierten en casos de uso reales pues tienen una correspondencia directa con algún proceso o secuencia de procesos que será desarrollada por el sistema, a diferencia de los casos de uso del análisis cuya correspondencia es con los requerimientos del sistema.

brinden los servicios de persistencia; las cuales idealmente deberían de permitirnos abstraer y desacoplar a las clases del dominio de dicha responsabilidad. Junto con el conjunto de clases de persistencia se desarrollará el modelo de la base de datos en donde se establecerá un mapeo entre las clases y asociaciones del modelo conceptual y las entidades y relaciones del modelo de base de datos.

Tomando en cuenta lo anterior, el diseño de las clases que contengan un acoplamiento considerable con la plataforma de implementación (ASP/JSP y base de datos) se intentará posponer hasta la etapa de construcción o al menos su diseño incluirá solamente los elementos que se necesiten, procurando minimizar tanto como se pueda el mencionado acoplamiento. En especial cuando estas clases no sean parte del dominio del sistema, sino abstracciones inventadas en la solución.

2.2. Consideraciones Arquitectónicas y Patrones de Diseño.-

El desarrollo de software es por naturaleza una actividad creativa que permite a los desarrolladores implementar múltiples y muy variadas soluciones a un problema. La construcción de software brinda una flexibilidad extraordinaria comparada a otras actividades de creación intelectual. Pero si bien esta flexibilidad puede llegar a seducirnos, no hay que perder de vista que a final de cuentas un proyecto de desarrollo de software tiene un propósito específico e implica costos y riesgos. Por lo tanto se debe intentar minimizar el riesgo de elegir o construir soluciones poco adecuadas, muy costosas o de plano equivocadas. Dado que en la conformación de un equipo de desarrollo no se cuenta con la garantía de integrar solamente a los personajes ideales para cada función, se deben buscar técnicas y herramientas que minimicen el riesgo de que el equipo tome decisiones equivocadas que al final resulten en un sistema de software que no sea lo suficientemente o deseablemente adecuado.

Una de estas técnicas los constituyen los patrones de diseño[19] de software. Los patrones de diseño de software son soluciones que previamente han probado su efectividad en algún diseño de software (normalmente afrontan problemas frecuentes) y que es válida su aplicación a otros diseños con contextos similares. En nuestro proyecto

[19] Las metodologías de análisis, diseño y programación orientadas a objetos constituyen es si mismas otras técnicas para disminuir este riesgo.

en particular, afortunadamente tenemos disponibles varios patrones de diseño viables de ser utilizados.

Antes de empezar con la búsqueda de las clases y sobretodo antes de asignar las responsabilidades a las mismas es importante elegir una arquitectura adecuada. La arquitectura de un sistema de software está constituida por una serie de principios que debe cumplir un proceso de diseño de software. Dichos principios tienen como finalidad el encaminar la solución por construir hacia una dirección específica que a juicio de los impulsores de dicha arquitectura promueve la obtención de un resultado satisfactorio[20]. Como era de esperarse, una buena parte de los patrones de diseño de software están enfocados hacia la arquitectura de los sistemas.

Mientras que en el análisis nos ocupábamos de abstracciones que tenían que ver solamente con el dominio del problema, en el diseño, las abstracciones o mejor dicho las clases con las que trataremos serán tanto del dominio, como de otros entornos, esto es por que algunas clases deberán interactuar con la infraestructura de implantación, algunas otras deberán intermediar con los usuarios del sistema, otras deberán coordinar la acción de las demás clases, etc. Si bien es posible restringir las clases del sistema solo a aquellas que correspondan al dominio del problema, esta alternativa es poco recomendable por muchas razones entre las cuales tenemos las siguientes:

- Las Clases serían demasiado grandes.
- Serían difíciles de codificar, depurar y mantener.
- Sería improbable su reutilización.
- Tendrían elevado acoplamiento al entorno de implantación.
- Incluirían responsabilidades que nada tienen que ver con el dominio del problema (por lo tanto el modelado que hagan del mismo sería cuestionable).
- Su robustez muy probablemente implicaría altos requerimientos de recursos de cómputo y por lo tanto ineficiencia y pobre desempeño.

Por esto un patrón de diseño recurrente consiste en liberar a estas clases de las responsabilidades adicionales al modelado del sistema real. Este es uno de los primeros y más usados patrones de software orientado a objetos y se le conoce como Modelo-Vista-Controlador, el cual propugna por separar las clases de dominio o modelo de las responsabilidades de interactuar con el usuario, y para ello en cambio

[20] Es evidente que el juicio acerca de que si una solución es satisfactoria ó no, tiene su proporción de subjetividad.

contar con otro conjunto de clases que permitan al usuario visualizar y controlar al modelo. Idealmente las clases del dominio deberán tener cero acoplamiento con las clases de visualización y control[21]. A este patrón también se le conoce como Separación Dominio-Presentación.

La presentación es quizás uno de los factores más dependientes de la plataforma de implementación (y muy probablemente el más variable). Al separar nuestras clases del dominio del mismo logramos aislar un componente fundamental del sistema de las características específicas de la implantación. Otro factor con alta dependencia al entorno de implantación lo constituye la persistencia de la información o más propiamente dicho la persistencia del estado de nuestros objetos.

La persistencia de los objetos a diferencia de la presentación no es un aspecto independiente del modelo conceptual, en los sistemas reales los objetos perduran a través del tiempo pero en cambio, si no hacemos nada por nuestra parte, una instancia de clase desaparecerá una vez que termine de ejecutarse la aplicación que la creó. Por lo tanto si bien en el mundo real podemos decir que no se necesita nada para que un objeto perdure, el equivalente de ese mismo objeto en una aplicación de software es volátil y por lo tanto necesitamos diseñar los mecanismos para simular la perdurabilidad del objeto real en el objeto de software[22].

Lamentablemente la persistencia de los objetos esta ligada (como sería de esperarse) a los recursos de persistencia disponibles (bases de datos, sistemas de archivos, etc.). Por otra parte unos de los objetivos de la mayoría de proyectos de desarrollo orientados a objetos son el reuso y la portabilidad, y ambos son afectados fuertemente por este acoplamiento a las características de una plataforma de implantación específica. Por lo tanto existen patrones de diseño de software y arquitecturas encaminadas a disminuir o acabar con esta desafortunada dependencia.

Antes de revisar algún patrón de diseño valdría la pena resaltar que existen plataformas que automáticamente manejan la persistencia de objetos, liberando al desarrollador de lidiar con los esquemas físicos de persistencia, un ejemplo notable sería la plataforma alrededor de los Enterprise JavaBeans en donde el contenedor de los EJB provee de los

[21] Por supuesto se requiere de algún tipo de acoplamiento, pero lo que se recomienda es que el acoplamiento se desde las clases de visualización y control hacia las clases del modelo y no al revés, esto es que sean las primeras las que tengan referencias hacia las segundas.

[22] El hecho de que la persistencia ocurra de facto en un sistema real induce a la ambigüedad al definir a la misma como una responsabilidad a ser solventada por el modelo conceptual, sin embargo basta con entender que los objetos de los sistemas reales perduran, por lo tanto en un buen modelo de dichos sistemas, los objetos deberán perdurar también.

servicios de persistencia a los llamados Entity Beans de una manera casi transparente. El beneficio de estas tecnologías radica en que el desarrollador enfoca sus esfuerzos en representar solamente las reglas del negocio y se libera del esfuerzo de desarrollar los servicios de persistencia[23] además de que se obtiene el soporte a mecanismos de persistencia sofisticados como es el manejo transaccional y la implementación de un esquema de seguridad.

Sin embargo este tipo de solución tiene también sus inconvenientes, en primer lugar acopla a las clases a una solución de terceros (esto es el servidor de aplicaciones o contenedor). En ocasiones la funcionalidad que se requiere es muy poca comparada a la funcionalidad mínima que proporciona el contenedor (seguridad, persistencia, transacciones, concurrencia, etc.) por lo que los componentes resultarían más robustos de lo necesario[24] (y por lo tanto no tan eficientes como pudieran ser). Finalmente es frecuente que en la práctica no se tenga la libertad de escoger un esquema de persistencia en particular para los proyectos de desarrollo porque probablemente los requerimientos especifiquen un medio de almacenamiento específico o se tenga que interactuar con otro sistema o base de datos legada.

Como el propósito fundamental de este trabajo de tesis es la comparación entre las plataformas JSP y ASP, el aprovechamiento de tecnologías como la persistencia manejada por el contenedor del esquema EJB tiene poca relación con dicho propósito y en cambio si implica un diseño más diferenciado para ambas plataformas. Se decidió por lo tanto no aprovechar estos servicios y en cambio implementar la persistencia por nosotros mismos. Sin embargo vale la pena señalar que en una comparativa de las arquitecturas Microsoft DNA y Sun J2EE, los servicios de persistencia incorporados a la plataforma de los Enterprise JavaBeans resultarían bastante significativos.

La solución que se decidió implementar para reducir el acoplamiento entre las clases de dominio y la plataforma de base de datos donde se almacenará el estado de los objetos del sistema es la de construir un subsistema compuesto de clases especializadas en interactuar con la plataforma de implantación específica que constituye el manejador de base de datos sirviendo de interfaz de la misma para las clases de dominio.

[23] En la práctica el desarrollo de servicios de persistencia absorbe la mayoría de los recursos de desarrollo de la capa del dominio (también es generalmente el componente más propenso a errores) y por lo tanto requiere mayores recursos para depuración y mantenimiento.

[24] Por ello se denomina a los Entity EJB como componentes de peso pesado (heavyweight components).

Este subsistema o capa proveerá de una serie de servicios relacionados con la persistencia de todos aquellos objetos cuyo estado requiera ser conservado, constituyendo de hecho un nivel de indirección que abstrae al dominio de las particularidades de una plataforma específica de persistencia (en este caso el manejador de base de datos). La capa de persistencia se encargará de lidiar con diferentes API's exclusivas o genéricas (como OLEDB ó JDBC) así como con lenguajes específicos (por ejemplo las variaciones en sintaxis del SQL soportado por varios manejadores de bases de datos) haciendo transparente a las clases de dominio la resolución de dichas diferencias.

Adicionalmente a la inclusión de esta capa de persistencia[25], se decidió aprovechar otros patrones de diseño relacionados con la responsabilidad de conservar el estado de los objetos, En primer lugar tenemos al patrón "Representación de Objetos como Tablas" el cual propone definir una tabla en la base de datos para cada clase de objeto, entonces los renglones de la tabla representan al estado de las diferentes instancias de esta clase de objetos. El mapeo de los atributos de los objetos, cuando estos son de tipos de datos primitivos (número, cadena, booleano, etc.) es inmediato, al corresponderlos con columnas de la tabla. Sin embargo muchas veces tendremos atributos que no son de tipos de datos primitivos[26].

Otro patrón de diseño es: "Identificador de Objetos" el cual evita que se dupliquen objetos en la materialización o se dupliquen registros en las tablas durante la desmaterialización o pasivación de dichos objetos[27]. Este patrón propone incluir un atributo adicional a las clases de objetos cuya función es la de identificar única y efectivamente a cada instancia del resto. Este atributo adicional normalmente sirve también como llave primaria de la tabla que mapea a la clase, por lo tanto no solo resuelve la probable duplicidad en la materialización y desmaterialización, sino que también nos permite mapear atributos no primitivos (cuyo tipo es otra clase) por medio de llaves foráneas.

Con las consideraciones anteriores en cuenta, podemos vislumbrar que la implementación de un caso de uso consistiría en términos generales en la materialización de un conjunto de clases de dominio (por medio de las clases de la capa de persistencia). La ejecución de

[25] Que en si esta basada en el patrón de diseño "indirección" el cual propone asignar una responsabilidad dada a un objeto intermedio para que medie entre otros objetos, componentes ó servicios evitando así que estos terminen directamente acoplados entre sí.
[26] Por ejemplo, las relaciones entre clases de agregación o componente normalmente se resuelven con atributos cuyo tipo es la clase relacionada.
[27] La materialización o activación es el acto de generar una instancia de clase a partir de su almacenamiento persistente y la desmaterialización o pasivación es el acto opuesto.

una serie de operaciones sobre dichas clases de dominio, la probable actualización de la base de datos y la generación de una página HTML (capa de presentación) que expondría visualmente el estado (al menos algunos atributos que sean relevantes al caso de uso) de las clases de dominio.

Es evidente que para que el sistema realice correctamente las funciones que se requieren, es decir, en términos de orientación a objetos, que los casos de uso cumplan con su contrato[28]. Necesitamos que alguna entidad se encargue de coordinar la acción de las múltiples clases que participarían en el caso de uso (incluyendo las clases de la capa de persistencia). Existe un patrón de diseño que se ocupa de este aspecto. El patrón "Controlador"[29] propone la asignación de la responsabilidad de manejar o controlar un evento o función del sistema a alguna de las siguientes alternativas:

- Una clase que represente al sistema global.
- Una clase que represente a la empresa u organización.
- Una clase que represente a un objeto del sistema real cuya naturaleza es activa (por ejemplo una persona o rol) y que en el dominio real es válido asociar con dicha función o evento.
- Una clase artificial (que no tiene que ver con ningún objeto real) encargada de manejar la correcta ejecución de la función o evento (conocida como controlador de caso de uso).

Dadas varias consideraciones que fueron efectuadas, se decidió implementar la alternativa de controladores de casos de uso, esto es que habrá una clase artificial por cada caso de uso. Este controlador se encargará de coordinar y manejar la secuencia de operaciones y mensajes entre las diferentes clases de dominio que intervienen en el caso. De manera tal que se asegure la correcta ejecución del caso de uso correspondiente.

A esta altura es pertinente bosquejar la arquitectura general de la aplicación que nos proponemos construir. Dados los requerimientos y las características intrínsecas del sistema, es evidente que el cúmulo de alternativas termina reduciéndose a unas pocas solamente. Esto por que, como ya dijimos, deberemos implementar el esquema request-

[28] Los contratos son documentos que formalizan los alcances de un elemento de software, expresando las condiciones requeridas para la correcta ejecución de una operación (precondiciones) y describiendo el estado que deberá resultar (poscondiciones).

[29] El patrón controlador (de los patrones GRASP) es de hecho uno de varios patrones de diseño que podría considerarse, también debemos mencionar al patrón Fachada que es bastante similar pues a groso modo propone simplificar la interfaz de un subsistema al encapsular las múltiples interfaces de las clases que componen el subsistema.

response. En donde en respuesta a la acción del usuario un navegador (browser) generará una petición HTTP que será interpretada en el servidor (probablemente esta petición involucre la ejecución de uno o más casos de uso) el cual como respuesta generaría una página HTML.

En general existen dos alternativas a este esquema. En la primera se le delega a las diferentes páginas (de guiones ASP ó JSP) la responsabilidad de atender una petición específica del usuario. Por lo que se le denomina "arquitectura centrada en la página". Esta arquitectura implica normalmente que dentro de cada página exista el código que se requiera para atender dicha petición, ya sea directamente o a través del acceso a otras capas o subsistemas. Podemos decir que las capas de visualización y de control se encuentran integradas en un solo elemento que es la página[30]. Si bien esta arquitectura no es necesariamente lo opuesto a una arquitectura de múltiples capas (de hecho pueden coexistir). Lo que sí es cierto es que al utilizar un diseño centrado en la página resulta en la fabricación de elementos de software complejos con tareas o responsabilidades muy variadas y por lo tanto difíciles sobre todo de mantener. Lo cual es a final de cuentas el primer problema que la arquitectura de múltiples capas pretende atacar.

En la arquitectura centrada en páginas, cada página es responsable tanto de interpretar la petición[31] (request) y basado en dicha interpretación generar una página con el contenido pertinente (response). Es decir que son dos responsabilidades que muy bien pueden implicar una serie de operaciones relativamente complejas. Al estar unidas ambas responsabilidades en un solo elemento de software (la página ASP/JSP), resulta evidente la posibilidad de que la complejidad se incremente considerablemente. Sobre todo dada la naturaleza intrínsecamente heterogénea de un proyecto de software para Internet/Intranet.

Una desventaja más de esta arquitectura centrada en páginas es el hecho de que induce muchas veces a una inevitable repetición de código, pues por ejemplo si se quiere implementar un esquema de seguridad, la implementación debe ser repetida en varias páginas. Para concluir con esta arquitectura diremos que en proyectos de tamaño muy reducido puede ser una alternativa menos costosa de realizar que alguna otra arquitectura más robusta.

---

[30] De hecho muchas veces esta arquitectura se emplea cuando no se diseña un sistema separado en capas, por lo que también es frecuente que en dichos sistemas las páginas incluyan las reglas de negocio y/o las funciones de la capa de dominio.

[31] Generalmente esto implica analizar un conjunto de parámetros provenientes de algún formulario HTML los cuales son transmitidos desde el navegador al servidor (a través del protocolo HTTP).

Una arquitectura más modular es la que se conoce como basada en despachadores o mediadores, en la cual uno o varios componentes son los que interceptan las peticiones generadas por los diversos clientes, invocan de ser necesario a la capa de dominio para efectuar alguna operación y finalmente le delegan la responsabilidad de generar la respuesta a otro conjunto de componentes especializados en la presentación.

Ésto es que esta tarea separa la responsabilidad de analizar e interpretar las peticiones (la cual la realizan los mediadores o despachadores) de la tarea de generar el contenido en HTML. Como resultado, la separación de responsabilidades está más acorde con una arquitectura de múltiples capas, además de que la labor de desarrollo puede dividirse en módulos más especializados, lo cual es positivo tomando en cuenta la especialización que normalmente se requiere en la construcción de la capa de presentación para lo cual normalmente se requiere de conocimientos y capacidades distintas a las que tienen que ver con la construcción de la capa de dominio o la capa de persistencia.

La arquitectura que hemos decidido implementar está basada en el esquema del despachador debido a la saludable separación de responsabilidades que propicia, lo cual es acorde al propósito de desarrollar un sistema de múltiples capas[32]. Por lo tanto podemos describir la arquitectura final que pensamos implementar como sigue:

1. Una clase despachadora encargada de interpretar las peticiones (request).
2. Una conjunto de clases controladoras de casos de uso
3. Un conjunto de clases de dominio encargadas de modelar el problema real y encapsular las reglas del negocio.
4. Un conjunto de clases especializadas en dar los servicio de persistencia a las clases de dominio
5. Una serie de páginas (JSP y ASP) y clases auxiliares encargadas de la visualización.

La interacción entre estos componentes en un caso de uso cualquiera sería como sigue:

[32] Una arquitectura basada en múltiples capas es por muchas razones más eficiente a una arquitectura centralizada, pero el factor más relevante para adoptar esta arquitectura en el presente trabajo de tesis es el hecho de que permite hacer una comparación más adecuada entre las plataformas ASP y JSP.

- El usuario por algún mecanismo dispara un evento que es transformado en una petición HTTP que es enviada por el navegador al servidor.
- La clase despachadora (1) estudia la naturaleza de la petición y delega su atención a alguna clase controladora de caso de uso (2).
- La clase controladora coordina la acción de las clases de domino (3) y las de persistencia (4) de manera tal que se cumpla con la ejecución del caso de uso, al finalizar la ejecución del caso, las clases de dominio reflejarán en su estado una situación probablemente relevante para el usuario. Es probable que la clase controladora requiera conocer los valores de los parámetros pasados en la petición.
- La clase controladora finaliza su participación delegando a una o más páginas (5) y clases auxiliares la responsabilidad de generar contenido HTML que refleje el estado de las clases de dominio al finalizar el caso de uso. Por lo tanto estas páginas deben tener referencias con las clases de dominio que fueron materializadas o activadas anteriormente.

2.3 Casos de Uso Reales.-

Los casos de uso reales describen con mayor detalle el comportamiento del sistema al recibir determinados estímulos por parte de los actores, abarcando determinada funcionalidad en particular. Además especifican un diseño concreto partiendo del caso de uso inicial del análisis e incluyendo, de ser necesario, tecnologías específicas (por ejemplo algún tipo de interfaz gráfica o algún API).

En nuestro caso, es evidente que requeriremos en algún momento introducirnos el entorno de las páginas de HTML e incluir dicho entorno en nuestros casos de uso como una pieza fundamental en la solución a desarrollar. En especial debemos considerar el esquema Request-Response que ya habíamos descrito.

A continuación presentaremos ejemplos de algunos de los casos de uso reales, en los cuales se presenta una trayectoria de ejecución normal con cierto grado de detalle y haciendo énfasis en el esquema Request-Response, en primer lugar veamos el caso de uso real “Un usuario no-registrado se registra (identifica)”:

<table>
<tr><td>Caso de Uso:</td><td colspan="2">Un usuario no-registrado se registra (identifica)</td></tr>
<tr><td>Tipo:</td><td colspan="2">Real</td></tr>
<tr><td>Actores:</td><td colspan="2">Usuario no-registrado</td></tr>
<tr><td>Propósito:</td><td colspan="2">Permitir que un usuario que con anterioridad se haya dado de alta (registrado por primera vez) en el sistema presente sus credenciales para su identificación con el fin de que sea tratado de acuerdo a los privilegios que le corresponden (Administrador, moderador, usuario normal).</td></tr>
<tr><td>Resumen:</td><td colspan="2">Un usuario no-registrado (invitado), encontrándose en alguna de las páginas que comprende el sistema de foros de discusión, por medio de la selección de una opción de menú, o a través de una liga de HTML o algún otro medio navega a la página de registro de usuarios. En dicha página se encontrará un formulario con una serie de campos que el usuario deberá llenar (probablemente contendría los campos nombre de usuario y contraseña) al terminar de hacerlo, el usuario oprimirá un botón (o algún otro artefacto) que sirva para remitir esta información al servidor web en donde se realizará una búsqueda en la base de datos de algún registro que corresponda con los datos introducidos. De encontrarse los registros correspondientes, el sistema deberá asumir que el usuario intentando registrarse es el mismo que alguna vez se dio de alta en el sistema y por lo tanto deberá realizar las acciones necesarias para que desde ese momento y durante el tiempo que dure la sesión de este cliente, al usuario le sean otorgados los privilegios que le corresponden. Por otra parte en caso de no encontrar un registro correspondiente en la base de datos, el sistema deberá responder informando de alguna forma de la anomalía ocurrida y probablemente desplegando la pantalla de registro para que el usuario vuelva a introducir sus credenciales de identificación.</td></tr>
<tr><td>Curso normal:</td><td>Acción de los Actores</td><td>Respuesta del Sistema</td></tr>
<tr><td></td><td>1. Este caso comienza cuando un usuario no-registrado navega a la página de registro de usuarios.</td><td>2. Presentar la página de registro la cual incluye un formulario con campos para la identificación del usuario (nombre, contraseña, etc.).</td></tr>
</table>

	3. El usuario llena los campos (nombre, contraseña, etc.) del formulario.	
	4. El usuario oprime un botón (HTML submit) para remitir la información del formulario al servidor web donde reside el sistema.	5. El sistema busca en la base de datos el registro que corresponda con los datos introducidos por el usuario. En caso de encontrarlo otorgará los privilegios correspondientes al usuario a esa sesión. De no encontrarlo el sistema indicaría esto presentando de nuevo la página de registro.

Como se puede suponer, los diagramas de secuencia del sistema elaborados durante el análisis constituyen una herramienta importante para la definición de los casos de uso reales del diseño, veamos ahora el caso de uso real “Un usuario modifica sus datos personales (perfil)”:

Caso de Uso:	**Un usuario modifica sus datos personales (perfil)**
Tipo:	Real
Actores:	Usuario ó Moderador ó Administrador
Propósito:	Permitir que un usuario que ya se ha registrado (identificado) pueda cambiar sus datos personales; los cuales pueden incluir: nombre, email, dirección, etc. Estos datos personales constituyen el perfil del usuario y se considera información pública, esto es que otros usuario pueden consultarla (la mayoría de esta información sería opcional).

<table>
<tr><td>Resumen:</td><td colspan="2">Un usuario que ya se ha registrado, encontrándose en alguna de las páginas que comprende el sistema de foros de discusión, por medio de la selección de una opción de menú, o a través de una liga de HTML o algún otro medio navega a la página de actualización de los datos del perfil. En dcha página se encontrará un formulario con una serie de campos que el usuario deberá llenar (probablemente contendría los campos: contraseña, email, dirección, etc.) al terminar de hacerlo, el usuario oprimirá un botón para remitir esta información al servidor web en donde se actualizará el o los registros correspondientes a dicho usuario con los datos introducidos.</td></tr>
<tr><td rowspan="4">Curso normal:</td><td>Acción de los Actores</td><td>Respuesta del Sistema</td></tr>
<tr><td>1. Este caso comienza cuando un usuario navega a la página de actualización de los datos del perfil.</td><td>2. Presentar la página de actualización la cual incluirá un formulario con los campos del perfil (contraseña, email, dirección, etc.). Preferentemente conteniendo los valores que hasta ese momento tenían.</td></tr>
<tr><td>3. El usuario llena los campos del formulario (ó modifica los que ya tienen valor).</td><td></td></tr>
<tr><td>4. El usuario oprime un botón (HTML submit) para remitir la información del formulario al servidor web donde reside el sistema.</td><td>5. El sistema actualizará el o los registros correspondientes a dicho usuario.</td></tr>
</table>

Ya habíamos manifestado que el caso de uso en el que el moderador de un foro depura los mensajes inactivos de hace más de 3 meses puede ser cubierto por medio de los casos de uso “Buscar mensajes y

tópicos" y "Eliminar un tópico existente". Sin embargo dado que la funcionalidad de búsqueda de mensajes será mayor que solo la búsqueda por fecha, decidimos redefinir el caso de uso real a simplemente: "El moderador depura mensajes":

Caso de Uso:	**El moderador depura mensajes**	
Tipo:	Real	
Actores:	Moderador ó Administrador	
Propósito:	Permitir que un moderador especifique algún criterio para encontrar mensajes a los cuales posteriormente el moderador puede decidirse a depurar a los mismos.	
Resumen:	Un moderador, encontrándose en alguna de las páginas que comprende el sistema de foros de discusión, por medio de la selección de una opción de menú, o a través de una liga de HTML o algún otro medio navega a la página de búsqueda de mensajes. En dicha página se encontrará un formulario con una serie de campos que representan a parámetros para realizar la búsqueda (pudiendo ser palabras clave del asunto o contenido, fecha de creación, autor, etc.) al especificar dichos parámetros el moderador oprimirá un botón para remitir esta información al servidor web en donde se realizara una búsqueda en la base de datos de los mensajes que cumplan con los criterios especificados, el servidor generará entonces una página (ó varias) con el listado de los mensajes encontrados. En dicha página el moderador tendrá la facilidad de seleccionar el o los mensajes que quiera depurar[33] (eliminar), para hacerlo, después de seleccionar dichos mensajes el moderador deberá oprimir un segundo botón para remitir al servidor la lista de mensajes seleccionados para ser depurados. El sistema ahora se encargará de eliminar de la base de datos el o los mensajes indicados.	
Curso normal:	Acción de los Actores	Respuesta del Sistema

[33] Es más eficiente que el moderador pueda seleccionar varios me nsajes a depurar y enviar dicha selección al servidor web en vez de que la depuración sea de mensaje por mensaje.

	1. Este caso comienza cuando un moderador navega a la página de búsqueda de mensajes.	2. Presentar la página de parámetros de búsqueda la cual incluirá un formulario con tales parámetros (palabras clave del asunto o contenido) fecha de creación, autor, etc.).
	3. El moderador especifica los parámetros de búsqueda.	
	4. El moderador oprime un botón (HTML submit) para remitir los parámetros al servidor web.	5. El sistema realizará la búsqueda de los mensajes que cumplan con los parámetros dados y generará una página conteniendo el listado de los mensajes indicados así como de algún medio para que el moderador pueda seleccionar uno o varios mensajes.
	6. El moderador selecciona el o los mensajes que desea depurar (Eliminar).	
	7. El moderador oprime un botón (HTML submit) para remitir su selección al servidor web.	8. El sistema eliminará de la base de datos los mensajes especificados.

Los casos de uso reales aquí presentados constituyen una muestra representativa de todos los casos de uso del sistema (de hecho la mayoría es más simple). Los siguientes pasos dentro de la fase de

diseño serán el diseño de la estructura de los objetos, el diseño del modelo de base de datos y en los capítulos posteriores el diseño del comportamiento de los objetos.

2.4 Diseño de la Estructura.-

2.4.1 Obtención de las Clases de Diseño.-

Una vez que hemos seleccionado una arquitectura adecuada para el desarrollo del proyecto, y apoyándonos en los conceptos adquiridos en el modelo conceptual del análisis, el siguiente paso dentro del proceso de diseño de la solución será la obtención de las clases de diseño. Dichas clases terminarán siendo las realizadoras de los casos de uso que hemos enumerado.

Durante el análisis encontramos un conjunto de abstracciones las cuales quedaron plasmadas en el diagrama del modelo conceptual, si la selección de estas abstracciones fue adecuada es muy probable que estas abstracciones se transformen en clases y se mantengan durante el diseño e implementación. La forma de determinar si una abstracción del modelo conceptual se debe conservar como clase es el estudio de las colaboraciones entre las clases candidatas para resolver los casos de uso. Esto es que la interacción de estas clases candidatas muestra la utilidad de conservar o no a las mismas.

Este enfoque para encontrar las clases, es por supuesto una alternativa entre muchas más, de acuerdo a las diferentes metodologías de análisis y diseño orientados a objetos. Sin embargo es importante señalar que no deja de tener algunos riesgos y desventajas. El riesgo principal es quizás que al subordinar la selección de clases de acuerdo a la utilidad funcional de los casos de uso, podemos caer en un diseño basado más en los procesos que en las abstracciones de datos[34]. Sin embargo hemos decidido mantener este enfoque centrado en los casos de uso principalmente porque a través de los mismos se fortalece que el proceso de desarrollo cumpla con las expectativas plasmadas en los requerimientos funcionales, lo cual es básico en todo proyecto de desarrollo de software.

[34] De hecho existen varios autores de metodologías de orientación a objetos que rechazan tajantemente la subordinación del proceso de análisis y diseño ante los casos de uso (Bertrand Meyer es quizás el más representativo), sin embargo el grueso de la comunidad de desarrollo orientado a objetos ha adoptado a los casos de uso como herramienta fundamental (para bien ó para mal).

Vamos pues a ejemplificar esta tarea de selección de las clases de diseño a través de la revisión de las abstracciones del modelo conceptual frente a la resolución de los casos de uso reales. En primer lugar salta a la vista que tenemos cuatro abstracciones (Usuario, Invitado, Moderador, Administrador) que representan en realidad un rol que toma un usuario real de la aplicación, esto es que por ejemplo en el caso de uso: “Un usuario no-registrado se registra (identifica)” el mismo usuario real pasa de ser un invitado a un usuario o administrador. Por otra parte un usuario puede ser moderador de algunos foros y usuario normal de otros, por lo que durante el caso de uso “Un usuario accede a un foro” es probable que el mismo usuario pase de ser usuario normal a moderador.

Con estos ejemplos hemos podido descubrir que estas cuatro abstracciones son en realidad un solo objeto que tiene varios estados, esto es que en un momento es invitado, en otro puede ser usuario, moderador o incluso administrador. Lo anterior sirve de ejemplo del proceso de búsqueda y depuración de clases durante la fase de diseño. Mediante procedimientos similares se estableció el siguiente conjunto de clases de dominio:

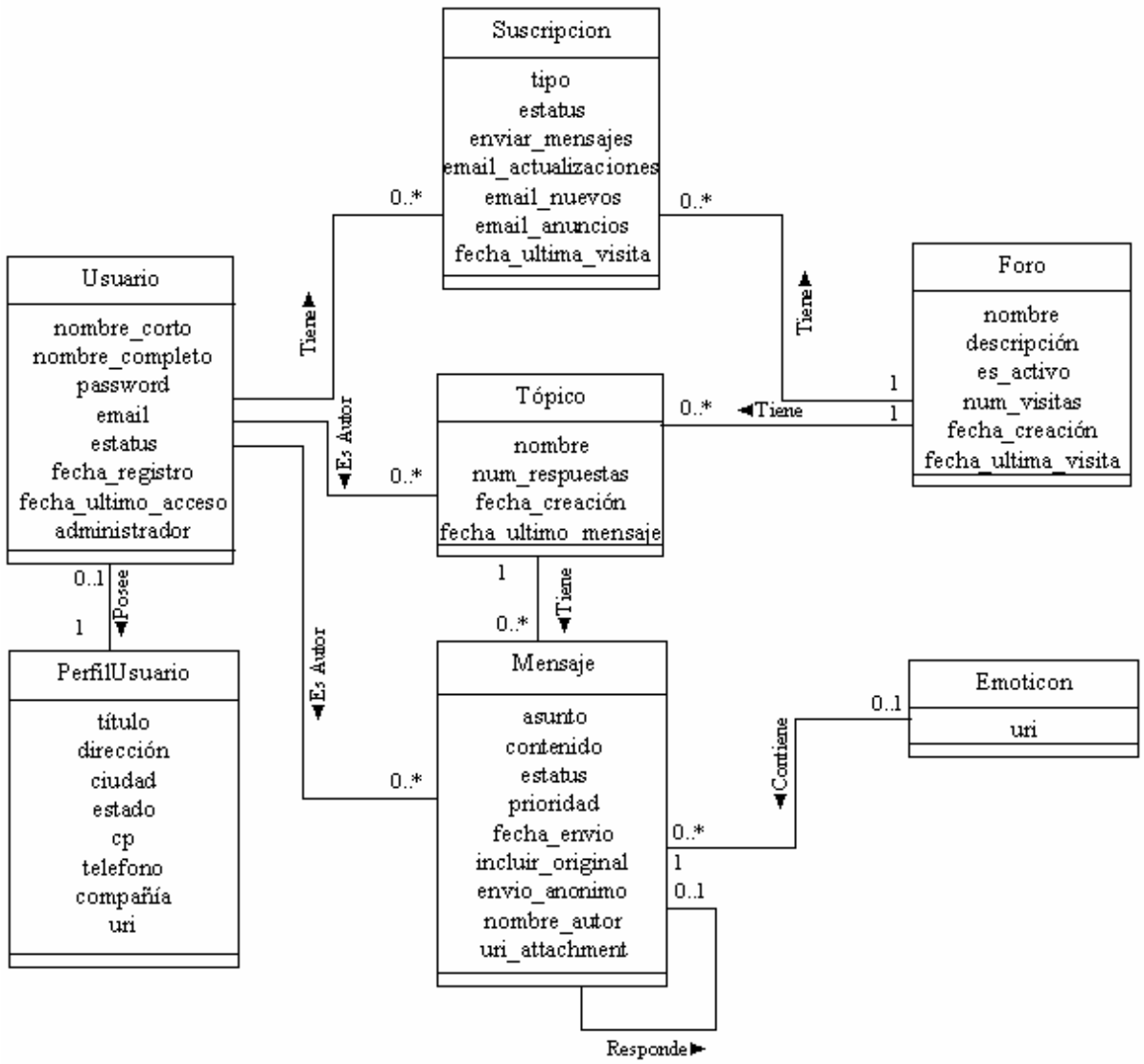


Diagrama 2.1
Clases de Dominio versión 1.0

Además de que la clase Usuario representa ahora a las cuatro abstracciones que mencionamos, otro cambio notable entre el modelo conceptual y este diagrama inicial de clases de dominio lo constituye el hecho de que las abstracciones “mensaje de correo” y “resumen diario” no son conservadas en el diagrama de clases, esto por que en esta etapa de diseño se consideró que dichas abstracciones no aportan por si mismas las suficientes razones para ser consideradas como una clase. Nos parece más adecuado que alguna clase (ya sea de la capa de dominio u otra) tome la responsabilidad de generar y enviar un mensaje de correo. Con este proceder estamos ejemplificando el hecho de que es válido descartar abstracciones del análisis durante el diseño, al igual que

es válido descartar clases encontradas en el diseño durante la implementación.

Otra diferencia importante entre el modelo conceptual frente a estas clases de dominio consiste en la inclusión de un mayor número de atributos de las clases. Mientras que en el análisis es suficiente con incluir solo los atributos indispensables para modelar al sistema real, en este diagrama de clases incluimos todos aquellos atributos que pensamos incluir en la implementación de la solución. Sin embargo es probable que durante la implementación se modifiquen algunos de estos atributos.

En este diagrama se incluyen dos clases que no fueron consideradas en el análisis. La clase Emoticon sirve simplemente para incluir un icono a un mensaje, esta clase es un ejemplo de una abstracción o entidad de tipo catalogo en cuanto a que el usuario escogería un icono en particular dentro de un conjunto de opciones preestablecidas. La clase Attachment por otra parte permitirá almacenar el URI de un archivo que se adjunte a los mensajes. Ambas abstracciones se agregan al conjunto de clases de dominio entre otras razones por que las instancias de ambas deberán incluir la capacidad de persistencia. Con la inclusión de estas clases se ejemplifica la validez de incluir funcionalidad que no fue contemplada en el análisis[35].

En esta primera versión de clases de dominio no se ha tomado en cuenta el patrón de diseño: “Identificador de Objetos”. En el que deberemos incluir un atributo adicional para identificar las instancias de estas clases de dominio. Sin embargo, además de incluir el identificador único de cada objeto, se decidió incluir los identificadores de objetos relacionados de manera tal que puedan implementarse funciones de búsqueda (finders); por ejemplo para buscar todos los tópicos de un usuario dado. Estos identificadores adicionales servirán también como llaves foráneas en el modelo entidad-relación para la base de datos.

Con las anteriores consideraciones en cuenta, presentamos ahora una nueva versión del diagrama de clases de dominio:

[35] Es normal que en etapas avanzadas de diseño o implementación surjan nuevos requerimientos en funcionalidad, normalmente el desarrollo iterativo resuelve dichos requerimientos adicionales con iteraciones adicionales, pero es común que incluso durante una iteración se decida incluir funcionalidad suplementaria a la inicialmente planeada.

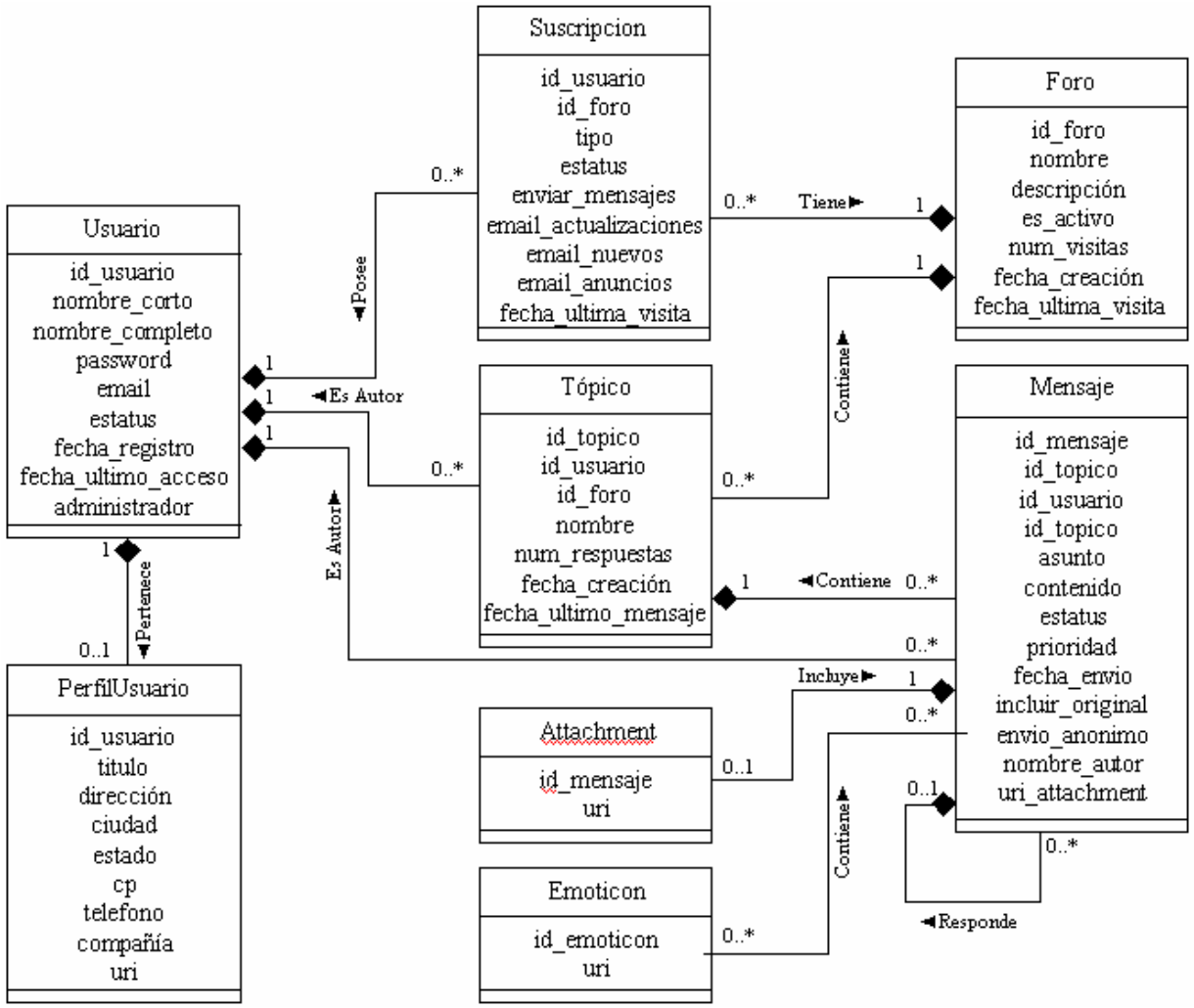


Diagrama 2.2
Clases de Dominio versión 1.1

Además de incluir los identificadores únicos de los objetos, este diagrama muestra que muchas de las asociaciones entre las clases son composiciones, esto es que por ejemplo un tópico solo puede pertenecer a un único foro, y que si se destruye el foro deben destruirse también todos sus tópicos.

Desde una perspectiva conceptual, todas las clases de diseño que hasta ahora hemos encontrado, mantienen un buen número de relaciones o asociaciones entre las mismas. Sin embargo estas relaciones no necesariamente deben de ser implementadas por algún mecanismo de composición (que pudiera ser por ejemplo un atributo cuyo tipo es el de la clase relacionada). Si bien una implementación formal de estas composiciones es más correcta desde un punto de vista estricto de la metodología orientada a objetos, por otra parte esta

alternativa produce normalmente complejos grafos de objetos cuya activación es muy costosa sobretodo para la plataforma Internet incluso utilizando técnicas como la activación retardada[36].

En una aplicación típica de Internet, uno de los principales factores a considerar es la posibilidad de alta concurrencia, ésto es que simultáneamente un elevado número de clientes pueden estar utilizando la aplicación. Por lo tanto un principio fundamental debe ser el minimizar la cantidad de recursos requeridos por sesión. Normalmente el recurso más crítico es la cantidad de memoria que una sesión requiere por lo tanto hemos decidido que en vez de implementar un diseño en el que se refleje una composición formal entre los objetos nos inclinamos por un diseño en donde los objetos se mantendrán separados. Al hacer esto facilitamos la optimización de los recursos por sesión al poder activar solamente a aquellos objetos que se necesiten[37].

