UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

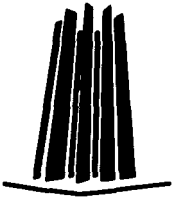
ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES ARAGON

"MODELO DE DESARROLLO PARA APLICACIONES TRANSACCIONALES SOBRE PLATAFORMA JAVA UTILIZANDO SOFTWARE LIBRE"

T E S I S

QUE PARA OBTENER EL TITULO DE:
INGENIERO EN COMPUTACIÓN
P R E S E N T A:

RAMIRO GONZÁLEZ SÁNCHEZ



ASESOR : ING. GLADIS E. FUENTES CHÁVEZ

MÉXICO, **2003**

A

PAGINACION

DISCONTINUA

UNIVERSIDAD NACIONAL AVENIMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES ARAGÓN

JEFATURA DE CARRERA DE INGENIERÍA EN COMPUTACIÓN

OFICIO:ENAR/JACO/0587/03.

ASUNTO: Asignación de Jurado.

LIC. ALBERTO IBARRA ROSAS
SECRETARIO ACADÉMICO
P r e s e n t e .

Por este conducto me permito presentar a usted el nombre de los profesores que sugiero integren el Sínodo del Examen Profesional del alumno **RAMIRO GONZÁLEZ SÁNCHEZ,** que presenta el tema de tesis "MODELO DE DESARROLLO PARA APLICACIONES TRANSACCIONALES SOBRE PLATAFORMA JAVA UTILIZANDO SOFTWARE LIBRE".

PRESIDENTE:	**ING. SILVIA VEGA MUYTOY**
VOCAL:	**M. EN C. JESÚS DÍAZ BARRIGA ARCEO**
SECRETARIO:	**ING. GLADIS E. FUENTES CHÁVEZ**
SUPLENTE :	**ING. CÉSAR FRANCISCO GERMAN ROSAS**
SUPLENTE:	**ING. VÍCTOR RAMÓN AGUILAR OCAMPO**

Quiero subrayar que el director de tesis es la **Ing. Gladis E. Fuentes Chávez,** el cual está incluido con base en lo que reza el reglamento de Exámenes Profesionales de esta Escuela.

Sin otro en particular, me es grato enviarle un cordial saludo.

A T E N T A M E N T E
"POR MI RAZA HABLARÁ EL ESPÍRITU"
San Juan de Aragón, Edo. de México, agosto 28 del 2003.
EL JEFE DE CARRERA

M. EN C. JESÚS DÍAZ BARRIGA ARCEO

c.c.p. Lic. Ma. Teresa Luna Sánchez.- Jefa del Departamento de Servicios Escolares.
Ing. Gladis E. Fuentes Chávez Asesor
Interesado.

JDA*vjd

Agradecimientos

A mi Madre Emiliana Sánchez Neri, que me enseñó que los límites están dentro de mi cabeza.

Al **Ing. Iker Vilá Pompeyo,** por enseñarme la importancia y la responsabilidad de los profesionistas con nuestra familia y nuestra comunidad.

Al **Arq. Carlos Zolezzi Féher,** por enseñarme a ser práctico.

Modelo de Desarrollo para Aplicaciones Transaccionales sobre Plataforma Java utilizando Software Libre

Ramiro González Sánchez

México, D.F., Octubre del 2002.

Contenido

Figuras

Introducción

TODA industria crece y se desarrolla gracias a un factor sin el cual no sería posible que subsista tecnología alguna: La estandarización. Para que los frutos de la industria telefónica o automotriz se volvieran artículos de uso común y tuvieran el sustento económico para su desarrollo fue necesario que compañías competidoras se aliaran para generar infraestructura y estándares básicos, con lo cual se abarataban costos y se ponía al alcance de más gente estos productos y servicios.

Algo similar ocurrió cuando surgió la industria de las microcomputadoras personales. Con la aparición de los clones Compaq de las primeras PC's IBM, se abrió la puerta para que otros fabricantes de clones produjeran sus propios equipos. Lo cual abarató costos, generando un círculo virtuoso. Ya que a mayor abaratamiento de los costos, se vendían un mayor número de unidades, con lo que se ampliaba el mercado, y se atraían nuevos productores, entre otras cosas. Todos estos equipos tenían en común que funcionaban de una manera similar a la PC IBM original, pero proporcionando a las aplicaciones de software una Plataforma idéntica de operación.

Desgraciadamente, esto que ocurrió en el mundo de las PC's, no es algo que ocurra en general en el mundo del hardware y del software. Muchos años han pasado de desarrollo en la industria y son múltiples los lenguajes, sistemas operativos, máquinas y periféricos que han existido, cada uno con sus bondades y sus desventajas. Sin embargo, cada vez se ha hecho más evidente que el carecer de estándares en la industria genera más problemas que soluciones.

Por esto cada día surgen nuevos estándares en la industria los cuales son apoyados por diversas compañías. Incluso son las mismas compañías las que se asocian para producir estándares que luego liberan para uso libre entre todos los demás participantes de la industria. Poco a poco se van estandarizando los equipos en algunos componentes (almacenamiento, comunicaciones, dispositivos de entrada-salida, etc.) y también en su operación lógica (sistemas operativos, protocolos, lenguajes de programación, etc.).

La estandarización dentro de la industria de la computación inició en el hardware. Es en el hardware donde resulta más evidente la necesidad de estandarizar componentes e interfases. Es así como surgieron estándares de monitores, de discos, teclados, memorias RAM y de fuentes de poder, sólo por mencionar algunos ejemplos. Un paso importante y necesario dentro de la estandarización de la industria es la del software. En este sentido ha ocurrido en los sistemas operativos al estandarizarse la operación del sistema operativo Unix, por ejemplo. Otro ejemplo es la estandarización de lenguajes de programación como el Lenguaje Ansi C. De forma reciente podemos citar el surgimiento del Lenguaje C++. Pero estos pasos iniciales aún son muy primitivos si tomamos en cuenta otras industrias, como la industria de la construcción, de las telecomunicaciones o la electrónica.

El sentido de esta estandarización es crear aplicaciones que sea posible ejecutar en cualquier tipo de equipo y bajo cualquier sistema operativo. De esta manera una empresa no estaría arraigada con la tecnología que compra en un determinado momento. Otra posibilidad es independizar a los usuarios aplicativos del soporte de una sola compañía. Incluso, los más

radicales, propugnan porque sea libre. Es decir, que el software se distribuya con el código fuente, de tal manera que el usuario sea capaz de modificarlo y adaptarlo a sus necesidades.

Una de las piezas fundamentales para lograr que las aplicaciones de negocio sean independientes de la plataforma es precisamente la Plataforma Aplicativa. Una Plataforma Aplicativa es un conjunto común de servicios los cuales debe proveer un sistema para que una aplicación de negocio funcione. Una de las plataformas que ofrece estos servicios es Java y está compuesta, a su vez, por dos plataformas las cuales se denominan: Java2 Standard Edition y Java2 Enterprise Edition, estas se integran con varios componentes y estándares los cuales proveen y definen servicios que una aplicación desarrollada para esta plataforma puede explotar para atender una función de negocio.

La ventaja de contar con una plataforma estándar es liberar a los desarrolladores y a las organizaciones de tecnologías propietarias o de tecnologías soportadas por una sola compañía que monopoliza el estándar. De esta manera una organización puede basar sus aplicaciones en una plataforma que pueda instalar sobre cualquier equipo que la soporte, es decir, las aplicaciones se vuelven independientes realmente del equipo en el que ejecutan. Además, esta plataforma especifica los servicios que se deben proveer pero no limita a las compañías en cuanto a la forma de implementarla, lo cual permite el desarrollo de múltiples productos que proveen el soporte de diversas maneras. Así se pueden tener múltiples proveedores a elegir, eliminando el monopolio y permite escoger la implementación más apropiada para los requerimientos específicos de cada organización.

Así es como nace la Plataforma Aplicativa Java (el Lenguaja Java tuvo un origen diferente), con la idea de independizar a las aplicaciones del hardware y sistema operativo sobre el que ejecutan. Dando también independencia a los desarrolladores y clientes aplicativos de los proveedores. Pero utilizar Java no está exento de problemas. Uno de ellos es que dada la flexibilidad y la cantidad de herramientas y opciones disponibles, es confuso entender cual es el camino a seguir para comprender la tecnología y posteriormente la mejor manera de aplicarla. Ya que la plataforma no especifica modelos de desarrollo o de diseño de aplicaciones.

Por otro lado, utilizar una Plataforma implica desde luego, instalar hardware y software, los cuales en conjunto proveerán los recursos computacionales para que las aplicaciones ahí instaladas operen. Generalmente, esto implica un costo alto. Piénsese por ejemplo, una plataforma aplicativa basada en COBOL-CICS-DB2, esta plataforma es de uso común en instituciones financieras de todo el mundo. Sin embargo, reproducir este ambiente en instituciones académicas es muy costoso. Por lo que los alumnos de la carrera de Ingeniería en Computación, verán este tipo de aplicaciones solo hasta que estén en sus centros de trabajo. Otro ejemplo podría ser la Plataforma Windows, ésta se caracteriza por el monopolio que sostiene Microsoft. Utilizarla para desarrollar aplicaciones implica por parte de las instituciones obtener licencias tanto del Sistema Operativo como de las herramientas de desarrollo, las cuales no están apegadas a estándar alguno que no sea propiedad de Microsoft.

La Plataforma Java, desde su creación, se definió como una plataforma "libre", es decir, que para su uso no se requiere el pago de licencias, he incluso el código de implementación de referencia está disponible para cualquier individuo o institución que desee utilizarla en el equipo deseado. Y aunque la Plataforma tiene derechos reservados para la empresa Sun Microsystems Inc., esta funge solo como empresa que custodia el estándar. Esto impide que otras compañías la modifiquen y a su vez impide que esto genere desapego al estándar, como ya ocurrió con un intento fallido por parte de Microsoft de generar su propuesta de lo que debía ser el entorno y estándar Java. El desapego al estándar iría en contra de lo que se quiere

lograr que es la estandarización para independizar al desarrollador y a los usuarios aplicativos de los proveedores.

Al mismo tiempo, la plataforma Java está contagiada por el espíritu del software libre, el cual está definido por la licencia GPL (GNU Public License), la cual hace el software de dominio público e impide la monopolización de estándares. Este movimiento abre una gran oportunidad para países como el nuestro, ya que pone en nuestras manos tecnología de una manera gratuita y nos da las herramientas para producir nuestro propio software sin depender de soportes externos ni costosos.

Todo esto requiere estudiar y comprender estas tecnologías para poder aprovecharlas de una manera adecuada. Y es en esta parte donde se hace necesario contar con guías y modelos que nos permitan incorporarnos rápidamente en el manejo de estas tecnologías. Esta tesis se propone como una guía para que estudiantes y académicos puedan tener acceso a una parte primordial del desarrollo de Aplicaciones que es la Plataforma Java.

Es posible con pocos recursos económicos adquirir o mejor aún, armar una PC la cual sirva para implementar software gratuito para instalar la plataforma Java. Esto con el fin de tener un "Laboratorio Virtual", donde se puedan desarrollar aplicaciones tal y como se generarían en ambientes reales y costosos como lo son los utilizados por empresas y organizaciones para soportar su producción. Este ambiente de desarrollo sería apropiado para que los estudiantes de la Universidad Nacional y otras instituciones adquieran y experimenten los conocimientos necesarios para producir las aplicaciones que se requieren hoy en día.

También se requiere, que junto con la infraestructura hardware y software, se definan caminos a seguir de capacitación para comprender la plataforma y metodologías de diseño y desarrollo los cuales se enfoquen en las necesidades básicas y que sean independientes del giro comercial o utilitario que tengan las aplicaciones finales. Esto con el fin de que el alumno tenga una amplia visión de la utilización de la plataforma Java y un método para acercarse a las diferentes variantes de implementación y utilización que se puede encontrar en la vida diaria de esta tecnología.

Además, los conocimientos adquiridos en el entendimiento y uso de la Plataforma Java, sirven también como fundamento para comprender otras plataformas disponibles para generar aplicaciones distribuidas multiusuario construidas en múltiples capas.

El objetivo de esta tesis es el de facilitar a Alumnos y Profesores de la carrera de Ingeniería en Computación el acceso a la Plataforma Java con pocos recursos, de tal manera que se practique dentro de la carrera el desarrollo de aplicaciones empresariales distribuidas. En esta tesis se abordará la Plataforma Java desde una perspectiva Aplicativa, con instrucciones claras y concisas de desarrollo de aplicaciones.

Los temas están divididos de la siguiente manera:

Capitulo 1: La Plataforma Java
Es una breve descripción de los componentes, servicios y estándares que componen la plataforma.

Capitulo 2: Hardware y Software requeridos
Describe el hardware mínimo y software recomendado para iniciar a desarrollar aplicaciones.

Capítulo 3: Mapa de Capacitación
Una vez que se cuenta con la infraestructura de software y hardware, es necesario capacitarse en la plataforma de acuerdo a la especialidad que se quiera alcanzar.

Capítulo 4: Estructura básica de Aplicación
Explica cual es la separación en componentes recomendada para construir una aplicación de negocios.

Capítulo 5: Modelos básicos de Programación
Se muestra como se aplican los conceptos del capítulo 4 para construir Aplicaciones comunes.

Capítulo 6: Metodología de Análisis y Diseño básica
Se dan las guías generales para realizar un Análisis y Diseño básico.

Capítulo 7: Escenarios aplicativos y productos comerciales
Se plantean algunas aplicaciones de la Plataforma Java2, así como un panorama de los principales productos y herramientas comerciales existentes en el mercado.

Capítulo 8: Aplicación Ejemplo
Se muestra una aplicación simple, construida sobre la base de los componentes básicos desarrollados en el capítulo 5.

Conclusiones
Se muestran cuales son las implicaciones del desarrollo y futura adopción de la Plataforma Java como estándar para desarrollo de aplicaciones. Se hace un resumen del ambiente de desarrollo recomendado.

Apéndices
Se incluye el código fuente completo de la aplicación ejemplo. Así como listas de recursos valiosos para profundizar en el tema.

Capítulo 1. La Plataforma Java

Las empresas modernas requieren de sistemas para su operación diaria. Estos sistemas pueden ser de apoyo a la operación como los sistemas de monitoreo de la producción; la operación misma del negocio, como en el caso de los sistemas que operan en cajeros automáticos; o de tipo administrativo: Recursos Humanos, Contabilidad, Gestión, etc. Todos estos sistemas tienen en común que son sistemas multiusuario y requieren generalmente una diversidad de interfases, es decir, diferentes medios para ofrecer las funciones a diferentes tipos de usuarios: Terminales remotas, terminales inteligentes, terminales tontas, Internet, Intranet, organizadores personales y otras que en un futuro no será raro utilizar como teléfonos celulares o intercomunicadores de dos vías.

Es conveniente dividir las aplicaciones de negocios en funciones operativas del negocio llamadas Transacciones. Un ejemplo típico de una transacción es la transferencia de saldo entre cuentas de un cliente de un banco. Esta transacción correspondería por ejemplo, a una aplicación de cheques y es multiusuario ya que simultáneamente varios clientes pueden solicitar una transferencia en un determinado momento. Además, los clientes podrían hacerlo a través de diferentes canales de acceso: vía telefónica, por cajero automático, en ventanilla de una sucursal bancaria o más recientemente por Internet.

De esta forma, es evidente que un negocio requiere de implementar una cantidad importante de Transacciones para operar. Y todas estas Transacciones tienen funcionalidad común, la cual no se programa para cada Transacción, sino que se programa una sola vez para que sobre esta funcionalidad básica se desarrollen las Transacciones específicas del negocio. Si ampliamos nuestro punto de vista, descubriremos también que sin importar el tipo de negocio, o el tipo de aplicaciones, todas las Transacciones de Negocio tienen funcionalidad en común:

- Atienden múltiples usuarios
- Requieren diferentes tipos de acceso
- Acceden a bases de datos
- Requieren restringir el acceso de diferentes maneras
- Realizan operaciones de consulta y actualización concurrentemente
- Requieren habilitar el acceso a la aplicación desde diferentes puntos geográficos

Toda esta funcionalidad, sería ociosa codificarla y reinventarla cada vez que se desarrolla una aplicación o aún más en cada empresa. La historia del software nos ha enseñado que es necesario construir piezas básicas y reutilizables, sobre las cuales se construya el siguiente nivel. Estas piezas cuando se desarrollan de una manera integral y sistemática para resolver una necesidad general de desarrollo de aplicaciones se conoce como Plataforma Aplicativa. Sin una plataforma aplicativa sería muy costoso implementar los sistemas que requieren los negocios modernos.

¿Qué es una Plataforma Aplicativa?

Una Plataforma Aplicativa es un conjunto de servicios estandarizados que permiten el desarrollo de aplicaciones de software. Estos servicios son:

- Acceder a Bases de Datos
- Manejo de múltiples sesiones
- Manejo de diversos tipos de acceso a los componentes aplicativos
- Control de la seguridad
- Administración de aplicaciones (horarios y disponibilidad)
- Interfases con Sistemas Existentes

Junto con otros servicios que puede incluir la Plataforma Aplicativa. Es sobre estos servicios que se construyen las Transacciones de que se componen las aplicaciones de negocio.

A lo largo de la historia de los sistemas han existido multitud de Plataformas Aplicativas, que sin darles ese nombre sirvieron en su momento para desarrollar aplicaciones de negocio. Así podemos hablar de la Plataforma Windows, la Plataforma Unix, la Plataforma Macintosh, la Plataforma AS/400, los Sistemas 36 y una larga lista de sistemas que han surgido a lo largo de la historia.

Hoy en día las aplicaciones de negocio se basan en 2 componentes principales: La Plataforma Aplicativa y el Manejador de Bases de Datos. Ambos definen la Arquitectura Aplicativa de un negocio y es cada vez más importante una correcta planeación estratégica. De la elección de la arquitectura aplicativa dependerá:

- Los medios de acceso que soportará el negocio
  - Servicios internos
  - Servicios vía Internet
  - Servicios por línea telefónica
- El número y la eficiencia de servicios de información
  - Consulta de información detallada
  - Consulta de información de resumen
  - Operaciones directas de modificación de datos

Desde este punto de vista, una compañía se puede quedar obsoleta en el momento en que su Arquitectura Aplicativa se vuelve obsoleta. Un ejemplo simple, es el caso de compañías que controlan su nómina de manera interna. Si no tienen un equipo conectado a Internet o por otro medio directo desde donde puedan conectarse a un banco entonces no podrán realizar una dispersión de saldos de manera automática. Lo cual implicará manejar efectivo en sus manos. Independientemente del riesgo, el costo de operar de esta manera, además de ser alto, ha impedido a muchas empresas crecer dado que administrativamente no pueden controlar ese tipo de crecimiento.

En el siguiente esquema se muestran los dos componentes principales de una Arquitectura Aplicativa. Dependiendo de ella dependerá la conectividad a sistemas antiguos y el tipo y cantidad de clientes que se pueden atender.

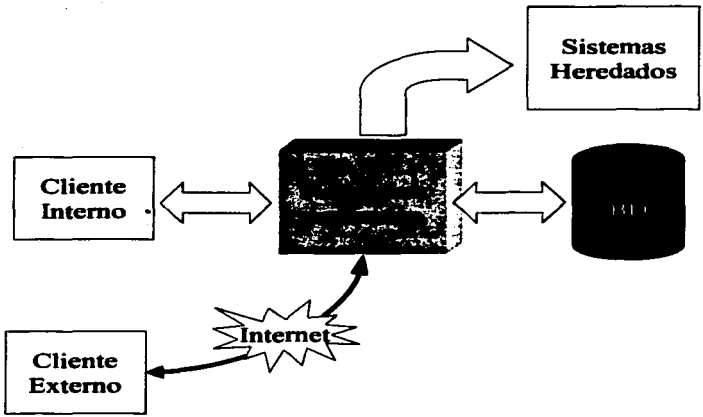


Figura 1 Componentes relacionados de una Plataforma Aplicativa.

¿Cuál es la diferencia entre el Sistema Operativo y la Plataforma Aplicativa?

El sistema operativo se encarga de administrar los recursos físicos y lógicos de una computadora como son: Disco Duro, Memoria RAM, Procesos, Colas de atención, Comunicaciones, etc. Una Plataforma Aplicativa es un conjunto de servicios que permiten desde un lenguaje de programación generar y controlar recursos orientados a un usuario como son: Pantallas, Reportes, Interfases, Servicios aplicativos, etc.

La confusión se genera cuando un Sistema Operativo es tan amplio que posee servicios aplicativos. Por ejemplo, Windows es un sistema operativo que además provee mecanismos a los lenguajes de programación para generar Ventanas de modo gráfico, accesar recursos como Impresoras a un alto nivel, siendo estos servicios aplicativos. Así pues, una aplicación escrita para ejecutar en Windows, dependerá de los servicios que le provee el sistema operativo y por lo tanto solo ejecutará bajo Windows.

Una Plataforma Aplicativa estaba intrínsecamente ligada al sistema operativo en el que ejecutaba. Es por eso que es posible hablar del sistema operativo Unix cuando nos referimos a una forma de gestionar los recursos de cómputo. Pero también es posible hablar de una Plataforma Unix, al referirnos a todo el conjunto de herramientas estándar para desarrollar aplicaciones de negocio bajo el ambiente Unix. Estas herramientas pueden ser el ambiente de ventanas Motif, utilerías de operación de archivos como "sort", "tar", etc. Dependiendo incluso del proveedor de Unix, se podían disponer de diferentes lenguajes y herramientas para construir aplicaciones.

Otra confusión menos común pero que también ocurre, es la de confundir la Base de Datos con una Plataforma Aplicativa. Y es por eso que escuchamos de aplicaciones basadas en Oracle o en SQL Server. Esto se debe también a que estas bases de datos rebasan sus funciones básicas y proveen de herramientas y servicios que corresponderían a una Plataforma Aplicativa. Por ejemplo, Oracle provee de su propio lenguaje interno para construir Stored Procedures los cuales sustituyen programas aplicativos. Otra herramienta que provee Oracle como compañía es el Developer, con el cual se pueden generar reportes y pantallas.

Sin embargo, en todo momento una compañía que decide implementar sus aplicaciones sobre un Sistema Operativo "muy poderoso" o sobre una Base de Datos "muy poderosa", debe cuestionarse si es la mejor inversión que puede hacer. Porque en ambos casos crea una dependencia con el hardware sobre el que pueda ejecutar un Sistema Operativo, o al tipo de aplicaciones que puede soportar una Base de Datos. En ambos casos, se vuelve dependiente de un solo proveedor de tecnología.

Cabe mencionar, que hasta hace pocos años las empresas no tenían elección. Ya que no existía una definición de Plataforma Aplicativa que fuese estándar y soportada por múltiples proveedores. La cual además pudiese ejecutar en varios tipos de hardware y sistemas operativos. Y además con posibilidades de interconexión con varias bases de datos.

Expectativas que debe cumplir una Plataforma Aplicativa

El día de hoy, las aplicaciones de negocio requieren además:

Productividad en la programación

La velocidad con que una empresa se adapta a los cambios dependerá de la velocidad con que pueda incorporar esos cambios en sus aplicaciones. Un cambio en la legislación, un nuevo producto o servicio en el mercado, un nuevo canal de acceso a sus servicios, una fusión de empresas, una reorganización interna de la empresa, cambios en la operación de un producto o servicios, entre otras causas provocarán inevitablemente modificar las aplicaciones de una empresa. El no poder atender dichos cambios de manera inmediata provocarán pérdidas a la empresa o incluso su salida del mercado. Implementar estos cambios dependerán de que la Plataforma Aplicativa elegida facilite la programación de aplicaciones y que existan personas con el perfil requerido para efectuar dichos cambios.

Respuesta oportuna a la demanda

Un crecimiento en el volumen de transacciones (época Navideña, época de Semana Santa), también puede afectar a las compañías de manera negativa. Es importante que los sistemas de una compañía también puedan crecer. Algunas veces este crecimiento se resuelve instalando más CPU's o más memoria. Pero en otras ocasiones es necesario un cambio completo de Arquitectura o de Equipo, por ejemplo, cuando es necesario cambiar de un sistema basado en PC's a un sistema centralizado en un equipo Unix. Obviamente este no es el único caso y ocurre a diferentes niveles dentro de las empresas. Un ejemplo reciente de esto fue el problema con las compañías de comunicación celular, al cambiar la legislación sobre la materia el número de usuarios se incrementó, elevándose también el número de llamadas simultáneas que debían atender las compañías. El ruteo de estas llamadas lo hacen equipos modernos de cómputo donde además de efectuarse el ruteo de la llamada se tiene que validar que el número telefónico en cuestión esté autorizado a realizarla y además debe contabilizar la duración de la llamada para actualizar la cuenta del cliente. Todo esto debe ocurrir en milésimas de segundo, de otra manera no sería práctico. Para que una compañía pueda soportar un crecimiento de este tipo sus aplicaciones deben ser fáciles de migrar a otro equipo. Evidentemente las compañías que hoy día basan su operación en ambientes de PC's en red y

con ambientes Windows tienen comprometido su crecimiento a solamente el que pueda soportar este ambiente.

Integración con sistemas existentes

Las empresas a lo largo del tiempo han acumulado en sus sistemas Reglas del Negocio e información de transacciones y clientes los cuales son valiosos. Convertir toda esta información o migrarla a otra base de datos es sumamente complejo y costoso. Por otro lado reconstruir las Reglas del Negocio en un nuevo lenguaje es una tarea aún más compleja y costosa. Por eso en ocasiones cuando un cliente necesita adoptar una nueva plataforma se busca que la nueva se pueda conectar y acceder a los servicios de los sistemas existentes. Esto permite por un lado adoptar nueva tecnología sin detener la operación, mientras se construyen los nuevos servicios.

Si la plataforma nueva no tiene opciones de conectividad con los sistemas anteriores, entonces se tendrá una situación donde la información se fragmenta. Esto provocará errores en la operación y en la atención de clientes. Imaginen un banco en donde no se pueda depositar un cheque de una cuenta de provincia a una cuenta de la ciudad de México, porque no hay interoperabilidad entre los sistemas, la calidad de servicio de dicho banco se demerita, sin embargo esto no es algo que se deba imaginar, ocurre en la realidad y es más frecuente de lo que se supone.

Libertad de elección

Todo producto software es desarrollado y soportado por una empresa o grupo de individuos. Es prácticamente imposible que un solo grupo o empresa pueda controlar y producir todas las posibilidades de conectividad, interacción y agregados que puede requerir una plataforma para operar. Una plataforma única y estándar permitiría que diferentes compañías proveyeran de servicios, componentes y herramientas para soportar las diferentes necesidades que surgen en el desarrollo de aplicaciones. Nuevamente quizás un ejemplo sea más claro: ¿Cuántos años tiene en el mercado Outlook de Microsoft? ¿Cuántos años más se necesitan para que esa aplicación pueda manejar correctamente nombres con 2 apellidos como se manejan en América Latina? Esta última respuesta depende de la división de Microsoft que desarrolle esta herramienta. NO depende de los usuarios. Obviamente nadie está obligado a usar Outlook. Pero en cierta forma sí, por varias razones, primera: Los usuarios están prácticamente obligados a comprar una computadora con Windows preinstalado, a la fecha en México no existen compañías que vendan computadoras personales sin un Sistema Operativo o que le devuelva al comprador dinero por un software que no quiere; segunda: Además el número de productos similares a Outlook depende del número de compañías que quieran pagar licencias de desarrollo a Microsoft, porque para desarrollar aplicaciones como Outlook que utilizan servicios internos del sistema operativo se requiere pagar por dicha información. No es información que se pueda bajar de un sitio en Internet. Es de esta manera que los usuarios de computadores personales en México no tienen alternativa, deben comprar una computadora y pagar forzosamente una licencia a Microsoft y esto es lo contrario de Libertad de elección.

Mantenimiento de la Seguridad

Para dar servicios verdaderamente distribuidos a usuarios finales se requiere utilizar medios de acceso remoto como líneas telefónicas o Internet. Es imprescindible que al momento de permitir el contacto con las aplicaciones de la empresa con el mundo exterior se cuide también quién tiene acceso y a qué. Esto debe ser algo que maneje la plataforma y que no requiera de modificar el código. Ya que desarrollar el código de cualquier aplicación es una tarea compleja y muy propensa a errores que frecuentemente se presta a menosprecio. Los programadores experimentados saben que una vez que un programa funciona nunca más hay

que volverlo a tocar. Entre menos se toca el código menos probabilidades hay de introducir errores en él. Por esto al mantener la seguridad al margen del código disminuyen las probabilidades de introducir errores. Al mismo tiempo no se pone en manos del programador la seguridad de un sistema. Son muy frecuentes los casos donde los programadores introducen en el código puertas traseras (*backdoors*) que ponen en riesgo la seguridad de las empresas. Estas puertas traseras son fragmentos del código que permiten acceder a funciones avanzadas del sistema y que solo los programadores conocen. El evitar que los programadores construyan estos segmentos de código, implica examinar línea por línea el código fuente por otro programador del mismo o mejor nivel que el programador desarrollador, lo cual lo hace inviable.

Necesidad de una Plataforma Aplicativa

Siempre existe una Plataforma Aplicativa. El problema es que no siempre es una decisión planeada. Ya que no existen aplicaciones que se construyan desde cero, accediendo directamente servicios del sistema operativo. Así que, sea esta una decisión planeada o no, siempre estamos utilizando una Arquitectura Aplicativa y como parte de ésta, la Plataforma Aplicativa. Lo importante es distinguirla para poderla aislar, controlar y en un momento dado cambiar.

Tómese como ejemplo la empresa donde se trabaja y responda:

a) ¿En qué equipo se ejecuta la nómina?

b) ¿En qué lenguaje o con qué paquete se desarrolló la aplicación?

c) ¿Qué tipo de usuarios accede esta aplicación?

d) ¿Con qué tipo de terminales o computadoras la acceden?

e) ¿En qué base de datos o sistema de archivos se almacena la información?

f) ¿Cómo es que está protegida esta información?

Todas estas respuestas están directamente relacionadas con la Arquitectura y la Plataforma Aplicativa. Ahora, la pregunta más importante es:

¿Quién tomó las decisiones para elegir los componentes de esta Arquitectura Aplicativa?

En muchos casos la respuesta es una de dos: Nadie ó El Vendedor de los componentes. Bajo esta perspectiva es lógico suponer que la empresa que tenga el mejor marketing es la que domina el mercado. Siendo estas decisiones tecnológicas y de ingeniería se vuelven decisiones subjetivas.

Descripción de la Plataforma Java2

La Plataforma Java2 surgió como una respuesta a las necesidades y problemas antes descritos. El problema actual que se pretende resolver se resume en un postulado: Convertir Internet de ser un medio de publicación de contenidos a un medio Transaccional. Es decir, convertirlo en un medio para acceder a aplicaciones transaccionales. Siendo este el principal enfoque de Java2 se ha diseñado de tal manera que sea su alcance el más amplio posible. Actualmente la versión de la Plataforma es la versión 2 y es la que se describe a continuación. El nombre oficial es Java2 Enterprise Edition, por razones de simplicidad la llamaremos Java. La Plataforma se compone de:

- Componentes,

- Contenedores y
- Servicios

Todo esto está soportado en Estándares. A continuación va una breve descripción de estos elementos.

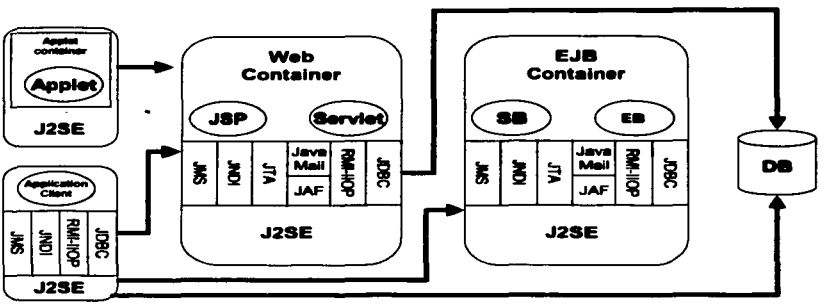


Figura 2 Componentes de la plataforma J2EE

Componentes

En programación Orientada a Objetos un componente es una pieza de código con una función completa. Este concepto generalmente se confunde con Objeto. Mientras un objeto puede ser un componente, un componente no necesariamente es un objeto. Un componente se diferencia de un objeto simple en que se apega a un protocolo o estándar para permitir su acoplamiento a otros componentes. Este protocolo puede permitir su invocación local, remota o ambas. Adicionalmente un componente puede estar compuesto internamente de varios objetos o incluso no estar escrito en un lenguaje orientado a objetos. Es por eso que el concepto de componente es más amplio que el de un objeto y es conveniente diferenciarlo.

Con los componentes se construyen las aplicaciones. En la plataforma Java, para construir componentes se utiliza el Lenguaje Java, pero esto no impide que se puedan comunicar con otros componentes escritos en otros lenguajes. Dependiendo del tipo de protocolo de comunicación existente los componentes pueden ser:

a) JavaBeans

Son objetos que siguen un estándar que define cómo se accesan y modifican los datos internos, y que proveen información acerca de su funcionamiento interno para facilitar su uso. El protocolo dinámico de comunicación soportado por estos componentes se le conoce como el mecanismo de Reflection y es la capacidad que tienen estos objetos de generar información a otros componentes acerca de sus propiedades internas: Métodos disponibles, variables internas, parámetros esperados, información del autor, entre otras. Estos componentes pueden ser objetos visibles o no, es decir, que pueden generar una salida visible por el usuario como un botón, una ventana, una tabla, o no generarla como por ejemplo un javabean que accede a

una tabla en una base de datos mediante SQL. Sean pues objetos visibles o no, están diseñados para ser utilizados desde herramientas visuales de programación. Esto es algo confuso, ya que la pregunta común es: ¿Cómo se puede utilizar un objeto no visible con una herramienta visual? La respuesta es, que las herramientas visuales de desarrollo permiten manipular gráficamente los javabeans de tal manera que permiten interconectarlos unos con otros, un ejemplo típico es cuando se conecta un javabean visible que despliega una tabla con un javabean no visible que ejecuta los postulados SQL para acceder a la base de datos. La conexión no se describe con código sino que se traza una línea de una caja que representa un javabean al otro. Otras características del estándar de los javabeans es que se establecen varias facetas del comportamiento de estos componentes lo cual permite su acoplamiento y trabajo coordinado con otros javabeans, como por ejemplo: Generan distintos eventos los cuales pueden ser "escuchados" por otros javabeans, puede tener un comportamiento o datos internos que son inicializables mediante propiedades definidas en un archivo y no mediante código, pueden escuchar eventos generados por otros javabeans y ejecutar alguna función predeterminada y pueden incluir información de cómo acceder a sus métodos internos la cual es útil para herramientas de desarrollo.

b) Applets

Son componentes que ejecutan dentro de un Browser para Web. Sirven para desarrollar interfases para Internet más dinámicas de lo que se puede hacer con HTML puro. Su protocolo de comunicación soportado es el protocolo HTTP que es el protocolo utilizado en internet para transmitir información en formato HTML ó XML.

c) Java Applications

Son programas que ejecutan en su propia Máquina Virtual de Java. Estos programas pueden estar en ambiente gráfico o en modo texto. Soportan varios protocolos de intercomunicación: CORBA, RMI, IIOP, HTTP, TCP/IP, UDP.

d) Java Server Page's

Son páginas Html que tienen incrustado código en Java para producir contenido dinámico. Es decir, son páginas que generan páginas Html dependiendo de la lógica programada en ellos. El protocolo soportado por estos componentes es HTTP.

e) Servlet's

Son componentes especializados en procesar peticiones de Web. Es decir, son utilizados para manejar peticiones que llegan al servidor aplicativo vía el protocolo HTTP. Al igual que las JSP's, estos también generan principalmente HTML como salida.

f) Enterprise JavaBeans

Son componentes que se utilizan para encapsular las reglas del negocio. Dicho de otra manera, son los que aplican las operaciones y validaciones sobre los datos que maneja la aplicación. Lo protocolos de comunicación que soportan son: CORBA y RMI-IIOP.

