CIMAT

Centro de Investigación en Matemáticas, A.C.

Arquitectura para colaboración

T E S I S

Que para obtener el grado de

Maestro en Ciencias de la Computación

P r e s e n t a :

Edgar Bernal

Director de Tesis:

Dr. Cuauhtémoc Lemus Olalde

Guanajuato, Gto. Junio de 2010

TABLA DE CONTENIDO

1. INDICE DE FIGURAS

RESUMEN

El objetivo de esta tesis es describir de manera práctica el desarrollo de una herramienta de colaboración estadística que permite compartir diferentes análisis en un ambiente web con las tecnologías actuales. Esta aplicación se inicia desde su nivel arquitectural y describiendo de manera general una de sus aplicaciones y los problemas que esta resuelve en el campo de investigación y la industria interesada en la utilización de este tipo de herramientas y técnicas avanzadas.

En general se intenta dar solución a establecer un ambiente distribuido para compañías e institutos de desarrollo con el propósito de que puedan acelerar la evaluación y adopción de nuevos sistemas desarrollados dentro de la organización. Esto con el fin de tomar ventaja al gestionar el conocimiento de manera centralizada (Highsmith, 1999) dentro de una organización.

La herramienta desarrollada que se implementó fué utilizando las tecnologías más actuales hasta el momento. Utilizando los patrones de diseño necesarios para su correcto funcionamiento, como por ejemplo inversión de control. Entre lo más destacado que cabe mencionar de esta herramienta son el uso de J2EE (Java Platform, Enterprise Edition) (Alur, 2003) como plataforma de programación, spring como controlador de eventos, tomcat como server y oracle como RDBMS (Relational Database Model System). En el contexto muy particular de CIMAT, esta podría ser aplicada como ambiente colaborativo, consolidando los resultados de las tesis de posgrado, proyectos de vinculación de la Gerencias que forman CIMAT. La extensión de esta tesis podría ayudar a tener un control de los trabajos realizados y más aun, una plataforma para desarrollo de nuevas aplicaciones visto y organizado como línea de producto.

CAPÍTULO 1 - INTRODUCCIÓN

A pesar de los esfuerzos por mejorar las practicas de ingeniería y tecnología, en sistemas de software intensivos y robustos, estos continúan presentando altos riesgos y dificultades en su diseño, construcción, instalación y sobre todo en su evolución.

Intentos recientes para manejar estas dificultades se han centrado en la etapa de diseño y evaluación comúnmente llamado como el *nivel arquitectural* del desarrollo del sistema.

Estas innovaciones han venido madurando rápidamente dentro de varias comunidades de investigación y aplicación, mostrando diferentes intereses, influencias, visiones e intenciones. Existe un consenso general sobre la importancia del *nivel arquitectural de los sistemas de desarrollo* y que este nivel consiste en la temprana toma de decisiones sobre la estructura general del diseño, metas, requerimientos y estrategias de desarrollo. El tener una arquitectura contribuye al desarrollo, operación y mantenimiento de un sistema desde su etapa inicial hasta su instalación.

Una arquitectura es definida como la organización fundamental de un sistema, junto con sus componentes, sus relaciones con cada uno y su ambiente, así como los principios que gobiernan su diseño y evolución. Esta definición pretende cubrir una variedad de usos del término arquitectura reconociendo sus elementos principales comunes. Entre estos hay la necesidad de entender y controlar esos elementos del diseño de sistemas como la utilidad el costo y su riesgo. (Lars Lundberg, 2007)

El objetivo de esta tesis es describir de manera práctica el desarrollo de una herramienta de colaboración estadística que permite compartir diferentes análisis en un ambiente web con las tecnologías actuales. Esta aplicación se inicia desde su nivel arquitectural y describiendo de manera general una de sus aplicaciones y los problemas que esta resuelve en el campo de investigación y la industria interesada en la utilización de este tipo de herramientas y técnicas avanzadas.

Por otro lado, este trabajo puede resultar de interés para arquitectos de software, desarrolladores y administradores de proyectos. Se describen las técnicas y representación de arquitecturas que han sido utilizadas satisfactoriamente en desarrollos reales.

UML (Unified Modeling Language) es usado a lo largo de este trabajo. El foco de esta tesis esta en arquitecturas de software de gran-escala. En el capítulo 2 se explica el porqué del trabajo realizado, problema identificado y estrategia o metodología del trabajo de tesis desarrollado. En capítulo 3 se explica el caso de estudio que se estará tomando como base para mostrar la practicidad de lo realizado en la tesis.

En el capítulo 4 se presenta una explicación de cómo las arquitecturas de software contribuyeron en la presente investigación. En capítulo 5 se expone en detalle lenguaje utilizado y plataforma de software. En capítulo 6 se muestran los resultados obtenidos en el transcurso de la investigación así como la propuesta de la arquitectura (bondades, beneficios, limitaciones). En capítulo 7 se comentan los trabajos que se pueden realizar tomando como base la investigación realizada y otras aplicaciones que pudieran considerar de base la arquitectura propuesta. Finalmente, en el capítulo 8 se exponen algunos comentarios finales de la investigación y propuestas. Se agregan algunos anexos donde se muestran de manera breve el uso de UML como herramienta de representación en el diseño de aplicaciones de software, y otro más con una de las metodologías todavía utilizadas hoy en día llamada RUP (Bergström, 2004). Esta metodología es la que se siguió en el desarrollo de la herramienta y cabe mencionar que son de las más formales hasta el momento.

CAPÍTULO 2 - JUSTIFICACIÓN

El problema que intenta dar solución esta tesis es el de establecer un ambiente distribuido para compañías e institutos de desarrollo con el propósito de que puedan acelerar la evaluación y adopción de nuevos sistemas desarrollados dentro de la organización. Esto con el fin de tomar ventaja al gestionar el conocimiento de manera centralizada (Highsmith, 1999) dentro de una organización.

Generalmente en organizaciones donde el manejo de la información y su análisis es primordial, nos encontramos que distintos equipos proporcionan soluciones a diversos problemas utilizando técnicas estadísticas, numéricas, lógicas, computacionales, etc. Muchas veces se resuelve el mismo problema en múltiples ocasiones por el desconocimiento que ya antes se había trabajado en su solución. El problema dentro de la organización es claramente la comunicación entre los diferentes equipos o departamentos y el compartir los resultados o avances obtenidos. No se cuenta con un control estricto de los estudios realizados a la información, el número de problemas resueltos es cuantioso y la mayoría de las veces implica el análisis de grandes cantidades de datos.

Ahora bien, imaginando el mejor de los escenarios, estas organizaciones podrían contar con una herramienta de software configurable de tal manera que cada análisis a estos datos y solución propuesta fueran publicadas, modificadas, mejoradas y extendidas a otros problemas similares. Todo esto de manera transparente para el analista. Esta herramienta podría ser capaz de interactuar con sistemas externos y establecer una comunicación segura a través de una interface Web.

En esta tesis vamos a plantear las implicaciones del desarrollo de una herramienta con estas características desde el punto de vista de su arquitectura. A manera de caso de estudio tomaremos un ejemplo en particular donde se desarrolló una parte de la herramienta para resolver el caso donde la necesidad de publicar análisis y soluciones estadísticas fué el punto focal. El análisis extensivo de gran escala de datos y el trabajo conjunto de equipos de software y estadística fue necesario.

En la Figura 2.1 se pueden fácilmente identificar cuatro secciones, la primera sección muestra los dispositivos con capacidad para interactuar con el sistema. La segunda sección es la base del sistema, es donde se desarrolla la parte que organiza las peticiones a los análisis así como de administrar los recursos del ambiente tanto de datos como de procesamiento. Una tercera sección nos muestra el repositorio de datos

donde se puede almacenar la información a ser analizada. Aquí cabe mencionar que estas secciones no necesariamente están obligadas a estar en una misma infraestructura de hardware. Finalmente, encontramos una cuarta sección, que es básicamente donde se encuentran los sistemas externos a los que esta herramienta hace interface para dar servicio hasta la sección de los dispositivos cliente.

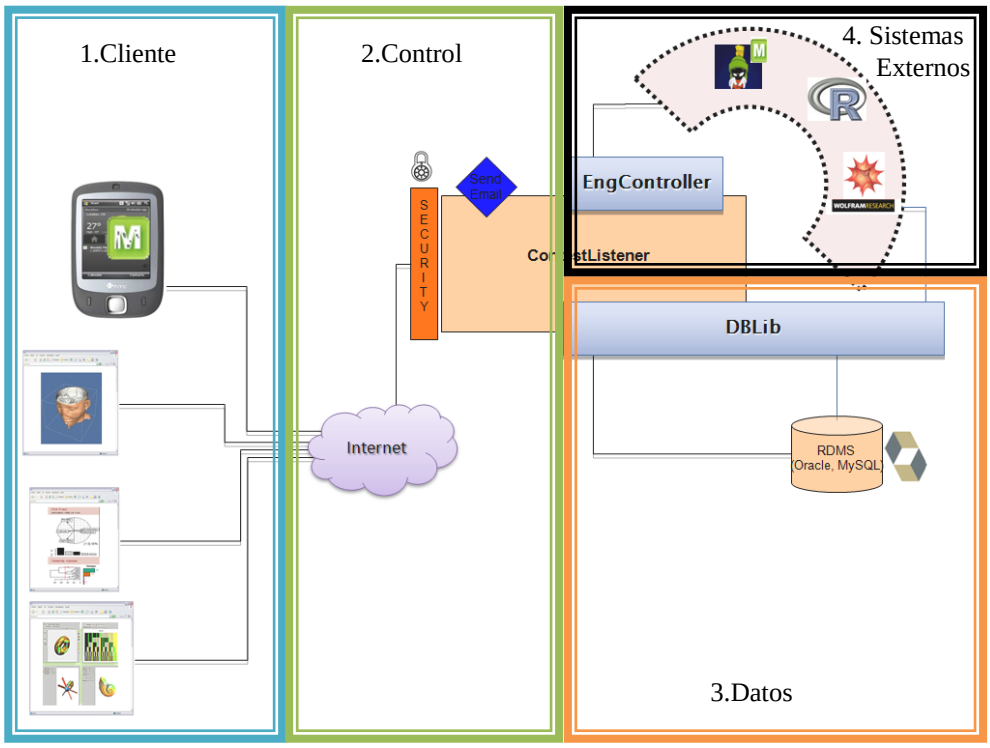


Figura 2.1. Diagrama general de ET2.0

El objetivo principal de la tesis es desarrollar cada una de las secciones. Las secciones 2 y 3 desde el punto de vista de arquitectura y tecnologías de software requeridas para la implementación de esta herramienta. La sección 1, estará lista para funcionar en equipos con acceso a internet. Para la sección 4 usaremos la herramienta llamada R (Spector, 2009). Esta herramienta es un software libre para cómputo estadístico y gráficos. R proporciona una amplia variedad de técnicas graficas y estadísticas (lineales y no lineales, pruebas estadísticas clásicas, análisis de series de tiempo, clasificación, etc.) además de ser altamente extensible. Una de las fortalezas de R es la facilidad con la que puede publicar gráficos de alta calidad. Incluye una forma efectiva de manejar datos y almacenarlos con facilidad, cuenta con un lenguaje de programación simple y con una gran gama de aplicaciones en diferentes dominios.

Como se muestra en la Figura 2.2, lo que estamos logrando con este trabajo es englobar toda esta funcionalidad de R, y extendiéndola para su acceso en un ambiente distribuido, de manera remota y de forma segura, pues la herramienta también está provista de un mecanismo de acceso por roles y usuarios. Es decir, está la posibilidad de restringir a usuarios el uso de análisis específicos de acuerdo a su rol.

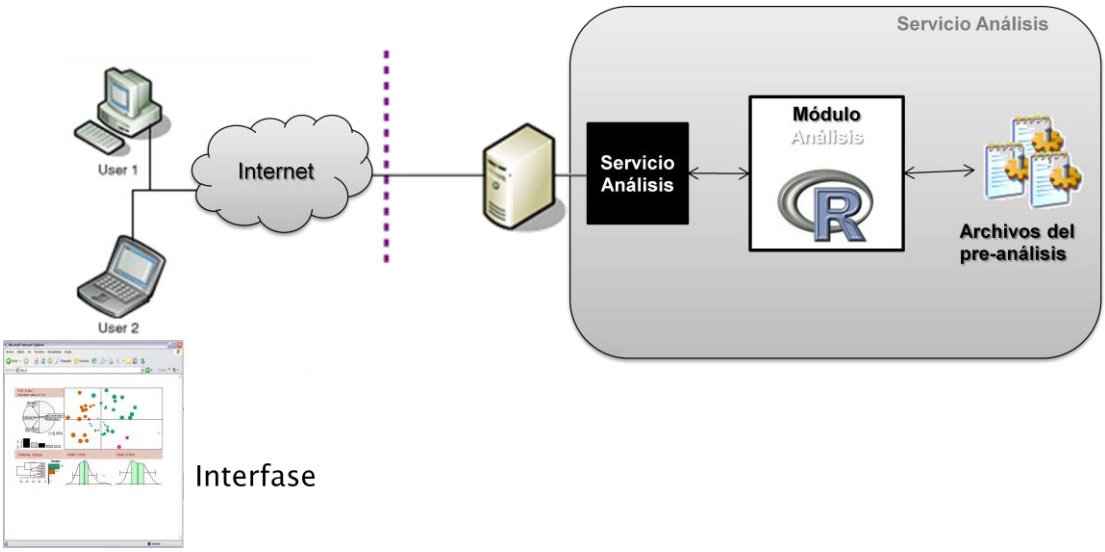


Figura 2.2. Diagrama general de ET2.0 con funcionalidad de R

CAPÍTULO 3 - CASO DE ESTUDIO

Introducción

El sistema que ha sido desarrollado (sistema ET2.0) está provisto de un mecanismo de seguridad el cual valida por medio de nombre de usuario y clave de acceso el uso indebido de personas no autorizadas.

También se cuenta con un mecanismo por medio del cual si cualquier usuario desea ingresar directamente a alguna de las pantallas provistas por el sistema automáticamente será redireccionado a la pantalla de acceso para que proporcione el nombre de usuario y clave de acceso proporcionado.

El sistema almacena información sensible como claves de acceso de manera encriptada utilizando algoritmos avanzados SHA1 (Secure Hash Algorithm 1), por lo cual, solamente el usuario tiene el conocimiento de la clave e inclusive el administrador no tiene acceso a él.

El Sistema ET 2.0 es una herramienta de software que tiene como objetivo agilizar la creación de reportes a partir del análisis de la información de tickets de venta. Está diseñada con una arquitectura MVC (Model View Control), utilizando J2EE (Java Platform, Enterprise Edition), DWR (Direct Web Remoting), AJAX (Asynchronous JavaScript and XML, Holdener, 2008) y ORACLE (Niemiec, 2007).

El presente capítulo es un resumen técnico de la herramienta de software ET 2.0. Contiene información sobre la instalación, configuración, ejecución y desarrollo de la aplicación.

Por otro lado el objetivo de este capítulo es dar una descripción de una de las aplicaciones reales de esta tesis. También puede tomarse como punto de referencia técnica sobre la herramienta de software ET2.0, conteniendo los detalles internos de la aplicación, asi como la instalación y configuración de la misma para alguna otra aplicación. Esto con la finalidad de proporcionar una base reutilizable, evitando así comenzar desde cero cada vez que se quiera hacer alguna otra implementación.

Arquitectura de la aplicación

La arquitectura de la aplicación ET2.0 es una arquitectura J2EE, basada en un modelo de aplicaciones distribuidas de diversas capas o niveles (tier).