Con estrecha relación a la intención de minimizar los recursos requeridos por sesión está otro patrón del diseño en el que se propone diseñar clases especiales de visualización para casos de uso específicos, por ejemplo para el caso de uso “Un usuario busca dentro de los tópicos y mensajes sobre la base de sus atributos” es de esperarse que como resultado de la búsqueda al usuario se le presente un listado de los tópicos ó mensajes que satisficieron los criterios especificados, dicho listado seguramente incluiría algunos de los atributos de la clase mensaje (pero no todos necesariamente) y probablemente requeriría de atributos de otras clases (por ejemplo el nombre de usuario del autor o el nombre del foro, etc.) si bien con las clases que hasta ahora hemos descrito es suficiente para realizar el caso de uso. Probablemente sería más eficiente contar con una clase que contuviera solamente los atributos que nos interesan, sobre todo teniendo en cuenta que una búsqueda podría arrojar un alto número de resultados.

Por lo tanto vamos a incluir dos clases especiales a nuestro diagrama a las que llamaremos: VistaBusquedaTopico y VistaBusquedaMensaje; que además de que incluirán los atributos necesarios y suficientes para

[36] En la activación retardada (lazy activation) los objetos compuestos son activados solo hasta el momento en que se van a utilizar, sin embargo, si bien este patrón de diseño mejora el desempeño en cuanto al tiempo de activación, en cambio no reduce el costo de almacenar grandes y complejos grafos de objetos en memoria; sobre todo ante las perspectivas de elevada concurrencia que puede presentar una aplicación en Internet.

[37] Por supuesto existen esquemas en donde se resuelve este aspecto de minimizar los recursos por sesión sin sacrificar un diseño formal orientado a objetos en el que se hace énfasis en la composición de objetos, sin embargo dichas soluciones normalmente requieren de servicios más o menos complejos de sincronización y concurrencia, (por ejemplo los contenedores de Enterprise Java Beans brindan dichos servicios) cuya implementación en nuestro caso va más allá del propósito de esta tesis.

construir los listados de los resultados de búsquedas de tópicos y mensajes, incluirían la responsabilidad de efectuar dichas búsquedas.

Cuando un usuario acceda a un tópico es de esperarse que se le muestre un listado con los mensajes que pertenecen a dicho tópico. Se decidió que dicho listado muestre el nombre corto y la dirección de correo electrónico del autor de manera que un usuario pueda responder por correo un mensaje, sin embargo los dos atributos pertenecen a la clase Usuario y no a la clase Mensaje. Similarmente cuando se envíe por correo electrónico el mensaje es de esperarse que se requiera indicar cual es el nombre del tópico a que pertenece. Por lo tanto se decidió incluir otra clase especial que incluya tanto los atributos de la clase Mensaje como atributos de otras clases, esta nueva clase se llamará: VistaMensaje.

Similarmente cuando los usuarios accedan a un foro se requiere mostrar un listado con los tópicos del mismo. Y también se tiene la necesidad de mostrar atributos que no son de la clase Topico por lo tanto se implementará la clase llamada VistaTopico la cual incluirá estos atributos adicionales.

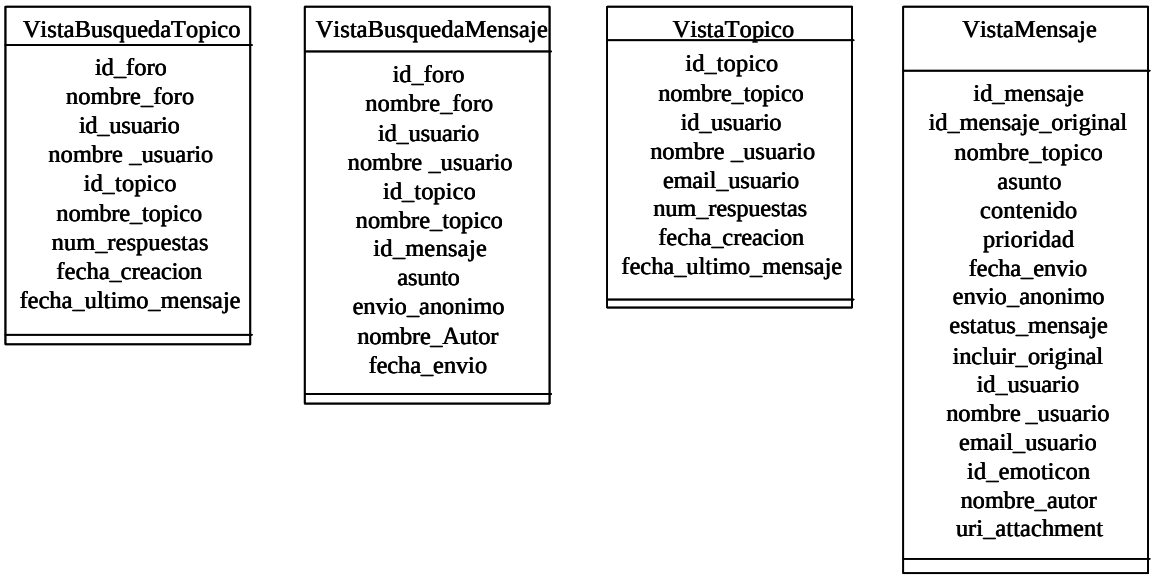


Diagrama 2.3
Clases de dominio adicionales

La naturaleza de estas cuatro clases es un poco vaga, pues su funcionalidad está directamente relacionada con la capa de visualización

más que con la de dominio[38]. Sin embargo, la estructura y comportamiento de estas clases es muy parecida al de las clases de dominio[39] que habíamos encontrado. Además estas clases requerirán de servicios de materialización realizados por la capa persistencia. Por lo tanto se decidió integrar estas nuevas clases a la capa del dominio.

Adicionalmente al conjunto de clases de la capa de dominio, es de esperarse que posteriormente se descubran e integren más clases de manera tal que los casos de uso puedan ser realizados, por ejemplo tenemos a las dos clases artificiales que introdujimos durante la elección de la arquitectura: La clase despachadora encargada de interpretar las peticiones y las clases controladoras de casos de uso. Otro conjunto de clases son aquellas que prestan los servicios de persistencia; es de esperar también que la capa de visualización comprenda otro conjunto más de clases.

Sin embargo debido al fuerte acoplamiento de estas clases adicionales a la plataforma de implantación, se decidió no incluir en esta etapa de diseño la definición de las mismas. Posponiendo dicha actividad para fases posteriores en las que se construirán tanto las clases de dominio como clases adicionales de visualización, control, y auxiliares.

2.5 Diseño del Modelo de Base de Datos.-

Como dijimos con anterioridad, la persistencia de los objetos del sistema se realizará en una base de datos relacional a la cual tendrán acceso las clases de la capa de persistencia para materializar y desmaterializar a las clases de dominio. Sin embargo dado que la persistencia se realizará en una base de datos relacional no basta con el diseño de las clases de la capa de persistencia para construir nuestro esquema de persistencia completo. Esto es que necesitamos elaborar el modelo de base de datos en el que se tendrán las tablas donde se salvaguardará el estado de las instancias de las clases de dominio.

También dijimos previamente que aprovecharíamos el patrón de diseño “Representación de Objetos como Tablas” por lo que para cada clase de dominio implementaremos una tabla en la base de datos cuyos registros representen los estados de las instancias de dicha clase.

---

[38] Esto por que la decisión de implementarse esta motivada por la necesidad de optimizar el desempeño durante la consulta de tópicos o mensajes.

[39] Principalmente por que requieren servicios de persistencia los cuales utilizarían a la base de datos para la activación de los objetos.

Finalmente planteamos la utilización del patrón de diseño “Identificador de Objetos” para evitar la duplicación durante la activación y pasivación de instancias e incluso adelantamos que los identificadores servirían también como llaves primarias y foráneas. Por lo tanto la construcción del modelo de base de datos resulta en este caso una tarea relativamente sencilla.

El modelo conceptual de base de datos es el siguiente:

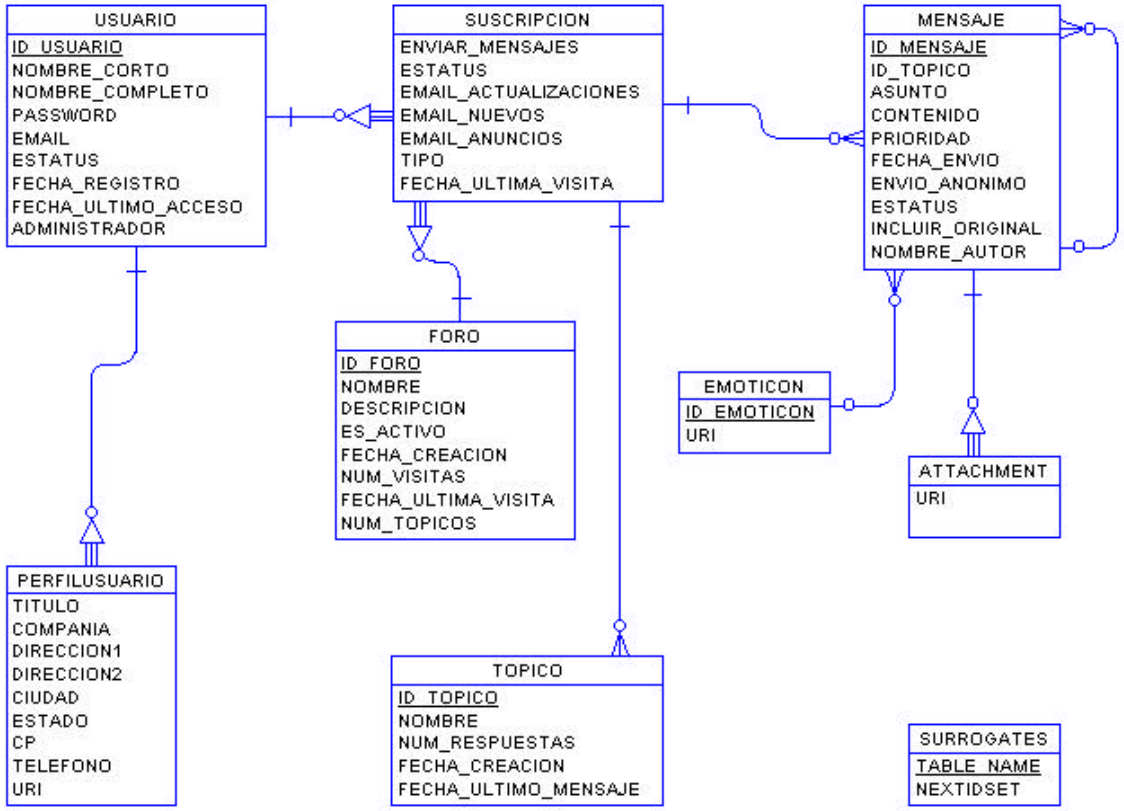


Diagrama 2.4
Modelo conceptual de base de datos

Todas las entidades de este modelo con excepción de “Surrogates” mapean directamente a alguna clase del dominio. Compartiendo los mismos atributos. En el caso de Surrogates se implementará en una tabla donde se almacenarán los identificadores de las demás tablas de manera tal que las clases encargadas de la generación de los identificadores deberán consultar y actualizar esta tabla.

Dado que es un modelo conceptual solo se muestran en el diagrama las relaciones y sus cardinalidades pero no se muestran en cambio las

columnas que implementan dichas relaciones por medio de llaves foráneas. Tampoco se muestra el tipo de dato de los distintos atributos.

En cambio en los modelos físicos de base de datos sí se especifican los tipos de datos de las columnas de las tablas y se incluyen las columnas de las llaves foráneas. En este caso incluimos el modelo físico de base de datos para Microsoft SQL Server 7.0, pero afortunadamente dado que la lógica de nuestro sistema residirá en las clases de dominio, control, persistencia, etc. La dependencia del mismo a esta plataforma específica es mínima y se buscará mantener este acoplamiento lo más reducido posible durante la construcción de dichas clases.

El modelo físico de base de datos es el siguiente:

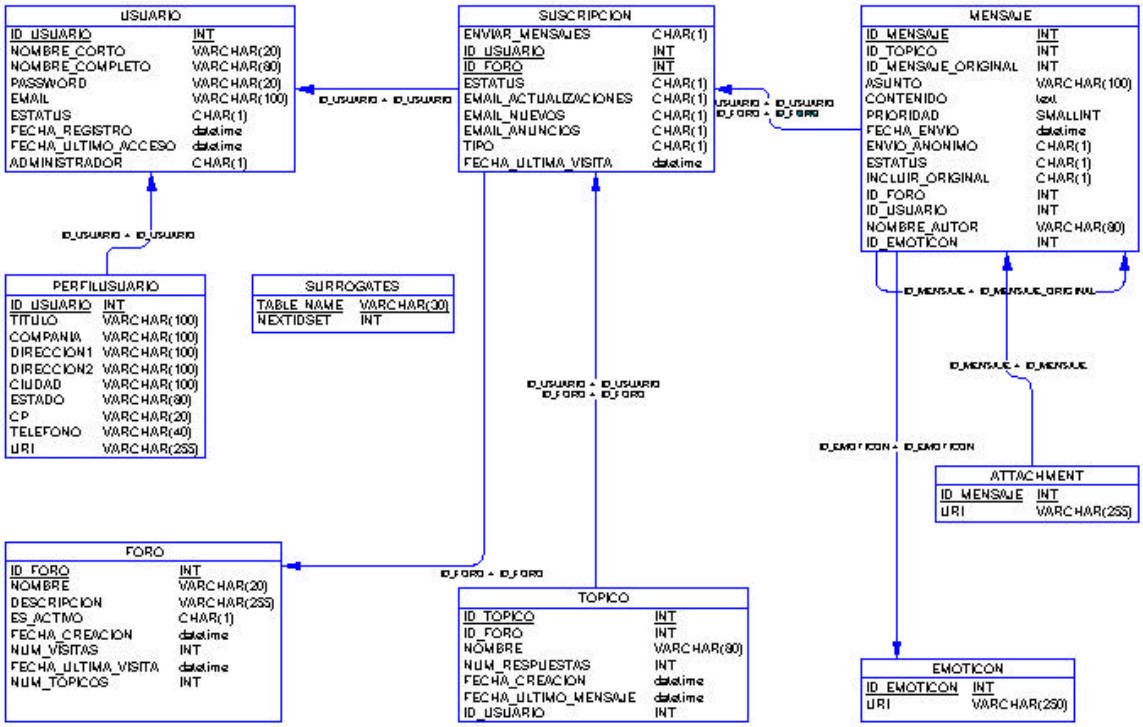


Diagrama 2.5
Modelo físico de base de datos

Con este diagrama concluimos la elaboración del modelo de base de datos. A través de este modelo resultará sencilla la obtención de un guión de generación de los objetos de la base de datos por medio de

alguna herramienta CASE. Con esto concluye también lo que pretendíamos alcanzar en el presente capítulo.

Recapitulando lo realizado en este capítulo tenemos que hemos comenzado el diseño de la solución. El modelo conceptual que desarrollamos en la etapa de análisis fue aprovechado para la búsqueda de las clases de diseño a partir de las abstracciones del modelo conceptual. El diagrama de clases de diseño constituye un producto del diseño que será aprovechado durante todo el resto del proceso de desarrollo de nuestro sistema. Así mismo se perfecciono la especificación de los casos de uso convirtiéndose en casos de uso reales los cuales incluyen elementos que auxiliarán en la posterior implementación de los casos de uso por medio de diagramas de secuencia y/o interacción entre clases.

Si bien en este punto concluimos el presente capítulo la fase de diseño no concluye pues solo hemos visto la parte que tiene que ver con la estructura de las clases pero no el comportamiento de las mismas. Por otra parte el hecho de que el comportamiento de las clases lo hayamos puesto en los posteriores capítulos donde además se abordará la elaboración y codificación de las clases no es un hecho circunstancial, sino al contrario pues se quiere hacer énfasis en que las fases de diseño y desarrollo no son en la práctica dos actividades separadas y serializadas. En vez de esto son dos actividades que se intersectan y retroalimentan y que si bien normalmente el diseño deberá empezar antes que el desarrollo pues es base del mismo, esto no implica que deberá de finalizar el primero para iniciar el segundo.

Otro producto elaborado en este capítulo lo constituye el modelo de base de datos el cual nos servirá en el desarrollo para construir el esquema de persistencia. Finalmente cabe resaltar que en este capítulo se definieron algunas de las directrices arquitectónicas que se espera seguir durante el desarrollo de la solución. Con lo hasta ahora alcanzado consideramos que es suficiente para poder continuar en capítulos posteriores tanto con el diseño como con el mismo desarrollo y codificación.

CAPÍTULO 3. DESARROLLO EN LA PLATAFORMA SERVLETS/JSP

3.1 Introducción al Desarrollo con Servlets/JSP.-

En el presente capítulo se abordará el desarrollo del proyecto de foros de discusión a través del web dentro de la plataforma de Servlets y JSP. Es en el desarrollo donde los productos obtenidos durante el análisis y diseño se consolidan en componentes de software que constituyen la solución del sistema.

En esta etapa de desarrollo se presenta la tarea de codificar los componentes de software tratando de apegarse a los lineamientos plasmados en el diseño arquitectónico del proyecto pero considerando además las características propias de la plataforma que en este caso es la plataforma Java.

Nuestro conjunto de clases que ya hemos obtenido se materializarán en verdaderos componentes de software y dado que la plataforma Java es completamente orientada a objetos, en principio la generación de nuestras clases deberá de ser un proceso natural. Todo este proceso deberá estar circunscrito a cumplir con la funcionalidad expresada en los casos de uso reales que ya tenemos delimitados.

Como se estableció en las consideraciones arquitectónicas y patrones de la fase de diseño, nuestro sistema estará dividido en varias capas o subsistemas. En la construcción de estas capas se aprovecharán las facilidades y herramientas que ofrezca cada plataforma de manera tal que el software construido cumpla tanto con los lineamientos arquitectónicos nuestros como con los estándares y lineamientos generales que impone o promueve la plataforma.

3.2. Desarrollo de la Capa de Dominio.-

El conjunto de clases que componen la capa de dominio tiene la responsabilidad de modelar el sistema al que la solución de software pretende representar o automatizar. En general la capa de dominio estará compuesta en su mayoría por clases que tienen una correlación directa con objetos reales con la probable adición de clases auxiliares

que facilitan la interacción entre las primeras y entre las clases de las demás capas del sistema de software.

La labor de codificación de las clases de dominio representa la materialización de aquellas abstracciones que fueron identificadas desde la etapa de análisis las cuales se afinaron y depuraron en la etapa de diseño hasta convertirse en una colección de clases bien definida que quedo plasmado en los diagramas de clases de dominio que presentamos en el capítulo anterior.

Durante el diseño establecimos incluso los atributos fundamentales de las clases de dominio los cuales es lógico suponer que ahora dichos atributos se convertirán en propiedades de las clases en java por desarrollar. Sin embargo en este punto debemos considerar uno de los lineamientos que marca la plataforma Java para desarrollo de aplicaciones de Internet/Intranet, el cual exhorta a desarrollar este tipo de clases siguiendo estructura del estándar de desarrollo de componentes de Java que constituyen los llamados Java Beans[40].

La especificación para los Java Beans define los estándares que debe seguir un componente en la plataforma de Java sin embargo la parte de la especificación que más importancia tiene en nuestro tipo de aplicación es aquélla que tiene que ver en la forma que se definen las propiedades de las clases, lo cual es a través de funciones especiales que permiten el acceso a los contenedores privados de las propiedades. Como dato anecdótico podemos decir que la especificación de Java Beans originalmente no estaba proyectada a las aplicaciones Internet/Intranet, sino más bien era un API pensado para la estandarización de componentes a los cuales se podía administrar dentro de herramientas visuales a las que se les conoce como beanboxes. Sin embargo con la introducción de la plataforma J2EE en especial en lo referente al desarrollo mediante Servlets y JSP's se retomó a la especificación de los Java Beans ahora utilizados en este nuevo esquema.

Veamos ahora como se materializan nuestras clases de dominio siguiendo la estructura de un Bean. En el

[40] No confundir con los Enterprise Java Beans (EJB) que mencionamos en el capítulo anterior, mientras que los EJB son componentes que aprovechan una serie de servicios del contenedor para implementar su funcionalidad y es principalmente gracias a estos servicios del contenedor que permiten una gran capacidad de escalamiento por otra parte Los Java Beans son clases que cumplen con una especificación mucho más simple y que sin embargo se ha terminado por convertir en un estándar para el desarrollo de componentes en Java.

Apéndice A tenemos el código de la clase Usuario de la que habíamos definido su estructura durante el diseño. Dicha clase cuenta con la lista de atributos que previamente habíamos definido siguiendo el esquema de los Java Beans.

Podemos ver que internamente la clase Usuario contiene variables privadas para guardar el valor de las propiedades de la clase y adicionalmente cuenta con métodos públicos de acceso a dichas propiedades[41] de acuerdo a la especificación de los Java Beans.

En el código de la clase Usuario encontramos que se hace referencia a otras clases auxiliares como por ejemplo Estatus o StrBoolean, estas clases contienen métodos estáticos y/o atributos públicos por lo que en vez de tener algo como:

estatus = 'A';

Tenemos en cambio:

estatus = Estatus.ACTIVO;

Con lo que determinamos un punto único de acceso a este tipo de constantes que normalmente se esconden en el código de los sistemas de software y que a la larga dificultad el posterior mantenimiento del mismo. En el Apéndice B tenemos la implementación de la clase auxiliar Estatus.

En un buen diseño basado en el patrón de diseño de software Modelo-Vista-Controlador sería en las clases de dominio donde se debería incluir todas las validaciones y reglas a cumplir, sin embargo en la práctica los sistemas de software normalmente se enfrentan a la disyuntiva de por un lado cumplir estrictamente con este patrón de diseño o por otro lado utilizar un diseño más laxo en este sentido, esto generalmente se debe a que resulta más sencillo o eficiente asignar algunas de estas validaciones a otras capas. Por ejemplo una validación para nuestra clase Usuario sería el prevenir que se pueda crear un Usuario cuyo nombre de usuario ya este apartado por otra instancia. Si bien es posible realizar dicha validación utilizando solamente clases de dominio, debemos evaluar el hecho de que una base de datos relacional contiene las herramientas (en este caso constraints) para llevar a cabo este tipo de validación de una manera muy eficiente.

[41] A estos métodos en la jerga de los Java Beans se les conoce como Seters Y Geters debido a que uno de los requisitos de la especificación de los Java Beans es que dichos métodos empiecen con los prefijos set y get respectivamente.

Por otra parte el hecho de que en nuestro caso las clases de dominio correrán en el servidor y para que el cliente pueda acceder a las mismas se requiere de un HTTP request (esto es que no es inmediata o constante la interacción entre el cliente y el servidor). Por esto algunas veces en diseños para Internet/Intranet algunas de las responsabilidades que estrictamente pertenecen a la capa de dominio son movidas a la capa de Visualización dada la inmediatez de respuesta de dicha capa ante la interacción del usuario del sistema. Obviamente una decisión de este tipo va en contra del patrón Modelo-Vista-Controlador[42]. Por lo tanto una estrategia relativamente socorrida con relación a este punto es la de repetir este tipo de validaciones tanto en la capa de dominio como en la de Visualización, de esta manera se gana la interacción inmediata que tiene la capa de visualización con el usuario sin despojar a la capa de dominio de dichas validaciones, sin embargo como la mayoría de las decisiones de diseño ésta también tiene sus desventajas, principalmente en este caso sería la duplicidad de código y los riesgos que esta duplicidad conlleva.

Dado que el objetivo de esta tesis es comparar las plataformas JSP/Servlets contra ASP, no incluiremos demasiadas validaciones de este tipo ni haremos énfasis en este aspecto para evitar el desviarnos de nuestro objetivo principal ante las diferencias, detalles, ventajas y desventaja que las plataformas Sun J2EE y MS DNA presentan al implementar reglas de negocio en un sistema de software.

3.3. Desarrollo de la Capa de Persistencia. -

Veamos ahora cual sería la implementación de persistencia de la clase Usuario, en este caso a través de una nueva clase a la que llamaremos UsuarioPersist. El código de esta clase se muestra en el Apéndice C.

En el capítulo anterior en le que abordamos el diseño de nuestra solución, definimos que la capa de persistencia sería la responsable de la materialización o activación y de la desmaterialización o pasivación de los objetos de la capa de dominio, por lo tanto vemos que en esta clase tenemos métodos como:

[42] Es bastante más polémico el asignar responsabilidades de validación a la capa de dominio que a la capa de persistencia dado que en realidad la persistencia es a fin de cuentas una de las cualidades de un sistema real por modelar en un sistema de software.

- *getUsuariosToSendEmail*
- *getUsuario*
- *getUsuarioLogin*

Los cuales se encargan de “activar” una instancia de la clase Usuario de acuerdo algún parámetro dado; cabe mencionar que el método *getUsuariosToSendEmail* activa todo un arreglo de instancias de Usuario. De manera similar los métodos:

- *update*
- *put*
- *removeUsuario*

Nos permiten “pasivar” una instancia de la clase Usuario, esto es que la base de datos es actualizada con una instancia de Usuario. El método *update* actualiza la base de datos con los últimos valores de los atributos para un objeto de tipo Usuario ya existente, el método *put* en cambio actualiza la base de datos con una instancia nueva de Usuario y finalmente el método *removeUsuario* elimina de la base de datos los registros correspondientes a una instancia dada[43].

Podemos observar en el código que el acceso a la base de datos es a través del API JDBC lo cual es una situación común para soluciones basadas en las tecnología Servlets/JSP (y en la plataforma J2EE en general), en cuanto al SQL utilizado, este se invoca a través del método estático get de una clase auxiliar llamada Settings, la cual permite un nuevo nivel de indirección entre una base de datos específica y la clase UsuarioPersist. Por ejemplo, la invocación Settings.get("sql.UsuarioPersist.get") debería regresar una cadena del tipo:

*‘SELECT * FROM USUARIO WHERE ID_USUARIO = ?’*

En este query podemos ver que efectivamente estamos utilizando el patrón de diseño: “Representación de Objetos como Tablas” que como dijimos propone asignar una tabla en la base de datos para cada clase de objeto en donde los renglones de la tabla corresponden a cada una de las instancias mientras que las columnas corresponden a cada uno de los atributos de la clase. En este ejemplo podemos ver que se estaría activando una instancia de Usuario por medio del atributo *id_usuario*.

[43] Dado que el atributo *id_usuario* es un identificador único de la clase Usuario el método *removeUsuario* solo necesita este valor para eliminar de la base de datos a dicha instancia, esto es no se requiere pasar como parámetro a la función un objeto de tipo Usuario.

En el código de la clase UsuarioPersist encontramos construcciones típicas del lenguaje Java como por ejemplo los bloques con sincronización, los cuales son bloques de código o funciones a las cuales se evita que se tenga un acceso simultaneo de varios hilos de ejecución en el mismo bloque. También vemos bloques con atrapado de excepciones, Los cuales son bloques que desde el desarrollo se conoce que existen probabilidades de que durante la ejecución ocurra algún tipo de error y por lo tanto en el código resultante es anticipada esta situación, indicándose que camino a seguir en caso de que dichos errores realmente ocurran.

Podemos ver también que la clase UsuarioPersist es una clase de tipo singleton, esto es que su constructor es privado y en cambio se provee de una función para acceder a una instancia única de la clase, por consideraciones que no tienen que ver con los propósitos de esta tesis se decidió implementar ésta y otras clases utilizando el patrón de diseño singleton.

Además de Settings, la misma clase Usuario y de las clases pertenecientes a JDBC (java.sql.*) en el código de UsuarioPersist se utiliza otras clases como ConnectionPool la cual provee de un pool de conexiones a la base de datos[44]. Otra clase utilizada por UsuarioPersist que es interesante ver es UsuarioIDGenerator el cual genera identificadores únicos en el atributo id_usuario para las instancias de Usuario nuevas, su código se muestra en el Apéndice D.

Se decidió implementar esta clase para la obtención de identificadores únicos en vez de utilizar algún objeto de la base de datos como por ejemplo, secuencias, identitys, columnas auto-numéricas, etc. Al implementar esta funcionalidad por nosotros mismos desacoplamos nuestra solución a una base datos en particular.

3.4. Clases de control de los casos de uso.-

Durante la fase de diseño se menciono la utilidad de contar con un conjunto de clases especializadas en controlar y dirigir a las clases de dominio y persistencia con el fin de cumplir con la ejecución adecuada de los casos de uso. Dichas clases las llamamos clases controladoras de

[44] Un pool de recursos es una infraestructura que permite a varios clientes compartir un recurso lo cual normalmente implica beneficios en desempeño y escalabilidad, en un pool de conexiones un grupo de procesos comparten una o varias conexiones lo que implica que las conexiones no son mantenidas permanente por un proceso y además que una vez desocupada la conexión esta no se cierra sino que se devuelve al pool para que otro proceso la pueda utilizar. Muchos servicios de pool de recursos son brindados por el contenedor o runtime de una aplicación, sin embargo en la implementación de nuestro proyecto en la plataforma Java se decidió utilizar un pool propio.

casos de uso y como mencionamos es muy probable que requieran conocer los parámetros del Request de HTTP para poder cumplir con su misión. En particular para la solución sobre la plataforma Java esto significa que las clases controladoras de casos de uso deberán poseer una referencia a la instancia de HttpServletRequest esta clase incluye la función getParameter que permiten obtener el valor de los parámetros por medio del nombre de los mismos.

Veamos ahora el código de la clase controladora del caso de uso “Un usuario no-registrado se registra (identifica)”, Esta clase fue llamada GetUserFromLoginCommand y su código se muestra en el Apéndice E.

Podemos ver que la clase GetUserFromLoginCommand invoca a las clases de dominio y persistencia necesarias para validar que el nombre de usuario y la contraseña presentes como parámetros en el HTTP Request. Un rasgo que resalta de esta clase es que una vez Materializado el Bean Usuario el mismo es almacenado en la instancia de la clase HttpSession lo cual garantiza que para el resto de la sesión entre el usuario y la aplicación[45], el Bean Usuario estará disponible en todo momento. Esa es toda la responsabilidad de esta clase, sin embargo la mayoría de las clases de control deberían coordinar a las clases de dominio y persistencia para después obtener y poner a disposición de páginas JSP el contenido dinámico de las mismas. Veamos otro ejemplo de clase controladora de caso de uso.

Habíamos expuesto en el capítulo anterior la necesidad de que cuando un usuario acceda a un tópico se le muestre un listado con los mensajes que pertenecen a dicho tópico. Dijimos también que dicho listado incluiría atributos de los mensajes, usuarios, etc. Por lo tanto la decisión de diseño tomada fue el definir una nueva clase de dominio VistaMensaje que incluyera dichos atributos, es evidente que la clase VistaMensajePersist sería la encargada de activar y pasivar a la primera.

En el Apéndice F se muestra el código de la clase GetAllMessagesOfTopicCommand la cual se encarga de controlar a estas dos clases con el objetivo de cumplir con uno de los casos de uso más importantes de la aplicación: “Un usuario lee los mensajes de un tópico”. Podemos ver que la clase controladora obtiene del HTTP Request el valor de id_topico a través de la función getParameter

[45] Sin embargo, almacenar información a nivel sesión tiene inconvenientes importantes, sobre todo se reduce la capacidad de escalamiento a nivel concurrencia debido a que por cada sesión concurrente se requiere disponer de los recursos (en este caso memoria del servidor) para que se pueda dar servicio a cada sesión individual. Por lo tanto se requiere determinar a cuantos usuarios simultáneos la aplicación deberá ser capaz de atender para saber si se cuenta con los recursos disponibles para hacerlo en caso que se tomen decisiones de este tipo.

posteriormente se invoca a VistaMensajePersist para activar a las instancias de VistaMensaje que pertenecen a ese tópico, cabe hacer notar que todos estos mensajes son puestos en una clase contenedora llamada MensajeTree la cual permite acceder a los mensajes de acuerdo a los mensajes que son a su vez respuestas a otros mensajes, en el Apéndice G se muestra el código de MensajeTree.

Por medio de MensajeTree cuando se visualicen los mensajes en la página JSP, se podrá mostrar la jerarquía de los mensajes de acuerdo a los que son contestación de otros mensajes. Regresando a la clase GetAllMessagesOfTopicCommand podemos ver que el objeto contenedor MensajeTree es colocado en el objeto Request por medio del método setAttribute, dicho método es la herramienta ideal para pasar mensajes y conservar temporalmente las referencias a objetos durante el tiempo en que es resuelta la petición en el servidor pasando dichas referencias entre las diversas páginas JSP y los diferentes Servlets y clases auxiliares (como las clases controladoras de casos de uso). Cuando el servidor web finaliza la respuesta (HTTP Response) las referencias creadas por medio del método Request.setAttribute serán eliminadas automáticamente.

Después de que una o más clases controladoras dirigen la correcta ejecución de un caso de uso terminan delegando la responsabilidad de generar el contenido HTML a una o varias páginas JSP (las cuales analizaremos posteriormente en este capítulo) que serán desplegadas al final por el navegador del usuario. Dado que una misma clase controladora podría preceder a distintas páginas JSP. Dichas clases no almacenan una referencia específica con ninguna de estas páginas sino que la relación es dinámica, para ello el servlet despachador[46] crea instancias de las clases controladoras pasando el nombre físico de la página JSP como parámetro del constructor de dichas clases. En el Apéndice H se muestra el código del Servlet despachador.

En esta clase despachadora, llamada MainServlet, existen dos estructuras de datos básicas: La primera es una colección de todos los comandos de la aplicación, cada comando representa a uno de los casos de uso, por la manera en que fue implementado el sistema cada comando está indexado por una clave siendo el comando en sí una instancia de alguna de las clases controladoras de casos de uso. En el método initCommands() se definen todos los comandos de la aplicación y en el código es posible ver que para cada comando tenemos a la clave o índice, y el instanciamiento o construcción de alguna de las clases

[46] Que como mencionamos, es el primero en interpretar el contenido del HTTP Request para después invocar a alguna de las clases controladoras de caso de uso.

controladoras, además es posible ver que el constructor de las clases controladoras normalmente recibe como parámetro el nombre de una o más páginas JSP, esto es que cada comando tiene asociado una página JSP la cual es la que será desplegada al finalizar la ejecución de ese comando (o caso de uso según se quiera ver).

La otra estructura de datos relevante lo constituye un conjunto de clases auxiliares que implementa la navegación entre las distintas páginas de la aplicación. Como la mayoría de los sistemas de software nuestro sistema requiere de una secuencia de páginas o pasos para realizar un caso de uso. Estas clases auxiliares son especializaciones de una clase llamada NavItem y consisten en clases que construyen una serie de elementos de HTML (como botones o cuadros desplegables) para navegar de una página a otra tomando en cuenta los casos de uso definidos.

Gran parte del código de esta clase tiene que ver con inicialización de atributos y estructuras de datos, sin embargo la función primordial de este servlet es la de despachador, y esto se puede comprobar al revisar el código de los métodos doGet y doPost los cuales reciben una petición del tipo HTTP request y la redireccionan a alguno de las clases controladoras de casos de uso, es aquí donde la clave utilizada para la identificación de los comandos entra en juego pues dentro del request vendría indicada dicha clave.

3.5. Desarrollo de la Capa de Visualización.-

Hasta el momento ya hemos visto ejemplos de algunas de las clases de dominio, de persistencia, de control e incluso la clase despachadora. En conjunto estas clases colaboran para atender una petición hecha a través de un HTTP request y esto implicara la ejecución de alguno de nuestros casos de uso. Sin embargo al finalizar estas clases se debe responder a la petición HTTP response generando la página de HTML apropiada para dicha petición. En la plataforma J2EE esta labor puede asignársele tanto a Servlets como a JSP's[47], sin embargo por diversos motivos (sobretodo facilidad e inmediatez de mantenimiento) se prefiere la opción de utilizar JSP's para generar el código HTML.

[47] Los JSP's se transforman en servlets al ejecutarse (normalmente durante la primera ejecución) sin embargo desde el punto de vista de desarrollo se pueden considerar a los servlets y las JSP's como dos elementos distintos pues mientras que para la construcción de un servlet las reglas y sintaxis de Java son primordiales, para construir una JSP la reglas de HTML son preponderantes pese a que normalmente la JSP tendrá código de Java embebido.

Veamos ahora algunos ejemplos de las JSP's utilizadas en esta solución, en primer lugar tendríamos a la página "forumtop.jsp" la cual se encarga de mostrar todos los tópicos del foro de discusión y cuyo código se muestra en el Apéndice I.

Como podemos ver la página JSP es básicamente una página de HTML pero con una serie de tags adicionales en los cuales hay embebido código Java y otros atributos especiales. En particular la función principal de esta pantalla es mostrar un conjunto de tópicos los cuales están referenciados en el objeto Request (se supone que algún objeto de control de caso de uso debe crear esta referencia). El código Java embebido en la página JSP tiene como función crear HTML dinámico dependiendo de los valores de un conjunto de parámetros pasados a la página (normalmente por medio de Java Beans referenciados en el objeto Request). El código Java es ejecutado en el servidor de manera tal que al cliente le llega tanto el HTML estático como el dinámico por lo que el browser interpreta esto como cualquier otra página de HTML.

Un aspecto a resaltar es el hecho de que normalmente cuando existen secciones de código JSP que se repiten en varias páginas un procedimiento normal para no duplicar dichas secciones es el de separar las mismas en páginas JSP externas, las cuales se incluyen en cada una de las páginas que repiten dicha funcionalidad. En la página listada se están incluyendo dos páginas adicionales: top.jsp y bottom.jsp.

La mayoría del contenido dinámico de nuestra página está contenido en los Java Beans que modelan nuestras clases de dominio, entonces es algo común que las páginas JSP de nuestra aplicación tengan constantemente secciones de código donde se consultan los atributos de estos Java Beans, cabe mencionar que existen dos métodos principales para obtener el valor de un atributo de un Java Bean dentro de JSP, por ejemplo para la propiedad "Nombre" del Bean "foro", El primero método para obtener la propiedad es usar un tag especial de JSP, lo que sería algo como esto:

```
<jsp:getProperty name="foro" property="nombre" />
```

El segundo método es utilizando el tag para incrustar código java más la forma abreviada de la salida estándar, lo que sería algo como esto:

```
<%=foro.getNombre()%>
```

Cabe hacer notar que para esta aplicación se decidió usar la segunda forma o sintaxis pues existen situaciones en las que es más legible esta última, por ejemplo cuando dicha propiedad no se escribe directamente a la página sino que se pasa como parámetro a algún método o función digamos para formatear el atributo antes de escribirlo a la salida estándar. Sin embargo la forma recomendada por los documentos oficiales de la especificación JSP es la primera. Se puede concluir que usar una u otra forma es al final de cuentas una cuestión de gustos más que de otra cosa.

En la página listada ("forumtop.jsp"), como en muchas otras de la aplicación, se tiene la necesidad de generar HTML dinámico basado no en uno sino en varios Java Beans del mismo tipo los cuales son referenciados en el objeto Request por medio de un arreglo de Java Beans, entonces se puede decir que estas páginas lo que tienen que hacer es recorrer el arreglo (por medio de un ciclo) para mostrar los valores de todos los Beans dentro del arreglo. Sin embargo tenemos que tener en cuenta el tipo de cliente para el que estamos desarrollando (en este caso un browser de HTML). Resulta poco amigable para el usuario final que se le presenten páginas de HTML que muestran la información de cientos o miles de registros, normalmente lo que se hace para presentar un volumen manejable de registros es paginar el conjunto de resultados para que de este modo el usuario pueda navegar entre varias páginas las cuales muestran un subconjunto del total de registros en donde dicho subconjunto muestra un volumen manejable de información. Para hacer esto en muchas de las páginas JSP de la aplicación nos valdremos de una clase de Java auxiliar llamada "PagedIterator" la cual dividirá un conjunto de resultados en varios bloques de tamaño manejable para el usuario.

Usando la clase "PagedIterator" ahora en vez de tener que recorrer todos los Beans del arreglo referenciado en este caso en el objeto Request, debemos en cambio recorrer todos los Beans correspondientes a una página en particular, para ello esta clase auxiliar implementa la interfaz estándar de Java "Iterator" por lo que para recorrer cada página utilizamos un ciclo bastante simple por medio de un simple "while".

Mientras que para armar el HTML dinámico para construir el HTTP Response que se enviará al cliente para ser desplegado en el browser, se requiere consultar los atributos de una serie de objetos de la capa de dominio, en el sentido opuesto para que el browser construya el HTTP Request la página HTML debe contener uno o más formularios de HTML conteniendo estos elementos de HTML (normalmente del tipo Input o Textarea) los cuales pasan como parámetros tanto en el método Get

como en el Post del HTTP Request. Algunos de estos elementos requieren tener valores default por lo que son llenados desde el armado de la página.

Algo que es importante resaltar es que dentro de la página JSP se incluye código en JavaScript el cual se ejecuta del lado del cliente. Este código implica que hay cierta lógica, dentro de la capa de visualización lo cual como vimos desde el capítulo referente al diseño no es lo ideal. Existe sin embargo una característica de la plataforma que motivó esta decisión de flexibilizar este lineamiento inicial de diseño, dado que la interacción del cliente con el servidor no es continua e inmediata, además se debe tener en cuenta la elevada concurrencia que una aplicación de este tipo debe enfrentar. Lo cierto es que el tipo de interacción que el usuario espera de las aplicaciones actuales está muy por encima de lo que es posible hacer a través de puro HTML y manteniendo la lógica de negocios de lado del servidor exclusivamente.

Este es un ejemplo típico de un escenario donde se tienen dos argumentos bastante válidos y al mismo tiempo opuestos, respecto a alguna característica de un sistema de software. Muchas veces este tipo de disyuntivas que se presentan normalmente en la fase de transición entre el diseño y el desarrollo del sistema obligan a tomar una decisión que muchas veces se sitúa en un termino medio entre ese conjunto de lineamientos opuestos. En general se puede decir que no hay soluciones 100% eficaces por lo que los buenos diseños de software tienen mucho que ver con encontrar esos puntos intermedios entre una serie patrones y lineamientos de diseño opuestos entre sí.

En cuanto a nuestro sistema de foros de discusión en este aspecto en particular el punto intermedio que se decidió para el desarrollo del sistema, establece que la lógica del lado del cliente por medio de JavaScript estará limitado a ciertas características básicas de HTML Dinámico del lado del cliente (DHTML) para hacer más grata la experiencia durante la interacción del usuario con el sistema desarrollado. Adicionalmente y sobre todo para las páginas donde el usuario captura información se incluirá cierta lógica básica para validar la captura de información hecha por el usuario antes de mandar dicha información al servidor a través de un HTTP Request (por ejemplo verificar que todos los campos obligatorios de un formulario sean llenados). Es decir que las reglas de negocio trasladadas a la capa de visualización tendrán que ver más con reglas generales y básicas que con el control de un caso de uso u otro tipo de reglas más especificas del sistema de foros de discusión.