Contenedores

Los componentes Java J2EE, JSP's, Servlets y EJB's, no pueden ejecutarse por sí solos. Requieren de un contenedor que es software especializado en ejecutar un cierto tipo de componentes el cual a su vez ejecuta dentro de una máquina virtual java. En el caso de los Applications y de los Applets estos requieren de una máquina virtual de Java del lado del cliente para ejecutar, hablando siempre de aplicaciones distribuidas. Así tenemos los siguientes entornos de ejecución:

A) Máquina Virtual Java

También conocida como Runtime de java. Sirve para ejecutar Applications y Applets en los clientes. Los applets se diferencian de los applications en que son componentes que ejecutan en una máquina virtual Java integrada a un browser de web.

B) Contenedor Cliente

Es un modo especial de distribuir y ejecutar un Application o un Applet para ejecutar el componente dentro de un dominio de seguridad de un Servidor Aplicativo. Esto permite que un componente cliente accese todos los recursos que tenga permitidos dentro del dominio con un solo usuario y password.

C) Contenedor Web

Es un contenedor especializado en ejecutar páginas JSP y Servlets. Ejecuta normalmente en una máquina Servidora (*server*).

D) Contenedor EJB

Especializado en ejecutar componentes de Enterprise Java Beans. Ejecuta en una máquina servidora (server).

Servicios

Existen varios servicios con propósitos especializados:

JDBC

Conectividad a bases de datos.

RMI-IIOP

Protocolo de comunicaciones para accesar objetos Java remotos.

JavaMail

Interfaz para servidores de correo electrónico.

JTA

Servicios para coordinar transacciones con múltiples bases de datos y/o múltiples servidores aplicativos.

JNDI

Servicios para accesar servicios de directorios como Domain Name Servers (DNS) y servidores LDAP (utilizados principalmente para registrar los componentes EJB y ponerlos disponibles a las aplicaciones).

JMS

Servicio de mensajería instantánea entre aplicaciones.

Estándares

Los estándares que soportan todas estas definiciones están agrupados en el Java2 Enterprise Edition v. 1.3.1 Estos estándares son:

- Enterprise JavaBeansTM (EJB) 2.0
- Java™ Servlets 2.3

- JavaServer Pages™ (JSP) 1.2
- Java™ Messaging Service (JMS) 1.0.2
- J2EE™ Connector 1.0
- JDBC™ Standard Extension 2.0
- Java™ Transaction API (JTA) 1.0
- JavaMail™ 1.2
- Java™ API for XML Parsing (JAXP) 1.1

Lenguaje, Plataforma, SDK, JDK, Runtime y otros tecnicismos

El ambiente Java está lleno de múltiples definiciones que normalmente confunden cuando se está iniciando en la plataforma. La plataforma se implementa en 2 productos:

- Java2 Standard Edition: Es el producto orientado a ejecutar en las máquinas "Cliente". Existen de éste 2 versiones también: el SDK, que contiene la máquina virtual de Java, el compilador y la librería estándar de Java; y el Runtime que solamente contiene la máquina Virtual de Java. Ambas versiones también cuentan con un "plug in", el cual sirve para sustituir la Máquina Virtual que contienen los actuales Browsers con la versión más actual. Desde el punto de vista funcional el SDK es el paquete utilizado por los desarrolladores de aplicaciones, el runtime se instala en las máquinas de los usuarios y se utiliza para ejecutar las aplicaciones java.
- Java2 Enterprise Edition: Es el producto diseñado para ejecutar en los servidores de red y es el que contiene los Contenedores Web y EJB para alojar los componentes que atenderán a los diferentes tipos de Clientes. El J2EE (en su nombre abreviado) requiere del J2SE para funcionar. En la figura 2 mostrada antes se muestran todas estas piezas y su relación.

Entonces, se conoce como la Plataforma Java a todas estas piezas de software. El Lenguaje Java es un lenguaje de programación que forma parte de la Plataforma.

El JDK, Java Development Kit, es el nombre que antes se utilizaba para referirse a lo que ahora se conoce como SDK, Software Development Kit.

Existen otros componentes y estándares pero esos no se discutirán aquí:

Java ME: Java Micro Edition, que se utiliza para habilitar aplicaciones Java en componentes como teléfonos celulares, Organizadores Personales de bolsillo y otros.

Jini: Es un estándar para interconectar TODO tipo de aparatos: microondas, televisores, radios, de tal manera que algún día será factible interconectar todos los aparatos de una casa en una red casera y controlarlos todos con un solo protocolo.

Ventajas y desventajas de la Plataforma Java

Ventajas

Es una plataforma Libre. Es decir, se puede adquirir la implementación de referencia de manera gratuita, simplemente bajándola por Internet desde el sitio java.sun.com. No se recomienda utilizarla para ambientes de producción reales pero para eso existen implementaciones comerciales las cuales generalmente tienen costo. Existe al menos una que es gratuita y que produce la organización Jboss. Jboss es muy estable, madura y compite con las versiones comerciales.

El Lenguaje de Programación Java es más simple de aprender que C++.

El Lenguaje Java tiene una sintaxis similar a todos los lenguajes de la familia del Lenguaje C. Por lo que si se conoce alguno de estos lenguajes será muy fácil aprender Java.

Existen múltiples proveedores de herramientas y servidores aplicativos basados en Java y para Java.

Soporta diferentes tipos de aplicaciones: Aplicaciones StandAlone, Multiusuario, Multicapas, de Internet, para Intranets, en modo Gráfico, en modo Texto, con acceso a Bases de Datos o solamente con Archivos texto.

Desventajas

Actualmente no existe mucho soporte en México. Sin embargo, este cada vez va creciendo más.

Java es principalmente una plataforma de Integración. Es decir, que permite integrar diferentes y diversos componentes y tecnologías lo cual implica que es necesario conocer múltiples tecnologías para producir una aplicación. Precisamente el objetivo de esta Tesis es presentar cuales son los componentes básicos para producir una aplicación distribuida con acceso por Internet.

La Plataforma aún está en evolución. Esto ha provocado que en poco tiempo se hayan producido demasiados cambios lo cual ha dejado versiones de Java que ahora son obsoletas y ya no están soportadas. En el caso particular de México, esto también puede ser una ventaja. Ya que estamos en el momento de adoptar una Plataforma más madura y evolucionada que sus versiones anteriores y que hoy en día está muy cercana a su madurez y estabilidad ideal.

Resumen

La Plataforma Java se compone de:

Java2 Standard Edition: Versión para los clientes.

Java2 Enterprise Edition: Para las máquinas que actuarán como servidores de aplicaciones.

El **Lenguaje** Java es **parte de** la **Plataforma** y es el lenguaje con el que se codifican los componentes aplicativos. Hay varios estándares que componen la plataforma. Estos estándares definen cómo se deben construir los **Componentes** que se ejecutarán dentro de los **Contenedores** y los **Servicios** que tendrán disponibles para accesar a otros sistemas.

En el siguiente capítulo se explicará qué componentes Hardware y Software se necesitan para instalar un Laboratorio de desarrollo de aplicaciones basadas en Java2 mutiusuario, multicapas y con acceso a base de datos con SQL.

Capítulo 2: Hardware y Software requeridos

El objetivo es instalar un ambiente para el Desarrollo de Aplicaciones multiusuario accesibles por Internet. Otro objetivo es hacerlo con componentes de fácil acceso. Sin embargo, se cuida en todo momento que sea funcional.

Hardware

El equipo de mayor disponibilidad es una PC. La forma más económica para adquirir una es armando uno mismo su equipo. Esto requiere habilidades principalmente de configuración del Sistema Operativo. Los requerimientos de hardware dependen principalmente del Servidor aplicativo que se va a utilizar y de la herramienta de desarrollo elegida, ya que son estos componentes los que consumen más recursos. El equipo que se requiere como mínimo se detalla a continuación:

CPU y Motherboard

El CPU más potente y económico es el producido por AMD. El mínimo recomendado es: AMD Duron a 600 Mhz. El recomendado es el Thunderbyte de 1Gz. La elección del procesador dependerá principalmente del costo. Pero por menos de 100 dólares se puede conseguir un procesador suficiente para nuestros requerimientos. Como Motherboard debe elegirse uno que NO tenga todos los dispositivos integrados (audio, video, red). Estos son más caros pero a la larga se justifica con las posibilidades de crecimiento que proporcionan. Un Motherboard cuesta alrededor de 130 dólares.

Disco Duro

Se requieren como mínimo 2 GB para instalar el sistema operativo.

Además se requiere:

- 1 Gigabyte para instalar el Software
- 1 Gigabyte para datos

Por lo que en total necesitaremos 4 GB para instalar todo lo necesario. Actualmente existen discos IDE de 20GB que se pueden conseguir también por 100 dólares o menos.

Memoria

La memoria mínima requerida es de 256MB. La recomendada es de 512MB. Actualmente una memoria de 512MB se consigue por 70 dólares o menos.

Monitor

Un monitor de 17" pulgadas es el recomendado, esto permitirá abrir múltiples ventanas simultáneamente lo cual es común durante el desarrollo de aplicaciones. Un monitor cuesta hasta 300 dólares dependiendo de la calidad de la pantalla y la resolución de esta.

Unidad de CDROM

Existe una amplia variedad de CDROM's en el Mercado. Dado los volúmenes del software es más conveniente manejarlo en CD's que en disquetes. Lo ideal es conseguir un grabador de CD's. Existen en un rango de precios que van desde 60 dólares hasta 180 los más veloces.

Impresora Opcional

Es muy práctico tener una disponible, principalmente para imprimir instrucciones de instalación y fragmentos de código o diagramas.

Red Opcional

Para realizar pruebas apegadas al entorno real, será necesario probar ejecutando remotamente la aplicación. Si no se tiene una red, de cualquier forma se puede instalar en casi todos los sistemas operativos un dispositivo que se conoce como loopback, el cual simula una conexión a red.

Un equipo completo con las características antes descritas cuesta como máximo 1000 dólares.

Software

Todo el software utilizado para este equipo será gratuito (o casi gratuito). Aunque se requiere acceso a Internet de banda ancha o tiempo disponible para conectarse varias horas para obtener una copia del software aquí descrito. Una vez que se consigue una copia esta se puede reproducir tantas veces sea necesario, lo cual es una ventaja adicional del software libre.

Sistema Operativo

El sistema operativo gratuito de mayor uso es Linux. Existen varias distribuciones de este sistema operativo. La recomendada para este Laboratorio es la distribución conocida como Mandrake Linux 8.2 ó superior. Esta versión es una distribución basada en Red Hat, pero más fácil de utilizar y configurar. Contiene una amplia selección de software y todos sus componentes vienen integrados para facilitar su operación. Todos los ejercicios aquí descritos se construirán utilizando este sistema operativo. Se puede conseguir en www.linux-mandrake.com.

Java2 Standard Edition

La versión actual disponible para Linux es la v. 1.4.1, esta se consigue en java.sun.com. También debe descargarse la documentación la cual viene en un paquete diferente.

Java2 Enterprise Edition

La versión actual disponible para Linux es la v. 1.3.1, se consigue también en java.sun.com. También debe descargarse la documentación la cual viene en un paquete diferente. Este paquete se utilizará exclusivamente para el desarrollo inicial. Una vez que se comprendan los detalles básicos de desarrollo y empaquetamiento de aplicaciones es importante avanzar hacia un contenedor de producción como es Jboss 3.0. Este último se puede descargar de www.jboss.org y es la versión recomendada para implementación de aplicaciones reales.

Ant

La versión actual es la v. 1.4.1. se consigue en jakarta.apache.org. Este componente es opcional, pero una vez que se utiliza no se deja de usar. Se utiliza para automatizar las tareas de compilación y empaquetamiento de las aplicaciones.

Struts

La versión actual estable es la v. 1.0 se consigue en jakarta.apache.org. Este es un framework de desarrollo. Se utiliza para implementar un patrón de diseño conocido como MVC, por sus siglas en inglés: Model-View-Controller. Hablaremos más de este patrón de diseño en el capítulo 4. Estructura Básica de Aplicación.

Herramienta de Desarrollo

La herramienta gratuita recomendada es Forte for Java 4 Community Edition, actualmente sufre de un cambio de nombre, por lo que se encontrará bajo el nombre de Sun ONE Studio 4. Se puede obtener en http://wwws.sun.com/software/sundev/jde/index.html.También debe descargarse la documentación la cual viene en diversos documentos PDF. Esta herramienta es la que utilizaremos para editar los programas Java, compilarlos, depurarlos y otras funciones relacionados con el desarrollo de las aplicaciones. El nombre en inglés de una herramienta como esta es Integrated Development Enviroment (IDE), dado que permite integrar en una sola interfaz diferentes herramientas de desarrollo.

Browser de Internet

El browser recomendado es Netscape Navigator v. 6.2. Disponible para Linux. Se consigue en www.netscape.com. Aunque Linux Mandrake 8.2 incluye Mozilla 0.9.8 el cual es la versión en la que está basado Netscape. Así que para ahorrar una descarga de 11 MB aproximadamente, se puede utilizar este. Para obtener un browser completo con las últimas actualizaciones se recomienda descargar e instalar Mozilla 1.1, el cual se puede obtener del sitio www.mozilla.org.

Base de Datos

Mandrake incluye 2 de las bases de datos más populares en Linux: MySql y PostgreSQL. Ambas se pueden configurar para ser utilizadas con Java. Sin embargo, para evitar complejidad en la configuración, para desarrollo se utilizará la base de datos que viene preconfigurada en el J2SDK EE de sun que es Cloudscpae. Una vez dados los primeros pasos en el desarrollo de aplicaciones se puede migrar a una configuración más robusta utilizando Jboss 3.0 y PostgreSQL. Para esta tesis se eligió PostgreSQL debido principalmente a sus mayores capacidades de manejo de datos, ya que posee una arquitectura para el manejo de datos superior, lo cual lo a hecho popular como el "Oracle gratuito". Además de que posee el mejor soporte para SQL dentro del mercado de bases de datos gratuitas.

Acrobat Reader

Indispensable para leer la documentación del sistema operativo o de Forte for Java. Mandrake incluye un "clon" gratuito llamado xpdf. Pero es mejor bajar la versión oficial de www.adobe.com.

Instalación

Armar una PC podría ser otra Tesis. En Internet existen instrucciones y referencias de cómo hacerlo. También en revistas especializadas en PC's es frecuente encontrar artículos describiendo el proceso de armado de una PC. También en la ciudad de méxico existen lugares donde al comprar todos los componentes básicos en el mismo lugar te arman la máquina gratuitamente. Por lo que aquí no daré detalles de esto. Lo que se pretende es instalar todo el software necesario para el desarrollo de aplicaciones para Web. El siguiente esquema muestra las relaciones entre cada componente de una instalación típica para desarrollo básico:

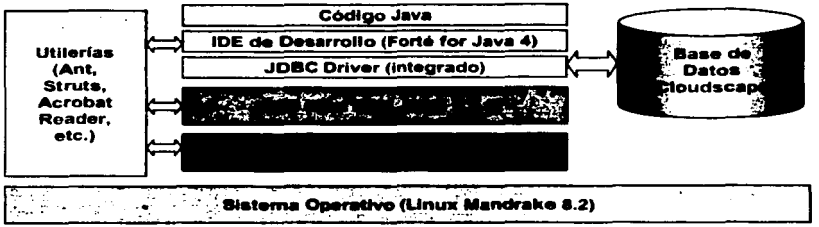


Figura 3 Configuración de J2EE con el software de Referencia.

Una configuración más sofisticada sería la siguiente:

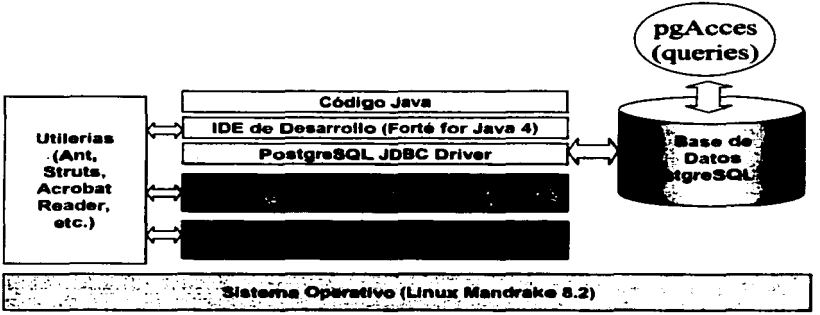


Figura 4 Configuración recomendada para J2EE con software libre.

La diferencia entre estas dos configuraciones es la curva de aprendizaje. Para el caso básico sólo hay que aprender a utilizar el J2SDK EE de sun, el cual ya trae integrada y configurada una base de datos, además de que incluye una interfaz gráfica para efectuar el empaquetamiento e instalación de aplicaciones. En el segundo caso, se debería aprender Jboss el cual es un contenedor muy poderoso pero por lo mismo más complejo de aprender a usar, por otro lado se requieren forzosamente conocimientos de la herramienta Ant para simplificar el proceso de empaquetamiento e instalación de aplicaciones. Así como también, la conexión a la base de datos que no es trivial para el caso de Jboss y PostgreSQL.

En cuanto al software, cada producto referido viene acompañado de instrucciones precisas de cómo lograr su instalación y configuración, excepto para Jboss. Por lo que no detallaré el

procedimiento. Solamente se incluye una guía rápida de instalación para impacientes. El único requisito es conocer muy bien el manejo de particiones de disco duro en PC y los conocimientos básicos de Linux. Por lo que antes de comenzar lo mejor es conseguir, también en Internet, el Manual de Usuario de Linux Mandrake 8.2. La dirección es: http://www.mandrakelinux.com/es/fdoc.php3.

Pasos de Instalación

1. Decidir el particionamiento del disco duro. Las particiones del disco duro recomendadas son:

"raíz": 1 Gigabyte

/home: 1 Gigabyte

/usr: 1 Gigabyte

swap: 1 Gigabyte

2. Instalar Mandrake Linux 8.2

Asegurarse de instalar los grupos de paquetes: "Development Tools" y "Databases".

Desinstalar el paquete kaffe, con el usuario de root:

```
rpm -e --nodeps kaffe
```

Configurar Mandrake Linux al estilo particular. Si nunca se ha usado Linux, leer la guía de usuario que viene en el producto. Leer principalmente las partes relacionadas a instalación de software.

3. Instalar SDK v. 1.4

Descargar de java.sun.com el archivo: j2sdk-1_4_0_01-linux-i386-rpm.bin

Se instala con el usuario de root, con el comando:

```
urpmi j2sdk-1_4_0_01-fcs-linux-i386.rpm
```

Editar /etc/profile y agregar

```
JAVA_HOME=/usr/java/j2sdk1.4.0_01
If [ "$UID" -ge 500] && ! echo ${PATH} | grep -q $JAVA_HOME/bin ; then
  Export PATH=$PATH:$JAVA_HOME/bin
Fi
```

4. Instalar J2EE Sdk v. 1.3.1

Se descarga de java.sun.com el archivo: j2sdkee-1.3.1-linux.tar.gz y se "desempaca", de preferencia en el directorio de home (me refiero aquí al directorio de inicio del usuario utilizado), con el siguiente comando:

```
tar xvzf j2sdkee-1.3.1-linux.tar.gz
```

Esto generará el subdirectorio j2sdkee1.3.1/

Dentro de este subdirectorio está el subdirectorio bin, que es donde están los scripts que arrancan y detienen el servidor aplicativo y la base de datos Cloudscape que viene preconfigurada para accesarla desde las aplicaciones Java.

También es necesario agregar la variable de entorno J2EE_HOME apuntando al directorio anteriormente mencionado. Para realizar esto se edita el archivo .bash_Profile (nótese el punto inicial en el nombre del archivo), el cual se encuentra en el directorio home del usuario. Ejemplo:

```
[ramiro]$ vi .bash_Profile
export J2EE_HOME=~/j2sdkee1.3.1
```

Esta línea y el nombre del archivo a modificar varían dependiendo del shell de Linux que se esté utilizando. Estas instrucciones corresponden al shell Bash, que es el utilizado por omisión en Linux Mandrake. Para que este cambio se aplique es necesario terminar la sesión de Linux e iniciar de nuevo.

Nótese también, que esta instalación del J2EE SDK se hizo en el subdirectorio home de un usuario personal, ya que no se recomienda utilizar la misma instalación para varios usuarios. Lo mejor es que cada desarrollador tenga su propio ambiente. Lo que sí se puede compartir es la instalación de Java y la de la base de datos, si es que no se utiliza la base de datos Cloudscape que viene preconfigurada con el J2EE SDK.

Para probar la instalación se puede ejecutar desde el directorio j2sdkee1.3.1/bin el comando:

```
./j2ee -verbose
```

esto arrancará el Servidor Aplicativo y se quedará ejecutando en la ventana desde donde se ejecutó el comando. Luego se abre una ventana de cualquier browser de web y se conecta uno a la dirección http://localhost:8000

Aparecerá la siguiente ventana:

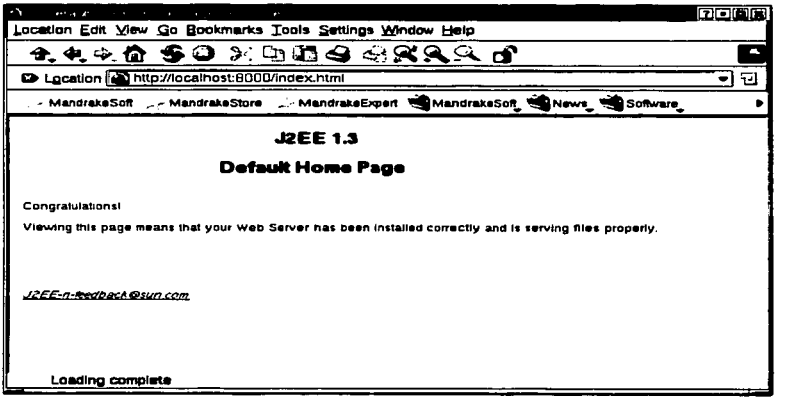


Figura 5 Pantalla principal de la implementación de referencia de J2EE 1.3

Para detener el servidor aplicativo se abre otra ventana de Shell y se ejecuta el comando:

```
./j2ee -stop
```

Linux Mandrake, incluye varios Web Browser, los cuales instala por omisión: Mozilla, Konqueror (KDE), Galeon (Gnome), Opera y otros. Esto es importante, porque las aplicaciones construidas para Internet es conveniente probarlas con varios Web browsers para verificar la correcta ejecución en diferentes clientes. Los browser no incluidos son: Netscape y Microsoft Explorer. El MS Explorer sólo ejecuta en Windows, así que para utilizarlo será necesario tener en red con nuestra computadora de desarrollo alguna máquina con Windows. En el caso de Netscape, las versiones más recientes, a partir de 6.0, están basadas en Mozilla, por lo que no es necesario instalar Netscape, a menos que se quiera probar con las versiones antiguas, las cuales existen para Linux en el sitio www.netscape.com e instalan sin problema en Linux Mandrake. Generalmente si una aplicación funciona con Netscape, funcionará con MS Explorer, dado que Netscape es más estricto en la interpretación de las páginas en HTML. Lo contrario no sucede. Pero de cualquier forma se recomienda revisar una aplicación en MS Explorer, ya que la presentación en uno y otro browser suele diferir.

5. Instalación de PostgreSQL v. 7.2

Linux Mandrake 8.2 incluye PostgreSQL. Para instalarlo basta con seleccionar durante la instalación la opción de Database Server. También se instalará MySQL que es otra base de datos muy popular. Se eligió para esta Tesis PostgreSQL ya que es la base de datos con el soporte más amplio del lenguaje SQL y por lo tanto se acerca más al SQL manejado por las bases de datos comerciales como Oracle, DB2 o SQL Server. Aunque generalmente para el uso aplicativo de SQL, solo se requieren construcciones básicas disponibles en cualquier base de datos, es mejor asegurar desde un inicio la mayor compatibilidad posible para poder implementar en cualquier momento extensiones a las aplicaciones para procesos batch o con otras herramientas auxiliares, por ejemplo, para generar respaldos de información, transferir información entre bases de datos, etc.

Si durante la instalación de Linux Mandrake no se seleccionó la opción para instalar las bases de datos, se puede agregar PostgreSQL instalando los siguientes paquetes:

```
postgresql-server-7.2-12mdk
postgresql-7.2-12mdk
postgresql-jdbc-7.2-12mdk
```

También es recomendable instalar los paquetes:

```
postgresql-docs-7.2-12mdk
postgresql-tk-7.2-12mdk
```

Los paquetes rpm incluidos en los CD's de Mandrake se instalan con el programa del menú: Configuration->Packaging->Software Manager.

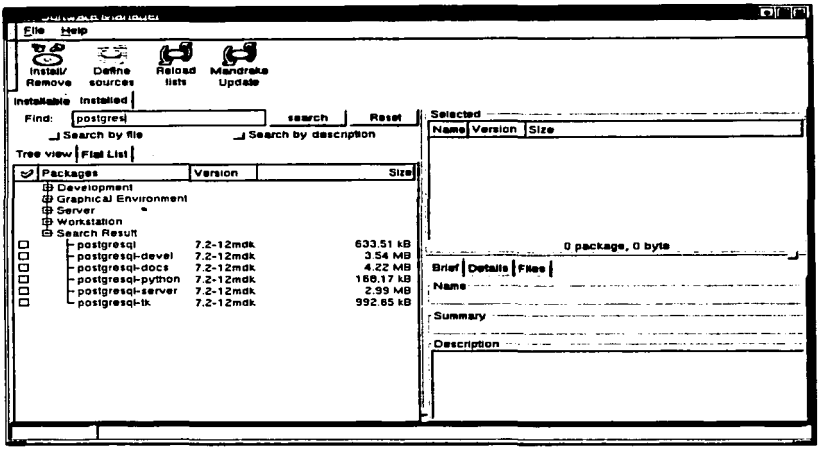


Figura 6 Software Manager de Mandrake mostrando los paquetes de postgreSQL 7.2

Una vez instalado el software, hay que crear una base de datos de trabajo. Esta base de datos puede ser por usuario o una base de datos común para un equipo de desarrollo.

Para simplificar, optaremos por configurar una sola base de datos para un solo usuario. Lo ideal es que cada programador tenga una base de datos para trabajar. Cuando se trabaja en equipo se instalan además otras 2 bases de datos: Desarrollo y Preproducción. Las bases de datos por usuario se utilizan para pruebas unitarias. El ambiente de Desarrollo se utiliza para pruebas integrales y de sistema. El ambiente de preproducción debe ser una replica del ambiente de Producción y se le utiliza para pruebas de Volumen y pruebas de Aceptación.

Para crear la base de datos de trabajo se ejecutan los siguientes pasos:

- Se conecta uno con el usuario de root, desde una ventana de shell con el comando su.
- Después se cambia al usuario postgres. Este usuario es el propietario del software PostgreSQL 7.2, en la instalación por omisión de Linux Mandrake.
- El siguiente paso es crear un usuario con el comando createuser
- Por último con el nuevo usuario se crea una base de datos.

La secuencia de comandos es:

```
[ramiro@dell ramiro] su
(se pide la password de root)
[root@dell root] su - postgres
```

```
bash-2.05$ createuser
bash-2.05$ exit
[root@dell root] exit
[ramiro@dell ramiro] createdb
```

Con estos comandos se habrá creado una base de datos con el nombre del usuario. En este caso: ramiro. PostgreSQL organiza las bases de datos en "Clusters", que es un subdirectorio en donde se guardan la información de las bases de datos.

El Cluster en Linux Mandrake está configurado en: /var/lib/pgsql.

Ahora hay que asignarle passwords a los usuarios de postgres y ramiro (o el particular que se esté utilizando), dentro de la base de datos. Esto es necesario debido a que para poder conectarse a la base de datos necesitamos habilitar la conexión vía el protocolo TCP/IP. Los pasos son los siguientes:

- Se abre un shell con el usuario postgres (su y luego su – postgres).
- Se ejecuta el comando: `psql template1`
- Se ejecuta el comando:
  `Alter user postgres with encrypted password 'ramiro';`
- Luego se repite para "ramiro":
  `Alter user ramiro with encrypted password 'rgs';`
- Se termina la sesión de psql con el comando: \q
- Se edita el archivo /var/lib/pgsql/data/pg_hba.conf
- Se inserta al final del archivo la línea:
  `host all 0.0.0.0 0.0.0.0 md5`
  esta línea le indica a PostgreSQL que todos los usuarios que se conecten a cualquier base de datos vía TCP/IP se les solicite su password encriptado.
- Luego se edita el archivo /var/lib/pgsql/data/postgresql.conf y se elimina el símbolo de comentario (#) en las siguientes líneas:
  `tcpip_socket=true`
  `port=5432`
- Se reinicia postgresql con root: `service postgresql restart`

Ahora será posible arrancar una conexión a la base de datos con la utilería gráfica: pgacces: Seleccione en el menú: Applications->Database->PostgreSQL Acces

En el menú se selecciona Database->Open, se dan los parámetros de base de datos, usuario y password. El programa se conectará pero seguirá la pantalla en blanco, porque no hemos creado ninguna tabla de momento. Pero se pueden ver las tablas del sistema, seleccionando ahora en el menú: Database->Preferences->View system tables.

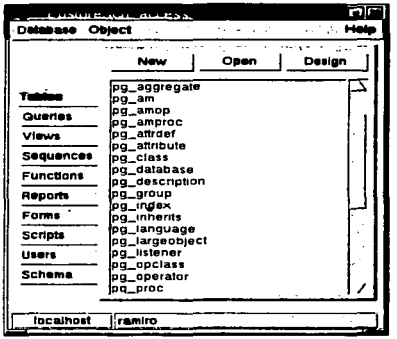


Figura 7 pgAccess mostrando las tablas del sistema de una base de datos PostgreSQL en linux.

6. Configuración de la conexión JDBC

El servidor aplicativo J2EE para poder abrir conexiones a la base de datos desde los componentes Java, requiere que se registren los drivers de conexión para cada proveedor de base de datos. Es decir, que dependiendo del tipo de base de datos a la que se requiera accesar desde un componente J2EE, necesitamos previamente configurar el driver JDBC correspondiente. En el caso de PostgreSQL, este driver se encuentra en: /usr/share/pgsql . El nombre del paquete que contiene el driver es: jdbc7.2dev-1.2.jar. Este driver está todavía en fase de desarrollo, por lo que no será posible utilizarlo para generar Entity Beans con Container Managed Persistence (CMP 2.0), para ser utilizado con el J2SDK EE de sun, pero es posible utilizarlo para generar objetos que acceden con postulados java directos a la base de datos mediante JDBC. Lo mejor es estar constantemente monitoreando en el sitio jdbc.postgresql.com para obtener una versión actualizada. Al momento de escribir esto existe ya una versión nueva llamada: devpgjdbc2.jar

Los pasos de configuración del driver para JDBC se encuentran el Configuration Guide del J2EE.

Brevemente los pasos son los siguientes:

```
cp /usr/share/pgsql/jdbc7.2dev-1.2.jar $J2EE_HOME/lib/system
cd $J2EE_HOME/bin
vi userconfig.sh
```

En este archivo se insertan las líneas:

```
J2EE_CLASSPATH=/usr/share/pgsql/jdbc7.2dev-1.2.jar
export J2EE_CLASSPATH
```

Se salva el archivo y luego se ejecuta:

```
./j2eeadmin -addJdbcDriver org.postgresql.Driver
```

```
./j2eeadmin -addJdbcDatasource jdbc/PostgreSQL
jdbc:postgresql://localhost:5432/ramiro
```

En estos ejemplos se tomó el driver que viene en la distribución de Mandrake, pero si se descarga una versión actualizada del sitio jdbc.postgresql.com, se deben repetir los pasos modificando respectivamente en cada comando el nombre del nuevo driver.

7. Instalación de Forte for Java 4 Community Edition

En el sitio de sun http://wwws.sun.com/software/sundev/jde/index.html, se pueden encontrar las ligas para descargar la última versión disponible de este producto. Es una descarga de 37.91MB que con un modem de 56kb toma aproximadamente 2 horas y cuarto. El archivo una vez descargado en nuestra máquina para Linux se llama: ffj-ce-linux-en.bin

La documentación completa se encuentra en este mismo sitio. La liga directa es: http://wwws.sun.com/software/sundev/jde/documentation/index.html

El archivo descargado se instala ejecutando el archivo de esta manera:

```
./ffj_ce_linux-en.bin
```

Una vez instalado se arranca con el comando: ./runide.sh

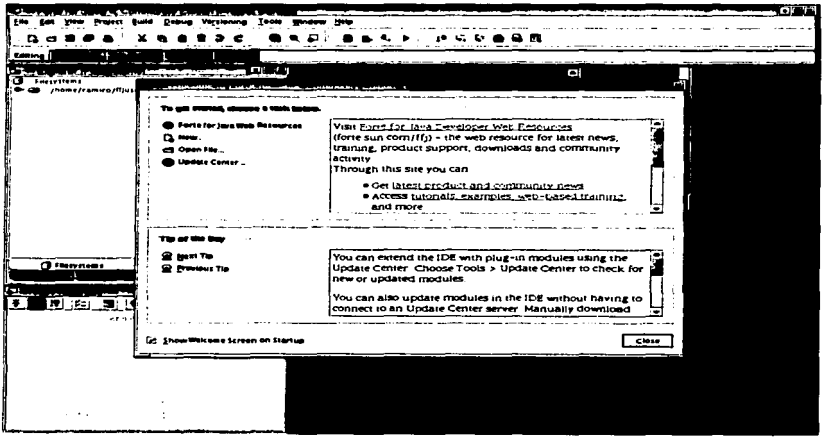


Figura 8 Pantalla del wizard de primera ejecución de Forté for Java 4.

