# ESCUELA POLITÉCNICA NACIONAL

## ESCUELA DE INGENIERÍA

# HERRAMIENTA GENERADORA DE APLICACIONES WEB CON ARQUITECTURA MVC EN JAVA

**PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN**

**AYALA LEON KLEBER EDUARDO**
**HINOJOSA ALVEAR ARTURO XAVIER**

**DIRECTOR: ING. FRANCISCO VILLAVICENCIO**

Quito, Febrero 2007

**CONTENIDO**

**ÍNDICE DE FIGURAS.**

**Capitulo Uno**



**Capitulo Dos**



**Capitulo Tres**

**Capitulo Cuatro**

**ÍNDICE DE TABLAS**

**Capitulo Dos**



**Capitulo Cuatro**

## INTRODUCCIÓN

El proceso de codificación de sistemas de información requiere bastante tiempo y esfuerzo por parte de los equipos de programación, cabe destacar que gran parte de este tiempo se gasta en la construcción de código que permita la manipulación de datos; es decir la construcción de operaciones de (inserción, edición, eliminación, despliegue de datos). Dicho tiempo puede ser optimizado si brindamos a los desarrolladores una herramienta que ayude a generar este código; permitiendo concentrar sus esfuerzos para implementar la lógica del negocio.

Dentro de este contexto, el presente estudio busca aplicar los conocimientos de ingeniería a fin de construir una herramienta que permita generar el código para manipulación de datos a partir de un esquema de base de datos.

La principal funcionalidad de la herramienta es la generación de código, para lo cuál partimos de los metadatos que guarda el dmbs; los mismos que dan información acerca de las estructuras de datos, sus campos, tipos y longitud. Aplicando este esquema a varias plantillas xml, podemos obtener las distintas capas de una arquitectura sólida que permita la manipulación de dichos datos.

El primer paso consiste en obtener la metadata de un dbms para aplicarla a plantillas XML, cada plantilla XML permite construir una capa, de acuerdo a la arquitectura “Modelo Vista Controlador”.

Una vez generado el código se realizará un esfuerzo adicional a fin de construir un archivo de configuración XML, que permita al servidor web identificar los archivos que contienen la solución, donde se encuentran guardados y como se comunican entre sí; finalmente se construirá una funcionalidad para poder guardar el código generado bajo una estructura de directorios estándar, la misma que podrá ser configurada por el usuario.

La herramienta será construida para generar código de aplicaciones web escritas en java, debido a que es el lenguaje que mayor complejidad muestra a la hora de desarrollar aplicaciones web.

# CAPITULO 1. INTRODUCCION

## 1.1 DEFINICION DEL PROBLEMA

El mundo actual requiere de soluciones informáticas cada vez más complejas las cuales involucran mayor esfuerzo por parte de los equipos de desarrollo de software.

Uno de los procesos más costosos es el de la codificación debido a:

- *Tiempo excesivo de codificación para manipulación de datos*. Construcción de operaciones básicas sobre las estructuras de datos: Inserción, Eliminación, Edición, Búsqueda.
- *Multiplicación de errores.* Comúnmente los programadores copian el código a fin de ahorrar tiempo. Si dicho código contiene errores será replicado a lo largo de la aplicación.

## 1.2 JUSTIFICACION

Dentro de este contexto el presente estudio busca optimizar el proceso de codificación; Aplicando los conocimientos de ingeniería a fin de construir una herramienta generadora de código que facilite el trabajo de los desarrolladores.

Dicha herramienta permitirá la generación de código de una aplicación web, generada a partir de un esquema de base de datos.

Esta aplicación permitirá el manejo básico de las estructuras de datos generando operaciones como: Inserción, Edición, Eliminación, Búsqueda.

Permitiendo así obtener las siguientes ventajas.

- *Aplicación Fácil de Mantener*. Cuando se requieran cambios son fáciles de hacer debido a la separación por capas.
- *Código Estandarizado*. Al ser generado presentara un estándar que será fácil de seguir por los programadores
- *Aplicación Escalable*. Podrá crecer de una forma ordenada.
- *Reducción de errores de codificación*. El código será generado por lo cuál se evitaran errores humanos en la codificación.

- *Los programadores pueden concentrar sus esfuerzos en la lógica del negocio*. Gracias a que las tareas cotidianas serán resueltas por la herramienta generadora de código.
- *Reducción de tiempo en la construcción de aplicaciones*.

Las tecnologías involucradas en la construcción de la herramienta generadora de código son:

- Arquitectura Modelo Vista Controlador MVC.
- JAVA 5.
- Struts Framework

### 1.2.1 ARQUITECTURA MVC

La arquitectura MVC (*Model View Controller*) fue introducida como parte de la del lenguaje de programación Smalltalk. Fue diseñada para reducir el esfuerzo de programación necesario en la implementación de sistemas. Sus características principales son que: el Modelo, las Vistas y los Controladores se tratan como entidades separadas; esto hace que cualquier cambio producido en el Modelo se refleje automáticamente en cada una de las Vistas. [11] [12]

En la figura 1.1 vemos la arquitectura MVC en su forma más general. Hay un Modelo, múltiples Controladores que manipulan ese Modelo, y hay varias Vistas de los datos del Modelo, que cambian cuando cambia el estado de ese Modelo.

***Definición de las partes***

***El Modelo.-*** Representa los datos y las reglas de negocio que gobiernan la manipulación de los mismos. El Modelo no tiene conocimiento específico de los Controladores o de las Vistas, ni siquiera contiene referencias a ellos. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el Modelo y sus Vistas, y notificar a las Vistas cuando cambia el Modelo.

---

[11] JAVA SUN MICROSYSTEMS, Model-View-Controller, http://java.sun.com/blueprints/patterns/MVC.html, Ultimo Acceso: Junio 2006 Junio 2006.

[12 ]JAVA SUN MICROSYSTEMS, Web-Tier Application Framework Design http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html , Ultimo Acceso: Junio 2006.

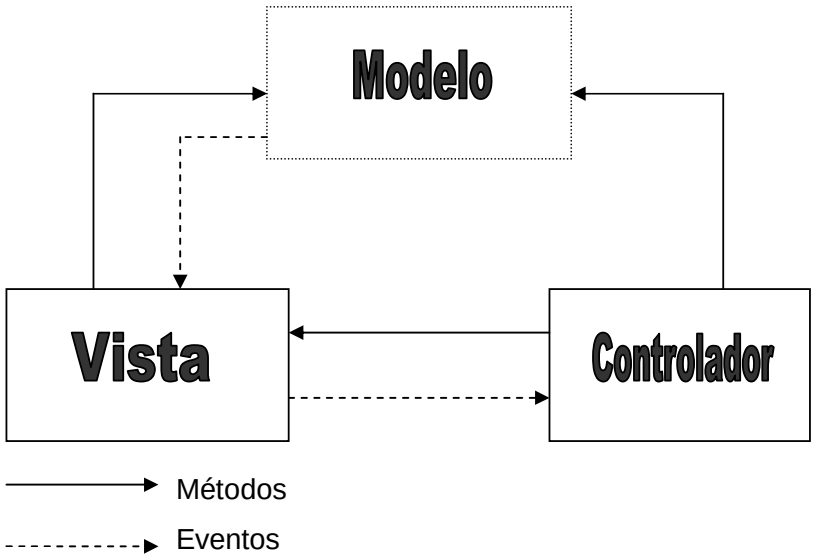



**Figura 1. 1 Arquitectura MVC.**

***La Vista.-*** Muestra el contenido del modelo, la vista accede a los datos a través del modelo y especifica como los datos deben ser desplegados. También es responsable de mantener la consistencia en la presentación cuando el modelo sufre cambios. Esto lo puede lograr gracias a una referencia al propio Modelo.

***El Controlador***.- Es el objeto que proporciona significado a las ordenes del usuario, actuando sobre los datos representados por el Modelo. Cuando se realiza algún cambio, entra en acción, bien sea por cambios en la información del Modelo o por alteraciones de la Vista.

*1.2.1.1 VENTAJAS*

Este modelo de arquitectura presenta varias ventajas:

- Hay una clara separación entre los componentes de un sistema. Lo cual nos permite implementarlos por separado.
- Reutilización de los Componentes del modelo. Un modelo empresarial puede ser desplegado fácilmente en distintas vistas. Dicho modelo puede ser probado e incluso modificado fácilmente sin afectar a las vistas.
- Hay un API muy bien definido. Cualquiera que use el API, podrá reemplazar el Modelo, la Vista o el Controlador, sin aparente dificultad.
- La conexión entre el Modelo y sus Vistas es dinámica. Se produce en tiempo de ejecución, no en tiempo de compilación.

- Nuevos tipos de clientes; Para soportar un nuevo tipo de cliente simplemente se debería escribir la vista y una pequeña parte de la lógica y atarlas a una aplicación existente.

Al incorporar el modelo de arquitectura MVC a un diseño, las piezas de un programa se pueden construir por separado y luego unirlas en tiempo de ejecución. Si uno de los Componentes, posteriormente, se observa que funciona mal, puede reemplazarse sin que las otras piezas se vean afectadas.

#### *1.2.1.2 DESVENTAJAS*

Este modelo no presenta mayores desventajas:

- Complejidad en el diseño de la vista. Se presenta debido a que se introducen varias clases para hacer posible la separación entre el modelo la vista y el controlador.

### 1.2.2 CARACTERISTICAS GENERALES DE JDK 1.5

El 23 de mayo de 1995, la empresa Sun Microsystems anunció el lanzamiento de la tecnología Java. Desde ese entonces hemos visto cómo la plataforma Java ha ido evolucionando desde aplicaciones simples de escritorio y aplicaciones Web, hasta aplicaciones empresariales distribuidas. Para el cuarto trimestre del 2004 lanzo, la última versión del Java Standard Edition Platform (conocido como "tiger project") fue liberada como Java 5. [10] [13]

El principal objetivo de esta versión JDK 5 es la facilidad de desarrollo con las nuevas características como tipos de datos genéricos, el loop for-each, autoboxing, unboxing, enums, varargs, static imports.

El JDK 5 está diseñado para hacer que los programas se estructuren de manera más ordenada, más corta y fortificando la seguridad. Además, proporciona un soporte lingüístico para idiomas comunes.

#### *1.2.2.1 Tipos Genéricos, For-Each*

Los tipos genéricos permiten hacer conversiones automáticos de objetos, permitiendo subir o bajar automáticamente las jerarquías de objetos.

Los tipos genéricos nos permiten especificar el tipo de elemento de una colección (*Collection)* en tiempo de compilación. En lugar de especificar una lista, especificamos su contenido.

El la figura 1.2 podemos observar el uso del For-each para iterar a través de una Lista llamada "todo" que no es más que una lista de Double. Notemos que no hay necesidad de hacer casting y el for es muy fácil de leer:

---

[10] RAHUL TYAGI, Java Programming: Explore the advanced coding features of JDK 1.5, http://builder.com.com/5100-6370-5147404.html, Ultimo Acceso: Junio 2006.

[13] SUN MICROSYSTEMS , New Features and Enhancements J2SE 5.0, http://java.sun.com/j2se/1.5.0/docs/relnotes/features.html, Ultimo Acceso: Junio 2006.

```
public double sumar (Collection todo) {

double total = 0;

for(Double i : todo){

total += i;

}

return total;

}
```

**Figura 1. 2 Ejemplo de tipos genéricos.**

Usando los tipos de datos genéricos mejoraremos la claridad en el código y evitaremos problemas con los tipos de datos.

Se puede utilizar la sentencia  for-each en los siguientes casos:

- Eliminar elementos mientras recorremos un Collection.
- Modificar la posición actual en un arreglo o lista.
- Iterar en múltiples Collections o arrays.

*1.2.2.2  Conversion automatica de tipos. ( Autoboxing - Unboxing.)*

Es la conversión automática de tipo que realiza el compilador de java entre los tipos primitivos y sus correspondientes clases wraper. Ejemplo: int e Integer, double y Double, etc.

La figura 1.3 muestra las ventajas que presenta el uso de autoboxing. El mismo que evita tareas tediosas como hacer cast.

| Con Autoboxing | Sin Autoboxing |
|---|---|
| `int i;`<br>`Integer j;`<br>`i = 1;`<br>`j = 2;`<br>`i = j;`<br>`j = i;` | `int i;`<br>`Integer j;`<br>`i = 1;`<br>`j = new Integer(2);`<br>`i = j.intValue();`<br>`j = new Integer(i);` |

**Figura 1. 3 Ejemplo de autoboxing.**

Es recomendable usar esta característica cuando hay una incompatibilidad entre las tipos de referencia y los datos primitivos. Aunque esto no es apropiado para cálculos científicos. Un Integer no es un substituto para un int. El autoboxing y unboxing simplemente ocultan las diferencias entre los wrappers y los datos primitivos.

*1.2.2.3 Enumerated Types (enums)*

Un enumerated type es un tipo de dato que contiene un conjunto de valores constantes.