Veamos otro ejemplo de las páginas JSP utilizadas en el sistema de foros de discusión, en este caso la página “profile.jsp” la cual permite que el usuario capture una serie de información personal para ser registrada en el sistema. El código de esta página se muestra en el Apéndice J.

La funcionalidad de esta página a diferencia de la primera listada tiene que ver más con la captura de información que con la visualización de las propiedades de un conjunto de clases del dominio. Para ello podemos ver que se incluye un formulario HTML con una serie de elementos Input y Textarea en los cuales el usuario capturará la información pertinente para que después cada uno de estos elementos se envíen como parámetros al servidor dentro del HTTP Request a través del método Post.

Podemos notar que en la página (al igual que en la primera presentada) tenemos también una serie de elementos ocultos en el formulario HTML los cuales también se envían como parámetros al servidor pero en este caso el usuario no tiene la posibilidad de editar directamente el valor de dichos elementos (sin embargo algunos de ellos son modificados por código en JavaScript).

Muchos de los elementos del formulario son llenados con valores por default, supongamos que el usuario previamente ya había capturado la información de su perfil, entonces cuando regrese a esta página lo más normal sería que los campos ya traigan la información previamente ingresada.

Un elemento importante de esta página es la función ValidaForma de JavaScript la cual se encarga de verificar que los campos obligatorios del formulario sean llenados por el usuario, en caso de faltar uno de estos campos se le indica al usuario cual es elemento que falta, si la verificación no encuentra ningún valor faltante entonces se procede con hacer el submit de la página.

De la simple visualización de los listados de las dos páginas JSP presentadas podemos ver que estas páginas tienen muchos elementos comunes, En ambas páginas se incluyen las páginas adicionales: top.jsp y bottom.jsp. En ambas páginas se tiene un formulario apuntando a un recurso llamado webforums (con la propiedad action del formulario). Dicho nombre de recurso en realidad es un alias para el Servlet despachador que ya describimos con anterioridad.

Otro elemento común que de hecho se encontrará en todas las páginas JSP de la aplicación el un campo oculto llamado “cmd” el cual contendrá la clave del comando que será invocado por la clase despachadora al procesar el HTTP Request generado por el browser. Cuando en una página se pueda invocar a más de un comando, esto se hará posible utilizando código en JavaScript para cambiar el valor de este campo oculto de tal manera que se ponga la clave correspondiente a la interacción que el usuario tenga con la página desplegada en el browser.

Con lo presentado en este capítulo hemos visto ejemplos de muchos de los componentes de cada una de las capas que comprenden el sistema de foros de discusión, todo esto sin hacer una descripción exhaustiva del código presentado (pues no es este el propósito de esta tesis). Hemos visto como los lineamientos expresados en las consideraciones arquitectónicas y los patrones de diseño fueron fundamentales para el desarrollo del sistema en este caso en la plataforma Java de Servlets/JSP, pudimos comprobar que algunos de estos lineamientos se siguieron pero con modificaciones al buscar un termino medio entre el diseño ideal y las características reales de la plataforma de desarrollo.

En el siguiente capítulo se mostrara el desarrollo equivalente de cada una de las capas y componentes pero ahora sobre la plataforma ASP/COM para posteriormente tener los elementos de comparación suficientes para realizar una evaluación aceptable de ambas plataformas.

CAPÍTULO 4. DESARROLLO EN LA PLATAFORMA ASP/COM+

4.1. Introducción al Desarrollo con ASP/COM+.-

En el presente capítulo se abordará ahora el desarrollo del proyecto de foros de discusión a través del web dentro de la plataforma de ASP/COM+. Utilizando los mismos productos obtenidos durante el análisis y diseño que se consolidaron al construir la aplicación en la plataforma de Servlets/JSP.

La plataforma ASP/COM+ posee una serie de características particulares las cuales obligan a adaptar algunos de los lineamientos plasmados durante el diseño, como en cualquier desarrollo se termina llegando a un termino medio entre los lineamientos del diseño y los lineamientos propios de la plataforma.

A diferencia de la plataforma Servlets/Java donde no hay otra alternativa mas que de desarrollar la solución completa utilizando el mismo lenguaje de programación (Java), para la plataforma ASP/COM+ tenemos la oportunidad de escoger entre un conjunto numeroso de lenguajes sobre todo para las capas de dominio y persistencia. Ahora esta flexibilidad tiene sin embargo inconvenientes pues durante el ciclo de vida de la aplicación es normal que se realicen adecuaciones al mismo (mantenimiento) y esta tarea de por sí difícil se dificulta más cuando se tiene que hacer en una aplicación heterogénea en los lenguajes utilizados. Posteriormente en esta tesis se hará un análisis más profundo de las ventajas y desventajas de una u otra alternativa.

Hasta la fase de diseño, se han obtenido un conjunto de clases que ahora durante el desarrollo se materializarán en componentes de software, a diferencia de la plataforma Java donde la orientación a objetos es completa debido al lenguaje, en la plataforma ASP/COM+ el grado de orientación a objetos depende del lenguaje seleccionado para desarrollar cada una de las capas del sistema. Peor aun la capa de visualización (esto es el desarrollo sobre ASP solamente) definitivamente no es orientada a objetos.

En el diseño hemos decidido desarrollar un sistema de múltiples capas y la interacción entre estas capas deberá ajustarse a las reglas que impone el modelo COM[48]. El cual es un protocolo en donde son las interfaces las que tienen relevancia por sobre las clases. Sin embargo COM es perfectamente compatible con un diseño y programación orientados a objetos.

Ahora bien la plataforma COM nos da la libertad de elegir una variedad de lenguajes y entornos de programación algunos más orientados a objetos que otros, pero en la práctica la flexibilidad inicial se ve restringida por una serie de factores como son: Dominio del lenguaje por parte de los recursos que se disponen, tiempo estimado de desarrollo el cual aunque es válido argumentar que depende en gran medida del grado de dominio que se tenga sobre dicho lenguaje, por otra parte no se puede negar que existen lenguajes y entornos de programación que por características propias de los mismos hacen que la tarea de desarrollo sea más rápida que en otros; esto por muchas razones siendo la más evidente que para llevar a cabo una tarea específica en un lenguaje baste con algunas cuantas líneas de código mientras que en otros se requiera un mayor número. Pero existen otras razones inclusive quizás más importantes como podría ser el tipo de herramientas disponibles para depuración, etc. Definitivamente no está dentro del alcance de esta tesis hacer una comparación de lenguajes sino más bien de las características propias de las dos plataformas que hemos elegido.

Teniendo en cuenta lo anterior mencionado se decidió implementar la capa de dominio utilizando al lenguaje Visual Basic, las dos razones principales son la sencillez del lenguaje en si y el alto grado de integración y soporte de este lenguaje con la plataforma ASP. Se tenían otros candidatos (sobresaliendo Delphi y C++) pero Visual Basic es sumamente difícil de vencer en cuanto a tiempo de desarrollo lo cual en la práctica es un factor preponderante en los desarrollos Intranet/Internet los cuales normalmente están constreñidos a calendarios notablemente más estrechos que otro tipo de desarrollos[49] y entonces resulta primordial el elegir las herramientas que disminuyan los tiempos de desarrollo y en este caso las características del lenguaje

[48] Si bien la tecnología COM abarca bastantes herramientas de desarrollo basado en componentes el aspecto fundamental de esta tecnología es definir un esquema de interacción entre componentes a través de interfaces, a lo que se le conoce como IDL (lenguaje de definición de interfaces por sus siglas en ingles). COM es una tecnología similar en sus alcances a los de CORBA.

[49] El alto grado de competencia en la industria para el desarrollo sobre Internet/Intranet se ha traducido en una reducción considerable de los tiempos de desarrollo, dado que constantemente están surgiendo tecnologías que empujan a la obsolescencia a los sistemas sobre Internet/Intranet en un grado mayor a cualquier otro nicho de la tecnología de la información. Los proyectos de desarrollo sobre Internet/Intranet han sufrido este fenómeno de minimización de los calendarios de desarrollo.

y del entorno de programación disponible para Visual Basic lo hacen una alternativa atractiva para muchos de los desarrollos Internet/Intranet sobre la plataforma ASP/COM+.

Sin embargo esta elección tiene sus desventajas, siendo la principal el hecho de que este lenguaje no es orientado a objetos pues no cumple con todos los requisitos para serlo, por lo que es etiquetado como un lenguaje basado en objetos solamente. Ahora bien la principal carencia en cuanto a orientación a objetos se refiere es el nulo soporte a la herencia, y en este caso tenemos suerte pues este aspecto no ha sido explotado para el diseño de nuestras clases. Sin embargo si fuera el caso que la herencia sea un aspecto fundamental de nuestro sistema definitivamente tendríamos que optar por lenguaje que cumpliera cabalmente con esta característica.

Procederemos ahora a abordar el desarrollo de cada una de las capas y al igual que con la plataforma anterior, esta labor estará encausada a cumplir con los lineamientos arquitectónicos que hemos trazado en la fase de diseño, así como con los lineamientos que la plataforma ASP/COM+ impone o promueve.

4.2. Desarrollo de la Capa de Dominio.-

Como ya hemos dicho anteriormente, el conjunto de clases que componen la capa de dominio tiene la responsabilidad de modelar el sistema al que la solución de software pretende representar o automatizar. La capa de dominio está compuesta en su mayoría por clases que representan directamente a los objetos reales que el sistema modela, mas un conjunto de clases auxiliares que facilitan la interacción entre las clases de dominio y las otras capas del sistema de software.

Ahora bien ya hemos elegido a Visual Basic como el lenguaje donde desarrollaremos las clases de dominio, y posteriormente comprobaremos que la facilidad que se tiene de desarrollar componentes COM+ en dicho lenguaje es bastante; a diferencia de otros entornos de programación donde el desarrollo de componentes COM/COM+ implica un trabajo de más bajo nivel para, por ejemplo, especificar el conjunto de Interfaces COM que un subsistema implementa[50]. Cuando se construye un componente COM en Visual Basic

[50] Normalmente un subsistema en COM es un conjunto de clases que implementan cierto número de interfaces. Este conjunto de interfaces se distribuye en un archivo de tipo DLL (Librería de ligado dinámico) o

automáticamente se generan los elementos necesarios para acceder a dicho subsistema por medio de COM, esto es que no es necesario definir separadamente esta serie de interfaces COM[51].

Al igual que con los Java Beans, las propiedades que hemos encontrado en nuestras clases de dominio se deberán de convertir en propiedades de los componentes ActiveX, cada lenguaje capaz de generar componentes COM define una forma y/o sintaxis para enlazar la interfaz COM con la implementación de dicha interfaz por métodos y funciones definidos a través de código, en el caso de Visual Basic la sintaxis es bastante simple por medio de un par de palabras reservadas del lenguaje. Que en realidad no es una sintaxis específica para publicar una interfaz a través de COM sino más bien la sintaxis normal para definir las propiedades de cualquier clase en Visual Basic; esto último es igual para los métodos y funciones.

Veremos ahora como se materializan en Visual Basic las clases de dominio como componentes COM. En este caso la clase de diseño Usuario la cual en su versión COM se muestra en el Apéndice K.

En el código anterior es notoria la forma en que son definidos los atributos de nuestra clase, esto por medio de declaraciones del tipo: *Property Let* para establecer el valor de la propiedad y *Property Get* para consultarlo. Los valores de estas propiedades son almacenados internamente en una variable privada llamada *udtProps* que es de un tipo definido por nosotros llamado *UsuarioProps*, ésta es la definición de ese tipo de datos:

```
Public Type UsuarioProps
  id_usuario As Long
  nombre_corto As String * 20
  nombre_completo As String * 80
  password As String * 20
  email As String * 100
  estatus As String * 1
  fecha_registro As Date
  fecha_ultimo_acceso As Date
  administrador As String * 1
End Type
```

Una peculiaridad que tiene la clase Usuario en Visual Basic y que no la tiene la versión de Java es una función llamada GetState y un método

de tipo EXE (Ejecutable). A este subsistema se le denomina en la terminología de COM como servidor ActiveX.

[51] Al conjunto de interfaces que un servidor ActiveX expone se le conoce como librería de tipos.

llamado SetState, el propósito de estas dos rutinas es la de proveer de una interfaz para serializar y deserializar el estado de las instancias de nuestra clase. Por cuestiones de desempeño, resulta sumamente costoso el mantener referencias de objetos entre llamadas a métodos de otros objetos en COM; esto es que, por ejemplo, cuando posteriormente abordemos la capa de persistencia veremos que en vez de directamente interactuar los objetos de la capa de dominio con los objetos de la capa de persistencia, la capa de persistencia simplemente se ocupará del estado de las instancias de dominio sin tratar con dichas instancias directamente. Aquí podemos ver un ejemplo de un cambio importante entre nuestras clases de diseño y las clases finalmente desarrolladas debido a características propias de la plataforma de desarrollo.

Al igual que para el desarrollo en Java del capítulo anterior se decidió implementar una serie de clases auxiliares cuya función es la de contener una conjunto de valores discretos que constituyen el total de valores válidos para algunas propiedades especificas. En nuestra clase Usuario estamos usando a dos de estas clases: StrBoolean y Estatus las cuales son parte de una librería ActiveX a la que llamamos WebForumsTypes. De esta forma en vez de tener algo como:

estatus = "A"

Tenemos en cambio:

udtProps.estatus = stesta.ACTIVO

Cabe hacer notar que a diferencia de Java en Visual Basic no hay métodos de clase estáticos, por lo tanto hay que crear una instancia a estas clases auxiliares antes de poder obtener estos valores. La justificación de la creación de estas clases es la misma que para el desarrollo sobre Java, esto es que con ellas determinamos un punto único de acceso a este tipo de constantes las cuales normalmente se esconden en el código de los sistemas de software y que a la larga dificultan el posterior mantenimiento del mismo. Como ejemplo el Apéndice L nos muestra la implementación de la clase auxiliar Estatus:

Ahora veremos otro aspecto importante de la implementación de las clases de dominio en Visual Basic, como ya indicamos anteriormente no habrá interacción directa entre la capa de persistencia y la de dominio, sino que la relación entre ambas capas será por medio de la serialización y deserialización de todos las propiedades de cada clase de dominio (su estado). Sin embargo este escenario introduce ciertas dificultades al momento de activar o pasivar un conjunto de clases de

dominio (por ejemplo cuando se activan todos los tópicos de un foro en particular).

A diferencia del desarrollo en Java en el cual la activación múltiple se resolvía simplemente utilizando arreglos cuyos elementos son referencias directas a las múltiples instancias de la clase de dominio en particular. Para la plataforma ASP/COM+ esto ya no será posible pues en vez de que la capa de persistencia manipule directamente referencias a clases de la capa de dominio, el manejo se reduce al estado serializado de las clases de dominio; y por lo tanto la activación múltiple lidiará ahora con el estado serializado de múltiples instancias. Ahora bien aunque existen distintas alternativas para el manejo de las múltiples instancias, la opción más apegada a los lineamientos de desarrollo de la plataforma ASP/COM+ sobre Visual Basic sería sin duda el uso de colecciones de objetos para la interacción con múltiples instancias relacionadas entre sí.

Para permitir el manejo de colecciones de instancias de una clase en particular se decidió construir un conjunto de clases adicionales con este propósito. Veamos por ejemplo a la clase para el manejo de una colección de instancias de la clase Usuario la cual se llama a su vez VectorUsuario, el código de esta clase se muestra en el Apéndice M.

Podemos observar que en la clase VectorUsuario volvemos a tener el método SetState con el cual se reconstruye el estado de este objeto a través del valor serializado del mismo. El estado serializado de la clase VectorUsuario es en realidad la unión de los estados serializados de todas las instancias de la clase Usuario correspondientes. De esta manera los componentes de la capa de persistencia materializarán desde la base de datos un conjunto de objetos Usuario los cuales serán manejados en otras capas a través de la clase VectorUsuario, similarmente se tendrán clases para el manejo de colecciones de otras clases del dominio.

Un método importante de la clase VectorUsuario es PagedIterator, este método es de hecho una función la cual regresa una referencia a un objeto de tipo PagedIterator, Posteriormente analizaremos la funcionalidad de PagedIterator con mayor detalle, por este momento basta con decir que este método nos permite acceder a subconjuntos de la colección completa.

Desde el capítulo anterior abordamos las disyuntivas a que se enfrentan comúnmente los desarrollos de sistemas de software; la cual consiste en por un lado cumplir con los lineamientos de diseño y por el

otro lado ajustarse a los lineamientos prácticos debidos a una serie de factores entre los cuales destacan las características propias de la plataforma de implantación. Por ejemplo tenemos el lineamiento de diseño que nos dice que nosotros debemos implementar toda la lógica relacionada a las reglas de negocio dentro de la capa de dominio, sin embargo esto no siempre resulta en una mayor eficiencia del producto desarrollado, esto es que existen otras capas en las que al implementar algunas de las reglas de negocio es más eficiente que dejárselo a las clases de dominio solamente.

Al igual que en el desarrollo en Java se decidió en este caso el relegar algunas responsabilidades a otras capas. Notablemente a la capa de persistencia se le asignaron responsabilidades que es más fácil y eficiente implementar en esta capa, por ejemplo, existe una serie de validaciones como son evitar duplicidad al momento de dar de alta nuevos usuarios, etc. También ya se abordo desde el capítulo anterior la necesidad de relegar validaciones simples a la capa de visualización (por medio de JavaScript) en esta versión del sistema se mantiene esta necesidad, por lo que posteriormente veremos algunos ejemplos en los que reglas de negocio especificas son cubiertas por otras capas además de la de dominio.

Con el mismo argumento esgrimido para la versión Java, esto es que tenemos el propósito de que la comparación que haremos entre las plataformas JSP/Servlets contra ASP/COM+ se enfoque más en las características intrínsecas de estas plataformas en vez de perderse en las características que se producen por tomar decisiones de diseño no relacionadas con la plataforma de implantación. Para ello se dispuso simplificar enormemente la complejidad de la capa de dominio minimizando las reglas de negocio de las que esta capa es responsable de hacer cumplir.

4.3. Desarrollo de la Capa de Persistencia. -

Ahora veremos la implementación de la capa de persistencia en la plataforma ASP/COM+, al igual que para la capa de dominio se decidió implementar esta capa utilizando el lenguaje Visual Basic, pero cabe hacer notar que hubiera resultado perfectamente válido seleccionar un lenguaje distinto del de la capa de dominio, pues el requisito es que dicho lenguaje soporte la interacción de componentes a través de COM.

En primer lugar veremos la clase que se encargará de la persistencia de la clase de dominio Usuario, y al igual que con Java llamaremos a esta clase UsuarioPersist, su código lo tenemos en el Apéndice N.

La clase UsuarioPersist es la responsable de la activación y pasivación de la clase de dominio Usuario, y para ello la clase UsuarioPersist cuenta con una serie de funciones como:

- *getUsuariosToSendEmail*
- *getUsuario*
- *getUsuarioLogin*

Con los cuales es posible activar una instancia de la clase Usuario de acuerdo algún parámetro dado. La función *getUsuariosToSendEmail* permite activar todo un conjunto de instancias de Usuario. Cabe hacer notar que en realidad estas funciones obtienen el estado de instancias de la clase Usuario desde la base de datos y dicho estado es devuelto en forma serializada (esto es que como habíamos dicho no se regresan referencias directas a las clases de dominio).

De manera similar las funciones y métodos:

- *update*
- *put*
- *removeUsuario*

Nos permiten "pasivar" una instancia de la clase Usuario, esto es que la base de datos es actualizada con una instancia de Usuario. La función *update* actualiza la base de datos con los últimos valores de los atributos para un objeto de tipo Usuario ya existente, la función *put* en cambio actualiza la base de datos con una instancia nueva de Usuario y finalmente el método *removeUsuario* elimina de la base de datos los registros correspondientes a una instancia de la clase Usuario dada[52].

Podemos ver que la clase tiene métodos privados que le permiten serializar y deserializar el estado de una instancia de la clase Usuario, tal como se dijo las clases de la capa de persistencia no interactuarán con las clases de dominio directamente sino a través del estado de las mismas. Esta decisión está motivada por cuestiones de desempeño en la plataforma ASP/COM+ sin embargo en el código de la clase UsuarioPersist es posible ver la principal desventaja de esto, la cual es

[52] Al igual que en la versión de Java de esta clase, solo es necesario pasar el valor de *id_usuario* a este método para eliminar un Usuario lo cual evita el costo de tener que serializar el estado completo de la clase de dominio para poderla eliminar de la base de datos.

que se está violando en cierta medida el encapsulamiento de las propiedades de la clase Usuario, las cuales son accedidas de una manera poco acorde a las reglas básicas de la programación orientada a objetos.

Un aspecto importante de esta clase es el tipo de acceso a la base de datos, en este caso es a través del API ADO el cual se ha convertido en el API por defecto de acceso a esquemas de persistencia (especialmente bases de datos) para la plataforma ASP/COM+. A diferencia de la versión Java las sentencias de SQL se encuentran directamente embebidas en el código de Visual Basic (sin embargo la decisión de hacerlo así no tiene que ver con alguna restricción de la plataforma). Observando la línea de código:

*strSQL = "SELECT * FROM USUARIO WHERE ID_USUARIO = " & id_usuario*

Podemos ver que al igual que en la versión Java se está utilizando el patrón de diseño: "Representación de Objetos como Tablas" bajo la cual se asigna una tabla en la base de datos para cada clase de objeto y en donde los renglones de la tabla corresponden uno a uno instancias distintas de cada clase y las columnas corresponden a cada una de las propiedades de la clase. Específicamente con el ejemplo anterior se estaría activando una instancia de Usuario por medio del atributo *id_usuario*.

En el código de la clase UsuarioPersist se muestra una forma de manejo de errores típica del lenguaje Visual Basic, otra característica todavía más importante es una serie de partes del código relacionados con los servicios disponibles a través de COM+[53], en el código de la clase se puede ver segmentos relacionados con el control del robusto sistema transaccional que COM+ ofrece, adicionalmente están disponibles otros servicios de COM+ como, por ejemplo, pool de conexiones, un simplificado esquema de seguridad y distribución, etc. Por otro lado, existen otras características avanzadas de COM+ que no es posible implementar con Visual Basic, en capítulos posteriores se abordará con mayor profundidad estos aspectos.

Al igual que para la plataforma Java se decidió implementar una clase llamada UsuarioIDGenerator con la cual generarán identificadores únicos

[53] Los servicios avanzados que COM+ ofrece, provienen de la evolución de productos como Microsoft Transaction Server (MTS) los cuales han pasado a ser (desde Windows 2000) servicios otorgados por el sistema operativo que son disponibles a cualquier componente COM+ en vez de requerir la utilización de un producto adicional.

para el atributo id_usuario en las instancias de Usuario nuevas, su código se muestra en el Apéndice O.

Para las clases de dominio similares a Usuario, donde es posible que se generen nuevas instancias de las mismas en determinados casos de uso, se decidió implementar estas clases para obtener identificadores únicos en vez de utilizar alguna herramienta de la base de datos y de esta manera obtenemos una solución desacoplada a una base datos en particular.

Con las clases de la capa persistencia aquí presentadas hemos visto las características y funcionalidad principal de todas las demás clases de esta capa, ahora veremos la capa de control la cual permite integrar tanto a la capa de dominio como a la de persistencia para efectivamente cumplir con los casos de uso planteados.

4.4. Clases de control de los casos de uso.-

En la fase de diseño se describió la utilidad de contar con un conjunto de clases especializadas en controlar y dirigir a las clases de dominio y persistencia con el fin de cumplir con la ejecución adecuada de los casos de uso, ahora bien, en el caso de la plataforma Java no hubo ningún inconveniente en desarrollar la capa de control efectivamente con un conjunto de clases a las que se llamó clases controladoras de casos de uso. Sin embargo la estrecha dependencia de esta capa con elementos de HTTP como por ejemplo el procesamiento de los parámetros pasados en el HTTP request y la interacción con la capa de visualización para producir el HTTP response introducen ciertas consideraciones especificas para la plataforma ASP.

ASP es una tecnología que facilita la producción de contenido HTML dinámico del lado del servidor, sin embargo es una tecnología no relacionada con la programación orientada a objetos[54], ahora bien, esta tecnología está fuertemente apoyada en modelo de distribución en componentes que significa COM/COM+ por medio del cual es posible integrar componentes hechos en herramientas que si cumplen con las reglas de la orientación a objetos. Si bien es posible desarrollas una solución para esta plataforma en donde casi toda la funcionalidad este expresada en componentes, en la práctica esto no resulta del todo conveniente pues se pierde una de las principales ventajas de ASP que

[54] A diferencia de los JSP y Servlets las cuales son tecnologías exclusivas de Java, el cual a su vez es un lenguaje que cumple plenamente con las reglas de la programación orientada a objetos.

es la flexibilidad de integrar en un solo punto los elementos puros de HTML/HTTP con las reglas de negocio especificas de la solución.

Para ahondar en este aspecto es importante reflexionar que el contenido visual y de información de una aplicación de Internet/Intranet está constantemente cambiando, por lo que resulta inviable programar estos dos aspectos en componentes monolíticos. De hacerse así se requeriría de un esfuerzo considerable para cada ocasión que se deban modificar. En la práctica cotidiana los desarrollos de sistemas que utilizan la tecnología ASP deben de enfrentarse a tomar la decisión sobre que tanto de la solución se realizará en componentes ActiveX separados y que tanto se realizará por medio de los guiones del lado del servidor que conforman ASP propiamente. Estos guiones se realizan comúnmente en dos diferentes lenguajes interpretados VBScript y JavaScript. El problema aquí es que estos lenguajes cumplen con muy pocos de los fundamentos de la programación orientada a objetos.

La tecnología ASP provee de herramientas que simplifican notablemente el manejo de los diversos elementos correspondientes a una solución para Internet/Intranet. Parte de esta simplicidad se debe al uso de estos flexibles y laxos lenguajes interpretados. Por lo tanto si deseamos aprovechar esta flexibilidad debemos pagar el precio de utilizar en mayor o menor medida un entorno poco fértil para implementar un diseño orientado a objetos. Dado que nuestro interés está en comparar esas dos plataformas por lo tanto creemos que la decisión más adecuada a este interés es el aprovechar al máximo las ventajas de cada plataforma. Lo cual no haríamos si decidiéramos trasladar por completo el desarrollo del sistema en componentes ActiveX monolíticos para poder cumplir cabalmente con un diseño orientado a objetos.

Apoyados en estos argumentos presentaremos ahora los entes encargados de controlar la correcta ejecución de casos de uso, y a diferencia de la solución en Java, no serán clases las que se encarguen de dicha tarea, serán simplemente funciones hechas en alguno de los lenguajes interpretados que ASP posee. Específicamente el lenguaje a utilizar será VBScript y para cada caso de uso tendremos una función encargada de controlar a las clases pertinentes tanto de la capa de dominio como la de persistencia. En el Apéndice P se muestra el código de la función que controlará el caso de uso “Un usuario no-registrado se registra (identifica)”.

Podemos ver que la función GetUserFromLogin invoca a las clases de dominio y persistencia necesarias para validar que el nombre de usuario

y la contraseña presentes como parámetros en el HTTP request. Específicamente los identificadores *usuario* y *usuariodb* hacen referencia a instancias de las clases Usuario y UsuarioPersist respectivamente. Estas y otras clases de dominio son definidas en nuestra aplicación ASP de la siguiente manera.

```
<OBJECT RUNAT="Server" ID="usuario" PROGID="WebForumsDomain.Usuario"></OBJECT>
<OBJECT RUNAT="Server" ID="usuariodb" PROGID="WebForumsPersist.UsuarioPersist"></OBJECT>
```

Existe otra alternativa de hacer referencia a objetos COM desde páginas ASP; ésta consiste en utilizar la función CreateObject del objeto Server. Sin embargo la alternativa por nosotros usada, es preferible, pues en caso de no utilizarse la referencia a los objetos COM dentro de código interpretado en la página ASP, no se invocará la creación del objeto COM lo que es una importante optimización respecto a la utilización de la función CreateObject.

En la función GetUserFromLogin se puede ver como a través de la clase de la capa de persistencia UsuarioPersist se activa desde la base de datos una instancia de la clase de la capa de dominio Usuario. Sin embargo a diferencia de la versión en Java la interacción entre estas dos capas es a nivel del estado serializado. Una vez activada la instancia de Usuario otros miembros de la aplicación ASP podrán hacer uso de esta referencia, esto es que el objeto Usuario activado queda disponible para que otros componentes de ASP puedan hacer uso del mismo.

En el Apéndice Q se muestra otro ejemplo de función controladora de caso de uso; La función GetAllMessagesOfTopic aprovecha a la clase de persistencia VMensajePersist y a la clase de dominio VectorVMensaje para permitir mostrar el listado de mensajes que pertenecen a un tópico. Al igual que el caso anterior los componentes COM son referenciados con el tag “OBJECT” y con el atributo de “RunAt=Server”, como se muestra a continuación:

```
<OBJECT RUNAT="Server" ID="vectorvmensaje"
PROGID="WebForumsDomain.VectorVMensaje"></OBJECT>
<OBJECT RUNAT="Server" ID="vmensajedb" PROGID="WebForumsPersist.VMensajePersist"></OBJECT>
<OBJECT RUNAT="Server" ID="ModoVisualizacion"
PROGID="WebForumsTypes.ModoVisualizacion"></OBJECT>
```

Con la función GetAllMessagesOfTopic se encarga de controlar uno de los casos de uso más importantes de la aplicación: “Un usuario lee los mensajes de un tópico”. Esta función obtiene del objeto Request el valor de id_topico, todos los valores de los parámetros del HTTP request están

contenidos en colecciones pertenecientes al objeto Request, y es posible obtener su valor por medio de una cadena con el nombre del parámetro para este parámetro en particular usamos la sintaxis: Request("id_topico"). Una vez obtenido el valor de este parámetro se invoca a VMensajePersist para obtener el estado de todos los mensajes de dicho tópico utilizando la clase de dominio VectorVMensaje. Como habíamos mencionado previamente la clase VectorVMensaje contiene una función que regresa una instancia de la clase MensajeTree, En el Apéndice R se muestra el código de esta clase.

A través de la clase MensajeTree la página ASP correspondiente podrá mostrar la jerarquía de los mensajes en cuanto a que mensaje es respuesta de otro mensaje.

En los dos ejemplos de funciones controladoras que hemos visto se materializan una serie de objetos los cuales quedan disponibles para que posteriormente otro componente de la aplicación los tome para generar el contenido de la página HTML. A diferencia de la versión Java donde el método tradicional para pasar referencias entre páginas JSP y servlets es a través de los métodos setAttribute y getAttribute, en ASP no existe una herramienta similar, posteriormente ahondaremos en como se pasa el control desde una página ASP a otra, pero por el momento cabe mencionar que las referencias por medio de los tags “OBJECT” son globales a todas las páginas que intervienen en generar la respuesta a una petición HTTP. Otra forma de conservar referencias entre varias páginas es utilizando variables globales del código interpretado VBScript ó JavaScript.

Tenemos la necesidad de mantener estas referencias globales entre páginas ASP, por ejemplo entre la página con el código que controla el caso de uso, y la página donde se generará la respuesta (HTTP response) a la petición del usuario (HTTP request). Para ello debemos cumplir con algunas restricciones en cuanto a la manera en que se invoca a una página ASP desde otra. Estas restricciones las veremos con mayor detalle un poco más adelante.

A continuación mostraremos el código despachador que bajo los mismos argumentos de las funciones controladoras de casos de uso, no se desarrollará como una clase sino como un componente de ASP. Este componente será encargado de interpretar la petición del usuario y delegar la responsabilidad a la función controladora de caso de uso adecuada; para finalmente invocar a la página ASP encargada de producir el contenido de HTML que se desplegará finalmente como respuesta a la petición del usuario y de la ejecución del caso de uso. En

esta plataforma el componente despachador se desarrollara como un conjunto de páginas ASP. En el Apéndice S se muestra el código de la página “main.asp”.

La página “main.asp” es la primera en tomar el control al generarse la petición del usuario. Un interesante elemento de esta página lo constituyen las sentencias del tipo:

```
<!-- #INCLUDE FILE="common.asp" -->
```

Con la sentencia anterior se incluye el código de una segunda página ASP (“common.asp” en este ejemplo) en la página “main.asp”. A esto se le conoce como inclusión del lado del servidor y es una de las maneras de conectar y coordinar el procesamiento de varias páginas ASP. Las sentencias de inclusión en la parte superior hacen referencia a las páginas con referencias globales y funciones auxiliares que se utilizarán por los otros elementos de la aplicación ASP (especialmente las funciones controladoras de casos de uso y las páginas ASP que generan el contenido HTML). Ahora bien al final de “main.asp” hay una última sentencia de inclusión que invoca a la página “dispatcher.asp” cuyo código se muestra en el Apéndice T.

En esta página está verdaderamente la lógica de despacho que permite que se invoque a la función controladora del caso de uso que corresponda a la petición del usuario. Pero antes de revisar dicha funcionalidad, veremos el propósito de la serie de inclusiones del lado del servidor que se encuentran al final de la página.

La serie de inclusiones del lado del servidor combinadas con código VBScript tienen como propósito delegar la generación de la respuesta a la petición del usuario a alguna página ASP que generará el código. Ésta es una manera de integrar varias páginas ASP para que entre ellas resuelvan una petición dada, permitiendo (como ya habíamos señalado) que las páginas compartan referencias globales entre sí. Sin embargo las inclusiones del lado del servidor presentan inconvenientes importantes debido principalmente a que esta forma de integrar páginas es equivalente a unir todas las páginas incluidas en una sola; entonces cuando se hacen declaraciones de variables en una página estas se hacen globales al resto de las páginas incluidas lo que nos genera cierta dificultad en la integración, la manera en que fue atenuado este inconveniente fue la de reservar una página ASP llamada "dim.asp" en donde se hacen todas las definiciones de variables y otras páginas donde se hacen las eferencias de objetos COM externos como los de las capas de dominio y persistencia y otros componentes auxiliares.

Existe una alternativa para esta forma de integrar páginas ASP utilizando el método Execute del objeto Server. Sin embargo nosotros de hecho requerimos que las referencias hechas en una página se conserven en otras. Para ello podríamos utilizar algún contenedor para guardar las referencias por ejemplo utilizando un objeto de tipo colección o diccionario a nivel sesión. Sin embargo esto resulta en importantes costos en el desempeño de la aplicación. Nuestra alternativa de usar inclusiones del lado del servidor dificulta la integración de las páginas pero presenta a nuestro juicio la mejor alternativa en cuestiones de desempeño.

Regresando a la funcionalidad de despachar a la función controladora de caso de uso que se encargará de responder la petición del usuario; tenemos la siguiente línea de código:

Execute "nextASP = " & commands.Item(cmd)

En esta línea tenemos un componente llamado “commands” que regresa el nombre de la función controladora del caso de uso la cual es ejecutada y al hacerlo regresa el nombre de la página ASP que generará el contenido HTML final. De hecho en nuestra aplicación ASP tenemos varias colecciones y diccionarios muchos de los cuales son inicializados al principio de la ejecución del sistema. ASP provee de una forma estándar para realizar actividades ligadas a los eventos de inicialización y finalización del sistema así como la inicialización y finalización de una sesión. Para esto se emplaza dicha funcionalidad en un archivo especial llamado “global.asa” el cual es mostrado en el Apéndice U.

Podemos ver que tenemos el diccionario “commands” que contiene todos los comandos de la aplicación; cada comando representa a uno de los casos de uso, cada comando está indexado por una clave dentro del diccionario mientras que el comando es una cadena para invocar a alguna de las funciones controladoras de casos de uso. En el método initCommands se definen todos los comandos de la aplicación y en el código es posible ver que para cada comando tenemos a la clave del diccionario y la cadena para invocar a alguna de las funciones controladoras de caso de uso, además es posible ver que la llamada a las funciones controladoras normalmente recibe como parámetro el nombre de una o más páginas ASP, esto es que cada comando tiene asociado una página ASP la cual es la que será desplegada al finalizar la ejecución de la función correspondiente. El diccionario “commands” es el mismo que se utiliza en la página ASP “dispatcher.asp” para elegir la función controladora de casos de uso a la que se despachará la respuesta de la petición del usuario.

Tenemos otra estructura de datos relevante la cual es un componente que implementa la navegación entre las distintas páginas de la aplicación; facilitando de esta manera la navegación entre una secuencia de páginas relacionadas entre sí. Este componente es una colección compuesta por objetos que a su vez construyen una serie de elementos de HTML (como botones o cuadros desplegables) para navegar de una página a otra tomando en cuenta los casos de uso definidos.

Con lo hasta ahora mostrado se han visto los componentes primordiales que tienen que ver con la ejecución de un caso de uso. Las clases de la capa de dominio y persistencia más las funciones en VBScript controladoras de casos de uso más las páginas ASP que fungen como despachadoras permiten que la petición hecha por el usuario a través de su browser sea resuelta de la manera que corresponda al caso de uso en particular que se este llevando a cabo. Sin embargo una vez terminada la actuación de todos estos componentes el sistema de foros de discusión debe generar una página de HTML con el contenido apropiado a la petición hecha por el usuario. Esto último es responsabilidad de una capa más del sistema la cual es la capa de visualización que a continuación detallaremos.

4.5. Desarrollo de la Capa de Visualización.-

La capa de visualización estará compuesta por una serie de páginas ASP las cuales contienen tanto HTML estático como código interpretado que produciría dinámicamente otras partes de la página final que es enviada al cliente para ser desplegada en un browser.

Estas páginas ASP de la capa de visualización aprovecharán las referencias globales construidas por los demás componentes de la aplicación. Para clarificar la estructura básica de las páginas ASP de la capa de visualización, veremos ahora algunos ejemplos de las mismas. En primer lugar tenemos a la página “forumtop.asp” la cual se encarga de mostrar todos los tópicos del foro de discusión, el código de esta página se encuentra en el Apéndice V.

Como habíamos dicho la página “forumtop.asp” es básicamente una página de HTML pero con una serie de tags adicionales en los cuales hay embebido código VBScript (interpretado del lado del servidor) y otros atributos especiales. En particular la función principal de esta pantalla es mostrar un conjunto de tópicos los cuales están referenciados globalmente, por ejemplo tenemos una referencia identificada por

vectorvtopico el cual apunta a una instancia de la clase VectorVTopico; esta instancia se supone fue previamente materializada por alguna de las funciones de control de caso de uso.

Hay que recordar que esta página sería invocada por la página “dispatcher.asp” la cual después de ejecutar la función controladora de caso de uso adecuada, le pasa el control a esta página por medio de una sentencia de inclusión del lado del servidor. Por cierto que esta página también tiene dos sentencias de inclusión del lado del servidor las cuales invocan a las páginas “top.asp” y “bottom.asp”.

Un alto porcentaje del contenido dinámico de la página está contenido en los componentes COM+ de la capa de dominio referenciados por la aplicación. Es por eso que en importantes secciones del código VBScript interpretado tenemos la consulta a los atributos de estos objetos de la capa de dominio. Por ejemplo tenemos la propiedad “nombre” del componente identificado por “foro”. Por ejemplo si quisiéramos escribir en la página el valor de esta propiedad, ésta sería la manera de hacerlo:

```
<%=foro.nombre%>
```

En la página “forumtop.asp” como en muchas otras páginas ASP de la capa de visualización se tiene la necesidad de generar HTML dinámico utilizando aquellos componentes de la capa de dominio especializados en mantener colecciones de otros objetos de dominio. Por ejemplo la clase VectorVTopico mantiene una colección de objetos Vtopico; entonces lo que tienen que hacer estas páginas es recorrer la colección (por medio de un ciclo) para mostrar los valores de todos los componentes individuales.

Aquí entra en juego el mismo argumento que se manejo en la versión Java el cual nos dice que tenemos que tener en cuenta el tipo de cliente para el que estamos desarrollando (en este caso un browser de HTML), y que resulta poco amigable para el usuario final que se le presenten páginas de HTML que muestran la información de cientos o miles de registros; entonces lo que se haremos para presentar un volumen manejable de registros es paginar el conjunto de resultados y de este modo el usuario podrá navegar entre varias páginas las cuales mostrarán un subconjunto del total de registros en donde dicho subconjunto muestra un volumen manejable de información. Para hacer esto en estas páginas ASP nos valdremos del componente auxiliar llamado “PagedIterator” el cual dividirá un conjunto de resultados en varios bloques de tamaño manejable para el usuario.

Cada clase de dominio que maneje colecciones de objetos individuales contiene una función llamada también “PagedIterator” la cual regresa una referencia a un objeto de esa misma clase, ahora bien, cuando en las páginas ASP de la capa de visualización se desea desplegar los objetos individuales que corresponden a una sola página en particular entonces se le pasa como parámetro a la función el número de página en la que estamos interesados así como el número de objetos por página. El componente “PagedIterator” implementa la interfaz estándar de COM “_NewEnum” la cual proporciona una forma estándar de recorrer los elementos de una colección. Gracias a esto en nuestra página ASP podemos entonces utilizar un ciclo estándar de VBScript usando la sintaxis “For Each” y acceder entonces a cada uno de los objetos individuales que corresponden a la página actual.

Durante la construcción de la respuesta al cliente (HTTP response) normalmente en nuestra aplicación se deberá consultar los atributos de una serie de objetos de la capa de dominio. Por otra parte para que el browser construya la petición al servidor (HTTP request) la página HTML debe contener uno o más formularios de HTML conteniendo estos elementos de HTML (normalmente del tipo Input o Textarea) los cuales pasan como parámetros tanto en el método Get como en el Post del HTTP request. Algunos de estos elementos requerirán tener valores default por lo que estos valores deberán ser llenados desde el armado de la página.

Al igual que en las páginas JSP de la versión Java, las páginas ASP de esta versión incluyen código en JavaScript el cual se ejecuta del lado del cliente. No es nuestra intención argumentar de nuevo las razones por las que se traslada lógica del negocio a la capa de visualización, en vez de ello diremos que los mismos argumentos que se manejaron para hacer esto en la versión de Java se repiten de nuevo para esta versión.

Veremos ahora el listado de otra página ASP de la capa de visualización, la página “profile.asp” permite que el usuario capture la información personal de un usuario del sistema para ser registrada en la base de datos. Este listado lo presentamos en el Apéndice W.

La página “profile.asp” contiene una funcionalidad diferente de la página previa, pues tiene que ver más con la captura de información que con la visualización de las propiedades de un conjunto de clases del dominio. Para ello se incluye un formulario HTML con un grupo de elementos Input y Textarea con los cuales el usuario capturará la información pertinente. Al final cada uno de estos elementos se enviarán

como parámetros al servidor dentro del HTTP request por medio del método Post.

En las páginas ASP de la capa de visualización, será normal que se tenga una serie de elementos ocultos en el formulario HTML los cuales también se envían como parámetros al servidor con la diferencia que el usuario no tiene la posibilidad de editar directamente el valor de dichos elementos (aunque la interacción del usuario podría hacer que cambiarán su valor por medio de JavaScript).

También es común que muchos de los elementos de los formularios HTML sean llenados con valores por default, para la última página pongamos por ejemplo que el usuario previamente ya había capturado la información de su perfil, entonces cuando el mismo usuario regrese a esta página lo más normal sería que los campos ya traigan los valores previamente ingresados.