8. Configurar el acceso a PostgreSQL desde Forte

Se copia el driver de PostgreSQL al directorio lib/ext de Forte. En mi caso:

```
cp /usr/share/pgsql/devpgjdbc2.jar /opt/forte4j/lib/ext
```

Se reinicia Forte: `/opt/forte4j/bin/runide.sh`

y luego se selecciona en la ventana de Explorer->Runtime ->Databases->Drivers

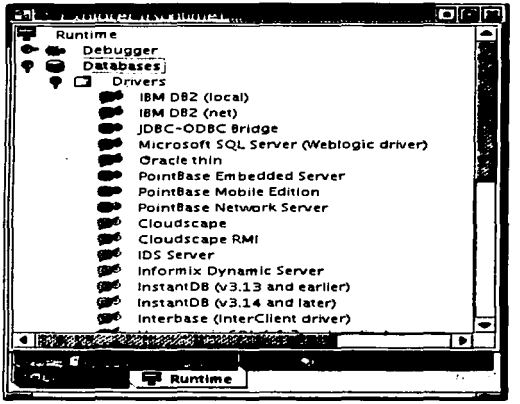


Figura 9 Ventana del explorer de Forté for Java 4 mostrando los drivers JDBC configurados y disponibles.

Aquí aparece la lista de bases de datos que están preconfiguradas. Descendiendo en la lista debe aparecer "PostgreSQL (v. 7.0 and later)". Sobre esta línea se presiona botón derecho y se selecciona "Connect using..." y se llena la pantalla de la siguiente manera:

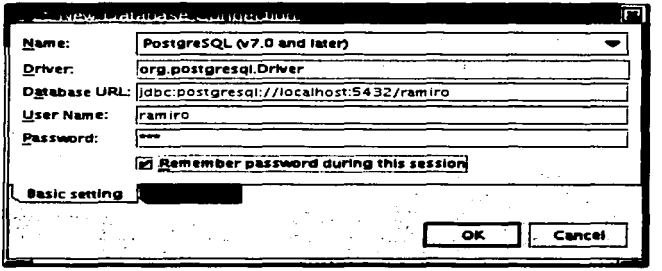


Figura 10 Configuración del driver jdbc para PostgreSQL.

Se presiona Ok, y en la parte inferior del Explorer aparece la nueva conexión. Solamente hay que abrirla para probarla:

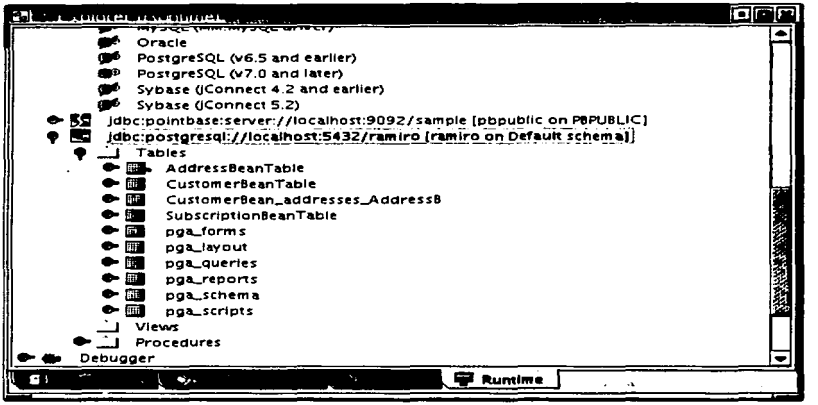


Figura 11 Tablas accedidas por el driver jdbc en el Explorer de Forté for Java 4.

El soporte del Driver no está completo. Un ejemplo de esto es que para las tablas no aparecen los nombres de las columnas. Sin embargo al ejecutar un query de SQL, con botón derecho "Execute command" sobre la conexión, se ejecuta correctamente:

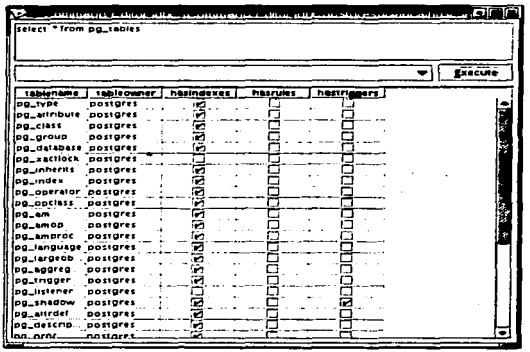


Figura 12 Ejemplo de ejecución de un query desde Forté for Java 4.

9. Instalar Ant

Ant es una herramienta para automatizar la recompilación de aplicaciones construidas en Java. Dependiendo del proyecto es posible que se utilice. Forte for Java incluye una versión de Ant "integrada". Además tiene la facilidad de generar los scripts para recompilar un proyecto desde fuera de la herramienta utilizando solamente Ant. Se recomienda su instalación principalmente para poder seguir el tutorial de J2EE del que se hablará más adelante.

Ant se descarga desde el sitio http://jakarta.apache.org/ant. El archivo es: jakarta-ant-1.4.1-bin.tar.gz.

Este archivo simplemente se descompacta en algún lugar común a todos los usuarios, por ejemplo: /opt.

Luego se inserta la variable de entorno ANT_HOME apuntando al directorio donde se instaló. Por ejemplo: ANT_HOME=/opt/jakarta-ant-1.4.1

También hay que agregar al PATH la ruta $ANT_HOME/bin para que la puedan utilizar todos los usuarios. Estos cambios de variables de entorno, se recomienda hacerlos sobre el archivo /etc/profile para que los cambios los reciban todos los usuarios del sistema.

10. Instalar Struts

Esta es una herramienta de control de flujo de pantallas en las aplicaciones Web. No es propiamente software. Es más bien un Framework de Desarrollo. Para su instalación se descarga el archivo jakarta-struts-1.0.2.tar.gz del sitio: http://jakarta.struts.org/struts y se

"desempaca" en un directorio de acceso común a todos, por ejemplo /opt. Se le utilizará para construir aplicaciones más adelante.

11. Instalar Jboss

Para una fase más madura de desarrollo es recomendable utilizar un contenedor de J2EE completo y robusto como lo es Jboss. Por simplificidad en esta tesis se utilizará el contenedor de referencia de Sun, pero una vez que se aprende lo básico hay que dar el salto a Jboss. Este salto es fácil darlo si primero se aprende a utilizar Ant y a construir scripts de Ant para simplificar las tareas de compilación, empaquetamiento e instalación de las aplicaciones.

Resumen

El software libre actualmente está muy maduro y es fácil de usar. El software aquí referido se puede descargar de Internet o mediante copias en CD lo cual es legal en el caso del software aquí referido. Cada producto se debe instalar y configurar de acuerdo a las instrucciones contenidas en cada producto. En el inicio de este capítulo se mencionaba que el software es "casi gratuito". El costo está en el hecho de que todo este software requiere una lectura más cuidadosa de los manuales y de las instrucciones de instalación y configuración. Algo a lo que comúnmente no estamos acostumbrados a hacer es precisamente a leer los manuales. Sin embargo, como Ingenieros en Computación estamos obligados a hacerlo. Ya que las plataformas de desarrollo de la vida real son igualmente complejas como todos los productos aquí mencionados.

Capítulo 3. Mapa de capacitación

Roles de Desarrollo

Existen diferentes metodologías de desarrollo. Pero en lo esencial existen roles de desarrollo, los cuales consisten de perfiles especializados en una o algunas de las fases características del desarrollo aplicativo.

Así existen los siguientes roles o perfiles de desarrollo:

Planeación Estratégica

Es el encargado de definir cual será la infraestructura tecnológica de la compañía o de la organización en cuestión. Sobre la base de las necesidades de procesamiento de datos, de las necesidades generales del negocio y de las tendencias tecnológicas, define cual será la infraestructura y la organización necesaria para la operación de los sistemas de la empresa o institución. En particular define qué funciones serán centralizadas, cuales distribuidas, cuáles serán los departamentos o células de trabajo que se encargarán de desarrollar, mantener y operar los sistemas de la organización. También define cuestiones administrativas como por ejemplo, la contratación de proveedores o de empleados internos, etc. Una decisión de planeación estratégica es la base de datos que se utilizará. Imaginen por ejemplo la Universidad Nacional Autónoma de México y háganse estas preguntas: ¿Las aplicaciones para control escolar son aplicaciones centralizadas o descentralizadas? ¿Las desarrolla una entidad rectora o cada escuela y facultad desarrolla sus propios sistemas? ¿La información de estas aplicaciones está en una plataforma común o en diversas plataformas? ¿Se utiliza software comercial? ¿Cuánto se invierte en licencias de software comercial y en soporte a dichos productos? ¿Cuánto representa dentro del gasto institucional el hardware y el software utilizado? ¿De los sistemas de la UNAM cuál es el porcentaje de software que desarrolla internamente contra el adquirido a compañías de software comerciales?

De la mano de las necesidades de la empresa van las necesidades de sistemas. Hoy en día prácticamente no hay industria que no se vea beneficiada por utilizar sistemas computacionales. Este rol es decisivo, ya que de su comprensión de las necesidades del negocio y de la compresión de las tendencias tecnológicas, dependerá el gasto de la empresa en software.

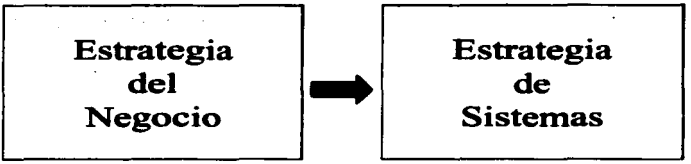


Figura 13 Origen de los requerimientos a sistemas.

De una falta de decisión de estos aspectos dependerá el gasto que deberá dedicar una organización para mantener operando sus sistemas. Existen a la larga también factores que pueden afectar la inversión en tecnología, como por ejemplo, el hecho de que un proveedor desaparezca, o que la tecnología se vuelva rápidamente obsoleta. Ejemplos de ambos casos se han visto varios en la historia de los sistemas basados en PC, pero también en otras plataformas. No estoy en contra de que se compre tecnología, que es finalmente lo que significa el comprar software. Pero si estoy en contra de no "adquirirla", es decir, de ser simples consumidores de tecnología en lugar de ser aprendices para posteriormente ser productores y finalmente desarrolladores de tecnología. Países como Japón basaron totalmente su desarrollo en la adquisición de la tecnología.

Hablando en términos económicos utilizar software gratuito puede ser más costoso en términos de tiempo, fallas y capacitación que los productos comerciales. Pero a largo plazo son mucho más económicos porque no existe dependencia con compañías comerciales que fijan sus precios en dólares y que no están sujetas a ningún control que no sean los del mercado.

Un país como México, tiene el suficiente tamaño como para producir y sustentar el software que se requiere en todas las instituciones públicas. Pero a la fecha, ninguna institución gubernamental se ha erigido como área de Planeación Estratégica para tomar este tipo de decisiones. Hoy en día el país trata de modernizarse mediante la implementación del programa e-México, una decisión aparentemente simple: Sofware comercial vs Software libre, significará un ahorro o un gasto considerable durante muchos años para el país.

Arquitectura Tecnológica

Otro rol que es cada vez más significativo en las empresas, es el de Arquitectura Tecnológica o Arquitecto de Software. Esta persona es la encargada de definir la estructura general de los sistemas. Define cuantos sistemas se construirán, cuántos se comprarán y se modificarán, de qué manera se comunicarán estos sistemas, como compartirán la información, cuáles serán los protocolos de comunicación, etc. También se encarga esta persona, apoyada en asesores, de definir el hardware sobre el que se montará el software construido o comprado. Este puesto adquiere mayor relevancia en nuestros días, ya que es una necesidad que los sistemas compartan información, el que lograr esto sea un esfuerzo simple o complejo depende tanto de la Planeación Estratégica como de la Arquitectura Tecnológica.

Un ejemplo de lo que sucede cuando no existe ni una ni otra es el siguiente: Empresas con diferentes tipos de equipos, por ejemplo máquinas Unix de diferentes proveedores o marcas, pc's, máquinas de fabricantes que ya no existen y con redes de comunicaciones diferentes: Ethernet y Token Ring. Aunado a esta babel, sistemas desarrollados en diversos lenguajes sin soporte para comunicaciones, por ejemplo: COBOL, RPG por mencionar algunos conocidos. También estas empresas frecuentemente utilizan tecnología "propietaria", es decir, tecnología

que pertenece a una sola compañía que no está basada en estándares. Además a este escenario, debemos agregar que los datos de estas compañías están dispersos en archivos "planos", archivos VSAM, Dbase, FoxPro, DB2, MS Access, etc.

Empresas como la anteriormente descrita no es la excepción. Desgraciadamente es la empresa "típica". Al costo de mantener tales sistemas tan diversos, en plataformas tan diferentes, además del costo software y soporte, hay que agregar también el costo en personal. Generalmente se requieren diferentes tipos de expertos con diferentes habilidades.

Ambos roles: Planeación y Arquitectura, son los más importantes en una organización. Y generalmente nunca escribirán una línea de código. Pero sus decisiones (o la falta de ellas) influirán en la cantidad y estructura de las líneas de código que se escriban. El resultado de ambos perfiles se esquematiza en la siguiente figura. Por un lado está la Estrategia del Negocio, las cuales resultan en Requerimientos de Negocio y los cuales se convertiran en Requerimientos de Sistemas, el planeador de la estrategia debe decidir cuál será la infraestructura tecnológica para Desarrollo y que posteriormente se utilizará en Producción. Por otro lado, está la estrategia de Sistemas de la cual es responsable Arquitectura Tecnológica. Ambas estrategias coinciden en los Sistemas, por un lado porque la infraestructura tecnológica de producción debe tender hacia la infraestructura deseada (la de Desarrollo) y porque deben mantenerse los sistemas actuales operando sobre la infraestructura ya existente.

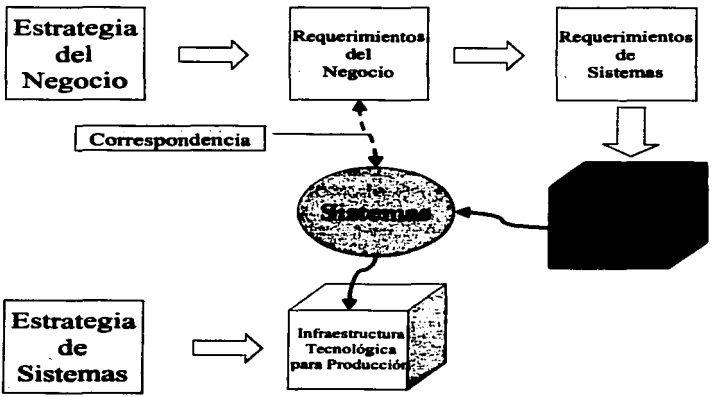


Figura 14 Interrelación entre las Estrategias, Infraestructura y Requerimientos de sistemas de una empresa.

Analista de Requerimientos

Es el encargado de entrevistar a los usuarios para obtener reglas del negocio, o explicado de otra manera, convierte los requerimientos del negocio en requerimientos de sistemas. Con su labor inicia el desarrollo de sistemas. Su labor es importante porque el éxito de un proyecto

depende del análisis, ya que en esta fase se establecen las funciones y condiciones que deberá satisfacer el software para ser de utilidad para los usuarios. El 90% de los fracasos en el software se deben a estos simples factores:

- el software no cumple con los requerimientos ó
- el software no funciona.

En el primer caso, se debe a que los requerimientos estuvieron incompletos o de alguna manera los requerimientos no se tomaron en cuenta para el diseño del sistema. Generalmente se culpa a los usuarios por no ser lo suficientemente "claros" al momento de hacer sus requerimientos. Pero la realidad es que, el responsable de obtener requerimientos "claros" y completos es el Analista de Requerimientos, no el usuario. En el segundo caso, software que no funciona, se debe a que el software no fue debidamente probado y por lo tanto no se detectaron fallas evidentes (son evidentes al menos para el usuario, no así para los desarrolladores). En este caso el responsable es el Inspector de Calidad, pero mucho del trabajo de esta persona se facilita si el Analista de Requerimientos establece con precisión cuales son las "Respuestas" esperadas del software bajo determinadas condiciones y para las condiciones de excepción. Debe tomarse en cuenta, que la persona que mejor idea tiene del resultado final es el Analista de Requerimientos y no el Inspector de Calidad.

Diseñador de Sistemas

Este rol es el encargado de convertir los requerimientos del sistema en componentes de software. Estos componentes pueden ser: Programas, archivos, bases de datos, estructuras de datos, protocolos de comunicación, etc. Para ello, puede utilizar diversas metodologías como el Diseño Estructurado de Sistemas, o el Diseño Orientado al Objeto. Nótese que la labor del Analista de Requerimientos debería ser universal, puesto que hasta el momento de los requerimientos no existe la definición de cómo se construirá el código ni bajo qué métodos. Sin embargo, es común que las metodologías de desarrollo establezcan también métodos y procedimientos para recabar los requerimientos con la finalidad de facilitar la "conversión" de esos requerimientos en "componentes" de sistemas. Parte del trabajo del diseñador de sistemas, es la de establecer la mejor manera de "particionar" el sistema, es decir, dividirlo en bloques más simples de manipular y de analizar para su diseño. Dependiendo de la metodología de diseño que se siga, el resultado será la definición de módulos, funciones y programas, para el caso de Diseño Estructurado y Casos de uso, Escenarios y Clases para el caso de Diseño Orientado a Objetos

¿Cuál método es superior? Es hoy en día una pregunta ociosa. Ya que dependiendo del problema cada metodología presenta ventajas y desventajas. En general se puede decir, que para procesos interactivos es mejor emplear una metodología Orientada al Objeto. Mientras que para procesos batch (donde no hay interacción con el usuario) es mucho mejor el Diseño Estructurado. Discutir detalladamente las ventajas y desventajas de ambas metodologías bajo situaciones específicas es también buen tema para toda una Tesis.

Desarrollador de Componentes

Una vez que el Diseñador establece los componentes que será necesario construir, el Desarrollador es el encargado de producirlos. Aquí se utiliza la palabra componente para referirnos en general a diferentes tipos de código que se puede requerir para producir un sistema, por ejemplo: Programas, Clases, Scripts, Macros, etc.

El desarrollador o programador, es el que recibe la mayor presión durante el desarrollo de los sistemas. Pero en la práctica es notorio, que los proyectos de sistemas fracasan más frecuentemente por requerimientos mal definidos o incompletos (Analista de Requerimientos)

y por deficiencias en el diseño (Diseñador de Sistemas) que por una mala o inadecuada codificación. Esto siempre y cuando exista un Analista y un Diseñador, ya que cada vez es más frecuente que se deje en manos del Programador la responsabilidad de analizar y diseñar los sistemas.

Para producir los componentes, los desarrolladores pueden seguir 3 métodos:

a) Utilizar lógica preconstruida.

b) Utilizar modelos preconstruidos.

c) Utilizar un generador de código.

En el caso de los generadores de código, es mejor utilizarlos para producir componentes "genéricos". Es decir, componentes que no tienen una lógica particular. Por ejemplo, las pantallas de mantenimiento de catálogos o las ya clásicas pantallas de Altas, Bajas y Cambios. La única desventaja de los generadores de código es su alto costo, pero si se considera el ahorro en tiempo de codificación y pruebas resultan ser muy económicos.

La lógica preconstruida, es cuando se escribe código desde cero, o casi cero, pero basados en estructuras generales de desarrollo. Por ejemplo, el siguiente es un ejemplo de Lógica Preconstruida:

```
Proceso Secuencial Simple
Ejecuta Prólogo; // Se abren archivos, se leen parámetros, etc.
Lee Archivo de Entrada
Si es fin de archivo Entonces
        Cuidado el archivo está vacío
        Termina
Fin-Si.
Mientras NO sea fin de archivo
        Procesa registro actual
        Graba Archivo de Salida
        Lee Archivo de Entrada
Fin-Mientras
Ejecuta Epilogo; // Cierra archivos, imprime cifras de control, etc.
Fin.
```

Este "proceso" es el más simple de todos y se le utiliza para procesar el contenido de un archivo de principio a fin o para procesar un conjunto de renglones obtenidos de una tabla de una base de datos. Un programador cuando tenga ante sí el requerimiento de generar un programa que lea todo un archivo y procese registro por registro y produzca una salida, no tendrá que reinventar la rueda. Simplemente toma la lógica "preconstruida" apropiada y escribe el código siguiendo la estructura indicada.

En otras palabras, la Lógica Preconstruida se puede considerar como el algoritmo universal, que sirve para generar programas en cualquier lenguaje.

Lo interesante de todo esto, es que todos los programadores principiantes tienen la noción de que las posibilidades de hacer programas diferentes son infinitas. En la práctica se descubre que esto no es así. Procesos diferentes de tipo Batch (no interactivos) son solamente 5 diferentes. Procesos Online (interactivos) son solamente 2. Todos los diferentes tipos de procesamiento de información caen en alguno de estos 7 modelos o son resultado de combinar algunos de ellos.

Esto significa que un programador que conozca estas 7 estructuras generales puede construir cualquier programa que se le presente. En el caso (probable) de que se presente un programa que no se adecue a la aplicación de alguno de estos modelos, normalmente se trata de programas mal diseñados o de los que comúnmente se conocen como "programas maravilla". Los programas maravilla son aquellos sistemas que carecen de diseño y por lo tanto de subdivisiones, de tal forma que en un sólo componente se inserta toda o la mayoría de la funcionalidad del sistema.

Por último están los modelos preconstruidos. Estos son muy similares a la lógica preconstruida, con la excepción de que en este caso se trata de código ya preconstruido. En otras palabras es lógica preconstruida ya implementada en un lenguaje de programación específico y para una finalidad específica. La única limitación que tienen es que sólo son aplicables a un lenguaje en particular y para situaciones específicas. Por ejemplo, tomando la Lógica preconstruida del Proceso Secuencial Simple se pueden obtener dos modelos preconstruidos: Programa para procesar un archivo plano de inicio a fin en modo secuencial y Programa para procesar un Conjunto de renglones obtenido de un Cursor de base de datos. En ambos casos la lógica es la misma, pero en un caso se trata de archivos "planos" y en otro de datos que se obtienen de una base de datos mediante un Cursor (construcción de SQL).

Debido a que en la práctica muchos componentes son muy parecidos, es cuando surge la necesidad de preconstruir estos modelos de tal manera que la siguiente vez que se necesite un componente similar se reutilice el código base. Esto permite también estandarizar el código en estructuras bien conocidas por todos los programadores de una organización.

El siguiente paso de los modelos preconstruidos es precisamente el de los generadores de código. Un generador de código, no es otra cosa que un sistema que tiene la habilidad de seleccionar modelos y configurarlos de acuerdo a parámetros introducidos por el usuario. Pero construir un generador de código no es tarea simple, por lo que se opta por el esquema de Modelos Preconstruidos.

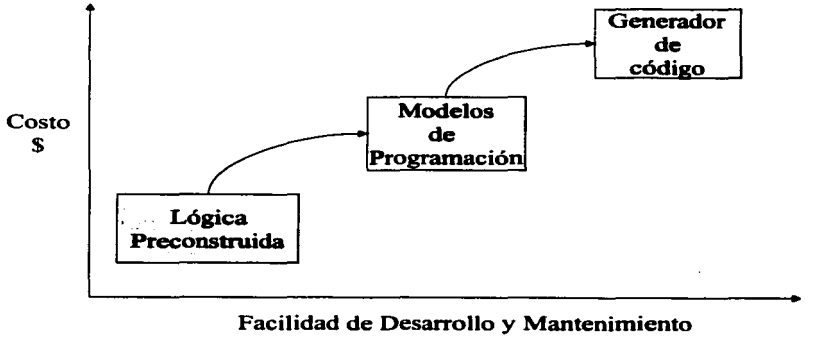


Figura 15 Herramientas de desarrollo. A mayores facilidades de desarrollo mayor es su costo.

Adicionalmente, los Desarrolladores de Componentes se suelen especializar y reciben nombres de acuerdo al tipo de componentes que desarrollan, en el caso de la Tecnología Java se tienen las siguientes especialidades:

- Desarrollador de componentes Web (Servlets y JSP's)
- Desarrollador de Enterprise Java Beans (Session, Entity y Messages Beans)
- Desarrollador de JavaBeans (Clases Java auxiliares)
- Desarrollador de Interfaz (HTML, JavaScript, XML)
- Diseño Gráfico Web (Define cómo se verán y se dispondrán los elementos de manera que sean atractivos y fáciles de usar)

Ensamblador (Integrador) y Empaquetador de Componentes

Trabaja con componentes preconstruidos. Se encarga de integrarlos de tal manera que se comuniquen de la manera más apropiada para cumplir los requerimientos del sistema. Esto se conoce como la "plomería" del sistema. Es en esta fase donde también participa el Arquitecto Tecnológico, porque él define la manera apropiada de interconectar componentes a diferentes niveles. El ensamblador también se encarga de "empacar" la aplicación en módulos que sean fácilmente instalables. Esta parte es importante, porque muchos problemas del desarrollo del software se originan debido a que no hay una forma fácil y automatizada de instalar el sistema bajo diferentes ambientes y es este punto donde disponer de herramientas como Ant es necesario. Se requieren al menos dos ambientes: Desarrollo y Producción. No tomar en cuenta esto origina que también sean frecuentes los sistemas que funcionan en el ambiente de Desarrollo, pero no funcionan en el ambiente de Producción. Es menos común, pero también llega a ocurrir que un sistema funciona en Producción, pero no en Desarrollo.

Solo falta mencionar que este recurso generalmente es el más capacitado ya que escribe el código más delicado (interconexión de componentes) pero también actúa como "árbitro", ya que cuando algo falla debe tener la capacidad de determinar si la falla está en algún componente y en cual o en el código de "plomería".

Instalador (Implementador) de Paquetes

Este rol, lo desempeña un recurso que conozca bien el ambiente sobre el que se instalará la aplicación. Depende totalmente de su conocimiento del ambiente, así como de la documentación del sistema.

Inspector de Calidad

Su labor parece simple: Inspeccionar que cada componente se apega a los requerimientos del sistema y que los requerimientos del sistema se apegan a los del negocio. Sin embargo, esta es la tarea más compleja dentro del equipo de trabajo y normalmente es la más menospreciada. Se requiere un rol con nociones técnicas, con nociones del negocio y con una alta especialización en Pruebas de Sistemas.

Administrador de Base de Datos

Sus funciones son:

- Diseñar el Modelo Lógico de Datos
- Diseñar el Modelo Físico
- Instalar el software del manejador de base de datos

- Implementar el Modelo Físico
- Afinar las consultas
- Afinar la base de datos

Pueden existir otros roles, dependiendo de la plataforma sobre la cual se desarrolla. Aquí se mencionan sólo los relevantes para la Plataforma Java2. Además existen otros roles que son más bien organizadores, como el del Líder de Proyecto, los cuales no se mencionan aquí por no estar directamente involucrados en cuestiones técnicas sino mas bien administrativas. Obviamente son roles que impactan en el desarrollo de un proyecto, pero están fuera del alcance de esta Tesis.

Requerimientos de capacitación

Cada perfil o rol tiene diferentes requerimientos de capacitación. Algunos roles son universales (sus requerimientos no dependen de la plataforma aplicativa) y otros son roles especializados en una plataforma. Aquí solo se relacionan los roles que tienen requerimientos específicos de capacitación para la Plataforma Java2. Para cada perfil se muestran los mapas de capacitación necesarios.

Diseñador J2EE

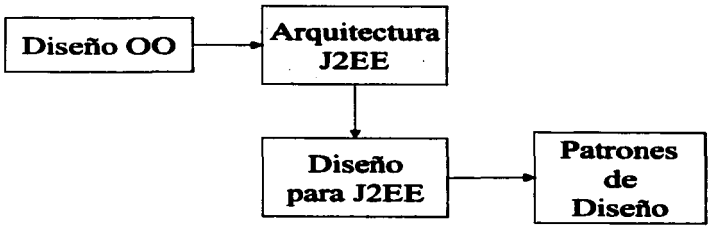


Figura 16 Ruta de capacitación Diseñador J2EE.,

Desarrollador J2EE

Mapa común a todos los desarrolladores:

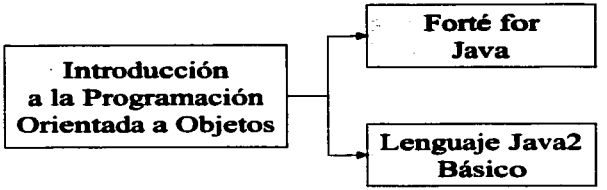


Figura 17 Ruta de capacitación Desarrollador J2EE.

Desarrollador de JavaBeans (Clases Java auxiliares)

No requiere capacitación adicional.

Desarrollador de componentes Web (Servlets y JSP's)

En este perfil, es básico aprender SQL, sin embargo, dado que no hay una dependencia directa respecto de los otros componentes, no hay una secuencia recomendada de estudio.

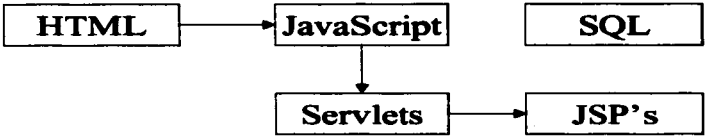


Figura 18 Ruta de capacitación Desarrollador Componentes Web.

Desarrollador de Enterprise Java Beans (Session, Entity y Messages Beans)

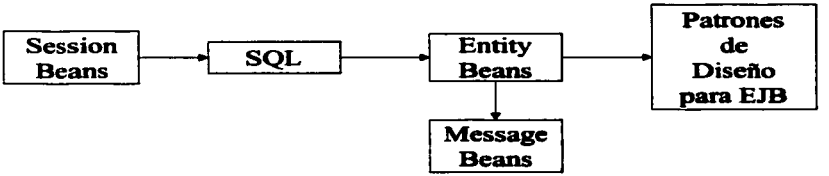


Figura 19 Ruta de capacitación Desarrollador EJB

Integrador J2EE

Este perfil debe conocer un poco de todo. Pero principalmente de los mecanismos de comunicación entre componentes. En el siguiente mapa se muestran los requerimientos de capacitación.

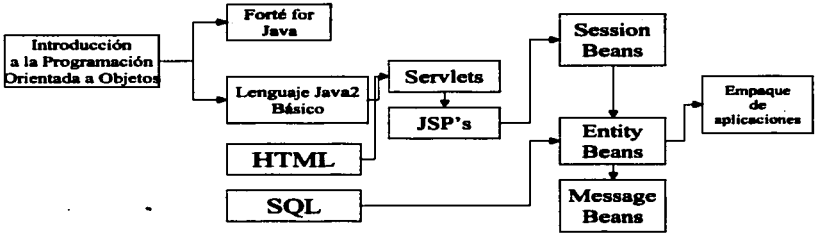


Figura 20 Ruta de capacitación Integrador J2EE.

Este es el perfil más completo en cuanto a temas que cubre, sin embargo, el nivel en que los cubre es a menos detalle que los otros perfiles.

Instalador J2EE

Este perfil, depende totalmente de la implementación que se utilizará para instalar las aplicaciones. Al menos debe conocer la implementación de Referencia, que es el Sun Java2 Software Development Kit, Enterprise Edition. Otras implementaciones son: IBM WebSphere, Oracle Application Server, HP Bluestone Total e-Server, o JBoss.

De estas implementaciones solo JBoss es de software libre. Implementaciones gratuitas son: Sun SDK EE y HP Bluestone Total e-Server. Sin embargo, casi todos los proveedores tienen versiones gratuitas de sus productos exclusivamente para desarrollo, por lo que se tiene una amplia gama de opciones. Conocer cada uno de estas implementaciones se centra en estudiar la instalación, configuración y uso de la implementación. Principalmente las tareas de seguridad e instalación de aplicaciones J2EE, configuración de accesos bases de datos y configuraciones especiales que pudieran requerir las aplicaciones.

Otros conocimientos necesarios

Todas estas tecnologías requieren de utilizar una máquina con algún sistema operativo. Y hoy en día el sistema más al alcance de todos es Windows, porque al comprar una computadora el consumidor está obligado a comprar una licencia de este sistema operativo. Pero en los sistemas empresariales también es común encontrar sistemas unix y más recientemente Linux. Es por eso que la plataforma recomendada en esta tesis es Linux, porque además de ser una plataforma muy estable, es lo más accesible parecido a un unix. De esta manera se tendrá un panorama más amplio de los sistemas operativos empresariales. Además habrá que agregar otros conocimientos útiles y que deben ser tomados en cuenta para el desarrollo de sistemas aplicativos para Web, la siguiente es una lista no exhaustiva de estos requerimientos:

- Estándares de codificación para Java
- Ant
- Struts (framework de desarrollo que se utilizará más adelante)
- Linux Instalación
- Linux Utilización

- Linux Administración
- XML Básico
- Seguridad en la Web
- Protocolos de Internet: HTTP, FTP, POP, SMTP

Fuentes de materiales gratuitos para autoaprendizaje

En Internet se puede encontrar casi todo lo que se necesita. La siguiente es una relación de sitios en donde se puede obtener información tanto para principiantes como para avanzados de cada tema aquí mencionado, un probable impedimento para acceder a estos sitios quizás sea el idioma, todos se encuentran en idioma Inglés (desafortunadamente, los mejores están en inglés), pero debe tomarse en cuenta que también como profesionales, estamos obligados a comprender al menos un idioma adicional al materno.

- HTML

Existen miles de sitios en Internet para aprender HTML. El problema es encontrar uno que esté orientado a nuestras necesidades específicas, que en nuestro caso es construir aplicaciones. En primer lugar está el siguiente sitio para aprender lo básico de HTML:

http://www.davesite.com/webstation/html/

Una vez aprendido lo básico, habrá que aprender el tema más importante para construir aplicaciones que es el generar FORMS, es decir, pantallas de captura. Ya que esto nos permitirá que el usuario captura información necesaria para las aplicaciones. Este tutorial de formas se recomienda tomarlo junto con Servlets, de esta manera los temas quedaran enlazados y no pasará mucho tiempo entre uno y otro, ya que en realidad van de la mano. El sitio recomendado es el siguiente:

http://www.pagetutor.com/pagetutor/forms/index.html

- JavaScript

http://www.wdvl.com/Authoring/JavaScript/Tutorial/

- SQL

SQL es un lenguaje estándar de acceso a base de datos. Normalmente se aprende a manejar junto con alguna base de datos. En este caso, como utilizaremos PostgreSQL, tomaremos el tutorial de PostgreSQL, el cual incluye lo referente al lenguaje de consultas a la base de datos.

http://www.ca.postgresql.org/docs/aw_pgsql_book/index.html

- Programación Orientada a Objetos

Al igual que SQL, que se aprenden de la mano de un producto específico, los conceptos de programación Orientada a Objetos es mejor aprenderlos de la mano de un lenguaje de programación. Para nuestro caso particular, casi todos los tutoriales de Java incluyen lecciones de programación Orientada a Objetos. Aunque también existen tutoriales para usuarios con alguna experiencia en POO con algún otro lenguaje como C++.