En la figura 1.5 podemos ver un ejemplo de enumerados.

```
enum DiasSemana (Lunes, Martes, Miércoles, Jueves,  Viernes)
```

**Figura 1. 4 Ejemplo de enumerados.**

*1.2.2.4 Static Imports*

Los Static imports permiten acceso a los miembros estáticos extendiendo el tipo. Todos los campos estáticos, métodos, etc. estarán disponibles para nuestra Clase usando los static imports. Por ejemplo:

En la figura 1.5 podemos ver un ejemplo de static imports.

```
import static java.lang.Math.*;

...

r = cos(PI * theta);
```

**Figura 1. 5 Ejemplo de static imports.**

*1.2.2.5 Varargs*

Los Varargs son métodos que reciben un número arbitrario de argumentos requeridos para crear un arreglo. Varargs automatiza y oculta el proceso.

En la figura 1.6 podemos ver un ejemplo de varargs.

```
public static String format(String pattern, Object... arguments);


String result = MessageFormat.format("At {1,time} on {1,date}, there was {2} on planet " + "{0,number,integer}.", 7, new Date(), "a disturbance in the Force");
```

**Figura 1. 6 Ejemplo de varargs**

*1.2.2.6 Metadata*

Inclusión de metadata relacionada a las Clases Java, interfaces, campos y métodos. Las herramientas de desarrollo usarán el soporte de esta metadata para la inserción automática de código fuente adicional, para debbug, y para la creación y modificación de archivos XML externos.

## 1.3 STRUTS FRAMEWORK

Struts Framework es un estándar para desarrollo de aplicaciones web, el mismo que es una implementación de la arquitectura MVC: [1]

### 1.3.1 CARACTERISTICAS GENERALES DE STRUTS FRAMEWORK

- Aplicación de código abierto (*Open Source*).
- Basado en la arquitectura MVC.
- Guarda la información del ruteo de la aplicación en un archivo xml “struts-config.xml”.
- Provee por si mimo la Vista y el Controlador. El programador implementa el Modelo.

---

[1] APACHE , Apache Struts, http://struts.apache.org/ Ultimo Acceso: Junio 2006.

### 1.3.2 ARQUITECTURA.

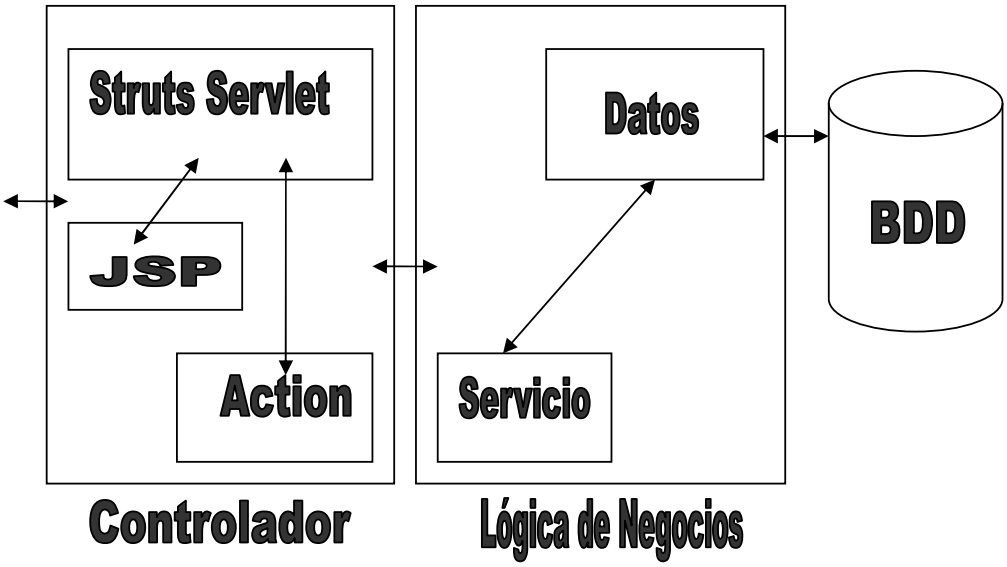



**Figura 1. 7 Arquitectura Struts Framework**

Las solicitudes entrantes son interceptadas por el controlador de Struts Servlet. El archivo de configuración struts-config.xml es usado por el controlador para determinar el flujo que se llevara acabo. Dicho flujo consiste en la interacción de las siguientes transacciones.

*Desde la vista hacia la acción.-* El usuario ejecuta una acción como un clic o el envió de un formulario. El controlador recibe la solicitud, busca el mapeo correcto para esta petición y la reenvía a una acción. La acción llama al Modelo para ejecutar una función.

*De la acción a la vista.* Después de la llamada la función solicitada retorna a la clase de acción, la acción retorna un recurso a la vista y la página de resultado es desplegada hacia el usuario.

La figura 1.8, muestra la interacción entre los diferentes componentes de struts framework

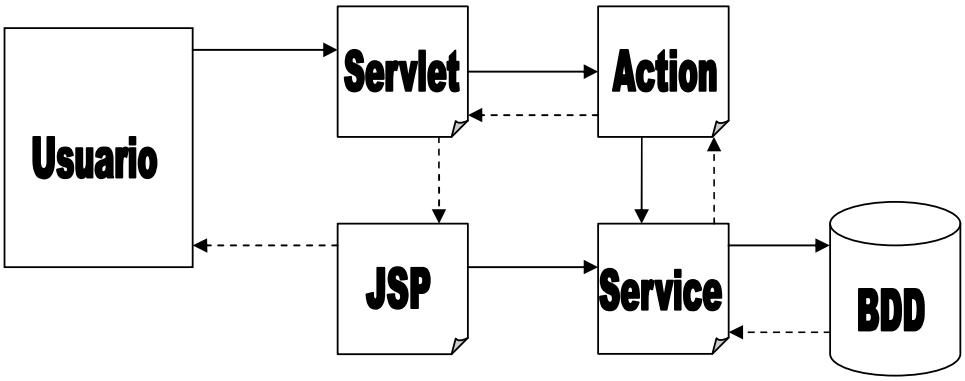



**Figura 1. 8 Explicación modelo Struts Framework**

1. **Usuario.** Hace un click en una página.
2. **Servlet.** Recibe la petición y busca su correspondiente mapeo dentro del archivo "struts-config.xml", y rutea el flujo a la acción.
3. **Acción.** Hace una llamada al servicio.
4. **Servicio.** Hace una llamada a la capa de datos, la misma que retorna un conjunto de datos de respuesta.
5. **Servicio.** Retorna el control a la acción.
6. **Acción.** Retorna una vista a una página jsp.
7. **Servlet** Busca el mapeo mas adecuado para la petición solicitada y lo asigna a la página jsp mas adecuada.
8. **JSP** Una página es invocada la misma que envía el html para que sea desplegado en el navegador de Internet.
9. **Usuario.** los datos son visualizados como una nueva página html.

### 1.3.3 COMPONENTES DE STRUTS FRAMEWORK

*1.3.3.1 El Controlador.*

Encargado de recibir todas las peticiones entrantes, Su principal función es rutiar las peticiones entrantes a una clase de acción.

*1.3.3.2 Archivo de configuración struts-config.xml.*

Este archivo contiene la información para el ruteo de peticiones de la aplicación. Este archivo debe estar situado en la carpeta “web-inf” de la aplicación.

*1.3.3.3 Clases de Acción.*

Actúan como puentes entre las invocaciones de los clientes y los servicios de negocios. Las clases de acción procesan las peticiones y retornan objetos denominados Respuesta de Acción, los cuales son capaces de identificar que componente se debe identificar. Forman parte de la capa del Controlador.

*1.3.3.4 Recursos de la Vista (View Resources).*

Conformado por: páginas jsp, páginas html, Java Script, Hojas de estilo, Java Beans, Struts JSP Tags.

*1.3.3.5 Formularios de Acción (ActionForms).*

Permiten validar los datos enviados por el usuario antes de que estos sean procesados,

*1.3.3.6 Componentes de Modelo (Model Components).*

Struts Framework es capaz de soportar cualquier modelo Ejemplo:

- JavaBeans
- EJB
- CORBA
- JDO

### 1.3.4 VENTAJAS

- *Uso de Etiquetas.* Promueve el código reusable y permite abstraer el código java de las páginas jsp.
- *Librería de Etiquetas.* Proporciona un conjunto grande de etiquetas para distintas aplicaciones.
- *Código Abierto.* Cuentas con todas las ventajas del código abierto. Como la modificación del código fuente de struts framework.
- JDO

### 1.3.5 DESVENTAJAS.

- *Cambios Frecuentes.* Struts Framework esta sujeto a cambios constantes debido a que se encuentra en un proceso de maduración. Dichos cambios no aseguran que los programas escritos en antiguas versiones sean soportados en las nuevas versiones.
- *Alcance Limitado.* Soporte para determinados contenedores de aplicaciones web.

## 1.4 METODOLOGIA DE DESARROLLO

A lo largo del proceso de desarrollo es importante contar con una metodología de desarrollo de software debido a que se necesita manejar las distintas etapas el ciclo de vida del software. Con la utilización de una metodología apropiada se puede asegurar el éxito en el proceso de desarrollo. [7] [8] [9]

Existen una variedad de metodologías de las cuales veremos tres, las mismas que nos brindan características comunes que nos pueden ser de utilidad: Racional Unified Process, Programación Extrema, Prototipos.

### 1.4.1 Racional Unified Process

La metodología RUP, llamada así por sus siglas en inglés Rational Unified Process, divide en cuatro fases el desarrollo del software:

***Inicio***, Panorama claro del proyecto (*visionamiento*).
***Elaboración***, Determinar la arquitectura.
***Construcción***, Codificación del proyecto.
***Transmisión***, Obtener una versión del producto funcional que se pueda entregar al cliente.

Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, el cual consiste en reproducir el ciclo de vida en cascada. Los Objetivos de una

---

[7] MICROSOFT, Métodos Heterodoxos en Desarrollo de Software, http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/heterodox.asp, Ultimo Acceso: Junio 2006.

[8] PABLO FIGUEROA, Metodología de desarrollo de software Orientado por Objetos, http://www.cs.ualberta.ca/~pfiguero/soo/metod/, Ultimo Acceso: Junio 2006.

[9] RATIONAL, Rational Unified Proces, http://www.augustana.ab.ca/~mohrj/courses/2000.winter/csc220/papers/rup_best_practices/rup_bestpractices.html#1, Ultimo Acceso: Febrero 2007.

iteración se establecen en función de la evaluación de las iteraciones de fases anteriores.

El ciclo de vida que se desarrolla por cada iteración, es llevada bajo dos disciplinas:

***Disciplina de Desarrollo***

**Ingeniería de Negocios**: Obtener las reglas de negocio.

**Requerimientos**: Impregnar las necesidades de negocio en un sistema automatizado.

**Análisis y Diseño**: Acoplar los requerimientos a la arquitectura de software.

**Implementación**: Creando software que se ajuste a la arquitectura y que tenga el comportamiento deseado.

**Pruebas**: Permite validar la calidad del software y el cumplimiento de los requerimientos.

***Disciplina de Soporte***

***Configuración y administración del cambio***: Versionamiento del proyecto.

***Administración del proyecto***: Administración de horarios y recursos.

***Ambiente***: Administración del ambiente de desarrollo.

***Distribución***: Entrega del proyecto.

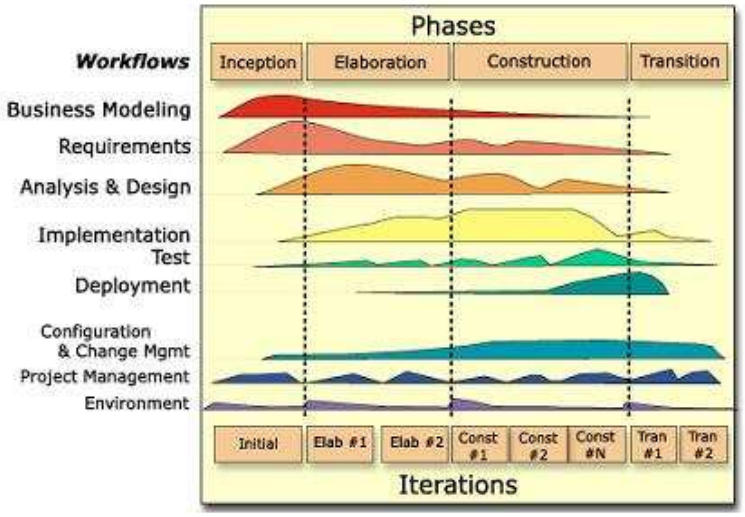



**Figura 1. 9 Fases e iteraciones.** [9]

En la figura 1.9 podemos observar las Fases e iteraciones de la mitología RUP. Es recomendable que a cada una de estas iteraciones se las clasifique y ordene según su prioridad, y que cada una se convierte luego en un entregable al cliente. Esto trae como beneficio la retroalimentación que se tendría en cada entregable o en cada iteración.