- Cliente
- Web
- Back-end

El modelo en el que está basada ET2.0 se llama MVC (Model View Control, Gorton, 2006, Erich Gamma, 1994), que describe esta arquitectura conformada por capas, donde cada capa juega un papel importante; la vista al cliente, el control de los datos y las reglas de negocio pertenecen a cada nivel, de tal manera que el código es más fácil de leer y mantener. ET2.0 define en cada capa el comportamiento y reglas de negocio específicas.

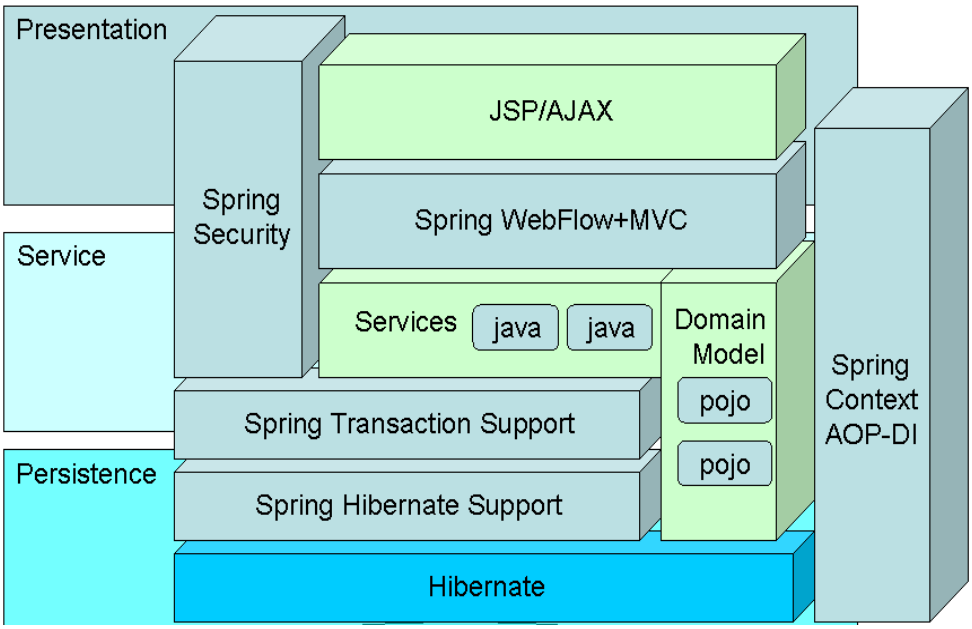


Figura 3.1. Arquitectura de Software de la Aplicacion

Capa Cliente

Para la capa de Cliente hace uso del Navegador web Internet Explorer, siendo la versión 7.0.6001 requisito indispensable para el correcto funcionamiento de la aplicación así como el tener habilitado la ejecución de scripts de cliente (JavaScript).

Direct Web Remoting (DWR)

DWR (Salkosuo, 2008) es el API (Application Programming Interface) que le permite a la aplicación ET realizar llamadas remotas desde el código JavaScript (ejecutándose en el navegador), a los objetos Java (POJOs) del servidor.

DWR también es utilizado para la actualización "inmediatamente" de las páginas web que se muestran al usuario.

En ET2.0 DWR se compone de dos partes: una parte que se ejecuta en el lado cliente (en este caso, el navegador web) y otra parte que se ejecuta en el servidor (en este caso, el contenedor de Servlets).

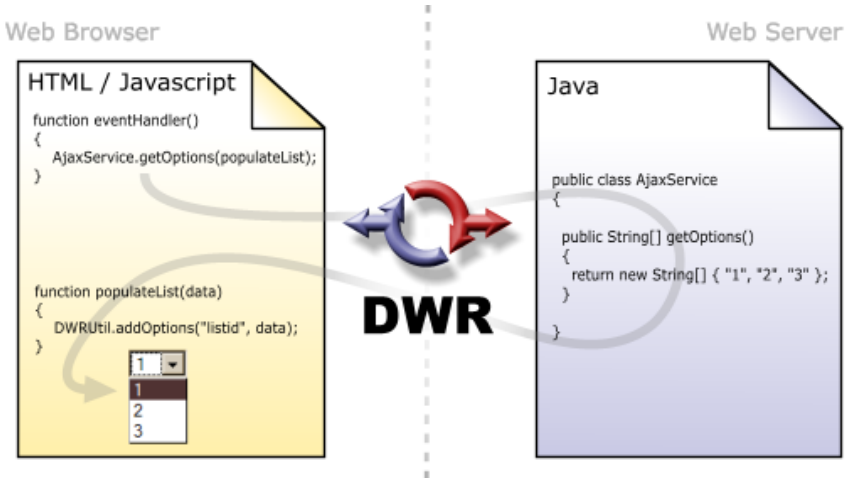


Figura 3.2. Esquema del funcionamiento de DWR

Invocación Remota (Directa) de Métodos

DWR proporciona un envoltorio (wrapper) para permitir la invocación de métodos de objetos Java (POJOs) desde el cliente. El sistema de RPC (Remote Procedure Call) implementado por DWR se basa en un Servlet (uk.ltd.getahead.dwr.DWRServlet), el cual, utilizando la información que le llega a través de la petición HTTP (HTTP Request), se encarga de instanciar los objetos necesarios y de realizar la invocación del método solicitado, pasándole los parámetros enviados desde el cliente. Todas estas operaciones se realizan utilizando el API de reflexión de Java (SPRING).

Configurar el servidor

El motor de DWR es el objeto DWRServlet. En esta clase se centralizan todas las posibles funcionalidades que ofrece la librería DWR: desde la generación del código JavaScript a utilizar en el cliente, hasta el marshalling de tipos (pasando, por supuesto, por la invocación a los métodos remotos).