- Lenguaje Java

La mejor fuente de información es el tutorial de Java de Sun. La dirección es:

http://java.sun.com/docs/books/tutorial/index.html

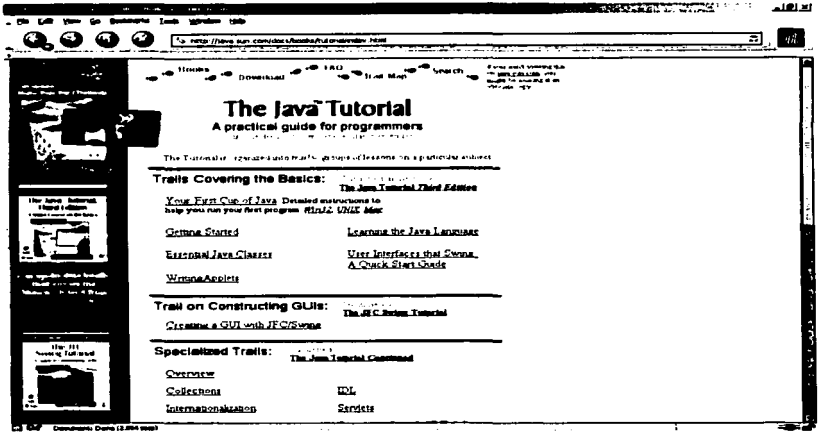


Figura 21 The Java Tutorial en internet.

El tutorial se divide en 4 partes:

- Basics
- Constructing GUIs
- Specialized Trails
- Trails available online only

Se recomienda tomar la parte de Basics en su totalidad. La 2ª. parte, Constructing GUIs, solo se utiliza para construir clientes gráficos, como por ejemplo, las aplicaciones que se construyen con Visual Basic, pero para la construcción de aplicaciones web no es necesario.

Los tracks especializados abarcan diversos temas los cuales no necesariamente aplican a todo tipo de aplicaciones. De este grupo de temas se deben tomar al menos los siguientes:

- Collections
- Internationalization
- JavaBeans
- JDBC Database access

- Security in Java2 SDK 1.2
- JAR Files

Del 4°. grupo de temas, no es necesario tomar alguno.

- Forté for Java

En la página de descarga de Forté for Java, se encuentra el documento para iniciar en su uso, conocido como Getting Started Guide.pdf, pero la mayoría de la documentación de uso del producto viene en el Help del software.

- Servlets y JSP's

http://www.coreservlets.com/

Este sitio corresponde al mejor libro que hay sobre el tema. En este sitio se puede encontrar material de cursos, código de ejemplo y también el libro en formato PDF.

- Enterprise JavaBeans (Session, Entity y Mensaje) y Empaque de aplicaciones J2EE

http://java.sun.com/j2ee/tutorial/1_3-fcs/

De nuevo, la mejor fuente es el tutorial "oficial" de Sun:

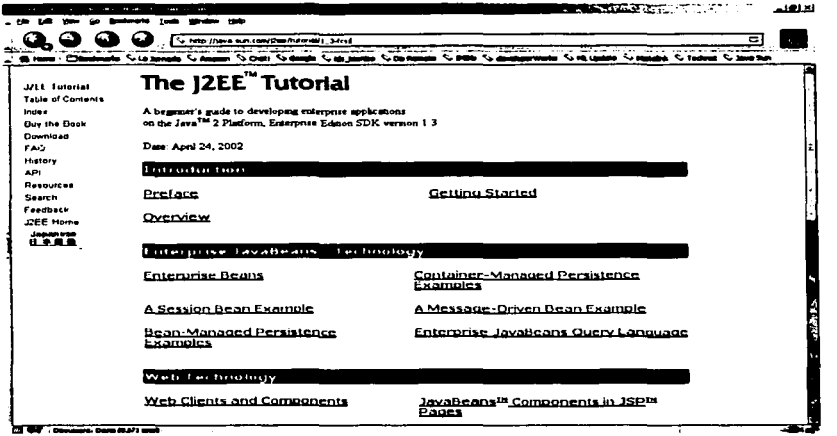


Figura 22 The J2EE Tutorial en internet.

Este tutorial también está dividido en 4 partes:

- Introduction
- Enterprise JavaBeans Technology

- Web Technology
- Platform Services

La 3ª. parte, Web Technology, se refiere a Servlets y JSPs. Obviamente esto corresponde a los mismos temas que se tratan en el sitio anteriormente mencionado.

- Patrones de Diseño EJB

http://www.theserverside.com/patterns/index.jsp

En este sitio se puede descargar el libro correspondiente a patrones de diseño para EJB's.

- XML

Un sitio de XML "puro":

http://www.w3schools.com/xml/default.asp

Para utilizar XML desde Java:

http://java.sun.com/xml/tutorial_intro.html

- Diseño Orientado a Objetos

Desgraciadamente, hay muy poca información específica de métodos de diseño OO, las únicas fuentes de referencia son los libros. Al final de esta tesis se encuentra la bibliografía completa recomendada para este y los otros temas aquí tratados.

- Arquitectura, Diseño para J2EE y Patrones de Diseño J2EE

http://java.sun.com/blueprints/

- Estándares de codificación para Java

http://java.sun.com/docs/codeconv/

- Ant (scripts para automatización de compilación y empaquetamiento)

http://jakarta.apache.org/ant/index.html

- Struts (framework de desarrollo que se utilizará más adelante)

http://jakarta.apache.org/struts/index.html

- Linux

http://www.mandrakecampus.com/

- Seguridad en la web

De este tema, tampoco hay tutoriales en Internet, pero se puede recibir un White Paper, relacionado con el tema, accesando la página:

http://www.verisign.com

- HTTP, FTP, POP, SMTP (protocolos de recepción y transmisión de email)

http://www.ou.edu/research/electron/internet/zen-toc.htm

Sobre Internet y otros temas.

- Sitios de Temas Generales

DeveloperWorks IBM. En este sitio hay muy buenos tutoriales relacionados con Java y con Linux. Destacan los tutoriales de Container Managed Persistence (CMP y CMR).

http://www-106.ibm.com/developerworks/

Technet Oracle. Información relativa a Java y el uso de JDBC.

http://otn.oracle.com

Java Guru

http://www.jguru.com

Cómo se podrá apreciar hay una cantidad considerable de información en la red. Pero esta cantidad abismal se concentra al rededor de 3 temas eje:

- **Lenguaje Java**
- **Tecnología J2EE:**
  - **Componentes Web (Servlets y JSP)**
  - **Componentes EJB (Session, Entity y Message)**

En torno de estos 3 temas, se concentra la información relacionada. Y es que, aprender solamente Java no es suficiente y al mismo tiempo es la puerta para todo. Porque Java actúa como un pegamento que permite integrar múltiples "cosas". Así, por ejemplo, si se requiere construir una aplicación que envíe emails, tomando las direcciones de correo de una base de datos, se requerirá de usar: JDBC y JavaMail API. JDBC es parte de Java Standard Edition, mientras JavaMail API es parte de Java Enterprise Edition. En ambos casos, desde el punto de vista del desarrollador ambos son librerías que se utilizan y ya. Pero en la práctica se requieren conocimientos adicionales: SQL y entendimiento del funcionamiento de los protocolos de email (pop y smtp). Esta es la dificultad y la facilidad de Java: Con Java se puede escribir casi cualquier tipo de aplicación, con el costo de tener que aprender los estándares que soportan cierta funcionalidad. En el siguiente capítulo estableceremos un patrón general para construir aplicaciones interactivas. De tal manera, que nos permite establecer un orden para poner cada tecnología en su lugar.

Capítulo 4. Estructura básica de aplicación

Estructura Modelo-Vista-Control

Para construir aplicaciones, primero se deben particionar. Dependiendo del método de diseño empleado, esta partición puede ser de diferentes maneras. Generalmente para aplicaciones Web, que es el modelo recomendado actualmente, la partición se hace por eventos. Un evento es la interacción que existe entre el usuario aplicativo y la funcionalidad del sistema.

Por ejemplo, un sistema de reservaciones para un hotel, tiene al menos los siguientes eventos con los respectivos actores:

Evento	Actor
Consulta de disponibilidad y precio	Cliente
Reservación	Cliente
Consulta de Reservaciones	Administrador
Check in	Recepcionista (a petición del Cliente)
Check out	Recepcionista (a petición del Cliente)

Es fácil identificar los eventos si se pregunta uno, cuáles son las "situaciones" que se presentarán en el Ciclo de Vida del negocio para cada uno de los actores participantes. Esto implica, conocer de antemano el Ciclo de Vida del negocio.

En el "*Capitulo 6. Metodología de Análisis y Diseño Básica*" se detalla un poco más este procedimiento. El resultado es, que la aplicación a construir se divide en Eventos. Cada evento se compondrá de un conjunto de "pantallas" que el sistema utilizará para interactuar con el "Actor" y detrás de cada una de estas pantallas habrá programas interactuando con el "Actor".

Un diagrama simplificado de este esquema es el siguiente:

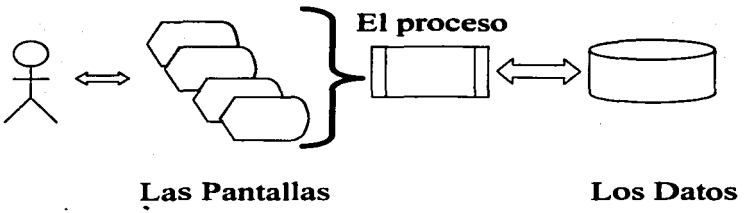


Figura 23 El componente más simple de una aplicación: Un Evento y la respuesta del sistema.

De este sencillo esquema se desprende que es lógico separar en 3 componentes una aplicación:

- La presentación
- La lógica del negocio
- El acceso a datos

Esta separación se conoce también como 3-capas lógicas. Este modelo de programación tiene una amplia aceptación bajo diferentes esquemas de desarrollo y para diferentes lenguajes y plataformas. Pero esta "separación" en la realidad, es una separación lógica, no física. Es decir, que los programas físicamente pueden ser 2 ó varios componentes no directamente relacionados con esta separación de 3 capas. De tal manera que, por ejemplo, en un componente se manejaba la presentación y parte de la lógica del negocio, mientras en otro componente se manejaba el resto de la lógica del negocio y el acceso a datos. Resultando de esta manera una codificación en 2 capas. Eso se representa de esta manera:

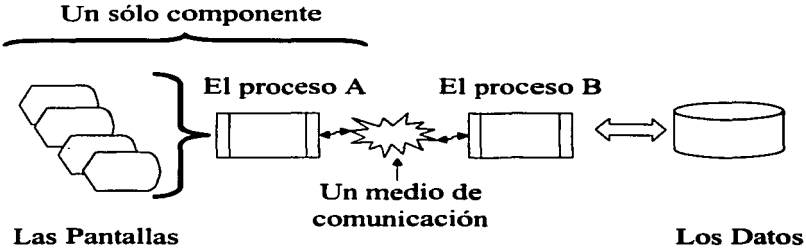


Figura 24 Atención de un evento con software en 2 capas.

Este esquema es la programación "tradicional" de lenguajes como Visual Basic.

En el caso de la Plataforma Java2, este esquema se lleva a un nivel de mayor de separación. De tal manera que cada función específica del proceso se separa en un componente bien

diferenciado. Y esta separación es de manera lógica y física. Este modelo, que no es nuevo, se conoce como Model2 ó más comúnmente como Model-View-Controller.

El MVC consiste en separar en 3 grupos los componentes aplicativos:

- Modelo: Es el componente que efectúa las acciones solicitadas por el usuario, por ejemplo, "Registrar una reservación". En otras palabras, son los componentes que efectúan las Transacciones soportadas por el sistema: modifican datos, registran los cambios, validan la información introducida por el usuario entre otras operaciones comunes.
- Vista: Son los componentes encargados de la interacción con el usuario, es decir: pantallas donde se presentan datos o formas de captura.
- Control: Son los componentes que determinan el flujo de la aplicación entre pantallas y la validez de las operaciones solicitadas por el usuario.

En el siguiente diagrama se muestra una vista simplificada de este modelo:

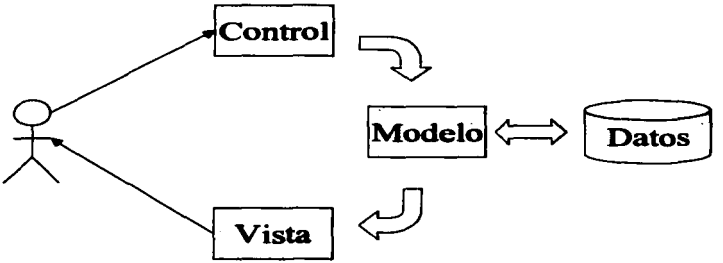


Figura 25 Diagrama de patrón básico: Model View Controller (Modelo Vista Control)

En este diagrama el proceso es el siguiente:

a) El módulo de Control recibe los "gestos" del usuario, valida la información proporcionada por este y también sirve de "puente" entre los datos en el "formato del usuario" y los datos en el "formato del sistema".

b) Si los "gestos del usuario son válidos, entonces se "arma" una petición de un "servicio" o "transacción". Esta petición se efectúa al módulo del Modelo de la aplicación.

c) El Modelo, recibe la petición con todos los datos necesarios. Estos datos los recibe por medio de parámetros o a través de objetos almacenados en la "sesión" del usuario (no mostrada en el diagrama). Ejecuta el "servicio" o "transacción" solicitada, accediendo para ello a los datos almacenados y tal vez generando nueva información en la base de datos.

d) Una vez procesada la información, se genera una respuesta y entonces se transfiere el control al módulo de Vista.

e) De nuevo, el paso de la información puede ser por parámetros o por objetos almacenados en la sesión. En las aplicaciones Java para web, normalmente es a través de la sesión.

f) La vista, de acuerdo a la respuesta formatea una pantalla reconocible por el usuario. Es decir, efectúa la conversión inversa de datos en "formato del sistema" a datos en "formato del usuario".

g) Finalmente, se le envía la respuesta al usuario.

Implementación del MVC en Java2 EE

La plataforma Java2 Enterprise Edition, prefiere seguir el patrón de diseño MVC para la construcción de aplicaciones. Y la implementación recomendada es la que se muestra en este diagrama:

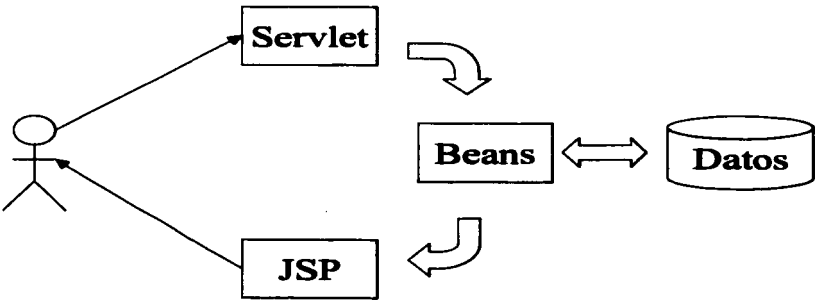


Figura 26 Model View Controller implementado con componentes J2EE.

En este diagrama, se muestra una aplicación típica para Web Transaccional. Los Beans, pueden ser Javabeans (clases Java "normales") o Enterprise JavaBeans (EJB's). Los Beans corresponden al Modelo del MVC.

El Servlet es el Controlador y se encarga de recibir los "gestos" del usuario, que en este caso, adoptan la forma de un Request de http.

La JSP se encarga de generar la salida para el usuario, que adopta la forma de un Response también de http. Y estos son los elementos básicos de una aplicación Web.

1. Se recibe un "request".
2. Se procesa
3. Se genera un "response".

Al diagrama anterior le faltan los siguientes elementos:

a) El paso de información, que se hace normalmente por la "sesión" del usuario. Más adelante explicaré en que consiste.

b) Los detalles de implementación del Modelo.

c) Cómo aplican los frameworks de desarrollo en el MVC.

Todos estos elementos pueden quedar de una de las siguientes formas:

1. Con EJBs
2. Con JavaBeans

El modelo más complejo es con EJBs y es el que se muestra en el diagrama siguiente.

En el caso de JavaBeans los SessionBeans se sustituyen por clases Java "normales" y los EntityBeans por Objetos de Acceso a Datos, es decir, clases Java especializadas en el acceso a datos.

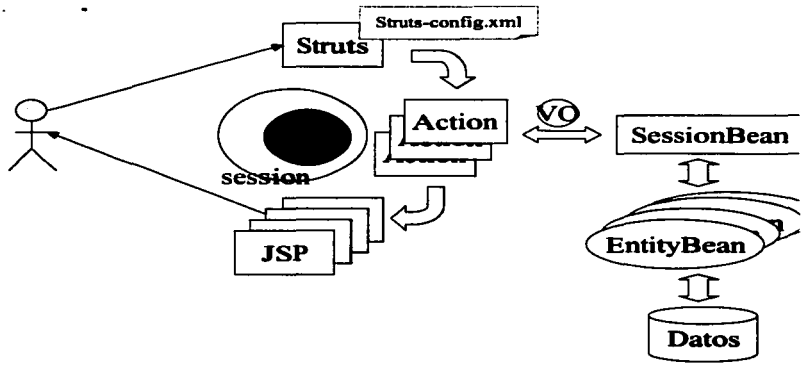


Figura 27 Model View Controller implementado con componentes J2EE y Struts.

En este diagrama el modelo está implementado con EJB's. Estos son de 2 tipos básicos:

a) Session: que controlan el flujo de la aplicación. Sólo se diferencian de otros tipos de clases java, en que son componentes Transaccionales, es decir, que siempre ejecutan dentro de una transacción de sistemas. Para lograr esto, deben tener una estructura predefinida en el estándar y poseen la habilidad también, de ser objetos administrados por el contenedor EJB. En otras palabras, el contenedor decide cuando y cuantos objetos de tipo Session, generar para atender las peticiones que le hacen.

b) Entity: Son objetos especializados en el acceso a datos. Normalmente, a una base de datos de SQL. También están administrados por el contenedor EJB y tienen la habilidad de participar dentro de Transacciones y ser compartidos entre diferentes usuarios.

La única desventaja de los EJB's, es que son objetos que consumen más memoria y cpu que los objetos "normales". Pero tienen a su favor, que bien utilizados pueden soportar múltiples usuarios concurrentes. Esto se debe, a que normalmente los contenedores reutilizan, literalmente, los Enterprise Javabeans para atender múltiples usuarios. Cosa que no sucede con los Javabeans "normales", ya que estos se crean y se destruyen por cada ejecución para cada usuario.

De esto se desprende un consejo de diseño:

Si la aplicación tendrá pocos usuarios, no es ninguna ventaja compartir objetos, y por lo tanto se puede desarrollar una aplicación rápida y con poco consumo de memoria utilizando sólo JavaBeans.

Por el contrario, si la aplicación tendrá múltiples usuarios concurrentes (digamos, más de 50 simultáneamente), entonces debemos procurar al máximo compartir y reutilizar objetos. Esto se logra automáticamente utilizando EJB's. Y obviamente, una aplicación así construida requerirá más memoria que si se hubiese construido con JavaBeans solamente y fuesen pocos usuarios. Pero consumirá mucho menos memoria y cpu, en cargas de trabajo elevadas.

Sin embargo, la carga de usuarios concurrentes no es suficiente para decidir si una aplicación se construirá con EJB's o no. Otro punto de decisión es también la seguridad requerida. Por ejemplo, una aplicación para Banco, normalmente será accedida por múltiples usuarios simultáneamente, pero la probabilidad de que dichos usuarios compartan información es muy baja ya que cada usuario normalmente opera solo sobre sus propias cuentas. Pero se vuelve muy importante la seguridad, y en este caso los EJB's siempre ejecutan sobre un "entorno" protegido (el contenedor) y los JavaBeans se pueden ejecutar directamente sobre cualquier máquina virtual java. De tal manera, que si una función como "Transferencia de Saldo" se programara en un JavaBean, cualquier usuario pudiese ejecutar una función no autorizada simplemente con un java.exe.

En el diagrama se observan también los siguientes elementos:

Actions

Struts, es un framework de control de flujo basado en MVC. En este caso la interacción con el usuario se divide en pequeñas tareas. Estas tareas se atienden mediante Actions que son clases de Java heredadas de un componente del mismo nombre: Action. Mediante un archivo de configuración llamado struts-config.xml, se le indica a Struts, cual es el Action que atenderá una cierta petición en particular del usuario.

También en este archivo de configuración se indican cuales serán las "vistas" que se utilizarán en caso de error o de éxito de la operación. Este punto es interesante, porque mediante la introducción del ServletController, se vuelve la aplicación verdaderamente dinámica. Es decir, que la siguiente pantalla que verá el usuario depende de la lógica de la aplicación y no está "fija" de ninguna manera.

Value Object (VO)

Los datos introducidos por el usuario, y generados por el sistema, se transfieren entre cada uno de los componentes mediante Objetos de Valor. Estos objetos, permiten encapsular múltiples datos en uno solo para simplificar la escritura del código. Existen 2 Value Objects casi siempre durante la ejecución de la aplicación: Un ValueObject que se encuentra almacenado en el Session del usuario y otro que es una copia que se utiliza para transferir entre la capa Web (el Servlet controller y los Actions) y la capa del Modelo (los beans).

Session

Este es un "objeto" muy especial. Es un área de memoria donde se pueden almacenar temporalmente objetos, los cuales permanecen ahí mientras dura la Sesión del usuario, es decir, mientras el usuario está "conectado" a la aplicación. Tiene 2 funciones principales:

a) Servir de puente entre los diferentes componentes para compartir objetos, como los ValueObjects.

b) Permitir almacenar el estado de la "conversación" con el usuario entre cada "click" que ejecuta el usuario.

Flujo de la aplicación

Con todos estos elementos el flujo de ejecución queda de esta manera:

a) El usuario mediante un browser se conecta a una página "de inicio".

b) Desde esta página solicita una operación, se genera un Request de http.

c) Este request lo recibe el Servlet Controller.

d) El Servlet Controller previamente cargó a memoria el archivo de configuración.

e) Con esta información determina cual es la información que debe tomar del "Request" y en que ValueObject la debe almacenar. Este ValueObject lo guarda en la sesión del usuario.

f) Después el Servlet Controller transfiere el control al Action correspondiente de acuerdo también a los parámetros del archivo de configuración y le pasa copia del ValueObject.

g) El Action, valida la información del ValueObject y de ser válida la información invoca el SessionBean correspondiente para procesar el ValueObject.

h) El SessionBean, auxiliándose de los EntityBeans para acceder los datos, procesa el ValueObject y guarda en el mismo, el resultado del proceso.

i) De acuerdo al resultado del proceso en el SessionBean, el Action devuelve una respuesta de "éxito" o "error" al Servlet Controller. También, actualiza el ValueObject almacenado en la sesión del usuario.

j) El Servlet Controller dependiendo del éxito o fracaso de la operación determina cual es la vista que corresponde, de acuerdo al archivo de configuración. Y le transfiere el control a la página correspondiente (normalmente una JSP).

k) La página que recibe el control, toma el Value Object almacenado en la sesión del usuario y formatea un Response de http, en otras palabras genera una página en HTML con los datos generados por el sistema.

l) El usuario recibe su página HTML y continúa la "conversación" con el sistema.

Este modelo de desarrollo en principio puede parecer confuso, dado el número de componentes que hay que construir para procesar una simple forma de HTML. Pero tiene varias ventajas y por eso es el modelo adoptado:

1) Los componentes son numerosos, pero pequeños. Algunos de ellos con estructuras idénticas, como por ejemplo los Action.

2) Cada componente realiza una función específica, por lo que, es fácil dividir el trabajo entre programadores especializados: Unos encargados de las vistas, otros de los SessionBeans, otro de los Entitys, etc.
3) Los componentes del Modelo, se pueden reutilizar para construir aplicaciones que no sean Web, como pueden ser los clientes Swing (presentación gráfica).
4) Se minimiza la cantidad de código Java que se introduce en las páginas JSP. De esta manera es fácil introducir herramientas de diseño Web, para mejorar la presentación gráfica de las páginas.
5) El patrón de diseño MVC, es un patrón poderoso, porque permite la máxima flexibilidad sin sacrificar complejidad en el código.
6) Dado que cada componente está especializado en una tarea específica, es fácil introducir diferentes herramientas de diferentes proveedores para generar automáticamente componentes específicos. Como por ejemplo, Generadores de EntityBeans, Generadores de Actions, Generadores de SessionBeans. Por el momento, generadores gratuitos de EJB's no existen. Pero se pueden utilizar provisionalmente programas prototipo para no iniciar desde cero a construir estos componentes.

Ejemplo

El ejemplo aquí presentado es un flujo de los más simples: Logon al sistema. Es decir, es un conjunto de pantallas que permiten identificar a un usuario mediante username y password. El proceso es muy simple, ya que solo se busca la coincidencia en la base de datos de los datos introducidos por el usuario. De coincidir estos datos, entonces el usuario puede acceder al sistema. De no ser así es rechazado. El sistema maneja hasta 3 reintentos, después de los cuales se le direcciona a una página de ayuda. Aunque en nuestro ejemplo no hay mucha ayuda para proporcionar. Pero en la vida real, normalmente se provee de un teléfono o una dirección email a donde dirigirse para solicitar ayuda con el acceso.

Esta pantalla no es muy segura debido a dos factores:

a) La conexión entre el usuario final y el servidor J2EE no está encriptada. Por lo que alguien puede "scanear" la red y obtener una password que viaje en ese momento por la red.
b) Las passwords almacenadas en la base de datos no están encriptadas.

El primer problema se resuelve implementando seguridad mediante una conexión SSL. Este tipo de conexiones encriptan y decriptan la comunicación entre el cliente y el servidor, pero esto está mas allá del alcance de esta tesis. Por otro lado, utilizar una conexión SSL, no afecta el código, solo afecta la configuración del Web Server que se esté utilizando.

El segundo problema depende de la base de datos utilizada y del software disponible. Algunas bases de datos ofrecen tipos de datos encriptados por ejemplo. En otros casos los servidores aplicativos incluyen librerías para encripción y decripción. En otros casos, incluso existen librerías que proveen por completo el servicio de autenticación y validación de los usuarios.

De cualquier forma, el 80% de las aplicaciones solo requieren seguridad básica. Y para ese tipo de aplicaciones el código aquí mostrado será suficiente.

El siguiente diagrama muestra las relaciones entre los componentes:

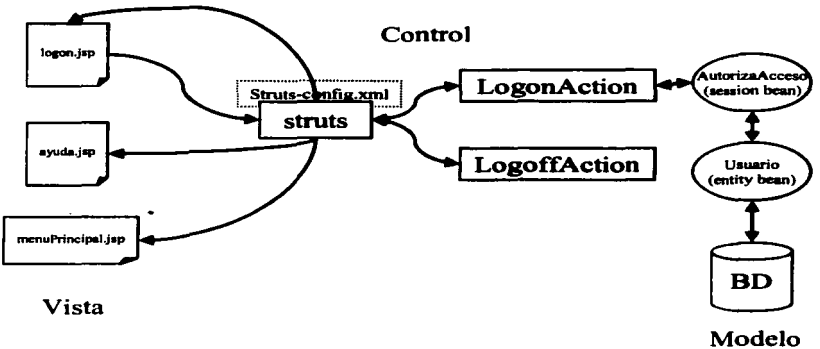


Figura 28 Estructura de una aplicación basada en Struts.

En total son 3 JSP's, 1 Form (no mostrado en el diagrama), 2 Action's y 2 Enterprise Javabeans, 1 Javabean (no mostrado en el diagrama) y 1 archivo de configuración (struts-config.xml).

El archivo de configuración es el que une todas las piezas, de la siguiente manera:

- Especifica cuales son los Request esperados por la aplicación.
- Especifica por cada Request, que Action lo va a procesar.
- También indica para cada Request, cual es el objeto que se utilizará para recibir los datos en variables java. Es decir, cómo viajarán los datos entre cada componente.
- Define cuales son las posibles respuestas de los Actions y quién generará los Response's, para cada una de estas.

En este ejemplo, se tienen:

- 1 Request: La flecha que va hacia el controlador de Struts.
- 3 Response's: Las 3 flechas que salen del controlador de Struts.

Los posibles flujos que se pueden dar para esta aplicación son los siguientes:

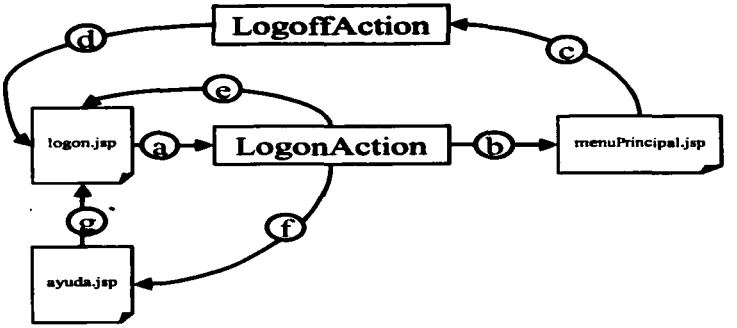


Figura 29 Flujo de una aplicación basada en Struts.

a) El usuario accede a la pantalla de inicio de la aplicación: logon.jsp, y en esta captura su identificador de usuario y password. Estos datos viajan en una Forma de HTML, hacia el controlador de Struts, el cual toma la forma y la envía hacia el LogonAction para que la procese.

El LogonAction es invocado en el método "perform" y recibe los siguientes parámetros:

```
public ActionForward perform(
   ActionMapping        mapping,
   ActionForm           form,
   HttpServletRequest   request,
   HttpServletResponse  response)
```

mapping: Objeto para enviar a Struts la "respuesta" de hacia donde dirigir el flujo
form: Objeto con los datos introducidos por el usuario
request: El request original tal como lo recibió Struts
response: El objeto response si es que se desea enviar una "respuesta directa"

b) El LogonAction procesa el usuario/password con la ayuda del Session Bean (ver diagrama de componentes) y determina que el usuario está autorizada para acceder a la aplicación. Le envía a Struts una respuesta positiva y entonces se le muestra el menú principal de la aplicación al usuario (ver más adelante el significado de un "forward success"):

```
return (mapping.findForward("success"));
```

c) Dentro del menú principal el usuario "disparará" otros eventos (para nuestro ejemplo el menú está vacío), y en un cierto momento decidirá salirse de la aplicación. Dentro del menú habrá un "link" para salir del sistema. Este request de "logoff" lo recibe Struts y de acuerdo a su archivo de configuración, envía el request a LogoffAction para que lo procese.

d) El proceso de LogoffAction, realiza labores de limpieza de la aplicación y deshabilita la sesión del usuario. También le indica a Struts cuál es la siguiente pantalla a donde se debe transferir al usuario.

e) Si el usuario/password introducidos no son correctos, se devuelve al usuario a la pantalla de logon.

f) Si el usuario excede el número de intentos permitidos se le ofrece un mensaje de ayuda.

g) Dentro de la pantalla de ayuda se le da la opción de regresar a la pantalla de inicio.

Cada transición mostrada en el diagrama anterior, es una transición configurada en el archivo struts-config.xml. En otras palabras, las pantallas (jsps) y los Actions, siempre interactúan con Struts de por medio. Así, cuando LogonAction, determina que el usuario está autorizado para acceder a la aplicación, envía a Struts un mensaje de "success", y este mensaje lo interpreta Struts, como que hay que transferir el control a menuPrincipal.jsp.

Todas estas interacciones están configuradas mediante xml, en el archivo struts-config:

```
...
<forward name="success" path="/menuPrincipal.jsp"
redirect="false" />
<forward name="logon" path="/logon.jsp" redirect="false"
...
    <action     path="/logon"
                type="org.apache.struts.example.LogonAction"
                name="logonForm"
               scope="request"
               input="/logon.jsp">
    </action>
...
```

En este ejemplo, los tags: success y logon, indican que cuando struts reciba como respuesta de los Action alguno de estos mensajes, deberá transferir el control a la página indicada en el path.

El tag de Action, declara que cuando se reciba un request de tipo "/logon" deberá generar un objeto Java de tipo "logonForm" (configurado antes en el mismo archivo) y transferir el control a la clase LogonAction.

LogonAction entonces, podrá acceder a los datos que introdujo el usuario en la pantalla de logon.jsp y podrá transferir el control hacia "success" o hacia "logon", de acuerdo a la lógica del negocio.

El código completo de este ejemplo se encuentra en el fólder básicos/ del código que acompaña esta Tesis.

Resumen

Es necesario dividir una aplicación transaccional en pequeños bloques de acuerdo a los diferentes eventos del negocio.

Los eventos del negocio pueden ser las acciones que "dispara" un usuario, como por ejemplo: Cuando el usuario se firma en el sistema, cuando el usuario consulta un saldo o cuando un usuario selecciona un artículo para guardarlo en su carrito de compras.

Otro tipo de eventos son aquellos que son disparados por el tiempo. Por ejemplo: El envío de totales al final de la operación al responsable de la aplicación o la operación de movimientos almacenados para ser operados durante la noche.

Cada evento, puede generar una o varias respuestas programadas. Cada una de estas respuestas consiste en datos que se le muestran al usuario o datos que se insertan o modifican en la base de datos.

Para aplicaciones web transaccionales, cada uno de estos eventos se mapea a una transacción o servicio del sistema, como por ejemplo la validación del usuario para ingresar al sistema.

El patrón de diseño que se utiliza para codificar estos servicios y la interacción del usuario con el sistema es el conocido como MVC (Model-View-Controller) el cual separa las tareas aplicativas en componentes especializados.

Una función del sistema separada de esta manera parece muy compleja, pero esta complejidad no surge del patrón de diseño utilizado, sino de la naturaleza de las aplicaciones web.

Esta complejidad se debe a que en una aplicación web no se tiene control acerca del flujo que seguirá el usuario dentro de las pantallas que componen la aplicación. Resultando en que este flujo es totalmente dinámico.

Para controlar esta complejidad, es necesario utilizar una herramienta que nos permita "mapear" todas las posibles peticiones del usuario a componentes especializados en atender cada una de estas peticiones.

El framework Struts es la herramienta más utilizada para efectuar esta función. Ya que permite controlar el flujo en un archivo de configuración y no dentro del código. Si el flujo se programara en código sería aún más complejo dar mantenimiento a este tipo de aplicaciones.

En el próximo capítulo veremos diferentes tipos de interacciones que puede haber con el usuario al desarrollar una aplicación y se podrá apreciar que la aplicación básica cubre todas estas posibilidades sin agregar mayor complejidad que la hasta aquí vista.

Capítulo 5. Modelos básicos de programación

Transacciones básicas y su implementación

Una vez que una aplicación de negocios ha sido dividida en transacciones y para cada una de estas se ha definido el flujo de ejecución deseado es fácil convertirlo a código.

Los factores más importantes en el desarrollo de este tipo de aplicaciones son:

a) La codificación de los "servicios" y el acceso de estos a los datos.

b) El control del flujo en la interacción con el usuario.

Servicios y Acceso a datos

A cada función del sistema se le conoce como un servicio. Este servicio se codifica en un Javabean o en un Enterprise Javabean. Un servicio se distingue porque recibe la información completa que necesita para producir un resultado. Este proceso se ejecuta de manera incondicional.

Un ejemplo de servicio es: Consulta de Saldo. El servicio de "Consulta de Saldo" recibe un número de cuenta y hace lo necesario para regresar el Saldo. Para ello se auxilia de clases auxiliares que acceden a los datos.

La suma de todos los servicios de una aplicación es lo que se conoce como: Las Reglas del Negocio. El patrón básico para implementar esto se conoce como Session Façade. Se le conoce así porque se utiliza básicamente un Session Bean el cual actúa como un frontend para efectuar el servicio y detrás de él se encuentran los Entity Beans que acceden y modifican los datos según sea necesario. El Session Bean actúa como un director de orquesta y los Entitys son los obreros que hacen el "trabajo sucio".