### 1.4.2 Programación Extrema (XP)

Es una de las metodologías de desarrollo de software más exitosas en la actualidad utilizada para proyectos que cuentan con un equipo pequeño y un plazo corto. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

---

[9] RATIONAL, Rational Unified Proces, http://www.augustana.ab.ca/~mohrj/courses/2000.winter/csc220/papers/rup_best_practices/rup_bestpractices.html#1, Ultimo Acceso: Febrero 2007.

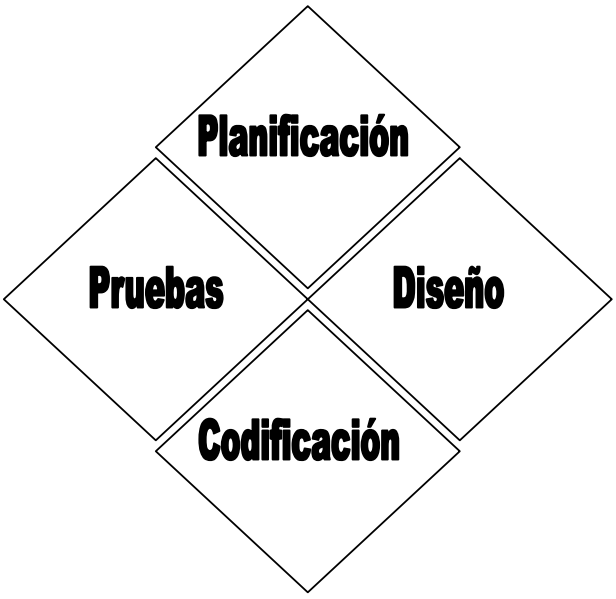



**Figura 1. 10 Metodología Programación Extrema**

**Elementos del Proceso.**

***Pruebas Unitarias***: Pruebas a los procesos a fin de determinar posibles errores que pudieran darse en la etapa de codificación.

***Refabricación***: Creación de patrones, componentes reutilizables.

***Programación en pares***: una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Esto permite escribir código más rápido ya que los desarrolladores comparten su conocimiento. Otra ventaja es que analiza y cubre de mejor manera los requerimientos del cliente.

*1.4.2.1 Ventajas*

- Retroalimentación continúa
- El costo del cambio no depende de la fase o etapa
- No introduce funcionalidades antes que sean necesarias
- El cliente o el usuario se convierte en miembro del equipo.
- Saber el estado real y el progreso del proyecto.
- Decidir como se implementan los procesos.

- Pedir al cliente en cualquier momento aclaraciones de los requerimientos.

*1.4.2.2 Desventajas*

- El manejo del cambio se convierte en parte sustantiva del proceso
- Añadir, cambiar o quitar requerimientos en cualquier momento
- No se puede estimar eficientemente el esfuerzo para construir el sistema.

### 1.4.3 Desarrollo por Prototipos.

El modelo por prototipos es una versión cíclica del modelo en cascada, luego del análisis se diseña un prototipo e inicia el desarrollo, este prototipo es liberado hacia el cliente para evaluación, una vez que aprueba el cliente se produce una nueva iteración en el ciclo es decir se vuelve al desarrollo a fin de mejorar dicho prototipo.

***Etapas del Método con Prototipos***

- Identificación De Requerimientos Conocidos.
- Desarrollo De Un Modelo De Trabajo.
- Participación Del Usuario.
- Revisión Del Prototipo.
- Iteración Del Proceso De Refinamiento.

En la figura 1.11 podemos observar El método con prototipos.

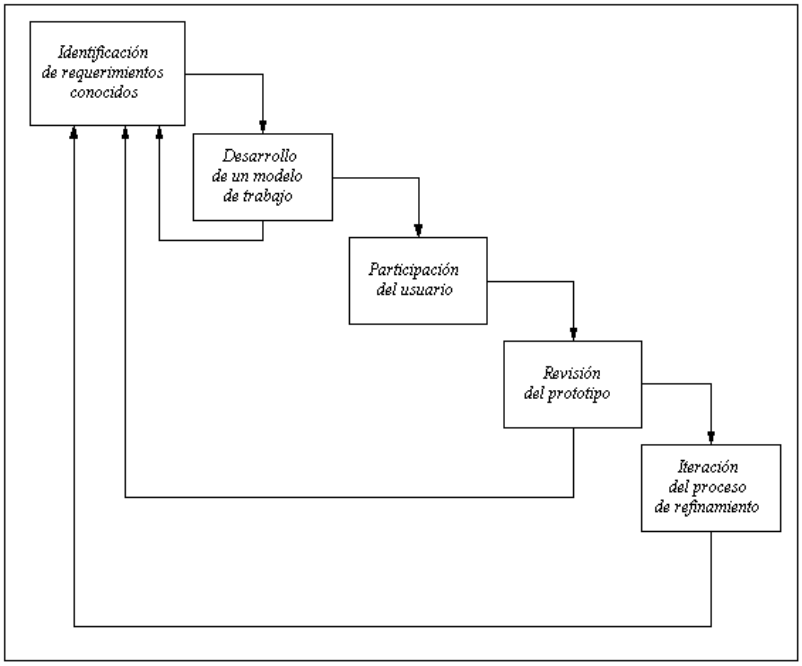



Figura 1. 11 **Metodología Prototipos**

*1.4.3.1 Ventajas*

- Disminución de errores de diseño.- El usuario podrá ver el producto desde etapas tempranas de construcción lo que permitirá corregir errores de diseño en esta etapa.
- Retroalimentación.- El cliente puede compartir su conocimiento acerca del negocio, expresar de mejor manera sus requerimientos en base a los prototipos que son construidos.

*1.4.3.2 Desventajas*

- Rechazo.- Los prototipos pueden ser fácilmente rechazados por los usuarios.

- Distorsión de ideas.- Los desarrolladores pueden tomar demasiado tiempo realizando prototipos perdiendo así el tiempo que deben dedicar a resolver los problemas de negocio.

En base al análisis de estas tres metodologías hemos seleccionado la metodología RUP debido a:

- Clara definición de requerimientos.
- Verificar la factibilidad del diseño del sistema mientras este es construido.
- Lenguaje de modelado claro de fácil comprensión.

La naturaleza de la metodología es iterativa para este caso hemos considerado necesario realizar una sola iteración debido al tamaño del sistema, de la fase de pruebas con usuarios obtendremos la retroalimentación necesaria que nos permita realizar las mejoras necesarias para el correcto funcionamiento de la herramienta.

Para el análisis de requerimientos de este sistema utilizaremos el estándar provisto por la IEEE. “IEEE Recommended Practice for Software Requirements Specification. AN-SI/IEEE std. 830, 1998.” El cuál nos permitirá obtener una definición clara de los requerimientos. Para modelar el sistema utilizaremos el Lenguaje de Modelado Unificado (UML - Unified Modeling Language). Del cual implementaremos tres diagramas que consideramos nos permiten visualizar cada una de las partes estructurales del sistema y abarcan la vista de casos de uso (diagrama de Casos de Uso), la vista estática (diagrama de Clases) y la vista dinámica (diagrama de Actividad)

### 1.4.4 Lenguaje Unificado de Modelado (UML)

Describe un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos, y describe la semántica esencial de lo que estos diagramas y símbolos significan. [14]

Podemos definir UML como un lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software. UML entrega una forma de modelar cosas conceptuales como lo son procesos de negocio y funciones de sistema, además de cosas concretas como lo son escribir clases en un lenguaje determinado, esquemas de base de datos y componentes de software reusables.

#### *1.4.4.1 Diagrama de Clases.*

Un diagrama de clases sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de agrupamiento.

Un diagrama de clases esta compuesto por los siguientes elementos:

- Clase: atributos, métodos y visibilidad.
- Relaciones: Herencia, Composición, Agregación, Asociación y Uso.

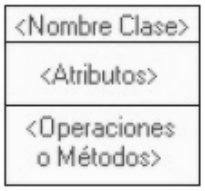



Figura 1. 12 **Diagrama de Clases**

[14] Terry Quatrani, Introduction to the Unified Modeling Language, ftp://ftp.software.ibm.com/software/rational/web/whitepapers/2003/intro_rdn.pdf, Ultimo Acceso: Noviembre 2006.

*1.4.4.2 Diagrama de Casos de Uso.*

El diagrama de casos de uso representa la forma en como un Cliente (Actor) opera con el sistema en desarrollo, además de la forma, tipo y orden en como los elementos interactuan (operaciones o casos de uso).

Un diagrama de casos de uso consta de los siguientes elementos:

- Actor.
- Casos de Uso.
- Relaciones de Uso, Herencia y Comunicación.

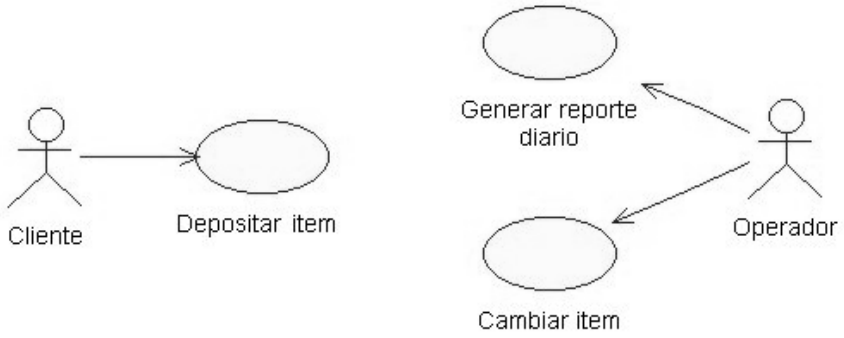



Figura 1. 13 **Diagrama de Casos de Uso**

*1.4.4.3 Diagramas de Actividad.*

Un diagrama de actividades puede considerarse como un caso especial de un diagrama de estados en el cual casi todos los estados son estados acción identifican una acción que se ejecuta al estar en él y casi todas las transiciones evolucionan al término de dicha acción ejecutada en el estado anterior. Permiten representar transiciones internas al margen de las transiciones o eventos externos.

Componentes

- **Inicio**: El inicio de un diagrama de actividad es representado por un círculo de color negro sólido.

- **Actividad**: Una actividad representa la acción que será realizada por el sistema la cual es representada dentro de un ovalo.

- **Transición**: Una transición ocurre cuando se lleva a cabo el cambio de una actividad a otra, la transición es representada simplemente por una línea con una flecha en su terminación para indicar dirección.

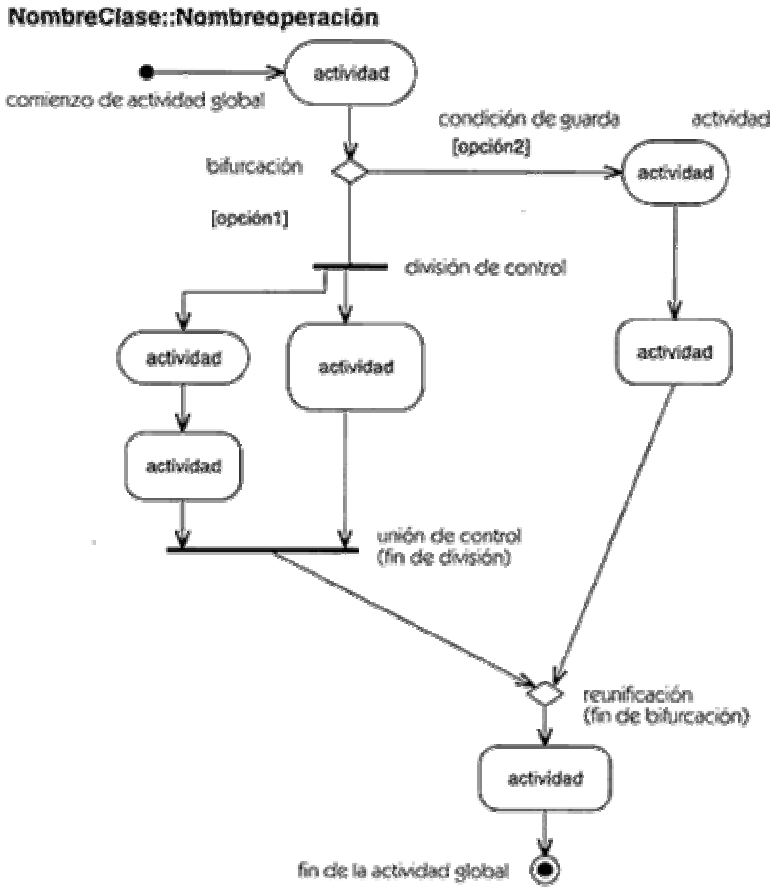



Figura 1. 14 **Diagrama de Actividad**

# CAPITULO 2. ANÁLISIS Y DISEÑO

De acuerdo con la metodología que escogimos empezaremos con la fase de ***Inicio***, donde obtendremos un panorama claro del sistema y definiremos los requerimientos del sistema.