Tenemos un elemento importante en esta página, el cual es la función ValidaForma de JavaScript que se encarga de verificar que los campos obligatorios del formulario sean llenados por el usuario, si faltara por capturar uno de estos campos se le indicaría al usuario cual es elemento faltante; en caso contrario se procedería con el submit de la página.

La mayoría de las páginas ASP de la capa de visualización de nuestra aplicación tendrán muchos elementos comunes. Por ejemplo en todas tendremos las sentencias de inclusión del lado del servidor para las páginas: “top.asp” y “bottom.asp”. Otro elemento común sería el contar con un formulario apuntando a la página “main.asp” (en la propiedad action), por medio de esto último garantizamos que el submit hecho por el browser se direccione a la página que ya describimos anteriormente y en donde se invoca a la página despachadora de casos de uso.

Al igual que en la versión Java tenemos otro elemento común en las páginas ASP de la capa de visualización, este elemento es la inclusión de un campo oculto llamado “cmd” el cual contendrá la clave del comando que será invocado por la página despachadora al procesar la petición del usuario (HTTP request). En las páginas donde se pueda invocar a más de un comando esto se hará utilizando JavaScript para cambiar el valor de este campo oculto, de tal manera que la interacción del usuario determinaría el caso de uso a ejecutar una vez realizado el submit por el browser.

En este capítulo hemos visto ejemplos de muchos de los componentes de cada una de las capas que comprenden el sistema de foros de discusión. Hemos comprobado como los lineamientos expresados en las consideraciones arquitectónicas y los patrones de diseño fueron fundamentales para el desarrollo del sistema en la plataforma de ASP/COM+. En general estos lineamientos se siguieron pero con modificaciones debido a las características particulares de la plataforma de desarrollo.

Gracias a lo realizado en este capítulo y el anterior tenemos ahora los elementos suficientes (a nuestro criterio) para realizar un análisis comparativo de ambas plataformas. Por lo tanto en el siguiente capítulo se mostrara los resultados de dicho análisis.

CAPÍTULO 5. COMPARACIÓN DE LAS TECNOLOGÍAS

5.1. Introducción a la comparación de las tecnologías.-

En los capítulos anteriores abordamos el análisis, diseño y finalmente el desarrollo de nuestro sistema de foros de discusión, el cual sirvió de caso práctico para mostrar las cualidades que poseen las tecnologías ASP/COM+ y JSP/Servlets. Sin lugar a dudas, el haber realizado el desarrollo de este caso práctico nos ha dado bastantes herramientas para ahora abordar la comparación de las mismas.

La comparación de las tecnologías ASP/COM+ y Servlets/JSP no es una tarea sencilla. Existen muchos aspectos concernientes a cada una de las dos que dificultan la comparación de las mismas. El principal inconveniente para comparar las dos tecnologías es que mientras que con ASP/COM+ hay una sola implementación la cual ofrece las herramientas necesarias y suficientes para desarrollar aplicaciones Internet/Intranet; Por el otro lado, la tecnología Servlets/JSP es una especificación. Esto quiere decir que hay un conjunto de implementaciones (Servidores de aplicación J2EE) que se adhieren a la especificación y por medio de los cuales también podemos desarrollar aplicaciones Internet/Intranet.

El significado de esto es que el comparar de igual a igual estas dos tecnologías es por lo tanto inapropiado. Para comparar estas tecnologías no basta con una simple diferenciación entre las características que ambas plataformas tengan. Nuestra estrategia en cambio será la de elegir una serie de atributos generales que en nuestra opinión son significativos y a partir de los mismos revisar cual es el estado que guarda cada una de las tecnologías respecto a dicho atributo.

Si bien el desarrollo del sistema de foros de discusión nos provee de elementos para abordar la comparación de ambas tecnologías, resultaría iluso pretender que es suficiente con dicho desarrollo para pretender calificar a cualquiera de las dos tecnologías. Ambas tecnologías y sobre todo sus plataformas subyacentes (Windows DNA y J2EE) son demasiado extensas como para que un solo caso práctico baste para este fin. Por lo tanto además de las consideraciones que provengan directamente del desarrollo del caso práctico, incluiremos también elementos adicionales que provienen tanto de nuestra experiencia

personal como de la investigación realizadas por nosotros en función de este trabajo de tesis.

5.2 Comparación de ambas tecnologías a partir del desarrollo del sistema de foros de discusión.-

El propósito fundamental de hacer el desarrollo del sistema de foros de discusión sobre ambas tecnologías y sus plataformas subyacentes en este trabajo de tesis es el de tener un caso práctico que nos permita explorar las características principales de ambas. Con dicho caso práctico podemos mostrar como se resuelven algunas de las disyuntivas inherentes a desarrollar una aplicación Internet/Intranet usando ambas tecnologías. Sin embargo no debemos ignorar el hecho de que es prácticamente imposible que una única aplicación abarque todos los aspectos de una plataforma tan extensa como J2EE o Windows DNA.

El desarrollo de las dos versiones de la capa de presentación nos dio la oportunidad de revisar y comparar las dos tecnologías de acceso a bases de datos que ambas plataformas incluyen, JDBC y ADO. Por un lado JDBC es un API para conexión a bases de datos relacionales al que un buen número de proveedores se han adherido para crear manejadores de JDBC que se conectan a una base de datos relacional específica. Por el otro ADO es una librería de objetos que mapean jerárquicamente las principales entidades de una fuente de datos. ADO está situado encima de OLE DB el cual es una librería para el acceso universal a fuentes de datos de tipo relacional, multidimencional o del tipo de directorios.

Un aspecto que resaltó durante el desarrollo de nuestro sistema de foros de discusión, es que en la versión ASP/COM+ pudimos aprovechar del servicio de pool de conexiones el cual es obtenido sin ninguna dificultad al ser parte integral de la plataforma[55]. En cambio dicha funcionalidad (que en nuestra opinión es fundamental en una aplicación de alta concurrencia) lo tuvimos que implementar por medio de código en la versión JSP/Servlets dado que al momento de la elaboración del sistema no formaba parte de la especificación JDBC[56].

[55] De hecho es un servicio otorgado por COM+.

[56] De hecho el manejo de pools de conexiones ha sido integrado en la especificación JDBC 2.0 que forma parte de J2EE 1.3, sin embargo al momento del desarrollo de la versión JSP/Servlets de nuestro sistema, dicha funcionalidad no se tenía. Cabe hacer notar que algunos servidores J2EE sí proveen este servicio como valor agregado.

Si bien JDBC y ADO son muy similares en los objetos que presentan para modelar una fuente de datos. JDBC está más orientado a un acceso estático a las bases de datos relacionales. ADO por su parte resalta gracias a la funcionalidad de Recordsets actualizables y desconectables. Como tantas otras tecnologías de ambas plataformas, es poca la diferencia real entre las cualidades de ADO y JDBC para ofrecer conectividad a bases de datos; sin embargo, en nuestra opinión OLE DB y ADO son ligeramente superiores a JDBC, esto probablemente por que son tecnologías más recientes.

El desarrollo de la capa de dominio también nos dio la oportunidad de comparar algunas de las cualidades de ambas plataformas respecto al desarrollo de componentes que contienen las reglas de negocio a modelar por nuestra aplicación. En primer lugar hay que señalar que se decidió utilizar esquemas simples para la construcción de esta capa. Esto es que para la versión ASP/COM+ se desarrollaron componentes en Visual Basic aprovechando la sencillez de este lenguaje así como sus notables cualidades en el desarrollo rápido de aplicaciones. En cuanto a la versión JSP/Servlets se decidió implementar la capa de dominio a través de la construcción de Java Beans. Además ambos casos las responsabilidades asignadas a estos componentes de la capa de dominio fueron disminuidas favoreciendo en cambio la asignación de responsabilidades a la capa de control.

El hecho de decidir construir los componentes de la capa de dominio de la versión ASP/COM+ utilizando Visual Basic significa un inconveniente importante. Como ya se mencionó previamente esto implica que estos componentes tienen la restricción de manejar un esquema de hilado que solo es efectivo cuando la referencia a estos componentes no se mantiene entre dos páginas ASP distintas dado que cada página ASP tiene su propio hilo de ejecución. Entonces como los componentes hechos en Visual Basic soportan el esquema de hilado STA[57].

Cuando una referencia a un componente hecho en Visual Basic se mantiene entre diversos procesos de servicio de las peticiones de los clientes (request), dicha referencia estará traspasando su apartado. Esto significa que cada llamada al componente por una hebra de ejecución distinta a la hebra que creo la referencia requerirá de la actuación de un ente mediador encargado de resolver esas llamadas que traspasan el

[57] STA significa apartado de un solo hilo. Un apartado COM (COM apartment) es una agrupación de componentes COM que comparten las mismas características de concurrencia. Al estar dentro del apartado COM estos componentes mantienen protegido su estado interno del acceso concurrente. Un apartado del tipo STA solo permite que un solo hilo de ejecución acceda a los componentes que en él residen.

apartado actual lo que implica un serio menoscabo al desempeño del componente, a dicho mediador se le conoce como *proxy*[58].

La plataforma Windows DNA ofrece, sin embargo, opciones para desarrollar componentes que no manifiesten esta importante desventaja en el esquema de hilado de los componentes de Visual Basic. Estamos hablando de desarrollar componentes utilizando Visual C++ y la librería ATL[59] (ActiveX Template Library), con lo que se pueden desarrollar componentes con esquemas de hilado flexibles. El problema con esta alternativa es que la dificultad de desarrollar componentes de esta manera es considerablemente superior a aquellos desarrollados en Visual Basic. Por esto, en la práctica es bastante mayor la proporción de componentes que se desarrollan en Visual Basic que los que se desarrollan en Visual C++ y ATL.

Ahora bien, el diseño que se siguió en el desarrollo del caso práctico de la versión ASP/COM+, la limitación en el esquema de hilado de los componentes desarrollados en Visual Basic fue minimizada. Esto por que en vez de guardar referencias de los componentes directamente, lo que se está guardando es el estado serializado de los distintos componentes, el costo de serializar y deserializar el estado de estos componentes es menor al que se incurriría de mantener referencias que traspasaran los apartados de ejecución de la aplicación. La desventaja de este esquema es que terminan incrementando la complejidad inherente de los componentes desarrollados.

Por parte de la versión JSP/Servlets, como ya dijimos, también se optó por un esquema simplificado para construir la capa de dominio por medio de la construcción de Java Beans. La plataforma J2EE ofrece una tecnología robusta para el desarrollo de componentes que integren las reglas de negocio de una aplicación. Este esquema lo conforman los EJB (Enterprise Java Beans) los cuales representan la estrategia oficial recomendada para el desarrollo de la capa de dominio de las aplicaciones sobre esta plataforma.

Sin embargo, como ya se dijo, en la practica los EJB's son componentes de peso pesado dado que independientemente de necesitarlos o no, tienen asociados una serie de servicios como

[58] Un proxy es un objeto que provee la interfaz para acceder a otro objeto que corre en un entorno distinto, las llamadas al segundo objeto son interceptadas por el proxy el cual las adecua para poder acceder al segundo objeto respetando el entorno del mismo. Normalmente un proxy esta asociado con un tercer objeto llamado stub el cual se ubica en el entorno del objeto inicialmente llamado, en tal caso el proxy hace los llamados al stub el cual a su vez llama al objeto que realmente contiene la implementación de la función llamada.

[59] ATL significa librería ActiveX tipo template (ActiveX Template Library).

seguridad, manejo transaccional, etc. Entre otras razones, este ha sido uno de los motivos por los que el esquema EJB no se ha consolidado por completo como la opción indiscutible para el desarrollo de componentes de la capa intermedia sobre la plataforma J2EE[60]. Un problema importante de los EJB, es que su desempeño es inferior a otros esquemas de componentes ligeros tales como el modelar la capa de dominio utilizando Servlets y Java Beans solamente. Entonces cuando no se tiene los recursos para contar con un poderoso contenedor de EJB y, sin embargo, no podemos admitir un desempeño bajo o regular de los componentes de la capa intermedia; No queda otro remedio que construir esta capa con un esquema ajeno al de los EJB.

Podemos resumir que ambas plataformas ofrecen alternativas para desarrollar componentes de la capa de dominio mucho más robustas de las que se utilizaron en este trabajo de tesis. Sin embargo, las alternativas tomadas definitivamente son una muestra representativa de esquemas que no son solo válidos sino incluso ampliamente utilizados por la industria en el desarrollo de aplicaciones Internet/Intranet.

Basados tanto en el desarrollo hecho en este trabajo de tesis como también en nuestra experiencia personal afirmamos que ambas plataformas ofrecen esquemas de desarrollos de componentes de la capa de dominio o intermedia que son eficaces, sin embargo también creemos que gracias a las características propias del lenguaje Java, es en la plataforma J2EE donde más se facilita la observancia de muchos de los principios de la ingeniería de software y sobretodo de orientación a objetos mientras que con Windows DNA dichos principios son promovidos aunque no de la misma manera.

Durante el diseño se determino que para asegurar la correcta ejecución de los casos de uso se desarrollaría una capa adicional de control. Ahora bien, para la versión JSP/Servlets fue precisamente desarrollando servlets como se logró construir dicha capa y debemos decir que no se tuvo ninguna dificultad de desarrollar y codificar las clases de diseño de la capa de control. En cambio la capa de control fue desarrollada en la versión ASP/COM+ por medio de funciones usando el lenguaje VBScript y pudimos constatar las evidentes carencias de dicho lenguaje al hacerlo así.

Tal como fueron concebidas durante la etapa del diseño, las clases de control requieren interpretar el contenido de una petición HTTP pues solo de esta manera se pueden conocer los parámetros requeridos por

[60] Incluso varios de los proveedores de servidores de aplicaciones J2EE promueven tecnologías alternativas a los EJB para el desarrollo de componentes de la capa intermedia.

cualquiera de los casos de uso. Mientras que los servlets fácilmente cubren con esta funcionalidad sin detrimento en ninguna de las características que se definieron durante el diseño. Por otro lado, para la versión ASP/COM+, para acceder a la funcionalidad de interpretar cada uno de los parámetros de la petición HTTP se tuvo que implementar a la capa de control como funciones VBScript de una página ASP. Los lenguajes de guión pueden ser adecuados para programar el contenido dinámico de una página ASP, sin embargo es común que en aplicaciones ASP sea necesario utilizar a lenguajes de guión para funcionalidad ajena a la capa de visualización, y definitivamente esto termina evidenciando las indudables carencias de estos lenguajes que a su vez se traducen en carencias de la plataforma en sí misma.

La última parte del desarrollo en ambas versiones consistió en las páginas JSP y ASP que generan el contenido HTML que se desplegará en el browser. Ahora bien este aspecto, sin lugar a dudas, es en el que más semejanzas se tuvo en desarrollo final de ambas versiones[61]. Un buen diseño de una aplicación de múltiples capas debe buscar que en la capa de visualización se reduzca al mínimo la lógica a implementar a través de código. Para ello, se debe evitar incluir responsabilidades a la capa de visualización ajenas a las que tienen que ver con el desplegado del contenido visual así como con la interfaz con el usuario. Por lo tanto, los lenguajes de guión soportados por ASP no deberían de ser insuficientes para dicha tarea.

De hecho la simplicidad de los lenguajes de guión VBScript y JavaScript comparados con el lenguaje Java se puede considerar como una ventaja de la tecnología ASP/COM+ ante la tecnología JSP/Servlets. Sin embargo, en nuestra opinión esta simpleza sólo es relevante cuando se decide delegar la construcción de las páginas JSP's o ASP's a equipos cuya orientación sea más de diseño que de desarrollo. En tal escenario es muy probable que dicho equipo fuera más productivo en el entorno más simplificado provisto por ASP. Sin embargo, incluso limitándose a la lógica propia de la capa de visualización, es común que se requiera codificar funcionalidad relativamente compleja (sobretodo cuando estamos hablando de interfaces ricas), lo que hace poco recomendable el dejar completamente la responsabilidad de construcción de las páginas a un equipo cuya especialidad es el diseño y con pocas habilidades de desarrollo. Podemos decir que la situación ideal sería el contar con un equipo que cuente con especialistas en diseño

[61] El parecido no es coincidencial, la tecnología JSP surgió en gran medida para aprovechar parte del éxito que consiguió la tecnología ASP, por lo tanto el diseño de las JSP esta basado en gran medida con las mismas características de ASP. De hecho esta replica de las propiedades de ASP tiene un buen número de detractores en la propia comunidad de desarrollo con Java.

interactuando con otro equipo de desarrollo para que ambos equipos, en su conjunto, sean los responsables de la construcción de las páginas.

Una característica que no se utilizó en este trabajo de tesis lo constituyen los llamados tags personalizados (custom tags) de la tecnología JSP. Los tags personalizados proporcionan una forma de evitar mezclar el código que contiene la lógica de presentación con el contenido HTML de una página JSP. Si bien la principal característica de las tecnologías JSP y ASP es la de integrar en un solo lugar (la página) el código que controla el contenido dinámico con los elementos de HTML estáticos. Lamentablemente en una elevada proporción, las páginas ASP y JSP terminan siendo difíciles de mantener debido a presentar una complicada mezcla de código y contenido estático. Ahora bien, con la inclusión de los tags Personalizados se pretende eliminar al mínimo el código como tal en una página JSP substituyéndolo por un tag similar a los que forman parte de HTML con la diferencia de que el mismo tiene asociada cierta funcionalidad en particular.

Los tags personalizados permiten que un equipo especializado en diseño controle cuestiones de funcionalidad sin necesidad de que dicho equipo tenga contacto alguno con código. El precio a pagar al usar esta tecnología es que la misma implica un esfuerzo un poco mayor para el equipo de desarrollo. Esto es por que, desde un punto de vista simplista, resulta mucho más fácil simplemente codificar directamente la página que desarrollar todos los elementos que se requieren para implementar un tag personalizado. Sin embargo, cuando se consideran las dificultades inherentes al mantenimiento de una aplicación, los tags personalizados han demostrado ser una opción acertada hacia el incremento de la capacidad de mantenimiento de una aplicación.

En nuestra opinión, ambas plataformas ofrecen tecnologías muy parecidas y además muy parejas en el aspecto del desarrollo de página HTML dinámicas del lado del servidor. De hecho esto lo atestiguamos durante el desarrollo de la aplicación de foros de discusión dado que las páginas ASP y JSP de dicho desarrollo son prácticamente iguales, con la única diferencia de utilizar lenguajes distintos. Y si bien creemos que la simplicidad de los lenguajes de guión es una ventaja de esta tecnología, otras características de JSP (como los tags personalizados) son igualmente atractivas y terminan balanceando el cotejo entre las mismas.

Los lineamientos de diseño que dirigieron el desarrollo en ambas tecnologías, son como era de esperarse, lineamientos que incluyen cierto grado de subjetividad. Para disminuir el factor de subjetividad,

nos apoyamos en una serie de patrones de diseño que cuentan con una aceptación importante en la industria del desarrollo de software. Por otra parte, las características inherentes de ambas plataformas imponen a su vez una serie de lineamientos (lineamientos de la plataforma). Una vez desarrolladas ambas versiones, es interesante revisar la concordancia que se tuvo entre los lineamientos de diseño frente a los lineamientos de la plataforma; De esta manera podemos comparar la adecuación de dichas tecnologías a nuestro diseño original, sin embargo no se debe perder de vista el factor de subjetividad presente en los lineamientos de diseño.

Las capas de dominio y persistencia, si bien conforman el subsistema más importante de cualquier sistema incluyendo el nuestro. Para propósitos de la comparativa entre ambas tecnologías, no nos brindan muchos elementos para inclinar la balanza a una u otra de las tecnologías. Pues en general ambas tecnologías nos proporcionaron herramientas más que suficientes para cumplir con un desarrollo de las capas de dominio y persistencia no muy alejado del inicialmente definido por el diseño de la solución.

Debemos aclarar que en ninguno de los dos desarrollos de las capas de dominio y persistencia se emplearon esquemas demasiado complejos. Ambas tecnologías proveen de esquemas de desarrollo robustos cuyo objetivo es el permitir la construcción de sistemas de alto desempeño, con grandes capacidades de escalamiento o crecimiento o con la capacidad de responder a ambientes de elevada concurrencia. Por supuesto que el utilizar este tipo de esquemas dificulta el desarrollo de componentes pues se deben de seguir reglas mucho más estrictas que las que se siguieron en esta solución.

En el desarrollo de la capa de control de la versión ASP/COM+ es donde más disparidad encontramos entre los lineamientos de diseño y los lineamientos de desarrollo debidos a las características propias de la plataforma. Es aquí donde sin lugar a dudas la balanza se inclina más por la tecnología JSP/Servlets contra ASP/COM+ tomando en cuenta nuestro caso practico solamente. Los lenguajes de guión de ASP con los que tuvimos que construir la capa de control fueron sencillamente inadecuados para dicha tarea. Como ya dijimos es debatible el utilizar nuestros lineamientos de diseño para evaluar o comparar estas dos tecnologías, pero lo que no es tan debatible es que los lenguajes de guión de ASP resultan muchas veces insuficientes para implementar lógica de negocios de cierta complejidad, tanto es así, que uno de los objetivos que tuvo Microsoft en el desarrollo de ASP.NET es la de

permitir la utilización de lenguajes mucho más robustos que los lenguajes de guión en esta nueva tecnología.

Podemos decir que el acto de desarrollar el sistema de foros de discusión en ambas tecnologías/plataformas nos brindó la oportunidad de revisar muchas de las cualidades y particularidades de las mismas. Uno de los objetivos de desarrollar este caso práctico es el de encontrar y mostrar aspectos de ambas tecnologías que difícilmente podrían obtenerse de la simple comparación hipotética de las mismas. Además al haber llevado a cabo dicho desarrollo se pudieron exponer algunas facetas y tecnologías que no están relacionadas propiamente con alguna de las plataformas y que sin embargo tienen una importancia fundamental en el desarrollo de aplicaciones Internet/Intranet. Tal es el caso del uso de patrones de diseño, el uso de arquitecturas de múltiples capas, el empleo de DHTML, etc.

Aprovechando los conocimientos obtenidos al desarrollar de este caso práctico, así como de la experiencia profesional adquirida de manera personal, procederemos ahora a resumir algunos de los atributos principales que ambas tecnologías y plataformas poseen.

5.3 Comparación de atributos.-

Ya hemos señalado en este capítulo las ventajas y desventajas que cada tecnología presentó en la construcción del sistema de foros de discusión. Definitivamente los casos prácticos auxilian notablemente la tarea de comparar tecnologías como las que aquí nos conciernen, sin embargo, no podemos perder de vista que la comparación resultante de un caso práctico está acotada por los límites propios del caso práctico. Es poco realista el suponer que se puede encontrar un caso práctico que cubra por completo todos los atributos que cualquiera de estas dos tecnologías poseen. En cambio es razonable presumir que un caso práctico solo revela parcialmente las capacidades de cualquiera de estas dos tecnologías y que, además, la elección de un caso práctico no es inmune a la posibilidad de parcialidad hacia alguna de las dos tecnologías.

Una importante proporción de nuestro esfuerzo ha sido en el sentido de disminuir este factor de parcialidad eligiendo un caso práctico en el cual se disminuya la posibilidad de formar una opinión sesgada. En parte ésta es la razón de que el caso práctico seleccionado no emplea a fondo muchas de las características más potentes de ambas tecnologías pues esto lo haría más susceptible de favorecer a alguna de las mismas; esto

por que normalmente este tipo de características son poco genéricas si no es que nativas de alguna de las dos tecnologías.

El costo de elegir un caso práctico en donde no se aprovechan todas las características de la tecnología es que con dicho caso práctico estamos limitando la comparación a solamente aquellas características comunes a ambas tecnologías y que fueron cubiertas por el caso práctico. Sin embargo, resulta importante cubrir el mayor espectro posible de las características de ambas tecnologías para que nuestro juicio tenga una mayor validez a la de simplemente basar el mismo al desarrollo de un caso práctico. Por lo tanto presentaremos ahora una serie de factores de comparación que consideramos es importante tomar en cuenta para emitir un juicio acerca de la comparación entre ambas tecnologías[62].

5.3.1 Desempeño.-

Sin lugar a dudas el desempeño resulta ser un atributo fundamental a la hora de evaluar cualquier tecnología de información o plataforma informática. Muchos incluso considerarán que cualquier otro atributo es un valor agregado pero que el desempeño es el aspecto fundamental que determina si una tecnología o plataforma es viable ó no. Lamentablemente la medición del desempeño de estas dos tecnologías presenta serias dificultades (algunas de estas dificultades son comunes a ambas). Algunas de las principales dificultades que presenta la medición del desempeño son las siguientes:

- El desempeño de las aplicaciones sobre una u otra tecnología depende en gran medida de las características del desarrollo hecho por nosotros, esto es que los resultados que se obtuvieran de la medición llana del desempeño en aplicaciones sobre estas tecnologías serían atribuibles en un amplio porcentaje a los aciertos y errores cometidos por nosotros durante el diseño y desarrollo de la aplicación.
- Muy relacionado con el punto anterior tenemos el hecho de que las dos tecnologías imponen una serie de lineamientos de la plataforma, ahora bien como pudimos comprobar en los capítulos anteriores, el desarrollo de hecho se rige por la combinación de lineamientos de diseño junto con los lineamientos de la plataforma y por lo tanto el desarrollo final deberá encontrar un punto medio entre los mismos. Esto

[62] Vale la pena tener en cuenta que muchos de los factores que serán presentados tienen relación estrecha entre ellos, esto es que la marginalidad o abundancia en un factor probablemente influirá en cierta medida en alguno de los otros factores también mostrados.

significa que un factor preponderante en el desempeño final de la aplicación dependerá de que tanto se siguieron los lineamientos impuestos por la tecnología y es aventurado creer que en ambas tecnologías se cumplen con la misma precisión.

- Mientras que la tecnología ASP/COM+ es como ya dijimos un único producto, por otro lado, la tecnología JSP/Servlets es una especificación que siguen una serie de productos, los cuales poseen características específicas que influyen en el desempeño obtenido. Probablemente la mejor manera de hacer una evaluación imparcial de medición sería utilizando el cociente entre el desempeño y el costo. Ahora bien los requisitos para realizar una evaluación de este tipo están por fuera de los alcances de este trabajo de tesis.
- A pesar de los aciertos y errores cometidos durante el diseño y el desarrollo de la aplicación en ambas tecnologías, hay otros factores que influyen tanto o más que las cualidades del desarrollo o de la tecnología por sí misma. Por ejemplo en una aplicación de este tipo el acceso a la base de datos tiene un peso crucial en el desempeño obtenido. Esto implica que al medir el desempeño de la aplicación en una u otra tecnología estamos simultáneamente midiendo el desempeño conjunto de una serie de servicios y componentes ajenos a nuestra área de interés.

Para concluir con el aspecto de la medición del desempeño, vale la pena remarcar que como sería de suponer, los dos propulsores de ambas tecnologías: Microsoft y SUN, anuncian que "su tecnología" presenta un mejor desempeño ante la otra, y ambos dan sus mediciones y argumentos: Por un lado las JSP se compilan a Servlets antes de su ejecución mientras que las ASP son interpretadas cada vez, por otra parte los ASP corren en código nativo mientras que los Servlets y JSP corren normalmente en "bytecodes"[63], etc. Sin pretender caer en esta polémica mercadotécnica más que técnica, en nuestra opinión muchas veces tienen más peso en el desempeño final la calidad del diseño y desarrollo efectuado que el peso que tienen estas características propias de cada una de las plataformas.

Por lo tanto el hacer un estudio comparativo del desempeño de las dos tecnologías en nuestra opinión plantea dificultades cuyo alcance está por encima de los recursos con que se contó para la realización de este trabajo de tesis. Cabe mencionar, sin embargo, que existen disponibles estudios comparativos de ambas plataformas en los que se

[63] EL código intermedio e independiente de la plataforma el cual es interpretado por las máquinas virtuales de Java.

mide el desempeño, estos estudios si bien tienen sus detractores[64] podría dar una idea de cual es el desempeño real aproximado de cada tecnología. Muchos de estos estudios que miden el desempeño muestran un desempeño ligeramente superior de la tecnología JSP/Servlets sobre la tecnología ASP/COM+ (esta última usando componentes escritos en Visual Basic), cuando se utilizan componentes EJB el desempeño se reduce un poco pero sigue siendo muy similar a ASP/COM+, en dichos estudios también se muestra que aplicaciones hechas con la tecnología ASP/COM+ pero donde los componentes se construyen con VC++/ATL el desempeño es superior al equivalente en JSP/Servlets.

Concluimos por lo tanto que desafortunadamente no contamos con los medios para determinar cual tecnología ofrece un mejor desempeño. Sin embargo, tenemos fuertes indicios de que la diferencia en los desempeños de ambas plataformas es reducida. De lo que si no nos cabe duda es que la definición una arquitectura adecuada normalmente representará el factor con mayor proporción en el desempeño final de una aplicación.

5.3.2 Curva de aprendizaje.-

Cuando un individuo es introducido a una nueva habilidad o conocimiento, es común que requiera de varias sesiones de aprendizaje o entrenamiento para adquirir esa nueva habilidad o conocimiento. A esto se le conoce como la curva de aprendizaje y se puede resumir como el tiempo que se necesita para que un individuo adquiera una habilidad o conocimiento hasta ese momento desconocido. Se le conoce como curva de aprendizaje por que es posible trazar en un gráfico[65] la relación entre el tiempo invertido en aprender el nuevo conocimiento o habilidad (normalmente representado en el eje horizontal) frente al nivel de conocimiento o dominio adquirido (en el eje vertical).

Adentrándonos en la teoría subyacente a la llamada curva de aprendizaje, se dice que existe un punto al que se le conoce como asimptote[66] que representa un punto donde no existe posibilidad de incrementar el nivel de dominio adquirido. El asimptote representa el nivel máximo de dominio de dicha habilidad o conocimiento, mientras más cerca se encuentra el nivel de dominio adquirido del asimptote horizontal de la curva de aprendizaje, el incremento del mismo crecerá de una manera cada vez menos importante.

[64] Los cuales son normalmente aquellos que apoyan a la tecnología cuyo desempeño resulta superado en tales estudios comparativos.
[65] De hecho estos tipos de gráficos son comunes en estudios de sicología.
[66] En geometría un asimptote representa una curva que se aproxima a otra curva dada sin nunca alcanzarla.

Definitivamente la curva de aprendizaje de la tecnología ASP/COM+ es más inmediata que la de la tecnología Servlets/JSP, esto es que para un individuo que desconozca completamente ambas tecnologías[67] el número de sesiones de aprendizaje o tiempo invertido será menor en la tecnología ASP/COM+ que en la de Servlets/JSP. Sin embargo, la curva de aprendizaje de la tecnología ASP/COM+ es afectada por algunos de los defectos intrínsecos de dicha tecnología; haciendo la selección de un ganador en este factor algo más difícil de lo que inicialmente se podría pensar.

En gran medida la sencillez de la tecnología ASP/COM+ estriba en que es mucho menos restrictiva que su contraparte de JSP/Servlets. Esto no solo por que esta tecnología no obliga que un proceso de desarrollo se circunscriba a una metodología rigurosa como por ejemplo la programación orientada a objetos[68]. En cambio con la tecnología JSP/Servlets el propio lenguaje limita al desarrollo a apegarse a reglas más estrictas[69].

Posteriormente veremos a la capacidad de escalamiento como factor de comparación, pero es importante resaltar que para poder desarrollar soluciones que escalen bien en la tecnología ASP/COM+ si se requiere cumplir con una serie de reglas estrictas, entonces se presenta un fenómeno interesante en cuanto a la curva de aprendizaje:

Mientras que la curva de aprendizaje de la tecnología Java se comporta sin sobresaltos desde la fase de desarrollo de soluciones básicas hasta el desarrollo de soluciones complejas en donde es primordial el escalamiento manteniendo el desempeño ante una elevada concurrencia. Por ejemplo a través de distribución de la carga por medio de clusters, etc., al igual que la integración de robustos esquemas de prevención y recuperación de fallas; Esto es otra historia con ASP/COM+. No solo el programador está obligado a cumplir con reglas que no se tenían para el desarrollo de soluciones básicas, sino que la flexibilidad de la tecnología parece estrecharse en estos escenarios. Por ejemplo, el desarrollar componentes COM+ en un entorno tan sencillo como Visual Basic deja de ser una opción válida debido a que este lenguaje no provee de los esquemas de apartados COM más potentes

[67] Obviamente existen factores como los antecedentes individuales que influyen considerablemente en las curvas de aprendizaje que un individuo en particular tendría ante ambas plataformas.
[68] De hecho es posible que un individuo sin educación formal en programación desarrolle usando ASP.
[69] Aunque Java es un lenguaje orientado a objetos, no es suficiente desarrollar en Java para asegurar un desarrollo orientado a objetos. Es más importante el compromiso que un equipo de desarrollo toma respecto a una arquitectura de diseño y desarrollo que las características propias del lenguaje de programación seleccionada por dicho equipo.

que se requieren para una aplicación de alta concurrencia. Si bien es completamente factible desarrollar soluciones muy robustas en la tecnología ASP/COM+ (o hablando de manera más genérica en la plataforma Windows DNA), la dificultad es notablemente mayor que en la tecnología Java, en donde muchas de estas responsabilidades las toma la plataforma propiamente.

Entonces nuestra conclusión final en este factor es que la curva de aprendizaje para desarrollar soluciones de nivel de básico a medio es notablemente mejor en la tecnología ASP/COM+. Mientras que si la curva de aprendizaje es respecto al desarrollo de soluciones robustas es la tecnología JSP/Servlets la que muestra una ligera ventaja[70].

5.3.3 Herramientas de desarrollo.-

Un factor de comparación muy importante lo constituyen las herramientas de desarrollo pues estas determinan en gran medida la productividad durante la fase de desarrollo. Independientemente de características intrínsecas de las tecnologías, es primordial contar con herramientas de desarrollo que permitan un eficaz aprovechamiento de los recursos humanos disponibles en un proyecto de desarrollo de software[71].

El contar con herramientas de desarrollo adecuadas permite que se agilicen las labores cotidianas y extraordinarias que realiza un equipo de desarrollo. En la situación actual de la industria, el contar con herramientas de desarrollo poderosas ha dejado de ser un lujo o un factor secundario, principalmente por que la complejidad de los sistemas informáticos se incrementa cada vez más mientras que los plazos de desarrollo se mantienen o incluso se reducen. Esta situación sería insostenible si a la par del incremento en complejidad de los sistemas no se tuviera un incremento en la productividad de los equipos de desarrollo, este incremento se debe en buena medida a la utilización de herramientas de desarrollo cada vez más efectivas.

Podemos resumir el estado actual de las herramientas de desarrollo con que cuentan ambas tecnologías, de la siguiente manera:

[70] En el caso particular del sistema que hemos desarrollado la solución en si presenta un nivel de complejidad medio, lo que significa que son muy similares las curvas de aprendizaje en ambas plataformas para desarrollar un sistema con las característica que elegimos para este trabajo de tesis.

[71] En el ámbito del desarrollo de software, los recursos humanos constituyen, sin duda alguna, el recurso más valioso.

La tecnología ASP/COM+ cuenta sobre todo con la suite Microsoft Visual Studio la cual concentra un rico conjunto de herramientas bastante sólidas las cuales han tenido una evolución considerable desde su surgimiento por lo que en este momento se encuentran en un confortable nivel de maduración. Visual Studio permite desarrollar cada una de las capas de la aplicación y la integración entre las múltiples herramientas es algo sobresaliente en esta suite. Adicionalmente la tecnología ASP/COM+ provee de herramientas de terceros bastante poderosas[72] que si bien no abarcan todo el espectro de la tecnología ASP/COM+, dentro de sus alcances proveen de una alternativa muy competitiva a la suite de Microsoft.

Por otro lado, la tecnología JSP/Servlets o más propiamente J2EE cuenta con múltiples herramientas de diferentes compañías. No existe un producto que sobresalga pero en cambio existen varios de ellos con muy buenas cualidades. Sin embargo. Comparativamente no hay un producto que ofrezca el mismo número de herramientas que Visual Studio presenta para la otra tecnología. Y aunque, si bien, el conjunto de herramientas para la tecnología JSP/Servlets es similar o incluso mayor; muchas veces los equipos de desarrollo se ven obligados a usar más de un producto pues requieren herramientas que no se encuentran por completo en una sola oferta. El costo de esto proviene del hecho de que muchas veces resulta difícil el integrar las herramientas entre diferentes productos o suites.

Lo anterior describe una desventaja de la tecnología JSP/Servlets, pues en ocasiones el equipo de desarrollo estará limitado a usar un solo producto cuyas herramientas de desarrollo disponibles no son las mejores o probablemente son insuficientes. Mientras que en otros casos, parte de la labor del equipo de desarrollo se debe reservar para la integración entre las herramientas de múltiples productos.

Otra desventaja de la tecnología JSP/Servlets es que pese a que en teoría parte de una tecnología universal basada en estándares, en la práctica se da el fenómeno de que comúnmente las compañías que elaboran herramientas de desarrollo, también son propietarias de servidores de aplicación; los cuales si bien se adhieren a los estándares definidos por Sun, por otro lado incluyen peculiaridades que lógicamente buscan diferenciar su producto del de la competencia. Ahora bien es obvio que las compañías proveedoras de las herramientas de desarrollo incluirán características específicas de su servidor de aplicaciones en la suite de desarrollo como, por ejemplo, herramientas para facilitar la

[72] Destacando las herramientas producidas por Borland, por ejemplo: C++ Builder y Delphi.

implantación de los componentes de software en el servidor de aplicaciones propio.

Independientemente de la “universalidad” de Java, existen las suficientes diferencias entre los diversos servidores de aplicaciones como para hacer que la tarea de instalación y puesta a punto, así como la de migrar una aplicación de un servidor a otro, sea una labor relativamente ardua. Esto es que existe cierto acoplamiento (relativamente reducido pero que no podemos ignorar) entre una aplicación típica J2EE y un servidor de aplicaciones específico. Por lo tanto un factor de mucho peso al momento de tomar una decisión respecto a la elección de un producto o suite de desarrollo específica es sin lugar a dudas la integración que presentan las herramientas de desarrollo ofrecidas con el servidor de aplicaciones donde se ejecutará la solución. En un buen número de ocasiones (probablemente la mayoría) se termina eligiendo de facto a aquellas herramientas que provienen de la misma compañía del servidor de aplicaciones.

El fenómeno anterior es algo natural y que es de esperarse pues evidentemente cada proveedor de herramientas de desarrollo que a su vez lo sea de servidores de aplicaciones, buscará que ambos productos alcancen la mayor participación de mercado posible en cada uno de sus nichos. Dado que una manera de diferenciar a su favor ambos productos es ofrecer una alta integración entre los mismos, entonces es normal que el proveedor integre ambos productos de manera tal que la decisión de usar uno de los productos influya notablemente en la decisión del otro.

Lamentablemente este fenómeno ocasiona que la decisión de elegir una herramienta de desarrollo u otra, se base en gran medida en el acoplamiento de esa herramienta respecto al servidor de aplicaciones a utilizar; en detrimento de la comparación simple entre las cualidades intrínsecas de cada herramienta. Ahora bien, este fenómeno no es exclusivo a la tecnología JSP/Servlets, también se da en ASP/COM+ pero con la diferencia que como solo hay una implementación de la plataforma, entonces el acoplamiento a ésta es de hecho una característica intrínseca de la herramienta. En la tecnología de Java en cambio, comúnmente los equipos de desarrollo cambiarán constantemente de herramientas de desarrollo entre distintos proyectos lo que tiene un costo en capacitación y probablemente en menor nivel de dominio de la herramienta.

Nuestra opinión es que la tecnología ASP/COM+ es la ganadora en el factor de herramientas de desarrollo aunque la diferencia no es

abrumadora dado que constantemente las herramientas de ambas tecnologías están evolucionando e incorporando nuevas y más poderosas características.

5.3.4 Cuota de mercado.-

La cuota de mercado es otro factor de considerable importancia al establecer una comparativa entre ambas tecnologías. Sin embargo, vale la pena manifestar que este factor es en realidad circunstancial, pues en realidad la cuota del mercado depende a su vez de una serie de factores muchos de los cuales nada tienen que ver con características o cualidades propias de la tecnología. Uno de estos factores, por ejemplo, sería la efectividad mercadotecnia o el control de la empresa que instituye la tecnología sobre algún nicho tecnológico o de mercado relacionado con la tecnología. El carácter circunstancial de la cuota de mercado se debe a que una elevada proporción de los factores que inciden en la cuota de mercado son de una naturaleza extrínseca a las cualidades de la tecnología en sí.

Con todo y el carácter circunstancial de este factor, este no deja de ser un elemento primordial para que la industria se incline por una u otra tecnología. Esto constituye un círculo virtuoso o vicioso según se quiera ver. Las razones de este fenómeno son muchas, estas son algunas de estas razones:

- Las empresas incurren en costos cada vez que adoptan tecnologías nuevas. El costo de evaluar una tecnología que no será finalmente aprovechada se considera injustificado. Entonces las empresas tienen pocos incentivos para embarcarse en la tarea de evaluar tecnologías y prefieren aprovechar evaluaciones de terceros, entonces la cuota de mercado es en cierta medida un indicativo de la evaluación que la industria en general le da a una tecnología.
- Los costos de desarrollo al usar tecnologías/plataformas de amplia aceptación en el mercado son consistentemente menores. Esto porque existe mayor disponibilidad de servicios y recursos (por ejemplo programadores), existe también una mayor competitividad la cual se refleja en el precio, etc.
- Se disminuyen los riesgos al usar tecnologías de amplia aceptación pues, en teoría, estas serán tecnologías que han

sido probadas por un mayor número de empresas[73]; es decir, que han llevado a cabo un proceso de maduración más intenso.
- La mayor aceptación de una tecnología implica que hay más dinero invertido por la industria en la misma. Lo que permite suponer que la vida útil de dicha tecnología será mayor dado que se tiene el bastimento monetario para sostenerla por más tiempo.

Las características propias de cada tecnología tienen una incidencia notable respecto a la cuota de mercado. Adicionalmente resultaría desacertado el analizar como un solo mercado a todas las áreas donde estas tecnologías son utilizadas, existen nichos de mercado en donde una de las tecnologías domina mientras que en otros es la otra la que lleva la delantera.

El medir la cuota de mercado de estas dos tecnologías no es una tarea sencilla, ni desde el punto de vista metodológico hasta el punto de vista práctico. El mercado es muy heterogéneo y además los recursos necesarios para abordar un estudio de mercado de este tipo son considerables, ahora bien, existen estudios cuyos resultados son públicos[74] que muestran algunas cifras indicativas a cerca de la cuota de mercado de estas tecnologías.