Como cualquier otro servlet, DWRServlet está declarado (y "mapeado" a alguna URL) en el fichero WEB-INF/web.xml de la aplicación ET.

```
<servlet>
  <servlet-name>dwr-invoker</servlet-name>
  <display-name>DWR Servlet</display-name>
  <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
  <init-param>
    <param-name>debug</param-name>
    <param-value>true</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>dwr-invoker</servlet-name>
  <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

DOJO – Open Source Java Script Library

Otra característica importante de la aplicación es AJAX, la cual permite la comunicación asíncrona entre el navegador y el servidor. Esta podría realizar con el comando Java Script XMLHttpRequest. Sin embargo se eligió el framework Dojo (Russell, 2008) porque provee de una capa de abstracción (dojo.io.bind) para varios navegadores web con la que se pueden usar otros transportes (como IFrames ocultos) y diferentes formatos de datos. De esta forma se pueden obtener los campos que se van a enviar como parámetros del formulario de Análisis de una manera más sistemática.

Sistema de paquetes

Dojo provee de un sistema de paquetes que facilitó el desarrollo modular. El script de inicio inicializa una serie de jerarquías de paquetes de espacios de nombre (io, event, etc.) bajo el paquete raíz dojo.

Los paquetes usados de Dojo contienen múltiples archivos. Cuando el paquete es cargado, sus dependencias son también cargadas.

Capa Web

Como ya se menciono la arquitectura de la aplicación está basada en capas, mientras el diseño de la misma está basado en un Modelo-Vista-Controlador (MVC) el cual es un patrón de diseño bastante utilizado en aplicaciones web por la separación que provee entre los elemento que constituyen una aplicación Web.

Elementos del patrón:

- Modelo: datos y reglas de negocio.
- Vista: muestra la información del modelo al usuario.
- Controlador: gestiona las entradas del usuario.

El flujo que sigue el control es el siguiente:

1. El usuario interactúa con la interfaz de usuario (por ejemplo, el usuario pulsa un botón, enlace, etc.)
2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.
3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el estado de un análisis a leído). Los controladores de la aplicación están estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado de los análisis fallidos del usuario). El modelo no tiene conocimiento directo sobre la vista. Sin embargo, se usa el patrón de observador para proveer cierta induración entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la

vista para que se actualice. Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.

5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

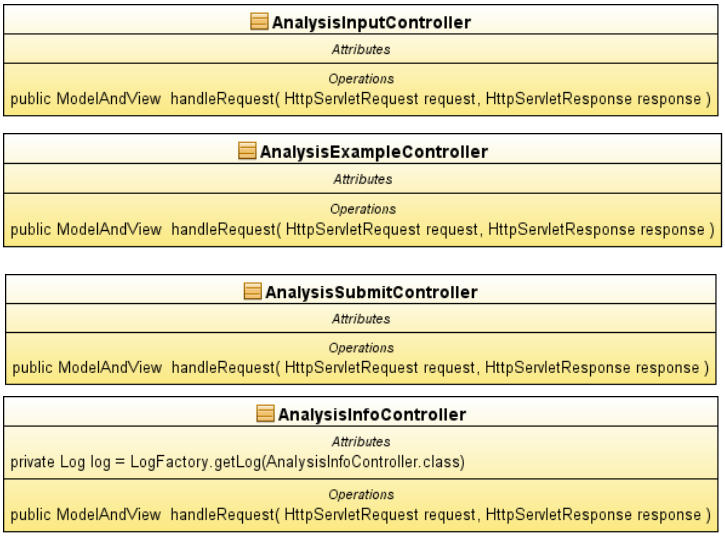
Para la implementación de este diseño se utiliza en ET2.0 el framework SPRING configurado con los siguientes paquetes:

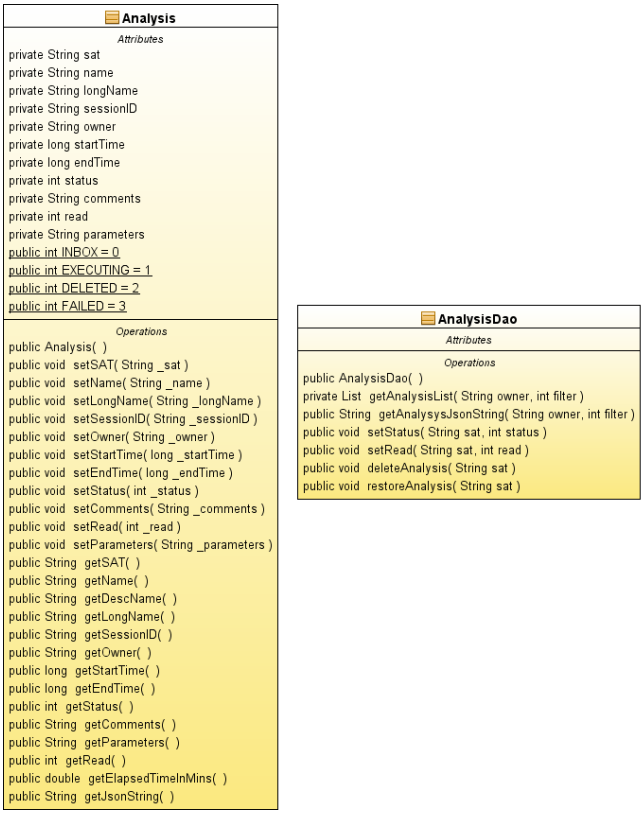
Modelado UML de ET2.0

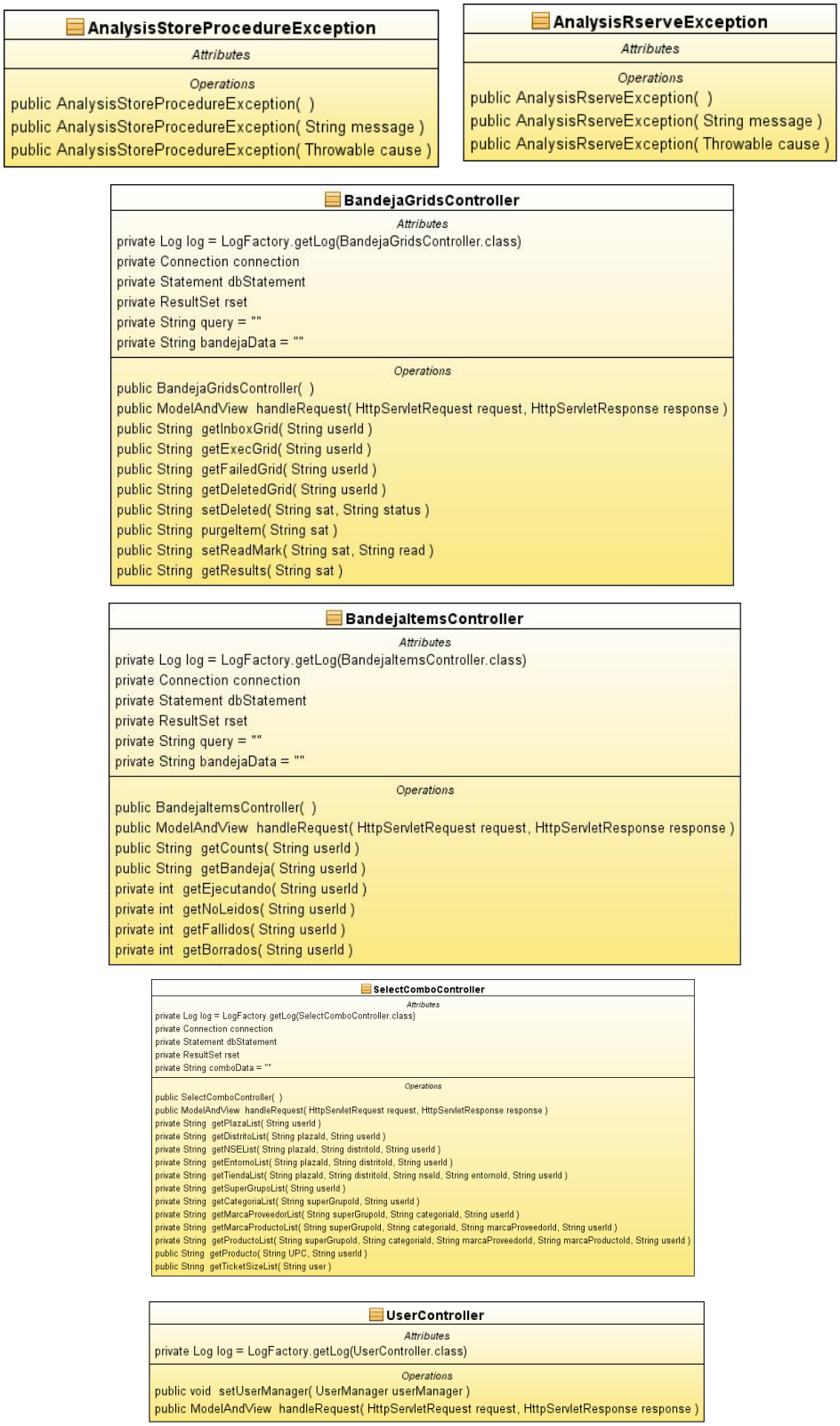
Paquetes de componente de aplicación

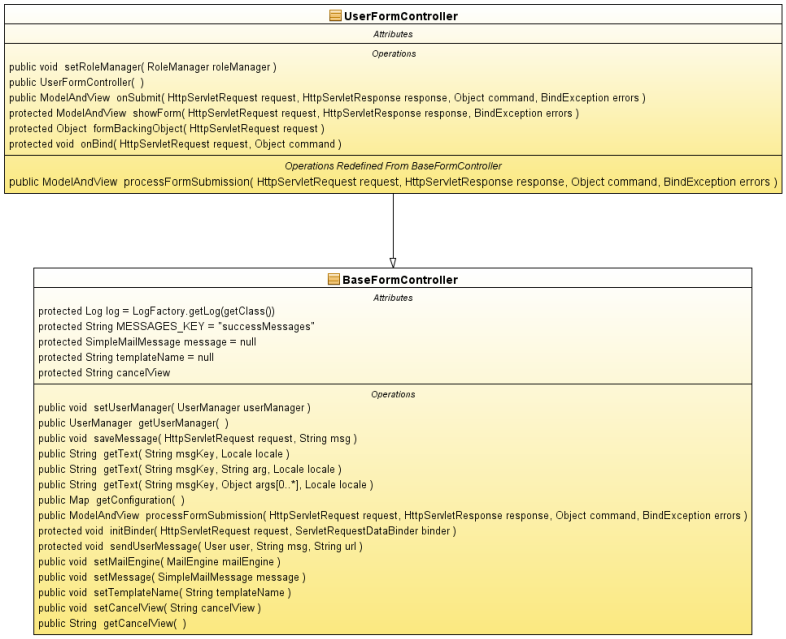
Nombre del Paquete
<default package>
com
com::dao
com:: dao::hibernate
com:: exception
com:: model
com:: service
com:: service::dwr
com:: service::impl
com:: util
com:: webapp
com:: webapp::action
com:: webapp::filter
com:: webapp::listener
com:: webapp::taglib
com:: webapp::util
Java
java::io
java::lang
java::lang::reflect
java::sql
java::text
java::util
javax
javax::mail
javax::servlet
javax::servlet::http
javax::servlet::jsp
javax::servlet::jsp::tagext
net
net::sf
net::sf::navigator
net::sf::navigator::menu
org
org::acegisecurity
org::acegisecurity::providers
org::acegisecurity::providers::dao
org::acegisecurity::userdetails
org::apache
org::apache::commons
org::apache::commons::beanutils
org::apache::commons::logging
org::apache::commons::validator
org::apache::velocity
org::apache::velocity::app
org::springframework
org::springframework::aop
org::springframework::context
org::springframework::core
org::springframework::core::io
org::springframework::mail
org::springframework::orm
org::springframework::orm::hibernate3
org::springframework::orm::hibernate3::support
org::springframework::validation
org::springframework::web
org::springframework::web::bind
org::springframework::web::context
org::springframework::web::filter
org::springframework::web::servlet
org::springframework::web::servlet::mvc
org::springframework::web::servlet::support

Las clases principales de la aplicación se muestran a continuación.






AnalysisStoreProcedureException
Attributes
Operations
public AnalysisStoreProcedureException( )
public AnalysisStoreProcedureException(String message)
public AnalysisStoreProcedureException(Throwable cause)
AnalysisRserveException
Attributes
Operations
public AnalysisRserveException( )
public AnalysisRserveException(String message)
public AnalysisRserveException(Throwable cause)
BandejaGridsController
Attributes
private Log log = LogFactory.getLog(BandejaGridsController.class)
private Connection connection
private Statement dbStatement
private ResultSet rset
private String query = ""
private String bandejaData = ""
Operations
public BandejaGridsController( )
public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)
public String getInboxGrid(String userId)
public String getExecGrid(String userId)
public String getFailedGrid(String userId)
public String getDeletedGrid(String userId)
public String setDeleted(String sat, String status)
public String purgeItem(String sat)
public String setReadMark(String sat, String read)
public String getResults(String sat)
BandejaItemsController
Attributes
private Log log = LogFactory.getLog(BandejaItemsController.class)
private Connection connection
private Statement dbStatement
private ResultSet rset
private String query = ""
private String bandejaData = ""
Operations
public BandejaItemsController( )
public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)
public String getCounts(String userId)
public String getBandeja(String userId)
private int getEjecutando(String userId)
private int getNoLeidos(String userId)
private int getFallidos(String userId)
private int getBorrados(String userId)
SelectComboController
Attributes
private Log log = LogFactory.getLog(SelectComboController.class)
private Connection connection
private Statement dbStatement
private ResultSet rset
private String comboData = ""
Operations
public SelectComboController( )
public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)
private String getPlazaList(String userId)
private String getDistritoList(String plazaId, String userId)
private String getNSEList(String plazaId, String distritoId, String userId)
private String getEntornoList(String plazaId, String distritoId, String userId)
private String getTiendaList(String plazaId, String distritoId, String nseId, String entornoId, String userId)
private String getSuperGrupoList(String userId)
private String getCategoriaList(String superGrupoId, String userId)
private String getMarcaProveedorList(String superGrupoId, String categoriaId, String userId)
private String getMarcaProductoList(String superGrupoId, String categoriaId, String marcaProveedorId, String userId)
private String getProductoList(String superGrupoId, String categoriaId, String marcaProveedorId, String marcaProductoId, String userId)
public String getProducto(String UPC, String userId)
public String getTicketSizeList(String user)
UserController
Attributes
private Log log = LogFactory.getLog(UserController.class)
Operations
public void setUserManager(UserManager userManager)
public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)



Contenedor de JSP y Servlets

ET2.0 utiliza Tomcat (Vivek Chopra, 2004) como su servidor web con soporte de servlets y JSPs (JavaServer Pages). Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a se presenta en combinación con el servidor web Apache por lo que Tomcat puede funcionar como servidor web por sí mismo. La versión usada por la aplicación es:

Tomcat 5.25:

- Implementado a partir de las especificaciones Servlet 2.4 y JSP 2.0
- Recolección de basura reducida
- Capa envolvente nativa para Windows y Unix para la integración de las plataformas
- Análisis rápido JSP

Estructura de directorios

La jerarquía de directorios de instalación de Tomcat incluye:

bin - arranque, cierre, y otros scripts y ejecutables
common - clases comunes que pueden utilizar Catalina y las aplicaciones web
conf - ficheros XML y los correspondientes DTD para la configuración de Tomcat

logs - logs de Catalina y de las aplicaciones
server - clases utilizadas solamente por Catalina
shared - clases compartidas por todas las aplicaciones web
webapps - directorio que contiene las aplicaciones web
work - almacenamiento temporal de ficheros y directorios

Es "webapps" el directorio donde se despliega la aplicación ET2.0.

Capa Back-end

Esta capa representa el repositorio de datos usado en la aplicación. El cual para e-Tickets es ORACLE RDBMS 10.2

Este sistema de base de datos es básicamente una herramienta cliente-servidor para la gestión de base de datos. Es un manejador de base de datos relacional que hace uso de los recursos del sistema informático en todas las arquitecturas de hardware, para garantizar su aprovechamiento al máximo en ambientes cargados de información.

Es el conjunto de datos que proporciona la capacidad de almacenar y acudir a estos de forma recurrente con un modelo definido como relacional. Además es una suite de productos que ofrece una gran variedad de herramientas; es el mayor y más usado Sistema Manejador de Base de Dato Relacional (RDBMS siglas en Inglés) en el mundo.

CAPÍTULO 4 - ARQUITECTURAS DE SOFTWARE

Arquitecturas de Software envuelve la integración de metodologías y modelos del desarrollo de software, los cuales se distinguen de manera particular de las metodologías de análisis y diseño. La estructura de soluciones complejas de software es parte de la descripción del problema, agregado a la complejidad del desarrollo de software. Arquitectura de Software es la parte central de los métodos y técnicas que nos ayudan a manejar estas complejidades del desarrollo de software.

Una arquitectura de Software presenta un conjunto de vistas del sistema de software a manera de componentes y conectores. Los componentes encapsulan un conjunto de funcionalidades. Mientras que los conectores realizan la interacción en tiempo de ejecución entre los componentes. El diseño del sistema alcanza ciertas características de calidad basada en su composición englobando componentes y conectores. La arquitectura de un sistema de software puede ser descrito en un documento llamado descripción de arquitectura. Así mismo, un diseño de arquitectura de software no sigue del todo las metodologías existentes de diseño de software. Estos se complementan mostrando puntos de vista adicionales del sistema.

Existe una gran variedad de páginas web y libros sobre arquitecturas de software que proponen diferentes definiciones del término de arquitectura de software. El Instituto de Ingenieros Eléctricos y Electrónica (IEEE) recientemente publicaron una serie de practicas recomendadas de Arquitecturas de Software: IEEE 1471 (IEEE, September 2000, Lankhorst, 2005). En donde se incluyen definiciones de sistema, arquitecto, arquitectura, vistas arquitecturales y puntos de vista arquitecturales. IEEE 1471 define los siguientes términos:

- El *sistema* es un conjunto de componentes que juntos logran una función o un conjunto de funciones especificas.
- La *arquitectura* es la organización fundamental de un sistema incorporado a sus componentes, sus relaciones el uno al otro, al ambiente, y a los principios que dirigen su diseño y evolución.
- La *descripción arquitectural* es el conjunto de los artefactos que documentan la arquitectura.
- La *vista arquitectural* es una representación de un sistema o de una parte particular de un sistema desde una perspectiva particular.

- El *punto de vista arquitectural* es una plantilla que describe cómo crear y utilizar una visión arquitectural. Un punto de vista incluye un nombre, los individuos involucrados en el sistema, las preocupaciones tratadas por el punto de vista, y el modelado y convenciones analíticas.

Otra manera de ver la arquitectura del software es pensar en algunas de las preguntas típicas que se pueden contestar por las vistas de la arquitectura:

- ¿Cuáles son los subsistemas o los componentes del software?
- ¿Cuáles son las responsabilidades de los componentes?
- ¿Cuáles son los interfaces proporcionadas por estos subsistemas o componentes?
- ¿Qué subsistemas o componentes son afectados por un cambio al software?
- ¿Cuál es el esfuerzo en probar el sistema si cambiamos este componente?
- ¿Cuál es la arquitectura del software?
- ¿Qué componentes están implicados en la instalación de este cambio?
- ¿Cuáles son las parte del sistema a ser se distribuidas físicamente?
- ¿Cómo un cambio afectará el funcionamiento del sistema?
- ¿Un cambio a esta interfaz afectan a qué equipos del desarrollo?
- ¿Cuánto esfuerzo está implicado en el desarrollo de esta funcionalidad?

Atributos de una arquitectura de software

Hay varios atributos y cualidades de interés de la arquitectura del software. Estas calidades son importantes porque afectan el diseño y el desarrollo de diversas partes del software. Algunas de estas cualidades incluyen:
La adaptabilidad cultural – soporte de idiomas múltiples y las diferencias culturales
Seguridad – Prevención al acceso no autorizado a los datos
Integridad de los datos – Evitar el manejo de datos no corruptos

Características de mantenimiento
Portabilidad – La facilidad con que el software puede ser portado a otras plataformas
Extensibilidad - capacidad de agregar nuevas funciones y de cambiar las existentes
Facilidad de entendimiento - facilidad con que el software puede ser entendido para poder realizar cambios
Validación- el software se puede probado con eficacia
Usabilidad - una medida de la eficacia de la interfaz de usuario

Aspectos operacionales del sistema:
Disponibilidad – porcentaje de tiempo que el sistema esta funcionando
Actualizaciones – La forma en que el sistema es capaz de actualizarse mientras se encuentra en ejecución
Confiabilidad – habilidad para ejecutar las funciones requeridas sobre un periodo de tiempo
Recuperación – Tiempo requerido para recuperarse de alguna falla ocurrida

Rendimiento
Tiempo de respuesta – es la pronta respuesta a condiciones normales y extremas del sistema
Escalabilidad– es la capacidad del sistema para incrementar/decrementar su funcionalidad
Capacidad de operar– es la capacidad del sistema para soportar grandes cantidades de cargas de datos/operaciones y mantener sus tiempos de respuesta

Todos estos atributos de arquitectura son típicamente las preocupaciones del equipo de arquitectos. Los miembros del equipo de arquitectura necesitan constantemente evaluar la arquitectura de software para determinar si esta satisface la meta con respecto a estas características. A primera vista, esta evaluación puede parecer sencilla, sin embargo no lo es. Comúnmente, el favorecer a uno de estos atributos puede resultar en un decremento no-lineal en un atributo diferente. Por ejemplo, un incremento en la facilidad de modificación del sistema puede reducir considerablemente su rendimiento. Un incremento en su facilidad de actualización puede afectar en su facilidad de ser probado.
Por otro lado, los requerimientos con respecto a estas propiedades no son fácilmente especificadas. Muchos de estos atributos puedes solamente ser medidos cualitativamente con la práctica. Los arquitectos deben constantemente prioritizando y manejar los cambios entre estos atributos para un proyecto en específico.

Ha llegado a ser una práctica estándar en la industria del software el combinar el análisis y diseño de software junto con la codificación. Este diseño facilita el entendimiento de la estructura del software a ser desarrollado. El crear una arquitectura simplemente reconoce la necesidad de enfocarse en una imagen más amplia del diseño del software y proporcionar una guía al equipo de diseño y desarrollo. En el nivel de arquitectura de software, estamos más interesados en los subsistemas, componentes e interfaces que en las clases y los métodos. Esto es, la arquitectura de software provee de un plan de diseño y un esquema del sistema. Esto

es una abstracción que ayuda a manejar la complejidad del sistema. El propósito de la arquitectura de software no solo de describir los aspectos importantes para otros sino la de exponerles que el arquitecto puede razonar acerca del diseño. Tener una arquitectura bien definida puede evitarnos muchos de los problemas que ocurren durante el proceso de desarrollo. El objetivo de una arquitectura bien definida obtendrá un sistema que será de fácil diseño, desarrollo y mantenimiento.

CAPÍTULO 5 - Model-View-Controller ("MVC")

Model-View-Controller ("MVC") es el modelo de arquitectura recomendado para aplicaciones interactivas. MVC, organiza una aplicación interactiva en tres módulos separados: uno para el modelo de la aplicación con su representación de datos y lógica de negocios, el segundo para vistas que proporcionan representación de datos y entrada de datos del usuario, y el tercero para un controlador para procesar las peticiones y controlar el flujo. La mayoría de las aplicaciones web usan alguna variación del patrón de diseño MVC.

El patrón de diseño MVC proporciona varios beneficios de diseño en la aplicaciones como por ejemplo, separa los problemas de diseño en persistencia de datos y comportamiento, presentación, y control, decrementando la duplicación de código, centralizando el control, y haciendo a la aplicación mucho más fácil de modificar y extender. MVC también ayuda a los desarrolladores con diferentes habilidades a centrarse en la implementación modular a través de interfaces bien definidas. Por ejemplo, una aplicación J2EE puede incluir desarrolladores de custom tags, vistas, lógica de la aplicación, funcionalidad de base de datos, y networking. Un diseño basado en MVC puede centralizar el control de tal aplicación facilitando la adición de seguridad, logging, y flujo de pantallas. Nuevas fuentes de datos son fácilmente agregadas a una aplicación basada en MVC mediante la creación de código que adapta la nueva fuente de datos a una nueva vista.

Este capítulo describe cómo usar MVC para organizar el diseño de una aplicación J2EE Web. Una aplicación J2EE provee respuestas a peticiones por medio de HTTP (Hypertext Transfer Protocol). En el nivel más alto, la capa Web realiza cuatro cosas básicas en un orden específico: interpreta requests del cliente, envía estos requests a la capa de lógica de negocios, selecciona la siguiente vista a ser desplegada, y genera y entrega la siguiente vista.

La capa de controlador recibe cada request de HTTP e invoca la operación solicitada en la lógica de negocio. Basado en los resultados de la operación y el estado actual del modelo, el controlador selecciona la siguiente vista a ser desplegada. Finalmente, el controlador genera la vista seleccionada y la transmite al cliente para su presentación.

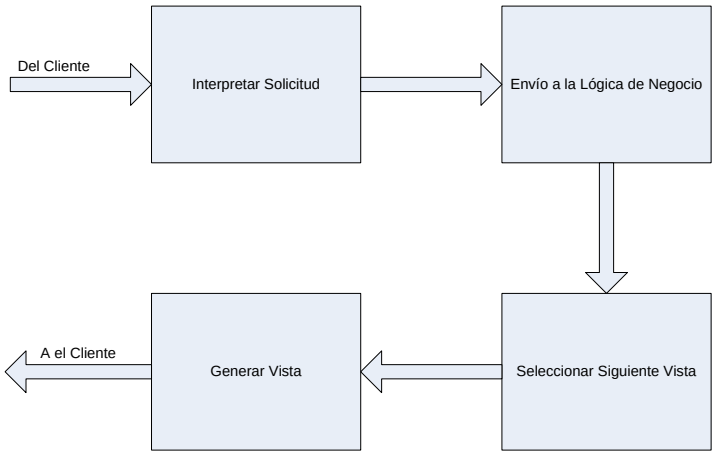


Figura 5.1. Funcionamiento de una aplicación enterprise Web

La Figura 5.1muestra una aplicación enterprise Web comúnmente que tiene las siguientes características:
Un diseño de aplicación debe tener una estrategia para dar servicio a tipos de clientes actuales y futuros.
Un controlador debe se mantenible y extensible. Sus tareas incluyen mapas de requests a las operaciones de la aplicación, seleccionando y ensamblando vistas, y manejando el flujo de pantalla. Una buena estructura puede minimizar la complejidad en el código.
El diseño de API del modelo de la aplicación y selección de tecnología tienen importantes implicaciones para la complejidad de la aplicación resultante, escalabilidad, y calidad del software.
Seleccionando una tecnología apropiada para la generación dinámica de contenido mejora la eficiencia del desarrollo y mantenimiento.
Las mejores prácticas del modelo son la de implementar una aplicación J2EE Web usando un framework de aplicación Web que se ajuste a la necesidades de la aplicación a desarrollar. Las siguientes secciones describen de manera general el diseño de una aplicación J2EE Web.

Estructura de una aplicación Web

La estructura general es la más importante en consideración en un diseño Web. Varios frameworks de aplicación Web implementan alguna forma del "Modelo 2" de arquitectura, donde el servlet maneja la comunicación con el cliente y la ejecución de la lógica de negocio, y presentación esta principalmente en la paginas JSP.
La literatura de aplicación Web en plataforma J2EE frecuentemente usa los términos "Modelo 1" y "Modelo 2" sin explicación alguna. Esta terminología viene de las

primeras versiones de la especificación de JSP, la cual es describen dos usos básicos de patrones de diseño para las paginas JSP. Mientras que estos términos han desaparecido del documento de especificación, se mantienen en uso. Modelo 1 y Modelo 2 simplemente se refieren a la ausencia o presencia (respectivamente) de un servlet controlador que atiende los requests de la capa de cliente y selecciona las vistas.
Una arquitectura de Modelo 1 consiste en un Web browser accesando directamente a JSPs de una aplicación Web. Las paginas JSPs accesan JavaBeans que representan el modelo de la aplicación asi como la siguiente vista a desplegar (pagina JSP, servlet, pagina HTML – Hypertext Markup Language) es determinada ya sea por hyperlinks seleccionados en el documento fuente o por el request por parámetros. Una capa de control de una aplicación basada en el Modelo 1 es descentralizado, debido a que la pagina actualmente desplegada determina la siguiente página a desplegarse. Además, cada página JSP o servlet procesa sus propias entradas de datos (i.e. parámetros de tipo GET o POST).
Una arquitectura de tipo Modelo 2 introduce un controlador servlet entre el browser y las paginas JSPs. El controlador centraliza la lógica para atender requests a la siguiente vista basada en el request del URL, parámetros de entrada, estado actual de la aplicación. El controlador también maneja la selección de la vista, la cual desacopla las paginas JSPs y servlets uno del otro. Aplicación de tipo Modelo 2 son más fáciles de mantener y extender, ya que las vistas no hacen referencia unas con otras directamente. El servlet controlador en el Modelo 2 proporciona un único punto de control para seguridad y logging, además de que frecuentemente encapsula los datos de entrada en una forma que puede usar el back-end del modelo MVC. Por estas razones, arquitecturas de tipo Modelo 2 son recomendadas por la mayoría de las aplicaciones interactivas.

En particular, para este trabajo estaremos enfocados en arquitecturas J2EE de tipo Modelo 2. Arquitecturas de tipo Modelo 2 usan servlets para procesar los requests y seleccionar las vistas. El patrón de diseño “Front Controller” centraliza el procesamiento de requests de la aplicación en un solo componente. Cada tipo de cliente Web envía peticiones y recibe respuestas de una única URL, lo cual simplifica el desarrollo de la capa de cliente. El “Front Controller” recibe peticiones desde el cliente y las direcciona a la capa de modelo de la aplicación. El “Front Controller” también es responsable de seleccionar y dar formato a la siguiente vista del cliente. El controlador en una aplicación es también un ejemplo del patrón de diseño “Mediator”, ya que este desacopla los componentes de vista de otros.
En la plataforma J2EE, un “Front Controller” es típicamente un servlet.

Diseño del controlador en una aplicación Web

Un controlador de una aplicación Web basada en MVC mapea las peticiones a las operaciones a la capa de modelo de la aplicación, y selecciona las vistas basadas en el modelo y el estado actual de la sesión. Debido a que aplicaciones enterprise crecen con el tiempo, cosiderar su extensibilidad es un requerimiento importante.

Cuando el controlador recibe una petición por HTTP, este necesita ser capaz de distinguir qué operación de la aplicación está siendo solicitada. Por ejemplo, ¿Cómo puede el cliente solicitar al server crear un nuevo usuario? Hay varios métodos para indicar al servido que operación realizar. Los métodos más comunes son,
Indicar la operación en un campo oculto en la forma, en este caso una operación POST es enviada al controlador, por ejemplo:

```
<FORM METHOD="POST"
ACTION="http://myServer/myApp/myServlet">
  <INPUT TYPE="HIDDEN" NAME="OP" VALUE="createUser"/>
  <!-- other form contents... -->
</FORM>
```

El bloque de código anterior indica la operación en una petición HTTP POST

Usar un mapeo de servlet para asociar todas las URLs con sufijo en particular a un servlet específico. Un mapeo de servlet es un descriptor que compara los paths de peticiones a un patrón y redirecciona estas peticiones al servlet correspondiente. Por ejemplo, imaginemos que un descriptor define el siguiente mapeo de servlet:

```
<servlet-mapping>
  <servlet-name>myServlet</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Imaginemos también que el path de servlet de contexto es,

```
http://myServer/myApp/servlets.
```

En este caso el contenedor de servlets redireccionaria la petición con URL

```
http://myServer/myApp/createUser.do
```

`to myServlet`, ya que la petición en URL concuerda al patrón `*.do`. El Servlet `myServlet` puede extraer el nombre de la operación solicitada a partir del URL.

Una vez que el controlador ha determinado que operación debe ejecutarse, este debe invocar el método correspondiente en la capa de modelo con los parámetros derivados de la petición. Un diseño de un controlador podría usar multiples secciones de `if-then-else`, como se muestra,

```
if (op.equals("createUser")) {
  model.createUser(request.getAttribute("user"),
          request.getAttribute("pass"));
} else if (op.equals("changeUserInfo") {
  // ... and so on...
}
```

Un mal diseño de un controlador

Este enfoque de largas secciones de `if-then-else` da como resultado un método de `service` muy extenso y difícil de manejar y extender. Un mejor acercamiento es el uso del patrón de diseño “Command”. Asi el controlador buscar podría buscar por una subclase de una clase llamada Action por nombre y delegar la petición a esta. El siguiente es un ejemplo,

```
        // Action.java:
public abstract class Action {
  protected Model model;
  public Action(Model model) { this.model = model; }
  public abstract String getName();
  public abstract Object perform(HttpServletRequest req);
};

// CreateUserAction.java:
public class CreateUserAction extends Action {
  public CreateUserAction(Model model) {
    super(model);
  }
  public String getName() { return "createUser"; }
  public Object perform(HttpServletRequest req) {
```

```
    return model.createUser(req.getAttribute("user"),
                req.getAttribute("pass"));
  }
}
```

Una clase abstracta Action y una subclases concreta

En el código anterior se define una clase abstracta `Action`, la cual tiene un nombre y un método `perform` que ejecuta un método del modelo correspondiente a su nombre. Por ejemplo, la subclase `CreateUserAction` tien el nombre "createUser". Su método `perform` invoca el método del modelo `createUser` usando parámetros extraidos de la petición de HTTP.

```
public class ControllerServlet extends HttpServlet {
  private HashMap actions;
  public void init() throws ServletException {
    actions = new HashMap();
    CreateUserAction cua = new CreateUserAction(model);
    actions.put(cua.getName(), cua);
    //... create and add more actions
  }
  public void doPost(HttpServletRequest req,
          HttpServletResponse resp)
    throws IOException, ServletException {

    // First identify operation "op" from URL.
    // method getOperation() is defined elsewhere.
    String op = getOperation(req.getRequestURL());
    // Aquí se busca y se ejecuta la acción correpondiente
    Action action = (Action)actions.get(op);
    Object result = null;
    try {
       result = action.perform(req);
    } catch (NullPointerException npx) {
       //... handle error condition: no such action
    }
    // ... Use result to determine next view
```

```
  }
//... other methods...
}
```

Usando un Map para identificar y ejecutar las Acciones

El código listado arriba muestra un controlador servlet que mantiene un hash map de objetos `Action`, cada uno indexado por su nombre. Cuando el servlet es cargado, el servlet contenedor llama el método `init`, el cual llena el hash map con objetos `Action` que invocan las operaciones del modelo. La llave del hash map es el nombre de la operación. Cada vez que el método de servicio del servlet recibe una petición, este identifica el nombre de la operación a ejecutarse, busca la `Action` correspondiente en el hash map, y ejecuta esta acción invoando el método `perform` de `Action`. `Action` regresa un objeto de resultado mismo que el servlet utiliza, junto con otros datos, para decidir que vista se la que se mostrara posteriormente. Cuando este controlador recibe una petición conteniendo el nombre `createUser`, este busca una instancia de `CreateUserAction` en el hash map. Entonces se invoca el método `perform` de `Action`, el cual utiliza el modelo para crear al usuario.

Controlando el flujo dinámico de pantalla

La sucesión de vistas que el usuario de la aplicación visualiza es llamada el flujo de pantallas. Un controlador de una aplicación Web controla el flujo de pantallas seleccionando la siguiente vista que el usuario visualiza. En sitios Web estáticos, las pantallas (usualmente pagina Web) son estáticamente ligadas una con otra. En contraste, un controlador dinámico selecciona la siguiente pantalla en respuesta a las acciones del usuario y a los resultados de las operaciones del modelo.

En esta sección, el término vista significa un recurso Web con una URL desde el cual contenido Web está disponible. Una vista puede ser una página JSP, un servlet, contenido estático, alguna combinación de los tres, ensamblados en una página. Típicamente, la siguiente vista a ser desplegada depende de:

La vista actualmente visualizada

Los resultados de cualquier operación en el modelo de la aplicación regresado por la invocación a métodos del modelo.

Posiblemente otro estado del lado del servidor, mantenido en `PageContext`, `ServletRequest`, `HttpSession`, and `ServletContext`.

Por ejemplo, la siguiente vista a desplegar después del login depende de:

La vista actual

El id y password del usuario contenido en el request
El resultado de la operación de autenticación del usuario, y
Posiblemente otros estados del lado del servidor. Como ejemplo, el estado puede incluir un número máximo de usuarios permitidos.
El controlador usa estos datos para determinar que vista es la siguiente en ser desplegada. Un controlador Web despliega una vista mediante el re-direccionamiento de la petición a una página JSP, servlet, u otro componente que envía la vista en un formato compatible al cliente; por ejemplo, regresa HTML al Web browser.

El controlador en un ejemplo de aplicación podría usar dos componentes para seleccionar y generar vistas: un manejador de flujo de pantallas, el cual selecciona la siguiente vista a visualizar; y el servicio de templates, el cual en realidad genera el contenido de la vista. El controlador usa el manejador de flujo de pantallas para seleccionar un vista, y re direccionar la petición al servicio de templates, el cual ensambla y entrega una vista al cliente. Ambos, el manejado de flujo de pantallas y el servlet de templates son componentes genéricos que son utilizados en cualquier aplicación Web. El diseño basado en componentes reduce el acoplamiento de componentes, promueve el reúso de código y simplifica el diseño del controlador.

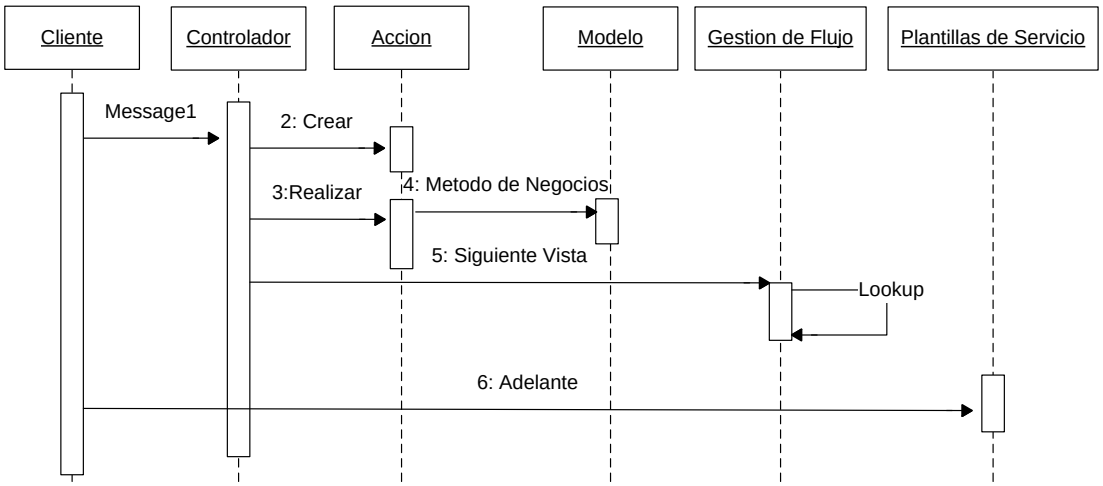


Figura 5.2. Controlador de la aplicación Web

En la Figura 5.2 se tiene un diagrama de interaccion de objetos que muestra al controlador de la aplicación interactuando con otras clases. El diagrama muestra la siguiente secuencia de llamadas:
El controlador recibe un POST desde el cliente.
El controlador crea una `Action` correspondiente a la operación solicitada.

El controlador llama el medodo `perform` de `Action`.
`perform` llama al método del modelo de negocio..
El controlador llama el manejador de flujo de pantallas para seleccionar la siguiente vista a desplegar.
El manejador de flujo de pantallas determina la siguiente vista y regresa su nombre al controlador.
El controlador redirecciona la petición al sevicio de templates, el cual ensambla y entrega la vista seleccionada al cliente.

CAPÍTULO 6 - ET2.0: AMBIENTE DE DESARROLLO

IDE Utilizado, Lenguaje utilizado, Plataforma Hardware, Plataforma Software

La herramienta desarrollada ET2.0 se implementó utilizando las tecnologías más actuales hasta el momento. Utilizando los patrones de diseño necesarios para su correcto funcionamiento, como por ejemplo inversión de control. Entre lo más destacado que cabe mencionar de esta herramienta son el uso de J2EE (Alur, 2003) como plataforma de programación, spring como controlador de eventos, tomcat como server y oracle como RDBMS.