Esta separación, entre la interacción con el usuario y el servicio en sí mismo es muy importante. Porque el servicio en sí y la interacción con los usuarios son en la práctica totalmente independientes uno de otro. Y eso permite atender diferentes tipos de interfaces con los usuarios, compartiendo Reglas del Negocio.

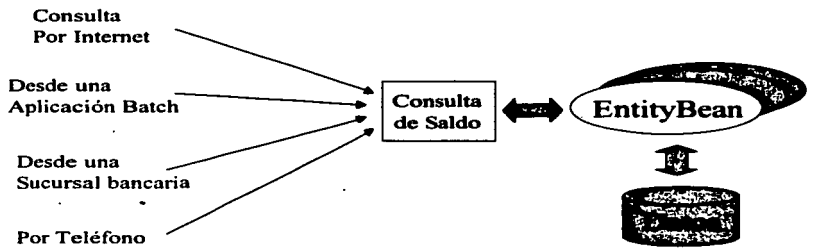


Figura 30 Separación de la interfaz del cliente y el servicio proporcionado por la aplicación.

Para generar estos Servicios y sus clases de apoyo para acceso a los datos hay 2 opciones nuevamente:

a) Utilizar un framework de desarrollo comercial.

Estos frameworks son costosos pero permiten ahorrar muchas horas de codificación. Uno de mis favoritos es: Business Components for Java de Oracle (BC4J). Este framework genera automáticamente el acceso a base de datos mediante "vistas" (queries de sql) y provee un mecanismo simple para generar los servicios encapsulando el acceso a datos (conocidos aquí como Application Module).

b) Codificación manual con el auxilio de herramientas libres.

Es este rubro existen varias herramientas, las más populares son:

Xdoclet: permite generar Enterprise Javabeans mediante tags "inteligentes". Se codifica solamente el Bean de implementación y esta herramienta genera el código de las interfaces. Combinada con Ant, permite automatizar también el proceso de compilación y empaquetamiento de los componentes para diferentes contenedores J2EE y diferentes bases de datos.

JDO: Java Data Objects, no es propiamente un framework de desarrollo de "servicios" pero es muy útil ya que genera también automáticamente el acceso a datos. Es una opción muy útil cuando no se desea utilizar Enterprise Javabeans.

La diferencia entre uno y otro enfoque es la interfaz gráfica. En la primera opción la programación es mediante "Wizards" en ambiente gráfico. En el segundo caso, se trata de herramientas que trabajan totalmente en "modo texto". Lo cual obliga a codificar manualmente ya sea el código o los archivos de configuración para generar el código. Sin embargo, partiendo de que se tiene una estructura clara y de aplicación general para las aplicaciones, con un poco de experiencia el proceso se vuelve "rutinario" e incluso llega a ser mucho más poderoso que la interfaz gráfica. Ya que se tiene control absoluto sobre la generación del código y es fácil hacer modificaciones simples ya que el código se recompila,

empaqueta e instala de manera automática. Cosa que no resulta con la interfaz gráfica: Una vez que se modifica algo, se deben repetir los 12 ó 20 clicks necesarios para "reajustar" la aplicación completa. Y esto a primera vista pareciera no ser un gran problema, pero dada la naturaleza del software, al estar implementando constantemente cambios en el código, el repetir esta secuencia de 12 ó 20 clicks es muy propenso a errores.

Mi recomendación simple, escalable y universal: Session Beans con Entity Beans. Los Session Beans inician como Stateless beans y si algún requerimiento de negocio nos obliga, se convierte a un Statefull bean. Los Entity Beans siempre son de tipo CMP (Container Managed Persistence) y las relaciones entre entidades se controlan mediante CMR (Container Managed Relationships). La transaccionalidad es siempre "required" y de tipo CMT (Container Managed Transaction).

El código se genera con el auxilio de Ant y Xdoclet. Es importante recalcar que los Enterprise JavaBeans deben ser de acuerdo a la versión 2.0 de la especificación (la actualmente soportada por casi todos los contenedores).

Hasta hace algunos meses había mucho debate sobre la conveniencia de utilizar o no los EJBs, pero con las mejoras introducidas en EJB 2.0 y con las mejoras introducidas en los contenedores actuales, el debate está prácticamente terminado. La única razón por la cual alguien codificaría sin EJBs, sería solamente por desconocimiento de los mismos. Aunque también otro factor que ha influido es el costo de las herramientas para desarrollo e implementación de los EJB's. Casi todos los proveedores de herramientas y contenedores regalan la versión de su herramienta que soporta solamente componentes Web. Mientras que el producto que soporta componentes EJB lo venden en rangos de precios que inician en los 1000 dólares por licencia.

Pero cada día surgen opciones. Mientras escribo esto, la opción para desarrollar EJB's es la mencionada 2 párrafos arriba. Y para instalarlos y ejecutarlos es Jboss 3.0.

Control del Flujo

Una vez que se ha decidido como implementar los servicios y como se accesarán los datos y como construir dichos componentes, entonces se puede resolver la otra mitad del problema: La interacción con el usuario.

El caso general aplicado en el capítulo 4, consistió en descomponer la aplicación en Request's y Response's. Esto siguiendo la naturaleza propia del protocolo http que es el utilizado por las aplicaciones web. Para cada Request hay un Action, el cual aplica la Lógica del Negocio y determina cual será el Response. Normalmente a su vez, dentro del Response se le dan opciones al usuario que generarán un Request nuevamente y el ciclo continúa indefinidamente.

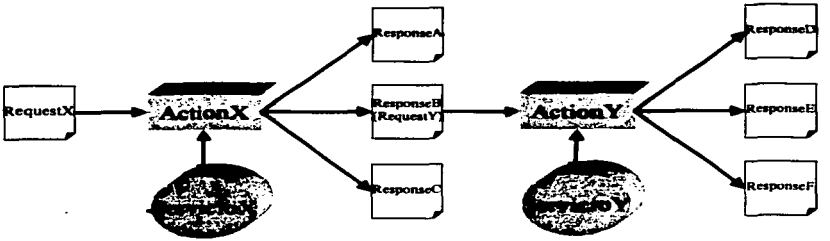


Figura 31 Navegación entre acciones del usuario (Request) y acciones del Sistema (Action - Servicio - Response).

Para simplificar la descripción de este flujo, lo mejor es utilizar el framework de Struts. Este framework incluye algunas otras facilidades para la generación de Formas de HTML y el acceso a bases de datos. Es decisión de cada proyecto utilizar estas facilidades o no, pero el utilizar el framework del control de flujo es una necesidad.

El siguiente paso es describir el flujo de la aplicación en el archivo de configuración struts-config.xml. Después se pueden construir el resto de los componentes prácticamente en cualquier orden:

a) Las páginas JSP's junto con sus respectivas clases Form ó una sola clase Form que contenga todos los datos de las JSP's relacionados.

b) Las clases Action.

c) Los Servicios y los Value Objects.

Cada uno de estos componentes se puede probar por separado y su desarrollo es independiente uno de otro. Las únicas coincidencias son las siguientes:

Entre las páginas JSP y los Action la interfaz es mediante las clases Form. Las clases Form contienen los datos que introduce el usuario y también los que responde el sistema.

Entre los Action y los Servicios están los Value Objects, que son objetos que encapsulan los parámetros necesarios para efectuar un servicio. Los Action son los encargados de tomar los datos de los objetos Form y convertirlos en un Value Object. Una vez que el Value Object está completo se transfiere al Servicio. También mediante Value Objects el servicio devuelve los datos que se enviarán al usuario o a otros servicios conectados por el mismo Action.

La siguiente sección muestra los diferentes tipos de interacciones entre las Páginas JSP (la vista) y los Action sin un orden en particular. Se sobreentiende que los Action acceden a un objeto de Servicio para determinar la respuesta al usuario y que de por medio entre todos ellos van los objetos Form y los Value Objects.

Diagramas de interacción

Transacción de consulta

Esta interacción se caracteriza porque el usuario consulta una gran cantidad de información dependiendo de un criterio de búsqueda o seleccionando categorías en un árbol. El problema consiste en permitir al usuario "paginar" a través de la información de tal manera que pueda avanzar o retroceder a libertad. Esta navegación puede ser página por página o libre renglón por renglón. Lo más simple y eficiente es introducir un objeto de "Control de Páginas". Este objeto será el encargado de almacenar las páginas ya "vistas" por el usuario. De tal manera que cuando el usuario solicita una página primero se verifica si esta página existe en memoria, de no ser así se solicita al Servicio una nueva página. Esta nueva página se almacena y se presenta al usuario. Opcionalmente se le puede dar al usuario la facilidad de definir el tamaño deseado de sus páginas. La ventaja es que un solo objeto de "Control de Páginas" puede servir para todo tipo de datos y/o vistas de datos. En este caso es una sóla página de "navegación" con un solo Action.

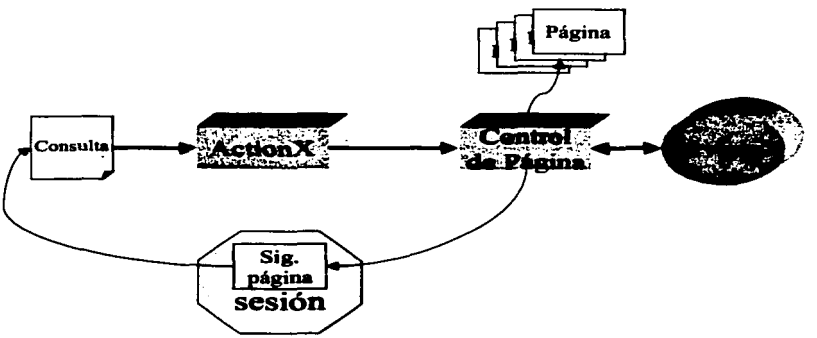


Figura 32 Diagrama Model View Controller para una Consulta con múltiples páginas de información.

Transacción de reservación

Este tipo de interacción se caracteriza porque el usuario debe tomar varias decisiones a lo largo de diferentes pantallas. Tómese como ejemplo una reservación de vuelo. El enfoque aquí, es tomar cada pantalla como una fracción de una pantalla virtual que incluye toda la información que el usuario debe capturar para hacer la reservación. En este caso varias pantallas son procesadas por un solo Action que normalmente integra muchos Servicios (Consulta de Vuelos, Consulta de Tarifas, Consulta de Disponibilidad, Generador de Itinerarios, Registro de Reservación, Alta del Cliente, etc.).

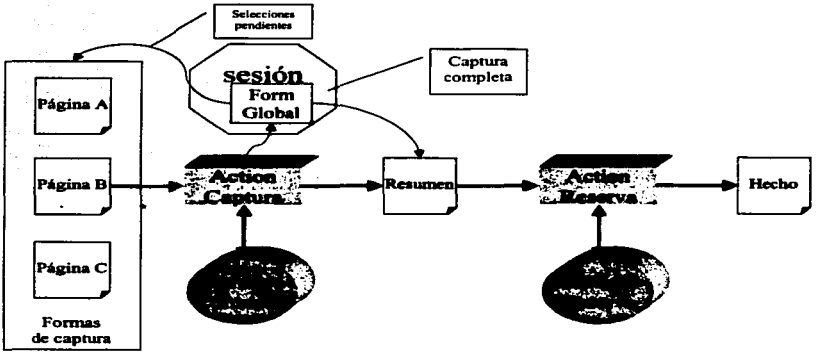


Figura 33 Diagrama Model View Controller de una transacción de Reservación.

Transacción de alta

Este es el caso más simple de captura de información. Una pantalla de captura y despliegue de errores y otra pantalla de confirmación de la alta. Solo se requiere un Action.

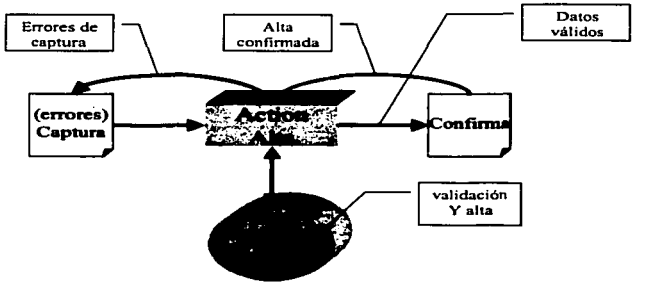


Figura 34 Diagrama Model View Controller de una transacción de Alta.

Transacción de intercambio

Un ejemplo de transacción de Intercambio es el Traspaso de Fondos en una operación bancaria. El requisito es que la transacción primero se "simule" antes de llevarse a cabo. Por ejemplo, para realizar el Traspaso de Fondos, se requiere primero "bloquear" o "reservar" el monto del retiro y después "bloquear" la cuenta de depósito. Una vez que ambas operaciones se han efectuado con éxito entonces se procede a efectuar el traspaso opcionalmente con la confirmación del usuario. Se requieren 4 páginas y 1 Action. Este modelo es útil para integrar Transacciones que normalmente son independientes (en nuestro ejemplo, las Transacciones de Retiro y Depósito). También se requiere que la transacción se controle en el Servicio, es decir tendrá que ser un Statefull session bean con Bean Managed Transaction.

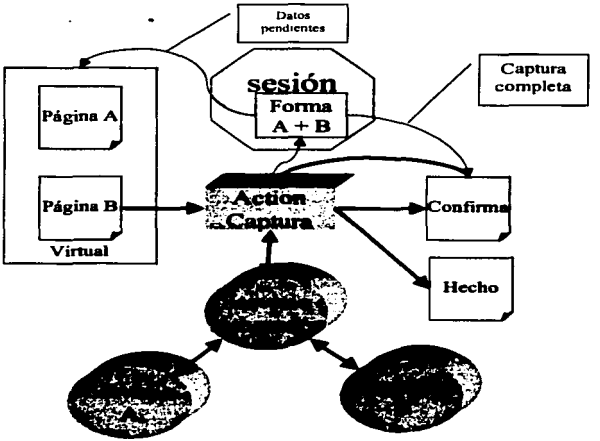


Figura 35 Diagrama Model View Controller de una transacción de Intercambio.

Transacción de cambio

Similar a la Transacción de alta, con la diferencia de que aquí previamente se eligen los datos a modificar.

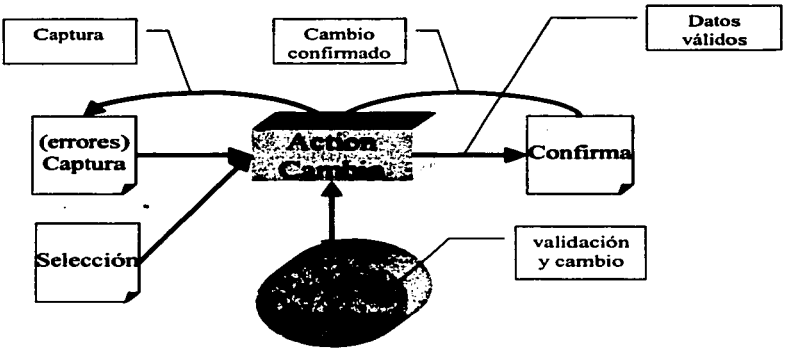


Figura 36 Diagrama Model View Controller de una transacción de Cambio.

Transacción de baja

La transacción más simple de todas. Se selecciona la información y simplemente se confirma la operación.

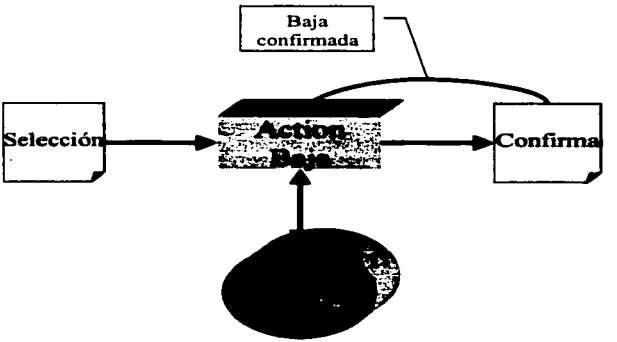


Figura 37 Diagrama Model View Controller de una transacción de Baja.

Estos modelos, una vez que se dominan, dan la impresión de ser el mismo todos ellos. La diferencia está en la codificación del Action. La tendencia natural de desarrollo de software es construir un solo componente que maneje la lógica de Alta-Bajas-Cambios. Esto lo que provoca es un código similar al siguiente:

```
If (operacion.equals("alta")) {
    //codigo de alta
} else if (operacion.equals("cambio") {
    //codigo de cambio
} else if (operación.equals("baja") {
    //código de baja
} else {
    //codigo de operación invalida
}
```

Luego, al hacer esto, se continúa la tendencia tratando de factorizar cada línea de código de tal manera que aplique lo mejor posible para las 3 operaciones. Obviamente en cada línea de código que esté compartida, habrá que incluir una secuencia de "if's" similares, porque también normalmente las reglas y validaciones que aplican en una Alta son diferentes a las de una Baja, por ejemplo.

Esta forma de codificación hay que evitarla al máximo. Y la mejor forma de evitarlo es teniendo código independiente que controle el flujo para cada operación. El código que en todo caso puede estar compartido es el del servicio. Pero aún en ese caso el servicio contará con métodos específicos para soportar la operación indicada.

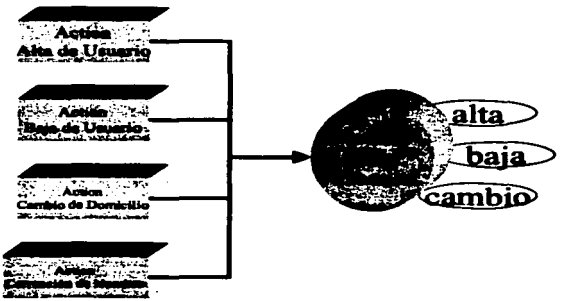


Figura 38 Diferentes servicios son atendidos por diferentes componentes Action.

Nunca es suficiente recalcar que, entre menos "if" existan en el código, la programación se simplifica y el mantenimiento se facilita.

Otra tendencia al "compartir" código, es integrar todas las funciones en una sola pantalla. Esta tendencia es ampliamente aceptada, salvo por el hecho de que se generan páginas multifuncionales las cuales pueden hacer compleja en sí la página, pero simplificar la interfaz para el usuario, ya que sólo debe "aprender" a utilizar una pantalla. Esto dependerá de las preferencias del diseñador, pero hay que tener cuidado de nunca "compartir" los Actions. En otras palabras, una sola página, puede permitir disparar distintos tipos de Request's, pero por cada Request, sólo existirá un Action.

Sólo basta agregar que estos 6 modelos de interacción se combinan para generar una aplicación completa. Generalmente se inicia con una Consulta, de la cual se continúa con una Alta, Baja ó Cambio. En otras ocasiones se integran varias Consultas, para componer una Reservación. Y finalmente, cuando las aplicaciones van evolucionando y se hace necesario integrar varias operaciones en una sola, se llega a la Transacción de Intercambio.

En el próximo capítulo se verá un método simple de cómo descomponer una Aplicación en Transacciones.

Capítulo 6. Metodología de Análisis y Diseño básica

Componentes genéricos de toda metodología de análisis y diseño

Construir aplicaciones transaccionales de negocios es un problema viejo. No así el construir aplicaciones para Internet. Sin embargo, construir aplicaciones en general se basa en 3 puntos básicos (y lógicos) para construir cualquier software:

a) ¿Qué se espera que haga el software y cuando?

b) ¿Cómo es que lo hará?

c) ¿Cuál es la mejor forma de distribuir la complejidad en componentes más simples?

La primera pregunta la responde la fase de Análisis. La segunda, el Diseño. La tercera, el Diseño Detallado y es generalmente soportada por estándares de diseño y construcción, aunque dependiendo de la metodología utilizada el Diseño General y el Diseño Detallado se engloban en una sola fase conocida como Diseño.

Sin importar el lenguaje en el que se construirá la aplicación, ni el tipo de programación que se utilizará, sea esta Orientada a Objetos o Estructurada, es más o menos simple darse cuenta que la fase de Análisis debiese ser "universal". Es decir, que debe responderse la pregunta a "¿Qué es lo que se espera que haga el software...?" de una manera que no dependa de la forma en que lo hará o se construirá.

Sin embargo, en múltiples metodologías se puede apreciar, que siempre se recomienda una forma de efectuar este análisis.

Es por esto, que el camino recomendado es tomar una metodología de Análisis por sí sola. Es decir, una metodología exclusivamente para análisis. Y después enfocarse en una metodología de Diseño de acuerdo al tipo de sistema que se construirá.

Una metodología simple de Análisis, pero poderosa, permite entender los pasos esenciales del Análisis de una manera que pueda ser sencilla de llevarse a la práctica y con objetivos claros de cuales son los productos tangibles que se desean obtener. Este último concepto: "tangible", es indispensable para un buen análisis. Normalmente en los proyectos se tiene la idea que el Análisis es una fase tediosa de "simple" documentación la cual servirá para "justificar" el desarrollo de un sistema y "dar mas ó menos" la idea de lo que se desea construir. En otros casos la fase de Análisis, se entiende como una etapa en la cual se desarrollan "entrevistas" con los usuarios para entender "qué es lo que quieren que haga el sistema". En ambos casos esta idea es equivocada.

Un Análisis completo, debe responder con precisión cuáles son las funciones que debe ejecutar un sistema, cuando las debe ejecutar y quién las puede ejecutar. El nivel de detalle de esta respuesta es al nivel de Entidad de Datos. Ya que es muy importante que en la fase de análisis se determine cuales son los datos de entrada, los que se almacenan y cuales de salida.

La forma más rústica de entender esto, es regresando al concepto básico de proceso. Se entiende que un proceso tiene una entrada, una transformación (el proceso) y una salida. Si visualizamos un sistema como un conjunto de procesos, entonces tendremos que para tener un análisis completo debemos utilizar una metodología que nos permita responder cuáles son las entradas, salidas y transformaciones necesarias para que la aplicación sirva de soporte al negocio. Esta metodología deberá además permitir organizar de una manera sencilla toda esta información.

Visto desde esta perspectiva, tendremos que un sistema se compone de:

a) Las respuestas que se esperan del sistema

b) Los datos

c) Los procesos

d) Las interacciones datos – procesos

Las respuestas que se esperan del sistema, son la parte "visible" del sistema. Es decir, son las pantallas, reportes y archivos que generará el sistema. Para poder producir estas salidas, el sistema depende de datos proporcionados por el usuario, así como de datos almacenados en el sistema y otros generados por él mismo. Obviamente, el manejo de todos estos datos requiere de procesos. Finalmente existe una interacción entre los datos y los procesos la cual es necesario "mapear" para poder verificar que nuestro análisis esté completo. Esta verificación consiste en asegurarse que no haya datos sin su proceso correspondiente y viceversa procesos sin los datos adecuados.

Así, de esta manera, llegamos a que una metodología de análisis debe responder a la pregunta de ¿qué hará el sistema? De tal manera que establezca con precisión cuáles son: las respuestas del sistema, los datos necesarios para generar dichas respuestas, los procesos que generarán dichas respuestas y como se conjugan datos y procesos para generar una respuesta específica. El lograr un análisis completo del sistema que se desea desarrollar permite hacer un diseño con facilidad.

Lo importante aquí es distinguir que sin análisis no se puede diseñar y sin diseño no se puede construir software. Esto parece simple, pero es sumamente difícil llevarlo a cabo. En la práctica en el 99% de los casos se parte de un análisis incompleto, lo cual lleva a un mal diseño y por consecuencia una mala construcción del sistema.

Sobre el tema existen múltiples libros, metodologías, recomendaciones y la gran mayoría de ellos se centran en cómo hacer el análisis de complejísimos sistemas. En la práctica el 80% de las aplicaciones de negocios son aplicaciones simples de altas, bajas y cambios. Y los problemas complejos generalmente se reducen a:

a) ¿Cómo autenticar usuarios?

El cuál se resuelve con el esquema de Usuario-Password normalmente.

b) ¿Cómo soportar concurrencia?

El cual lo resuelve Java con la arquitectura multicapa y con el soporte de una buena base de datos multiusuario.

c) ¿Cómo soportar transacciones?

Nuevamente Java al rescate: Enterprise Javabeans.

En la práctica, resolviendo estos 3 problemas se tiene el 99% de las posibilidades de dificultades que encontraremos para construir un sistema. Por supuesto, que también en la

práctica se presentarán otros problemas, pero el dominar los conceptos básicos del Análisis y comprender los patrones básicos de Diseño de aplicaciones nos permitirá evolucionar hacia otros niveles del desarrollo de software.

Una metodología que cumple con los requisitos antes mencionados es la conocida como Análisis de Requerimientos basado en Eventos. Esta metodología además resuelve otro problema común en el desarrollo del software: Los usuarios NO saben que sistema es el que necesitan, mucho menos el que quieren. Este problema se resuelve dividiendo primero el problema en Eventos, de tal manera que cada Evento se puede desarrollar independientemente de los otros. Lo cual da una mayor flexibilidad en el desarrollo, ya que se pueden paralelizar el trabajo donde cada línea de desarrollo corresponde exclusivamente a 1 evento.

Análisis de requerimientos basado en eventos

El objetivo de esta metodología es representar un sistema con 4 modelos básicos:

A) El modelo de comportamiento

B) El modelo de datos

C) El modelo de procesos

D) El modelo de interacciones (datos-procesos)

Para ir ejemplificando, el uso de cada uno de estos modelos, haremos simultáneamente el proceso de análisis de la aplicación de ejemplo de esta tesis. La aplicación de ejemplo consiste en una aplicación de Control de Horarios Escolares.

La aplicación consiste en un sistema, que por un lado permita capturar las materias y horarios disponibles y por el otro a los alumnos elegir las materias en las que estará inscrito en un ciclo escolar.

El sistema mantiene un orden de precedencias, de tal forma que dependiendo del promedio general, los alumnos tienen precedencia en la elección de sus materias. Cuando un alumno con un promedio más bajo que el de otros alumnos intenta inscribirse a una materia, su petición es encolada, hasta que todos los alumnos, con mejor promedio que él, hayan confirmado sus preferencias. De esta manera, se garantiza que los alumnos con mejor promedio tengan total libertad de elección en sus horarios.

El administrador del sistema, podrá ver cuáles son las materias con su matrícula completa, las que están en espera de confirmación de aceptación y así como las materias elegidas por los alumnos. El administrador también procura que los alumnos escojan materias que no se "encimen" en el horario. El sistema puede ser tan complejo como se quiera, pero en este caso algunas de las posibles extensiones que NO están contenidas en este ejemplo son:

a) Soporte de materias encadenadas, es decir, validar que el alumno cumpla con ciertos prerrequisitos para tomar cierta materia.

b) Requerir la autorización de la solicitud de inscripción por parte del Profesor o Supervisor de la materia.

c) Soporte de listas de espera.

d) Notificaciones de aceptación vía email a los solicitantes.

... entre otras.

La estructura general del análisis a desarrollar se muestra en el siguiente diagrama:

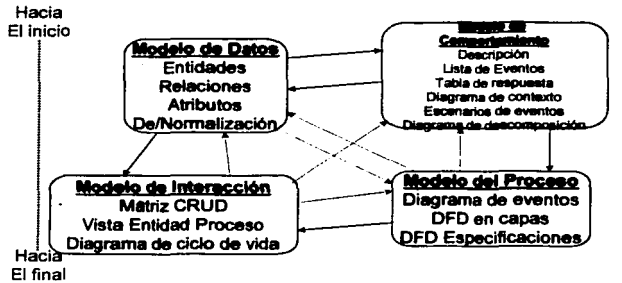


Figura 39 Elementos del Análisis de un Sistema.

Las líneas continuas representan el flujo de la primera iteración de análisis, las líneas punteadas representan las subsecuentes iteraciones. Las iteraciones son necesarias ya que la información que se va obteniendo durante el análisis no corresponde necesariamente a la estructura del modelo que se espera obtener. Así, que durante la fase de análisis se obtiene un poco de información acerca de los datos, luego otro poco de cómo se procesan, luego otro poco de cuáles serán las acciones de los usuarios, etc. El modelo se va completando poco.

Por otro lado, el sistema primero se particiona en Eventos, luego cada Evento es un segmento totalmente independiente de los otros. De tal forma, que este proceso de Análisis se vuelve un proceso compuesto de varios subprocesos, es decir el Análisis total se compone de la suma de los Análisis independientes de cada Evento.

Aquí el problema que surge es: ¿Cómo determino que ya terminé la fase de Análisis?

Para responder esta pregunta es necesario delimitar el alcance del sistema. Y la mejor forma de delimitarlo es enumerando cuáles serán los Eventos del Negocio que soportará este sistema. Dado que el alcance del sistema está delimitado por Eventos, y además de que toda la información recabada se organiza por Evento, es que esta metodología se conoce como basada en Eventos.

Entonces, el primer paso es establecer la lista de Eventos. Esta lista forma parte de lo que se conoce como Modelo de Comportamiento y por lo tanto es el primero modelo a construir de nuestro sistema.

Una vez que se tiene la lista de Eventos, se procede a describir cuáles serán los datos que manejará el sistema y que son necesarios para producir las respuestas que se esperan del sistema (Modelo de Datos). Para producir estas respuestas será necesario hablar de Procesos. Al final se genera un modelo para verificar que esté completo nuestro Análisis y este se conoce como el Modelo de Interacciones.

Obviamente, no hay una secuencia estricta a seguir para desarrollar todos los modelos. Normalmente se parte de la lista de Eventos, y después cada Evento se trata como si fuese un subsistema o módulo totalmente independiente. De tal forma, que mientras en un Evento se están describiendo los datos, en otros es posible estar trabajando en los procesos o incluso con las interacciones.

Y esta es la ventaja de esta metodología, ya mencionada, que permite dividir el trabajo y paralelizar el trabajo si es que hay los recursos disponibles.

El Modelo de Comportamiento

En este modelo definimos el sistema como un conjunto de cajas negras, en donde estableceremos cuál será el detonador del proceso y las respuestas que se generarán pero sin entrar en detalle de cómo es que se generarán.

El modelo se forma de:

- Lista de Eventos
- Tabla de Respuestas
- Diagrama de Contexto
- Escenarios
- Estructura Funcional

Lista de Eventos

Esta lista se obtiene de sesiones de trabajo (o entrevistas) con los usuarios. En estas sesiones se obtienen los eventos significativos para el negocio.

Puede haber 2 tipos de eventos:

a) Eventos disparados por el usuario: como por ejemplo, cuando el alumno entra al sistema y consulta cuales son las materias disponibles para cierto horario.

b) Eventos disparados por el tiempo: por ejemplo, cuando se cumple la fecha límite de inscripción para cierta categoría o rango de alumnos.

Aquí es importante definir qué es un Evento. Cuando nos referimos a un Evento, en realidad hablamos de un Evento de Negocio, y nos referimos a las interacciones que habrá entre los usuarios del sistema y el sistema, así como los cambios externos que pueden afectar a nuestro sistema. Normalmente para efectos de análisis, sólo se consideran los eventos externos "disparados" por el tiempo, como podrían ser: El cumplimiento de una fecha límite, o la llegada de una fecha especial como las quincenas o los fines de mes, etc. Pero por supuesto que puede haber otros eventos externos que afecten el sistema, como por ejemplo: la llegada de un archivo a cierto directorio de cierto servidor, la llegada vía email de un correo, el estado crítico de algún recurso de la plataforma, como por ejemplo la memoria, el cpu, etc.

Es tarea del Analista, con base en su experiencia y su conocimiento del negocio en cuestión, distinguir cuáles de todos los eventos posibles, son los eventos relevantes del negocio. Ya que puede darse el caso que se detecten numerosos eventos, los cuales no son de relevancia para el negocio (no lo afectan) o que no correspondan a eventos que deban ser automatizados.

Siguiendo nuestro ejemplo de aplicación, la siguiente es la lista de Eventos soportados por esta versión:

#		
1	Solicitud de Inscripción	El alumno consulta las materias disponibles y solicita su inscripción a ellas.
2	Alta de Materia	El administrador inserta una nueva materia disponible para inscripciones.
3	Consulta de Status Materias	El administrador consulta el número de inscripciones aceptadas y pendientes x materia.
4	Consulta x Alumno	El administrador consulta las materias a las que se ha inscrito un alumno.

Solo tenemos 4 eventos "significativos" para este sistema. Que consiste básicamente en el ciclo de vida de una materia: Alta, inscripciones, consulta de status. Esta funcionalidad es la más básica pero es suficiente para ejemplificar el uso de esta metodología.

Tabla de Respuestas

El siguiente paso es definir la tabla de respuestas. Esta se construye a partir de la tabla anterior. Pero ahora, debemos agregar la información acerca de qué respuestas se esperan del sistema para cada evento. Una respuesta puede ser: un mensaje en pantalla, un reporte, un archivo o la inserción o modificación de datos en archivos del sistema o en una base de datos.

La siguiente es la tabla de respuestas para nuestra aplicación:

1	Alumno	Solicitud inscripción	Registro de solicitud	Asignación de clave de inscripción	Alumno
				Alta de solicitud	Base de datos
				Proceso de solicitud	Base de datos
2	Administrador	Nueva Materia	Registro de materia	Asignación de clave de materia	Administrador
				Alta de la materia	Base de datos
3	Administrador	Consulta x Materia	Status de materia	Resumen de solicitudes por status	Administrador
4	Administrador	Consulta x Alumno	Status de Alumno	Resumen de materias por alumno por status	Administrador

En esta tabla, el número de evento relaciona quien dispara el evento, que tipo de información es la que dispara, la respuesta que se espera y las principales salidas que generará la aplicación. Se identifican los destinos externos, que en este caso son los mismos usuarios del sistema y la base de datos de la aplicación. En otros casos podría ser un servidor de email, por ejemplo para las notificaciones de aceptación, u otro sistema, por ejemplo, cuando se trata de pedidos y se debe notificar al sistema de inventarios, de envíos, de marketing, etc.

Diagrama de Contexto

El siguiente paso en nuestra lista es un Diagrama de Contexto. Este diagrama nos servirá para identificar con qué sistemas y qué perfiles de usuarios tendrán interacción con el nuevo. También nos permite identificar, cuales son los eventos y las respuestas por las cuales se relacionan:

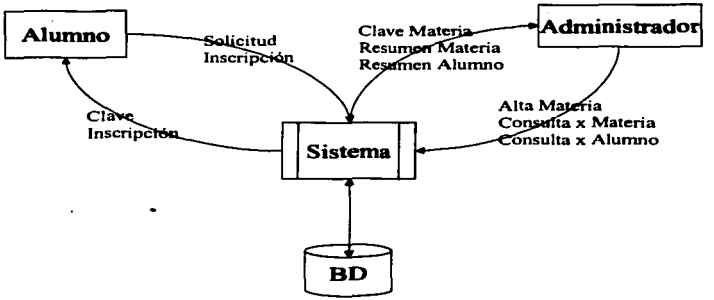


Figura 40 Ejemplo de Diagrama de Contexto.

Este diagrama es importante, porque nos ayuda a distinguir todas las interacciones que tendrá el sistema con cada "Entidad" externa (sistemas o usuarios) y para cada Entidad, nos permite ver cuáles son esas interacciones. Esto nos permitirá entender mejor cuáles son las respuestas esperadas por cada entidad y el mejor medio para hacérselas llegar.