Al parecer la tecnología ASP lleva una importante ventaja sobre la tecnología JSP en el ámbito de Internet. Esto es que el número de páginas que aprovechan la funcionalidad de ASP sobrepasa ampliamente a las que utilizan a JSP. De hecho en este ámbito son otras tecnologías (PHP, ColdFusion, etc.) las que más compiten con ASP como medios de desarrollar contenido dinámico del lado del servidor. Algunas explicaciones de esta importante ventaja de ASP tienen que ver con el mayor tiempo en el mercado que esta tecnología posee. Otra razón la constituye el hecho de que ASP aprovecha a su vez la cuota de mercado que tiene el servidor Web Microsoft IIS, el cual posiciona a la tecnología a ASP como la alternativa obvia de desarrollo de contenido dinámico en dicho servidor.

Si bien ASP domina a JSP en el desarrollo de páginas web sobre Internet. Por otro lado en los desarrollos hechos en los grandes corporativos o en los desarrollos de tipo Intranet, es JSP quien tiene una clara ventaja sobre ASP. En este tipo de escenario las páginas JSP/ASP

[73] Aunque las constantes vulnerabilidades encontradas en un servidor web de importante presencia en el mercado parecerían contradecir o al menos ignorar este punto.
[74] De hecho existen estudios de mercado que probablemente muestren una visión del mercado más clara a la de los que nosotros pudimos tener acceso, lamentablemente dichos estudios no son públicos dado que tiene un costo el obtener sus resultados, costo bastante elevado por cierto.

son normalmente la portada de robustos sistemas los cuales hacen un mucho mayor uso de los demás elementos de la plataforma J2EE o Windows DNA[75]. Si bien parece haber consenso de que JSP domina este nicho, es difícil determinar en que medida lo hace pues Windows sigue siendo el cliente dominante en la mayoría de las empresas, las cuales si bien oficialmente recurren a J2EE como su arquitectura para sistemas distribuidos, simultáneamente mantienen sistemas (de menor escala) desarrollados en la plataforma Windows DNA.

Otro elemento determinante en la cuota de mercado lo constituye la fuerza laboral que está atrás de cada tecnología. Una empresa preferirá normalmente abarcar tecnologías con las que sea fácil o poco costoso obtener recursos humanos. Estudios realizados por Gartner sobre una muestra de empresas y corporativos, dicen que en la actualidad un 70% de empresas desarrollan sobre Java y también un 70% lo hace utilizando Visual Basic[76]. Estos números son un indicativo de la oferta de desarrolladores en estos lenguajes. Los desarrollos hechos en Java abarcan a desarrollos J2EE como de otro tipo al igual que los de Visual Basic incluyen tanto a sistemas cliente/servidor y Win32 como la utilización de múltiples capas y ASP. El estudio sugiere que los desarrollos en el lenguaje Java en los próximos años tendrán que ver cada vez más con J2EE, mientras que los desarrollos en Visual Basic evolucionarán cada vez más en esquemas distribuidos Windows DNA pero es más probable que se orienten bajo la nueva plataforma Microsoft .NET.

En nuestra opinión, las cuotas de mercado de las tecnologías ASP/COM+ y JSP/Servlets serán mejor cuantificadas si se determina específicamente el nicho de mercado al que se tiene interés en evaluar las cuotas de ambas tecnologías dada la evidente heterogeneidad entre todos los nichos donde ambas tecnologías son aplicables. Resumiendo (y quizás simplificando excesivamente), podemos decir que ASP mantiene una mayor cuota de mercado en desarrollos con relativamente poca complejidad mientras que es JSP el que sobresale cuando se trata de sistemas de mayor robustez.

5.3.5 Portabilidad.-

La portabilidad es un factor que puede no ser importante para el desarrollo de un sistema de software o que por el contrario puede ser un

[75] En dichos desarrollos el énfasis esta dado en la capa intermedia (reglas de negocio) más que en la capa de presentación (ASP/JSP).

[76] Estos valores no son exclusivos y de hecho demuestran nuestra afirmación de que en la práctica los corporativos mantienen una mezcla de plataformas y arquitecturas de desarrollo.

factor primordial, valdría la pena dar una definición de lo que es la portabilidad:

La portabilidad de una aplicación es la medición de la dificultad que presenta el mover e implantar una misma aplicación de un entorno original a otro distinto. Se dice que una aplicación es portable cuando el esfuerzo de adaptarla para que corra en un entorno diferente al original es menor que el esfuerzo que requeriría volverla a desarrollar en el nuevo entorno.

No hay contienda en este factor, las aplicaciones ASP/COM+ están limitadas a correr en el entorno que constituye el sistema operativo Microsoft Windows[77] en especial en sus modalidades de servidor. Por otra parte la tecnología JSP/Servlets aprovecha la indiscutiblemente enorme portabilidad de la tecnología J2EE. Ya mencionamos que la migración entre servidores de aplicación de esta plataforma no es transparente, pero definitivamente el costo de adaptación para migrar una aplicación de un servidor de aplicaciones a otro es mucho menor al costo de desarrollar la aplicación desde cero.

Ahora bien señalamos en un principio que este es un factor que puede no ser importante para el desarrollo de un sistema de software y esto se debe a que no todos los sistemas requieren o está previsto que se puedan portar entre diferentes entornos[78]. Por lo tanto en aquellos sistemas donde la portabilidad sea una necesidad fundamental, la tecnología JSP/Servlets se constituirá como la única alternativa viable para ser tomada en cuenta.

5.3.6 Capacidad de Escalamiento.-

La capacidad de escalamiento es un factor que muestra que tan factible es mantener el desempeño de una aplicación a niveles aceptables mientras la carga o utilización de la aplicación aumenta. La capacidad de escalamiento depende tanto de características propias de la aplicación, como de características intrínsecas de las tecnologías subyacentes. Obviamente para comparar las tecnologías estamos interesados específicamente en las características pertenecientes a la tecnología y no a las de una aplicación en particular.

[77] Existen soluciones de terceros (Chili!soft y Halcyonsoft) que permiten la implementación de la tecnología ASP en diversos sistemas operativos, sin embargo, los productos que ofrecen no resuelven satisfactoriamente la portabilidad de componentes COM; Por lo tanto, es justificado afirmar que la plataforma ASP/COM+ solo esta soportada por el sistema operativo Windows.

[78] Aunque el hecho que durante la etapa que va desde el análisis al desarrollo de una aplicación no se prevea que esta requiera ser portada a otro entorno, esto no significa que durante toda la vida útil del sistema no pueda surgir esta necesidad.

Ambas tecnologías y sus tecnologías poseen los suficientes atributos para permitir desarrollar soluciones sumamente escalables cuyo desempeño se mantenga ante una elevada concurrencia. La tecnología de servidores de alto desempeño para el manejo de aplicaciones críticas ha ido avanzando considerablemente, reduciendo, además, los costos subyacentes. Con anterioridad el esquema prevaleciente era el emplear equipos de cómputo potentes y masivamente redundantes. El costo de estos equipos era (y sigue siendo) considerable pero era el precio que se tenía que pagar para tener un esquema de alto desempeño y con pocas probabilidades de fallo en donde una aplicación podía escalarse tanto como el equipo podía soportar.

Este era un esquema basado principalmente en el hardware. Ahora bien en estos momentos la tecnología de componentes distribuidos y redundantes es aprovechada para producir esquemas con tanto o más rendimiento que los que anteriores se tenían pero a un costo bastante menor. Para ello se integran esquemas que consisten tanto en hardware como software destacando por ejemplo la utilización de clusters donde cada nodo del cluster es un equipo de hardware mucho menos costoso que uno de los mainframes previamente utilizados. Se tienen ahora equipos redundantes que, además, se distribuyen la carga de manera que se consigue tener un rendimiento máximo y una disminución de la probabilidad de caída o falta de disponibilidad equiparable e incluso superior a los que se tenían con esquemas anteriores.

Ahora bien, estos esquemas mixtos de software y hardware actuales implican que las aplicaciones desarrolladas para funcionar sobre los mismos y así poder escalar, requieren apegarse a reglas más estrictas que las que inicialmente se tenían. Como ejemplo clásico de esto tenemos que una aplicación altamente escalable debería permitir que el estado de una sesión se pueda trasladar de un nodo a otro dentro de un cluster. Esto implica que el estado de la sesión no debe mantener referencias de memoria absolutas o que dichas referencias deben ser factibles de ser trasladadas junto con la sesión que las mantiene; esto implica también que el estado de la sesión debe de fácilmente serializado para transportarlo de un nodo a otro. Estamos hablando aquí de cuestiones de bajo nivel a las que mientras menos se necesite tener que acceder directamente (esto es que la propia tecnología se encargue de ellas) se simplificará notablemente el desarrollo de una aplicación escalable.

Como ya mencionamos, la curva de aprendizaje de ASP/COM+ (o en realidad Microsoft DNA) se dificulta cuando se trata de desarrollar aplicaciones con estas características, la razón de esto es principalmente

por que los lineamientos de desarrollo son mucho más estrechos para una aplicación de este tipo. El problema con el factor de escalamiento es que comúnmente se requiere escalar aplicaciones que inicialmente no fueron diseñadas tomando en cuenta la carga a las que después se les quiere someter. Y dado que es mucho más complicado desarrollar una aplicación que se apegue a los lineamientos más estrictos de la tecnología ASP/COM+, cuando dicha aplicación requiere ser escalada comúnmente se da el fenómeno de tener que volver a desarrollar muchos componentes que no se adaptan al nuevo esquema.

En cambio en la tecnología JSP/Servlets son menos diferentes los lineamientos para desarrollar una aplicación relativamente sencilla que una compleja y por lo tanto en términos generales las aplicaciones hechas en esta tecnología escalan (la mayoría de las veces) más fácilmente. Por lo tanto, en nuestra opinión la tecnología JSP/Servlets muestra una relativa ventaja en el factor de la capacidad de escalamiento.

5.3.7 Documentación disponible.-

Un factor de gran importancia en cualquier tecnología o plataforma lo constituye la disponibilidad en cantidad y calidad de documentación (oficial y de terceros) sobre la misma. Cuando la disponibilidad de documentación es abundante, esto brinda un importante respaldo a la labor de delinear una arquitectura del sistema durante el diseño tanto como para utilizar adecuadamente los servicios que presta la tecnología y codificar algoritmos que se apeguen a los lineamientos de la tecnología durante el desarrollo.

La documentación presenta entre otras cosas las mejores prácticas de diseño y desarrollo de acuerdo a la tecnología, entonces el contar con documentación rica y abundante nos permite determinar si un sistema se apega o no a los lineamientos de la tecnología. La documentación de la tecnología debe de ser un marco de referencia que nos permita evaluar (de una manera sencilla) las cualidades e imperfecciones que un sistema presenta. Para ello la documentación debe de tener elementos generales para poder calificar al sistema en su conjunto así como elementos particulares para poder calificar un componente o subsistema determinado.

La cantidad y calidad de la documentación disponible es un factor muy relacionado con la curva de aprendizaje. Las razones de esto son evidentes, incluso contando con buena documentación es posible contrarrestar la dificultad intrínseca de una tecnología. También está

relacionada con la cuota de mercado pues mientras más aceptación tenga en la industria alguna tecnología, esto hará que se genere más documentación de terceros, la cual a menudo es tanto o más importante que la documentación oficial.

Por supuesto la utilidad de la documentación no termina simplemente en lo que hemos asentado aquí. Simplemente hemos dado algunos elementos que en nuestra opinión son suficientes para revelar la importancia de la disponibilidad de documentación como factor de comparación. Ahora bien respecto a la comparación entre nuestras dos tecnologías, la documentación disponible muestra la alta aceptación que ambas tecnologías tienen en la industria (lo que hace que se genere una demanda por documentación). Además, las dos empresas patrocinadoras de las dos tecnologías y sus tecnologías subyacentes proveen a su vez de excelente calidad y gran cantidad de documentación oficial[79].

5.3.8 Apoyo a la orientación a objetos.-

Definitivamente las metodologías de orientación a objetos se han constituido como las que reciben la mayor atención por parte de la industria en el contexto actual. Por momentos, incluso parece ser no del todo justificada la preponderancia que se le da a las metodologías de orientación a objetos sobre otros aspectos de la ingeniería del software. Sin restarle los meritos a los aspectos de la ingeniería del software complementarios a la orientación a objetos no podemos negar que esta última constituye una importante respuesta a los problemas que enfrenta el desarrollo de software en la actualidad.

Por supuesto que no basta con que una tecnología apoye a la orientación a objetos para que los sistemas sobre dicha tecnología aprovechen las ventajas que la orientación a objetos presenta. De hecho es más importante el compromiso que un equipo de desarrollo toma para cumplir con una metodología orientada a objetos. Sin embargo, es una gran ventaja el contar con todo aquello que facilite el seguir con dicha metodología. Por esto el apoyo a la orientación a objetos constituye un importante factor a ser tomado en cuenta en la comparación entre las dos tecnologías que nos interesan en este trabajo de tesis.

[79] Obviamente una razón de la gran cantidad de documentación disponible sobre las dos tecnologías que nos interesan es el hecho de que con ello se fortalece la penetración de dichas tecnologías. Desde un punto de vista pragmático esto demuestra el interés por dominar el mercado de aplicaciones empresariales y de las tecnologías de Internet y la Web por parte de las empresas que impulsan las dos tecnologías.

Ya antes habíamos resaltado el hecho que mientras que la tecnología JSP/Servlets se suscribe al uso de Java el cual es un lenguaje completamente orientado a objetos, mientras que la tecnología ASP/COM+ está basada en un modelo de componentes en donde la orientación a objetos es más recomendada que decididamente impulsada, ya no digamos exigida. Definitivamente hay otras metodologías y filosofías, como por ejemplo el desarrollo rápido de aplicaciones, que tiene mayor peso en la tecnología ASP/COM+ que la orientación a objetos.

Si bien es cierto que es perfectamente factible hacer un desarrollo profundamente orientado a objetos (sobre en la capa de dominio) sobre la tecnología ASP/COM+, también lo es el hecho de que resulta más natural hacerlo en la tecnología JSP/Servlets pues el propio lenguaje facilita esta labor al contar con todos los elementos que un lenguaje completamente orientado a objetos posee. Por lo tanto nuestra opinión es que la tecnología JSP/Servlets proporciona de un mayor apoyo a la orientación a objetos.

5.3.9 Apoyo al desarrollo basado en componentes.-

Sin tener la misma resonancia, el desarrollo basado en componentes es un aspecto que en nuestra opinión representa beneficios tan importantes como los que se obtienen de la orientación a objetos. Gracias al desarrollo basado en componentes se consigue hacer realidad una de las más preciadas metas de la ingeniería del software: El reuso. De hecho podemos decir que el desarrollo basado en componentes es, en cierto sentido, producto de la evolución de una metodología añeja del desarrollo de software: La programación modular.

A través del desarrollo basado en componentes es posible alcanzar otro de los objetivos de la ingeniería del software “Cómpralo en vez de construirlo”. Que no busca otra cosa sino hacer que en el desarrollo de software se impulse la utilización de componentes desarrollados por terceros que sean expertos en ese campo. Un grave problema de la industria hoy en día es que no se aprovecha lo suficiente la experiencia de compañías especializadas en desarrollar componentes para contrarrestar los cada vez más insuficientes tiempos de desarrollo a los que los proyectos están constreñidos. Por supuesto ha habido avances (sobre todo en los últimos años), pero estamos lejos de la situación ideal; los recursos en un proyecto son desperdiciados inútilmente en construir elementos de software que muy bien podrían sustituirse por un componente desarrollado por terceros.

Ambas tecnologías apoyan vigorosamente el desarrollo basado en componentes. La tecnología JSP/Servlets a través de los Beans y los EJB. Pese a su relativa juventud, el esquema de componentes de la tecnología Java innegablemente ha tenido un considerable éxito tanto para favorecer el reuso como para impulsar la utilización de componentes de terceros. Por su parte la tecnología ASP/COM+ a través del modelo de componentes COM+ el cual es la evolución de tecnologías como OLE[80] y MTS[81] las cuales han madurado por varios años hasta llegar al aventajado punto en el que actualmente se encuentra.

Sin llegar a consolidar al desarrollo basado en componentes al punto ideal donde se podrían cosechar muchos más frutos que los que actualmente son alcanzados, las dos tecnologías decididamente han tomado el camino de respaldar vigorosamente dicha metodología la cual definitivamente se ha afianzado para constituirse como un elemento fundamental de ambas.

Si bien ambas tecnologías presentan diferencias en cuanto a la penetración que la metodología de desarrollo basado en componentes ha logrado, y además de que difieren en cuanto a la forma en que se integra dicha metodología en cada tecnología. Sin embargo, en nuestra opinión el apoyo de ambas tecnologías a la utilización de componentes es equiparable y las diferencias entre ambas no son significativas como para determinar algún predominio de cualquiera de ellas. Por lo tanto, en nuestra opinión el apoyo al desarrollo basado en componentes no representa un factor para elegir o discriminar alguna de las tecnologías.

5.3.10 Apoyo a otros elementos de la ingeniería del software.-

Adicionalmente a la orientación a objetos y al desarrollo basado en componentes, la ingeniería del software integra a otras metodologías y corrientes adicionales cuya importancia no podemos ignorar. Pese a lo que algunos ideólogos de la orientación a objetos pudieran afirmar, ésta, junto con el desarrollo basado en componentes, no constituyen la panacea del desarrollo del software y no son suficientes para garantizar el éxito de un proyecto. Es por eso que muchos teóricos de la

[80] Es una tecnología desarrollada por Microsoft inicialmente para permitir el uso de documentos que a su vez integrarán elementos y servicios de diversas fuentes (componentes). Esta tecnología evolucionó hasta para ser utilizada en un modelo completo COM (y su versión distribuida DCOM). El día de hoy a esta tecnología se le conoce como ActiveX y también forma parte de COM+.

[81] MTS significa servidor de transacciones de Microsoft y es un producto desarrollado para correr sobre Windows NT y permitir (entre otros servicios) el manejo distribuido de transacciones entre componentes COM. La funcionalidad de MTS fue integrada como parte fundamental del sistema operativo Windows 2000 (dejando de ser un producto separado) en el modelo de componentes COM+.

orientación a objetos en realidad describen metodologías que van más allá del simple análisis y diseño orientado a objetos o de la programación orientada a objetos[82].

La ingeniería del software se puede definir como la aplicación de técnicas, metodologías y herramientas con el propósito de producir software de calidad que satisfaga las necesidades de los usuarios sin salirse del calendario o presupuesto. Esto es, aplicar la "ingeniería" en la producción, mantenimiento y administración del software.

En términos reales, una buena forma de medir el apoyo que una herramienta da a la ingeniería del software es determinar la cantidad y calidad de herramientas CASE[83] con que cuenta. Ahora bien dado el innegable éxito de Java y Windows como tecnologías de desarrollo, la oferta de herramientas CASE es significativa en ambas tecnologías. Por supuesto en la tecnología JSP/Servlets las herramientas CASE tienen que ver primordialmente con el carácter orientado a objetos que lógicamente tiene la tecnología. Mientras que con la tecnología ASP/COM+ no es tanta la dependencia (aunque sigue siendo importante) de las herramientas CASE con las metodologías de orientación a objetos.

Otro aspecto que influye en el factor del apoyo a la ingeniería del software refleja la aceptación de la propia industria por la tecnología. A medida que el interés por la tecnología crece, esto significa que más demanda habrá por guías para cumplir con el desarrollo de software de calidad. Entre estas guías podemos contar con los patrones de los que incluso ya se hizo uso durante el capítulo de diseño. Ahora bien, a diferencia de las herramientas CASE, estos elementos de la ingeniería del software no están tan ligados a las características de una tecnología determinada[84].

Algunos podrían argumentar que alguna tecnología apoya más a alguna de las finalidades de la ingeniería del software que la otra. Por ejemplo, el reuso aparentemente es favorecido en la tecnología JSP/Servlets dado el carácter orientado a objetos de la misma. Sin

[82] Por ejemplo la metodología más famosa, El RUP (proceso unificado de desarrollo, de los creadores del UML) además de incluir fundamentos de la orientación a objetos incluye también otros elementos adicionales que la complementan.

[83] CASE significa (por sus siglas en ingles) ingeniería del software asistida por computadora. Hay herramientas CASE que cubren prácticamente todo el ciclo de análisis, diseño y desarrollo; incluyendo también herramientas de documentación, implantación, control del proyecto, medición y un sinnúmero de otras funciones.

[84] Podemos decir que los patrones de diseño o las metodologías de la ingeniería del software no están tan ligados a una plataforma. Esto por que uno de sus propósitos es comúnmente el de mantener su validez en entornos distintos. Mientras más desacoplada este una herramienta de este tipo a cualquier plataforma, mayor será su aplicabilidad y por lo tanto será más valiosa.

embargo, en nuestra opinión el reuso depende más de la convicción con que se diseñen y desarrollen los componentes de software teniendo presente desde un principio el probable reuso. En este aspecto no consideramos que sean significativas las ventajas o desventajas de una tecnología. Otro ejemplo es el desarrollo rápido de aplicaciones (RAD) el cual aparentemente es mejor logrado en ASP/COM+[85]; sin embargo, tampoco existe una diferencia determinante en este factor.

La tecnología Java es quizá ligeramente superior en el apoyo a la ingeniería del software gracias a la orientación a objetos. Adicionalmente a la orientación a objetos en nuestra opinión no existe una diferencia significativa en el apoyo a la ingeniería del software entre ambas tecnologías.

5.3.11 Uniformidad y robustez de las plataformas subyacentes.-

Las dos tecnologías estudiadas en este trabajo de tesis: JSP/Servlets y ASP/COM+ a su vez forman parte de plataformas más completas que abarcan el desarrollo de aplicaciones de alto desempeño. Estas plataformas subyacentes incluyen una serie de servicios, herramientas y tecnologías adicionales dedicadas a apoyar a los componentes y subsistemas de cada una de las capas de una aplicación. Es importante tanto la calidad y cantidad (robustez) de estos servicios disponibles, como que estos mantengan una relativa uniformidad entre capas. La plataforma Windows DNA [86] presta estos servicios a aplicaciones ASP/COM+ mientras que la plataforma J2EE lo hace a las aplicaciones JSP/Servlets.

5.3.11.1 Windows DNA.-

La plataforma Windows DNA provee de una arquitectura para construir aplicaciones distribuidas utilizando un conjunto de servicios y herramientas de Microsoft Windows los cuales promulgan por un desarrollo basado en capas (específicamente en tres capas: capa de presentación, capa de aplicación y capa de persistencia). A continuación describiremos a algunos de las tecnologías, servicios y herramientas fundamentales de la plataforma Windows DNA:

[85] Aunque las cualidades RAD de la plataforma se muestran principalmente en aplicaciones cliente/servidor, no tanto en una aplicación ASP propiamente dicha.
[86] También conocida como Microsoft DNA o también Windows DNA 2000.

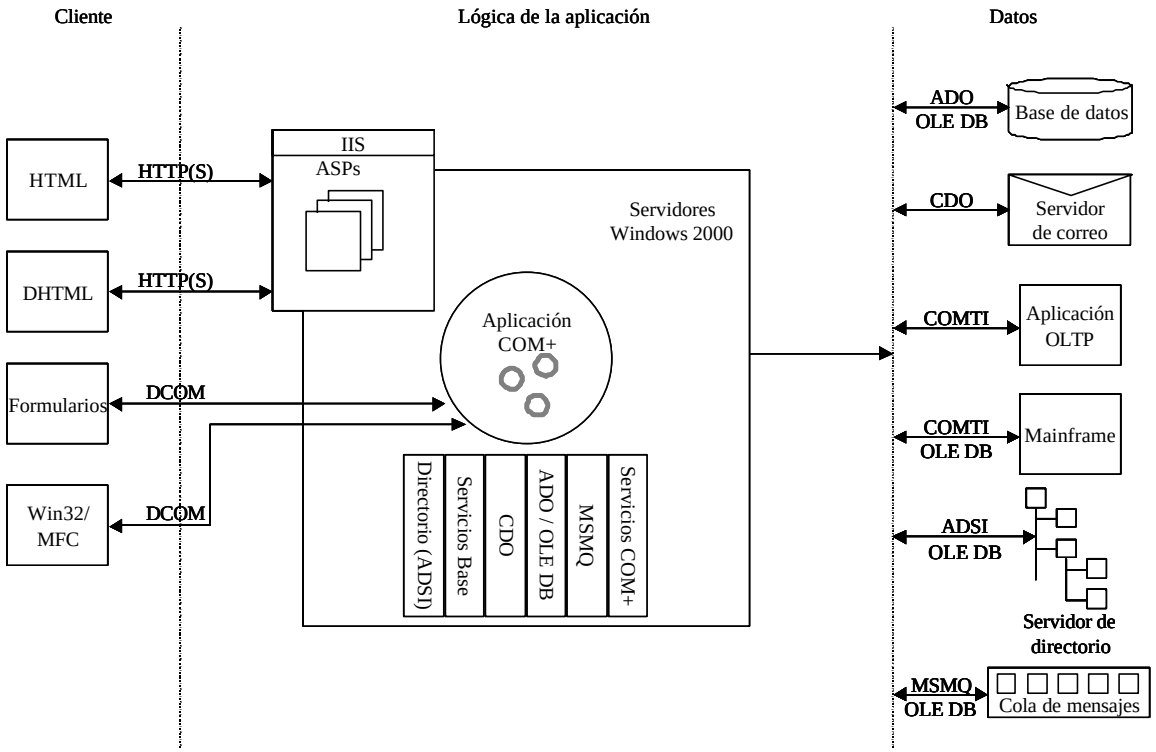
Para el desarrollo de la capa de presentación, Windows DNA provee de una serie de servicios que permiten la construcción de clientes delgados[87] hasta clientes de rica interfaz[88]. Para ello provee al browser Internet Explorer que además de permitir la presentación de contenido HTML, también permite la presentación de páginas DHTML, además permite la inclusión de funcionalidad a través de lenguajes de guión que se ejecutan del lado del cliente, finalmente permite también la inclusión de controles ActiveX[89] en páginas DHTML.

Otro elemento fundamental de la plataforma Windows DNA relativo a la capa de presentación lo constituye el API Win32, que por si mismo constituye la tecnología de construcción de aplicaciones gráficas más exitosa de la historia. Por medio de Win32 es posible construir aplicaciones con una gran riqueza en la interfaz con el usuario y hace uso de COM y los controles ActiveX para constituirse como un entorno que favorece ampliamente el desarrollo y utilización de componentes y por lo tanto fomenta el reuso.

[87] Los clientes delgados (thin clients) presentan el menor acoplamiento ante una plataforma en particular, el ejemplo más representativo de este tipo de clientes sería un cliente basado en HTML exclusivamente. La desventaja de estos clientes es que para no incorporar características dependientes de una plataforma, estos son presentan normalmente una interfaz con el usuario relativamente rudimentaria.

[88] Los clientes de rica interfaz (rich clients) aprovechan todos los recursos disponibles por una plataforma para presentar una interfaz sumamente elaborada con el usuario (en donde la interactividad es un factor clave). La desventaja de estos clientes es que son por definición, dependientes de una plataforma en particular y normalmente son, además, poco portables.

[89] Los controles ActiveX constituyen una pieza fundamental del modelo COM. Los mismos tienen que ver con el soporte a la incrustación de componentes visuales en contenedores permitiendo el manejo de la funcionalidad del control ActiveX tanto de manera programática como durante el diseño del cliente contenedor.



La capa de aplicación incluye tanto a servicios que tienen que ver con la construcción de la capa de dominio (que asegura las reglas de negocio de la aplicación); además de una serie de servicios que facilitan la integración de otro tipo de componentes que tienen que ver más con la capa de control de nuestro diseño. La capa de aplicación de la plataforma Windows DNA incluye a su servidor web IIS del cual ASP es un elemento fundamental. Pese a que IIS y ASP tienen que ver lógicamente con la capa de presentación (en especial para clientes delgados), su función es específicamente de servidor o servicios que se dan a la capa de presentación. Como fuimos testigos durante el desarrollo de los foros de discusión ASP es un entorno de ejecución de guiones del lado del servidor para construir páginas HTML dinámicas.

Otro elemento fundamental de la capa de aplicación lo constituye COM+, adicionalmente a la concepción de COM como un modelo de desarrollo de componentes que abarca a toda la plataforma Windows DNA. COM+ incluye una serie de servicios y características especificas de la capa de aplicación (algunos de los cuales previamente formaban parte de MTS). Algunos de estos servicios son:

- Administración de instancias de componentes, que permite el mejoramiento del desempeño al permitir el reuso de los objetos

instanciados. Dependiendo el esquema de hilado de un componente[90] la utilización del mismo puede ser más eficiente por medio de un cache o por medio de la reutilización de dicha instancia por varios procesos[91]. A esta funcionalidad normalmente se le conoce como ORB[92].

- Control transaccional distribuido por medio del cual muchos componentes pueden participar en una transacción, permitiendo incluso la integración con otros entornos de manejo transaccional ajenos a COM+.
- Administración de la seguridad de los componentes que permite el empleo de roles e inclusive la limitación del acceso a nivel de métodos de componentes.
- Compartimiento de recursos entre componentes[93] lo que permite que varios componentes aprovechen y compartan recursos (sobretodo aquellos recursos que son limitados) como por ejemplo conexiones a bases de datos.
- Componentes de invocación en cola, los cuales permiten la utilización asíncrona de componentes cuando algún recurso no está disponible.
- Notificación de eventos, que es un esquema en el que múltiples componentes se suscriben a la notificación de un evento.
- Balance de carga, por medio del cual varios servidores pueden distribuirse la carga de servir a varios clientes para los cuales es transparente esta característica.
- Administración centralizada, Al utilizar el explorador de servicios de componentes (que forma parte de Windows 2000), se tiene una sola herramienta desde la cual es posible configurar el manejo de los componentes que corren bajo la tutela de COM+.

Otro componente fundamental de la capa de aplicación de Windows DNA lo constituye MSMQ[94], el cual permite que varios componentes colaboren de manera asíncrona y distribuida. MSMQ provee de un esquema confiable para la distribución de mensajes sobre redes que no necesariamente tengan un 100% de disponibilidad. Adicionalmente MSMQ permite la integración con otros productos de este tipo por ejemplo MQSeries de IBM.

[90] A estos componentes se les conoce como “Ágiles” pues tienen esquemas de hilado que no están restringidos a un apartamento de hilos de ejecución específico.
[91] A estas dos funcionalidades se les conoce en ingles como object caching y como object pooling.
[92] ORB significa un manejador de peticiones de objetos o instancias (Object Request Broker).
[93] A lo que se le denomina en ingles como “resource pooling”.
[94] Servidor de colas de mensajes de Microsoft.

Los servicios para la capa de persistencia se engloban por la estrategia de UDA (acceso a datos universal) de Microsoft. UDA consiste de dos tecnologías fundamentales[95]: OLE DB y ADO.

Por medio de OLE DB, Los sistemas proveedores de persistencia (a los que se conoce como fuentes de datos[96]) pueden exponer sus servicios de una manera efectiva pero simultáneamente flexible al no restringir las características o naturaleza de las fuentes de datos a un patrón en específico. OLE DB consiste fundamentalmente en una serie de definiciones de interfaces COM que encapsulan los servicios que las fuentes de datos son capaces de proveer. Podemos decir que OLE DB se encarga de ofrecer un esquema de acceso uniforme a un conjunto de fuentes de datos, este esquema es de bajo nivel pero de alto desempeño.

El otro elemento fundamental de UDA lo constituye ADO (ActiveX Data Objects), que representa un API basado en objetos y componentes para el desarrollo de soluciones que se conecten a fuentes de datos a través de OLE DB. A través de ADO se provee de un conjunto jerárquico de objetos que modelan los elementos principales de una fuente de datos. Por medio de ADO se pretende uniformar el código inherente al desarrollo de componentes de la capa de persistencia manteniendo un reducido acoplamiento a una fuente de datos en particular.

Adicionalmente de los servicios de estas tres capas, Windows DNA incluye una serie de servicios de sistema los cuales son aprovechados por cualquier capa de un sistema. Uno de estos servicios es la seguridad que es provista por cada una de las versiones del sistema operativo Windows, otro servicio sería el de directorios que es provisto por la tecnología Active Directory, también tenemos a los servicios de red y de acceso a recursos compartidos, etc. Todos estos servicios son inherentes al sistema operativo Windows, y por ende, disponibles para cualquier aplicación sobre la plataforma Windows DNA.

La plataforma Windows DNA incluye también un conjunto de herramientas, la más importante sería sin duda Visual Studio de la que ya hablamos con anterioridad. También se ofrecen herramientas administrativas como por ejemplo la consola administrativa de Microsoft

[95] Aunque ODBC es un estándar promovido por Microsoft para el acceso a bases de datos. Microsoft ahora promueve la utilización de tecnologías más modernas otorgándole un estatus de “obsoleto” a ODBC. Sin embargo, en la actualidad es usado por un gran número de sistemas independientemente de sus limitaciones.

[96] El ejemplo más típico de una fuente de datos es un manejador de bases de datos relacional, pero las fuentes de datos que exponen una interfaz OLE-DB agrupan a una amplia variedad de servicios y sistemas capaces de ofrecer alguna funcionalidad de persistencia de información.

(MMC) por medio de la cual es posible configurar y administrar muchos de los componentes y servicios de la plataforma Windows DNA. De hecho debido a la gran integración entre esta plataforma y el sistema operativo Windows podemos decir que muchas herramientas del sistema operativo son por lo tanto herramientas de la plataforma.

Alrededor de los servicios y herramientas fundamentales de Windows DNA, Microsoft provee de una serie de aplicaciones cuya integración con Windows DNA es profunda y por lo tanto son posicionadas como la alternativa lógica al existir la necesidad de contar con la funcionalidad que dichas aplicaciones poseen. Algunas de estas aplicaciones son:

- MS SQL Server.- El cual es un servidor de bases de datos con una amplia integración al resto de servicios y productos de Windows DNA.
- MS Exchange Server.- El cual provee de los servicios de correo y colaboración.
- MS Commerce Server.- El cual proporciona herramientas para facilitar la elaboración e implantación de componentes de la capa intermedia.
- MS Application Center.- El cual facilita la escalabilidad al permitir integrar y administrar grandes clusters de servidores (granjas de servidores).
- MS BizTalk Server.- Provee de herramientas y servicios para facilitar el intercambio de datos e información a través de XML.

A menudo es difícil determinar cual es el papel que juega algún producto de Microsoft en la arquitectura Windows DNA pues se deben separar las especificaciones formales de la plataforma de los documentos mercadotécnicos que promueven cada producto de la familia Microsoft, la cual es una tarea bastante compleja. Hay que recordar que a final de cuentas Windows DNA nació como una estrategia comercial para promocionar diversos productos de Microsoft como una arquitectura. Otro problema es que muchos productos cambian continuamente de nombre o dejan de ser productos separados para terminar siendo componentes del sistema operativo.

5.3.11.2 Sun J2EE.-

La tecnología Sun J2EE, constituye la visión de Sun y un importante número de otras empresas, en la que a través de Java como factor común se promueve un robusto esquema de desarrollo de soluciones empresariales de alto desempeño basado en componentes y en la

separación de capas. La plataforma J2EE está compuesta por dos partes:

- Una infraestructura de ejecución de aplicaciones.
- Un conjunto de API's que extienden el lenguaje Java.

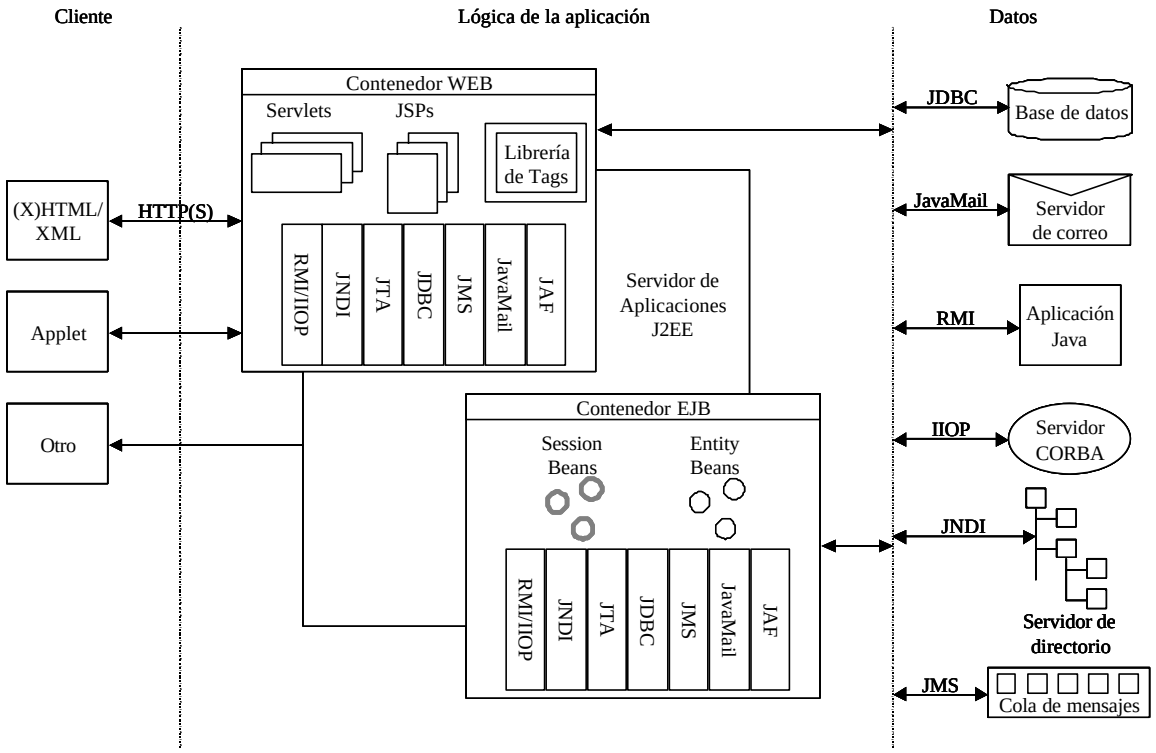
La plataforma J2EE integra este conjunto de API's[97] al elemento abstracto que constituye la infraestructura de ejecución. Un elemento clave es el hecho de que Sun no determina como deberá ser construida una infraestructura de ejecución J2EE; en cambio, señala una serie de puntos que cualquier infraestructura de ejecución J2EE debe cubrir[98]. Para ello se introduce el concepto de "contenedor" que representa un sistema de interacción entre una aplicación y la infraestructura encargada de hospedarla. Bajo este esquema al desarrollar una solución J2EE el equipo de desarrollo debe cubrir los siguientes compromisos:

- Proveer de los componentes de la aplicación, llámense servlets, JSP's, EJB's, etc.
- Proveer de los descriptores de despliegue[99], que no son otra cosa que archivos XML que describen al componente ante el contenedor.

Por su parte el contenedor deberá incluir los siguientes elementos o servicios:

- Contrato del contenedor, que no es otra cosa sino un API que el contenedor debe cumplir pues será utilizado por los componentes de la aplicación.
- Servicios del contenedor, que son API's que comúnmente todas las aplicaciones requerirán.
- Servicios declarativos, que son una serie de servicios otorgados de acuerdo a los descriptores de despliegue de cada componente de aplicación.
- Servicios adicionales del contenedor, los cuales son servicios adicionales como reutilización y compartimiento de recursos, recolección de basura[100], etc.

[97] Algunos de los cuales existían antes de que J2EE surgiera como una plataforma propiamente dicha.
[98] Por esto se dice que la plataforma J2EE es básicamente una especificación (A diferencia de Windows DNA que es la agrupación de una serie de productos y tecnologías).
[99] A lo que se le denomina en ingles como "deployment descriptors".
[100] La recolección de basura es un servicio que normalmente corre en segundo plano el cual libera recursos (normalmente memoria) no usados o cuya referencia terminó.



Plataforma J2EE

Los API's que forman parte de J2EE representan la única forma válida de acceder a recursos y servicios empresariales por los componentes de una aplicación. El contenedor es responsable de que cualquier componente de una aplicación realmente acceda a estos servicios[101]. Al limitar la forma de acceso a servicios empresariales con un API que representa la forma estándar de acceso, se evita que los componentes de una aplicación lo hagan usando esquemas nativos de un entorno específico, garantizándose de esta manera la portabilidad del componente[102]. Ésta es una lista de los API's que incluye la plataforma J2EE en su presente versión:

- JDBC (Conectividad de Java a bases de datos), a través de este API los componentes de una aplicación J2EE se conectan a una base de datos.
- RMI-IIOP (Protocolo Entre-ORB's para la invocación remota de métodos a través de la Internet), que no es otra cosa que un

[101] Los contenedores que forman parte de un servidor de aplicaciones J2EE no necesitan implementar tales servicios empresariales, simplemente deben ser capaces de proveer acceso a los mismos rigiéndose por la metodología que impone el API.

[102] Sin embargo, como ya mencionamos, en la práctica los servidores de aplicaciones presentan características avanzadas a las que al hacer uso de las mismas se acopla en cierta medida a una aplicación con un servidor de aplicaciones J2EE en particular.

puente entre el API RMI de Java para conectar a componentes CORBA.

- EJB (Componentes Java empresariales), este API define a su vez toda una arquitectura para el desarrollo de componentes empresariales siguiendo principios como: múltiples capas, transacciones, seguridad, escalabilidad, concurrencia, persistencia, etc.
- Java servlets, que define un modelo para el desarrollo de aplicaciones web orientado a objetos.
- JSP (Páginas Java del servidor), que presenta un esquema simplificado de desarrollo de aplicaciones web al hacer énfasis en el diseño de páginas sobre la codificación que implica el desarrollar directamente un servlet.
- JMS (Servicio de mensajes de Java), el cual presenta la alternativa de Java para el manejo de colas de mensajes, suscripción, interacción asíncrona, etc.
- JNDI (Interfaz de nombres y directorios de Java), por medio del cual se provee un acceso estandarizado a los servicios de nombrado y de directorios.
- JTA (API de transacciones de Java), por medio del cual los componentes de una aplicación implementan esquemas transaccionales distribuidos.
- JavaMail, el cual es un API para estandarizar el desarrollo de aplicaciones con manejo de correo electrónico independientemente del protocolo o entorno.

Se puede interpretar como que hay un contrato entre el componente y el contenedor en cuanto a que el componente se compromete a acceder a los recursos empresariales a través de los API's[103] mientras que el contenedor se compromete a otorgar el acceso a dichos recursos. Adicionalmente a los recursos accedidos por medio de los API's, el contenedor provee de otros servicios adicionales a los que se les conoce como servicios declarativos.

Los servicios declarativos son otorgados a los componentes no por medio de la programación interna de los mismos sino por un elemento adicional que señala cuales de estos servicios el componente requiere. A estos elementos se les conoce como descriptores de despliegue. Los descriptores de despliegue indican si algún componente o grupo de componentes requieren de algún servicio en particular (como serían las transacciones, persistencia automática, etc.) dependiendo de su tipo. Esto resulta en una mayor flexibilidad al momento de desarrollo de los

[103] Normalmente esto se hace extendiendo algunas de las interfaces definidas por el API.

componentes al no tener que establecer durante la codificación del mismo si este deberá o no acceder a dichos servicios.