Java Platform, Enterprise Edition o Java EE (anteriormente conocido como Java 2 Platform, Enterprise Edition o J2EE hasta la versión 1.4), es una plataforma de programación—parte de la Plataforma Java—para desarrollar y ejecutar software de aplicaciones en Lenguaje de programación Java con arquitectura de N niveles distribuida, basándose ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones. La plataforma Java EE está definida por una *especificación*. Similar a otras especificaciones del Java Community Process, Java EE es también considerada informalmente como un estándar debido a que los suministradores deben cumplir ciertos requisitos de conformidad para declarar que sus productos son *conformes a Java EE*; estandarizado por The Java Community Process / JCP.

Java EE incluye varias especificaciones de API, tales como JDBC (Java Database Controller, Reese, 2000), RMI, e-mail, JMS, Servicios Web, XML, etc y define cómo coordinarlos. Java EE también configura algunas especificaciones únicas para Java EE para componentes. Estas incluyen Enterprise JavaBeans, servlets, portlets (siguiendo la especificación de Portlets Java), JavaServer Pages y varias tecnologías de servicios web. Esto permite al desarrollador crear una Aplicación de Empresa portable entre plataformas y escalable, a la vez que integrable con tecnologías anteriores. Otros beneficios añadidos son, por ejemplo, que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados, significando que los desarrolladores pueden concentrarse más en la lógica de negocio de los componentes en lugar de en tareas de mantenimiento de bajo nivel.

Por si sola la estructura de Java ofrece una técnica de desarrollo rápido, de gran escala e independiente para aplicaciones web.

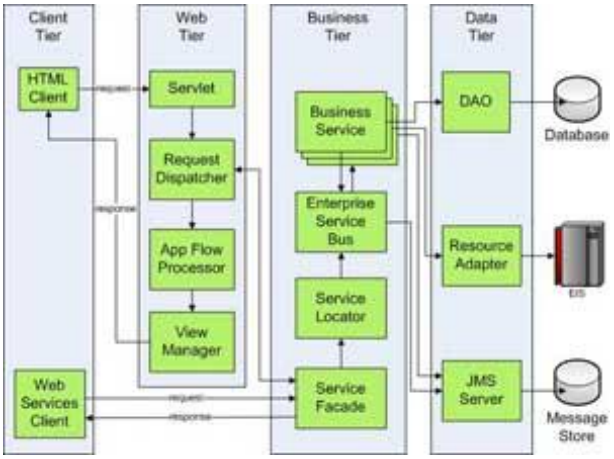


Figura 6.1. Arquitectura típica de aplicación J2EE

Tecnologías

Java Servlets

Los **Servlets** son completamente codificados en Java y son ejecutados del lado del servidor. Solamente la primera petición al server invoca el método init(), todos las peticiones llaman el método service*()* que a su vez regresa únicamente código HTML. Los Servlets permanecen en memoria y son ejecutados rápidamente.

Servlets proporcionan una funcionalidad amplia a Java, por ejemplo un Servlet es capaz de procesar formas en código HTML, usar sus propias clases de Java, enviar correos electrónicos, accesar a una base de datos, etc.

JavaServer Pages

JavaServer Pages (JSP)(Perry, 2003) es una tecnología relativamente nueva creada para integrar código del lado del servidor dentro de código estático HTML. JSPs tiene la ventaja de son ejecutados con Java y por tanto pueden reutilizare casi en cualquier sistema operativo. JSPs están relacionados con los Servlets de Java, en el sentido que los JSPs son traducidos en Java Servlets cuando son requeridos por primera vez. Con cada petición de un JSP, el Servlet generado ejecuta el método *service()*. Esto es, la principal diferencia entre Servlets y JSPs es que en un JSP se

inserta código dinámico de Java and JSP en un código estático de HTML, mientras que en los Servlets se inserta código estático HTML dentro del código de Java.

Apache o cualquier otro web server no son capaces de manejar Servlets o JSPs. Asi que para desarrollar ET2.0 se uso el servidor Tomcat. Tomcat es la referencia official de implementacion para Servlets y JSPs de SUN. Tomcat es un contenedor web basado en Java, es open source y completamente libre de usar.

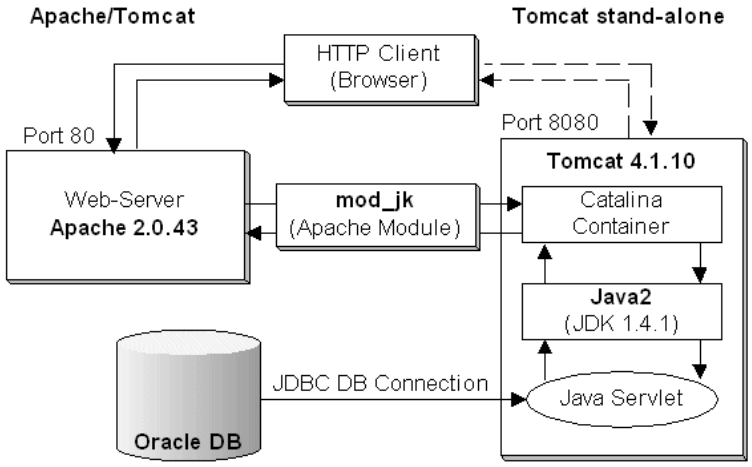


Figura 6.2. Configuración servidor Tomcat

Apache/Tomcat

El Servidor Apache HTTPD es un HTTP web server flexible que implementa los protocolos más recientes, incluyendo HTTP (RFC2616), es altamente configurable y extensible junto con otros modules externos. Es posible el escribir módulos utilizando las APIs de Apache pues se proporciona el código fuente. Este servido es capaz de correr en Winows NT/9x, Netware, OS/2, así como la mayoría de las versiones de Unix. Por otro lado Tomcat es un contenedor de servlets corriendo sobre Apache que es usado como la referencia de la implementación oficial de Servlets y JavaServer Pages.

Spring

El principal propósito de Spring (Craig Walls, 2007) es la de simplificar el desarrollo de J2EE. Uno de los principales beneficios que Spring ofrece es que elimina la necesidad de utilizar una variedad de archivos de propiedades manejando la

configuración de manera consistente a través de las aplicaciones y proyectos. El uso de **Inversión de Control** and **Inyección de Dependencia** ayuda a esta simplificación.

Spring puede visualizarse como una arquitectura en capas que consiste de siete módulos bien definidos. La función de cada componente se describe a continuación:

1. Spring Core:

El contenedor core proporciona la función fundamental de Spring. En este el componente primario es el `BeanFactory`, una implementación del patrón de diseño "Factory". El componente BeanFactory aplica el patrón de diseño IOC **(Inversion of Control)** para separar la configuración de la aplicación y la especificación de dependencia del código de la aplicación.

2. Spring Context/Application Context:

El Spring context es un archivo de configuración que provee información de contexto al framework Spring. El Spring context introduce servicios Enterprise tales como acceso J**NDI** (Java Naming and Directory Interface)**, integración EJB, e-mail, internacionalización, validacion, and funcionalidad de scheduling**.

3. Spring AOP: (Aspect-Oriented Programming)

El modulo de Spring AOP integra la funcionalidad de aspect-oriented directamente dentro del framework de Spring, esto a través de su característica de manejo de configuración. Como resultado podemos fácilmente habilitar AOP para cualquier objeto. Con Spring AOP podemos también incorporar manejo de transacciones declarativas en las aplicaciones.

4. Spring DAO:

La capa de abstracción de Spring JDBC y DAO ofrece de manera optimizada el manejo de conexiones a bases de datos, manejo de excepciones y el lanzamiento de errores para diferentes tipos de bases de datos.

5. Spring ORM:

El framework Spring puede ser integrado con diferentes frameworks ORM para proporciona manjeo de objetos relacionales, entre otros **JDO (Java Data Objects),**

Hibernate, OJB (Apache ObJectRelationalBridge) **and iBatis SQL** (Structured Query Language) **Maps.**

6. Spring Web:

El modulo de contexto Web contruye sobre el modulo de contexto de la aplicacion, proporcionando contexto para aplicaciones basadas en Web. Como resultado, el framework de Sping es capaz de integrarse con **Jakarta Struts, JSF and webworks.**

7. Spring Web MVC:

El framework MVC de Spring es altamente configurable basado en el patrón de diseño MVC (Model View Controller)

Inversión de Control (IoC)
Como se menciono Spring se basa en el tipo de Inversión de Control, la inyección de dependencia (Erl, 2009) – Esto es, no conectamos directamente nuestros componentes y servicios juntos en código sino que describimos los servicios que son requeridos por los componentes.

1. Dependency Lookup (service locator):

Es posible tomar una dependencia, encapsularla en una interface, registrarla en un diccionario y posteriormente buscarla con la ayuda de una clase llamada localizador (locator). Un ServiceLocator actúa como un punto central para obtener y almacenar referencias EJBHome, un ejemplo simple de esta funcionalidad es como sigue:

```
private OrderService getOrderService() {
  OrderServiceHome home = ServiceLocator.locate(OrderServiceHome);
  OrderService orderService = home.create();
}
```

2. Inyección de dependencia:

En estos objetos de la aplicación no son responsables de buscar los recursos de los cuales tienen una dependencia. En su lugar un contenedor de IoC configura el objeto externalizando la búsqueda del recurso desde el código de la aplicación hasta el contenedor. Esto es, las dependencias son inyectadas dentro de los objetos. Así, las búsquedas son completamente eliminadas de los objetos de la aplicación y esto puede ser usado desde fuera del contenedor. Veamos un ejemplo de código,

```
private OrderService orderService;

public void doRequest(HttpServletRequest request) {

  Order order = createOrder(request);

  orderService.createOrder(order);

  public void setOrderService(OrderService orderService) {

        this.orderService = orderService;

 }
```

En este caso no tenemos código para buscar el recurso. La referencia a OrderService es dada a nuestra clase por el contenedor de Spring a través del método setOrderService(). Con Spring, nunca tenemos problemas buscando dependencias.

Spring es identificado por la versión de Inversión de Control llamada Inyección de Dependencias. El concepto detrás de IoC es comúnmente expresado en el principio de Hollywood: “No llame, le llamaremos”. IoC mueve la responsabilidad de hacer que las cosas sucedan dentro del framework y lejos del código de la aplicación. Mientras que el código puede hacer llamadas tradicionales a funciones en librerías, un framework basado en IoC llama al código que escribimos.

Inyección de Dependencia es una forma de IoC que remueve explícitamente la dependencia al contenedor de APIs. Métodos ordinarios de Java usualmente inyectan dependencias tales como objetos de colaboración o valores de configuración dentro de las instancias de los objetos.

CAPÍTULO 7 – RESULTADOS Y TRABAJO FUTURO

Como en todo desarrollo de software, a lo largo de su implementación han ido surgiendo nuevos requerimiento y propuestas de mejora. Actualmente la aplicación ET2.0 cuenta con áreas de oportunidad que pueden mejorar. Dentro de las áreas de oportunidad se encuentra que la herramienta despliega un error cuando alguno de los análisis tarda más de una hora en procesar la información y generar un reporte, esto debido a la configuración interna del browser Internet Explorer. Por otra parte no se puede ejecutar más de un análisis a la vez, al procesar un análisis el segundo debe esperar a que el primero termine, por lo cual es una limitante en la herramienta. Además el usuario debe estar conectado siempre a la red para no perder la conexión con el sistema de ET2.0, dado que si se desconecta de la red el análisis se pierde.

Como resultado de estas restricciones se desarrolló una nueva versión de esta herramienta ET3.0, con la cual se solucionaron las áreas de oportunidad antes mencionadas. De esta manera se procesan diferentes análisis al mismo tiempo, los cuales no están limitados en tiempo para ser procesados, a la vez no es necesario que el usuario esté conectado a la red para generar los análisis.

De esta manera la aplicación ET3.0 tuvo como beneficio el brindar al usuario la posibilidad de ejecutar los análisis de una manera más rápida y confiable, sin tener el riesgo de perder los análisis ejecutados y así facilitar su trabajo. Deberá contar con las siguientes características:

- El usuario final debe poder generar varios análisis a la vez en una misma sesión, es decir que le permita ejecutar otros reportes sin importar que hayan finalizado reportes anteriores a esta ejecución.
- El usuario final al ejecutar un análisis, le permita almacenarlo dentro del mismo sistema de ET3.0 para posteriores visualizaciones del mismo análisis.
- El usuario podrá eliminar aquellos análisis que él haya generado en el momento que lo desee.
- El usuario final debe poder visualizar dentro del mismo sistema, el tiempo en que terminan de ejecutar los análisis sin importan que se estén corriendo varios simultáneamente
- El usuario debe poder identificar el análisis que se está ejecutando por su nombre y no por la letra de referencia, aunado a esto desplegar día de ejecución del análisis.
- Los análisis deben seguir en ejecución aún cuando el usuario se haya desconectado de la red.

En la Tabla 7.1 se muestra la evolución del sistema mostrando las diferentes versiones del sistema, denotando las características de cadea una.

Version	Caracteristicas
ET1.0	Soporte de multiples análisis con ejecución síncrona. Limitado por el tiempo de expiración de conexión a servidor.
ET2.0	Está provisto de un mecanismo de seguridad el cual valida por medio de nombre de usuario y clave de acceso el uso indebido de personas no autorizadas.
ET3.0	Procesa diferentes análisis al mismo tiempo, los cuales no estarán limitados en tiempo para ser procesados, a la vez no es necesario que el usuario esté conectado a la red para generar los análisis.
ET4.0 – Trabajo futuro	Una extensión de la herramienta tal que permite obtener de manera sistemática reportes de análisis generados a partir de fechas y/o períodos dados por los usuarios. Esto es, darle la capacidad a esta herramienta de calendarizar la ejecución de análisis.

Tabla 7.1. Versiones de ET

ET1.0, ET2.0 y ET3.0 son versiones actualmente en ejecución en ambientes de producción de algunas companias nacionales que en general son usadas extensivamente a diario por equipos especializados en economía y directores quienes de una manera sencilla y eficiente obtienen sus reportes para la toma de desiciones y planeación estrategia de su organización.

Adicional a la versión ET3.0, de antemano se han visualizado otras áreas de oportunidad para ser implementadas en un futuro. Entre otras, continuando con las versiones consideramos como ET4.0 una extensión de la herramienta tal que permite obtener de manera sistemática reportes de análisis generados a partir de fechas y/o

períodos dados por los usuarios. Esto es, darle la capacidad a esta herramienta de calendarizar la ejecución de análisis. La investigación de estas nuevas versiones queda encaminada con esta tesis.

Uno de los resultados a corto plazo que han venido sucediendo es la motivación de implementar esta arquitectura dentro de las instalaciones del INEGI. Esta es una organización donde se presenta la problemática que propone resolverse con esta tesis. En el INEGI el manejo de la información y su análisis es primordial, cuenta con distintos equipos que proporcionan soluciones a diversos problemas utilizando técnicas estadísticas. Muchas veces se ataca el mismo problema en múltiples ocasiones por el desconocimiento que ya antes se había trabajado en su solución. No se cuenta con un control estricto de los estudios realizados a la información, el número de problemas resueltos es cuantioso y la mayoría de las veces implica el análisis de grandes cantidades de datos.

Instituciones como el INEGI son el tipo de organizaciones que pueden ser beneficiadas con esta solución. Se espera que adicional a esta institución, otras más puedan tomar ventaja de esta propuesta.

En el contexto de CIMAT, como ambiente colaborativo, consolidando los resultados de las tesis de posgrado, proyectos de vinculación de la Gerencias que forman CIMAT. La extensión de esta tesis podría ayudar a tener un control de los trabajos realizados y más aun, una plataforma para desarrollo de nuevas aplicaciones visto y organizado como línea de producto. Las líneas de productos software permiten la reutilización sistemática en los casos en los que se tienen familias de productos, es decir productos similares, diferenciados por algunas características. Promoviendo de este modo la industrialización del desarrollo software. Una línea de productos de software es un conjunto de sistemas intensivos de software que comparten un conjunto de características comunes que satisfacen las necesidades específicas de un segmento de mercado particular y que son desarrollados a partir de un conjunto de activos núcleo comunes en un modo preestablecido. Este nuevo paradigma permite a las empresas mejorar la calidad del software que producen así como reducir los costes y tiempos de lanzamiento.