## 2.1 DEFINICIÓN DE ESTÁNDARES DE CODIFICACIÓN

### 2.1.1 ESTÁNDAR DE CODIFICACIÓN

El estándar que se utilizará para la codificación de la aplicación, está basado en prácticas comunes propuestas por varias industrias dedicadas al desarrollo de software. [6],

#### *2.1.1.1 Estilo y Nombres* |

Prefijos y Postfijos:

La Tabla 2.1 muestra el detalle de los prefijos y postfijos que se proponen en el estándar.

| OBJETO | PREFIJO | POSTFIJO | EJEMPLO |
|---|---|---|---|
| Interfaz | I | | IMyInterface |
| Variables Privadas | m_ | | m_MyValue |
| Atributos | | Attribute | MyOwnAttribute |
| Excepciones | | Exception | MyNewException |

**Tabla 2.1 Especificación de Prefijos y Postfijos**

Se usará la convención Pascal para nombres de clases, métodos y constantes y la convención Camel para variables y argumentos de los métodos. Un ejemplo se lo puede observar en la Figura 2.1:

---

[6] José Díaz, Estándares de Codificación Java, http://www.itprofessionals.com.pe/home/index.php?option=com_content&task=view&id=39&Itemid=51 Ultimo Acceso: Junio 2006.

```
public class miClase
{
        const int MiParametro;
        int varName;
        public void MiMetodo(int argNombre)
}
```

**Figura 2.1 Ejemplo de Convención Pascal**

- Declarar las variables locales lo más cerca posible de su primer uso.
- El nombre del archivo debe reflejar la clase que contiene.
- Colocar siempre las llaves ({) en una nueva línea.
- Colocar un espacio antes de abrir los paréntesis (() en la declaración y llamada de métodos.
- Nombrar a los métodos usando verbos. Ej.: ShowDialog ();
- Evitar el uso del nombre completo de los tipos, en su lugar usar 'using'.
- Alinear los comentarios al nivel del código que se está documentando.
- Las variables deben declarase al inicio, con una línea en blanco para separarlas de las propiedades o métodos.
- Los métodos que retornan un valor deben nombrarse de forma que se entienda que es lo que se retorna. Ej.: GetObjectState ();
- Usar nombres descriptivos en las variables, incluyendo el uso de palabras como 'index' o 'temp' en lugar de nombres de un solo carácter como 'i' o 't', además evitar el abreviar las palabras, por ejemplo usar 'number' en lugar de 'num'.
- Usar los tipos predefinidos en C# en lugar de sus alias. Ej.: Usar 'int' No 'Int32'.

La Convención de Estilos y Nombres se describe en la Figura 2.2:

```
using System;
class Demo : IDisposable
{
        private bool m_AlreadyDisposed = false;

        //Punto de entrada principal
        public static int Main ()
        {
                for (int index=0; index<10; index++)
                {
                        Console.WriteLine ("Hello World");
                }
        }
        private class MyNewException : Exception
        {
                public MyNewException (): base ()
                {
                }
        }

        /**
        * Implementacion de la clase
        */
}
```

**Figura 2.2 Convención de Estilos y Nombres**

*2.1.1.2 Practicas de Codificación*

- Con excepción de 0 y 1, nunca 'quemar' un valor numérico en el código, en su defecto usar constantes.
- Evitar el usar códigos de error como valor de retorno de los métodos.
- Usar siempre las llaves ({) para todas las funciones como if, while, do, etc. inclusive si solo encierran una sola sentencia.
- Evitar el uso del operador ternario (?).
- Evitar llamar a funciones que retornen valores booleanos en sentencias condicionales, de preferencia asignar el valor retornado a una variable y usar esta en el condicional.

- Usar siempre arreglos con base 0.
- Usar ‘const’ solo para constantes naturales como PI o el número de meses del año, para otras constantes propias preferir el uso de ‘*readonly’*.
- Usar aserciones en cada suposición dentro del código.
- Capturar solamente excepciones explícitas como ‘FileNotFoundException’ en lugar de ‘Exception’.
- En los métodos que propagan excepciones, usar siempre la excepción original.
- Inicializar siempre los arreglos usando una sentencia ‘for’.
- No usar variables públicas o protegidas, usar variables privadas y proveer propiedades para el acceso.
- Evitar el uso de ‘new’ para la herencia, de preferencia usar ‘override’.
- Definir los métodos públicos y protegidos como virtuales en clases no selladas.

### 2.1.2 ESPECIFICACIÓN DE REQUERIMIENTOS

Según el apartado “1.4 Metodología de Desarrollo” utilizaremos para esta sección el estándar para análisis de requerimientos proporcionado por la IEEE.

#### *2.1.1.1 ESPECIFICACIÓN DE REQUERIMIENTOS DE SOFTWARE[5],*

La herramienta será capaz de generar el código para la construcción de una aplicación web, la misma que estará constituida en base al patrón de diseño “Modelo Vista Controlador”.

Dicha generación de código se realizará a partir de un esquema de base de datos, la herramienta generará el código para las operaciones de inserción,

---

[5] IEEE Recommended Practice for Software Requirements Specification. ANSI/IEEE std. 830, 1998.

.

edición, eliminación y borrado; Para cada una de las capas que conforman el patrón de diseño “Modelo Vista Controlador”.

Hemos seleccionado la generación de estas operaciones debido a que en nuestra experiencia cotidiana en el desarrollo de software, son las más comunes debido a que son la base de todo sistema de software. Motivo por el cual se requiere de la automatización de estos procesos por cuanto son repetitivos y toman valioso tiempo que puede ser utilizado para implementar la lógica del negocio. Dicha generación podría realizarse para cualquier tecnología que exista en el mercado, de las cuales analizamos las dos más representativas dentro del mercado ecuatoriano .NET y JAVA; nos decidimos por JAVA debido a que la curva de aprendizaje es mucho más compleja que en .NET es decir los desarrolladores emplean mucho más tiempo aprendiendo e implementando sistemas en JAVA; por este motivo es de mayor utilidad construir la herramienta para tecnología java[4],

*2.1.1.2 Funciones del producto*

*2.1.1.3 GENERACIÓN DE CÓDIGO*

En base a un esquema de datos, la herramienta genera el código para las operaciones de inserción, eliminación, borrado, actualización. Basado en el patrón de diseño “Modelo Vista Controlador”, como resultado se crea una carpeta que contiene archivos con el código generado, además un archivo que permite la compilación y posterior instalación de la aplicación

*2.1.1.4 Características de los usuarios*

Los usuarios de la herramienta serán técnicos, específicamente programadores, de aplicaciones web con experiencia en la plataforma “Java 2 Platform Enterprise Edition” J2EE.

---

[4] Carol Silwa, “.Net vs. Java: Five Factors to Consider”, http://www.computerworld.com/developmenttopics/development/story/0,10801,71221,00.html, Ultimo Acceso Enero 2007

Debido a que es una herramienta que genera código para aplicaiones web basados en la plataforma java.

*2.1.1.5 Restricciones*

- ***El código generado no será compilado***. La herramienta se encarga de la generación del código de la aplicación, mas no de la compilación del mismo. Para compilar y ejecutar le código generado se debe utilizar una herramienta de desarrollo de aplicaciones java como eclipse.
- ***La herramienta no implementa lógica de negocio.*** La herramienta genera código solo para operaciones comunes: Insertar, Editar, Eliminar, Borrar, mas no genera código específico del negocio.

*2.1.1.6 Suposiciones y dependencias*

Se asume que el sistema podrá obtener el esquema de la base de datos seleccionada por el usuario y además que dicho esquema contenga tablas.

*2.1.1.7 Requerimientos específicos*

*2.1.1.8 Interfaces externas*

*2.1.1.9 Interfaz de usuario.*

La interfaz de usuario contará con una pantalla que permita conectarse con un motor de base de datos y seleccionar la carpeta donde se guardará los resultados de la generación; finalmente un área que permita desplegar los resultados.

*2.1.1.10 Interfaz de hardware.*

Para el funcionamiento de la herramienta es necesario un computador con las siguientes características.

| Característica | Requerimiento Mínimo |
|---|---|
| Velocidad de procesador | 3 ghz |
| Memoria RAM | 512 MB |
| Disco duro | 40 GB |
| Interfaz de Red | 100 Mbps |

**Tabla 2.2 Requerimientos mínimos de hardware.**

*2.1.1.11 Interfaces de Software*

Para el funcionamiento de la herramienta es necesario disponer de la siguiente plataforma de software:

| Tipo | Plataforma de software | Versión |
|---|---|---|
| Sistema operativo | Windows XP.<br>MAC OSX.<br>Linux | 2002 o superior<br>10.4 |
| DBMS | MySQL | 5.0 |
| Plataforma de Programación | Java 2 Platform Standard Edition | 1.5 |
| Plataforma de aplicaciones | J2EE | 1.4 |
| Servidor de aplicaciones | JBOSS | 4.04 |

**Tabla 2.3 Plataforma de software**

*2.1.1.12 Interfaz de comunicación*

Controladores que permitan la comunicación entre java y un motor de base de datos.

*2.1.1.13 Requerimientos funcionales*

*REQ 1: Generación de Código*

*REQ 1.1: Generación de Código*

Entradas

- Esquema de base datos.

- Directorio para escribir el código generado

Procesos

La herramienta recibe el esquema de base de datos y ejecuta las siguientes acciones.

- Generar el Modelo.
- Generar la Vista.
- Generar el Controlador.
- Generar archivo de construcción (*deploy*)
- Escribir los archivos generados en el directorio seleccionado

Salidas

- Carpeta con código
- Mensaje de Confirmación.

#### *2.1.1.14 Atributos del sistema de software*

- **Autonomía:** La herramienta generadora debe contar con información propia.
- **Continuidad:** Brindará un servicio permanente.
- **Portabilidad:** La herramienta podrá ser instalada en cualquier plataforma que soporte tecnología Java.
- **Seguridad**: No incluye aspectos de seguridad.
- **Escalabilidad**: No incluye aspectos de estabilidad.

## 2.2 DISEÑO DE LA HERRAMIENTA

Acorde con la metodología de desarrollo especificado en la sección “1.4 Metodología de Desarrollo” empezamos la etapa de ***Elaboración***, en la cual se analizara y diseñara el sistema; Utilizaremos los casos de uso a fin de hacer el análisis del sistema y el diagrama de clases y actividad a fin de realizar el diseño del sistema.

### 2.2.1 DIAGRAMA DE CASOS DE USO

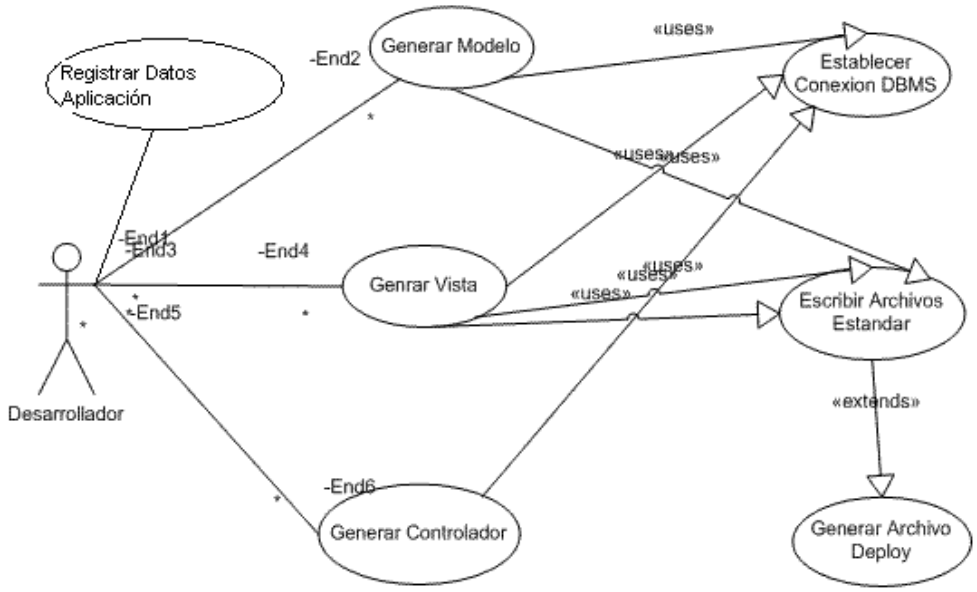



**Figura 2.3 Diagrama de Caso de Uso.**

***Caso de Uso: Registrar Datos Aplicación.***

Flujo Básico

1. El usuario llama a esta funcionalidad desde la pantalla principal por medio de la opción: "Registrar Datos Aplicación".
2. El usuario llena los parámetros solicitados por la aplicación.