Por ejemplo, uno de los requerimientos es que el alumno pueda inscribirse desde su casa vía Internet. Entonces, la aplicación no deberá ser muy cargada de gráficos y adornos, para que vía modem la aplicación sea "rápida". En cambio, el administrador podrá dar de alta y consultar los resúmenes desde la red local del sistema, esto permitirá crearle una interfaz más amigable y estética dado que dispondrá de un ancho de banda mayor. Por otro lado las funciones del administrador requieren mayor seguridad que la que se puede obtener por una conexión vía Internet.

Escenarios

Escenario 1
Solicitud de Inscripción

El alumno se conecta a la página de inicio de inscripción. Si no se ha registrado en la página se le solicita su número de cuenta y password. Una vez ejecutada esta fase se pasa a la página de lista de materias. En esta página se le muestran las materias disponibles para su inscripción. Opcionalmente se encuentran resaltadas aquellas en las que tiene "prioridad" de inscripción. Selecciona una de las materias en las que se desea inscribir. Al seleccionar una materia, ve los horarios disponibles y datos adicionales como nombre del Instructor, Créditos, etc. En esta pantalla de detalle tiene 2 opciones: Solicitar su inscripción o regresar a la pantalla inicial. Si solicita su inscripción se le presenta el status de su solicitud, y cual es el lugar que ocupa su petición, después se le regresa a la página de lista de materias. Conforme se va inscribiendo el alumno va visualizando en una sección de la pantalla, las solicitudes que hasta el momento ha realizado. El alumno tiene un límite máximo de materias a las que se puede inscribir y un límite mínimo de créditos que debe cubrir.

Figura 41 Ejemplo de una descripción de Escenario.

Este es un ejemplo de Escenario. Un escenario es una descripción gráfica o en modo texto, de cómo será utilizado el sistema, en este caso específicamente para realizar la solicitud de inscripción a una materia. De la descripción anterior se pueden distinguir algunos puntos:

a) Los escenarios también se les conoce como "Casos de Uso", los cuales a su vez pueden participar de escenarios compuestos, donde se mezclan con otros "Casos de Uso".

b) Se vuelve difícil entender como se delimita un Escenario, puesto que hay fragmentos que podrían (o que son) comunes a otros Eventos. Los escenarios nos permiten entender esta diferencia, ya que nos permiten conceptuar la interacción entre el usuario y el sistema, de tal manera que nos ayuda a definir el propósito de esta interacción. En este caso el objetivo es solicitar la inscripción de una o varias materias.

c) Por otro lado, también nos ayuda a distinguir algunos elementos que no necesariamente están en la descripción inicial del sistema, como puede ser, la necesidad de un control de usuario-password.

d) En este punto se debe evitar al máximo la tentación de "empezar" a diseñar la aplicación. Un comienzo precipitado de la fase de Diseño, provocará "prejuicios" que luego entorpecen el análisis, dado que tendemos a visualizar el sistema de acuerdo a nuestro "diseño preliminar" lo cual se puede volver un obstáculo para entender otros requerimientos. Es decir, tratamos de que todos los requerimientos subsecuentes "encajen" en nuestro "diseño preliminar".

e) Este ejercicio se repite para cada Evento. La persona responsable de la Calidad del Software, puede utilizar estas descripciones para diseñar el plan de pruebas del software.

Estructura Funcional

Esta se representa mediante un Diagrama de Descomposición. Este diagrama representa la funcionalidad del sistema (o mejor dicho, los eventos), de acuerdo a las áreas a las que esta dirigida dicha funcionalidad. Para nuestro ejemplo, tenemos 2 grupos de usuarios o áreas

claramente delimitados: Alumnos y Cuerpo Académico, sin embargo, todo sistema requiere también de un Administrador. Entre otras labores el administrador es el responsable de dar de alta información necesaria para la operación y de generar los usuarios y passwords necesarios para acceder al sistema. En nuestro sistema hasta el momento no se ha hablado de algún evento para identificar a los alumnos, este requerimiento tarde o temprano surge durante el análisis, ya que se necesita identificar qué alumno es el que está solicitando la inscripción y hay que asegurarse que el alumno es realmente quien dice ser.

Esta descomposición es importante, por 2 razones: Primero nos permite identificar las áreas del negocio relacionadas con el sistema, ya que habrá que definir un responsable por área para tomar decisiones de problemas o controversias que surjan durante el proyecto; por otro lado, también nos sirve para identificar Eventos que podrían caer en el ámbito de varios actores, por lo que habrá que asignar a alguno de ellos la responsabilidad sobre dicho evento. En nuestro caso, el cuerpo académico podría necesitar ver las solicitudes de inscripción de un alumno en particular, pero para nuestro caso, como dicha función será más utilizada por el propio alumno, se decidió dejar tal evento bajo el área de "Alumnado".

Para nuestro ejemplo, hay un área de usuarios que no tiene claramente una representación: Los Alumnos. Esto es frecuente en todo tipo de proyectos, y en dichas situaciones se vuelve muy importante el sentido común del Analista, ya que no tendrá a su disposición el tipo de usuario "típico" como para hacerle preguntas relacionadas con el sistema que se desea construir. De cualquier manera es importante que exista alguien con la autoridad para decidir acerca de decisiones que tengan que ver con esta área de usuarios.

Para nuestra aplicación el diagrama es el siguiente:

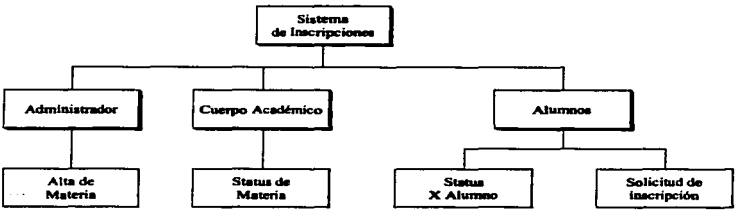


Figura 42 Diagrama de estructura funcional.

El Modelo de Datos

Entidades

De los 4 eventos de los que se compondrá nuestro sistema, se puede determinar que se requieren al menos las siguientes entidades:

- Materia: Representa una asignatura en la escuela.
- Horario: Representa un grupo con salón y horario establecido donde se imparte una materia.
- Alumno: Representa un alumno de la escuela.

- Solicitud: Representa el registro solicitado por un alumno a un horario.

Otras entidades no tan evidentes:

- Jerarquía Escuela-Área-Carrera-Materia: Representa la organización de una escuela por áreas y carreras. Para nuestro ejemplo esta jerarquía se omite.
- Prioridad: Representa el orden de preferencia de inscripción de los alumnos. El criterio básico sería asignar prioridad de inscripción dependiendo del promedio general del alumno y al número de créditos completados.
- Cuerpo Académico: Relación de Profesores y Horarios. Esta entidad sería necesaria en caso de que se requiriera la autorización de las solicitudes de inscripción por parte de los profesores. Para nuestro ejemplo, se omite.

Relaciones

En el diseño de bases de datos relacionales se distinguen los siguientes tipos de relaciones básicas:

Uno a uno

Uno a muchos

Muchos a muchos

De acuerdo al análisis relacional, las relaciones de Muchos a Muchos se deben descomponer a relaciones de tipo Uno a Muchos, mediante la agregación de una entidad de Relación. Todas estas relaciones se visualizan fácilmente mediante un Diagrama Entidad-Relación, el cual es el producto final de este modelo.

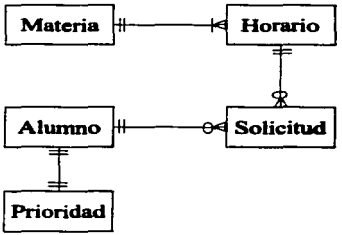


Figura 43 Diagrama Entidad Relación.

Atributos

En el mercado existen herramientas para hacer diagramas como el anterior, además de permitir definir en el mismo diagrama los atributos de cada entidad (es decir, los campos). Sin embargo, estas herramientas tienen precios prohibitivos y realmente no da muchas ventajas tener integrado el diagrama y la lista de atributos, excepto por la facilidad de poder generar los scripts de creación de la base de datos directamente de estas herramientas. Pero nuevamente las capacidades son limitadas, ya que solo soportan algunas bases de datos (principalmente las comerciales), y no permiten dar opciones avanzadas como el cálculo de espacios apropiado para cada entidad.

En este punto es preciso mencionar 2 consideraciones importantes para el desarrollo de una aplicación eficiente:

Por un lado, el Modelo de Datos de toda aplicación tiene 2 etapas, la primera el Modelo Lógico, y la segunda en Modelo Físico. El Modelo Lógico es responsabilidad del Diseñador, pero el Modelo Físico depende del Administrador de Base de Datos. Este rol es muy importante, porque el Modelo Físico incide directamente en el desempeño de la aplicación.

La segunda consideración es que, durante las etapas de Análisis y Diseño es común que los atributos cambien constantemente, tanto en su cantidad como en su definición. Por esto, lo recomendable es no tener la lista de atributos en el diagrama Entidad Relación, sino en una lista por separado que se puede manejar mucho mejor con una Hoja de Cálculo. En la actualidad es de mucho más ayuda una Suite de Aplicaciones de Oficina, como por ejemplo OpenOffice, que una herramienta muy sofisticada de diseño. En la Hoja de Cálculo, por ejemplo, es fácil revisar la nomenclatura de los atributos, organizar en columnas, el nombre lógico del atributo, el nombre físico, su definición lógica y su definición física, hacer ordenamientos por Entidad, por Nombre de Atributo, por Tipo, etc. Y todo esto es imposible de hacer en una herramienta de diseño, por lo menos no de una manera tan simple como lo es en una Hoja de Cálculo. Por otro lado, las Hojas de Cálculo poseen funciones de manejo de texto muy poderosas, con las cuales es posible generar los scripts de creación de tablas, hacer el cálculo de espacios y estandarizar nombres y formatos de los datos.

De hecho, muchos de los principales autores de libros de Análisis y Diseño, "confiesan" no utilizar ninguna herramienta especializada para diseño, sino simplemente Hojas de Cálculo y aplicaciones simples de Diagramación convencionales. Por otro lado, dado el alto precio de las herramientas de diseño, su costo-beneficio es también muy reducido, dada la cantidad de aplicaciones a desarrollar en que una persona se puede involucrar digamos en un año.

Una lista de atributos preliminar es como la siguiente:

Entidad	Nombre Lógico	Nombre Físico	NN	NB	ND	BG	Descripción	Tipo Lógico	Formato	Rango Lógico	Tipo Físico
Alumno	Apellido Materno	materno	X				Apellido materno	Texto			varchar(40)
Alumno	Apellido Paterno	paterno	X	X			Apellido paterno	Texto			varchar(40)
Alumno	Créditos Aprobados	creditosAprob	X				Créditos aprobados por el alumno	Cantidad positiva		>=0	integer
Alumno	Créditos No aprobados	creditosApr	X				Créditos no aprobados por el alumno	Cantidad positiva		>=0	integer
Alumno	Nombre pila	nombre	X	X			Nombre de pila	Texto			varchar(40)
Alumno	Número de Cuenta	numCuenta	X	X	X		Identificador único del alumno	Identificador	999999999-9		varchar(10)
Horario	Clave de Grupo	claveGrup	X	X			Identificador del Grupo al que corresponde	Identificador	9999		varchar(10)
Horario	Clave de Horario	claveHorario	X	X	X	X	Identificador único de este horario	Secuencia		>=0	integer
Horario	Clave de Materia	claveMater	X	X			Identificador de la Materia a la que corresponde	Identificador	A9999		integer
Horario	Cupo	cupo	X				Cupo del grupo para este horario	Cantidad positiva		0 60	integer
Materia	Clave de Materia	claveMater	X	X	X		Clave de Materia	Identificador	A9999		varchar(10)
Materia	Nombre Materia	nombreMat	X	X	X		Nombre de la Materia	Texto			varchar(40)
Prioridad	Número de Cuenta	numCuenta	X	X	X		Identificador único del alumno	Identificador	999999999-9		varchar(10)
Prioridad	Posición	posicion	X	X	X		Posición de preferencia de inscripción	Cantidad positiva		>=0	integer
Sesion	Clave de Horario	claveHorario	X	X			Horario al que pertenece esta sesión	Identificador		>=0	integer
Sesion	Día	dia	X	X			Día de la semana en el que ocurre la inscripción	Día de la semana		Lunes Viernes	varchar(10)
Sesion	Duración	duracion	X	X			Duración de esta sesión	Minutos		0 120	integer
Sesion	Inicio	inicio	X	X			Hora de inicio de la sesión	Hora			time
Solicitud	Clave de Horario	claveHorario	X	X			Identificador único de el horario solicitado	Identificador		>=0	integer
Solicitud	Número de Cuenta	numCuenta	X	X			Identificador único del alumno	Identificador	999999999-9		varchar(10)
Solicitud	Número de Solicitud	numSolic	X	X	X	X	Folio de esta solicitud	Secuencia		>=U	serial
Solicitud	Posición	posicion	X	X	X		Posición de preferencia de inscripción	Cantidad positiva		>=0	integer

Figura 44 Ejemplo de tabla de atributos.

En esta tabla se establecen desde un inicio los nombres físicos de los atributos, esto es importante ya que hacerlo durante la etapa de diseño cuando la lista está completa se vuelve una tarea engorrosa. A menos que en la etapa de análisis no se cuente con estándares de nomenclatura.

Las columnas abreviadas significan lo siguiente:

NN – Not Null, significa que forzosamente se debe dar valor a este atributo.

NB – Not Blank, significa que no puede ser "blanco", es decir, para datos de tipo texto no puede ser una cadena vacía o con espacios y para datos numéricos que no puede ser cero.

ND – Not duplicated, no acepta duplicados en la misma tabla, misma columna.

Cuando un campo es NN, NB y ND es un buen candidato para ser llave primaria. Sin embargo, no siempre tiene que serlo, por ejemplo, en la entidad Prioridad, el atributo de Posición es NN, NB y ND, pero no es llave primaria. Sin embargo al revés se cumple por definición que toda llave primaria es NN, NB y ND.

SG – System Generated, es una llave primaria que generará el sistema automáticamente, lo más simple es utilizar un campo de SECUENCE el cual es soportado por casi todas las bases de datos y se trata simplemente de una secuencia numérica de enteros (0, 1, 2, 3, ...), pero en ocasiones por razones de seguridad no es posible generarlo de esta manera, ya que alguien podría ingresar datos simplemente "inventando" una clave viendo cual es la última clave utilizada.

Esta lista de atributos es importante construirla porque permite detectar inconsistencias en el modelo todavía antes de diseñar. Por ejemplo, la tabla de horarios debe almacenar los días, inicio y duración en que se imparte la materia a un grupo, pero las materias se pueden impartir en 1, 2 ó 3 sesiones por semana. Y para cada sesión es necesario manejar el día, la hora y la duración. Por lo que se descubre que se necesita una nueva entidad: Sesión. El diagrama queda:

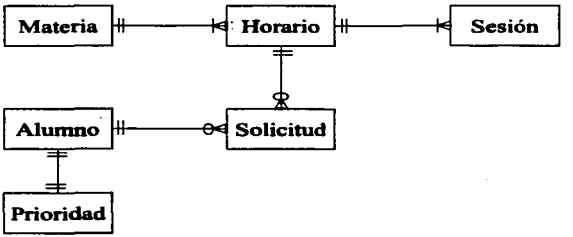


Figura 45 Diagrama Entidad Relación.

Normalización

Al final del proceso de Análisis (y de preferencia solo hasta el final), se Normaliza el modelo de datos. Esto consiste en verificar que nuestro modelo cumpla con las 3 reglas básicas de normalización de bases de datos. Se recomienda hacerlo al final, porque sólo hasta el final se tendrá mas o menos completo el modelo. Hacerlo antes implicará repetir el proceso al final. El no hacerlo antes no tiene efectos contraproducentes. El normalizar el modelo nos garantiza hasta cierto punto que nuestras inserciones y actualizaciones a la base de datos serán lo más eficientes posibles. No necesariamente así para las lecturas. También nos garantizará que no habrá redundancia de datos, y por lo tanto que no habrá código superfluo provocado por datos innecesarios.

El Modelo de Procesos

Respuestas

Para cada evento se genera un diagrama en el que se muestra cuales son las respuestas que generará el sistema y a quien va dirigida. También se muestra en este esquema cuales son las

entidades de datos que intervienen y los datos principales que se intercambian. Además, estas relaciones se muestran con una flecha que indique el flujo del dato. Este tipo de diagramas se conocen como DFD, Diagrama de Flujo de Datos. El siguiente diagrama ejemplifica este proceso para el Evento Solicitud de Inscripción.

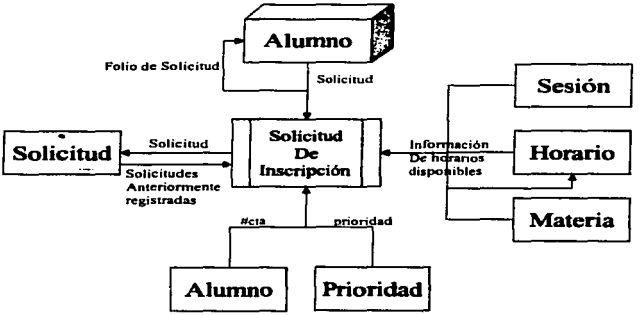


Figura 46 Diagrama de respuestas a un evento.

Descomposición de procesos

El diagrama anterior generalmente es suficiente para entender el proceso necesario que se llevará a cabo. Pero en ocasiones el proceso es muy complejo. Y es en tales casos en los que es necesario "abrir" el proceso, mostrando los subprocesos de los que consta. Tomando como ejemplo el mismo Evento, el diagrama de descomposición de procesos quedaría de la siguiente manera:

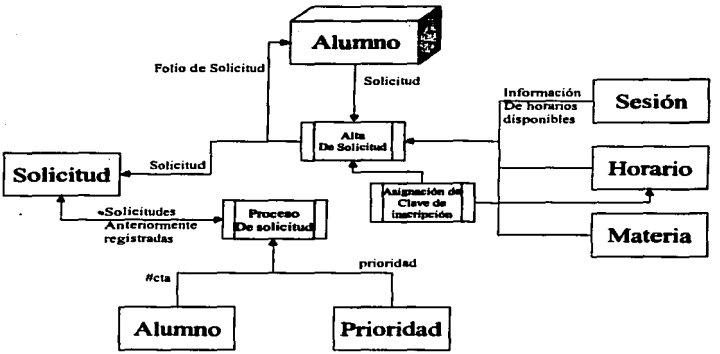


Figura 47 Descomposición de procesos.

El proceso compuesto de esta manera debe cumplir con los siguientes requisitos:

- Las líneas de entrada de datos deben coincidir con las del diagrama de más alto nivel. Es decir, el número de líneas de las entidades de datos y de actores deben ser la misma cantidad que en el diagrama sin descomponer.
- Internamente los subprocesos se pueden transferir datos.
- Los subprocesos no están en un orden particular, es decir, no reflejan el orden o la secuencia de cada subproceso, sino solamente las dependencias de información que existen, aunque implícitamente exista una secuencia.

Por ejemplo, el subproceso de "Alta de Solicitud" depende del subproceso de "Asignación de Clave de Inscripción" y a su vez el Subproceso de "Proceso de Solicitud" depende de la Solicitud previamente registrada por el subproceso de "Alta de Solicitud". De esto se puede apreciar que la secuencia de pasos se va dando naturalmente de acuerdo a las dependencias de información. Pero en las fases de análisis no es conveniente establecer la secuencia, ya que conforme cambien los datos, pueden cambiar la secuencia de pasos.

Especificación Evento-Respuestas

Con la información hasta aquí obtenida es posible generar una mini-especificación. Este mini-especificación debe contener cual es la información esperada y las respuestas a producir dependiendo de los datos de entrada.

En términos prácticos la mini-especificación contiene:

- Datos proporcionados por el usuario
- Notas acerca de las opciones que se le darán al usuario para hacer sus elecciones
- Datos obtenidos de las entidades del sistema, indicando las llaves de acceso para cada entidad.

- Validaciones requeridas para esta información.
- Reglas del negocio a aplicar sobre esta información o información generada en el proceso.
- Respuesta esperada: en términos de datos generados y como es que se generarán y hacia donde se dirigirán (almacenamiento interno, al usuario, generar archivo externo, etc.).
- Estas especificaciones se pueden acompañar de pantallas, bocetos de reportes, fórmulas matemáticas, etc.

Lo que NO debe ir en una mini-especificación es:

- . Pseudo código: Aunque las descripciones deben ser muy precisas y sobrias, es mejor no establecer desde un inicio una secuencia de proceso. La secuencia se definirá al momento del diseño.
- Incluir información que corresponda a otros diagramas o partes del modelo, por ejemplo, el diagrama Entidad-Relación, la lista de Atributos, etc. Es mejor incluir las referencias apropiadas.

El siguiente es un ejemplo de mini-especificación para el Evento de Alta de Solicitud:

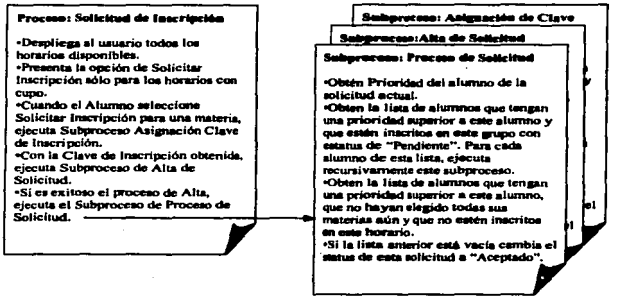


Figura 48 Ejemplo de mini-especificaciones.

El Modelo de Interacciones (datos-procesos)

Hasta este punto se puede decir que se tiene el Análisis completo, ya que están definidos los Eventos que soportará el sistema, las respuestas que generará el sistema, las transformaciones necesarias para generar dichas respuestas (los procesos) e incluso los datos que se necesitan para lograr tales objetivos.

Lo que falta es proceder con los pasos de verificación de nuestro modelo. Estos pasos nos permiten ligar los datos y los procesos de tal manera que existan correspondencias lógicas entre uno y otro. Es decir, que los procesos tengan los datos suficientes para lograr las transformaciones que se desean y que los datos tengan los "suficientes" procesos que los operen.

CRUD

El primer paso es generar una tabla de CRUD. CRUD son las siglas de CREATE-READ-UPDATE-DELETE, que corresponden a los postulados de SQL que componen el ciclo de vida de un renglón en la base de datos. Toda Entidad debe tener estos 4 postulados en algún punto del proceso. Supónganse que falte alguno de ellos y entonces estaremos hablando de un sistema incompleto. ¿Si falta el CREATE cómo se alimentarán los datos para una entidad en particular? ¿Si faltan los READ, entonces para que sirve el dato si nadie lo accederá? Estas son el tipo de preguntas que debemos hacernos en esta fase.

La tabla es una matriz, en la que se distribuyen todas las entidades del sistema y por otro lado todos los eventos. Y para cada intersección Entidad-Evento se indican las operaciones programadas para dicha combinación de Entidad-Evento. La siguiente tabla muestra esto:

	Entidad					
Evento	Alumno	Horario	Materia	Prioridad	Sesion	Solicitud
Solicitud de Inscripción	R	RU	R	R	R	CRU
Alta de Materia		C	C		C	
Consulta de Status Materias		R	R		R	RU
Consulta x Alumno	R	U	R	R	R	RU
Funciones faltantes	CUD	UD	UD	CUD	UD	D

Figura 49 Ejemplo de CRUD

En la tabla anterior se pueden apreciar las operaciones faltantes de la aplicación. Dependiendo del contexto de nuestro sistema es posible que sea correcto que se omitan las funciones. Por ejemplo:

Para el caso de Alumno y Prioridad, esta información podría ser obtenida de otro sistema. Aún en casos cómo estos se deben incluir las interfases de datos necesarias.

Lo mismo ocurre con Materia-Horario-Sesión.

Sin embargo, para Solicitud esto no puede ser así, dado que el objetivo central de nuestro sistema es controlar precisamente Solicitudes. Por lo que faltan 2 funciones: Cancelación de Solicitudes, la cual sería disparada por el alumno para cancelar una solicitud aún pendiente en la que ya no esté interesado; y la función de Depuración de Solicitudes, que se podría disparar cuando un Horario alcance su cupo máximo.

De acuerdo al análisis anterior surge la idea de mantener el subproceso de Proceso de Solicitud como un subproceso totalmente independiente del evento Solicitud de Inscripción, ya que será requerido en diferentes momentos. También es posible que se requieran 2 procesos diferentes: uno para el procesamiento de las dependencias de solicitudes para la combinación de un Alumno con un grupo en especial y otro que procese todas las solicitudes pendientes independientes del Alumno y del grupo que se trate. Ambos procesos por cuestiones de optimización serían diferentes. Pero este tipo de detalles, son problemas propios del Diseño, no del Análisis. Recuérdese que el Análisis se ocupa del "Qué" y el Diseño del "Cómo".

Pero este tipo de consideraciones es importante anotarlas, para que durante la etapa de Diseño no se pierdan de vista.

En este punto, se completaría el Análisis integrando los eventos faltantes para completar nuestra tabla de CRUD, o en su defecto se justificaría por escrito las omisiones de estas funciones. Cabe mencionar que este proceso de CRUD es muy útil para Renovación de Software, que es un proceso que se aplica cuando se tiene un sistema muy antiguo y con múltiples modificaciones, y esta técnica permite depurar un sistema comparando los procesos

contra los datos, pero en lugar de a nivel Entidad, a nivel Atributo, de tal manera, que para cada atributo existan sus respectivos procesos de CRUD.

Vista Entidad-Proceso

El siguiente punto de control de nuestro modelo es comparar que cada Entidad requerida para un Proceso en Particular esté conectada. Esto se realiza concatenando el diagrama de flujo de datos para un Evento en particular (o incluso para un subproceso) a un diagrama Entidad-Relación que muestre solamente las entidades involucradas. Ejemplo:

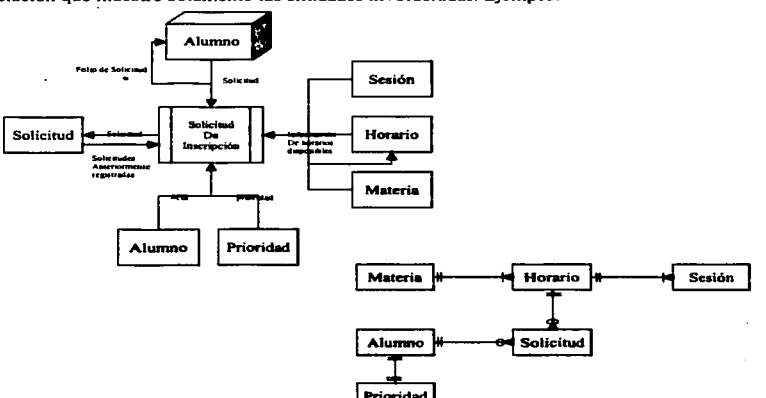


Figura 50 Vista Entidad-Proceso.

Para nuestro ejemplo, se debe notar que todas las entidades que se necesitan para la Solicitud de Inscripción, están conectadas de alguna manera en el diagrama Entidad-Relación. Podría darse el caso, de que un Evento para su proceso requiriera de Materia y Sesión, pero aparentemente no de Horario. En ese caso nuestro diagrama Entidad-Relación luciría de esta manera:

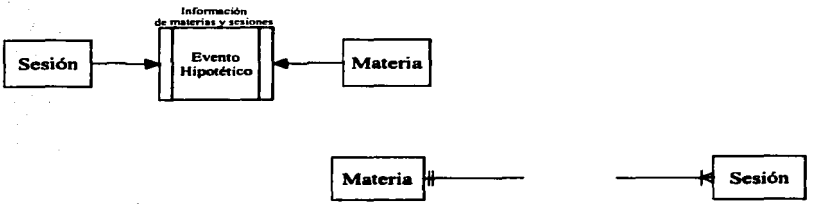


Figura 51 Vista Entidad Proceso con una Entidad faltante.

En este ejemplo hipotético, se puede apreciar que Materia y Sesión son las únicas Entidades utilizadas para el proceso del Evento "Hipotético", pero en el Diagrama de Entidad-Relación se observa que no están "conectadas" estas Entidades. Esto significa que nuestro Análisis no está completo para este Evento, ya que nos falta incluir la Entidad Horario. De tal manera que las Entidades requeridas queden "conectadas".

Para nuestro sistema es relativamente simple examinar esto. Pero en sistemas complejos con 40 ó más Entidades se vuelve una tarea difícil. Las herramientas de análisis para este paso sí son de mucha utilidad ya que permiten visualizar los Diagramas Entidad – Relación de una manera parcial, mostrando solo las Entidades en las que está uno interesado. Pero salvo esta excepción, en el resto de los diagramas y modelos no son de gran ayuda, y en otros no pueden generarlos (como es el caso del CRUD).

Ciclo de Vida de las Entidades

El último paso de validación de nuestro Análisis, tiene que ver con el entendimiento de las reglas del Negocio y de su correcta aplicación a nuestro Modelo, específicamente a las entidades.

En toda aplicación siempre hay una Entidad central, la cual es la razón de ser del sistema. En nuestro caso el sistema recibe Solicitudes de Inscripción y las organiza, valida, procesa y en su caso Acepta o Rechaza. Por lo que la Entidad principal es la Solicitud. Otros ejemplos son:

Sistema	Entidad Principal
Base de Datos de Clientes	Persona Física / Persona Moral
Sistema de Cheques	Cuentas
Sistema de Crédito	Cortes y Pagos
Control de Llamadas Telefónicas	Registro de Uso

Es de notarse que en algunos casos la Entidad principal es el "sujeto" de la "acción" como en una Base de Datos de Clientes, pero en otros sistemas es más importante la "acción" que el "sujeto", como en el caso del sistema de Control de Llamadas Telefónicas.

Para nuestro ejemplo, la Entidad más importante es la Solicitud, o sea la "acción" y no el "sujeto", que podrían ser el Alumno o la Materia. Esto depende del enfoque que se le dé al sistema. Y Depende también de la arquitectura global de los sistemas de la organización, lo ideal es mantener sistemas independientes para "Sujetos" y "Acciones":

- Sistema de Control Escolar (Alumnos: Sujetos)
- Sistema de Control Académico (Materias, Profesores: Sujetos)
- Sistema de Control de Inscripciones (Solicitudes de Inscripción: Acciones)
- Sistema de Calificaciones (Resultados académicos, ordinarios y extraordinarios: Acciones).

Otro ejemplo, en las empresas Financieras:

- Base de datos de Clientes (Nombres, Domicilios, Familias: Sujetos)
- Sistema de Cheques (Cuentas, Depósitos, Retiros: Acciones)
- Sistema de Créditos Personales (Solicitudes, Calificación, Riesgo, Disposiciones, Pagos: Acciones)
- Sistema de Créditos Hipotecarios (Solicitudes, Calificación, Riesgo, Adjudicación, Pagos: Acciones)
- Sistema de Servicios (Autorizaciones, Activaciones, Cancelaciones: Sujetos)
- Sistema de Contabilidad (Cargos, Abonos, Cuentas, Subcuentas, Códigos de Operación: Acciones)
- Sistema de Gestión (Gastos, Operaciones, Ingresos: Acciones)

En general, lo que aplica para desarrollar un programa aplica para los sistemas. Se debe evitar el desarrollar un Sistema "Maravilla" (que lo hace todo), así como tampoco se codifica un Programa "Maravilla".

Regresando al asunto de este inciso, una vez detectada la Entidad principal de nuestro sistema, se esquematizan en un diagrama los estados que puede adoptar dicha entidad durante su ciclo de vida (desde que se crea hasta que se elimina). En cada transición se indica cual es el evento que provoca el cambio de estado.

De esta manera se garantizan 2 cosas:

La primera es que nuestro Entidad principal cumple con los diferentes estados posibles de acuerdo a las reglas de negocio.

En segundo lugar, que nuestro modelo está completo ya que existe un Evento que se atiende para cada estado posible de nuestra entidad.

El diagrama de ejemplo para nuestra aplicación es el siguiente:

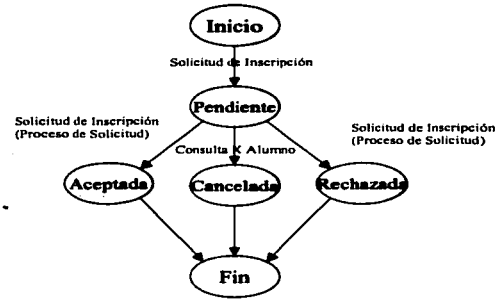


Figura 52 Ciclo de vida de una entidad.

Nótese que en el Evento Consulta X Alumno, será necesario incluir la opción de "Cancelar Solicitud" para tener el ciclo de vida completo de la entidad.

Transición al Diseño Físico

Versiones

Actualmente el software no se construye al 100% en un solo ciclo de desarrollo. Sino que se va desarrollando progresivamente en fases que se les llama iteraciones. En cada iteración se seleccionan los eventos que se desarrollarán y para cada uno de ellos se ejecuta un ciclo completo de desarrollo, desde Análisis hasta Instalación en Producción.

De esta manera se controla el riesgo en los proyectos, ya que al completar Eventos, se pueden calcular mejor los tiempos de desarrollo de los futuros segmentos del sistema. Normalmente también se seleccionan para iniciar los Eventos de mayor riesgo. De esta manera, el riesgo se reduce hacia el final del proyecto y no a la inversa como ocurre en proyectos mal administrados, es decir, que el riesgo se incrementa hacia el final del proyecto.

Dado que con cada iteración se libera una nueva funcionalidad, es típico llamar a cada una de ellas una Versión. Y es por eso que los eventos se agrupan en versiones, de tal manera que se establece un calendario de desarrollo con fechas de liberación para cada liberación.

Para nuestro sistema las versiones pueden quedar de la siguiente manera:

#	Evento	Versión
1	Solicitud de Inscripción	1
2	Alta de Materia	3
3	Consulta de Status Materias	4
4	Consulta x Alumno	2

Figura 53 Liberación de versiones.

Distribución del proceso

En ocasiones los módulos o segmentos de un sistema se separan en varios "Contenedores" con la intención de soportar una mayor carga o debido a cuestiones de seguridad, desempeño y/o

disponibilidad de la información. En tales casos se debe especificar en dónde residirá tanto la información, como los segmentos de código. Y en tales casos se construye una tabla donde se especifica como se distribuirán las entidades de datos en los diferentes sitios de la organización.

Para nuestro ejemplo el proceso y la información se ubicarán en un solo sitio, he incluso en una sola máquina que actuará como servidora de base de datos y servidor aplicativo.

Diseño básico orientado a objetos

Una vez que los requerimientos están completos para un evento, se puede pasar a la etapa de diseño de este evento. Aquí es importante recalcar la independencia de un evento con otro. Normalmente se piensa que no es posible diseñar una aplicación completa hasta no conocer todo el sistema. Ya que se corre el riesgo de duplicar código o no separar adecuadamente el código común y reutilizable del código específico para una función.

Sin embargo, el particionamiento por eventos tiene por objetivo descomponer el sistema en partes independientes integradas únicamente por los datos. Y por el contrario, se desea que el código compartido entre eventos sea mínimo o nulo.