Por otro lado, este mismo sistema en algún momento podría ser manejado como Software como Servicio (Software as a Service, SaaS) que es un modelo de distribución de software en donde la compañía de IT provee el servicio de mantenimiento, operación diaria, y soporte del software usado por el cliente. En otras palabras es tener la información, el procesamiento, los insumos y los resultados de la lógica de negocio del software. Las ventajas que esta herramienta puede proporciona

son las de que el cliente no necesariamente debe tener un área especializada para soportar el sistema, por lo que baja sus costos y su riesgo de inversión.

CAPÍTULO 8 – CONCLUSIONES

No sólo implicó estudiar las tecnologías actuales de software para la implementación y estudio de arquitecturas de software. Adicional a esto, temas de calidad del software, manejo de proyectos con equipos multidisciplinarios, técnicas de estimación de software fueron necesarias. Sin embargo, la combinación de estas áreas no son tratadas dentro de esta tesis, pues el objetivo primordial ha sido dar una muestra de los beneficios de contar con sistemas de colaboración basados en una arquitectura de software con las características identificadas en el presente trabajo. Consideramos que es importante mencionarlo pues en gran medida los resultados satisfactorios de la implementación de esta herramienta son consecuencia de todos ellos en conjunto.

Por supuesto que estas implementaciones (ET2.0, ET3.0, ET4.0) de esta tesis, solo han sido con el propósito de mostrar la factibilidad de esta tesis. Quedan abiertos otros retos desde el punto de vista de arquitectura de software. Y por supuesto volver a preguntarnos, ¿Qué subsistemas o componentes son afectados por un cambio al software?, ¿Cuál es el esfuerzo en probar el sistema si cambiamos este componente?, Qué componentes están implicados en la instalación de este cambio?, ¿Cuáles son las parte del sistema a ser se distribuidas físicamente?, ¿Cómo un cambio afectará el funcionamiento del sistema?, ¿Un cambio a esta interfaz afectan a qué equipos del desarrollo? ¿Cuánto esfuerzo está implicado en el desarrollo de esta funcionalidad?

Desde una perspectiva muy personal, espero que esta tesis ayude a dar una visión de los beneficios de desarrollar esquemas de colaboración basados en arquitecturas de software con el fin de tomar ventaja de ir acumulando el conocimiento generado de manera centralizada en las organizaciones que así lo requieran.

APÉNDICE A - UNIFIED MODELING LANGUAGE (UML)[1]

Una breve historia del UML

Los lenguajes para el modelo orientado a objetos aparecieron entre la segunda mitad de 1970 y finales de 1980 como una metodología; se enfrentaron con una nueva generación de lenguajes de programación orientado a objetos y el incremento de aplicaciones complejas, iniciando así a experimentar con alternativas para el análisis y diseño. El número de métodos orientado objetos se incrementó de aproximadamente 10 a más de 50 durante el período entre 1989 y 1994. Muchos usuarios de estos métodos tuvieron problemas para encontrar un lenguaje para el modelado que cubriera sus necesidades completamente. Aprendiendo de la experiencia, nuevas generaciones de estos métodos iniciaron para aparecer, con algunos métodos claramente definidos, siendo los más notables, Booch, Jacobson's OESD , y Rumbaugh's OMT (Object Modeling Technique). Otros métodos importantes fueron, Fusion, Shlaer - Mellor, y Coad-Yourdon, cada uno de estos fue un método completo, sin embargo cada uno fue reconocido por tener fortalezas y debilidades. En simples términos, el método de Booch fue particularmente expresivo durante la fase de diseño y construcción de proyectos, OOSE proporcionó un excelente soporte del uso de casos como una manera de manejar la captura de requerimientos, análisis y diseños de alto nivel, y OMT2 fue más usado para el análisis y sistemas de información a gran escala. Una crítica masiva de ideas comenzó en la segunda mitad de 1990, cuando Grady Booch (Rational Software Corporation), Ivar Jacobson (Objective) y James Rumbaugh (General Electric) iniciaron a adoptar ideas de cada uno de sus métodos, lo cual colectivamente llegó a ser reconocido como el líder de los métodos orientados a objetos en todo el mundo. Fue así como los principales autores de los métodos, Booch, OOSE y OMT, fueron motivados a crear un lenguaje unificado para el modelado.

El esfuerzo de UML inició oficialmente en Octubre de 1994, cuando Rumbaugh se integró a la Compañía Rational de Booch. El proyecto inicial se centró en la unificación de la metodología de Booch y el método de OMT de Rumbaugh. El Borrador 0.8 de UML (como fue llamado) fue liberado en Octubre de 1995. Por el mismo tiempo, Jacobson se integra a Rational y el objetivo del proyecto de UML fue expandido a incorporar OOSE. Su esfuerzo resultó al liberar la versión 0.9 del documento de UML en Junio de 1996. A lo largo de 1996, invitaron y recibieron comentarios de la comunidad en general de ingenieros de software. Durante ese

[1] The Unified Modeling Language User Guide, Booch, Rumbaugh, Jacobson

tiempo, varias organizaciones de software vieron a UML como una estrategia para sus negocios. Aquellas organizaciones contribuyeron al documento de UML versión 1.0 integrada por Digital Equipment Corporation, Hewlett-Packard, I-Logix, Intellicorp, IBM, ICON Computing, MCI SystemHouse, Microsoft, Oracle, Rational, Texas Instruments y Unisys. El resultado de esta colaboración dió lugar al UML 1.0, un lenguaje para el modelado que fue bien definido, expresivo, poderoso y aplicable al amplio dominio de problemas.

Introducción a UML

UML es un lenguaje estándar para la escritura de esquemas para el desarrollo de software. UML puede ser usado para visualizar, especificar, construir y documentar los modelos de un sistema de software. UML es apropiado para el modelado de sistemas desde un simple sistema de información hasta una aplicación distribuida basada en el Web e incluso sistemas de tiempo real. Es un lenguaje ampliamente expresivo, y sin embargo no es difícil de entender y usar.

UML es un lenguaje

Un lenguaje proporciona un vocabulario y reglas para la combinación de palabras, en tal vocabulario, para el propósito de comunicación. Un lenguaje para el modelado es un lenguaje en el cual su vocabulario y reglas se enfocan en la representación física y conceptual de un sistema.

El vocabulario y reglas de un lenguaje tales como son en UML explican cómo crear y leer modelos bien definidos, pero no especifica qué modelos podríamos crear ni mucho menos cuóndo crearlos. Ese es el propósito del proceso de desarrollo de software.

UML es un lenguaje para la visualización

Para muchos programadores, la distancia entre pensar y la implementación para entonces escribir código es casi nulo. Piénsalo y escríbelo. De hecho, algunas cosas son mejores casi directamente en el código. El texto es una manera mínima y directa de escribir expresiones y algoritmos. Aún en esos casos, el programador continúa haciendo algún tipo de modelado, a pesar de ser completamente mental. Sin embargo hay varios problemas con esto. Primero, comunicar aquellos modelos conceptúales a otros es propenso a errores a menos que cada persona involucrada conozca el mismo lenguaje. Es común que en organizaciones de software, desarrollen su propio lenguaje, y es difícil entenderlo para alguien externo a ella o si es nuevo en el grupo. Segundo, hay algunas cosas acerca del sistema que no se pueden entender a

menos que se construyan modelos directamente en el lenguaje de programación. Por ejemplo, el significado de una jerarquía de clases puede ser inferida, pero no entendida, al mirar el código por todas las clases en la jerarquía. Tercero, si el desarrollador, quien implementó el código nunca escribió los modelos que estuvieron es su cabeza, esa información sería perdida para siempre, o en el mejor de los casos, sólo parcialmente recreable con la implementación de código, una vez que el desarrollador es movido a otro proyecto. Escribiendo modelos en UML soluciona la tercer consecuencia: Un modelo explícito facilita la comunicación.

Algunas cosas son mejor modeladas textualmente; mientras que otras son mejores gráficamente. Realmente, en todos los sistemas interesantes, hay estructuras que van más allá de lo que puede ser representado en un lenguaje de programación. UML es un lenguaje gráfico. Esto es, se centra en el segundo problema descrito anteriormente.

UML es más que un montón de símbolos gráficos. Además, detrás de cada símbolo hay una semántica bien definida. De esta manera, un desarrollador puede escribir un modelo en UML, y otro desarrollador, o incluso otra herramienta, puede interpretar el modelo sin ambigüedades. Eso soluciona el primer problema antes mencionado.

UML es un lenguaje para la especificación

En este contexto, *especificar* significa construir modelos que son precisos, sin ambigüedades, y completos. En particular, UML se centra en la especificación de todos los análisis importantes como el análisis, diseño, e implementación que debe ser hecha en el desarrollo y la liberación del sistema de software.

UML es un lenguaje para la construcción

UML no es un lenguaje visual de programación, pero sus modelos pueden ser directamente conectados a una gran variedad de lenguajes de programación. Esto significa que es posible transformar un modelo de UML a un lenguaje de programación como Java, C++, o Visual Basic, o incluso en tablas relacionales o el almacenamiento persistente de una base de datos orientada a objetos.

UML es un lenguaje para la documentación

Una organización de software bien organizada produce todo tipo de documentos, además del simple código ejecutable. Estos documentos incluyen (aunque no se limita a estos):

- Requerimientos
- Arquitectura
- Diseño
- Código fuente
- Planes de proyecto
- Pruebas
- Prototipos
- Versiones

Dependiendo de la cultura de desarrollo, algunos de estos documentos son tratados más o menos formalmente que otros. De manera que estos documentos no son sólo para la liberación del proyecto, sino también para el control, medida, y comunicación acerca del sistema durante su desarrollo y después de la liberación.

Notación de UML[2]

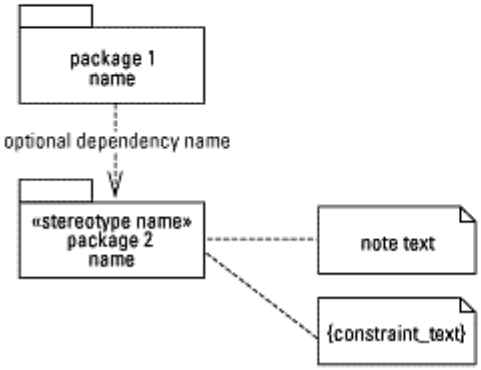
Como ya se dijo anteriormente, el UML es un lenguaje para la visualización, especificación, construcción, y documentación de sistemas de software. Como es un lenguaje, el UML tiene una sintaxis y semánticas bien definidas. La parte más notable de la sintaxis del UML es su notación gráfica.

[2] http://www.rational.com

GENERAL-PURPOSE CONCEPTS

Can be used on various diagram types

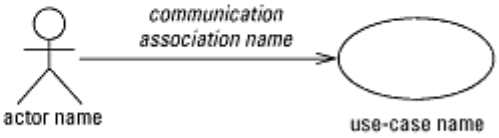
Package, dependency, note



USE-CASE DIAGRAM

Shows the system's use cases and which actors interact with them

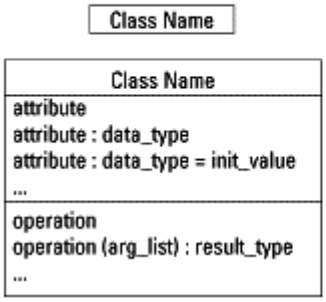
Actor, use case, and association



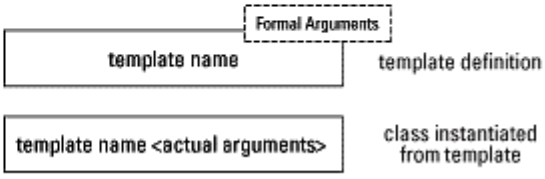
CLASS DIAGRAM

Shows the existence of classes and their relationships in the logical view of a system

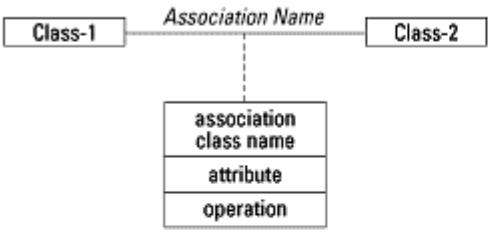
Class



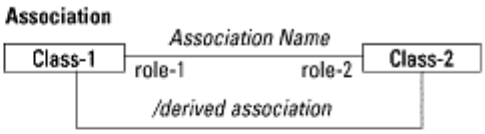
Parameterized class



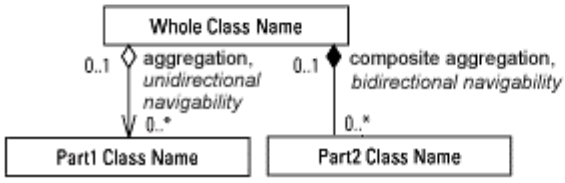
Association classes



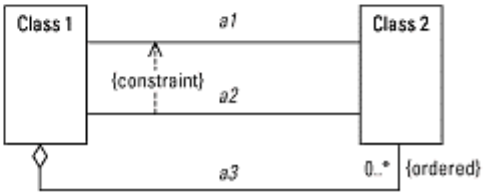
Role names and derived associations



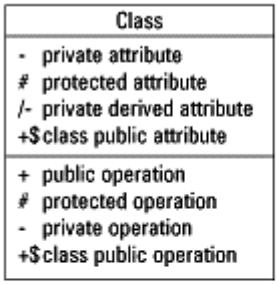
Aggregation, navigability, and multiplicity



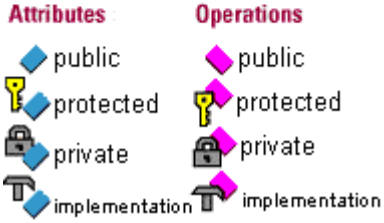
Constraints



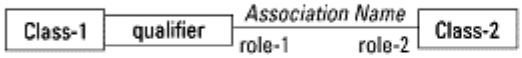
Visibility and properties



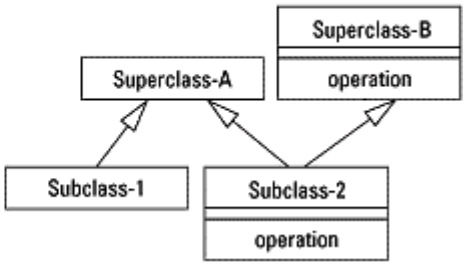
Optional visibility icons



Qualified association



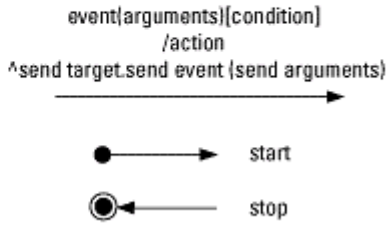
Generalization/specialization



STATE-TRANSITION DIAGRAM Shows the state space of a given context, the events that cause a transition from one state to another, and the actions that result