3. El sistema almacena los datos registrados por el usuario.

***Caso de Uso: Generar Modelo.***

Flujo Básico

1. El usuario llama a esta funcionalidad desde la pantalla principal por medio de la opción: "Agregar Entidades".
2. El usuario establece conexión Con un DBMS. Para lo cual pasa al caso de uso "Establecer Conexión con DBMS".
3. El sistema genera el código para la capa: "Modelo". Y lo despliega en pantalla agrupando en secciones las operaciones de inserción, edición, búsqueda y borrado de datos.
4. El usuario puede copiar el código a fin de compilarlo en un programa que lo permita.

Especificaciones suplementarias

1. En caso de no poder establecer conexión con el DBMS, el sistema mostrará un mensaje informativo.

***Caso de Uso: Generar Controlador.***

Flujo Básico

1. El usuario llama a esta funcionalidad desde la pantalla principal por medio de la opción: "Agregar Session Bean".
2. El usuario establece conexión Con un DBMS. Para lo cual pasa al caso de uso "Establecer Conexión con DBMS".
3. El sistema genera el código para la capa: "Controlador". Y lo despliega en pantalla agrupando en secciones las operaciones de inserción, edición, búsqueda y borrado de datos.

4. El usuario puede copiar el código a fin de compilarlo en un programa que lo permita.

Especificaciones suplementarias

1. En caso de no poder establecer conexión con el DBMS, el sistema mostrará un mensaje informativo.

***Caso de Uso: Generar Vista.***

Flujo Básico

1. El usuario llama a esta funcionalidad desde la pantalla principal por medio de la opción: "Generar Vista".
2. El usuario establece conexión Con un DBMS. Para lo cual pasa al caso de uso "Establecer Conexión con DBMS".
3. El sistema genera el código para la capa: "Vista". Y lo despliega en pantalla agrupando en secciones las operaciones de inserción, edición, búsqueda y borrado de datos.
4. El usuario puede copiar el código a fin de compilarlo en un programa que lo permita.

Especificaciones suplementarias

1. En caso de no poder establecer conexión con el DBMS, el sistema mostrará un mensaje informativo.

***Caso de Uso: Establecer Conexión Con DBMS.***

Flujo Básico

1. El usuario requiere establecer una conexión con el DBMS.

2. El sistema muestra una pantalla con las siguientes opciones.
   - Dirección IP del DBMS.
   - Usuario.
   - Contraseña.
   - Seleccionar el DBMS (De un listado de DBMS compatibles con el sistema)
3. El sistema muestra un mensaje de confirmación en caso de que la conexión sea exitosa.
4. El usuario confirma la utilización de esta conexión.
5. El sistema retorna el control al caso de uso que le llamo.

Especificaciones suplementarias

1. En caso de no poder establecer conexión con el DBMS, el sistema mostrará un mensaje informativo.

***Caso de Uso: Escribir Archivos Estándar.***

Flujo Básico

1. El usuario define la estructura de directorios mediante una pantalla de configuración.
2. El sistema escribe los archivos con el código generado y lo ubica según la especificación de directorios y nombre de archivos definido por el usuario.
3. El sistema despliega un mensaje confirmando la generación exitosa del archivo.

Especificaciones suplementarias

1. En caso de no poder escribir los archivos el sistema mostrara un mensaje de error.

***Casos de Uso: Generar Archivo Deploy.***

Flujo Básico

1. El caso de uso "Escribir Archivos Estándar" sede el control a este caso de uso.
2. El sistema genera un archivo que permitirá hacer un deploy de la aplicación. Dicho archivo contiene información de referencia acerca de los nombres y ubicaciones del código generado para cada capa del modelo.
3. El sistema retorna el flujo al caso de uso "Escribir Archivos Estándar"

Especificaciones suplementarias

1. En caso de no poder escribir los archivos el sistema, mostrará un mensaje de error.

### 2.2.2 DIAGRAMA DE CLASES

El diagrama de clases definitivo de la aplicación formara parte de los anexos
La Figura 2.5 muestra el Diagrama de Clases.

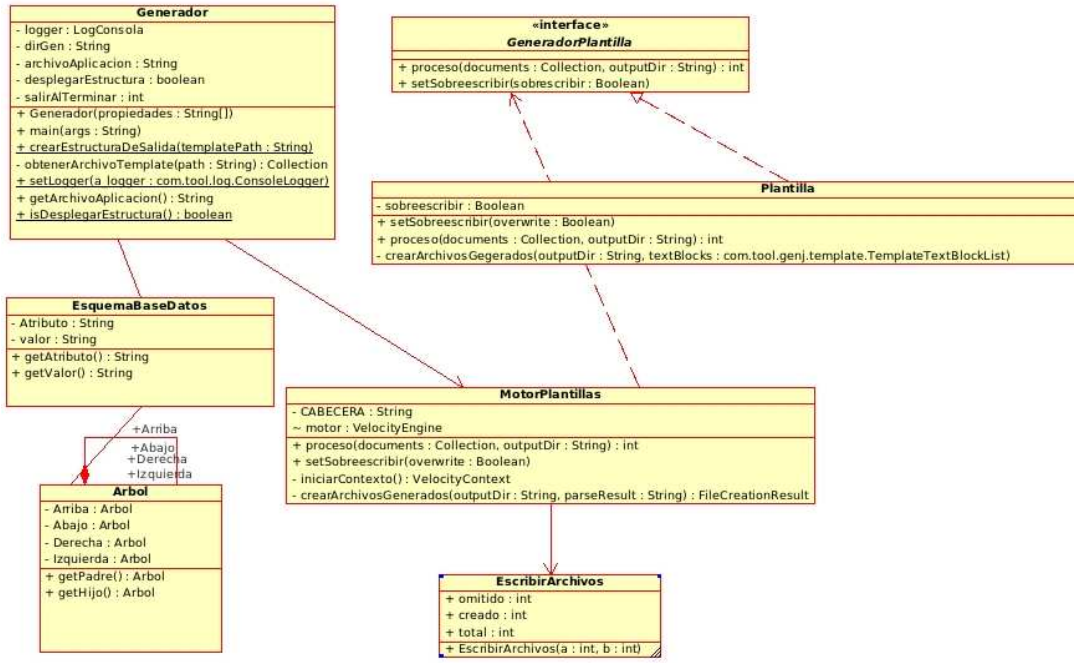



**Figura 2.4 Diagrama de Clases**

- **Esquema de Base de datos.-** Encargado de permitir la conexión con un dbms además de permitir obtener la metadata.
- **Generador.** Por medio de la metadata obtenida por la clase anterior y aplicada a una plantilla permitirá generar el código de cada capa.
- **Plantilla.** Contiene las distintas plantillas que permiten generar cada capa para el modelo MVC.
- **Motor de Plantillas.** Permite la administración de las plantillas.
- **Escribir Archivos.** Guarda los archivos de código que son generados por la herramienta para su posterior edición y uso.

### 2.2.3 DIAGRAMA DE ACTIVIDAD

Este diagrama muestra las actividades que se realiza la herramienta en el proceso de generación de código:

1. Ingresar información de la aplicación:
   El usuario registra información general de la aplicación que va a generar.
2. Establecer conexión con DBMS:
   La herramienta se conecta con motor de base de datos para obtener el un esquema que será utilizado para la posterior generación de código.
3. Obtener Meta Datos:
   Luego de que la conexión con la base de datos se ha establecido se procede a extraer los metadatos que proporcionan información sobre las tablas y los campos de las mismas.
4. Seleccionar Tablas:
   El usuario selecciona las tablas para los cuales generará el código.
5. Generar Código.
   Basados en las tablas seleccionadas se genera el código de las distintas capas de la arquitectura MVC, y adicional un archivo de deployment de la aplicación generada
6. Escribir Archivo:
   Es la actividad final la escritura en disco de la aplicación generada.

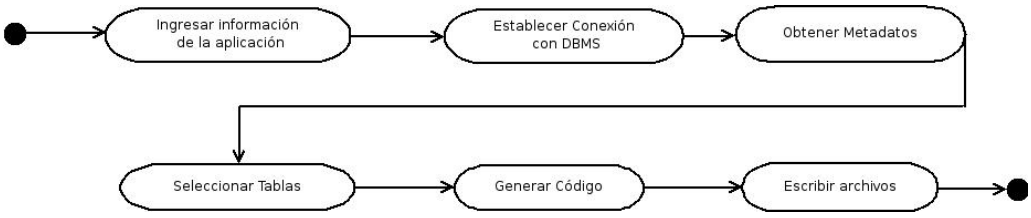



**Figura 2.5 Diagrama de Casos de Actividad.**

En la Figura 2.6 podemos ver el detalle lo que ocurre en “Ingresar Información de la aplicación” donde se registran los datos básicos que permiten la generación de la aplicación.

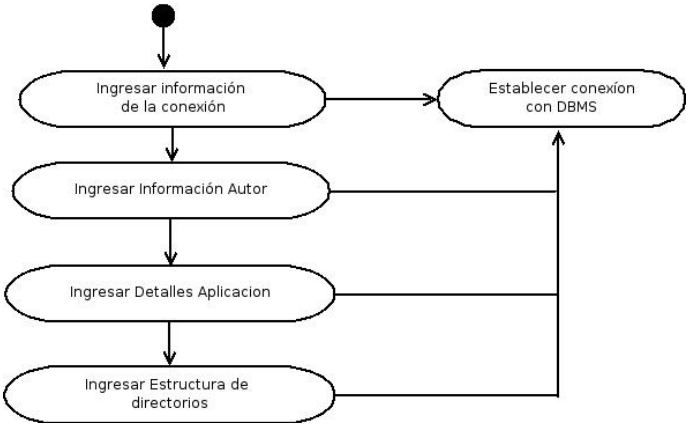



Figura 2.6 Detalle “Ingresar Información de la aplicación”

En la Figura 2.7 podemos observar el detalle de “Establecer conexión con dbms” Aquí el sistema intenta establecer una conexión con un dbms en caso de ser exitosa la conexión permite seleccionar un esquema de base de datos, y luego las tablas que pertenecen a este esquema.

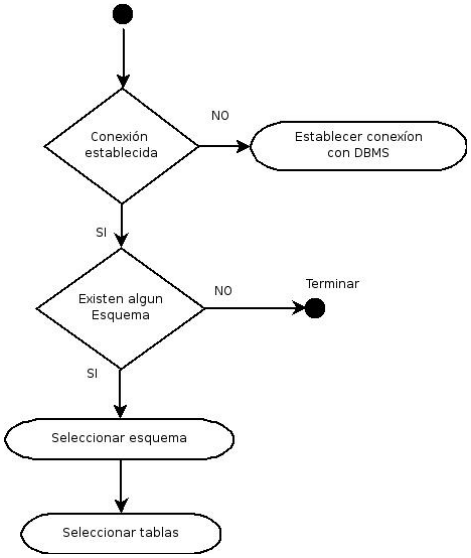



Figura 2.7 Detalle “Establecer conexión con dbms”

En la figura 2.8 podemos ver el detalle de “Obtener metadatos” donde se conecta a la base de datos para obtener sus metadatos y en caso de no poder realizar esta operación regresa el flujo a “Ingresar Información de la aplicación”.

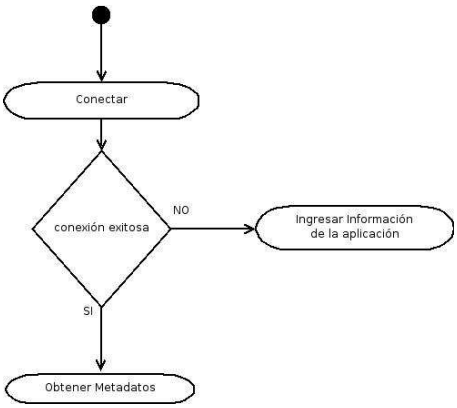



Figura 2.8 Detalle “Obtener Meta Datos”

En la figura 2.9 podemos visualizar el detalle de “Generar código” el mismo que representa el proceso medular de nuestra herramienta. Primero se mapea una tabla a un entity bean; a este se asocia un sesión bean, se verifica que estos pasaos sean correctos caso contrario se regresa al primer paso. A continuación se genera un archivo xml que se aplica a una plantilla a fin de generar el código y finalmente se guarda el archivo.