Esto es un concepto interesante, porque normalmente se intenta reutilizar el código lo más posible. Sin embargo, esto también conlleva riesgos, y el principal es que un cambio a código compartido tiene un mayor impacto en el sistema en general. De tal suerte, que al modificar un segmento de código que se utiliza en el 40% del sistema, implicará tener que probar de nuevo al menos el 40% del sistema.

En la práctica la fase más dolorosa del desarrollo de un sistema, son las pruebas. Mucho más que la codificación. Así que manteniendo el código lo más independiente posible de otros segmentos, permite un proceso de pruebas más sencillo.

Pero entonces, ¿no debo compartir código y evitar la reutilización? Tampoco. Ambos son los extremos del mismo problema. La mejor práctica es compartir los diseños, es decir, construir una arquitectura aplicativa y reutilizar esta una y otra vez. Por ejemplo, creando programas modelo los cuales se copian una y otra vez modificando aquellas partes específicas según se requiera.

En este punto es donde las herramientas juegan un papel determinante, ya que las herramientas nos permiten generar el 80% del código y modificar sólo el 20%. Sin embargo, estas herramientas son comerciales en su gran mayoría y costosas. Pero es en este rubro donde hay un amplio margen en el cual las universidades pueden inducir a sus alumnos a participar, primero desarrollando modelos y luego desarrollando generadores de código.

Ahora, para nuestra aplicación utilizaremos los modelos descritos en los capítulos anteriores:

La estructura general será Model-View-Controller.

Para facilitar el desarrollo utilizaremos Struts como Controller.

La capa de Vista se desarrollará con jsps.

Y el modelo de negocio, se implementará con Enterprise Javabeans, específicamente con Session Beans. La capa de datos se implementará con Container Managed Persistence Beans.

Finalmente, se seguirán las estructuras de flujo descritas en el capítulo 4. Estructura Básica de una Aplicación y el capítulo 5. Modelos Básicos de Programación.

Y por último, ¿Entonces donde está el diseño?

Ya lo hicimos.

¿En dónde?

Una de las terribles confusiones en sistemas es pensar que debo aplicar una y otra vez la teoría de Diseño de Sistemas (sea la metodología que sea) cada que desarrollo una aplicación. Para hacer una analogía, piénsese en los Métodos de Ingeniería para resolver un problema X. El primer paso es generalmente investigar si alguien ha resuelto antes el mismo problema, después buscar un problema similar, finalmente si ninguno de las dos opciones anteriores nos da una solución se inicia un proceso de análisis del problema.

En nuestro caso, iniciamos este tema diciendo que las aplicaciones transaccionales no son nuevas, pero que las aplicaciones de Internet si lo son. Pero ambos problemas están resueltos, ya que la transaccionalidad se resuelve utilizando Enterprise Javabeans. Las personas que diseñaron estos componentes lo hicieron pensando de tal forma que el desarrollador sepa simplemente elegir el tipo de componente a utilizar. Y el tipo de componente a utilizar serán los Session Beans que sirven de Interfaz para acceder los Entity beans , o sea los datos. Este patrón de diseño se conoce como Session Facade. El segundo problema es el desarrollo de aplicaciones para Internet. En este caso el patrón de diseño es el Model View Control. El cual ya describimos en los capítulos anteriores.

Por lo que podemos decir que nuestro diseño está completo.

Entonces, ¿Para qué sirve estudiar diseño de sistemas?

Sirve para:

- Entender porqué construimos los sistemas de una manera y no de otra.
- Poder diseñar aplicaciones que se alejan de los patrones comunes.
- Mejorar los diseños, y por ende, mejorar los sistemas.

Estos temas no podemos decir que están agotados. Constantemente se está evolucionando también en el diseño de aplicaciones. Pero por el momento, hay mucha gente que ya hizo la tarea por nosotros y llegó a la conclusión que el desarrollo de sistemas se debe basar en el uso de patrones de diseño. Por lo que el profesional de desarrollo se debe capacitar primero en entender los patrones de uso común, como el Session Facade, y después profundizar en patrones especializados para diferentes tipos de aplicaciones. El siguiente paso en el desarrollo de un profesional es por supuesto mejorar o generar nuevos patrones de diseño.

El Diseño moderno se limita entonces a especificar que patrones son los que se emplearán y especificar la lista de componentes a construir y la funcionalidad que deberán soportar. Los patrones de diseño nos dicen como se integran estos componentes. Los desarrolladores pasan a la etapa de construcción de estos componentes y en etapas posteriores del desarrollo, se ensamblan, se empaquetan y se instalan.

Así nuestra aplicación quedará de la siguiente manera:

El patrón básico es el Model-View-Controller, el cual lo vamos a implementar utilizando el framework de desarrollo Struts.

Struts establece que una aplicación web se descompone en:

- Páginas JSP y
- Action Form

Estos componentes nos resuelven la Vista y el Control de la aplicación. Pero Struts no establece nada acerca del Modelo. En nuestro caso el modelo está compuesto de Datos y de los flujos de control de esos datos. Pero además J2EE establece que los datos se implementen con EJB's de tipo Entity. Entonces nuestro modelo quedará:

Acceso a la base de datos: mediante Entity Beans de tipo Container Managed Persistence.

Para el acceso a los Entity beans, utilizaremos el patrón de diseño Session Facade. Este patrón se implementa con un EJB de tipo Session Bean. La estructura de nuestra aplicación será la siguiente:

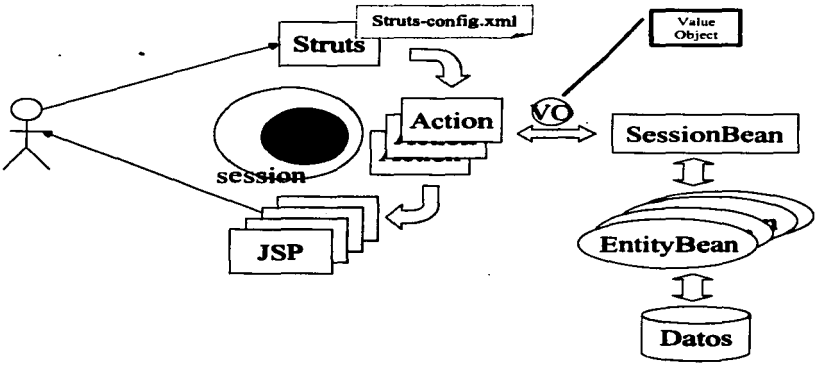


Figura 54 Model View Controller y el acceso a Entity Beans.

Esta estructura es la descrita en el capítulo 4. Solo nos falta mencionar que el paso de parámetros entre los Action Form y el Session Bean será mediante Value Objects.

Una vez establecida la estructura de nuestra aplicación, lo siguiente es detallar la funcionalidad y responsabilidades que recaerán en cada Clase y esto es lo que se conoce como Diseño Detallado. En este momento tenemos un grado de descomposición del sistema donde es fácil establecer las fronteras entre cada componente. Cuando el sistema se desarrolla entre 1 ó 2 personas quizás ni siquiera vale la pena hacer una especificación detallada de cada componente. Veamos porque:

JSP's

Estos son componentes visuales, es decir, contienen HTML que especifica tal cual cómo verá nuestra aplicación el usuario final. Normalmente es más fácil definir la pantalla en HTML que será la base posterior para insertar el código Java y convertirla en una JSP, que hacer un dibujo o diagrama para especificar como debe lucir la JSP.

Action Form

Estos no son objetos visuales. Pero su función se reduce principalmente a validar los datos ingresados en una forma de html por el usuario y segundo, establecer el flujo de la aplicación el cual depende del resultado de la validación. En todo caso aquí solo se especifica cuales son tales validaciones.

Entity Beans

Aquí no hay nada que especificar. Sólo se definen los campos y el mapeo de estos contra la tabla en la base de datos. Se incluyen también los métodos básicos de búsqueda y creación de los Entity y los métodos que establecen la relación de cada Entity con los otros Entities de la aplicación. Pero toda esta información es la definida en el Modelo de Datos de la fase de Análisis.

Session Bean

Aquí es donde residen las reglas del Negocio. Estos son los componentes que son el corazón de nuestra aplicación. Estos requieren una especificación clara de cuales son las validaciones y el flujo de las operaciones necesarios para generar el resultado esperado. Se apoya en los Entity Beans para generar su resultado. Sin embargo, si el proceso es simple es suficiente con la descripción de lo que se espera de cada método público de este componente.

Flujo Aplicativo

Este es el componente más importante de la aplicación. Físicamente no se representa en código, pero establece el mapa entre los componentes. Se parte de las pantallas que verá el usuario para establecer el flujo interno de invocaciones. En términos prácticos es el único que requiere ser especificado.

Esto en la práctica se describirá cuando se muestren las etapas de la construcción en el capítulo 8 de esta Tesis. Partiremos de un Flujo Aplicativo, se diseñarán las JSP's, se implementarán los Entity Beans y se irán uniendo las piezas mediante el Session Bean , los Action Forms y los Value Objects.

Antes en el capítulo 7, nos detendremos para explorar los diferentes tipos de aplicaciones que es posible construir y un breve acercamiento a los productos disponibles, tanto libres como comerciales, para llevar a la práctica estos conceptos.

Capítulo 7. Escenarios aplicativos y Productos comerciales disponibles

Escenarios Aplicativos

Aquí se trata de definir cuáles son los tipos de aplicaciones que se pueden implementar con J2EE. Cabe mencionar que la Plataforma Java en general soporta prácticamente cualquier tipo de aplicación, con excepción de aplicaciones que requieran proceso en tiempo real ya que este tipo de aplicaciones requieren procesamiento de alta velocidad el cual Java no soporta debido a que se sacrifica desempeño por portabilidad. Y las aplicaciones de alta velocidad son totalmente dependientes del hardware en que ejecutan.

Hecha esta excepción, las aplicaciones se pueden clasificar por el tipo de arquitectura en el que se implementan o dicho de otra manera, por el tipo de acceso que soportan a su funcionalidad. La otra clasificación es por el tipo de operaciones que realizan, que son más bien estereotipos conocidos de las aplicaciones y que se caracterizan por cierta funcionalidad principal.

Tipos de aplicación de acuerdo a su Arquitectura

Existen 5 tipos de aplicaciones soportadas por J2EE:

Aplicación Multicapas

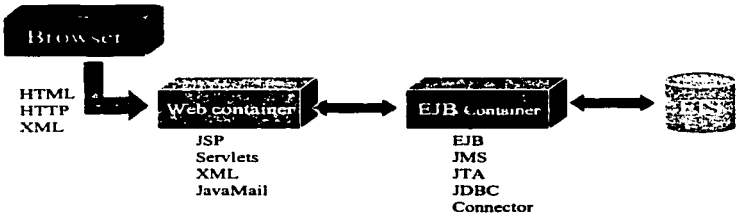


Figura 55 Aplicación Multicapas.

Este es el tipo de aplicación más flexible en cuanto a escalabilidad y mantenimiento. Se utiliza para aplicaciones Web principalmente. Separar la funcionalidad web del modelo del negocio permite "crecer" este tipo de aplicaciones para soportar otro tipo de clientes diferentes al cliente web. EIS, significa Enterprise Information System, y comúnmente es una base de datos

pero puede ser cualquier otra fuente de información o incluso otro Contenedor EJB o Web o Servidor Aplicativo.

Cliente Stand-Alone

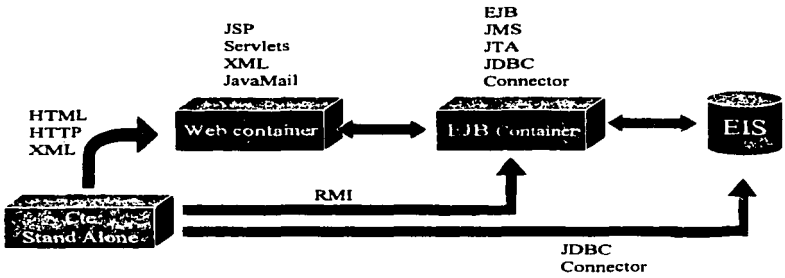


Figura 56 Aplicación Cliente Stand-Alone

Esta aplicación es diferente simplemente en el tipo de cliente. En este caso es un cliente que no requiere un Web Browser, y que se construye en modo gráfico con las librerías AWT o Swing de Java. Este cliente puede acceder un Web Container, un EJB Container o directamente al EIS. Es el cliente más poderoso que hay en la arquitectura Java y puede ser tan simple o tan complejo como se desee. Las aplicaciones típicas de este estilo son por ejemplo las aplicaciones gráficas para ejecutar en entornos de ventanas como Windows, Gnome o KDE. Este tipo de aplicación se elige cuando se desea una Interfaz con el usuario sumamente compleja como puede ser la de un editor de documentos.

Centrada en EJB

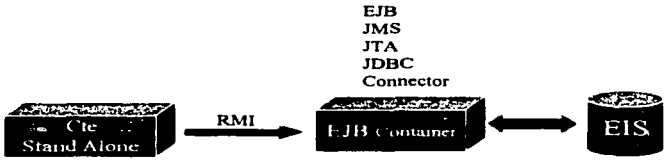


Figura 57 Aplicación Centrada en EJB.

Este tipo de aplicaciones se utiliza muy poco. Su aplicación principal es realizar operaciones Batch que no requieren intervención humana. Por ejemplo, una aplicación de tipo Multicapa cuya función es dar mantenimiento a la Base de datos de Clientes. Normalmente los clientes se registran uno por uno de manera interactiva. Pero puede suceder que sea necesario dar de alta una lista de clientes almacenada en un archivo. Entonces sería muy laborioso y propenso a errores capturar uno por uno de los clientes. En ese caso se construye un programita en Java

que lea el archivo, arme los parámetros requeridos y envíe la petición de alta directamente al EJB responsable de esa función.

Centrada en Web

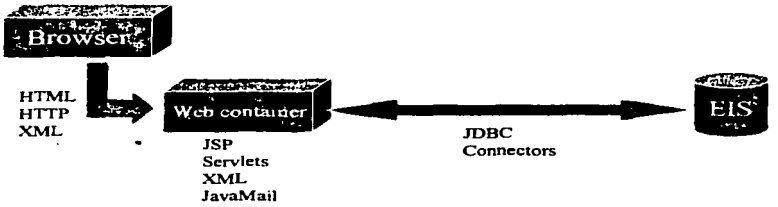


Figura 58 Aplicación Centrada en Web.

Este tipo de aplicaciones sólo soportan clientes Web y no necesitan mucho proceso. Generalmente solo son aplicaciones de consulta de información, de ahí que no se utilice una capa de EJB. En otras palabras, no operan datos y por lo tanto no requieren control transaccional. Y dado que sólo recibirán clientes web, todo se implementa en la capa del Web Container. Son las aplicaciones más simples de construir y se recomiendan para iniciar en el entendimiento de la tecnología J2EE.

Business-to-Business (B2B)

Este modelo es el empleado cuando se requiere interconectar las aplicaciones de 2 ó más empresas para intercambio de información. La estructura general es la siguiente:

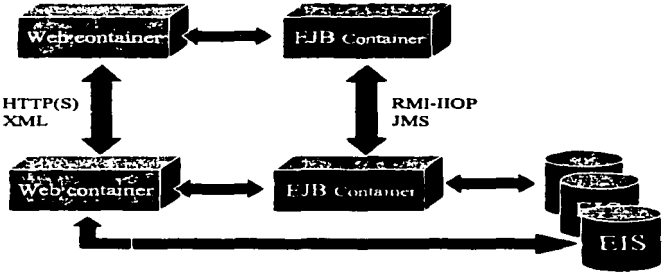


Figura 59 Aplicación Business-to-Business (B2B).

En este diagrama se muestran las 2 posibilidades, una interconexión a la capa Web utilizando XML como formato de intercambio de datos, o una interconexión a al contenedor de EJBs, la cual se realiza mediante protocolos Java (RMI-IIOP) o mediante componentes de envío de mensajes (JMS). Este tipo de aplicaciones son incipientes todavía, pero en un futuro serán

muy comunes. La industria está poniendo mucho esfuerzo en este tipo de aplicaciones y están naciendo estándares y librerías especiales para soportar este tipo de aplicaciones. A estos estándares se les conoce como WebServices y sus objetivos principales son: Establecer un estándar de intercambio de información y de directorio de aplicaciones y por el otro lado mecanismos estándar de interconexión de aplicaciones para interconectar aplicaciones que originalmente no estaban diseñadas para B2B.

Tipos de aplicación de acuerdo a su funcionalidad

Por el tipo de funcionalidad hay 2 tipos de aplicaciones:

Aplicaciones Transaccionales

Estas aplicaciones se caracterizan por el uso intensivo de datos pero a nivel Entidad. Son ejemplos de este tipo de aplicaciones los Sistemas Bancarios, de Telefonía, de Puntos de Venta, Inventarios, Manejo de órdenes de compra, etc.

En este tipo de aplicaciones toma mucha relevancia el Modelo de Datos, ya que del modelo de datos dependerá el desempeño de la aplicación. Además este tipo de aplicaciones se caracterizan por múltiples usuarios concurrentes los cuales normalmente acceden a información que es ajena totalmente a otros usuarios y por lo tanto no puede compartirse. Un ejemplo de esto es la función de Consulta de Saldo en los bancos. Esta función puede estar siendo utilizada simultáneamente por miles de usuarios pero cada uno accediendo a un dato específico diferente al de los otros. Todo el peso del proceso de esta operación recae en el manejador de Base de Datos, ya que simultáneamente debe resolver estos miles de accesos al campo de Saldo.

Dado que se requiere un tiempo de respuesta de milisegundos para este tipo de aplicaciones, la Interfaz es sumamente sencilla, con el fin de optimizar el desempeño global. A medida que el volumen de usuarios concurrentes crece se vuelve también muy importante el acceso al disco duro, ya que será necesario acceder a puntos del disco muy distantes uno de otro. Normalmente se utilizan arreglos de discos que tienen distribuida la información en un número muy grande de discos para permitir la lectura simultánea de varios bloques de información. Si la información estuviese en un solo disco, las peticiones tendrían que encolarse para ser atendidas.

En estos casos, sin embargo, Java ayuda de diferentes maneras para estas aplicaciones, por un lado el manejo de la transaccionalidad con los Enterprise JavaBeans, por el otro lado compartiendo un solo proceso entre múltiples usuarios concurrentes (esto también mediante los EJB's y los Servlets). Todo esto permite aprovechar de la mejor manera los recursos de cómputo. Sin embargo, el alcance de Java es hasta ahí (y de cualquier lenguaje), ya que hay un punto en el que todo depende de la base de datos y poco o nada puede hacer el desarrollador aplicativo.

Se vuelve muy relevante el papel del Diseñador del Modelo de Datos y del Administrador de la Base de Datos.

Aplicaciones de Información

Estas son aplicaciones que se caracterizan por no efectuar operaciones sobre la información y se limitan a realizar procesos de lectura de la información. Sin embargo, esta consulta de información se realiza masivamente. Es decir, se consulta información detallada para generar información estadística. Aquí nuevamente el diseño y desempeño de la base de datos se vuelven críticos, ya que de otra manera el tiempo de respuesta podría no ser el aceptable, pero un acceso de lectura a datos masivos es diferente al acceso masivo de un solo dato.

En este caso Java, ayuda proveyendo de herramientas para acceder a la información desde casi cualquier dispositivo y de muy diversas maneras. Aunque nuevamente el desempeño crítico dependerá del modelo de datos.

En este caso el modelo de datos no solo incluye la información a detalle, sino para minimizar consultas y lecturas, se le integran tablas con datos integrados a diferentes niveles.

Estos 2 tipos de aplicaciones contrastan uno con otro, en cuanto al diseño de las bases de datos. Mientras que en los sistemas transaccionales se prefieren modelos de datos normalizados, en los sistemas de información se prefieren modelos de datos denormalizados. Y como en ambos casos el desempeño depende del modelo de datos, es importante tener desde un inicio muy claro cual es el tipo de aplicación de la que se trata para elegir el modelo de datos adecuado para nuestra aplicación.

Servidores Aplicativos

Aquí no solo hablaremos los servidores aplicativos disponibles, sino también de productos relacionados indispensables para construir las aplicaciones .

Servidor Web

Apache

Apache es el servidor web más utilizado a escala mundial. Está protegido por la licencia de Software Libre, por lo que el código fuente está disponible para todos. No hay más que decir, salvo que es el estándar de facto de la industria. Solo cabe distinguir que un Servidor Web solo sirve para publicar páginas Web, y en cambio los Contenedores Web son componentes Java que permiten el procesar-publicar páginas JSP y Servlets. Esta aclaración es pertinente ya que constantemente se confunde un término y otro. Para contribuir a esta confusión, el proyecto Jakarta (del cual depende Apache), desarrolló un contenedor Web el cual se integraba a Apache, llamado Jserver. Sin embargo, este proyecto fue abandonado en favor de Tomcat.

Contenedores Web

Jetty

Es un contenedor poco conocido de software libre, escrito totalmente en Java. Este contenedor fue el elegido por Jboss (contenedor EJB) como su complemento para tener un Contenedor J2EE estándar. De momento son pocos sus seguidores (y por lo tanto, poco el desarrollo), pero hay que darle seguimiento a su evolución.

Tomcat

Es el estándar de facto. Cuando se desea simplemente construir una aplicación centrada en Web, basta con este contenedor para desarrollar e instalar nuestra aplicación. Tomcat es el contenedor Web de referencia, lo que significa que es el contenedor con la implementación guía del estándar.

Contenedores EJB

Jboss

Es la única implementación de Contenedor EJB de software libre. Está desarrollado por el Jboss Group el cual se sostiene de sus servicios de documentación y consultoría. El software en sí es gratuito y el código es abierto. Pero la documentación tiene un costo de $10.00 dólares mensuales. Aunque en general para el desarrollo de aplicación como las descritas en esta tesis la documentación básica que es gratuita es suficiente. Pero para aplicaciones más complejas es necesario. ¿Cómo qué cosas serían complejas? Pues como el manejo de Clusters (varias máquinas trabajando con contenedores en paralelo para soportar una mayor carga), Balanceo de carga, seguridad avanzada y transacciones distribuidas (transacciones entre diversas bases de datos).

Este producto es muy poderoso y de los mejores productos desarrollados bajo el concepto de Software libre. Compite en desempeño con los contenedores comerciales. Está escrito 100% en Java por lo que ejecutará en cualquier sistema operativo que tenga disponible una máquina virtual de Java.

La versión estándar de este producto viene integrada con Jetty, para de esta manera proveer el soporte completo: Web Server + Web Container + EJB Container, todo contenido en una sola máquina virtual de Java.

Contenedores J2EE (Web + EJB)

Oracle 9i Application Server

El contenedor de este producto es en realidad un contenedor llamado Orion Server. Es un contenedor también escrito 100% en java. Es pequeño, elegante y muy eficiente, aunque

carece de herramientas en modo gráfico para su administración. Aunque la verdad, su uso es tan simple que dichas herramientas son innecesarias. Por estar escrito en Java, puede correr en cualquier equipo con una máquina virtual de Java (que hoy en día es casi todos). Aunque es un producto comercial, se puede descargar del sitio de Oracle para hacer pruebas de desarrollo. Soporte Balanceo de carga y Clustering.

BEA WebLogic Server

Este es el contenedor J2EE comercial por excelencia. Es el competidor a vencer dado el tiempo que tiene en el mercado y el porcentaje que mantiene de este. Es el más robusto en cuanto a herramientas de administración y para desarrollo. Es costoso, pero los que lo utilizan se sienten bastante satisfechos. Es muy estable, de desempeño rápido, aunque no soporta la última versión del estándar J2EE que es la 1.3. Pero esto es común entre los productos Java, generalmente la estabilidad está peleada con el soporte al día del estándar. Por otro lado, este servidor va más allá de Java en cuanto a que soporta una amplia variedad de conexiones a otros productos aplicativos como por ejemplo SAP, Peoplesoft y herramientas de administración de contenidos: Publicación de documentos indexados, manejo de documentos electrónicos y servicios de comunicación de usuarios como foros de noticias, directorios de personas, etc.

IBM WebSphere Application Server

Este es otro de los competidores comerciales más fuertes. Este servidor aplicativo se caracteriza por estar diseñado para aplicaciones de alto volumen transaccional. Es muy estable pero carece de herramientas poderosas de administración como Bea o de la simplicidad del Oracle 9i Application Server. Sin embargo, esa complejidad en cuanto a su uso y su administración están compensadas por su estabilidad que podría decirse es "roca sólida" y su amplia conectividad a Mainframes, tanto IBM como de otros proveedores. Lo cual permite a empresas de alto volumen transaccional a convertir sus aplicaciones de Mainframes a aplicaciones Web de una manera gradual. Ya que primero se le utiliza como frontend simplemente con páginas JSP que acceden servicios en el Mainframe y posteriormente se van migrando los servicios transaccionales al contenedor EJB de una manera transparente para el usuario.

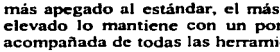
Sun ONE
Open Net Environment

Sun ONE Application Server

Este podría decirse que es el servidor Aplicativo que lo tiene todo, estabilidad, facilidad de uso, herramientas de administración y también un precio muy elevado. Solo para empresas de muy alto poder adquisitivo. Por supuesto, que por ser de la empresa que inventó Java, es el

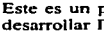
más apegado al estándar, el más puesto al día y no por eso perdiendo estabilidad. Su precio elevado lo mantiene con un porcentaje muy bajo del mercado. Es una solución que viene acompañada de todas las herramientas imaginables, tanto herramientas de desarrollo, así como Wizards para generar código, plugins de conectividad a otras plataformas y otros productos.

Herramientas de Desarrollo



NetBeans

Este es un proyecto de software libre que tiene como propósito generar infraestructura para desarrollar IDE's para diversos lenguajes y plataformas. El producto insignia es Netbeans for Java y es un IDE bastante simple de utilizar y con las herramientas básicas que uno espera encontrar en cualquier IDE. Es muy flexible en cuanto a que se le puede agregar funcionalidad mediante módulos "enchufables" y esto le permite una evolución rápida y en paralelo en diferentes direcciones.



Sun One Studio

Tiene 2 versiones: Community Edition, la cual es gratuita; y Enterprise Edition la cual es de paga. Ambas versiones están basadas en Netbeans, pero al producto original le han agregado módulos que amplían la funcionalidad de Netbeans, como por ejemplo un identador de código y un autodocumentador de código. Definitivamente superior a netbeans por estas razones. Además de que conserva la "pureza" del Netbeans original, de tal manera que los módulos compatibles con Netbeans son totalmente compatibles con Sun One Studio. Esto es similar a la relación que hay entre Linux el kernel y las distribuciones de Linux como Redhat y Mandrake, ambos son Linux pero con su especial manera de disponer las herramientas y las ayudas.

IBM VisualAge for Java

IBM ha desarrollado durante muchos años este IDE. Es como Netbeans un base para desarrollar IDE's para diferentes lenguajes. Así podemos ver que hay VisualAge for COBOL, for Fortran, for CSP, for C++, etc. En todos ellos el factor común es el manejo de un repositorio de código, el cual soporta control de versiones y la Interfaz que permite programación visual cuando el lenguaje soporta componentes visuales como lo es Java. Tiene una Interfaz totalmente orientada a Objetos y es totalmente configurable lo cual confunde un poco al inicio. Sin embargo es una herramienta de desarrollo muy poderosa si es que se tiene la memoria suficiente para correrlo, más de 256MB en RAM. Recientemente como una contribución a la comunidad de software libre, IBM liberó parte del código de VisualAge y

con esto se está desarrollando otro IDE llamado Eclipse, el cual es de distribución gratuita. Ambos productos se caracterizan por tener una Interfaz elegante y estable.

Oracle Jdeveloper

Este IDE es también comercial, es muy poderoso y entre sus factores fuertes están 2 frameworks de desarrollo que ahorran muchas horas de codificación y pruebas, de los cuales se habla más adelante. Se caracteriza por implementar las últimas tendencias del mercado en cuanto a ayudas de desarrollo, como por ejemplo, el soporte de Struts y el empaquetamiento e instalación de aplicaciones directamente al servidor aplicativo. En su contra está que es una herramienta con poco soporte para contenedores y bases de datos que no ostenten la marca Oracle.

JBuilder™

The leading Java™ development solution

Borland Jbuilder

Borland tiene una amplia tradición en herramientas de desarrollo, fue además el pionero de los IDE's con su famosísimo y añorado Turbo Pascal. Pasó una época en la que poco se supo de esta compañía y ahora regresa con herramientas para la plataforma Java. Este producto, continúa la tradición del Turbo Pascal con una Interfaz bien cuidada y con todo tipo de Wizards y herramientas para el desarrollo de aplicaciones. Es multiplataforma y cuenta con buen soporte. Difícil encontrarle un defecto.

Ant

A diferencia de todos los productos mencionados, este no es un IDE. Es una herramienta escrita en Java para automatizar la compilación, empaquetamiento e instalación de las aplicaciones Java. Permite escribir scripts con los cuales se automatizan las tareas antes descritas. Es el complemento ideal para distribuir código que se autocompila y se autoempaqueta. Casi todos los productos mencionados aquí lo soportan, o dicho en otras palabras, pueden generar scripts que ejecutados con Ant, permiten recompilar y empaquetar la

aplicación en cualquier plataforma que soporte Java. Es una herramienta indispensable para cualquier desarrollador de Java.

Frameworks de Desarrollo

Toda la plataforma Java está en un proceso de evolución que debe su éxito a que está pensada en una estrategia de corto, mediano y largo alcance. En el corto plazo están el Lenguaje y las especificaciones de los contenedores. En el mediano plazo están el soporte multiplataforma y la conectividad a todo tipo de recursos informáticos: Bases de datos, servidores aplicativos, servidores de email, de ftp, de seguridad, a otros lenguajes, al sistema operativo local, etc. Y en el largo plazo, el soporte de programación visual y generadores de código.

Actualmente, estamos en la etapa de "adolescencia" de la plataforma, y en esta etapa existen los frameworks de desarrollo que facilitan el desarrollar aplicaciones sin tener que empezar desde cero. La bondad que tienen estos frameworks es que permiten generar código sin perder el control de la aplicación y sin agregar proceso innecesario. En fases futuras, los IDE's antes mencionados se basaran en estos frameworks para producir aplicaciones de manera instantánea, aunque de hecho esto ya está ocurriendo.

Struts

Struts

Este framework es muy popular ya que se utiliza en el desarrollo de las aplicaciones web. Permite controlar el flujo de la aplicación y separar las capas de negocio de las de Interfaz web. Es 100% java puro y fácil de utilizar. Es indispensable su aprendizaje. Y es uno de los subproyectos del Jakarta Group, los mismos que producen Apache y Tomcat.

ORACLE

UIX

De los frameworks comerciales, este es uno de los que merece atención. Es un framework que permite definir la Interfaz con el usuario en XML de tal manera que a partir de una definición se pueden generar pantallas en diferentes formatos: para Web, para teléfonos celulares, para palm devices, etc. Es sencillo de usar, y el único inconveniente es que es un producto comercial.

BC4J

Este es un framework también de Oracle y permite encapsular el acceso a la base de datos mediante vistas, las cuales se procesan mediante métodos simples de java del estilo get y set, para leer y escribir, lo cual lo vuelve muy simple de usar. Todo el acceso a la base de datos se define de manera visual e integra al código el manejo de la paginación y el caché en memoria de la información. Este también es un producto comercial y es un buen ejemplo de lo que debería ser el acceso a una base de datos.

Xdoclet

Este proyecto de software libre, tiene como objetivo el generar tags de xml, los cuales al integrarse en el código de un programa en java, describan como generar código rutinario. Tiene varias aplicaciones, entre las más interesantes están el poder generar de manera automática fragmentos del código de los Enterprise JavaBeans y los Deployment Descriptors,

necesarios para empaquetar e instalar las aplicaciones. De la mano con Ant, sirve para generar también los scripts de empaquetamiento e instalación.

Bases de Datos

Actualmente, podría decirse que no hay base de datos que no soporte Java. El soporte se debe a que existe un estándar de cómo acceder a una base de datos desde Java llamado JDBC. Este estándar por un lado establece cual es la Interfaz esperada por un cliente de base de datos y por el otro lado establece como debe responder una base de datos a las peticiones de un cliente. Después cada proveedor de base de datos implementa este estándar mediante un componente que se conoce como el Driver JDBC. Cada cliente que desee acceder a la base de datos simplemente carga a memoria este Driver y tendrá acceso a la información.

Y dado que todas las bases de datos hoy en día soportan Java es pues muy difícil mencionarlas aquí a todas. Por lo que sólo se incluyen aquí 2 bases de datos que se caracterizan por ser libres: mySQL y PostgreSQL y de las comerciales aquellas que cuentan con el mejor soporte para Java debido su buen desempeño de conectividad.

MySQL

MySQL es quizás la más popular de las bases de datos gratuitas, existe mucho soporte para esta base de datos y diversos tipos de aplicaciones para acceder a los datos o para administrarla. Tiene en su contra que no posee una arquitectura tan refinada como PostgreSQL, pero para las aplicaciones que no requieran un desempeño excelente y de medio volumen es suficiente.

PostgreSQL

Es menos popular que MySQL, pero más poderosa, con una buena configuración puede competir en velocidad con las bases de datos comerciales. También de ambas bases de datos es la que cuenta con mejor soporte para Java. Por otro lado, tiene un mejor soporte del lenguaje SQL92 que mySQL. En su contra está el que por ser menos popular que mySQL, también tiene menos herramientas gráficas para ejecutar queries y para su administración.

DB2

3 son las compañías comerciales que apoyan con todo a la plataforma Java, una es Sun obviamente, la segunda es IBM y la tercera es Oracle. Este soporte se refleja en que todos sus

productos están orientados a soportar Java al 100%. Y es el caso de DB2 que es la base de datos principal de IBM. Posee los drivers más eficientes para el acceso desde Java y tiene la capacidad de escalar desde una PC, hasta un Mainframe.

Oracle9i Database

Oracle Server

Esta es la base más utilizada en las empresas, es también el líder a vencer en cuanto a desempeño. Soporta todas las últimas tecnologías en el mercado en cuanto acceso a datos, seguridad y medios de acceso. Sus drivers para Java son muy completos y constantemente se actualizan para corregir errores o mejorar el desempeño. Escala también desde una PC, hasta Mainframes basados en Unix o Linux.

INFORMIX

Informix

Este producto es de relativamente reciente adquisición por parte de IBM y por lo mismo tiene un amplio soporte de Java. IBM tiene serios problemas en cuanto a que Informix compite directamente con su producto estrella que es DB2, por lo que el futuro de Informix es incierto. Sin embargo, informix es una de las mejores bases de datos que hay en el mercado.

Microsoft SQL Server

SQL Server

Contra lo que se pudiese pensar, Microsoft el anti-Java, tiene soporte al 100% y certificado por Sun de Java en todas sus versiones de SQL Server. Los drivers se publican a la par que sale una nueva versión de SQL Server y son muy eficientes en su desempeño. El único inconveniente quizás para SQL Server es que es solo para plataformas Windows. Y aunque actualmente Windows soporta hasta 32 CPU's de 64 bits, carece de la estabilidad que ha ganado Unix a lo largo de los años.