Adicionalmente a los servicios codificados en cada componente que son accedidos por medio de los API's y a los servicios declarativos que son accedidos por medio de los descriptores de despliegue; la plataforma J2EE también contempla una serie de servicios adicionales que el servidor de aplicaciones provee en tiempo de ejecución. Los contenedores proveen de estos servicios contextuales al componente que albergan y su funcionalidad principal es la optimización de recursos para mejorar el desempeño, la seguridad, etc.

5.3.11.3 Conclusión en la comparación de las plataformas subyacentes.

En nuestra opinión ambas plataformas son bastante robustas, y permiten el desarrollo e implementación de sólidas aplicaciones de nivel empresarial, con esto no queremos decir que las características de las dos plataformas son equivalentes o que existe una similitud de cualidades. Si la comparación de JSP/Servlets y ASP/COM+ es en nuestra opinión bastante compleja, el comparar J2EE con Windows DNA lo es mucho más. Más que esto el resultado de cualquier comparación siempre tendrá su dosis de controversia.

Es por eso que no es nuestra intención tratar de hacer una aserción definitiva acerca de cual plataforma ofrece más o mejores servicios adicionales al momento de desarrollar una aplicación del tipo Internet/Intranet. Aclarado esto no podemos, sin embargo, dejar de mencionar que ambas plataformas presentan aspectos que valen la pena ser abordados, así como también cualidades que resaltan a una sobre la otra.

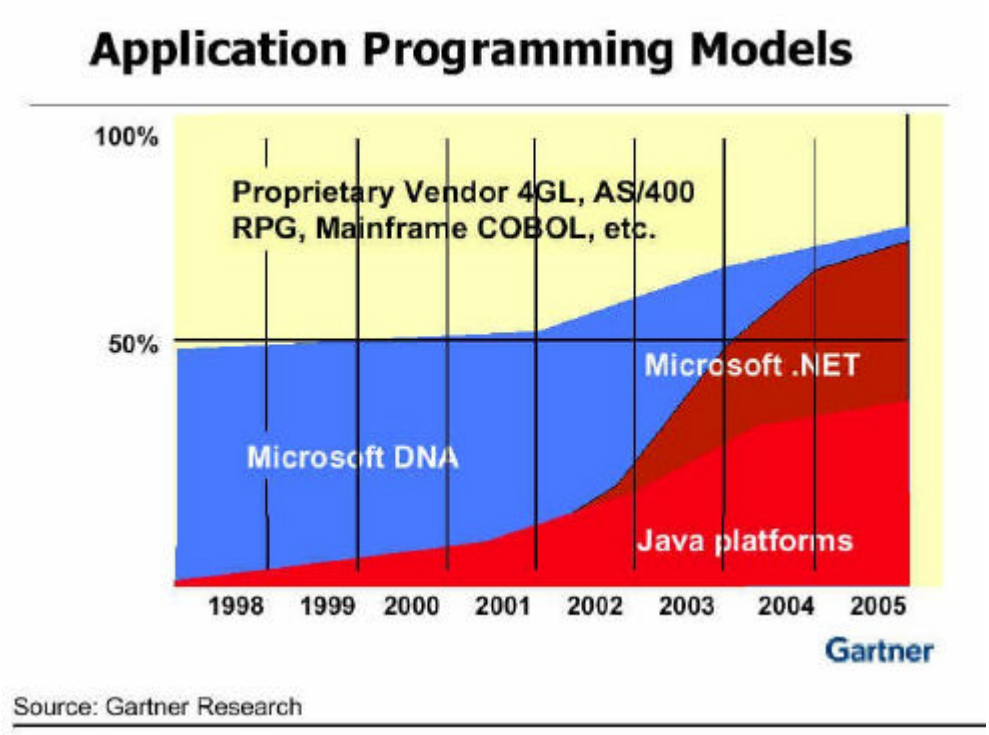
Un aspecto sobresaliente de la plataforma J2EE lo constituyen los descriptores de despliegue, sin duda estos representan una alternativa efectiva ante la normalmente ardua tarea de implantar una aplicación. En nuestra opinión en este sentido la plataforma J2EE muestra una ventaja ante el modo en que se implantan las aplicaciones sobre la plataforma Windows DNA. Por supuesto en la práctica las aplicaciones complejas para su instalación requieren de bastante más que definir un simple descriptor de despliegue pero definitivamente ellos constituyen un avance importante y en la dirección adecuada hacia la simplificación de dicho proceso.

Otro aspecto importante de resaltar es el hecho de que al mantener un claro dominio en la capa de desarrollo de aplicaciones cliente, la plataforma Windows DNA tiene una ventaja sobre J2EE; pues constituye esto un incentivo para que muchas veces se decida utilizar Windows DNA para todas las capas de una aplicación empresarial una vez que se opta inicialmente en la utilización de tecnologías Windows DNA para el desarrollo de la capa de presentación.

Al mismo tiempo, la relación indivisible entre la plataforma Windows DNA con el sistema operativo Windows es también una desventaja, pues esto significa que dicho sistema operativo termina siendo un requisito forzoso. Esta nula posibilidad de elección lógicamente no es algo que favorezca a la tecnología. Los limites y desventajas que el sistema operativo Windows posee son traspasados automáticamente a Windows DNA. En cambio la situación de la plataforma J2EE es diametralmente opuesta al mantener una mínima dependencia con el sistema operativo anfitrión.

La plataforma Windows DNA pareciera estar orientada por el hecho de que la tecnología como tal, nació después de que ya se tenían en el mercado algunos servicios y productos que posteriormente formaron parte de la misma. El elemento que aglutina todo en Windows DNA es sin lugar a dudas COM, pero la forma de acceder a algunos de los servicios de Windows DNA pese a ser siempre por medio de COM, no es (en nuestro punto de vista) completamente uniforme; con esto no queremos decir que los servicios son presentados como entidades individuales y separadas, pero en nuestra opinión los elementos de la plataforma Windows DNA no presentan la misma uniformidad o estandarización que se observan en la tecnología J2EE.

Un importante aspecto que debe ser considerado al evaluar la posibilidad de hacer un desarrollo sobre la plataforma Windows DNA es el hecho de que con el lanzamiento de Microsoft .NET es esta última plataforma la que ahora recibe todo el impulso y promoción por parte de Microsoft. Por ejemplo en el siguiente diagrama se muestra la cuota de mercado que se estima abarcaran estas plataformas:



Como podemos ver, se estima que Microsoft .NET remplazará a la plataforma Windows DNA en un periodo relativamente corto, sin embargo hay que tener en cuenta que en México existe un atraso importante en la industria en cuanto a la adopción de estas tecnologías sobre todo si hablamos de la pequeña o mediana industria. Por lo tanto es de esperar que en nuestro país se tarde más la adopción de .NET y por lo tanto la plataforma Windows DNA coexistirá más tiempo con nosotros.

Mientras que la plataforma J2EE es promovida por un número importante de empresas lideres en el desarrollo de soluciones empresariales. En el caso de Windows DNA es Microsoft el que se encarga de hacer lo mismo[104]. El caso es que independientemente de las características intrínsecas de ambas tecnologías, esto significa una importante desventaja de Windows DNA. Ahondaremos más sobre esto en las conclusiones finales de esta tesis.

[104] Esto no quiere decir que además de Microsoft no haya empresas importantes apoyando las tecnologías de Windows DNA, pero estas no son tan importantes (en el ámbito de sistemas de nivel empresarial) como las que apoyan a J2EE y, además, no lo hacen con la misma convicción. De esta situación muchos consideran que Microsoft es responsable en buena medida.

Nuestra conclusión acerca de las características de las tecnologías Windows DNA y J2EE es que ambas proveen de los elementos necesarios para la construcción de robustas aplicaciones de varias capas. Sin embargo, una fuerte desventaja la constituye el hecho de que Microsoft ahora promueve a .NET como su plataforma de desarrollo distribuido. También pensamos que J2EE está en una posición ventajosa pues su implementación cubre un amplio espectro de entornos operativos a diferencia de Windows DNA que está restringida al sistema operativo Windows; Cuando es en este último donde sin lugar a dudas será implantada la solución, la alternativa Windows DNA es definitivamente una opción bastante válida, en cambio cuando el entorno operativo no es Windows o puede variar en el futuro, Windows DNA deja de ser una opción atractiva.

5.4 Conclusiones.-

El desarrollo de aplicaciones Internet/Intranet a través de las tecnologías ASP/COM+ y JSP/Servlets representa sin duda una eficaz alternativa para generación de contenido HTML dinámico. El concepto de integrar simultáneamente en una página secciones de HTML junto con secciones de código permite desarrollar rápidamente contenido lo cual es una gran ayuda frente los ajustados tiempos de desarrollo a los que son constreñidos la mayoría de los proyectos en la actualidad.

La flexibilidad que ASP y JSP otorgan tiene un precio, el cual radica en los riesgos inherentes al aglutinar elementos propios de la interacción con el usuario (en la forma de contenido HTML) junto con el código necesario para producir dicho contenido. Podemos decir que la principal característica de las tecnologías ASP y JSP está al mismo tiempo opuesta al principio de separación entre la lógica del problema y la presentación[105]. A nuestro juicio es preferible este aspecto negativo del esquema utilizado en vez de la rigidez de otras alternativas tales como los guiones CGI.

En nuestra opinión las soluciones desarrolladas sobre la tecnología JSP/Servlets tienen mayores posibilidades de ser llevadas a cabo de una manera elegante con respecto a los patrones ampliamente aceptados por la ingeniería del software (sobretodo la programación orientada a objetos). Esto gracias a las características propias de Java como

[105] El código incrustado en una página ASP no esta necesariamente relacionado con la lógica del problema (lo ideal será que constituya parte de la lógica de presentación solamente), pero la posibilidad de incrustar lógica de negocio directamente en la página ASP/JSP esta ahí. Algunos autores proponen incluso esquemas en los que no es posible incrustar código en las páginas (las cuales son en realidad plantillas) sacrificando flexibilidad pero asegurando una estricta separación entre lógica y presentación.

lenguaje y de la plataforma J2EE en general. Ninguna de las dos alternativas, sin embargo, garantiza esta meta, aún se requerirá un decidido compromiso por parte del equipo de desarrollo (y de los equipos de análisis y diseño también, por supuesto) para producir una aplicación que se rija por estos lineamientos. Pero resulta una ventaja que en la tecnología elegida estos principios sean un elemento fundamental de la misma.

Durante el desarrollo del sistema de foros de discusión, algunos lineamientos de diseño tuvieron que ser reconsiderados en la versión ASP/COM+ principalmente por las reducidas características de los lenguajes de guión que ASP otorga. VBScript y JavaScript resultan ser muy buenas herramientas para el desarrollo de soluciones relativamente simples. Sin embargo, es innegable que para algunos aspectos más complejos estos lenguajes terminan quedándose cortos y el problema es que en ASP estamos forzados a usar estos lenguajes al menos para la parte de la lógica de presentación. Resulta en cambio difícil encontrar un escenario en que el lenguaje Java no sea suficiente para desarrollar la lógica de presentación en una página JSP.

Como describimos anteriormente, existen factores en que una de las tecnologías resalta sobre la otra y otros donde se da lo opuesto. En nuestra opinión la tecnología JSP/Servlets aventaja a la tecnología ASP/COM+ gracias a que presenta una mayor uniformidad e integración con la tecnología subyacente (J2EE), y también gracias a la notoria portabilidad de esta tecnología. Pero con este trabajo de tesis, además de la experiencia profesional que personalmente hemos acumulado en otros proyectos, nos hace afirmar que las ventajas o desventajas de ambas tecnologías no son lo suficientemente dispares como para descartar a cualquiera de las mismas.

En nuestra opinión, dependerá en mayor medida de los objetivos y requerimientos de un proyecto dado, para la elección de alguna de estas dos tecnologías. Esto es que habrá proyectos en los que determinados factores tendrán más peso que otros, entonces la decisión más acertada será aquélla que favorezca a la tecnología o plataforma que más destaca en aquellos factores que son importantes para dicho proyecto.

Adicionalmente existen otros factores que deberían ser tomados en cuenta, como, por ejemplo, los sistemas con que actualmente se cuenta. Sobretodo, si los mismos tendrán alguna interacción con el nuevo sistema, los conocimientos y habilidades del equipo de desarrollo respecto a las tecnologías en disputa, etc. En nuestra opinión gracias a lo parejo que ambas tecnologías se desempeñan, este tipo de factores

externos muchas veces tendrán mayor peso que los factores intrínsecos y a nuestro parecer es algo positivo que así sea.

Las características intrínsecas de ambas tecnologías y sus plataformas subyacentes son equiparables entre sí, si de ello dependiera únicamente para decidir sobre el usar alguna de las dos, se tendrían argumentos suficientes para sustentar dicha decisión cualquiera que ésta fuera. Sin embargo, existe un elemento circunstancial que en nuestra opinión disminuyen considerablemente los argumentos que se tuvieren a favor de la plataforma Windows DNA en general y de ASP/COM+ en particular.

Durante el desarrollo de esta tesis se ha ido consolidando .NET la cual es la nueva tecnología de desarrollo de Microsoft. En particular deberíamos mencionar a ASP.NET el cual reemplaza la funcionalidad antes proporcionada por ASP. Si bien ASP y ASP.NET tienen similitudes, la considerable cantidad de nuevas características hace impreciso el considerar a esta nueva tecnología como una versión mejorada de ASP.

La tecnología ASP.NET hace uso de la rica tecnología .NET y corrige muchas de las fallas que tenía ASP. Ahora están disponibles lenguajes muy completos (como C#) para desarrollar páginas con lo que se soluciona una de las principales objeciones que nosotros hicimos sobre ASP durante este trabajo de tesis. Por otro lado la tecnología .NET es mucho más uniforme que Windows DNA; y ahora sí la orientación a objetos es un aspecto fundamental de la tecnología. Particularmente resaltan de ASP.NET los llamados controles de servidor los que permiten simular al exitoso paradigma de desarrollo de aplicaciones en Windows; Los controles de servidor de ASP.NET ofrecen un modelo de programación orientado a objetos, basado en eventos, con persistencia automática del estado del control, permitiendo la asociación con fuentes de datos para el despliegue automático de los mismos; Entre otras características.

Ahora bien, aunque .NET es ya una realidad al momento de finalizar este trabajo de tesis, su adopción por el mercado es todavía una incógnita. Esto se acentúa aún más en el mercado mexicano donde ASP y JSP aún no se han consolidado (lamentablemente) y la industria apenas empieza a probar tecnologías de desarrollo posteriores al modelo Cliente/Servidor. Por lo tanto en muchos proyectos aún se contempla la utilización de ASP para el desarrollo de los mismos. La lentitud con que el mercado mexicano adopta estas tecnologías de desarrollo de aplicaciones sobre Internet/Intranet provoca que la comparación entre ASP/COM+ y JSP/Servlets todavía siga siendo válida por un tiempo. Sin

embargo, es nuestra opinión que dada una disyuntiva de ese tipo, también se considere la utilización de .NET y en especial de ASP.NET pues las considerables cualidades de esta nueva tecnología en cierto sentido contrarrestan la poca madurez de la misma.

El interés de este trabajo de tesis no esté en determinar una tecnología o plataforma que resulte ganadora en la comparación realizada, sino de auxiliar a aquellos que se encuentren en la disyuntiva de utilizar una o la otra. Para ello se intentaron resaltar algunas de las principales características que tanto la tecnología ASP/COM+ como la tecnología JSP/Servlets tienen. Es un hecho que existen aspectos en los que alguna de las tecnologías resalta sobre la otra por lo que una decisión inteligente al momento de decidirse sobre alguna de las mismas seria basarse en aquellas características que son cruciales para el proyecto en particular que se requiere desarrollar.

Por cuestiones que lamentablemente son más económicas o políticas que tecnológicas, la industria del desarrollo de software se ha polarizado siendo forzada a tomar partido por una u otra plataforma. Habrá quien acepte complaciente el hecho de tener que tomar un bando así como habrá quien rehúse tomar bando y prefiera mantener una posición intermedia. En cualquier caso ambas decisiones serán más sólidas si están sustentadas en un efectivo e imparcial análisis de las cualidades de estas plataformas. Este trabajo de tesis intenta ser una herramienta más en dicho sentido.

Definitivamente las notables ventajas que presentan los esquemas de aplicaciones distribuidas y con interfaz tipo Internet/Intranet son dignas de ser tomados en cuenta ante los también notables problemas asociados a los esquemas tradicionales (cliente/servidor, clientes pesados y nativos, etc.). Por diversas razones México muestra un atraso en la adopción de dichos esquemas por lo que es imperioso el disminuir esta brecha. Las ventajas están ahí solo falta el compromiso por tomar el camino que permita aprovecharlas.

APÉNDICES.

Apéndice A. Clase Usuario (versión Java).-

```
package com.netapplications.webforums_jsp.domain;

import java.util.Date;
import java.io.Serializable;
import com.netapplications.webforums_jsp.types.*;

public class Usuario extends Object implements Serializable {

  private int id_usuario;
  private String nombre_corto;
  private String nombre_completo;
  private String password;
  private String email;
  private String estatus;
  private Date fecha_registro;
  private Date fecha_ultimo_acceso;
  private String administrador;

  public Usuario() {
    id_usuario = 0;
    nombre_corto = "";
    nombre_completo = "";
    password = "";
    email = "";
    estatus = Estatus.ACTIVO;
    fecha_registro = new Date();
    fecha_ultimo_acceso = new Date();
    administrador = StrBoolean.FALSE;
  }
  public synchronized int getId_usuario() {
    return id_usuario;
  }
  public synchronized void setId_usuario(int newId_usuario) {
    id_usuario = newId_usuario;
  }
  public synchronized String getNombre_corto() {
    return nombre_corto;
  }
  public synchronized void setNombre_corto(String newNombre_corto) {
    nombre_corto = newNombre_corto;
  }
  public synchronized String getNombre_completo() {
    return nombre_completo;
  }
```

```
public synchronized void setNombre_completo(String newNombre_completo) {
  nombre_completo = newNombre_completo;
}
public synchronized String getPassword() {
  return password;
}
public synchronized void setPassword(String newPassword) {
  password = newPassword;
}
public synchronized String getEmail() {
  return email;
}
public synchronized String getEmail_nombreCorto() {
  if (email!=null
      && email.indexOf("@")>0
      && email.indexOf("@")<email.length()-1) {
    return "<a href=\"mailto:" + email + "\">" + nombre_corto + "</a>";
  } else {
    return nombre_corto;
  }
}
public synchronized void setEmail(String newEmail) {
  email = newEmail;
}
public synchronized String getEstatus() {
  return estatus;
}
public synchronized void setEstatus(String newEstatus) {
  if (newEstatus.toUpperCase().equals(Estatus.ACTIVO)) {
    estatus = Estatus.ACTIVO;
  } else {
    estatus = Estatus.INACTIVO;
  }
}
public synchronized Date getFecha_registro() {
  return fecha_registro;
}
public synchronized void setFecha_registro(Date newFecha_registro) {
  fecha_registro = newFecha_registro;
}
public synchronized Date getFecha_ultimo_acceso() {
  return fecha_ultimo_acceso;
}
public synchronized void setFecha_ultimo_acceso(Date newFecha_ultimo_acceso) {
  fecha_ultimo_acceso = newFecha_ultimo_acceso;
}
```

```
  public synchronized void setAdministrador(String newAdministrador) {
    if (newAdministrador.equalsIgnoreCase(StrBoolean.TRUE)) {
      administrador = StrBoolean.TRUE;
    } else {
      administrador = StrBoolean.FALSE;
    }
  }
  public synchronized String getAdministrador() {
    return administrador;
  }
}
```

Apéndice B. Clase Estatus (versión Java).-

```
package com.netapplications.webforums_jsp.types;

public class Estatus {
  public static final String ACTIVO = "A";
  public static final String INACTIVO = "I";

  public static String getDescription(String currentStatus) {
    if (currentStatus.equals(ACTIVO)) {
      return "Activo";
    } else if (currentStatus.equals(INACTIVO)) {
      return "Inactivo";
    } else {
      return "Desconocido";
    }
  }
}
```

Apéndice C. Clase UsuarioPersist (versión Java).-

```
package com.netapplications.webforums_jsp.persist;

import java.sql.*;
import java.util.*;
import com.netapplications.webforums_jsp.domain.Usuario;
import com.netapplications.webforums_jsp.util.Settings;

public class UsuarioPersist {
  private static UsuarioPersist instance;
  private Connection connection;
  private PreparedStatement getUsuarioStmt;
  private PreparedStatement putUsuarioStmt;
  private PreparedStatement remUsuarioStmt;
  private PreparedStatement updUsuarioStmt;
  private PreparedStatement getUsuarioLoginStmt;
  private PreparedStatement getAllReceiptsStmt;
  private UsuarioIDGenerator idgen;

  public static UsuarioPersist getInstance()
    throws PersistException {
    if (instance == null)
      instance = new UsuarioPersist();
    return instance;
  }
  private synchronized void parseStatements() throws SQLException {
    getUsuarioStmt = connection.prepareStatement(Settings.get("sql.UsuarioPersist.get"));
    getUsuarioLoginStmt = connection.prepareStatement(Settings.get("sql.UsuarioPersist.getLogin"));
    putUsuarioStmt = connection.prepareStatement(Settings.get("sql.UsuarioPersist.put"));
    remUsuarioStmt = connection.prepareStatement(Settings.get("sql.UsuarioPersist.rem"));
    updUsuarioStmt = connection.prepareStatement(Settings.get("sql.UsuarioPersist.upd"));
    getAllReceiptsStmt = connection.prepareStatement(Settings.get("sql.UsuarioPersist.getAllReceipts"));
  }
  private UsuarioPersist() throws PersistException {
    idgen = UsuarioIDGenerator.getInstance();
    try {
      connection = ConnectionPool.get();
      parseStatements();
      ConnectionPool.free(connection);
    }
    catch (SQLException se) {
      throw new PersistException("UsuarioPersist [UsuarioPersist()]: " + se.getMessage());
```

```
  }
}
private Usuario makeUsuario(ResultSet results)
  throws PersistException {
  try {
    Usuario Usuario = new Usuario();
    Usuario.setId_usuario(results.getInt("ID_USUARIO"));
    Usuario.setNombre_corto(results.getString("NOMBRE_CORTO"));
    Usuario.setNombre_completo(results.getString("NOMBRE_COMPLETO"));
    Usuario.setPassword(results.getString("PASSWORD"));
    Usuario.setEmail(results.getString("EMAIL"));
    Usuario.setEstatus(results.getString("ESTATUS"));
    Usuario.setFecha_registro((java.util.Date) results.getTimestamp("FECHA_REGISTRO"));
    Usuario.setFecha_ultimo_acceso((java.util.Date) results.getTimestamp("FECHA_ULTIMO_ACCESO"));
    Usuario.setAdministrador(results.getString("ADMINISTRADOR"));
    return Usuario;
  }
  catch (SQLException e) {
    throw new PersistException("UsuarioPersist [makeUsuario()]: " + e.getMessage());
  }
}
public synchronized Usuario[] getUsuariosToSendEmail(int id_mensaje)
  throws PersistException {
  Connection confrompool = null;
  try {
    confrompool = ConnectionPool.get();
    if (confrompool != connection) {
      synchronized (connection) {
        connection = confrompool;
      }
      parseStatements();
    }
    ResultSet results;
    Collection Usuarios = new ArrayList();
    synchronized(getAllReceiptsStmt) {
      getAllReceiptsStmt.clearParameters();
      getAllReceiptsStmt.setInt(1, id_mensaje);
      results = getAllReceiptsStmt.executeQuery();
    }
    while (results.next()) {
      Usuarios.add(makeUsuario(results));
    }
    return (Usuario[])Usuarios.toArray(new Usuario[0]);
  }
  catch (SQLException e) {
    throw new PersistException("UsuarioPersist (getUsuariosToSendEmail()): " + e.getMessage());
```

```
  }
  finally {
    if (confrompool!=null) {
      ConnectionPool.free(confrompool);
    }
  }
}
public Usuario getUsuario(int id_usuario)
  throws UnknownPersistException, PersistException {
  Connection confrompool = null;
  try {
    confrompool = ConnectionPool.get();
    if (confrompool != connection) {
      synchronized (connection) {
        connection = confrompool;
      }
      parseStatements();
    }
    ResultSet results;
    synchronized (getUsuarioStmt) {
      getUsuarioStmt.clearParameters();
      getUsuarioStmt.setInt(1, id_usuario);
      results = getUsuarioStmt.executeQuery();
    }
    if (results.next())
      return makeUsuario(results);
    else
      throw new UnknownPersistException("UsuarioPersist [getUsuario()]: " + "No se encontro el Usuario # " + id_usuario);
  }
  catch (SQLException e) {
    throw new PersistException("UsuarioPersist [getUsuario()]: " + e.getMessage());
  }
  finally {
    if (confrompool!=null) {
      ConnectionPool.free(confrompool);
    }
  }
}
public Usuario getUsuarioLogin(String nom_usuario, String PASSWORD)
  throws UnknownPersistException, PersistException {
  Connection confrompool = null;
  try {
    confrompool = ConnectionPool.get();
    if (confrompool != connection) {
      synchronized (connection) {
        connection = confrompool;
```

```
        }
        parseStatements();
      }
      ResultSet results;
      synchronized (getUsuarioStmt) {
        getUsuarioLoginStmt.clearParameters();
        getUsuarioLoginStmt.setString(1, nom_usuario);
        getUsuarioLoginStmt.setString(2, PASSWORD);
        results = getUsuarioLoginStmt.executeQuery();
      }
      if (results.next())
        return makeUsuario(results);
      else
        throw new UnknownPersistException("UsuarioPersist [getUsuarioLogin()]: " + "No se encontro el Usuario : " +
nom_usuario + " ó la contraseña es inválida");
    }
    catch (SQLException e) {
      throw new PersistException("UsuarioPersist [getUsuarioLogin()]: " + e.getMessage());
    }
    finally {
      if (confrompool!=null) {
        ConnectionPool.free(confrompool);
      }
    }
  }
  public void update(Usuario Usuario)
    throws UnknownPersistException, PersistException {
    Connection confrompool = null;
    try {
      confrompool = ConnectionPool.get();
      if (confrompool != connection) {
        synchronized (connection) {
          connection = confrompool;
        }
        parseStatements();
      }
      synchronized(updUsuarioStmt) {
        updUsuarioStmt.clearParameters();
        updUsuarioStmt.setString(1, Usuario.getNombre_corto());
        updUsuarioStmt.setString(2, Usuario.getNombre_completo());
        updUsuarioStmt.setString(3, Usuario.getPassword());
        updUsuarioStmt.setString(4, Usuario.getEmail());
        updUsuarioStmt.setString(5, Usuario.getEstatus());
        updUsuarioStmt.setTimestamp(6, new java.sql.Timestamp(Usuario.getFecha_registro().getTime()));
        updUsuarioStmt.setTimestamp(7, new java.sql.Timestamp(Usuario.getFecha_ultimo_acceso().getTime()));
        updUsuarioStmt.setString(8, Usuario.getAdministrador());
```

```
      updUsuarioStmt.setInt(9, Usuario.getId_usuario());
      int rowsChanged = updUsuarioStmt.executeUpdate();
      if (rowsChanged < 1)
        throw new UnknownPersistException("UsuarioPersist [update()]: " + "No se encontro el Usuario # " +
            Usuario.getId_usuario());
     }
    }
    catch (SQLException e) {
     throw new PersistException("UsuarioPersist [update()]: " + e.getMessage());
    }
    finally {
     if (confrompool!=null) {
      ConnectionPool.free(confrompool);
     }
    }
  }
  public void put(Usuario Usuario) throws PersistException {
    int new_id = idgen.getNextID();
    Connection confrompool = null;
    try {
     confrompool = ConnectionPool.get();
     if (confrompool != connection) {
      synchronized (connection) {
       connection = confrompool;
      }
      parseStatements();
     }
     synchronized(putUsuarioStmt) {
      putUsuarioStmt.clearParameters();
      Usuario.setId_usuario(new_id);
      putUsuarioStmt.setInt(1, Usuario.getId_usuario());
      putUsuarioStmt.setString(2, Usuario.getNombre_corto());
      putUsuarioStmt.setString(3, Usuario.getNombre_completo());
      putUsuarioStmt.setString(4, Usuario.getPassword());
      putUsuarioStmt.setString(5, Usuario.getEmail());
      putUsuarioStmt.setString(6, Usuario.getEstatus());
      putUsuarioStmt.setTimestamp(7, new java.sql.Timestamp(new java.util.Date().getTime()));
      putUsuarioStmt.setTimestamp(8, new java.sql.Timestamp(new java.util.Date().getTime()));
      putUsuarioStmt.setString(9, Usuario.getAdministrador());
      putUsuarioStmt.executeUpdate();
     }
    }
    catch (SQLException e) {
     throw new PersistException("UsuarioPersist [put()]: " + e.getMessage());
    }
    finally {
```

```
      if (confrompool!=null) {
        ConnectionPool.free(confrompool);
      }
    }
  }
  public void removeUsuario(int id_usuario) throws PersistException {
    Connection confrompool = null;
    try {
      confrompool = ConnectionPool.get();
      if (confrompool != connection) {
        synchronized (connection) {
          connection = confrompool;
        }
        parseStatements();
      }
      synchronized(remUsuarioStmt) {
        remUsuarioStmt.clearParameters();
        remUsuarioStmt.setInt(1, id_usuario);
        int rowsChanged = remUsuarioStmt.executeUpdate();
        if (rowsChanged < 1)
          throw new UnknownPersistException("UsuarioPersist: " + "No se pudo borrar el Usuario # "
            + id_usuario);
      }
    }
    catch (SQLException e) {
      throw new PersistException("UsuarioPersist [removeUsuario()]: " + e.getMessage());
    }
    finally {
      if (confrompool!=null) {
        ConnectionPool.free(confrompool);
      }
    }
  }
  public void destroy() {
  }
}
```

Apéndice D. Clase UsuarioIDGenerator (versión Java).-

```
package com.netapplications.webforums_jsp.persist;

import java.sql.*;
import com.netapplications.webforums_jsp.util.Settings;

public class UsuarioIDGenerator {
  private int increment = 100;
  private int max = 0;
  private int currentID = 0;

  private Connection connection;
  private static UsuarioIDGenerator instance;
  private PreparedStatement updSurrogateStmt;
  private PreparedStatement getSurrogateStmt;

  public static UsuarioIDGenerator getInstance() throws PersistException {
    instance = null;
    if (instance == null) {
      instance = new UsuarioIDGenerator();
    }
    return instance;
  }

  private synchronized void parseStatements() throws SQLException {
    getSurrogateStmt = connection.prepareStatement(Settings.get("sql.UsuarioIDGenerator.get"));
    updSurrogateStmt = connection.prepareStatement(Settings.get("sql.UsuarioIDGenerator.upd"));
  }

  private UsuarioIDGenerator() throws PersistException {
    try {
      connection = ConnectionPool.get();
      parseStatements();
      ConnectionPool.free(connection);
    }
    catch (SQLException se) {
      throw new PersistException("UsuarioIDGenerator [UsuarioIDGenerator()]: " + se.getMessage());
    }
  }

  public synchronized int getNextID() throws PersistException {
    if (++currentID > max) {
      Connection confrompool = null;
```

```
      try {
        confrompool = ConnectionPool.get();
        if (confrompool != connection) {
          synchronized (connection) {
            connection = confrompool;
          }
          parseStatements();
        }
        synchronized(updSurrogateStmt) {
          updSurrogateStmt.clearParameters();
          updSurrogateStmt.setInt(1, increment);
          int rowsChanged = updSurrogateStmt.executeUpdate();
          if (rowsChanged < 1) {
            throw new UnknownPersistException("UsuarioIDGenerator (getNextID()): " + " Failed to update Surrogates");
          }
        }
        synchronized(getSurrogateStmt) {
          ResultSet results = getSurrogateStmt.executeQuery();
          if (results.next()) {
            currentID = results.getInt("NEXTIDSET") - increment;
            max = currentID + increment - 1;
          } else {
            throw new UnknownPersistException("UsuarioIDGenerator [getNextID()]: " + " Failed to read Surrogates");
          }
        }
      } catch (SQLException e) {
        throw new PersistException("UsuarioIDGenerator [getNextID()]: " + e.getMessage());
      }
      finally {
        if (confrompool!=null) {
          ConnectionPool.free(confrompool);
        }
      }
    }
    return currentID;
  }
}
```

Apéndice E. Clase GetUserFromLoginCommand (versión Java).-

```
package com.netapplications.webforums_jsp.control;

import javax.servlet.*;
import javax.servlet.http.*;
import com.netapplications.webforums_jsp.domain.Usuario;
import com.netapplications.webforums_jsp.persist.*;
import com.netapplications.webforums_jsp.types.*;

public class GetUserFromLoginCommand extends Command {

  private String nextLoginOK;
  private String nextLoginWrong;

  public GetUserFromLoginCommand(String nextLoginOK, String nextLoginWrong) {
    this.nextLoginOK = nextLoginOK;
    this.nextLoginWrong = nextLoginWrong;
  }

  public String execute(HttpServletRequest req)
    throws CommandException {
    HttpSession session = req.getSession(true);
    try {
      UsuarioPersist udb = UsuarioPersist.getInstance();
      Usuario user = udb.getUsuarioLogin(req.getParameter("nombre_corto"),req.getParameter("password"));
      session.setAttribute("usuario", user);
      if (user.getAdministrador().equals(StrBoolean.TRUE)) {
       session.setAttribute("role", Roles.ADMINISTRATOR);
      } else {
       session.setAttribute("role", Roles.NOT_MEMBER);
      }
      return nextLoginOK;
    } catch (UnknownPersistException e) {
      session.setAttribute("role", Roles.NOT_LOGGED);
      req.setAttribute("webforums.msg", e.getMessage());
      return nextLoginWrong;
    } catch (PersistException e) {
      throw new CommandException("GetUserFromLoginCommand: " + e.getMessage());
    }
  }
}
```

Apéndice F. Clase GetAllMessagesOfTopicCommand (versión Java).-

```
package com.netapplications.webforums_jsp.control;

import javax.servlet.*;
import javax.servlet.http.*;
import com.netapplications.webforums_jsp.domain.Topico;
import com.netapplications.webforums_jsp.domain.VistaMensaje;
import com.netapplications.webforums_jsp.domain.MensajeTree;
import com.netapplications.webforums_jsp.persist.*;
import com.netapplications.webforums_jsp.types.*;
import com.netapplications.webforums_jsp.util.CookieUtils;

public class GetAllMessagesOfTopicCommand extends Command {
  private String nextThreadView;
  private String nextFlatView;
  private boolean sameMode;

  public GetAllMessagesOfTopicCommand(String nextThreadView, String nextFlatView, boolean sameMode) {
    this.nextThreadView = nextThreadView;
    this.nextFlatView = nextFlatView;
    this.sameMode = sameMode;
  }

  public GetAllMessagesOfTopicCommand(String nextThreadView, String nextFlatView) {
    this(nextThreadView, nextFlatView, true);
  }

  public String execute(HttpServletRequest req)
    throws CommandException {
      throw new CommandException("GetAllMessagesOfTopicCommand: This call is not allowed");
  }

  public String execute(HttpServletRequest req, HttpServletResponse res)
    throws CommandException {
    HttpSession session = req.getSession(true);
    try {
      TopicoPersist tdb = TopicoPersist.getInstance();
      VistaMensajePersist vmdb = VistaMensajePersist.getInstance();
      int id_topico;
      synchronized (req) {
        if (req.getAttribute("id_topico")!=null) {
```

```
        id_topico = Integer.parseInt((String) req.getAttribute("id_topico"));
       } else {
        id_topico = Integer.parseInt(req.getParameter("id_topico"));
       }
      }
      Topico topico = tdb.get(id_topico);
      req.setAttribute("topico", topico);
            MensajeTree mtree = vmdb.getMensajesTopicoTree(topico.getId_topico());
      req.setAttribute("arbolmensajes", mtree);
      String vmode = "";
      if (req.getParameter("view_mode")!=null) {
       vmode = req.getParameter("view_mode");
      }
      if (!(vmode.equalsIgnoreCase(ModoVisualizacion.ARBOL) || vmode.equalsIgnoreCase(ModoVisualizacion.PLANO)) &&
req.getAttribute("view_mode")!=null) {
       vmode = (String) req.getAttribute("view_mode");
      }
      if (!(vmode.equalsIgnoreCase(ModoVisualizacion.ARBOL) || vmode.equalsIgnoreCase(ModoVisualizacion.PLANO)) &&
CookieUtils.isFoundCookie(req, "view_mode")) {
       vmode = CookieUtils.getCookieValue(req, "view_mode");
      }
      if (!(vmode.equalsIgnoreCase(ModoVisualizacion.ARBOL) || vmode.equalsIgnoreCase(ModoVisualizacion.PLANO))) {
       vmode = ModoVisualizacion.ARBOL;
      }
      if (!sameMode) {
       vmode = ModoVisualizacion.alternaModo(vmode);
      }
      Cookie cookie = new Cookie("view_mode", vmode);
      cookie.setMaxAge(60*60*24*365); // un año
      res.addCookie(cookie);
      String rolForo = Roles.NOT_LOGGED;
      if (session.getAttribute("role")!=null) {
       rolForo = (String) session.getAttribute("role");
      }
      if (rolForo.equals(Roles.NOT_LOGGED)) {
       req.setAttribute("webforums.hint", "You need to register to be able to participate on this forum");
      } else if (rolForo.equals(Roles.NOT_MEMBER)) {
       req.setAttribute("webforums.hint", "You need to subscribe to be able to participate on this forum");
      } else if (rolForo.equals(Roles.SUSPENDED)) {
       req.setAttribute("webforums.hint", "Your subscription was suspended by one moderator of this forum");
      } else if (rolForo.equals(Roles.INACTIVE_MEMBER)) {
       req.setAttribute("webforums.hint", "You need to reactivate your subscription to be able to participate on this forum");
      }
      if (vmode.equalsIgnoreCase(ModoVisualizacion.ARBOL)) {
       return nextThreadView;
      } else {
```

```
        return nextFlatView;
      }
    }
    catch (NumberFormatException e) {
      throw new CommandException("GetAllMessagesOfTopicCommand: invalid ID");
    }
    catch (PersistException e) {
      throw new CommandException("GetAllMessagesOfTopicCommand: " + e.getMessage());
    }
  }
}
```

Apéndice G. Clase MensajeTree (versión Java).-

```
package com.netapplications.webforums_jsp.domain;

import java.util.*;

public class MensajeTree extends Object {
  private class MensajeNulo implements IMensajeMinimo {
    public int getId_mensaje() {
      return 0;
    }
    public int getId_mensaje_original() {
      return 0;
    }
  }
  private IMensajeMinimo mensaje_primero;
  private Vector nodos;

  public MensajeTree(IMensajeMinimo mensaje) {
    if (mensaje == null) {
      this.mensaje_primero = new MensajeNulo();
    } else {
      this.mensaje_primero = mensaje;
    }
    nodos = new Vector(5,2);
  }

  public MensajeTree() {
    this(null);
  }

  public boolean add(IMensajeMinimo mensaje) throws NoSuchElementException {
    if (mensaje.getId_mensaje_original() == mensaje_primero.getId_mensaje()) {
      return nodos.add(new MensajeTree(mensaje));
    } else {
      for (int i=0;i<nodos.size();i++) {
        try {
          if (((MensajeTree) nodos.get(i)).add(mensaje)) {
            return true;
          }
        } catch (NoSuchElementException ignore) {}
      }
      throw new NoSuchElementException("Element (" + mensaje.getId_mensaje()
        +") has no parents in the tree");
    }
  }
```

```
public Object[] toArray(Object[] a) {
  Collection mensajes = new ArrayList();
  // primero va el padre (si es que no es un mensaje nulo)
  if (mensaje_primero.getId_mensaje()>0) {
    mensajes.add(mensaje_primero);
  }
  // luego sus hijitos... y nietos, bisnietos, etc.
  for (int i=0;i<nodos.size();i++) {
    Object[] arr = ((MensajeTree) nodos.get(i)).toArray(a);
    for (int j=0;j<arr.length;j++) {
      mensajes.add(arr[j]);
    }
  }
  return (Object[])mensajes.toArray(a);
}

public Integer[] analyzeDepthArray() {
  return this.analyzeDepthArray(new Integer(0));
}

public Integer[] analyzeDepthArray(Integer currDeep) {
  Collection profundidades = new ArrayList();
  // primero va el padre (si es que no es un mensaje nulo)
  if (mensaje_primero.getId_mensaje()>0) {
    profundidades.add(currDeep);
  }
  // luego sus hijitos... y nietos, bisnietos, etc.
  for (int i=0;i<nodos.size();i++) {
    Integer[] arr = ((MensajeTree) nodos.get(i)).analyzeDepthArray(new Integer(currDeep.intValue()+1));
    for (int j=0;j<arr.length;j++) {
      profundidades.add(arr[j]);
    }
  }
  return (Integer[])profundidades.toArray(new Integer[0]);
}

public int[] depthArray() {
  int[] arr;
  Integer[] deep;
  deep = analyzeDepthArray();
  arr = new int[deep.length];
  for (int j=0;j<deep.length;j++){
    arr[j] = deep[j].intValue();
  }
  return arr;
```

```
  }
}
```