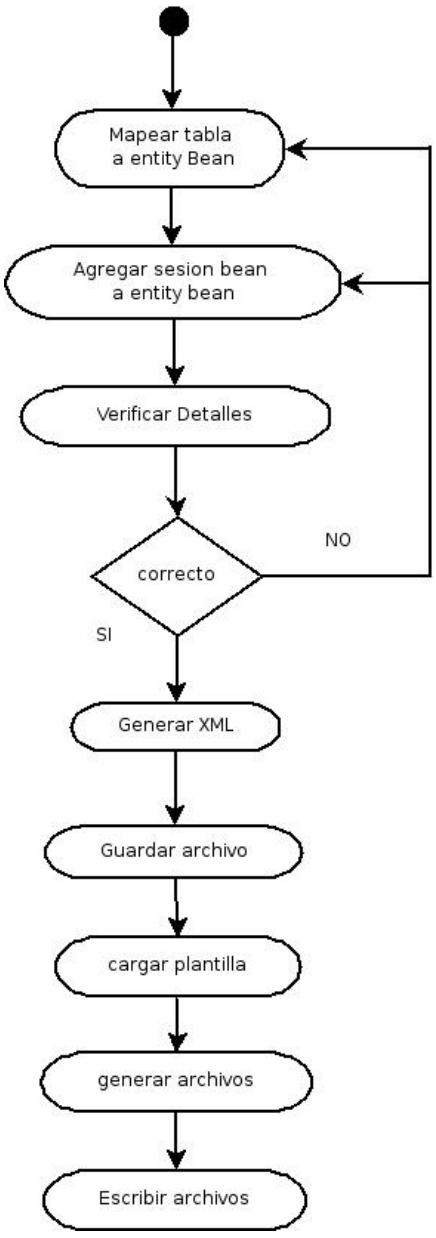



Figura 2.9 Detalle “Generar Código”

# CAPITULO 3. INPLEMENTACIÓN

Una vez concluida la etapa de Elaboración, iniciamos la etapa de ***Construcción*** donde se realizara la codificación del proyecto basado en los diseños obtenidos en la anterior etapa; la codificación se realizara en lenguaje java, y se utilizara la herramienta Eclipse debido a que es una herramienta gratuita que brida las facilidades necesarias para la construcción de la herramienta

## 3.1 IMPLEMENTACIÓN

El nombre escogido para la herramienta fue **genj** "generador de código para aplicaciones java", a partir de ahora nos vamos a referir a la herramienta como **genj.**

La implementación de **genj** empezó por definir los detalles del documento xml que contiene los parámetros de conexión a la base de datos; la estructura es la siguiente:


**Figura 3.1 Documento xml con parámetros de conexión al dbms.**

La raíz del documento xml se llama "*conf-genj"*. Contiene los siguientes elementos.

- **Bases-soportadas**. Contiene la descripción de las bases de datos soportadas por la herramienta.
- **Nombre**. Nombre de la base de datos que se muestra al usuario.
- **Driver-class**. Clase del driver de conexión.
- **Appserver**. Nombre del servidor donde esta instalada la base de datos
- **File**. Ruta física que indica la localización del driver del dbms.

A continuación un ejemplo del archivo xml, el cuál fue creado para conectarse a las bases de datos: “Oracle” y “mysql”.

```
<?xml version="1.0" encoding="utf-8"?>
<conf-genj>
        <bases-soportadas>
                <basededatos>
                        <nombre>Oracle</nombre>
                        <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
                        <appserver>Oracle</appserver>
                        <file>lib/ojdbc14.jar</file>
                </basededatos>
                <basededatos>
                        <nombre>MySQL</nombre>
                        <driver-class>com.mysql.jdbc.Driver</driver-class>
                        <appserver>mySQL</appserver>
                        <file>lib/mysql-connectorJ-309.jar</file>
                </basededatos>
        </bases-soportadas>
</conf-genj>
```

**Figura 3.2 Ejemplo Archivo configuración base de datos.**

**Estructura de Directorios:**

Representa la estructura de directorios donde son almacenados los diferentes archivos que requiere la herramienta para su funcionamiento.

- **src.** Directorio de código fuentes.
- **bin**. Directorio clases compiladas.
- **Conf.** Contiene archivos de configuración.
- **Lib.** Directorio de librerías.
- **Plantillas:** Almacena plantillas para la generación de código.

Dentro del directorio “SRC” que contiene el código fuente de la aplicación:

- **com.tool.genj**. Paquete raíz.
- **Exception**. Directorio con todas las clases de las excepciones que se manejaran en genj.
- **Gui.** Directorio con las clases de las interfaces gráficas.
- **Módulos**. Directorio con los diferentes módulos que conforman genj.
- **Motor**. Directorio con clases que conforman genj.
- **Plantillas**. Directorio con clases que manejan las plantillas de generación.
- **Util**. Directorio con librerías propias y de terceros que son utilizadas a lo largo de la aplicación.

Pantalla principal de genj. desde la cuál se puede acceder a las principales opciones del sistema ya sea desde el menú o desde los botones de acceso rápido.

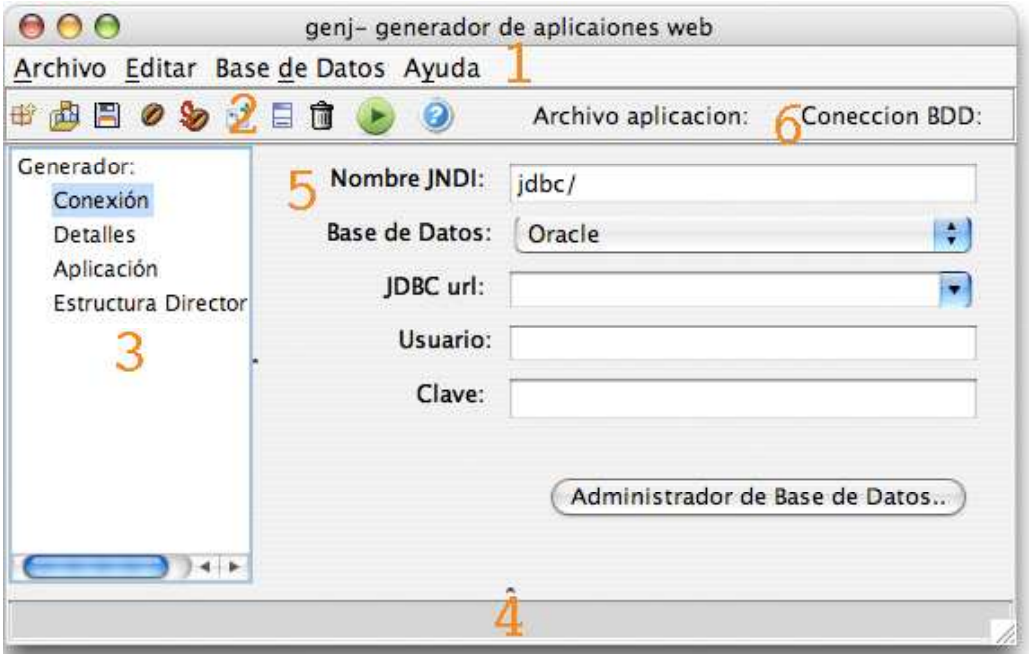



**Figura 3.3 Interfaz gráfica genj.**

**1 Barra de Menú.**

**Archivo.**

Contiene los sub. menús para crear modificar y generar aplicaciones:

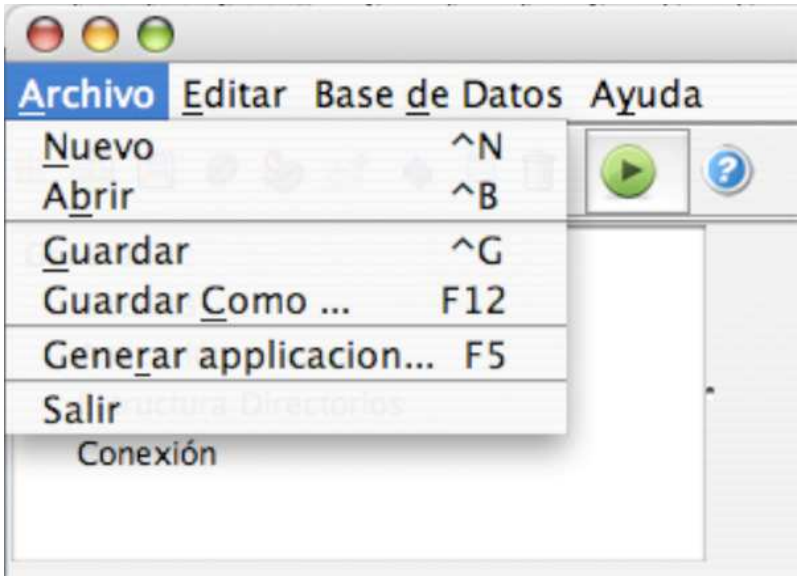



**Figura 3.4 Opción Archivo Barra Menú.**

- **Nuevo.** Permite crear nuevas aplicaciones.

- **Abrir.** Abrir una aplicación existente.
- **Guardar.** Guardar la aplicación.
- **Guardar Como:** Guardar con un nombre diferente la aplicación.
- **Generar Aplicación.** Genera el código fuente de la aplicación.
- **Salir.** Cerrar la herramienta.

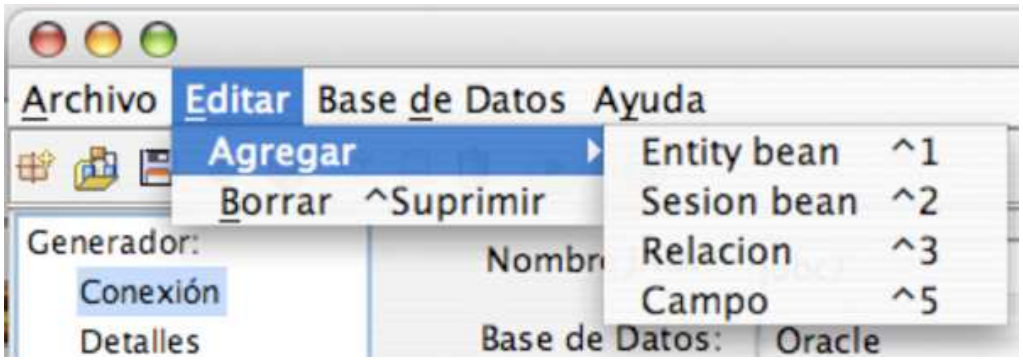



**Figura 3.5 Opción Editar.**

**Agregar.**

- **Entity Bean**. Esta relacionado a una tabla de la base de datos.
- **Session Bean**. Relacionado a un entity bean de la aplicación.
- **Relación**. una relación entre entidades.
- **Campo**. Son los campos que pertenecen a una entidad, cada campo posee propiedades como el tipo de dato, el tamaño, además permite incluir código xml a fin de validar el ingreso de datos.

**Borrar.**

Permite borra un elemento previamente seleccionado, entity, session, campo o método.

**Base de datos.**

Contiene los elementos que permiten establecer la conexión con la base de datos.


**Figura 3.6 Opciones de Base de Datos.**

- **Administrar Conexión.** Muestra un dialogo con la información de los drivers disponibles para las conexiones a las bases de datos.
- **Crear Conexión.** Establece la conexión con una base de datos específica.
- **Desconectar.** Desconecta una conexión con una base de datos.

**Ayuda.**


**Figura 3.7 Ayuda**

- **Acerca.** Muestra información acerca de genj.
- **Contenido.** Contenido de la ayuda.

**2 Barra de Herramientas**

La barra de herramientas proporciona acceso directo a las diferentes acciones de genj, que también están disponibles en la barra de Menú. Describiremos su función según La Figura 3.8 de izquierda a derecha

**Figura 3.8 Barra de herramientas.**

- Nueva aplicación.
- Abrir una aplicación existente.
- Guardar la aplicación.
- Agregar un entity bean.
- Agregar un session bean.
- Agregar una relación.
- Agregar un campo
- Borrar
- Generar la aplicación.
- Ayuda.

**3. Nombres de los detalles de la aplicación**

Aquí están los detalles de la aplicación a ser generada:

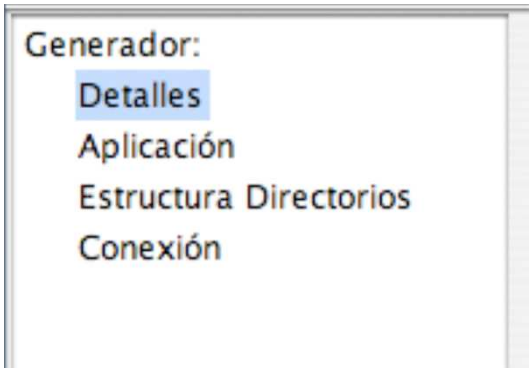



**Figura 3.9 Detalles de la aplicación.**

**4. Log de Genj**

Registra acciones que ejecuta la herramienta a fin de poder determinar su estado o corregir problemas que pudieran presentarse en la generación de código.

**5. Descripción de los detalles de la aplicación**

Aquí se despliegan una descripción de los detalles de la aplicación.

**6. Barra de estado**

Indica los diferentes estados de genj.

**Proceso de Codificación:**

En esta sección se muestra detalles de la codificación e implementación de genj en java. Las secciones de código fuente son las principales, todo el código fuente estará como anexo.