Tomando la decisión

El elegir un producto depende de las circunstancias de cada organización. Pero para fines individuales las decisiones no son muchas y se simplifican los criterios.

Primero hay que tomar en cuenta, si se necesita un ambiente para Aprendizaje, para Desarrollo o para Producción.

Para aprendizaje es mejor tomar el camino de las versiones de Referencia: Sun SDK y el Sun J2EE SDK, pero solo para iniciar el camino. Conforme se manejen los conceptos de contenedores y paquetes hay que moverse rápidamente a un ambiente de Desarrollo, como el que se describe más adelante.

Para ambientes de Desarrollo y Producción debemos elegir un ambiente que sea lo suficientemente robusto como para soportar carga de trabajo y sea estable. En el caso individual las opciones son obvias, los productos de software libre son lo bastante robustos y se pueden instalar en una PC. En este caso la decisión es Jboss junto con Tomcat.

Para ambientes de Producción, al ambiente anterior solo se agregaría Apache, para facilitar la publicación de páginas y no solo aplicaciones en un sitio.

En cuanto a la base de datos, aplicando el mismo criterio de utilizar software libre, la mejor decisión es PostgreSQL para el desarrollo de aplicaciones y su instalación en producción. Ya que tiene un mejor soporte en cuanto el volumen y la demanda de datos crece.

Para las empresas el tomar decisiones de los productos a utilizar es sumamente complejo ya que influyen variables no solo tecnológicas sino también comerciales y hasta políticas. Sin embargo, también hoy en día las empresas sufren por conseguir las licencias de su software, por lo que cada vez más son las empresas que empiezan a experimentar con software libre. Cada vez es más frecuente encontrar empresas corriendo sus sitios Web en Apache, sus páginas JSP en Tomcat y sus EJB's en Jboss. En cuanto a la base de datos les cuesta más trabajo tomar la decisión, pero poco a poco el mercado se está moviendo hacia el software libre.

Una de las razones también para este cambio es que paradójicamente, al menos en Estados Unidos y Europa, hay más gente capacitada para dar soporte del Software Libre que para los productos comerciales. Esto se debe principalmente a que en el Software Libre el soporte no está atado a una empresa. Como sí ocurre con las licencias comerciales. Normalmente las empresas que dependen del soporte de alguna de las grandes empresas de sistemas, están sujetas a sus precios y su disponibilidad del soporte. Y en algunos casos, dado que el soporte muy especializado viene del extranjero deben pagar los costos de traslado y viáticos.

Y por estas razones, que significan finalmente dinero, cada vez más las empresas voltean hacia los productos de software libre. Y anteriormente se ha visto que las tendencias de Estados Unidos y Europa, tarde o temprano llegan a países como el nuestro.

Ahora, todo estos conceptos es tiempo de ponerlos a la práctica. Por lo que en el próximo capítulo se hará una breve descripción del ciclo de desarrollo de una aplicación utilizando todos estos conceptos y productos. Ya que todos estos productos son independientes uno de otro y es confuso a veces el entender como es que se integran todos, o cual es la mejor manera de hacerlo. Además, de que casi siempre se pierde uno los detalles de por donde se empieza el desarrollo y cuáles son los pasos específicos que se deben dar con las herramientas para producir el producto final deseado.

Capítulo 8. Aplicación Ejemplo

Es tiempo de implementar en código los conceptos mostrados en esta Tesis.

Construiremos el código para el Evento #1 de esta aplicación que es el permitir a un Alumno presentar una Solicitud de Inscripción a una materia.

Los pasos son los siguientes:

a) De la fase de diseño detallado requerimos:
   - Diagrama del flujo de las pantallas para atender este evento. En cada pantalla se indican los eventos aplicativos que puede disparar el usuario desde cada pantalla. Aquí hay que diferenciar lo que es un Evento de Negocio de un Evento Aplicativo. El evento aplicativo es cuando el usuario da click en la pantalla sobre un botón o link de nuestra aplicación para disparar una acción del sistema.
   - También necesitaremos la descripción de los servicios que deberá implementar nuestro Session Bean que encapsulará el acceso a la base de datos.

b) Luego procederemos a la construcción de nuestra aplicación. Esta construcción se puede hacer en paralelo al menos en 2 partes:
   - La primera es el desarrollo de la parte visual, es decir, todas las páginas JSP de nuestra aplicación. Así como los Action de control que en un principio irán vacios.
   - La segunda es el desarrollo del modelo del negocio, es decir los Entity Beans y el Session Bean.

c) Cada componente se puede probar por separado, aunque para probar el Session Bean será necesario codificar un cliente java que después será descartado.

d) Una vez probado cada componente se procede a ensamblarlos y empaquetarlos.

e) Los componentes empaquetados se instalan en nuestro servidor aplicativo.

Cada una de estas actividades se descompone en otras varias. A continuación se detalla este proceso:

Diseño Detallado

El flujo que se espera para el Evento de Solicitud de Inscripción es el siguiente:

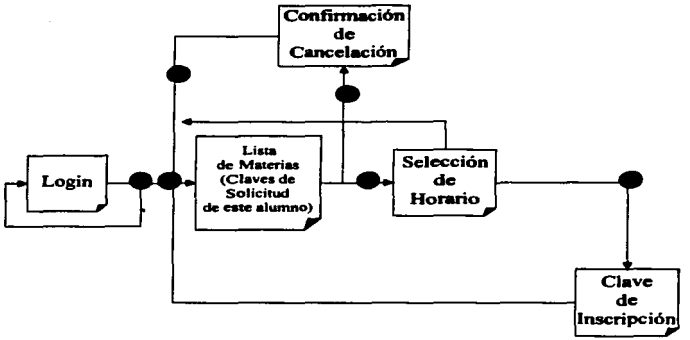


Figura 60 Navegación esperada de aplicación ejemplo.

En este diagrama los puntos representan Acciones que deben llevarse a cabo antes de presentar la siguiente página:

a) Validar el usuario-password

b) Preparar la lista de materias disponibles y el resumen de solicitudes hechas por el alumno actual.

c) Preparar la información detallada de la solicitud que desea cancelar el alumno.

d) Cancela la solicitud seleccionada por el alumno.

e) Preparar la lista de horarios disponibles para la materia seleccionada.

f) Dar de alta la solicitud, asignándole una clave de inscripción y procesando la solicitud de inscripción.

Cada una de estas acciones se codificarán en una clase ActionForm de Struts y se apoyarán en el Session Bean para generar la información a ser desplegada por las JSP's.

De estas acciones podemos definir cuáles son los servicios que debe proveer el Session Bean (Se omite aquí la descripción del mecanismo de seguridad el cual fue ejemplificado en el capítulo 4):

ManejadorSolicitudesBean:

- getMaterias: Devuelve un Collection con la lista de materias disponibles.
- getSolicitudes: Devuelve un Collection con la lista de solicitudes registradas por el alumno actual.
- getSolicitud: Obtiene la información completa de una solicitud.
- cancelaSolicitud: Cancela la solicitud indicada.

- getHorarios: Devuelve un Collection con la lista de horarios disponibles para una Materia.
- agregaSolicitud: Registra una nueva solicitud en el sistema.

Para cada tipo de dato devuelto entre el Session Bean y los Action será necesario establecer un Value Object de por medio. Por lo tanto, necesitaremos 3 Value Objects:

- Solicitud
- Materia
- Horario

Componentes de la Capa Web

El inventario de componentes es el siguiente:

1 Archivo de configuración de Struts

5 Páginas JSP

- Login.jsp
- ListaMaterias.jsp
- SelecciónHorario.jsp
- ConfirmacionCancelacion.jsp
- ClaveInscripcion.jsp

6 ActionForm

- LoginAction.java
- PrincipalAction.java
- ConfirmaCancelacionAction.java
- CancelaSolicitudAction.java
- ListaHorariosAction.java
- AltaSolicitudAction.java

3 Value Objects

- Solicitud
- Materia
- Horario

1 Form Value (único para todas las JSP's)

El código de este ejemplo se encuentra en el proyecto /SolicitudInscripcion.

Componentes de la Capa EJB

El inventario de componentes EJB:

1 Session Bean de Validación de usuarios (AutorizaAcceso), el cual se auxilia con el Entity Bean:

- AlumnoBean para autenticar el acceso del alumno en cuestión.

1 Session Bean ManejadorSolicitudes, el cual implementa los servicios de la aplicación descritos en la sección de Diseño Detallado de este capítulo. Este a su vez se auxilia de los Entities:

- MateriaBean
- HorarioBean
- SesionBean
- AlumnoBean
- PrioridadBean
- SolicitudBean

2 Clientes temporales para probar los Session Beans, uno para cada Session Bean.

El código está en los proyectos /ValidacionUsuarios y /ManejadorSolicitudes.

Ensamblado y Empaquetado

Para ensamblar ambas capas, es necesario incluir el código de los clientes temporales a los Action respectivos de acuerdo al Session bean que requieran acceder.

En otras palabras los Action deben:

a) Primero, obtener la referencia al Session Bean del cual solicitarán un servicio. Este código es genérico y conviene ponerlo en una clase de utilerías del proyecto.

b) Después invocar el método correspondiente de acuerdo al servicio que desean invocar. Este código es específico para cada Action.

Una vez conectados de esta manera, el siguiente paso es empaquetar la aplicación.

Primero se genera un paquete para la capa Web, conocido como el War.

Posteriormente se genera otros 2 paquetes para la capa EJB, conocidos como JAR, en este caso es uno por cada Session Bean utilizado.

Ambos paquetes se empaquetan nuevamente pero ahora juntos en un EAR. Aunque todos estos pasos se pueden ejecutar manualmente, es mejor construir un script con Ant, que lo haga por nosotros, así se evita el repetir pasos constantemente. También dependiendo del Servidor Aplicativo que se desee utilizar es necesario codificar los Deployment Descriptors de cada paquete mencionado (War, Jar y Ear).

En el proyecto /SolicitudInscripcion se encuentra el script de Ant para ejecutar estos pasos. Los comandos son:

Ant compile: para compilar la aplicación

Ant package: para generar el EAR.

Instalación

Antes de instalar la aplicación es necesario asegurarse que la base de datos PostgreSQL está ejecutando.

Después se ejecuta el comando:

Ant createTables: para crear las tablas en la base de datos

El paquete EAR generado anteriormente se instala en nuestro Servidor Aplicativo. En nuestro caso, el archivo solcInsc.ear se copia al subdirectorio $JBOSS_HOME/deploy para hacer esta operación. Donde $JBOSS_HOME es el directorio en donde está instalado Jboss.

Una vez instalada la aplicación, se puede acceder a esta mediante un Web Browser conectándose a la dirección:

http://localhost:8080/solcinsc

Conclusiones

El advenimiento del software libre en la industria de la computación ha significado toda una revolución en cuanto al desarrollo de estándares y aplicaciones. Pero el mayor impacto ha sido demostrar que unidos los desarrolladores son más fuertes que cualquier corporativo. El software libre ha demostrado ser superior en muchos sentidos, pero principalmente en incentivar el espíritu creativo de los programadores de todo el mundo. Hoy en día se puede polemizar acerca de qué es mejor, Linux o Windows. Pero lo cierto es que Linux ha logrado en 10 años, lo que ha Windows le costó 15 años y en muchos aspectos lo ha superado. Por otro lado, la filosofía principal detrás del software libre que es la colaboración y además el apego a estándares, ha demostrado que es el mejor camino para el desarrollo de la industria.

Cuando inicié a escribir esta tesis la máquina virtual de Java estaba en la versión 1.3.0, actualmente nos encontramos en la versión 1.4.1_01. Muchas nuevas funciones se han incorporado a la máquina anterior y no se diga comparada con las versiones 1.2 y 1.1. Hoy en día es posible programar procesos multi-hilos, con soporte de Entrada-Salida asíncronas y uso de expresiones regulares tipo Perl, adicionalmente se cuenta con soporte nativo de XML, de protocolos de red como Sockets TCP-IP y otros protocolos de intercomunicación como son http, CORBA ó IIOP. Todo esto se ha logrado gracias a que ahora el estándar Java se construye con la participación de toda la industria, excepción claro está de Microsoft que sabe que su monopolio corre un gran riesgo.

Pero por otro lado, también están los derechos de los consumidores que somos todos. La realidad es que una empresa que monopoliza el mercado solo le conviene a la misma empresa. El resto se tendría que conformar con ser espectadores y someterse a las políticas del monopolio. Obviamente esto no puede ser. Disponer del código fuente de nuestro software nos permite evolucionar y ser dueños de nuestra tecnología. La falta de tecnología es uno de los factores que debilitan a un pueblo y lo hacen dependiente. Como naciones de tercer mundo no podemos, ni debemos conformarnos con ser compradores de la tecnología sin ser conocedores y desarrolladores de la misma. De otra manera nuestra dependencia se prolongaría más y más.

El primer paso hacia nuestra "autosuficiencia informática" es en la construcción de aplicaciones. Hoy en día hay mucho desarrollo de aplicaciones en México, pero todo este desarrollo se limita a emplear herramientas propietarias. En lo que falta mucho por avanzar es en la construcción de aplicaciones con software libre, de tal manera que este fuese el primer paso, para después convertirnos en parte del movimiento de software libre. Y el mejor punto de inicio es precisamente en las Universidades. Y aunque el objetivo de la Universidad es difundir el conocimiento y no el enseñar productos, debemos llevar a la práctica los conocimientos adquiridos promoviendo el uso de productos apegados a estándares y de los cuales poseamos el código fuente. Y mejor que poseer el código fuente, código que podamos modificar y extender en su funcionalidad. El software libre ha demostrado que no es el hobby de algunos entusiastas, sino una necesidad para construir aplicaciones profesionales y robustas. Si los alumnos de la Universidad Nacional, dominarán las tecnologías básicas para la construcción de aplicaciones estarían a muchos pasos adelante de cualquier otra Universidad.

Además, no concibo que esto pudiese ser de otra manera. Ya que, no propiciar que el alumno se apegue a estándares aceptados en la industria, lo alejaría de la práctica a la que debe estar

vinculado para no producir un profesionista inútil. O en el mejor de los casos, profesionistas que tienen que iniciar un nuevo ciclo de reeducación después de finalizar su carrera.

Hoy en día, el mundo se está polarizando. Por un lado el desarrollo de aplicaciones basadas en Java, por el otro aplicaciones basadas en la plataforma .Net de Microsoft. Sería una necedad decir que Java es superior a .Net, pero a la inversa también. La única diferencia entre ambas plataformas es que mientras una es estándar y aceptada por la industria y además multiplataforma, la segunda es tecnología propietaria y solo soportada en equipos Windows. Entonces debemos preguntarnos, si queremos estudiantes que puedan desarrollar aplicaciones para cualquier plataforma o solo para Windows.

Otro problema es la piratería. Utilizar productos Windows, obliga a los estudiantes a adquirir equipos con Windows, y por lo tanto software para Windows. Antes los estudiantes no tenían alternativa, por lo que recurrían a la piratería. Pero ahora hay muchas alternativas en el software libre. Software que quizá no sea superior al de la marca Windows, pero dado que es gratuito es de mucho mayor beneficio que el otro. Además es de esta misma piratería de la que se ha alimentado el monopolio, puesto que como todos utilizan el software pirata se crea un círculo vicioso, ya que para intercambiar información con el resto del mundo estaba uno obligado a recurrir también a la piratería o adquirir una licencia forzosa. No porque uno hubiese evaluado el software, sino porque "es lo que todo mundo usa".

Entonces, no sigamos alimentando el monopolio, ya sea con licencias legales o con copias piratas y mejor utilicemos software gratuito.

Ahora, Java no es el único lenguaje gratuito. El lenguaje C, también lo es, y Perl, y PHP y otros menos conocidos como Python. ¿Entonces, por qué Java? Desde mi punto de vista por 3 poderosas razones:

La primera, es que Java es un lenguaje elegante y moderno. Moderno dado que está enfocado para construir Objetos pero además para llevarlos más allá, para construir componentes estándar y desacoplados. Todo esto reforzado por la estructura del lenguaje. Al aprender Java, simultáneamente se está aprendiendo Programación Orientada al Objeto y las mejores prácticas del desarrollo de software. Con otros lenguajes esto no es tan natural, ya que no imponen restricciones como las impone Java. Por ejemplo, Java es un lenguaje fuertemente tipificado, eso quiere decir que no puedo convertir una letra a un número de manera directa. Otros lenguajes como C ó Perl, lo permiten. Pero se ha visto que a la larga, esta "facilidad" provoca más problemas de los que resuelve con la supuesta "flexibilidad".

La segunda razón, es que Java posee el soporte de toda la industria. Hoy en día, prácticamente no hay empresa que no tenga productos compatibles con Java. Incluso Microsoft, debe incluir el soporte en su Base de Datos. Su diseño y especificaciones están regidas por el Java Community Process, que es el organismo que rige el futuro del estándar y donde no solo participan empresas sino también individuos.

La tercera y no menos importante: La portabilidad. Java ha demostrado ser 100% portable. Más portable que C, que depende de las librerías de modo Gráfico y de acceso a datos específicas para cada plataforma en la que ejecute, además de que está muy ligado a Unix. Además en Java, son también portables las conexiones a recursos externos, como por ejemplo el acceso a bases de datos es mediante el estándar JDBC. Este estándar es un solo estándar para acceder a cualquier base de datos. En el caso de cualquier otro lenguaje no hay un estándar similar. Esta portabilidad cada vez es más evidente al surgir cada vez más aplicaciones escritas en Java 100% puro, las cuales ejecutan en cualquier plataforma que soporte Java.

Por otro lado, Java no es una plataforma acabada, es una plataforma en constante evolución. Y ese es otro punto interesante de la plataforma, que tiene un camino trazado hacia el futuro, principalmente a la construcción de bloques cada vez de más alto nivel basados en las tecnologías básicas actualmente en existencia. Un ejemplo de esta evolución lo encontramos en los Servlets y las JSP's. Primero surgieron los Servlets, los cuales son componentes especializados en procesar peticiones de HTTP lo cual implicaba generar HTML. Pero la generación de HTML era explícita, es decir, el programador debía codificar el HTML tal cual como se deseaba que se generara internamente en el programa. Un cambio al formato HTML implicaba modificar el código y recompilar. Las JSP's vinieron a resolver este problema integrando HTML y Java desde el punto de vista del desarrollador. Pero tras bambalinas lo que ocurre es que la JSP se convierte a un Servlet y luego este se compila para generar un programa compilado y eficiente. En otras palabras, la tecnología de JSP's está construida sobre la tecnología de los Servlets, los cuales a su vez están construidos sobre la tecnología del Web Container, el cual a su vez está construido sobre la tecnología de la Plataforma Java. Nada de esto ha sido coincidencia.

De esta misma manera, ahora los componentes de acceso a bases de datos están en evolución. Cada vez es menos el código que se requiere escribir para acceder a una base de datos. Y promete ser menos cada vez. La idea es llegar a un punto, en el que por parámetros se indique cual es la tabla y las columnas que se desean acceder desde un programa de una manera declarativa sin tener que escribir código. Un ejemplo de esta tecnología es el framework BC4J, el cual está construido sobre JavaBeans y sobre Enterprise JavaBeans. Hoy en día este componente es tecnología propietaria de Oracle, pero no sería extraño que Oracle lo liberara como software libre. Y si no sucediera así, no es muy difícil que surja otro framework con similar funcionalidad.

En resumen, estamos en el mejor momento para adoptar Java como la tecnología eje que integre en una sola plataforma el resto de las tecnologías que existen en la industria. Java viene a ser un pegamento que nos permite unir Servidores Web, con servidores de Email, o con Servidores de Seguridad, acceso a bases de datos y en general todo lo que se necesite para construir aplicaciones poderosas y que resuelvan las necesidades de todos. Es una plataforma que ha llegado a un punto de madurez que lo hace superior a muchos lenguajes modernos y que todavía promete mucho más.

Muchos Consultores de Informática se preguntan hoy, cuál será el futuro del desarrollo de aplicaciones. Un mundo de Java o un mundo de .Net... la respuesta para mí es simple, el futuro no hay que esperarlo, hay que decidirlo. Y la decisión hay que tomarla ahora y no cuando el monopolio nos lo diga. Así que no hay que esperar más, Java está en Internet ¡Ya! ¡En este instante! No hay que ir a la tienda a comprar una licencia y una computadora a la que también le tengo que comprar otra licencia del sistema operativo y luego otra máquina para instalar una base de datos de la cual requiero otra licencia y por ende otra más del sistema operativo. Ni tampoco tengo que obligar a mis usuarios a que hagan lo mismo para conectarse a mi aplicación. Java y todos los productos relacionados están al alcance de nuestra mano. Nuestra única limitación es no querer aprender algo nuevo y salirse del esquema que hasta ahora creíamos que era el único.

Los estudiantes actuales y egresados de Ingeniería en Computación, tenemos la obligación de ser la punta de lanza que abra el camino a las nuevas tecnologías y que demostremos que si podemos emplear tecnología libre, con mucho más facilidad podremos dominar las tecnologías propietarias.

Así que debemos dar los pasos hacía una adopción de la tecnología de desarrollo de software a todos los niveles. Participando en proyectos de desarrollo interno primero y posteriormente en

los proyectos públicos de desarrollo. El primer paso es aprender a utilizar las herramientas de desarrollo que existen ya en el mercado. Y posteriormente mejorarlas o desarrollar unas nuevas.

La plataforma Java, permite integrar todo el cúmulo de tecnologías de sistemas que existen hoy en día. Así que un objetivo alcanzable a corto plazo es la implementación de fábricas de desarrollo de software basadas en esta tecnología. Adoptemos Linux como nuestro sistema operativo, PostgreSQL como nuestra base de datos y Sun ONE Studio como nuestra herramienta de desarrollo. Aprendamos los conceptos básicos de análisis de sistemas y capacitémonos en el empleo de los patrones de diseño tanto de componentes de software como patrones de diseño de bases de datos. Y de esta manera estaremos en capacidad de desarrollar casi cualquier aplicación de negocios distribuida, he incluso estaremos en capacidades por encima del 80 ó 90% de las compañías de software que actualmente existen en el país.

En los países industrializados, los grandes proyectos de software que perduran son desarrollados en las universidades. Esto nos debe dar una idea del gran potencial que hay en nuestra Universidad, de la cantidad de jóvenes que tienen deseos de hacer grandes cosas y que pueden hacerlo con una orientación en el camino correcto. No desperdiciemos esta oportunidad y empecemos a alientar que este tipo de iniciativas prosperen en nuestras escuelas. La Universidad Nacional, debe ser un recinto no solo de aulas físicas, sino también un punto de reunión virtual que mediante Internet permita la integración de grupos de trabajo que desarrollen software de todo tipo. Las aplicaciones de negocio pueden ser un buen principio.

"Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the universe trying to produce bigger and better idiots. So far the universe is winning."

"La programación hoy en día es una carrera entre los ingenieros de software luchando por construir más grandes y mejores programas a prueba de idiotas, y el universo tratando de producir más grandes y mejores idiotas. Hasta el momento el universo va ganando."

-- Rich Cook.

Cita tomada de "The Java Programming Language" de Ken Arnold, James Gosling y David Holmes.

Glosario

Ant. Utilería escrita en Java, con funciones similares al antiguo "make" de Lenguaje C. Se utiliza para ejecutar scripts que automatizan la compilación, empaquetamiento e instalación de aplicaciones Java.

Applet. Componente escrito en Java, diseñado para ser ejecutado dentro de una máquina virtual de un Browser Web. Debido a las limitaciones de HTML se requería de un mecanismo para presentar elementos gráficos al usuario de aplicaciones Web. Los applets son la respuesta a esta necesidad.

BMP. Bean Managed Persistence.

Bean Managed Persistence. Uno de los tipos de Entity Beans. Se caracterizan por ser componentes cuyo código de acceso y almacenamiento en la base de datos los proporciona el programador, contra los Container Managed Persistence, donde es el contenedor EJB el cual genera el código necesario.

CMP. Container Managed Persistence.

Container Managed Persistence. Uno de los tipos de Entity Beans. Se caracterizan porque el contenedor EJB genera el código necesario de acceso y almacenamiento de la base de datos requerido para las propiedades de este componente.

CMR. Container Managed Relationships.

Container Managed Relationships. Es el estandar que se utiliza para describir las relaciones entre Entity Beans, de tal manera que el contenedor EJB genere el código necesario para mantener lógicamente estas relaciones durante la ejecución de la aplicación.

Componente. En programación Orientada a Objetos, son programas que encapsulan cierta funcionalidad y que a diferencia de los objetos comunes, poseen un mecanismo estándar de comunicación con otros componentes igualmente apegados a dicho estándar. Este mecanismo de comunicación es importante porque en un sentido amplio permite la integración de piezas de software escritas en diferentes lenguajes, contra simples objetos que solo admiten la invocación desde otros objetos escritos en el mismo lenguaje.

CORBA. Common Object Request Broker Arquitecture, es un estándar de comunicaciones para interconectar objetos en plataformas heterogéneas. En otras palabras, permite invocar métodos de objetos de manera remota, los cuales pueden estar escritos en otros lenguajes y operando en sistemas operativos diversos. En realidad, CORBA es un conjunto de estándares que definen una serie de componentes los cuales permiten dicha comunicación. Uno de estos componentes es el Object Reques Broker que acopla los objetos a una plataforma común.

DNS. Domain Name Server. Servidor de nombres de dominio. Es un servidor que almacena nombres de equipos en internet o una intranet y los relaciona con su IP. De esta manera los búsquedas de equipos se hacen a través del nombre del equipo y no a través de una dirección numérica.

Dominio. Se le llama así a la parte final del nombre de un equipo. Por ejemplo, www.google.com es el nombre completo de un equipo que se llama www dentro del dominio google.com. Dentro de dicho dominio puede haber otros equipos como group.google.com, news.google.com, etc. Sirve para propósitos de seguridad y administración.

EJB. Enterprise Javabean.

Enterprise JavaBean. Son componentes cuyo propósito es proporcionar servicios transaccionales a las aplicaciones distribuidas. Pueden recibir invocaciones mediante el protocolos RMI-IIOP lo cual les permite recibir invocaciones locales o remotas, todo dentro de un ambiente de seguridad controlado.

Entity Bean. Es un tipo de Enterprise Javabean. Se caracterizan porque están asociados a entidades en una base de datos. Es decir, son el medio estándar para acceder a una base de datos.

GNU Public License. Es la licencia que define los permisos públicos sobre el software desarrollado por la organización GNU de software libre.

GPL. GNU Public License.

HTML. Es el lenguaje basado en texto utilizado en internet para describir páginas visibles en un Web Browser. Permite incluir textos con formato e imágenes.

HTTP. Es el protocolo de comunicaciones utilizado entre los Web Servers y Web Browsers para distribuir páginas escritas en HTML.

IIOP. Internet Inter-ORB Protocol. Es un protocolo de comunicaciones entre Object Reques Brokers (ORB). Un ORB es una interfaz que se encarga de acoplar invocaciones entre objetos remotos y es una parte del estándar CORBA, el protocolo IIOP es un mecanismo de transporte para enviar de un ORB a otro las invocaciones de los componentes que lo requieren.

J2EE. Java2 Enterprise Edition. Es la arquitectura de componentes definida para soportar aplicaciones distribuidas y con soporte transaccional para internet.

J2SE. Java2 Standard Edition. Es la definición de la plataforma Java básica.

Java Application. Es un tipo de programa que puede ser construido en Java. Son los programas autocontenidos más simples, ejecutan en su propia Java Virtual Machine. Pueden tener una interfaz de línea de comandos o gráfica dependiendo de sus código interno.

Java Virtual Machine. JVM, máquina Java, Runtime Machine. Es el programa que permite la ejecución de componentes Java en un sistema operativo específico. Existe una Java Virtual Machine para cada sistema operativo. Java es un lenguaje psudointerpretado: Primero el compilador convierte el código fuente a código binario universal que se conoce como javabytes, luego la JVM lee los javabytes y los interpreta para su ejecución.

Javabean. Es un componente definido por la especificación conocida como Javabeans Component Architecture. La idea es construir componentes que ejecuten en cualquier sistema operativo dentro de cualquier ambiente. Estos componentes soportan una interfaz que permite mediante un mecanismo conocido como Reflexión, el examinar un componente y determinar sus métodos y miembros públicos. También poseen mecanismos de configuración persistente, de tal manera que se pueden configurar durante diseño o durante ejecución del componente de acuerdo a la elecciones que hagan los programadores que los desarrollan y los que los usan.

JavaMail. Es una librería que permite la conexión a servidores de mail que manejen los protocolos: SMTP, POP e IMAP.

JAXP. Java API for XML Parsing. Librería para leer y escribir XML en java.

Jboss. Contenedor EJB gratuito desarrollado por JBoss Group. Este contenedor se acopla con Tomcat o Jelly (contenedores Web) para tener un Servidor J2EE completo.

JDBC. Java Database Connectivity. Librería para conectarse y ejecutar SQL en bases de datos que soporten el estándar SQL ANSI 92.

JDK. Java Development Kit. Es el conjunto de herramientas básicas para desarrollar programas Java. Incluye el compilador, el Runtime de java y la documentación.

JMS. Java Message Service. Estándar para soporte de comunicaciones asíncronas en Java. Permite enviar y recibir mensajes desde componentes Java mediante un mecanismo de colas, el cual puede ser persistente o no.

JNDI. Java Naming and Directory Interface. Librería para acceder a sistemas de nombres y directorios de una manera uniforme desde Java. Mediante JNDI es posible consultar servidores DNS, LDAP y Filesystems de la misma forma independientemente del tipo de servicio consultado.

JSP. Java Server Pages. Componentes de la plataforma J2EE escritos en HTML con Java incrustado. Generan páginas HTML con contenido dinámico.

JTA. Java Transaction API. Establece una interfaz estándar entre un Transaction Manager y los participantes dentro de una transacción distribuida. Normalmente, los programadores no necesitan preocuparse de esta interfaz a menos que quieran desarrollar su propio contenedor o driver de base de datos.

LDAP. Lightweight Directory Access Protocol. Es un protocolo de internet que permite la consulta de información en un directorio con una estructura de árbol. Los servidores LDAP almacenan esta información de diversas maneras, pero deben apegarse al estándar para su consulta.

RMI. Remote Method Invocation. Estándar de cómo se debe construir una invocación de un método de un objeto remoto. A diferencia de IIOP, RMI es un estándar a nivel de formato de la invocación, mientras que IIOP es un estándar de transporte, es decir de como se va a transferir esa invocación en la red.

RMI-IIOP. RMI over IIOP. Es el uso de invocaciones RMI sobre el protocolo de transporte IIOP.

Runtime. Java Runtime, JVM, Java Virtual Machine. Es el intérprete de javabyte codes.

SDK. Software Development Kit. Sinónimo de Java Developer Kit. Actualmente es el término preferido.

Servlet. Componente de la plataforma J2EE escrito en Java para generar HTML.

Session Bean. Uno de los tipos de Enterprise Javabeans. Se caracterizan por no ser persistentes. Sirven para definir flujos de información y reglas del negocio. También se utilizan para integrar en una sola transacción varias transacciones dispersas.

SFSB. Statefull Session Bean. Uno de los tipos que existen de Session Bean. Mantienen un objeto de sesión por cada usuario que se conecta a la aplicación.

SLSB. Stateless Session Bean. Uno de los tipos que existen de Session Bean. Se caracterizan por no guardar información de estado (sesión) de los clientes que se conectan a este componente.

Struts. Framework de desarrollo que facilita la implementación del patrón de diseño conocido como Model-View-Controller para aplicaciones Web construidas en Java.

TCP/IP. Conjunto de protocolos de red que permiten la interconexión de computadoras. Es el principal conjunto de protocolos que permite el funcionamiento de la Internet.

UDP. Protocolo de comunicaciones que es parte del conjunto de protocolos TCP/IP.

VM. Virtual Machine. Es el entorno virtual en el que ejecutan los programas de java. Para permitir que los programas java ejecuten en cualquier sistema operativo, la máquina virtual crea un entorno de ejecución especial para java que aisla la aplicación de los detalles del sistema operativo.

XML. eXtended Markup Language. Es un lenguaje que permite describir información mediante marcadores conocidos como tags (etiquetas). Este lenguaje a partir de conceptos

muy simples permite separar el contenido de la presentación de la información, lo que permite un manejo de la información más dinámico e "inteligente" de los programas de computadora.

Apéndices

A. Código fuente

Todo el código fuente se anexa en copia en CD.

B. Sitios Web de interés

Sitio	Contenido
www.mandrake.com	Linux Mandrake
java.sun.com	Sitio oficial de Sun para Java
jakarta.apache.org	The Jakarta Project Desarrolladores de Apache, Tomcat, Struts y otros proyectos relacionados con el soporte para aplicaciones Web.
www.jboss.org	The Jboss Group.
http://otn.oracle.com/tech/java/content.html	El Java Center de Oracle
http://www-106.ibm.com/developerworks/java/?loc=dwmain	Java Technology Zone de IBM
www.javafaq.com	Página con ligas a varios Faq's de java.
www.linux.com	Recursos para Linux
www.linux.org	Recursos para Linux
www.theserverside.com	Todo lo relacionado con Patrones de Diseño aplicados a Java en el Servidor.
http://www.fawcette.com/javapro/	JavaPro Magazine
http://www.sys-con.com/java/	Java Developers Journal
www.postgresql.org	Sitio oficial de PostgreSQL.

Bibliografía

Análisis de Requerimientos

Essential System Requirements
A Practical Guide to Event-Driven Methods
Autor: Bill Wiley
Diciembre 1999

Java Standard Edition

Para totalmente novatos en Programación Orientada a Objetos y Java:

Java2 for Dummies (For Dummies)
Autor: Barry Burd
Octubre 2001

Para personas con conocimientos de Programación Orientada a Objetos:

Java in a Nutshell (4th Edition)
Autor: David Flanagan
O'Reilly & Associates
4a. edición (Abril 2002)

Para futuros Gurús de Java:

The Java Programming Language, Third Edition
Autores: Ken Arnold, James Gosling, David Holmes
3a. Edición (Junio 15, 2000)

Java Enterprise Edition

Acerca de JSPs y Servlets (capa Web):

Core Servlets and JavaServer Pages (JSP)
Autor: Marty Hall
1a. edición (Mayo 26, 2000)

Acerca de EJB's:

Enterprise JavaBeans (3rd Edition)
Autor: Richard Monson-Haefel
3a. edición (Octubre 15, 2001)

Si solo quieres un libro de J2EE (mismos temas, menos detalles):

Professional Java Server Programming J2EE Edition
Autores: Wrox Multi Team, et al

Wrox Press Inc
1a. edición (Septiembre 2000)

Diseño de Aplicaciones

Acerca de la Plataforma J2EE:

Designing Enterprise Applications with the J2EE(TM) Platform
Autores: Inderjeet Singh, Mark Johnson, Enterprise Team (Editor), Enterprise Team
2a. edición (Marzo 25, 2002

Acerca de Patrones de Diseño

Core J2EE Patterns: Best Practices and Design Strategies
Autores: Deepak Alur, et al
Prentice Hall PTR
1a. edición (Junio 26, 2001)