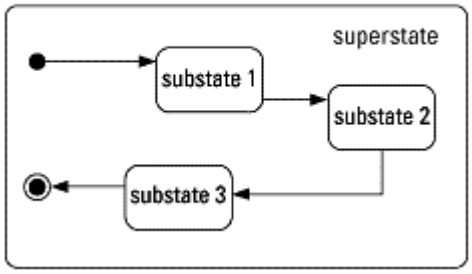
State icon

State transitions



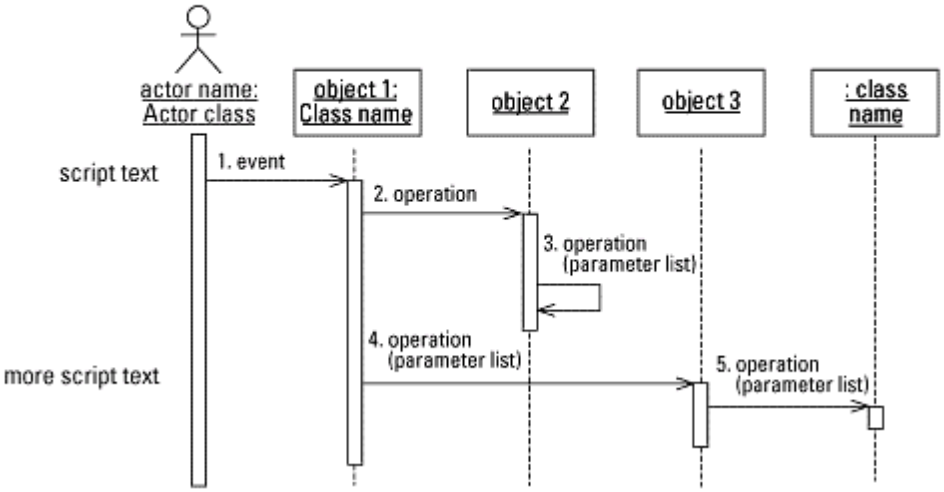
History Ⓗ

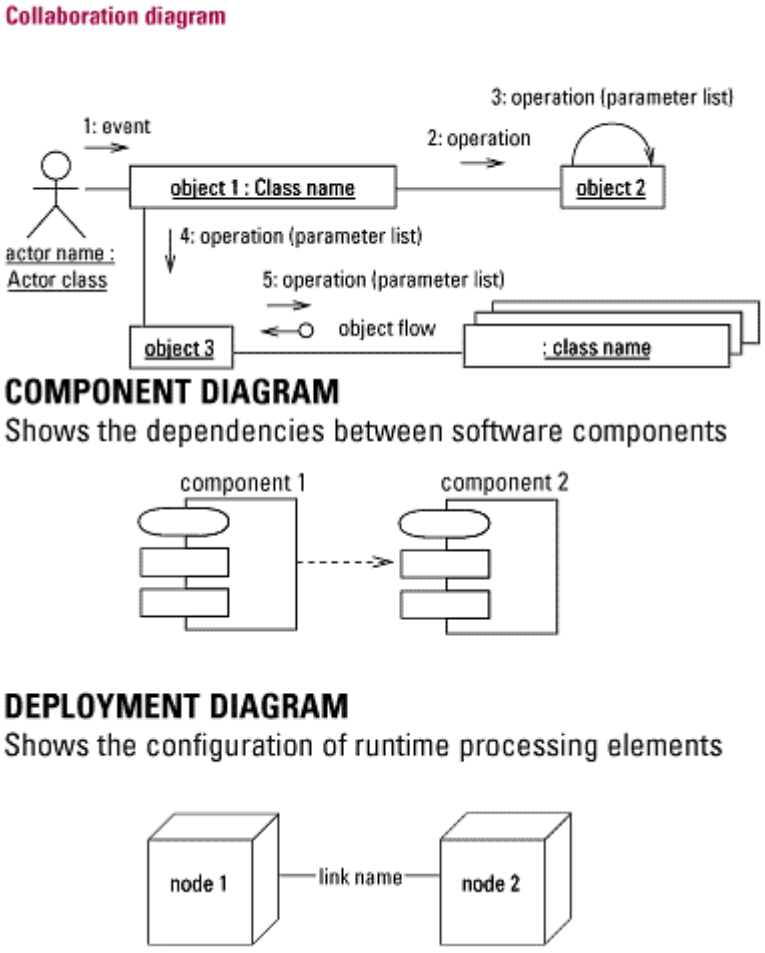
Nesting



INTERACTION DIAGRAMS Show objects in the system and how they interact

Sequence diagram





Diagramas

Un diagrama es una representación gráfica de un conjunto de elementos, la mayoría de las veces interpretados como un grafo de vértices (entidades) y arcos (relaciones). Un diagrama es una proyección dentro de un sistema. El UML incluye nueve de esos diagramas.

1.Diagrama de clases	Un diagrama estructural que muestra un conjunto de clases, interfaces, colaboraciones y sus relaciones.
2.Diagrama de objetos	Un diagrama estructural que muestra un conjunto de objetos y sus relaciones.

3.Diagrama de casos de uso	Un diagrama de comportamiento que muestra un conjunto de objetos y sus relaciones.
4.Diagrama de secuencia	Un diagrama de comportamiento que muestra una interacción, enfatizando el orden de los mensajes en el tiempo.
5.Diagrama de colaboración	Un diagrama de comportamiento que muestra un interacción, enfatizando en la organización estructural de los objetos que envían y reciben mensajes.
6.Diagrama de estados	Un diagrama de comportamiento que muestra una máquina de estados, enfatizando el comportamiento de un evento ordenado de un objeto.
7.Diagrama de actividad	Un diagrama de comportamiento que muestra una máquina de estados, enfatizando en el flujo de actividad en actividad.
8.Diagrama de componente	Un diagrama estructural que muestra un conjunto de componentes y sus relaciones.
9. Diagrama de distribución	Un diagrama estructural que muestra un conjunto de nodos y sus relaciones.

APÉNDICE B - PROCESO UNIFICADO RATIONAL[3] [4]

Un proceso es un conjunto de pasos parcialmente ordenados con el propósito de alcanzar una meta. En ingeniería de software, la meta es producir un software que eficientemente y predecidamente cumpla con las necesidades del negocio. El UML(Larman, 1997)es un proceso ampliamente independiente, lo cual significa que puede ser usado con un número de procesos de ingeniería de software. La meta del Proceso Unificado Rational es facilitar la producción de software de alta calidad que cumpla con las necesidades del usuario final dentro de los tiempos y recursos comprometidos.

Características del Proceso

El Proceso Unificado Rational es un proceso iterativo. Para simples sistemas, perfectamente sería capaz de definir secuencialmente el problema completo, la solución completa, construir el software, y entonces probar el producto final. Sin embargo, dada la complejidad y sofisticación demandada por los sistemas actuales, este acercamiento lineal para el desarrollo de sistemas es irrealista. Un acercamiento iterativo implica un incremento de entendimiento del problema a través de los refinamientos sucesivos y un crecimiento incremental de una solución efectiva sobre múltiples ciclos. Construir con el acercamiento iterativo es la flexibilidad de acomodar nuevos requerimientos o cambios en objetivos de negocios. Este también permite al proyecto identificar y resolver riesgos anticipadamente.

Las actividades del Proceso Unificado Rational enfatizan la creación y mantenimiento de modelos, más que documentos de papel. Modelos - especialmente aquellos especificados usando UML- proporcionan buenas representaciones semánticamente del sistema de software en desarrollo. Estos pueden ser vistos de diversas maneras, y la información puede ser instantáneamente capturada y controlada electrónicamente. El Proceso Unificado Rational se centra en los modelos más que en documentos de papel, esto para minimizar la sobrecarga asociada a la generación y mantenimiento de documentos y maximizar el contenido de la información relevante.

[3] *The Unified Modeling Language. Grady Booch, James Rumbaugh, Ivar Jacobson.*

[4] Rational es el nombre de la compañía quien desarrolló el proceso, por tanto a lo largo del texto se mantendrá como tal.

Desarrollo bajo el Proceso Unificado Rational es una arquitectura céntrica. El proceso se centra en las bases del desarrollo y en los fundamentos de una arquitectura de software. Teniendo una arquitectura robusta facilita el desarrollo en paralelo, minimiza el re-trabajo e incrementa la probabilidad de reusar componentes y un mantenimiento del sistema eventual. Este esquema arquitectural funciona como una base sólida lo contrario al desarrollo de planear y dirigir el software basado en componentes.

Las actividades de desarrollo bajo el Proceso Unificado Rational son el manejo de casos de uso. El Proceso Unificado Rational pone un fuerte énfasis en la construcción de sistemas basados en un completo entendimiento de cómo el sistema liberado será usado. La noción de casos de uso y escenarios son usados para alinear el flujo del proceso desde la captura de requerimientos hasta las pruebas.

El Proceso Unificado Rational soporta técnicas orientadas a objetos. Cada modelo es orientado a objetos. Los modelos del Proceso Unificado Rational son basados en los conceptos de objetos, clases y las relaciones entre ellos, usando el UML como su notación.

El Proceso Unificado Rational es un proceso configurable. Aún cuando ningún proceso es adaptable para todas las organizaciones de software, el Proceso Unificado Rational es adecuado perfectamente y puede ser ajustado al rango de proyectos desde los pequeños equipos de desarrollo de software hasta las grandes organizaciones de software.

El Proceso Unificado Rational fomenta el objetivo de continuar con la calidad de control y el manejo de riesgos. La evaluación de la calidad es construida en los procesos, en todas las actividades y envolviendo todos los participantes, usando la evaluación de objetivos y criterios. El manejo de riesgos es construido en los procesos, de tal manera que los riesgos o el éxito del proyecto son identificados y atacados anticipadamente en el proceso de desarrollo.

Fases e Iteraciones

Una fase es el lapso de tiempo entre dos grandes recorridos del proceso en los cuales un conjunto bien definido de objetivos son alcanzados, documentos son alcanzados y decisiones son hechas para moverse a otra fase o

no. Como la figura C.1 ilustra, el Proceso Unificado Rational consiste en las siguientes cuatro fases.

1. Iniciación	Establece el caso de negocios del proyecto
2. Elaboración	Establece un plan de proyecto y la arquitectura
3. Construcción	Implementa el sistema
4. Transición	Provee el sistema al usuario final

Las etapas de iniciación y elaboración abarcan las actividades de ingeniería del ciclo de vida del desarrollo. Construcción y transición constituye su producción.

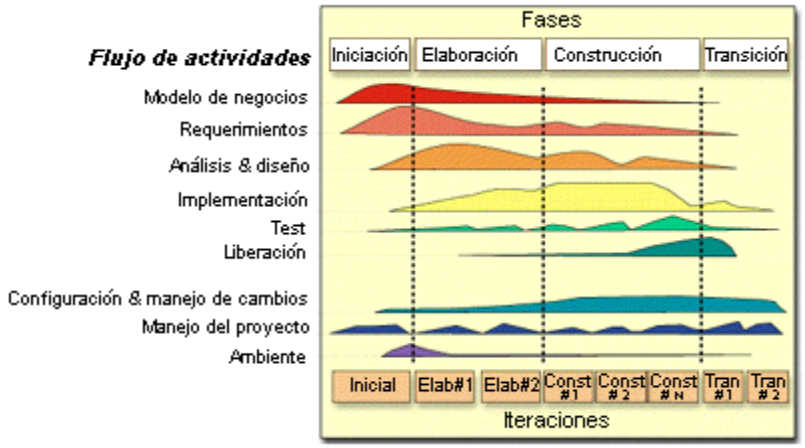


Figura C.1 *Ciclo de vida del desarrollo de software*

Dentro de cada fase hay un número de iteraciones. Una iteración representa un ciclo completo de desarrollo, desde la captura de requerimientos en el análisis hasta la implementación y pruebas, que resultan en la liberación del proyecto ejecutable.

Fases

Iniciación: Durante la fase de iniciación, se establece el caso de negocio para el sistema y se delimitan los alcances del sistema. El caso de negocios incluye los criterios de éxito, medición de riesgos, estimados de los recursos necesarios, y un plan de la fase mostrando una agenda de las principales actividades. Durante la iniciación, es común crear un prototipo ejecutable que sirve como una muestra del concepto.

Al final de la fase de iniciación, se examina el ciclo de vida de los objetivos del proyecto y se decide si verdaderamente es factible proceder con un desarrollo completo.
Elaboración: Las metas de la fase de elaboración son analizar el dominio del problema, establecer la arquitectura, desarrollar el plan del proyecto, y eliminar los grandes riesgos del proyecto. Las decisiones de arquitectura deben ser hechas con un entendimiento del sistema completo. Esto implica describir la mayoría de los requerimientos del sistema. Para verificar la arquitectura, se implementa un sistema que demuestra la selección de la arquitectura y ejecuta muchos de los casos de uso. Al final de la fase de elaboración, se examina detalladamente los objetivos y alcance del sistema, la selección de la arquitectura, la resolución de los principales riesgos, y se decide si se procederá con la construcción.

Construcción: Durante la fase de construcción, iterativamente e incrementalmente se desarrolla un producto completo que esta listo para ser usado por la comunidad de usuarios. Esto implica la descripción de los requerimientos restantes y los criterios de aceptación, eliminando del diseño, y completando la implementaron y pruebas del software. Al final de la fase de construcción, se decide si el software, y todos los usuarios están listos para su operación.

Transición: Durante la fase de transición, se distribuye el software a la comunidad de usuarios finales. Una vez que el sistema ha sido instalado al usuario final, frecuentemente detalles surgen que son requeridos adicionalmente con el propósito de ajustar al sistema, corregir algunos problemas no detectado, o terminar algunas funciones que pudieran haber sido pospuestas. Esta fase típicamente inicia con una versión beta del sistema, lo cual es reemplazado con el sistema de producción. Al final de esta fase, se decide si el ciclo de vida de los objetivos del proyecto han sido alcanzados y se determina si se debería iniciar otro ciclo desarrollo. Este también es un punto en el cual se discuten las experiencias aprendidas en el proyecto con el propósito de mejorar el proceso de desarrollo, lo cual será aplicado para un próximo proyecto.

Iteraciones

Cada fase del Proceso Unificado Rational puede se dividido en iteraciones. Una iteración es una repetición completa del desarrollo resultando una versión (interna o externa) de un ejecutable del producto constituyendo un

sub-conjunto del producto final en desarrollo, el cual implica que crezca incrementalmente de iteración en iteración hasta llegar a ser el sistema final. Cada iteración va a través de varios flujos de trabajos, sin embargo con el énfasis en cada flujo de trabajo, depende de cada fase. Durante la iniciación, la atención se centra en la captura de requerimientos. Durante la elaboración, la atención se centra hacia el análisis y diseño. En la construcción, la implementación es la actividad central, y la fase de transición se centra en la distribución del producto final.

Ciclos de desarrollo

Ir a través de las cuatro fases es llamado ciclo de desarrollo, y su resultado es la generación de un software. El primer paso a través de las cuatro fases es llamado ciclo de desarrollo inicial. A menos que la vida del producto termine, la existencia del producto se desarrolla gradualmente en sus siguientes generaciones por la misma secuencia de fases de iniciación, elaboración, construcción, y transición. Esta es la evolución del sistema, así de esa manera los ciclos de desarrollo después del ciclo inicial son sus ciclos de evolución.