La primera acción que realiza genj es la conexión a la base de datos, con la información de las bases de datos definidas en el archivo xml, anteriormente descrito:

```
/**
* Lee la información del archivo de configuración
*/
private void leer() {
bases.clear();
Document doc = ManejadorDetalles.getInstance().getDocument();
                        NodeList                databaseNodes                =
doc.getElementsByTagName(BASE_DE_DATOS);

for (int i = 0; i < databaseNodes.getLength(); i++) {
        BaseDeDatos dbInfo = new BaseDeDatos();
        Element database = (Element) databaseNodes.item(i);
        NodeList children = database.getChildNodes();
        for (int j = 0; j < children.getLength(); j++) {
          if (children.item(j) instanceof Element) {
                Element child = (Element) children.item(j);
                if (NOMBRE.equals(child.getNodeName())) {

        dbInfo.setNombreBd(child.getFirstChild().getNodeValue());
     } else if (DRIVER_CLASS.equals(child.getNodeName())) {
        dbInfo.setDriverClass(child.getFirstChild().getNodeValue());
    } else if (APPSERVER.equals(child.getNodeName())) {
        dbInfo.setTypeMapping(child.getFirstChild().getNodeValue());
        } else if (ARCHIVO.equals(child.getNodeName())) {
        dbInfo.setNombreArchivo(child.getFirstChild().getNodeValue());
                        }
                }
        }

        File driver = new File(dbInfo.getNombreArchivo());
```

```
if (!driver.exists()) {
GuiGenj.logConsola("Quitando referencia al driver: "
                                        + dbInfo.getNombreArchivo());

JOptionPane.showMessageDialog(GuiGenj.guiGenj, "El driver '"
+ dbInfo.getNombreBd() + "' no pudo ser localizado.\n"
+ "(Ubicacion: " + dbInfo.getNombreArchivo() + ")\n"
+ "El archivo a sido borrado.", "Driver no encontrado!",
javax.swing.JOptionPane.ERROR_MESSAGE);
} else {
        bases.add(dbInfo);
                }
        }
Map                     typeMappingsMap                     =
ManejadorDetalles.getInstance().obtenerPropiedadesXML(APPSERVER
S);
        String[] temp = (String[]) typeMappingsMap.get(NOMBRE);
                if (temp != null) {
                        tipoMapeos = temp;
                }
        }
```

Para establecer la conexión hay que especificar a que base nos vamos a conectar, la url de conexión el usuario y la clave.

En caso de que la base de datos a la que queremos conectarnos no esté soportada podemos agregar un nuevo driver de conexión especificando la ubicación y los detalles del nuevo driver:

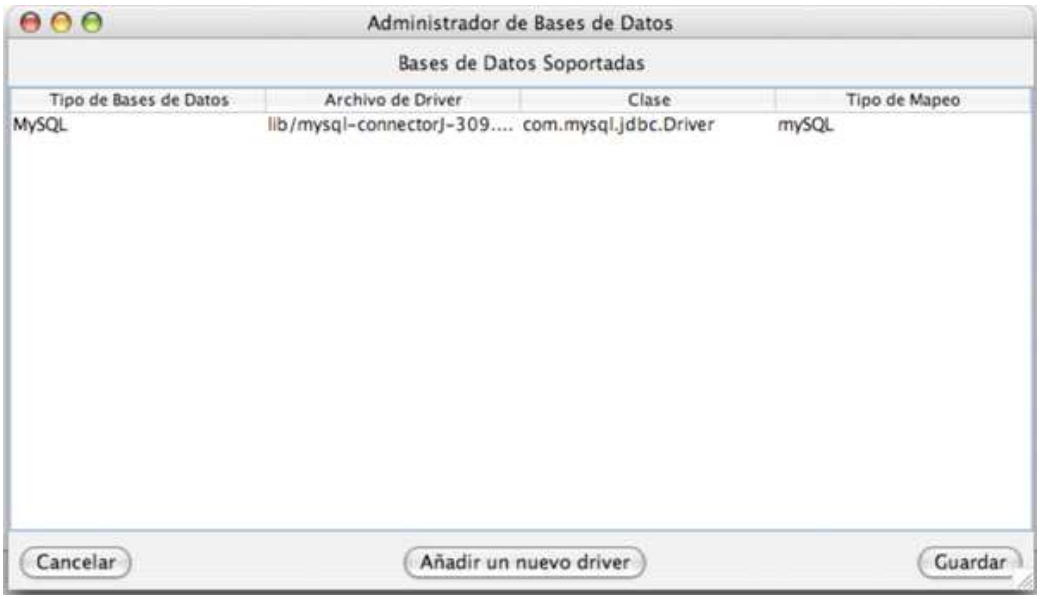



**Figura 3.10 Establecer conexión con la Base de Datos.**

Al añadir un nuevo driver hay que seleccionar el archivo que contenga este driver, deberá ser un archivo de extensión jar, o zip o class:


**Figura 3.11 Seleccionar un driver de base de datos.**

Luego de tener listos los parámetros hay que establecer la conexión indicando si queremos visualizar, tablas, vistas y sinónimos:

Para establecer la conexión utilizamos el siguiente método:

```
public ConexionBaseDatos(String url, String usuario, String clave,
                String clazz, String[] displayTypes) {
        this.url = url;
        this.usuario = usuario;
        this.clave = clave;
        this.clazz = clazz;
        this.displayTypes = displayTypes;
        int dbIndex = url.lastIndexOf("/");
        if (dbIndex != -1) {
                db = url.substring(dbIndex + 1);
        }
        ArrayList<String> allesquemas = new ArrayList<String>();
        try {
                Connection cx = connect();
                ResultSet esquemas = cx.getMetaData().getSchemas();
                while (esquemas.next()) {
                        String s =
esquemas.getString(esquema_NAME_COLUMN);
                                if (usuario.equals(s)) {
                                esquema = usuario;
                                break;
                        }
                        allesquemas.add(s);
                }
        if ((esquema == null) && (allesquemas.size() != 0)) {
            esquema = (String) JOptionPane.showInputDialog(
                GuiGenj.guiGenj,
```

```
                    "No existe un esquema con el nombre \"" + usuario + "\" en
esta base de datos!\n\n" +
                    "Seleccione un esquema de la lista, \n" ,
                    "Esquemas de Bases de Datos",
                    JOptionPane.QUESTION_MESSAGE,
                    null,
                    allesquemas.toArray(),
                    null);
                }
                GuiGenj.logConsola("Usando el esquema: " + esquemas);
        } catch (Exception e) {
                        e.printStackTrace();
                 }
}
```

En caso que el nombre de usuario no coincida con algún esquema de la base de datos, se solicita que se seleccione algún esquema disponible.

Se debe ingresar detalles de la aplicación como el autor, la versión, Nombre de la aplicación, descripción. Finalmente debemos especificar la estructura de directorios donde se van a generar las clases:

Luego de tener listos los parámetros generales de la aplicación empezamos a añadir las entidades que necesitamos, seleccionamos el icono de entitys y se despliega la siguiente pantalla:


**Figura 3.12 Pantalla para selección de entidades.**

Ahora es necesario definir las propiedades particulares de cada campo entidad.

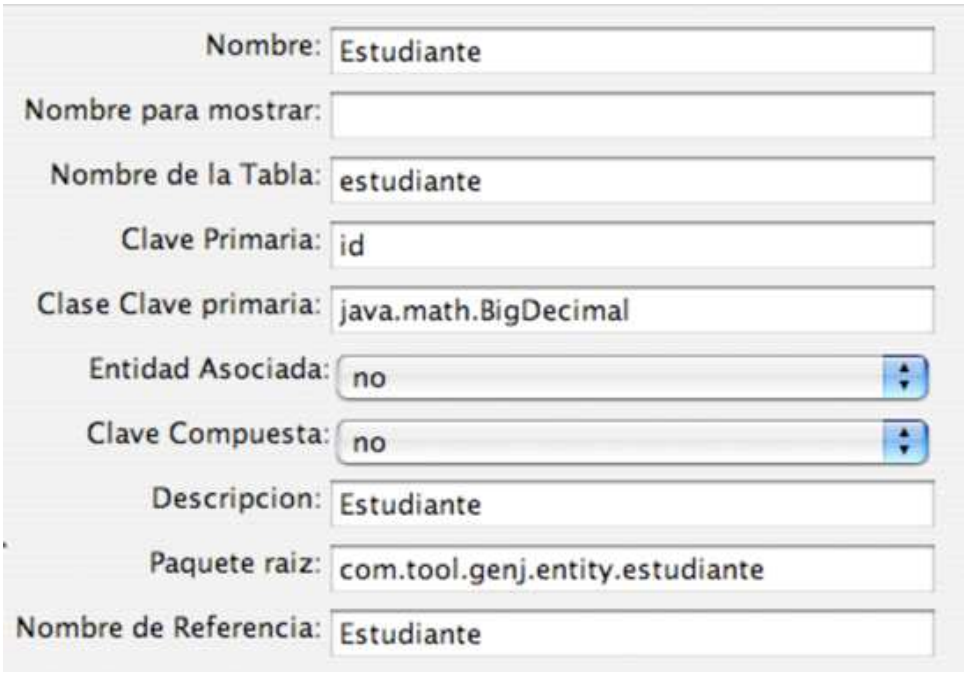



**Figura 3.13 Propiedades de una entidad.**

Los detalles de las entidades están representados en la clase Entidad en el método:

```
public void getXML(Element el) throws ParserConfigurationException {
    Document doc = el.getOwnerDocument();
    Element module = doc.createElement("module");
    module.setAttribute("name", "entity");
```

```
        Element nombre = doc.createElement("module-data");
        nombre.setAttribute("name", "name");
        if (txtNombre.getText() != null) {
            nombre.appendChild(doc.createTextNode(txtNombre.getText()));
        }
        module.appendChild(nombre);

        Element displaynombre = doc.createElement("module-data");
        displaynombre.setAttribute("name",  "display-name");
        if (txtNombreParaGui.getText() != null) {

displaynombre.appendChild(doc.createTextNode(txtNombreParaGui.getText()))
;
        }
        module.appendChild(displaynombre);


        Element descripcion = doc.createElement("module-data");
        descripcion.setAttribute("name","description");
        if (txtDescripcion.getText() != null) {

descripcion.appendChild(doc.createTextNode(txtDescripcion.getText()));
        }
        module.appendChild(descripcion);

        Element paqueteRaiz = doc.createElement("module-data");
        paqueteRaiz.setAttribute("name", "root-package");
        if (txtPauqueteRaiz.getText() != null) {

paqueteRaiz.appendChild(doc.createTextNode(txtPauqueteRaiz.getText()));
        }
        module.appendChild(paqueteRaiz);

        Element nombreTabla = doc.createElement("module-data");
        nombreTabla.setAttribute("name", "table-name");
        if (txtNombreTabla.getText() != null) {

nombreTabla.appendChild(doc.createTextNode(txtNombreTabla.getText()));
        }
        module.appendChild(nombreTabla);

        Element pKey = doc.createElement("module-data");
        pKey.setAttribute("name", "primary-key");
        pKey.appendChild(doc.createTextNode((esCompuesta()       ?       ""       :
txtPkey.getText())));
        module.appendChild(pKey);

        Element pKeyType = doc.createElement("module-data");
        pKeyType.setAttribute("name", "primary-key-type");
```

```
        if (txtTipoPkey.getText() != null) {
            pKeyType.appendChild(doc.createTextNode(txtTipoPkey.getText()));
        }

        module.appendChild(pKeyType);
        Element isComposite = doc.createElement("module-data");
        isComposite.setAttribute("name", "is-composite");
        isComposite.appendChild(doc.createTextNode((String)
cmBxEsComboCompuesto.getSelectedItem()));
        module.appendChild(isComposite);


        Element isAssociation = doc.createElement("module-data");
        isAssociation.setAttribute("name", "is-association");
        isAssociation.appendChild(doc.createTextNode((String)
cmBxEsEntidadAsociaciada.getSelectedItem()));
        module.appendChild(isAssociation);

        Element refName = doc.createElement("ref-name");
        if (txtNombreReferencia.getText() != null) {

refName.appendChild(doc.createTextNode(txtNombreReferencia.getText()));
        }
        module.appendChild(refName);

        Enumeration children = children();
        while (children.hasMoreElements()) {
            Bean hijo = (Bean) children.nextElement();
            hijo.getXML(module);
        }

        el.appendChild(module);
    }
```

Luego agregamos un session Bean. Y lo asociamos a su correspondiente entity Bean; en la figura podemos ver que se ha creado el session bean

Estudiante al cual se lo relaciona con el entity Bean “Estudiante”.