Apéndice H. Clase MainServlet (versión Java).-

```
package com.netapplications.webforums_jsp.servlets;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import com.netapplications.webforums_jsp.util.Settings;
import com.netapplications.webforums_jsp.control.*;
import com.netapplications.webforums_jsp.persist.*;
import com.netapplications.webforums_jsp.domain.*;
import com.netapplications.webforums_jsp.view.*;

public class MainServlet extends HttpServlet {

  private CommandsHashMap commands;
  private NavItemsVector navitems;
  private String error;
  private String jspdir;
  private String firstcmdkey = "go-home";
  private long refresh_time;
  private long last_refresh = 0;

  public void init(ServletConfig config) throws ServletException {
    super.init(config);
    initSettings();
    initServletVariables();
    initPersistCounters();
    initGlobalObjects();
    initCommands();
    initNavItems();
  }

  public String getServletInfo() {
    return "com.netapplications.webforums_jsp.servlets.MainServlet Information";
  }

  private synchronized void initSettings() throws ServletException {
    String path = getServletContext().getInitParameter("webforumsConfigFile");
    try {
      Settings.getInstance().load(path);
    } catch (IOException e) {
      throw new ServletException("initSettings: " + e.getMessage() + " {" + path + "}");
    }
  }
```

```
private synchronized void initServletVariables() throws ServletException {
  try {
    error = Settings.get("servlet.errorPage");
    jspdir = Settings.get("servlet.jspDir");
    refresh_time = Long.parseLong(Settings.get("servlet.refreshTime"));
  } catch (Exception e) {
    throw new ServletException("initServletVariables: " + e.getMessage());
  }
}

private synchronized void initPersistCounters() throws ServletException {
  try {
    ForoPersist fp = ForoPersist.getInstance();
    fp.fixCounts();
    TopicoPersist tp = TopicoPersist.getInstance();
    tp.fixCounts();
  }
  catch (PersistException e) {
    throw new ServletException("initPersistCounters: " + e.getMessage());
  }
}

private synchronized void initGlobalObjects() throws ServletException {
  try {
    EmoticonPersist ep = EmoticonPersist.getInstance();
    Emoticon[] emotList = ep.getAllEmoticons();
    getServletContext().setAttribute("emoticons", emotList);
    ForoPersist fp = ForoPersist.getInstance();
    Foro[] forosList = fp.getForosActivos();
    getServletContext().setAttribute("foros", forosList);
  }
  catch (PersistException e) {
    throw new ServletException("initGlobalObjects: " + e.getMessage());
  }
}

private synchronized void initCommands() {
  commands = new CommandsHashMap();
  commands.putCommand("go-home", new NullCommand("home.jsp"));
  commands.putCommand("go-login", new NullCommand("login.jsp"));
  commands.putCommand("do-login", new GetUserFromLoginCommand("home.jsp","login.jsp"));
  commands.putCommand("do-logout", new EndSessionCommand("home.jsp"));
  commands.putCommand("go-profile-within-forum", new GetUserProfileCommand("viewuser.jsp","login.jsp"));
  commands.putCommand("go-profile", new GetUserProfileCommand("profile.jsp","login.jsp"));
  commands.putCommand("do-upd-profile", new UpdUserProfileCommand("home.jsp"));
```

```
    commands.putCommand("go-add-user", new EndSessionCommand("register.jsp"));
    commands.putCommand("do-add-user", new AddUserCommand("home.jsp","register.jsp"));
    commands.putCommand("go-forum", new GetAllTopicsOfForumCommand("forumtop.jsp"));
    commands.putCommand("go-subscribe", new GetForumSubscriptionCommand("subscription.jsp"));
    commands.putCommand("do-subscribe", new UpdForumSubscriptionCommand("forumtop.jsp"));
    commands.putCommand("go-view-subscripcions", new GetAllSuscriptionsForumCommand("members.jsp"));
    commands.putCommand("go-upd-moderators", new GetAllActiveSuscriptionsForumCommand("moderators.jsp"));
    commands.putCommand("do-upd-moderators", new UpdSusbscriptionsTypeCommand("forumtop.jsp"));
    commands.putCommand("go-upd-suspended", new
GetAllActiveNormSuspSuscriptionsForumCommand("suspended.jsp"));
    commands.putCommand("do-upd-suspended", new UpdSusbscriptionsTypeCommand("forumtop.jsp"));
    commands.putCommand("go-topic", new GetAllMessagesOfTopicCommand("topicthread.jsp","topicflat.jsp",true));
    commands.putCommand("go-topic-switch-view", new
GetAllMessagesOfTopicCommand("topicthread.jsp","topicflat.jsp",false));
    commands.putCommand("go-add-forum", new NullCommand("newforum.jsp"));
    commands.putCommand("do-add-forum", new AddForumCommand("home.jsp"));
    commands.putCommand("go-add-topic", new NullCommand("newtopic.jsp"));
    commands.putCommand("go-find-topic", new NullCommand("findtopics.jsp"));
    commands.putCommand("do-find-topic", new FindTopicsCommand("topicsfound.jsp"));
    commands.putCommand("go-find-message", new NullCommand("findmessages.jsp"));
    commands.putCommand("do-find-message", new FindMessagesCommand("messagesfound.jsp"));
    commands.putCommand("do-del-message", new DelMessageCascadeCommand("topicthread.jsp","topicflat.jsp"));
    commands.putCommand("do-del-topic", new DelTopicCascadeCommand("forumtop.jsp",true));
    commands.putCommand("do-del-topic-found", new DelTopicCascadeCommand("findtopics.jsp",false));
    commands.putCommand("do-del-forum", new DelForumCascadeCommand("home.jsp"));
    commands.putCommand("do-edit-message", new UpdMessageContentCommand("topicthread.jsp","topicflat.jsp"));
    commands.putCommand("go-upd-emoticons", new NullCommand("emoticons.jsp"));
    commands.putCommand("do-del-emoticons", new DelEmoticonCascadeCommand("home.jsp"));
    commands.putCommand("do-upd-emoticons", new UpdEmoticonCommand("home.jsp"));
    commands.putCommand("go-next-topic", new FetchAnotherTopicOfForum("topicthread.jsp","topicflat.jsp",false));
    commands.putCommand("go-prev-topic", new FetchAnotherTopicOfForum("topicthread.jsp","topicflat.jsp",true));
  }

  private synchronized void initNavItems() {
    navitems = new NavItemsVector();
    navitems.addNavItem(
      new NavButton("go-home","home.gif","Ir a la página inicial","Inicio"),
      Roles.getRolesListExcluding(null),
      commands.getKeysListExcluding("go-home,do-login,do-logout,do-upd-profile,do-add-user,do-add-forum,do-del-
forum,do-upd-emoticons,do-del-emoticons"));
    navitems.addNavItem(
      new NavButton("go-login","register.gif","Ir a la página de registro","Registro"),
      Roles.NOT_LOGGED,
      "go-home,do-logout,do-del-forum");
    navitems.addNavItem(
      new NavButton("go-profile","profile.gif","Ir a la página de mantenimiento de sus datos personales","Perfil"),
```

```
      Roles.getRolesListExcluding(Roles.NOT_LOGGED),
      "go-home,do-login,do-upd-profile,do-add-user,do-add-forum,do-del-forum,do-upd-emoticons,do-del-emoticons");
    navitems.addNavItem(
      new NavButton("go-add-forum","newtopic.gif","Agregar un nuevo foro","Nuevo"),
      Roles.ADMINISTRATOR,
      "go-home,do-login,do-upd-profile,do-add-user,do-add-forum,do-del-forum,do-upd-emoticons,do-del-emoticons");
    navitems.addNavItem(
      new NavButton("go-upd-emoticons","newtopic.gif","Editar el catalogo de emoticons","Emoticons"),
      Roles.ADMINISTRATOR,
      "go-home,do-login,do-upd-profile,do-add-user,do-add-forum,do-del-forum,do-upd-emoticons,do-del-emoticons");
    navitems.addNavItem(
      new NavButton("go-forum","up.gif","Regresar a la página inicial del foro","Foro"),
      Roles.getRolesListExcluding(null),
      "go-topic,go-topic-switch-view,go-add-topic,do-del-message,do-edit-message,go-prev-topic,go-next-topic,go-
subscribe,go-upd-moderators,go-upd-suspended,go-profile-within-forum,go-view-subscripcions");
    navitems.addNavItem(
      new NavButton("go-prev-topic","prev.gif","Tópico anterior","Atras"),
      Roles.getRolesListExcluding(null),
      "go-topic,go-topic-switch-view,go-add-topic,do-del-message,do-edit-message,go-prev-topic,go-next-topic");
    navitems.addNavItem(
      new NavButton("go-next-topic","next.gif","Tópico siguiente","Adelante"),
      Roles.getRolesListExcluding(null),
      "go-topic,go-topic-switch-view,go-add-topic,do-del-message,do-edit-message,go-prev-topic,go-next-topic");
    navitems.addNavItem(
      new NavButton("go-subscribe","check.gif","Parámetros de Suscripción","Suscripción"),
      Roles.getRolesListExcluding(Roles.NOT_LOGGED),
      "go-forum,do-subscribe,do-upd-moderators,do-upd-suspended,do-del-topic");
    navitems.addNavItem(
      new NavButton("go-view-subscripcions","users.gif","Ver a todos los miembros del foro","Miembros"),
      Roles.getRolesListExcluding(null),
      "go-forum,do-subscribe,do-upd-moderators,do-upd-suspended,do-del-topic");
    navitems.addNavItem(
      new NavButton("go-upd-moderators","checkuser.gif","Definir a los moderadores del foro","Moderadores"),
      Roles.ADMINISTRATOR+","+Roles.ADMIN_MODERATOR,
      "go-forum,do-subscribe,do-upd-moderators,do-upd-suspended,do-del-topic");
    navitems.addNavItem(
      new NavButton("go-upd-suspended","checkuser.gif","Bloquear(Censurar)/Desbloquear usuarios","Bloqueo"),
      Roles.ADMINISTRATOR+","+Roles.ADMIN_MODERATOR+","+Roles.MODERATOR,
      "go-forum,do-subscribe,do-upd-moderators,do-upd-suspended,do-del-topic");
    navitems.addNavItem(
      new NavButton("go-add-topic","newtopic.gif","Agregar un nuevo tópico","Nuevo"),
      Roles.ADMINISTRATOR+","+Roles.ADMIN_MODERATOR+","+Roles.MODERATOR+","+Roles.MEMBER,
      "go-forum,do-subscribe,do-upd-moderators,do-upd-suspended,do-del-topic");
    navitems.addNavItem(
      new NavButton("go-topic-switch-view","viewmode.gif","Cambiar modo visualización lista/arbol","Modo"),
      Roles.getRolesListExcluding(null),
```

```
      "go-topic,go-topic-switch-view,do-del-message,do-edit-message,go-prev-topic,go-next-topic");
    navitems.addNavItem(
      new NavButton("go-find-topic","searchlist.gif","Búsqueda de topicos","Tópicos"),
      Roles.getRolesListExcluding(null),
      "go-home,do-login,do-upd-profile,do-add-user,do-add-forum,do-find-topic,do-del-forum,do-upd-emoticons,do-del-
emoticons");
    navitems.addNavItem(
      new NavButton("go-find-message","searchlist.gif","Búsqueda de mensajes","Mensajes"),
      Roles.getRolesListExcluding(null),
      "go-home,do-login,do-upd-profile,do-add-user,do-add-forum,do-find-message,do-del-forum,do-upd-emoticons,do-del-
emoticons");
    navitems.addNavItem(
      new NavComboBox("go-forum", "id_foro", "Navegación rápida entre foros", "Seleccione un Foro",
      new ForosICatalogProvider(getServletContext())),
      Roles.getRolesListExcluding(null),
      commands.getKeysListExcluding(null));
    navitems.addNavItem(
      new NavButton("do-logout","exit.gif","Finalizar su sesión en WebForums","Terminar"),
      Roles.getRolesListExcluding(Roles.NOT_LOGGED),
      "go-home,do-login,do-upd-profile,do-add-user,do-add-forum,do-del-forum,do-upd-emoticons,do-del-emoticons");
  }

  public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
    Command cmd;
    String next;
    HttpSession session = req.getSession(true);
    if (session.getAttribute("usuario")==null ){
       session.setAttribute("role", Roles.NOT_LOGGED);
    }
    try {
      String cmdKey;
      if (req.getParameter("cmd") != null) {
        cmdKey = req.getParameter("cmd");
      } else {
        cmdKey = firstcmdkey;
      }
      cmd = commands.getCommand(cmdKey);
      next = cmd.execute(req, res);
      CommandToken.set(req);
      if (cmd.globalObjectUpdated() || last_refresh+refresh_time < (new Date()).getTime()) {
        initGlobalObjects();
        navitems.rebuildHTML();
        last_refresh = (new Date()).getTime();
      }
      req.setAttribute("navitems", navitems.getHTML((String) session.getAttribute("role"),cmdKey));
    }
```

```
    catch (CommandException e) {
      req.setAttribute("javax.servlet.jsp.jspException", e);
      next = error;
    }
    RequestDispatcher rd;
    rd = getServletContext().getRequestDispatcher(jspdir + next);
    rd.forward(req, res);
  }

  public void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
    if (req.getAttribute("flag_multipart")==null && req.getContentType()!=null &&
req.getContentType().toLowerCase().indexOf("multipart/form-data")>=0) {
      req.setAttribute("flag_multipart", "NOTNULL");
      RequestDispatcher rd;
      rd = getServletContext().getRequestDispatcher(Settings.get("servlet.addMessageServletAlias"));
      rd.forward(req, res);
    } else {
      doGet(req, res);
    }
  }
}
```

Apéndice I. Página forumtop.jsp (versión JSP).-

```
<%@ page language="java"
import="java.util.*,com.netapplications.webforums_jsp.view.*,com.netapplications.webforums_jsp.util.*,com.netapplications.webforums_jsp.domain.*,com.netapplications.webforums_jsp.control.*" %>
<jsp:useBean id="foro" scope="page" class="com.netapplications.webforums_jsp.domain.Foro"/>
<jsp:useBean id="vwtopico" scope="page" class="com.netapplications.webforums_jsp.domain.VistaTopico"/>
<%
//response.addHeader("pragma", "no-cache");
//response.addHeader("cache-control", "private");
//response.addDateHeader("Expires", 1);
foro = (Foro)request.getAttribute("foro");
PagedIterator pi = new PagedIterator((VistaTopico[])request.getAttribute("topicos"),20);
Date fec_ult_visita = (Date) request.getAttribute("fec_ult_visita");
int currPage = PagedIteratorUtils.getCurrentPage(request, pi);
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>WebForums - Pagina de los Topicos del Foro</TITLE>
  <LINK rel="stylesheet" type="text/css" href="styles/webforums.css">
  <META name="GENERATOR" content="Microsoft(R) WordPad">
  <META http-equiv="KEYWORDS" content="WebForums">
  <META http-equiv="DESCRIPTION" content="Pagina de Topicos en un Foro de WebForums">
</HEAD>
<SCRIPT language="javascript">
<!--
function ver_perfil(id) {
  document.forma.cmd.value = "go-profile-within-forum";
  document.forma.id_usuario.value = id;
  document.forma.submit();
}
function ver_topico(id) {
  document.forma.cmd.value = "go-topic";
  document.forma.id_topico.value = id;
  document.forma.submit();
}
function go_page(page) {
  document.forma.page.value = page;
  document.forma.cmd.value = "go-forum";
  document.forma.submit();
}
function Borrar() {
  document.forma.idlist.value = ObtenSeleccionados();
  if (document.forma.idlist.value.length>0 && window.confirm("¿Está seguro de eliminar este(os) tópico(s)?")) {
    document.forma.cmd.value = "do-del-topic";
```

```
    document.forma.submit();
  }
}
//-->
</SCRIPT>
<SCRIPT language="javascript" src="javascript/botones.js"></SCRIPT>
<SCRIPT language="javascript" src="javascript/checkbox.js"></SCRIPT>
<BODY topMargin="0" leftMargin="0" rightMargin="0" bottomMargin="0">
<%@ include file="top.jsp" %>
  <FORM name="forma" action="webforums" method="post">
<%=PagedIteratorUtils.getPagesLinksHTML(pi, currPage)%>
    <TABLE align="center" border="0" cellPadding="0" cellSpacing="0" width="95%">
      <CAPTION>
        <SPAN class="TableTitle"><%=StringUtils.HTMLEncode(foro.getNombre())%></SPAN><BR>
        <SPAN class="SmallText"><%=StringUtils.HTMLEncode(foro.getDescripcion())%></SPAN>
      </CAPTION>
      <TR class="RowTitle">
        <TD>
<%
String rol = (String) session.getAttribute("role");
if (Roles.isModerator(rol)) {
%>
          <INPUT type="checkbox" name="chkHead" onclick="chkHead_onclick()"> 
<%
}
%>
          <SPAN class="RowTitle">Tópico</SPAN>
        </TD>
        <TD align="right"><SPAN class="RowTitle">Mens.  </SPAN></TD>
        <TD><SPAN class="RowTitle">Autor</SPAN></TD>
        <TD><SPAN class="RowTitle">Creación</SPAN></TD>
        <TD><SPAN class="RowTitle">Ult. Mens.</SPAN></TD>
      </TR>
<%
int i=0;
Iterator it = pi.iterator(currPage);
while (it.hasNext()) {
  vwtopico = (VistaTopico) it.next();
  if (i++%2==0) {
%>
      <TR class="RowUpper">
<%
  } else {
%>
      <TR class="RowDown">
<%
```

```
  }
%>
      <TD>
<%
  if (Roles.isModerator(rol)) {
%>
        <INPUT type="checkbox" name="<%=vwtopico.getId_topico()%>"> 
<%
  }
%>
        <SPAN class="MediumText"><A
href="javascript:ver_topico(<%=vwtopico.getId_topico()%>)"><%=StringUtils.HTMLEncode(vwtopico.getNombre_topico())%>
</A><%=HTMLUtils.getNewImageHTML(vwtopico.getFecha_creacion(), fec_ult_visita)%></SPAN>
      </TD>
      <TD align="right"><SPAN class="MediumText"><%=vwtopico.getNum_respuestas()%>  </SPAN></TD>
      <TD><SPAN class="MediumText"><A
href="javascript:ver_perfil(<%=vwtopico.getId_usuario()%>)"><%=vwtopico.getNombre_usuario()%></A></SPAN></TD>
      <TD><SPAN class="MediumText"><%=CustomFormats.StringDate(vwtopico.getFecha_creacion())%></SPAN></TD>
      <TD><SPAN
class="MediumText"><%=CustomFormats.StringDate(vwtopico.getFecha_ultimo_mensaje())%></SPAN></TD>
    </TR>
<%
}
if (Roles.isModerator(rol)) {
%>
    <TR>
      <TD colSpan="5" align="center">
        <BR>
<SCRIPT language="javascript">
<!--
writeButton('Borrar');
//-->
</SCRIPT>
      </TD>
    </TR>
<%
}
%>
  </TABLE>
  <INPUT type="Hidden" name="cmd" value="go-topic">
  <INPUT type="Hidden" name="id_foro" value="<%=foro.getId_foro()%>">
  <INPUT type="Hidden" name="id_topico">
  <INPUT type="Hidden" name="id_usuario">
  <INPUT type="Hidden" name="page" value="<%=currPage%>">
  <INPUT type="Hidden" name="idlist">
  <INPUT type="hidden" name="token" value="<%=request.getAttribute("token")%>">
```

```
  </FORM>
<%@ include file="bottom.jsp" %>
</BODY>
</HTML>
```

Apéndice J. Página profile.jsp (versión JSP).-

```
<%@ page language="java"
import="java.util.Iterator,com.netapplications.webforums_jsp.view.*,com.netapplications.webforums_jsp.util.*,com.netapplicat
ions.webforums_jsp.domain.*,com.netapplications.webforums_jsp.control.*" %>
<jsp:useBean id="user" scope="page" class="com.netapplications.webforums_jsp.domain.Usuario"/>
<jsp:useBean id="profile" scope="page" class="com.netapplications.webforums_jsp.domain.PerfilUsuario"/>
<%
//response.addHeader("pragma", "no-cache");
//response.addHeader("cache-control", "private");
//response.addDateHeader("Expires", 1);
user = (Usuario) request.getAttribute("usuario");
profile = (PerfilUsuario) request.getAttribute("profile");
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>WebForums - Pagina de Actualizacion de Perfil</TITLE>
  <LINK rel="stylesheet" type="text/css" href="styles/webforums.css">
  <META name="GENERATOR" content="Microsoft(R) WordPad">
  <META http-equiv="KEYWORDS" content="WebForums">
  <META http-equiv="DESCRIPTION" content="Pagina de Actualizacion de Perfil de WebForums">
</HEAD>
<SCRIPT language="javascript">
<!--
function ValidaForma() {
 var re = / /i;
  if (!document.forma.old_password.value.replace(re,"").length) {
    document.forma.old_password.focus();
    alert("La constraseña anterior es obligatoria.");
  } else if (document.forma.password_confirm.value!=document.forma.password.value) {
    document.forma.password.focus();
    alert("La nueva constraseña y su confirmación no coinciden.");
  } else if (!document.forma.nombre_completo.value.replace(re,"").length) {
    document.forma.nombre_completo.focus();
    alert("El nombre completo es obligatorio.");
  } else if (!document.forma.email.value.replace(re,"").length) {
    document.forma.email.focus();
    alert("El Email es obligatorio.");
  } else if (!(document.forma.email.value.indexOf("@")>0 &&
document.forma.email.value.indexOf("@")<document.forma.email.value.length-1)) {
    document.forma.email.focus();
    alert("El Email es invalido.");
  } else {
    return true;
  }
```

```
  return false;
}
function Aceptar(){
  if (ValidaForma()) {
    document.forma.cmd.value = "do-upd-profile";
    document.forma.submit();
  }
}
function Deshacer(){
  document.forma.reset();
}
//-->
</SCRIPT>
<SCRIPT language="javascript" src="javascript/botones.js"></SCRIPT>
<BODY topMargin="0" leftMargin="0" rightMargin="0" bottomMargin="0">
<%@ include file="top.jsp" %>
  <FORM name="forma" action="webforums" method="post" onSubmit="return ValidaForma();">
    <TABLE align="center" border="0" cellPadding="0" cellSpacing="0" width="95%">
    <DIV align="center"  class="SmallText">
Registre aquí información adicional de usted (su perfil)<BR>
También puede cambiar su Nombre Completo, Email y Contraseña.<BR>
<CITE>Si desea conservar su contraseña anterior deje en blanco<BR>
los campos de contraseña nueva y confirmación de contraseña nueva.</CITE>
    </DIV>
    <DIV align="center"  class="BigText">Perfil</DIV>
    <TABLE align="center" border="0" cellPadding="0" cellSpacing="0" width="90%">
      <TR class="RowTitle"><TD colspan="2" align="center"><SPAN class="RowTitle">Indique sus datos
personales</SPAN></TD></TR>
     <TR>
       <TD colSpan="2" align="center">
        <FIELDSET>
          <TABLE>
            <TR>
              <TD align="right" height="30px">
                <SPAN class="MediumText"> Alias (sólo lectura) </SPAN> </TD>
              <TD align="left">
                <INPUT name="nombre_corto" maxLength="20" size="20" readOnly onFocus="blur();"
value="<%=user.getNombre_corto()%>">
              </TD>
            </TR>
            <TR>
              <TD align="right" height="30px">
                <SPAN class="MediumText">* Contraseña Anterior </SPAN>
              </TD>
              <TD align="left">
                <INPUT type="password" name="old_password" maxLength="20" size="20">
```

```
    <INPUT type="hidden" name="pwd" value="<%=user.getPassword()%>">
  </TD>
</TR>
<TR>
  <TD align="right" height="30px">
    <SPAN class="MediumText"> Nueva Contraseña </SPAN>
  </TD>
  <TD align="left">
    <INPUT type="password" name="password" maxLength="20" size="20">
    <INPUT type="password" name="password_confirm" maxLength="20" size="20">
  </TD>
</TR>
<TR>
  <TD align="right" height="30px">
    <SPAN class="MediumText">* Nombre Completo </SPAN></TD>
  <TD align="left">
    <INPUT name="nombre_completo" maxLength="80" size="42" value="<%=user.getNombre_completo()%>">
  </TD>
</TR>
<TR>
  <TD align="right" height="30px">
    <SPAN class="MediumText">* Email </SPAN> </TD>
  <TD align="left">
    <INPUT name="email" maxLength="100" size="42" value="<%=user.getEmail()%>">
  </TD>
</TR>
<TR>
  <TD align="right" height="30px">
    <SPAN class="MediumText"> Titulo </SPAN> </TD>
  <TD align="left">
    <INPUT name="titulo" maxLength="100" size="42" value="<%=profile.getTitulo()%>">
  </TD>
</TR>
<TR>
  <TD align="right" height="30px">
    <SPAN class="MediumText"> Compañia </SPAN> </TD>
  <TD align="left">
    <INPUT name="compania" maxLength="100" size="42" value="<%=profile.getCompania()%>">
  </TD>
</TR>
<TR>
  <TD align="right" height="30px" rowSpan="2">
    <SPAN class="MediumText"> Dirección </SPAN> </TD>
  <TD align="left">
    <INPUT name="direccion1" maxLength="100" size="42" value="<%=profile.getDireccion1()%>">
  </TD>
```

```
          </TR>
          <TR>
            <TD align="left">
              <INPUT name="direccion2" maxLength="100" size="42" value="<%=profile.getDireccion2()%>">
            </TD>
          </TR>
          <TR>
            <TD align="right" height="30px">
              <SPAN class="MediumText"> Ciudad </SPAN> </TD>
            <TD align="left">
              <INPUT name="ciudad" maxLength="100" size="42" value="<%=profile.getCiudad()%>">
            </TD>
          </TR>
          <TR>
            <TD align="right" height="30px">
              <SPAN class="MediumText"> Estado </SPAN> </TD>
            <TD align="left">
              <INPUT name="estado" maxLength="100" size="42" value="<%=profile.getEstado()%>">
            </TD>
          </TR>
          <TR>
            <TD align="right" height="30px">
              <SPAN class="MediumText"> C.P. </SPAN> </TD>
            <TD align="left">
              <INPUT name="cp" maxLength="20" size="5" value="<%=profile.getCp()%>">
            </TD>
          </TR>
          <TR>
            <TD align="right" height="30px">
              <SPAN class="MediumText"> Teléfono </SPAN> </TD>
            <TD align="left">
              <INPUT name="telefono" maxLength="40" size="15" value="<%=profile.getTelefono()%>">
            </TD>
          </TR>
          <TR>
            <TD align="right" height="30px">
              <SPAN class="MediumText"> URI/URL </SPAN> </TD>
            <TD align="left">
              <TEXTAREA name="uri" cols="40" rows="5"><%=StringUtils.HTMLEncode(profile.getUri())%></TEXTAREA>
          </TD>
          </TR>
          <TR>
            <TD align="center" colSpan="2" height="70px">
<SCRIPT language="javascript">
<!--
writeButtonList('Aceptar,Deshacer',' ');
```

```
//-->
</SCRIPT>
            </TD>
          </TR>
        </TABLE>
      </FIELDSET>
    </TD>
  </TR>
 </TABLE>
 <INPUT type="Hidden" name="cmd" value="do-upd-profile">
 <INPUT type="Hidden" name="perfilusuario_id_usuario" value="<%=profile.getId_usuario()%>">
 <INPUT type="Hidden" name="id_foro">
 <INPUT type="Hidden" name="token" value="<%=request.getAttribute("token")%>">
</FORM>
<%@ include file="bottom.jsp" %>
</BODY>
</HTML>
```

Apéndice K. Clase Usuario (versión Visual Basic).-

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
  Persistable = 0  'NotPersistable
  DataBindingBehavior = 0  'vbNone
  DataSourceBehavior  = 0  'vbNone
  MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "Usuario"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
Attribute VB_Ext_KEY = "SavedWithClassBuilder6" ,"Yes"
Attribute VB_Ext_KEY = "Top_Level" ,"Yes"
Option Explicit

'local variable(s) to hold property value(s)
Private udtProps As UsuarioProps

Private stbool As WebForumsTypes.StrBoolean
Private stesta As WebForumsTypes.estatus

Public Property Let administrador(ByVal vData As String)
  If UCase(vData) = stbool.ValueTRUE Then
    udtProps.administrador = stbool.ValueTRUE
  Else
    udtProps.administrador = stbool.ValueFALSE
  End If
End Property

Public Property Get administrador() As String
  If udtProps.administrador = "" Then
    administrador = stbool.ValueFALSE
  Else
    administrador = udtProps.administrador
  End If
End Property

Public Property Let fecha_ultimo_acceso(ByVal vData As Date)
  udtProps.fecha_ultimo_acceso = vData
```

```
End Property

Public Property Get fecha_ultimo_acceso() As Date
  fecha_ultimo_acceso = udtProps.fecha_ultimo_acceso
End Property

Public Property Let fecha_registro(ByVal vData As Date)
  udtProps.fecha_registro = vData
End Property

Public Property Get fecha_registro() As Date
  fecha_registro = udtProps.fecha_registro
End Property

Public Property Let estatus(ByVal vData As String)
  If UCase(vData) = stesta.ACTIVO Then
    udtProps.estatus = stesta.ACTIVO
  Else
    udtProps.estatus = stesta.INACTIVO
  End If
End Property

Public Property Get estatus() As String
  If udtProps.estatus = "" Then
    estatus = stesta.INACTIVO
  Else
    estatus = udtProps.estatus
  End If
End Property

Public Property Let email(ByVal vData As String)
  udtProps.email = vData
End Property

Public Property Get email() As String
  email = Trim$(udtProps.email)
End Property

Public Property Let password(ByVal vData As String)
  udtProps.password = vData
End Property

Public Property Get password() As String
  password = Trim$(udtProps.password)
End Property
```

```
Public Property Let nombre_completo(ByVal vData As String)
  udtProps.nombre_completo = vData
End Property

Public Property Get nombre_completo() As String
  nombre_completo = Trim$(udtProps.nombre_completo)
End Property

Public Property Let nombre_corto(ByVal vData As String)
  udtProps.nombre_corto = vData
End Property

Public Property Get nombre_corto() As String
  nombre_corto = Trim$(udtProps.nombre_corto)
End Property

Public Property Let id_usuario(ByVal vData As Long)
  udtProps.id_usuario = vData
End Property

Public Property Get id_usuario() As Long
  id_usuario = udtProps.id_usuario
End Property

Public Property Get email_nombreCorto() As String
  Dim loc_email As String
  loc_email = Trim$(udtProps.email)
  If InStr(1, loc_email, "@", vbTextCompare) > 1 _
      And InStr(1, loc_email, "@", vbTextCompare) < Len(loc_email) Then
    email_nombreCorto = "<a href=""mailto:" & loc_email & """>" & _
      Trim$(udtProps.nombre_corto) & "</a>"
  Else
    email_nombreCorto = ""
  End If
End Property

Public Sub SetState(Buffer As String)
  Dim udtData As UsuarioData
  Dim PBag As PropertyBag
  Dim arrbyte_buff() As Byte

  arrbyte_buff = Buffer
  Set PBag = New PropertyBag
  With PBag
    .Contents = arrbyte_buff
    udtData.Buffer = .ReadProperty("fixedlength_state")
```

```
  End With
  Set PBag = Nothing
  LSet udtProps = udtData
End Sub

Public Function GetState() As String
  Dim udtData As UsuarioData
  Dim PBag As PropertyBag

  LSet udtData = udtProps
  Set PBag = New PropertyBag
  With PBag
    .WriteProperty "fixedlength_state", udtData.Buffer
    GetState = .Contents
  End With
  Set PBag = Nothing
End Function

Private Sub Class_Initialize()
  Set stbool = CreateObject("WebForumsTypes.StrBoolean")
  Set stesta = CreateObject("WebForumsTypes.Estatus")
End Sub
```

Apéndice L. Clase Estatus (versión Visual Basic).-

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
  Persistable = 0  'NotPersistable
  DataBindingBehavior = 0  'vbNone
  DataSourceBehavior  = 0  'vbNone
  MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "Estatus"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
Attribute VB_Ext_KEY = "SavedWithClassBuilder6" ,"Yes"
Attribute VB_Ext_KEY = "Top_Level" ,"Yes"
Public Property Get INACTIVO() As String
    INACTIVO = "I"
End Property

Public Property Get ACTIVO() As String
    ACTIVO = "A"
End Property

Public Function getDescription(currentStatus As String) As String
    If currentStatus = ACTIVO Then
        getDescription = "Activo"
    ElseIf currentStatus = INACTIVO Then
        getDescription = "Inactivo"
    Else
        getDescription = "Desconocido"
    End If
End Function
```

Apéndice M. Clase VectorUsuario (versión Visual Basic).-

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
  Persistable = 0  'NotPersistable
  DataBindingBehavior = 0  'vbNone
  DataSourceBehavior  = 0  'vbNone
  MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "VectorUsuario"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
Option Explicit

Private mVector As New CVectorObj

Public Function NewEnum() As IUnknown
Attribute NewEnum.VB_UserMemId = -4
Attribute NewEnum.VB_MemberFlags = "40"
  Set NewEnum = mVector.NewEnum
End Function

Private Sub Add(ByVal Buffer As String)
  Dim aNewUsuario As New Usuario

  With aNewUsuario
    .SetState Buffer
  End With
  With mVector
    Set .item(.Last + 1) = aNewUsuario
  End With
End Sub

Public Sub SetState(Buffer As String)
  Dim PBag As PropertyBag
  Dim itmBuffer As Variant
  Dim lngIndex As Long
  Dim arrbyte_buff() As Byte
  Dim total As Long
```

```
  arrbyte_buff = Buffer
  Set PBag = New PropertyBag
  With PBag
    .Contents = arrbyte_buff
    total = .ReadProperty("Count")
    If total > 0 Then
       mVector.Chunk = total
       mVector.Clear
       For lngIndex = 1 To total
         Add .ReadProperty("Item" & lngIndex)
       Next
    End If
  End With
End Sub

Public Function PagedIterator(Optional ByVal page As Long, Optional ByVal pageSize As Integer) As PagedIterator
  Dim objIter As PagedIterator
  Set objIter = New PagedIterator
  objIter.Init mVector.toArray, page, pageSize
  Set PagedIterator = objIter
End Function

Public Function item(ByVal Index As Long) As Usuario
  Set item = mVector.item(Index)
End Function

Public Function Count() As Long
  Count = mVector.Last
End Function
```

Apéndice N. Clase UsuarioPersist (versión Visual Basic).-

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
  Persistable = 0  'NotPersistable
  DataBindingBehavior = 0  'vbNone
  DataSourceBehavior  = 0  'vbNone
  MTSTransactionMode  = 3  'UsesTransaction
END
Attribute VB_Name = "UsuarioPersist"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
Attribute VB_Ext_KEY = "SavedWithClassBuilder6" ,"Yes"
Attribute VB_Ext_KEY = "Top_Level" ,"Yes"
Option Explicit

Implements ObjectControl

Private mobjContext As ObjectContext
Private mflgInMTS As Boolean

Public Function getUsuario(ByVal id_usuario As Long) As String
Dim rs As New adodb.Recordset
Dim strSQL As String
Dim udtProps As UsuarioProps

  On Error GoTo errhandler

  strSQL = "SELECT * FROM USUARIO WHERE ID_USUARIO = " & id_usuario
  rs.Open strSQL, DBCONNECTION, adOpenForwardOnly, adLockReadOnly
  With udtProps
    .id_usuario = id_usuario
    .nombre_corto = rs!nombre_corto & ""
    .nombre_completo = rs!nombre_completo & ""
    .password = rs!password & ""
    .email = rs!email & ""
    .estatus = rs!estatus & ""
    .fecha_registro = rs!fecha_registro
    .fecha_ultimo_acceso = rs!fecha_ultimo_acceso
    .administrador = rs!administrador & ""
```

```
  End With
  rs.Close
  Set rs = Nothing
  getUsuario = SerializeUsuario(udtProps)
  If mflgInMTS Then mobjContext.SetComplete
  Exit Function

errhandler:
  App.LogEvent "UsuarioPersist [getUsuario()]: " & Err.Number & ": " & Err.Description & " (id_usuario=" & id_usuario & ")"
  Set rs = Nothing
  If mflgInMTS Then mobjContext.SetAbort
  Err.Raise Err.Number
End Function

Public Function getUsuarioLogin(ByVal nombre_corto As String, ByVal password As String) As String
  Dim cn As New adodb.Connection
  Dim cm As New adodb.Command
  Dim rs As New adodb.Recordset
  Dim strSQL As String
  Dim udtProps As UsuarioProps

  On Error GoTo errhandler

  strSQL = "SELECT * FROM USUARIO WHERE NOMBRE_CORTO = ? AND PASSWORD = ?"
  cn.ConnectionString = DBCONNECTION
  cn.CursorLocation = adUseClient
  cn.Open
  cm.ActiveConnection = cn
  cm.CommandType = adCmdText
  cm.CommandText = strSQL
  cm.Parameters(0).Value = nombre_corto
  cm.Parameters(1).Value = password
  rs.CursorLocation = adUseClient
  rs.LockType = adLockReadOnly
  rs.CursorType = adOpenStatic
  Set rs = cm.Execute
  Set rs.ActiveConnection = Nothing
  cn.Close
  Set cm = Nothing
  Set cn = Nothing
  With rs
    If .EOF Then
      Err.Raise vbObjectError + 1001, , "No se encontro el Usuario : " + nombre_corto + " ó la contraseña es inválida"
    Else
      udtProps.id_usuario = .Fields("id_usuario").Value
      udtProps.nombre_corto = .Fields("nombre_corto").Value & ""
```

```
      udtProps.nombre_completo = .Fields("nombre_completo").Value & ""
      udtProps.password = .Fields("password").Value & ""
      udtProps.email = .Fields("email").Value & ""
      udtProps.estatus = .Fields("estatus").Value & ""
      udtProps.fecha_registro = .Fields("fecha_registro").Value
      udtProps.fecha_ultimo_acceso = .Fields("fecha_ultimo_acceso").Value
      udtProps.administrador = .Fields("administrador").Value & ""
    End If
    .Close
  End With
  Set rs = Nothing
  getUsuarioLogin = SerializeUsuario(udtProps)
  If mflgInMTS Then mobjContext.SetComplete
  Exit Function

errhandler:
  App.LogEvent "UsuarioPersist [getUsuarioLogin()]: " & Err.Number & ": " & Err.Description
  Set cm = Nothing
  Set cn = Nothing
  Set rs = Nothing
  If mflgInMTS Then mobjContext.SetAbort
  Err.Raise Err.Number
End Function

Public Function getUsuariosToSendEmail(ByVal id_mensaje As Long) As String
  Dim rs As New adodb.Recordset
  Dim strSQL As String
  Dim udtProps As UsuarioProps
  Dim lngIndex As Long
  Dim PBag As PropertyBag

  On Error GoTo errhandler

  Set PBag = New PropertyBag
  strSQL = "SELECT T1.* FROM USUARIO T1, SUSCRIPCION T2, TOPICO T3, MENSAJE T4 " _
    & "WHERE T1.ID_USUARIO = T2.ID_USUARIO AND T2.ID_FORO = T3.ID_FORO " _
    & "AND T3.ID_TOPICO = T4.ID_TOPICO AND T2.ESTATUS = 'A' " _
    & "AND T2.ENVIAR_MENSAJES = 'S' AND (T2.EMAIL_ACTUALIZACIONES = 'S' " _
    & "OR (T2.EMAIL_NUEVOS = 'S' AND T4.ID_MENSAJE_ORIGINAL IS NULL)) " _
    & "AND T4.ID_MENSAJE = " & id_mensaje
  rs.Open strSQL, DBCONNECTION, adOpenForwardOnly, adLockReadOnly
  With rs
    While Not .EOF
      lngIndex = lngIndex + 1
      udtProps.id_usuario = .Fields("id_usuario").Value
      udtProps.nombre_corto = .Fields("nombre_corto").Value & ""
```

```
      udtProps.nombre_completo = .Fields("nombre_completo").Value & ""
      udtProps.password = .Fields("password").Value & ""
      udtProps.email = .Fields("email").Value & ""
      udtProps.estatus = .Fields("estatus").Value & ""
      udtProps.fecha_registro = .Fields("fecha_registro").Value
      udtProps.fecha_ultimo_acceso = .Fields("fecha_ultimo_acceso").Value
      udtProps.administrador = .Fields("administrador").Value & ""
      PBag.WriteProperty "Item" & lngIndex, SerializeUsuario(udtProps)
      .MoveNext
    Wend
    .Close
  End With
  Set rs = Nothing
  PBag.WriteProperty "Count", lngIndex
  getUsuariosToSendEmail = PBag.Contents
  If mflgInMTS Then mobjContext.SetComplete
  Exit Function

errhandler:
  App.LogEvent "UsuarioPersist [getUsuariosToSendEmail()]: " & Err.Number & ": " & Err.Description & " (id_mensaje=" &
id_mensaje & ")"
  Set rs = Nothing
  If mflgInMTS Then mobjContext.SetAbort
  Err.Raise Err.Number
End Function

Public Function updateUsuario(ByVal Buffer As String) As String
  Dim udtProps As UsuarioProps

  On Error GoTo errhandler

  udtProps = DeserializeUsuario(Buffer)
  updateUsuario = saveUsuario(udtProps, False)
  If mflgInMTS Then mobjContext.SetComplete
  Exit Function

errhandler:
  App.LogEvent "UsuarioPersist [updateUsuario()]: " & Err.Number & ": " & Err.Description & " (id_usuario=" &
udtProps.id_usuario & ")"
  If mflgInMTS Then mobjContext.SetAbort
  Err.Raise Err.Number
End Function

Public Function putUsuario(ByVal Buffer As String) As String
  Dim udtProps As UsuarioProps
```

```
  On Error GoTo errhandler

  udtProps = DeserializeUsuario(Buffer)
  putUsuario = saveUsuario(udtProps, True)
  If mflgInMTS Then mobjContext.SetComplete
  Exit Function

errhandler:
  App.LogEvent "UsuarioPersist [putUsuario()]: " & Err.Number & ": " & Err.Description
  If mflgInMTS Then mobjContext.SetAbort
  Err.Raise Err.Number
End Function

Private Function saveUsuario(ByRef udtProps As UsuarioProps, ByVal isNew As Boolean) As String
Dim rs As New adodb.Recordset
Dim strSQL As String

  On Error GoTo errhandler

  strSQL = "SELECT * FROM USUARIO WHERE ID_USUARIO = " & udtProps.id_usuario
  rs.Open strSQL, DBCONNECTION, adOpenKeyset, adLockOptimistic
  With udtProps
    If isNew Then
      Dim UsuaIDGen As UsuarioIDGenerator
      If mflgInMTS Then
        Set UsuaIDGen = mobjCont ext.CreateInstance("WebForumsPersist.UsuarioIDGenerator")
      Else
        Set UsuaIDGen = CreateObject("WebForumsPersist.UsuarioIDGenerator")
      End If
      rs.AddNew
      .id_usuario = UsuaIDGen.getNextID
      Set UsuaIDGen = Nothing
      .fecha_registro = Now
    End If
    rs!id_usuario = .id_usuario
    rs!nombre_corto = Trim$(.nombre_corto)
    rs!nombre_completo = Trim$(.nombre_completo)
    rs!password = Trim$(.password)
    rs!email = Trim$(.email)
    rs!estatus = .estatus
    rs!fecha_registro = .fecha_registro
    rs!fecha_ultimo_acceso = .fecha_ultimo_acceso
    rs!administrador = .administrador
  End With
  rs.Update
  rs.Close
```

```
  Set rs = Nothing
  saveUsuario = SerializeUsuario(udtProps)
  Exit Function

errhandler:
  Set rs = Nothing
  Err.Raise Err.Number
End Function

Public Sub removeUsuario(ByVal id_usuario As Long)
Dim cn As adodb.Connection
Dim strSQL As String

  On Error GoTo errhandler

  Set cn = New adodb.Connection
  With cn
    .Open DBCONNECTION
    strSQL = "DELETE USUARIO WHERE ID_USUARIO = " & id_usuario
    .Execute strSQL
    .Close
  End With
  cn.Close
  Set cn = Nothing
  If mflgInMTS Then mobjContext.SetComplete
  Exit Sub

errhandler:
  App.LogEvent "UsuarioPersist [removeUsuario()]: " & Err.Number & ": " & Err.Description & " (id_usuario=" & id_usuario &
")"
  Set cn = Nothing
  If mflgInMTS Then mobjContext.SetAbort
  Err.Raise Err.Number
End Sub

Private Function DeserializeUsuario(ByVal Buffer As String) As UsuarioProps
  Dim udtData As UsuarioData
  Dim PBag As PropertyBag
  Dim arrbyte_buff() As Byte

  arrbyte_buff = Buffer
  Set PBag = New PropertyBag
  With PBag
    .Contents = arrbyte_buff
    udtData.Buffer = .ReadProperty("fixedlength_state")
  End With
```

```
  Set PBag = Nothing
  LSet DeserializeUsuario = udtData
End Function

Private Function SerializeUsuario(ByRef udtProps As UsuarioProps) As String
  Dim udtData As UsuarioData
  Dim PBag As PropertyBag

  LSet udtData = udtProps
  Set PBag = New PropertyBag
  With PBag
    .WriteProperty "fixedlength_state", udtData.Buffer
    SerializeUsuario = .Contents
  End With
  Set PBag = Nothing
End Function

Private Sub ObjectControl_Activate()
  Set mobjContext = GetObjectContext
  mflgInMTS = True
End Sub

Private Function ObjectControl_CanBePooled() As Boolean
  ObjectControl_CanBePooled = False
End Function

Private Sub ObjectControl_Deactivate()
  Set mobjContext = Nothing
End Sub
```