Procesos

El Proceso Unificado Rational consiste en nueve procesos:

1. Modelado de Negocios	Describe la estructura y dinámica de la organización
2. Requerimientos	Describe el método basado en casos de uso para extraer los requerimientos
3. Análisis y diseño	Describe las múltiples vistas de arquitectura
4. Implementación	Toma en cuenta el desarrollo de software, pruebas unitarias y la integración.
5. Pruebas	Describe los casos de prueba, procedimientos y medidas para el seguimiento de problemas
6. Liberación	Cubre la configuración del sistema donde el software será ejecutado
7. Dirección de configuración	Controla los cambios y mantiene la integridad de los artefactos del proyecto
8. Dirección del proyecto	Describe varias estrategias para el trabajo con un proceso iterativo
9. Ambiente	Cubre la infraestructura necesaria para desarrollar un sistema

Capturados dentro de cada proceso hay un conjunto de artefactos y actividades correlacionadas. Un artefacto es algún documento, reporte, o un ejecutable que es producido, manipulado o consumido. Una actividad describe las tareas --el pensamiento de pasos, la ejecución de pasos, y la revisión de pasos-- ejecutados por trabajadores para crear o modificar artefactos, junto con las técnicas y lineamientos para ejecutar la tarea, posiblemente incluyendo el uso de herramientas para ayudar a automatizar algunas de las tareas.

Importantes conexiones entre los artefactos son asociadas para ser asociados con ciertos de esos procesos. Por ejemplo, el modelo de uso de casos generado durante la captura de requerimientos es realizado por el modelo de diseño desde el proceso de análisis y diseño, implementado por el modelo de implementación desde el proceso de implementación, y verificado por el modelo de pruebas desde el proceso de pruebas.

Artefactos

Cada actividad del Proceso Unificado Rational tiene artefactos asociados, también requeridos como una entrada o generados como una salida. Algunos artefactos son usados para direccionar la salida a actividades subsecuentes, mantener como referencia de recursos en el proyecto, etc.

Modelos

Los modelos son el más importante tipo de artefacto en el Proceso Unificado Rational. Un modelo es una simplificación de la realidad, creado para un mejor entendimiento del sistema que esta siendo creado. En el Proceso Unificado Rational, hay nueve modelos que colectivamente cubren todas las decisiones importantes que van desde la visualización, especificación, construcción, y documentación en un sistema intensivo de software.

1.Modelo de negocios	Establece una abstracción de una organización
2.Modelo de dominio	Establece el contexto del sistema
3.Modelo de casos de uso	Establece los requerimientos funcionales del sistema
4.Modelo de análisis (opcional)	Establece la idea del diseño
5.Modelo de diseño	Establece el vocabulario del problema y su solución

6.Modelo de proceso (opcional)	Establece la concurrencia del sistema y los mecanismos de sincronización
7.Modelo de distribución	Establece la topología del hardware el cual el sistema será ejecutado
8.Modelo de implementación	Establece las partes usadas para construir y liberar el sistema.
9.Modelo de pruebas	Establece las partes por las cuales el sistema es validado y verificado

Una vista es una proyección dentro del modelo. En el Proceso Unificado Rational, la arquitectura de un sistema es capturada en cinco vistas; la vista de diseño, de proceso, distribución, implementación, y casos de uso.

Otros artefactos

Los artefactos del Proceso Unificado Rational son categorizados al igual que los artefactos de dirección o artefactos técnicos. Los artefactos técnicos del Proceso Unificado Rational pueden ser divididos en cuatro principales conjuntos:

1.Conjunto de requerimientos	Describe lo que el sistema debe hacer
2.Conjunto de diseño	Describe como el sistema será construido
3.Conjunto de implementación	Describe el ensamblado de los componentes del software desarrollado
4.Conjunto de distribución	Provee todos los datos para la configuración distribuible

Conjunto de requerimientos

Este conjunto agrupa toda la información para describir lo que el sistema debe hacer. Este puede contener un modelo de caso de uso, un modelo de requerimientos no funcionales, un modelo de dominio, un modelo de análisis, y cualquier otra forma de expresión para cubrir las necesidades del usuario, prototipos de interfase, constraints, etc.

Conjunto de diseño

Este conjunto agrupa la información para describir el cómo el sistema será construido y captura de decisiones acerca de cómo el sistema será construido, tomando en cuenta todas las limitaciones de tiempo, presupuesto, reuso, objetivos de calidad, etc. Este puede contener un modelo de diseño, un

modelo de pruebas, y otras formas de expresión de la naturaleza del sistema, prototipos, etc.

Conjunto de Implementación

Este conjunto agrupa toda la información acerca de los elementos de software que comprenden al sistema, incluyendo código fuente en varios lenguajes de programación, configuración de archivos, archivos de datos, componentes de software, junto con la información necesaria para ensamblar al sistema.

Conjunto de distribución

Este conjunto agrupa toda la información acerca de la manera en que realmente el software es empacado, embarcado, instalado, y ejecutado en el ambiente destino.

GLOSARIO

AJAX

Acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

Ambiente distribuido

Una colección de computadores separados físicamente y conectados entre sí por una red de comunicaciones distribuida; cada máquina posee sus componentes de hardware y software que el usuario percibe como un solo sistema (no necesita saber qué cosas están en qué máquinas). El usuario accede a los recursos remotos (RPC) de la misma manera en que accede a recursos locales, o un grupo de computadores que usan un software para conseguir un objetivo en común.

Base de datos relacional

es una base de datos en donde todos los datos visibles al usuario están organizados estrictamente como tablas de valores, y en donde todas las operaciones de la base de datos operan sobre estas tablas.

Callback

Es una porción de código ejecutable que es pasado como un argumento a otro código. Esto permite construir código de bajo nivel como capas de abstracción que pueden ser llamadas desde una subrutina (o función) definida en una capa de mayor nivel.

Capa Back-end

El front-end es la parte del software que interactúa con el o los usuarios y el back-end es la parte que procesa la entrada desde el front-end. La separación del sistema en "front ends" y "back ends" es un tipo de abstracción que ayuda a mantener las diferentes partes del sistema separadas. La idea general es que el front-end sea el responsable de recolectar los datos de entrada del usuario, que pueden ser de muchas y variadas formas, y procesarlas de una manera

conforme a la especificación que el back-end pueda usar. La conexión del front-end y el back-end es un tipo de interfaz.

Clase abstracta

Una de las características más útiles de cualquier lenguaje orientado a objetos es la posibilidad de declarar clases que definen como se utiliza solamente, sin tener que implementar métodos, son las clases abstractas. Mediante una clase abstracta se intenta fijar un conjunto mínimo de métodos (el comportamiento) y de atributos, que permitan modelar un cierto concepto, que será refinado y especializado mediante el mecanismo de la herencia. Como consecuencia, la implementación de la mayoría de los métodos de una clase abstracta podría no tener significado. Para resolver esto, Java proporciona los métodos abstractos

Contenedor de JSP

El denominado contenedor JSP (que sería un componente del servidor web) es el encargado de tomar la página, sustituir el código Java que contiene por el resultado de su ejecución, y enviarla al cliente

DOJO

Es un framework que contiene APIs y widgets (controles) para facilitar el desarrollo de aplicaciones Web que utilicen tecnología AJAX. Contiene un sistema de empaquetado inteligente, los efectos de UI, drag and drop APIs, widget APIs, abstracción de eventos, almacenamiento de APIs en el cliente, e interacción de APIs con AJAX.

DWR

Es una API (Application Programming Interface) open source que, utilizando la tecnología AJAX (Asynchronous JavaScript and XML), permite realizar llamadas remotas a objetos Java del servidor desde código JavaScript cliente.

Enterprise JavaBeans

GET

Se utiliza para recuperar información identificada por un URI por parte de los navegadores. Si el URI se refiere a un proceso generador de datos como un programa CGI, en lugar de él, se devuelven los datos generados por el programa. El método GET también se puede utilizar para pasar una pequeña cantidad de información al servidor en forma de pares atributo-valor añadidos al final del URI detrás de un símbolo de interrogación, ?.

HTML

HTML, siglas de HyperText Markup Language (Lenguaje de Marcas de Hipertexto), es el lenguaje de marcado predominante para la construcción de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes. HTML se escribe en forma de "etiquetas", rodeadas por corchetes angulares (<,>).

J2EE

Es una plataforma de programación—parte de la Plataforma Java—para desarrollar y ejecutar software de aplicaciones en Lenguaje de programación Java con arquitectura de N niveles distribuida, basándose ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones. La plataforma Java EE está definida por una especificación.

JavaScript

Es un lenguaje de programación interpretado, es decir, que no requiere compilación, utilizado principalmente en páginas web, con una sintaxis semejante a la del lenguaje Java y el lenguaje C. Al igual que Java, JavaScript es un lenguaje orientado a objetos propiamente dicho, ya que dispone de Herencia, si bien ésta se realiza siguiendo el paradigma de programación basada en prototipos, ya que las nuevas clases se generan clonando las clases base (prototipos) y extendiendo su funcionalidad.

JDBC

Es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice.

JMS

Es la solución creada por Sun Microsystems para el uso de colas de mensajes. Este es un estándar de mensajería que permite a los componentes de aplicaciones basados en la plataforma Java2 crear, enviar, recibir y leer mensajes. También hace posible la comunicación confiable de manera síncrona y asíncrona.

MVC (Model View Control)

es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista.

ORACLE

Es un sistema de gestión de base de datos relacional (o RDBMS por el acrónimo en inglés de Relational Data Base Management System), desarrollado por Oracle Corporation.

Patrón de diseño

Es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

POJOs

Acrónimo de Plain Old Java Object es una sigla creada por Martin Fowler, Rebecca Parsons y Josh MacKenzie en septiembre de 2000 y utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un framework en especial.

POST

El método POST se refiere normalmente a la invocación de procesos que generan datos que serán devueltos como respuesta a la petición. Además se utiliza para aportar datos de entrada a esos programas. En este caso los pares atributo-valor son incluidos en el cuerpo de la petición separados por ampersand.

R

Es un lenguaje y entorno de programación para análisis estadístico y gráfico. Se trata de un proyecto de software libre, resultado de la implementación GNU del premiado lenguaje S. R y S-Plus -versión comercial de S- son, probablemente, los dos lenguajes más utilizados en investigación por la comunidad estadística, siendo además muy populares en el campo de la investigación biomédica, la bioinformática y las matemáticas financieras. A

esto contribuye la posibilidad de cargar diferentes librerías o paquetes con finalidades específicas de cálculo o gráfico.

RDBMS
Es un Sistema Administrador de Bases de Datos Relacionales. RDBMS viene del acrónimo en inglés Relational Data Base Management System.

RMI
Java Remote Method Invocation) es un mecanismo ofrecido por Java para invocar un método de manera remota. Forma parte del entorno estándar de ejecución de Java y provee de un mecanismo simple para la comunicación de servidores en aplicaciones distribuidas basadas exclusivamente en Java.

Servlet
Son objetos que corren dentro del contexto de un contenedor de servlets (ej: Tomcat) y extienden su funcionalidad. También podrían correr dentro de un servidor de aplicaciones (ej: OC4J Oracle), que, además de contenedor para servlet, tendrá contenedor para objetos más avanzados, como son los EJB (Tomcat sólo es un contenedor de servlets).

SHA1
La familia SHA (Secure Hash Algorithm, Algoritmo de Hash Seguro) es un sistema de funciones hash criptográficas relacionadas de la Agencia de Seguridad Nacional de los Estados Unidos y publicadas por el National Institute of Standards and Technology (NIST). El primer miembro de la familia fue publicado en 1993 es oficialmente llamado SHA. Sin embargo, hoy día, no oficialmente se le llama SHA-0 para evitar confusiones con sus sucesores. Dos años más tarde el primer sucesor de SHA fue publicado con el nombre de SHA-1. Existen cuatro variantes más que se han publicado desde entonces cuyas diferencias se basan en un diseño algo modificado y rangos de salida incrementados: SHA-224, SHA-256, SHA-384, y SHA-512 (llamándose SHA-2 a todos ellos).

Software as a Service, SaaS
Software como Servicio (del inglés: Software as a Service, SaaS) es un modelo de distribución de software en donde la compañía de IT provee el servicio de mantenimiento, operación diaria, y soporte del software usado por el cliente. En otras palabras es tener la información, el procesamiento, los insumos y los

resultados de la lógica de negocio del software. En palabras simples: El cliente tiene el sistema hospedado en la compañía de IT.

SPRING

El Spring Framework (también conocido simplemente como Spring) es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java. Subclase concreta. Por su diseño el framework ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar para las prácticas comunes en la industria.

UML (Unified Modeling Language)

Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

URL

Significa Uniform Resource Locator, es decir, localizador uniforme de recurso. Es una secuencia de caracteres, de acuerdo a un formato estándar, que se usa para nombrar recursos, como documentos e imágenes en Internet, por su localización. Los URL fueron una innovación fundamental en la historia de la Internet. Fueron usadas por primera vez por Tim Berners-Lee en 1991, para permitir a los autores de documentos establecer hiperenlaces en la World Wide Web (WWW o Web). Desde 1994, en los estándares de la Internet, el concepto de URL ha sido incorporado dentro del más general de URI (Uniform Resource Identifier - Identificador Uniforme de Recurso), pero el término URL aún se utiliza ampliamente.

XML

Siglas en inglés de Extensible Markup Language (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto

XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.

BIBLIOGRAFIA

1. **Alur Deepak** Core J2EE Patterns: Best Practices and Design Strategies (2nd Edition) (Sun Core Series) [Book]. - [s.l.] : Prentice Hall PTR, 2003.
2. **Bergström Stefan** Adopting the Rational Unified Process [Book]. - [s.l.] : Addison-Wesley Professional , 2004.
3. **Craig Walls Ryan Breidenbach** Spring in Action [Book]. - [s.l.] : Manning Publications, 2007.
4. **Erich Gamma Richard Helm, Ralph Johnson** Design Patterns: Elements of Reusable Object-Oriented Software [Book]. - [s.l.] : Addison-Wesley Professional, 1994.
5. **Erl Thomas** SOA Design Patterns [Book]. - [s.l.] : Prentice Hall PTR, 2009.
6. **Gorton Ian** Essential Software Architecture [Book]. - [s.l.] : Springer, 2006.
7. **Highsmith James A.** Adaptive Software Development: A Collaborative Approach to Managing Complex Systems [Book]. - [s.l.] : Dorset House Publishing Company, 1999.
8. **Holdener Anthony T.** Ajax: The Definitive Guide [Book]. - [s.l.] : O'Reilly Media, 2008.
9. **IEEE Software Engineering Standards** Architectural Description of Software-Intensive Systems - IEEE Std 1471-2000 [Journal] // IEEE Computer Society. - September 2000.
10. **Lankhorst Marc** Enterprise Architecture at Work: Modelling, Communication and Analysis [Book]. - [s.l.] : Springer, 2005.
11. **Larman Craig** Applying UML and Patterns [Book]. - [s.l.] : Prentice Hall PTR, 1997.
12. **Lars Lundberg Jan Bosch, Daniel Häggander and Per-Olof Bengtsson** Quality Attributes in Software Architecture Design [Book]. - Soft Center, S-372 25 Ronneby, Sweden : University of Karlskrona/Ronneby, 2007.
13. **Niemiec Richard** Oracle Database 10g Performance Tuning Tips & Techniques [Book]. - [s.l.] : McGraw-Hill Osborne Media, 2007.
14. **Perry Bruce** Java Servlet & JSP Cookbook [Book]. - [s.l.] : O'Reilly Media, Inc, 2003.
15. **Reese George** Database Programming with JDBC and Java [Book]. - [s.l.] : O'Reilly Media, Inc, 2000.

16. **Russell Matthew A.** Dojo: The Definitive Guide [Book]. - [s.l.] : O'Reilly Media, Inc, 2008.
17. **Salkosuo Sami** DWR Java AJAX Applications [Book]. - [s.l.] : Packt Publishing , 2008.
18. **Spector Phil** Data Manipulation with R [Book]. - [s.l.] : Springer, 2009.
19. **Vivek Chopra Amit Bakore, Jon Eaves, and Ben Galbraith** Professional Apache Tomcat 5 [Book]. - [s.l.] : Wrox, 2004.

INDICE