Apéndice O. Clase UsuarioIDGenerator (versión Visual Basic).-

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
  Persistable = 0  'NotPersistable
  DataBindingBehavior = 0  'vbNone
  DataSourceBehavior  = 0  'vbNone
  MTSTransactionMode  = 1  'NoTransaction
END
Attribute VB_Name = "UsuarioIDGenerator"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
Attribute VB_Ext_KEY = "SavedWithClassBuilder6" ,"Yes"
Attribute VB_Ext_KEY = "Top_Level" ,"Yes"
Option Explicit

Implements ObjectControl

Private mobjContext As ObjectContext
Private mflgInMTS As Boolean

Public Function getNextID() As Long
  Dim strSQL As String
  Dim rs As New adodb.Recordset
  Dim increment As Long
  Dim max As Long
  Dim currentID As Long
  Dim spmMgr As SharedPropertyGroupManager
  Dim spmGroup As SharedPropertyGroup
  Dim spmPropCurrentID As SharedProperty
  Dim spmPropMax As SharedProperty
  Dim bResult As Boolean

  On Error GoTo errhandler

  Set spmMgr = CreateObject("MTxSpm.SharedPropertyGroupManager.1")
  Set spmGroup = spmMgr.CreatePropertyGroup("UsuarioIDGenerator", LockMethod, Process, bResult)
  Set spmPropCurrentID = spmGroup.CreateProperty("currentID", bResult)
  If Not bResult Then
    spmPropCurrentID.Value = 0
```

```
  End If
  currentID = spmPropCurrentID.Value
  Set spmPropMax = spmGroup.CreateProperty("max", bResult)
  If Not bResult Then
    spmPropMax.Value = 0
  End If
  max = spmPropMax.Value
  increment = 1
  currentID = currentID + 1
  If currentID > max Then
    strSQL = "SELECT TABLE_NAME, NEXTIDSET FROM SURROGATES WHERE TABLE_NAME = 'USUARIO'"
    With rs
      .Open strSQL, DBCONNECTION, adOpenKeyset, adLockOptimistic
      currentID = .Fields("NEXTIDSET")
      .Fields("NEXTIDSET") = currentID + increment
      .Update
      .Close
    End With
    Set rs = Nothing
    max = currentID + increment - 1
  End If
  spmPropCurrentID.Value = currentID
  spmPropMax.Value = max
  getNextID = currentID
  If mflgInMTS Then mobjContext.SetComplete
  Exit Function

errhandler:
  App.LogEvent "EmoticonIDGenerator [getNextID()]: " & Err.Number & ": " & Err.Description
  Set rs = Nothing
  If mflgInMTS Then mobjContext.SetAbort
  Err.Raise Err.Number
End Function

Private Sub ObjectControl_Activate()
  Set mobjContext = GetObjectContext
  mflgInMTS = True
End Sub

Private Function ObjectControl_CanBePooled() As Boolean
  ObjectControl_CanBePooled = False
End Function

Private Sub ObjectControl_Deactivate()
  Set mobjContext = Nothing
End Sub
```

Apéndice P. Función GetUserFromLogin (versión VBScript).-

```
Function GetUserFromLogin(nextLoginOK, nextLoginWrong)
  Dim usuarioState
  Dim errNumber, errSource, errDescription
  On Error Resume Next
  usuarioState = usuariodb.getUsuarioLogin(Request("nombre_corto"), Request("password"))
  errNumber = Err.Number
  errDescription = Err.Description
  errSource = Err.Source
  On Error Goto 0
  If errNumber = 0 Then
    usuario.SetState CStr(usuarioState)
    If usuario.administrador = StrBoolean.ValueTRUE Then
      Session("role")  = Roles.ADMINISTRATOR
    Else
      Session("role")  = Roles.NOT_MEMBER
    End If
    Session("usuario") = usuarioState
    GetUserFromLogin = nextLoginOK
  Else
    If errNumber = -2147220503 Then
      webforums_msg = errDescription
      GetUserFromLogin = nextLoginWrong
    Else
      Err.Raise errNumber, errSource, "GetUserFromLogin: " & errDescription
    End If
  End If
End Function
```

Apéndice Q. Función GetAllMessagesOfTopic (versión VBScript).-

```
Function GetAllMessagesOfTopic(nextThreadView, nextFlatView, sameMode)
  Dim loc_id_topico, vmode, rolForo

  If EsMayorQueCero(id_topico) Then
    loc_id_topico = id_topico
  ElseIf RequestCollectionsValid Then
    loc_id_topico = Request("id_topico")
  Else
    Err.Raise 455
  End If
  topico.SetState topicodb.getTopico(Clng(loc_id_topico))
  vectorvmensaje.SetState vmensajedb.getMensajesTopico(Clng(loc_id_topico))
  vmode = ""
  If RequestCollectionsValid Then
    If Len(Request("view_mode"))>0 Then vmode = Request("view_mode")
  Else
    vmode = view_mode
  End If
  If (vmode <> ModoVisualizacion.ARBOL And vmode <> ModoVisualizacion.PLANO) And Len(view_mode) > 0 Then
    vmode = view_mode
  End If
  If (vmode <> ModoVisualizacion.ARBOL And vmode <> ModoVisualizacion.PLANO) And
CookieUtils_isFoundCookie("view_mode") Then
    vmode = Request.Cookies("view_mode")
  End If
  If vmode <> ModoVisualizacion.ARBOL And vmode <> ModoVisualizacion.PLANO Then
    vmode = ModoVisualizacion.ARBOL
  End If
  If Not sameMode Then
    vmode = ModoVisualizacion.alternaModo(CStr(vmode))
  End If
  Response.Cookies("view_mode") = vmode
  Response.Cookies("view_mode").Expires = DateAdd("yyyy", 1, Now) 'un año
  rolForo = Roles.NOT_LOGGED
  If Len(Session("role")) > 0 Then
    rolForo = Session("role")
  End If
  If rolForo = Roles.NOT_LOGGED Then
    webforums_hint = "You need to <A href=""javascript:document.forma.cmd.value=\'go-
login\';document.forma.submit();"">register</A> to be able to participate on this forum"
  ElseIf rolForo = Roles.NOT_MEMBER Then
```

```
      webforums_hint = "You need to <A href=""javascript:document.forma.cmd.value=\'go-
subscribe\';document.forma.submit();"">subscribe</A> to be able to participate on this forum"
    ElseIf rolForo = Roles.SUSPENDED Then
      webforums_hint = "Your subscription was suspended by one moderator of this forum"
    ElseIf rolForo = Roles.INACTIVE_MEMBER Then
      webforums_hint = "You need to reactivate your subscription to be able to participate on this forum"
    End If
    If vmode = ModoVisualizacion.ARBOL Then
      GetAllMessagesOfTopic = nextThreadView
    Else
      GetAllMessagesOfTopic = nextFlatView
    End If
End Function
```

Apéndice R. Clase MensajeTree (versión Visual Basic).-

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
  Persistable = 0  'NotPersistable
  DataBindingBehavior = 0  'vbNone
  DataSourceBehavior  = 0  'vbNone
  MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "MensajeTree"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
Option Explicit

Public mensaje_primero As IMensajeMinimo

Private nodos As CVectorObj

Private Sub Class_Initialize()
  Set mensaje_primero = New MensajeNulo
End Sub

Friend Sub Build(mensajes As Variant)
  Dim mens As Variant
  For Each mens In mensajes
    Add mens
  Next
End Sub

Friend Sub Add(vMensaje As Variant)
  Dim flagParentFound As Boolean
  Dim aLeaf As New MensajeTree
  Dim i As Integer
  Dim mensaje As IMensajeMinimo

  On Error GoTo errhandler

  Set mensaje = vMensaje
  With nodos
    If mensaje.IdMensajeOriginal = mensaje_primero.IdMensaje Then
```

```
      Set aLeaf.mensaje_primero = mensaje
      If nodos Is Nothing Then
        Set nodos = New CVectorObj
        Set nodos.item(1) = aLeaf
      Else
        Set .item(.Last + 1) = aLeaf
      End If
      Exit Sub
    Else
      If Not nodos Is Nothing Then
        For i = 1 To .Last
          flagParentFound = True
          Set aLeaf = .item(i)
          aLeaf.Add vMensaje
          If flagParentFound Then Exit Sub
        Next i
      End If
    End If
  End With

  On Error GoTo 0

  Err.Raise vbObjectError + 1001, , "Element (" & mensaje.IdMensaje & _
    ") has no parents in the tree"
    Exit Sub

errhandler:
  If Err.Number = vbObjectError + 1001 Then
    flagParentFound = False
    Resume Next
  Else
    Err.Raise Err.Number
  End If
End Sub

Public Function toArray() As Object()
  Dim i As Integer, j As Integer
  Dim vectObj As CVectorObj
  Dim arr() As Object
  Dim num_nodos As Integer

  If Not nodos Is Nothing Then num_nodos = nodos.Last
  'else num_nodos=0
  If Not nodos Is Nothing Or mensaje_primero.IdMensaje > 0 Then
    Set vectObj = New CVectorObj
    ' primero va el padre (si es que no es un mensaje nulo)
```

```
    If mensaje_primero.IdMensaje > 0 Then
      Set vectObj.item(1) = mensaje_primero
    End If
    ' luego sus hijitos... y nietos, bisnietos, etc.
    For i = 1 To num_nodos
      arr = nodos.item(i).toArray
      For j = LBound(arr) To UBound(arr)
        Set vectObj.item(vectObj.Last + 1) = arr(j)
      Next j
    Next i
    toArray = vectObj.toArray
  End If
End Function

Public Function depthArray(Optional ByVal currDeep As Integer = 0) As Integer()
  Dim i As Integer, j As Integer
  Dim vectInt As CVectorInt
  Dim arr() As Integer
  Dim num_nodos As Integer

  If Not nodos Is Nothing Then num_nodos = nodos.Last
  'else num_nodos=0
  If Not nodos Is Nothing Or mensaje_primero.IdMensaje > 0 Then
    Set vectInt = New CVectorInt
    ' primero va el padre (si es que no es un mensaje nulo)
    If mensaje_primero.IdMensaje > 0 Then
      vectInt.item(1) = currDeep
    End If
    ' luego sus hijitos... y nietos, bisnietos, etc.
    For i = 1 To num_nodos
      arr = nodos.item(i).depthArray(currDeep + 1)
      For j = LBound(arr) To UBound(arr)
        vectInt.item(vectInt.Last + 1) = arr(j)
      Next j
    Next i
    depthArray = vectInt.toArray
  End If
End Function

Public Function depthVariantArray(Optional ByVal currDeep As Integer = 0) As Variant
  Dim i As Integer
  Dim vRetVal()
  Dim arrIntTmp() As Integer

  arrIntTmp = depthArray(currDeep)
  ReDim vRetVal(LBound(arrIntTmp) To UBound(arrIntTmp))
```

```
  For i = LBound(arrIntTmp) To UBound(arrIntTmp)
    vRetVal(i) = arrIntTmp(i)
  Next i
  depthVariantArray = vRetVal
End Function
```

Apéndice S. Página main.asp (versión ASP).-

```
<%@ Language=VBScript %>
<% Option Explicit %>
<!-- #INCLUDE FILE="common.asp" -->
<!-- #INCLUDE FILE="domain.asp" -->
<!-- #INCLUDE FILE="persist.asp" -->
<!-- #INCLUDE FILE="dim.asp" –>
<!-- #INCLUDE FILE="commands.asp" -->
<%
On Error Resume Next
RequestCollectionsValid = Request.Form.Count >= 0
RequestCollectionsValid = Err.number = 0
On Error Goto 0
If Len(Session("RequestParameters")) > 0 Then
	For Each item In Split(Session("RequestParameters"), ",")
		Select Case LCase(Trim(Left(item, InStr(1, item, "=")-1)))
			Case "cmd" : cmd = Trim(Mid(item, InStr(1, item, "=")+1))
			Case "id_foro" : id_foro = CLng(Trim(Mid(item, InStr(1, item, "=")+1)))
			Case "id_topico" : id_topico = CLng(Trim(Mid(item, InStr(1, item, "=")+1)))
			Case "id_mensaje" : id_mensaje = CLng(Trim(Mid(item, InStr(1, item, "=")+1)))
			Case "page" : currPage = CInt(Trim(Mid(item, InStr(1, item, "=")+1)))
			Case "view_mode" : view_mode = Trim(Mid(item, InStr(1, item, "=")+1))
			Case "webforums_msg" : webforums_msg = Trim(Mid(item, InStr(1, item, "=")+1))
		End Select
	Next
	Session.Contents.Remove("RequestParameters")
End If
If RequestCollectionsValid Then
	If IsEmpty(Request("cmd")) then
		cmd = "go-home"
	Else
		cmd = Request("cmd")
	End If
ElseIf len(cmd) = 0 Then
	cmd = "go-home"
End If
%>
<!-- #INCLUDE FILE="dispatcher.asp" –>
```

Apéndice T. Página dispatcher.asp (versión ASP).-

```
<%
webforums_hint = ""
webforums_msg = ""
globalObjectUpdated = false
'*****************************************************
Execute "nextASP = " & commands.Item(cmd)
'*****************************************************
If globalObjectUpdated Then
        updGlobalObjects
End If
Set nv = Server.CreateObject("WebForumsView.NavItemsVector")
navitems = nv.getHTML(Session("role"), CStr(cmd))
Set nv = Nothing
rol = Session("role")
'*****************************************************
currPage = getCurrentPage
token = CommandToken.getToken(true)
'*****************************************************

Select Case nextASP
        case "home.asp"
%>
<!-- #INCLUDE FILE="home.asp" -->
<%
        case "login.asp"
%>
<!-- #INCLUDE FILE="login.asp" -->
<%
        case "forumtop.asp"
%>
<!-- #INCLUDE FILE="forumtop.asp" -->
<%
        case "profile.asp"
%>
<!-- #INCLUDE FILE="profile.asp" -->
<%
        case "viewuser.asp"
%>
<!-- #INCLUDE FILE="viewuser.asp" -->
<%
        case "register.asp"
```

```
%>
<!-- #INCLUDE FILE="register.asp" -->
<%
        case "newtopic.asp"
%>
<!-- #INCLUDE FILE="newtopic.asp" -->
<%
        case "topicthread.asp"
%>
<!-- #INCLUDE FILE="topicthread.asp" -->
<%
        case "topicflat.asp"
%>
<!-- #INCLUDE FILE="topicflat.asp" -->
<%
        case "findtopics.asp"
%>
<!-- #INCLUDE FILE="findtopics.asp" -->
<%
        case "findmessages.asp"
%>
<!-- #INCLUDE FILE="findmessages.asp" -->
<%
        case "emoticons.asp"
%>
<!-- #INCLUDE FILE="emoticons.asp" -->
<%
        case "members.asp"
%>
<!-- #INCLUDE FILE="members.asp" -->
<%
        case "messagesfound.asp"
%>
<!-- #INCLUDE FILE="messagesfound.asp" -->
<%
        case "moderators.asp"
%>
<!-- #INCLUDE FILE="moderators.asp" -->
<%
        case "newforum.asp"
%>
<!-- #INCLUDE FILE="newforum.asp" –>
<%
        case "subscription.asp"
%>
<!-- #INCLUDE FILE="subscription.asp" -->
```

```
<%
        case "suspended.asp"
%>
<!-- #INCLUDE FILE="suspended.asp" -->
<%
        case "topicsfound.asp"
%>
<!-- #INCLUDE FILE="topicsfound.asp" -->
<%
        case Else
%>
<!-- #INCLUDE FILE="nullcommand.asp" -->
<%
End Select
%>
```

Apéndice U. Archivo global.asa (versión ASP).-

```
<!-- METADATA TYPE="typelib" FILE="C:\WINNT\System32\scrrun.dll" –>
<OBJECT RUNAT="Server" SCOPE="Application" ID="commands" PROGID="Scripting.Dictionary"></OBJECT>
<OBJECT RUNAT="Server" SCOPE="Application" ID="emoticons" PROGID="Scripting.Dictionary"></OBJECT>
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
Sub Session_OnStart
  Dim user, Roles
  Set user = Server.CreateObject("WebForumsDomain.Usuario")
  ' Al inicio de la session el usuario no se ha identificado
  Session("usuario") = user.GetState
  set user = nothing
  Set Roles = Server.CreateObject("WebForumsControl.Roles")
  Session("role")  = Roles.NOT_LOGGED
  Set Roles = Nothing
End Sub

Sub Application_OnStart
  initConstants
  initPersistCounters
  initGlobalObjects
  initCommands
  initNavItems
End Sub

Sub initConstants
  Dim settings
  Set settings = Server.CreateObject("WebForumsUtil.Settings")
  Application("attachmentRealPath") = settings.getSetting("asp.attachmentRealPath")
  If Right(Application("attachmentRealPath"), 1) <> "\" Then
    Application("attachmentRealPath") = Application("attachmentRealPath") & "\"
    settings.setSetting "asp.attachmentRealPath", Application("attachmentRealPath")
  End If
  Set settings = Nothing
End Sub

Sub initCommands
  commands.Add "go-home", """home.asp"""
  commands.Add "go-login", """login.asp"""
  commands.Add "do-login", "GetUserFromLogin(""home.asp"",""login.asp"")"
  commands.Add "do-logout", "EndSession(""home.asp"")"
  commands.Add "go-profile", "GetUserProfile(""profile.asp"",""login.asp"")"
  commands.Add "go-profile-within-forum", "GetUserProfile(""viewuser.asp"",""login.asp"")"
  commands.Add "do-upd-profile", "UpdUserProfile(""home.asp"")"
  commands.Add "go-add-user", "EndSession(""register.asp"")"
```

```
  commands.Add "do-add-user", "AddUser(""home.asp"",""register.asp"")"
  commands.Add "go-forum", "GetAllTopicsOfForum(""forumtop.asp"",false)"
  commands.Add "go-forum-combo", "GetAllTopicsOfForum(""forumtop.asp"",false)"
  commands.Add "go-subscribe", "GetForumSubscription(""subscription.asp"")"
  commands.Add "do-subscribe", "UpdForumSubscription(""forumtop.asp"")"
  commands.Add "go-view-subscripcions", "GetAllSuscriptionsForum(""members.asp"")"
  commands.Add "go-upd-moderators", "GetAllActiveSuscriptionsForum(""moderators.asp"")"
  commands.Add "do-upd-moderators", "UpdSusbscriptionsType(""forumtop.asp"")"
  commands.Add "go-upd-suspended", "GetAllActiveNormSuspSuscriptionsForum(""suspended.asp"")"
  commands.Add "do-upd-suspended", "UpdSusbscriptionsType(""forumtop.asp"")"
  commands.Add "go-topic", "GetAllMessagesOfTopic(""topicthread.asp"",""topicflat.asp"",true)"
  commands.Add "go-topic-switch-view", "GetAllMessagesOfTopic(""topicthread.asp"",""topicflat.asp"",false)"
  commands.Add "go-add-forum", """newforum.asp"""
  commands.Add "do-add-forum", "AddForum(""home.asp"")"
  commands.Add "go-add-topic", """newtopic.asp"""
  commands.Add "go-find-topic", """findtopics.asp"""
  commands.Add "do-find-topic", "FindTopics(""topicsfound.asp"")"
  commands.Add "go-find-message", """findmessages.asp"""
  commands.Add "do-find-message", "FindMessages(""messagesfound.asp"")"
  commands.Add "do-del-message", "DelMessageCascade(""topicthread.asp"",""topicflat.asp"")"
  commands.Add "do-del-topic", "DelTopicCascade(""forumtop.asp"",true)"
  commands.Add "do-del-topic-found", "DelTopicCascade(""findtopics.asp"",false)"
  commands.Add "do-del-forum", "DelForumCascade(""home.asp"")"
  commands.Add "do-edit-message", "UpdMessageContent(""topicthread.asp"",""topicflat.asp"")"
  commands.Add "go-upd-emoticons", """emoticons.asp"""
  commands.Add "do-del-emoticons", "DelEmoticonCascade(""home.asp"")"
  commands.Add "do-upd-emoticons", "UpdEmoticon(""home.asp"")"
  commands.Add "go-next-topic", "FetchAnotherTopicOfForum(""topicthread.asp"",""topicflat.asp"",false)"
  commands.Add "go-prev-topic", "FetchAnotherTopicOfForum(""topicthread.asp"",""topicflat.asp"",true)"
End Sub

Sub initPersistCounters
  Dim fp, tp
  Set fp = Server.CreateObject("WebForumsPersist.ForoPersist")
  fp.fixCounts
  set fp = nothing

  Set tp = Server.CreateObject("WebForumsPersist.TopicoPersist")
  tp.fixCounts
  set tp = nothing
End Sub

Sub initGlobalObjects
  Dim ep, fp, ve, auxState, item, HTMLUtils

  Set ep = Server.CreateObject("WebForumsPersist.EmoticonPersist")
```

```
  auxState = ep.getAllEmoticons
  Set ep = Nothing
  Application("emoticons") = auxState
  Set ve = Server.CreateObject("WebForumsDomain.VectorEmoticon")
  ve.SetState CStr(auxState)
  For Each item in ve
    emoticons.Add CStr(item.id_emoticon), item.uri
  Next
  Set ve = Nothing
  Set HTMLUtils = Server.CreateObject("WebForumsUtil.HTMLUtils")
  HTMLUtils.setEmoticonsTableHTML()
  Set HTMLUtils = Nothing
  Set fp = Server.CreateObject("WebForumsPersist.ForoPersist")
  Application("foros") = fp.getForosActivos
  Set fp = Nothing
End Sub

Function commandsGetKeysListExcluding(excList)
  Dim arrKeys, arrList, itm1, itm2, found, list
  arrExcList = Split(excList,",")
  arrKeys = commands.Keys
  list = ""
  For Each itm1 In arrKeys
    found = False
    For Each itm2 In arrExcList
      If itm1 = itm2 Then
        found = True
        Exit For
      End If
    Next
    If Not found Then
      list = list & itm1 & ","
    End If
  Next
  If Len(list) > 0 Then
    list = Left(list, Len(list) - 1)
  End If
  commandsGetKeysListExcluding = list
End Function

Sub initNavItems
  Dim nv, Roles
  Set Roles = Server.CreateObject("WebForumsControl.Roles")
  Set nv = Server.CreateObject("WebForumsView.NavItemsVector")

  nv.addNavButton _
```

```
    "go-home","home.gif","Ir a la página inicial","Inicio", _
    Roles.getRolesListExcluding, _
    commandsGetKeysListExcluding("go-home,do-login,do-logout,do-upd-profile,do-add-user,do-add-forum,do-del-
forum,do-upd-emoticons,do-del-emoticons")
  nv.addNavButton _
    "go-login","register.gif","Ir a la página de registro","Registro", _
    Roles.NOT_LOGGED, _
    "go-home,do-logout,do-del-forum"
  nv.addNavButton _
    "go-profile","profile.gif","Ir a la página de mantenimiento de sus datos personales","Perfil", _
    Roles.getRolesListExcluding(Roles.NOT_LOGGED), _
    "go-home,do-login,do-upd-profile,do-add-user,do-add-forum,do-del-forum,do-upd-emoticons,do-del-emoticons"
  nv.addNavButton _
    "go-add-forum","newtopic.gif","Agregar un nuevo foro","Nuevo", _
    Roles.ADMINISTRATOR, _
    "go-home,do-login,do-upd-profile,do-add-user,do-add-forum,do-del-forum,do-upd-emoticons,do-del-emoticons"
  nv.addNavButton _
    "go-upd-emoticons","newtopic.gif","Editar el catalogo de emoticons","Emoticons", _
    Roles.ADMINISTRATOR, _
    "go-home,do-login,do-upd-profile,do-add-user,do-add-forum,do-del-forum,do-upd-emoticons,do-del-emoticons"
  nv.addNavButton _
    "go-forum","up.gif","Regresar a la página inicial del foro","Foro", _
    Roles.getRolesListExcluding, _
    "go-topic,go-topic-switch-view,go-add-topic,do-del-message,do-edit-message,go-prev-topic,go-next-topic,go-
subscribe,go-upd-moderators,go-upd-suspended,go-profile-within-forum,go-view-subscripcions"
  nv.addNavButton _
    "go-prev-topic","prev.gif","Tópico anterior","Atras", _
    Roles.getRolesListExcluding, _
    "go-topic,go-topic-switch-view,go-add-topic,do-del-message,do-edit-message,go-prev-topic,go-next-topic"
  nv.addNavButton _
    "go-next-topic","next.gif","Tópico siguiente","Adelante", _
    Roles.getRolesListExcluding, _
    "go-topic,go-topic-switch-view,go-add-topic,do-del-message,do-edit-message,go-prev-topic,go-next-topic"
  nv.addNavButton _
    "go-subscribe","check.gif","Parámetros de Suscripción","Suscripción", _
    Roles.getRolesListExcluding(Roles.NOT_LOGGED), _
    "go-forum,go-forum-combo,do-subscribe,do-upd-moderators,do-upd-suspended,do-del-topic"
  nv.addNavButton _
    "go-view-subscripcions","users.gif","Ver a todos los miembros del foro","Miembros", _
    Roles.getRolesListExcluding, _
    "go-forum,go-forum-combo,do-subscribe,do-upd-moderators,do-upd-suspended,do-del-topic"
  nv.addNavButton _
    "go-upd-moderators","checkuser.gif","Definir a los moderadores del foro","Moderadores", _
    Roles.ADMINISTRATOR & "," & Roles.ADMIN_MODERATOR, _
    "go-forum,go-forum-combo,do-subscribe,do-upd-moderators,do-upd-suspended,do-del-topic"
  nv.addNavButton _
```

```
    "go-upd-suspended","checkuser.gif","Bloquear(Censurar)/Desbloquear usuarios","Bloqueo", _
    Roles.ADMINISTRATOR & "," & Roles.ADMIN_MODERATOR & "," & Roles.MODERATOR, _
    "go-forum,go-forum-combo,do-subscribe,do-upd-moderators,do-upd-suspended,do-del-topic"
  nv.addNavButton _
    "go-add-topic","newtopic.gif","Agregar un nuevo tópico","Nuevo", _
    Roles.ADMINISTRATOR & "," & Roles.ADMIN_MODERATOR & "," & Roles.MODERATOR & "," & Roles.MEMBER, _
    "go-forum,go-forum-combo,do-subscribe,do-upd-moderators,do-upd-suspended,do-del-topic"
  nv.addNavButton _
    "go-topic-switch-view","viewmode.gif","Cambiar modo visualización lista/arbol","Modo", _
    Roles.getRolesListExcluding, _
    "go-topic,go-topic-switch-view,do-del-message,do-edit-message,go-prev-topic,go-next-topic"
  nv.addNavButton _
    "go-find-topic","searchlist.gif","Búsqueda de topicos","Tópicos", _
    Roles.getRolesListExcluding, _
    "go-home,do-login,do-upd-profile,do-add-user,do-add-forum,do-find-topic,do-del-forum,do-upd-emoticons,do-del-
emoticons"
  nv.addNavButton _
    "go-find-message","searchlist.gif","Búsqueda de mensajes","Mensajes", _
    Roles.getRolesListExcluding, _
    "go-home,do-login,do-upd-profile,do-add-user,do-add-forum,do-find-message,do-del-forum,do-upd-emoticons,do-del-
emoticons"
  nv.addNavComboBox _
    "go-forum-combo", "id_foro", "Navegación rápida entre foros", "Seleccione un Foro",_
    "ForosICatalogProvider", _
    Roles.getRolesListExcluding, _
    commandsGetKeysListExcluding("")
  nv.addNavButton _
    "do-logout","exit.gif","Finalizar su sesión en WebForums","Terminar", _
    Roles.getRolesListExcluding(Roles.NOT_LOGGED), _
    "go-home,do-login,do-upd-profile,do-add-user,do-add-forum,do-del-forum,do-upd-emoticons,do-del-emoticons"

  Set Roles = Nothing
  Set nv = Nothing
End Sub
</SCRIPT>
```

Apéndice V. Página forumtop.asp (versión ASP).-

```
<%
Set pi = vectorvtopico.PagedIterator(currPage, 20)
currPage = pi.currentPage
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>WebForums - Pagina de los Topicos del Foro</TITLE>
  <LINK rel="stylesheet" type="text/css" href="styles/webforums.css">
  <META name="GENERATOR" content="Microsoft(R) WordPad">
  <META http-equiv="KEYWORDS" content="WebForums">
  <META http-equiv="DESCRIPTION" content="Pagina de Topicos en un Foro de WebForums">
</HEAD>
<SCRIPT language="javascript">
<!--
function ver_perfil(id) {
  document.forma.cmd.value = "go-profile-within-forum";
  document.forma.id_usuario.value = id;
  document.forma.submit();
}
function ver_topico(id) {
  document.forma.cmd.value = "go-topic";
  document.forma.id_topico.value = id;
  document.forma.submit();
}
function go_page(page) {
  document.forma.page.value = page;
  document.forma.cmd.value = "go-forum";
  document.forma.submit();
}
function Borrar() {
  document.forma.idlist.value = ObtenSeleccionados();
  if (document.forma.idlist.value.length>0 && window.confirm("¿Está seguro de eliminar este(os) tópico(s)?")) {
    document.forma.cmd.value = "do-del-topic";
    document.forma.submit();
  }
}
//-->
</SCRIPT>
<SCRIPT language="javascript" src="javascript/botones.js"></SCRIPT>
<SCRIPT language="javascript" src="javascript/checkbox.js"></SCRIPT>
```

```
<BODY topMargin="0" leftMargin="0" rightMargin="0" bottomMargin="0">
<!-- #INCLUDE FILE="top.asp" –>
  <FORM name="forma" action="main.asp" method="post">
<%=""%>
    <TABLE align="center" border="0" cellPadding="0" cellSpacing="0" width="95%">
      <CAPTION>
       <SPAN class="TableTitle"><%=Server.HTMLEncode(foro.nombre)%></SPAN><BR>
       <SPAN class="SmallText"><%=Server.HTMLEncode(foro.descripcion)%></SPAN>
      </CAPTION>
      <TR class="RowTitle">
       <TD>
<%
If Roles.isModerator(CStr(rol)) Then
%>
         <INPUT type="checkbox" name="chkHead" onclick="chkHead_onclick()"> 
<%
End If
%>
         <SPAN class="RowTitle">Tópico</SPAN>
       </TD>
       <TD align="right"><SPAN class="RowTitle">Mens.  </SPAN></TD>
       <TD><SPAN class="RowTitle">Autor</SPAN></TD>
       <TD><SPAN class="RowTitle">Creación</SPAN></TD>
       <TD><SPAN class="RowTitle">Ult. Mens.</SPAN></TD>
      </TR>
<%
i = 0
For Each item In pi
  If i Mod 2 = 0 Then
%>
      <TR class="RowUpper">
<%
  Else
%>
      <TR class="RowDown">
<%
  End If
%>
       <TD>
<%
  If Roles.isModerator(CStr(rol)) Then
%>
         <INPUT type="checkbox" name="<%=item.id_topico%>"> 
<%
  End If
%>
```

```
        <SPAN class="MediumText"><A
href="javascript:ver_topico(<%=item.id_topico%>)"><%=Server.HTMLEncode(item.nombre_topico)%></A><%=HTMLUtils.g
etNewImageHTML(CDate(item.fecha_creacion), CDate(fec_ult_visita))%></SPAN>
       </TD>
       <TD align="right"><SPAN class="MediumText"><%=item.num_respuestas%>  </SPAN></TD>
       <TD><SPAN class="MediumText"><A
href="javascript:ver_perfil(<%=item.id_usuario%>)"><%=Server.HTMLEncode(item.nombre_usuario)%></A></SPAN></TD>
       <TD><SPAN class="MediumText"><%=CustomFormats.StringDate(item.fecha_creacion)%></SPAN></TD>
       <TD><SPAN class="MediumText"><%=CustomFormats.StringDate(item.fecha_ultimo_mensaje)%></SPAN></TD>
      </TR>
<%
            i = i + 1
Next
If Roles.isModerator(CStr(rol)) Then
%>
      <TR>
       <TD colSpan="5" align="center">
        <BR>
<SCRIPT language="javascript">
<!--
writeButton('Borrar');
//-->
</SCRIPT>
       </TD>
      </TR>
<%
End If
%>
    </TABLE>
    <INPUT type="Hidden" name="cmd" value="go-topic">
    <INPUT type="Hidden" name="id_foro" value="<%=foro.id_foro%>">
    <INPUT type="Hidden" name="id_topico">
    <INPUT type="Hidden" name="id_usuario">
    <INPUT type="Hidden" name="page" value="<%=currPage%>">
    <INPUT type="Hidden" name="idlist">
    <INPUT type="hidden" name="token" value="<%=token%>">
  </FORM>
<!-- #INCLUDE FILE="bottom.asp" -->
</BODY>
```

Apéndice W. Página profile.asp (versión ASP).-

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>WebForums - Pagina de Actualizacion de Perfil</TITLE>
  <LINK rel="stylesheet" type="text/css" href="styles/webforums.css">
  <META name="GENERATOR" content="Microsoft(R) WordPad">
  <META http-equiv="KEYWORDS" content="WebForums">
  <META http-equiv="DESCRIPTION" content="Pagina de Actualizacion de Perfil de WebForums">
</HEAD>
<SCRIPT language="javascript">
<!--
function ValidaForma() {
 var re = / /i;
  if (!document.forma.old_password.value.replace(re,"").length) {
    document.forma.old_password.focus();
    alert("La constraseña anterior es obligatoria.");
  } else if (document.forma.password_confirm.value!=document.forma.password.value) {
    document.forma.password.focus();
    alert("La nueva constraseña y su confirmación no coinciden.");
  } else if (!document.forma.nombre_completo.value.replace(re,"").length) {
    document.forma.nombre_completo.focus();
    alert("El nombre completo es obligatorio.");
  } else if (!document.forma.email.value.replace(re,"").length) {
    document.forma.email.focus();
    alert("El Email es obligatorio.");
  } else if (!(document.forma.email.value.indexOf("@")>0 &&
document.forma.email.value.indexOf("@")<document.forma.email.value.length-1)) {
    document.forma.email.focus();
    alert("El Email es invalido.");
  } else {
    return true;
  }
  return false;
}
function Aceptar(){
  if (ValidaForma()) {
    document.forma.cmd.value = "do-upd-profile";
    document.forma.submit();
  }
}
function Deshacer(){
  document.forma.reset();
}
```

```
//-->
</SCRIPT>
<SCRIPT language="javascript" src="javascript/botones.js"></SCRIPT>
<BODY topMargin="0" leftMargin="0" rightMargin="0" bottomMargin="0">
<!-- #INCLUDE FILE="top.asp" -->
  <FORM name="forma" action="main.asp" method="post" onSubmit="return ValidaForma();">
    <TABLE align="center" border="0" cellPadding="0" cellSpacing="0" width="95%">
    <DIV align="center"  class="SmallText">
Registre aquí información adicional de usted (su perfil)<BR>
También puede cambiar su Nombre Completo, Email y Contraseña.<BR>
<CITE>Si desea conservar su contraseña anterior deje en blanco<BR>
los campos de contraseña nueva y confirmación de contraseña nueva.</CITE>
    </DIV>
    <DIV align="center"  class="BigText">Perfil</DIV>
    <TABLE align="center" border="0" cellPadding="0" cellSpacing="0" width="90%">
      <TR class="RowTitle"><TD colspan="2" align="center"><SPAN class="RowTitle">Indique sus datos
personales</SPAN></TD></TR>
      <TR>
       <TD colSpan="2" align="center">
        <FIELDSET>
          <TABLE>
            <TR>
              <TD align="right" height="30px">
                <SPAN class="MediumText"> Alias (sólo lectura) </SPAN> </TD>
              <TD align="left">
                <INPUT name="nombre_corto" maxLength="20" size="20" readOnly onFocus="blur();"
value="<%=usuario.nombre_corto%>">
              </TD>
            </TR>
            <TR>
              <TD align="right" height="30px">
                <SPAN class="MediumText">* Contraseña Anterior </SPAN>
              </TD>
              <TD align="left">
                <INPUT type="password" name="old_password" maxLength="20" size="20">
                <INPUT type="hidden" name="pwd" value="<%=usuario.password%>">
              </TD>
            </TR>
            <TR>
              <TD align="right" height="30px">
                <SPAN class="MediumText"> Nueva Contraseña </SPAN>
              </TD>
              <TD align="left">
                <INPUT type="password" name="password" maxLength="20" size="20">
                <INPUT type="password" name="password_confirm" maxLength="20" size="20">
              </TD>
```

```
</TR>
<TR>
  <TD align="right" height="30px">
    <SPAN class="MediumText">* Nombre Completo </SPAN></TD>
  <TD align="left">
    <INPUT name="nombre_completo" maxLength="80" size="42" value="<%=usuario.nombre_completo%>">
  </TD>
</TR>
<TR>
  <TD align="right" height="30px">
    <SPAN class="MediumText">* Email </SPAN> </TD>
  <TD align="left">
    <INPUT name="email" maxLength="100" size="42" value="<%=usuario.email%>">
  </TD>
</TR>
<TR>
  <TD align="right" height="30px">
    <SPAN class="MediumText"> Titulo </SPAN> </TD>
  <TD align="left">
    <INPUT name="titulo" maxLength="100" size="42" value="<%=perfilusuario.titulo%>">
  </TD>
</TR>
<TR>
  <TD align="right" height="30px">
    <SPAN class="MediumText"> Compañia </SPAN> </TD>
  <TD align="left">
    <INPUT name="compania" maxLength="100" size="42" value="<%=perfilusuario.compania%>">
  </TD>
</TR>
<TR>
  <TD align="right" height="30px" rowSpan="2">
    <SPAN class="MediumText"> Dirección </SPAN> </TD>
  <TD align="left">
    <INPUT name="direccion1" maxLength="100" size="42" value="<%=perfilusuario.direccion1%>">
  </TD>
</TR>
<TR>
  <TD align="left">
    <INPUT name="direccion2" maxLength="100" size="42" value="<%=perfilusuario.direccion2%>">
  </TD>
</TR>
<TR>
  <TD align="right" height="30px">
    <SPAN class="MediumText"> Ciudad </SPAN> </TD>
  <TD align="left">
    <INPUT name="ciudad" maxLength="100" size="42" value="<%=perfilusuario.ciudad%>">
```

```
            </TD>
          </TR>
          <TR>
            <TD align="right" height="30px">
              <SPAN class="MediumText"> Estado </SPAN> </TD>
            <TD align="left">
              <INPUT name="estado" maxLength="100" size="42" value="<%=perfilusuario.estado%>">
            </TD>
          </TR>
          <TR>
            <TD align="right" height="30px">
              <SPAN class="MediumText"> C.P. </SPAN> </TD>
            <TD align="left">
              <INPUT name="cp" maxLength="20" size="5" value="<%=perfilusuario.cp%>">
            </TD>
          </TR>
          <TR>
            <TD align="right" height="30px">
              <SPAN class="MediumText"> Teléfono </SPAN> </TD>
            <TD align="left">
              <INPUT name="telefono" maxLength="40" size="15" value="<%=perfilusuario.telefono%>">
            </TD>
          </TR>
          <TR>
            <TD align="right" height="30px">
              <SPAN class="MediumText"> URI/URL </SPAN> </TD>
            <TD align="left">
              <TEXTAREA name="uri" cols="40" rows="5"><%=Server.HTMLEncode(perfilusuario.uri)%></TEXTAREA>
          </TD>
          </TR>
          <TR>
            <TD align="center" colSpan="2" height="70px">
<SCRIPT language="javascript">
<!--
writeButtonList('Aceptar,Deshacer',' ');
//-->
</SCRIPT>
            </TD>
          </TR>
        </TABLE>
      </FIELDSET>
    </TD>
  </TR>
</TABLE>
<INPUT type="Hidden" name="cmd" value="do-upd-profile">
<INPUT type="Hidden" name="perfilusuario_id_usuario" value="<%=perfilusuario.id_usuario%>">
```

```
    <INPUT type="Hidden" name="id_foro">
    <INPUT type="Hidden" name="token" value="<%=token%>">
  </FORM>
<!-- #INCLUDE FILE="bottom.asp" -->
</BODY>
</HTML>
```

B I B L I O G R A F Í A

Alur, Deepak; Crupi, John; Malks, Dan
Core J2EE Patterns: Best Practices and Design Strategies
Prenticel Hall
Junio 2001
ISBN: 0-13-064884-1

Bergsten, Hans
JavaServer Pages, 2nd Edition
O'Reilly
Agosto 2002
ISBN: 0-596-00317-X

Bortniker, Matthew; Conard, James; Miller, Frank
Professional Visual Basic 6 MTS Programming
Wrox
Junio 1999
ISBN: 1-861002-44-0

Hall, Marty
Core Servlets and JavaServer Pages (JSP)
Prenticel Hall
Mayo 2000
ISBN: 0-13-089340-4

Homerm, Alex; Sussman, David; Francis, Brian; et al
Professional Active Server Pages 3.0
Wrox
Octubre 1999
ISBN: 1-861002-61-0

Homerm, Alex; Sussman, David; Najjar Joe
Professional MTS & MSMQ Programming with VB and ASP
Wrox
Mayo 1998
ISBN: 1-861001-46-0

Hunter, Jason; Crawford William
Java Servlet Programming, 2nd Edition
O'Reilly
Abril 2001
ISBN: 0-596-00040-5

Larman, Craig
UML y Patrones, Introducción al análisis y diseño orientado a objetos
Prentice may
1999
ISBN: 970-17-0261-1

Lhotka, Rockford
Professional Visual Basic 6 Distributed Objects
Wrox
Junio 1999
ISBN: 1-861002-07-6

Lhotka, Rockford
Visual Basic 6.0 Business Objects
Wrox
Octubre 1998
ISBN: 1-861001-07-X

Li Sing; Palmer, Grant; Timney, John; et al
Professional JSP 2nd Edition
Wrox
Abril 2001
ISBN: 1-861004-95-8

Meyer, Bertrand
Construcción de Software Orientado a Objetos, Segunda Edición
Prentice Hall
1999
ISBN: 84-8322-040-7

Nakhimovsky, Alexander; Myers, Tom; Wilcox, Mark; et al
Professional Java Server Programming J2EE Edition
Wrox
Septiembre 2000
ISBN: 1-861004-65-6

Wilcox, Mark; Toussaint, Alex; Tyagi, Sameer; et al
Professional Java Server Programming J2EE, 1.3 Edition
Wrox
ISBN: 1-861005-37-7
Septiembre 2001