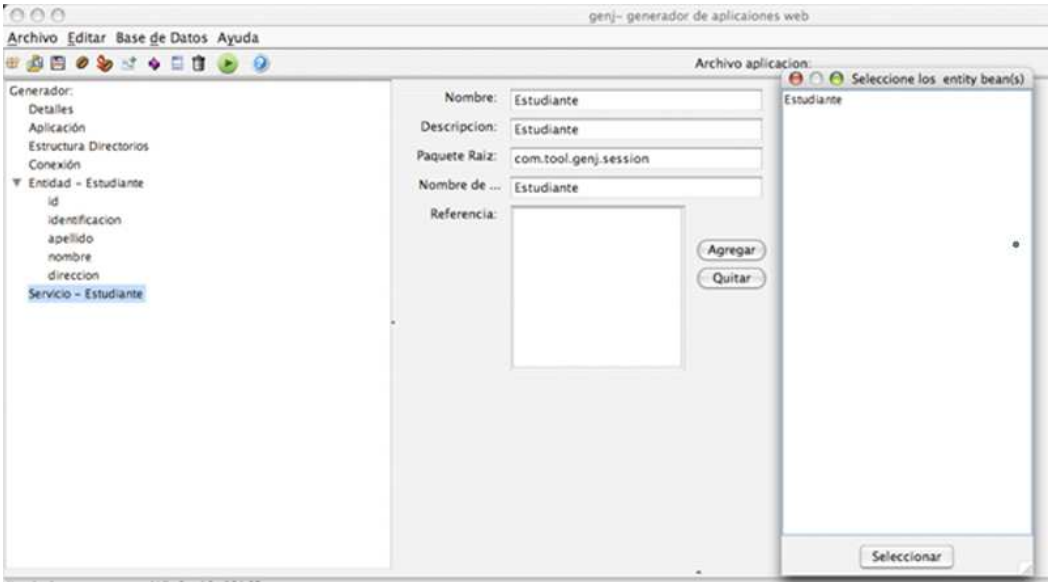



**Figura 3.14 Selección de Entity Bean.**

Este proceso de agregar entity beans y sessions beans se repite según la necesidad de la aplicación que se va a generar.

Básicamente todas estas pantallas de ingreso y manipulación de información tienen la misma estructura. Para obtener los datos ingresados en las pantallas en formato xml utilizamos el siguiente método:

```
public void getXML(Element elemento) throws ParserConfigurationException
{

        Document doc = elemento.getOwnerDocument();
        Element module = doc.createElement("module");
        module.setAttribute("name", "datasource");

        Element jndi = doc.createElement("module-data");
        jndi.setAttribute("name", "jndi-name");
```

```
            if (jndiText.getText() != null) {
              jndi.appendChild(doc.createTextNode(jndiText.getText()));
            }
            module.appendChild(jndi);

            Element mapping = doc.createElement("module-data");
            mapping.setAttribute("name", "mapping");


mapping.appendChild(doc.createTextNode(mappingCombo.getModel().getSele
ctedItem().toString()));
            module.appendChild(mapping);

            Element jdbcUrl = doc.createElement("module-data");
            jdbcUrl.setAttribute("name", "jdbc-url");

jdbcUrl.appendChild(doc.createTextNode(jdbcURLCombo.getEditor().getItem().
toString()));
            module.appendChild(jdbcUrl);

            Element userName = doc.createElement("module-data");
            userName.setAttribute("name", "user-name");
            if (userNameText.getText() != null) {

userName.appendChild(doc.createTextNode(userNameText.getText()));
            }
            module.appendChild(userName);

            Element password = doc.createElement("module-data");
            password.setAttribute("name", "password");
            if (passwordText.getText() != null) {

password.appendChild(doc.createTextNode(passwordText.getText()));
```

```
            }
            module.appendChild(password);

            elemento.appendChild(module);
        }
```

El método getXML(Element elemento) es una implementación de la interface Bean:

```
public interface Bean {

        public .JPanel getPanel();
        public String getNombreReferencia();
        public void getXML(Element el) throws
ParserConfigurationException;

}
```

Esta interfase tiene tres métodos:

***public .JPanel getPanel();***

Con este método se obtiene la interfaz gráfica que contiene los diferentes detalles de la aplicación, además es usada por la GUI principal para cambiar y mostrar en la misma ventana diferentes módulos, como Detalles, Aplicación, Estructura de Directorios, Entidades.

***public String getNombreReferencia();***

El nombre de referencia es el que se muestra en la ventana principal de la interfaz gráfica.

***public void getXML(Element el)***

En este método se debe implementar la obtención de la información de las interfaces gráficas y transformarlos a xml, para su posterior manipulación.

La interfaz será implementada por las siguientes clases:

- com.tool.genj.modulos.Aplicacion.java
- com.tool.genj.modulos.Bean.java
- com.tool.genj.modulos.Campo.java
- com.tool.genj.modulos.ConexionUIFrame.java
- com.tool.genj.modulos.Datasource.java
- com.tool.genj.modulos.Detalles.java
- com.tool.genj.modulos.Entidad.java
- com.tool.genj.modulos.EstructuraDirectorios.java
- com.tool.genj.modulos.MetodoNegocio.java
- com.tool.genj.modulos.PanelRelacion.java
- com.tool.genj.modulos.ParametrosNegocio.java
- com.tool.genj.modulos.Raiz.java
- com.tool.genj.modulos.Relacion.java
- com.tool.genj.modulos.Sesion.java

Todas las clases mencionadas anteriormente se utilizan para el manejo de la información de la aplicación como: detalles de la aplicación, autor, versión, conexión, estructura de directorios donde se va a generar la aplicación, detalles de los entity beans, sessions beans.

Con el método getXML obtendremos toda esa información para posteriormente guardarla en un archivo con los detalles completos de nuestra aplicación.

En este momento tenemos los detalles de nuestra aplicación completos y las entidades y objetos de sesión creados. Para continuar con el proceso hay que guardar la aplicación, y generando un archivo xml con esta estructura:

Este es un ejemplo del archivo generado; contiene todos los detalles de la aplicación:

```
<?xml version="1.0" encoding="utf-8"?>
<genj-detalles>
	<config>
		<author/>
		<version>1.0</version>
		<templates>
			<template-root>templates/genj</template-root>
			<config-param name="appserver" value="JBoss 4.x"/>
			<config-param name="businessTier" value="EJB 3.0"/>
			<config-param name="useRelations" value="false"/>
			<config-param name="webTier" value="Struts 1.2"/>
			<config-param name="serviceTier"
value="ServiceLocator"/>
		</templates>
	</config>
	<module name="app">
		<module-data name="name"/>
		<module-data name="version">1.0</module-data>
		<module-data name="description"/>
		<module-data name="root-package">com.tool.genj</module-
data>
	</module>
	<module name="paths">
		<module-data name="service_output">./src/java-
service/</module-data>
		<module-data name="ejb_output">./src/java-ejb/</module-data>
		<module-data name="web_output">./src/java-web/</module-data>
		<module-data name="test_output">./src/java-test/</module-data>
		<module-data name="jsp_output">./src/web/</module-data>
		<module-data name="config_output">./conf/</module-data>
	</module>
	<module name="datasource">
		<module-data name="jndi-name">jdbc/ejb</module-data>
```

```
            <module-data name="mapping">Oracle</module-data>
            <module-data name="jdbc-
url">jdbc:oracle:thin:@10.2.0.53:1521:BDESA</module-data>
            <module-data name="user-name">cruces</module-data>
            <module-data name="password">cruces</module-data>
        </module>
        <module name="entity">
            <module-data name="name">CruParametros</module-data>
            <module-data name="display-name"/>
            <module-data name="description">CruParametros</module-data>
            <module-data name="root-
package">com.tool.genj.entity.cruparametros</module-data>
            <module-data name="table-
name">CRU_PARAMETROS</module-data>
            <module-data name="primary-key"/>
            <module-data name="primary-key-
type">com.tool.genj.entity.cruparametros.CruParametrosPK</module-data>
            <module-data name="is-composite">si</module-data>
            <module-data name="is-association">no</module-data>
            <ref-name>CruParametros</ref-name>
            <module-data name="field">
                <module-data name="name">codigoParametro</module-
data>
                <module-data name="type">java.lang.String</module-
data>
                <module-data name="column-
name">CODIGO_PARAMETRO</module-data>
                <module-data name="required">true</module-data>
                <module-data name="sql-type">VARCHAR2(15)</module-
data>
                <module-data name="jdbc-type">VARCHAR</module-
data>
                <module-data name="primary-key">true</module-data>
```

```
                    <module-data        name="auto-primary-key">true</module-
data>
                    <module-data name="foreign-key">false</module-data>
                    <module-data                                  name="validation-
depends">maxlength</module-data>
                    <module-data                  name="validation-xml">&lt;arg1
key=&quot;${var:maxlength}&quot;                  nombre=&quot;maxlength&quot;
resource=&quot;false&quot;/&gt;
&lt;var&gt;
  &lt;var-nombre&gt;maxlength&lt;/var-nombre&gt;
  &lt;var-valor&gt;15&lt;/var-valor&gt;
&lt;/var&gt;</module-data>
                </module-data>
                <module-data name="field">
                    <module-data        name="name">anioParametro</module-
data>
                    <module-data        name="type">java.lang.Integer</module-
data>
                    <module-data                                    name="column-
name">ANIO_PARAMETRO</module-data>
                    <module-data name="required">true</module-data>
                    <module-data        name="sql-type">NUMBER(4)</module-
data>
                    <module-data name="jdbc-type">INTEGER</module-data>
                    <module-data name="primary-key">true</module-data>
                    <module-data        name="auto-primary-key">true</module-
data>
                    <module-data name="foreign-key">false</module-data>
                    <module-data                                  name="validation-
depends">integer</module-data>
                    <module-data name="validation-xml"/>
                </module-data>
                <module-data name="field">
```

<module-data name="name">descripcion</module-data>
<module-data name="type">java.lang.String</module-data>
<module-data name="column-name">DESCRIPCION</module-data>
<module-data name="required">true</module-data>
<module-data name="sql-type">VARCHAR2(100)</module-data>
<module-data name="jdbc-type">VARCHAR</module-data>
<module-data name="primary-key">false</module-data>
<module-data name="auto-primary-key">false</module-data>
<module-data name="foreign-key">false</module-data>
<module-data name="validation-depends">requerido, maxlength</module-data>
<module-data name="validation-xml"><arg1 key="${var:maxlength}" nombre="maxlength" resource="false"/>
<var>
<var-nombre>maxlength</var-nombre>
<var-valor>100</var-valor>
</var></module-data>
</module-data>
<module-data name="field">
<module-data name="name">tipoParametro</module-data>
<module-data name="type">java.lang.String</module-data>
<module-data name="column-name">TIPO_PARAMETRO</module-data>
<module-data name="required">true</module-data>

<module-data name="sql-type">VARCHAR2(3)</module-data>
<module-data name="jdbc-type">VARCHAR</module-data>
<module-data name="primary-key">false</module-data>
<module-data name="auto-primary-key">false</module-data>
<module-data name="foreign-key">false</module-data>
<module-data name="validation-depends">requerido, maxlength</module-data>
<module-data name="validation-xml"><arg1 key="${var:maxlength}" nombre="maxlength" resource="false"/>
<var>
<var-nombre>maxlength</var-nombre>
<var-valor>3</var-valor>
</var></module-data>
</module-data>
<module-data name="field">
<module-data name="name">valorFecha</module-data>
<module-data name="type">java.sql.Date</module-data>
<module-data name="column-name">VALOR_FECHA</module-data>
<module-data name="required">false</module-data>
<module-data name="sql-type">DATE</module-data>
<module-data name="jdbc-type">DATE</module-data>
<module-data name="primary-key">false</module-data>
<module-data name="auto-primary-key">false</module-data>
<module-data name="foreign-key">false</module-data>
<module-data name="validation-depends">date</module-data>
<module-data name="validation-xml"><var>

<var-nombre>datePattern</var-nombre>
<var-valor>${date_format}</var-valor>
</var></module-data>
</module-data>
<module-data name="field">
<module-data name="name">valorNumerico</module-data>
<module-data name="type">java.lang.Double</module-data>
<module-data name="column-name">VALOR_NUMERICO</module-data>
<module-data name="required">false</module-data>
<module-data name="sql-type">NUMBER(15, 2)</module-data>
<module-data name="jdbc-type">DOUBLE</module-data>
<module-data name="primary-key">false</module-data>
<module-data name="auto-primary-key">false</module-data>
<module-data name="foreign-key">false</module-data>
<module-data name="validation-depends">float, decimal</module-data>
<module-data name="validation-xml"><arg1 key="${var:decimalPrecision}" nombre="decimal" resource="false"/><arg2 key="${var:decimalLength}" nombre="decimal" resource="false"/><var>
<var-nombre>decimalLength</var-nombre>
<var-valor>15</var-valor>
</var><var>
<var-nombre>decimalPrecision</var-nombre>
<var-valor>2</var-valor>
</var></module-data>
</module-data>

```
            <module-data name="field">
                <module-data name="name">valorVarchar</module-data>
                <module-data name="type">java.lang.String</module-
data>
                <module-data name="column-
name">VALOR_VARCHAR</module-data>
                <module-data name="required">false</module-data>
                <module-data name="sql-type">VARCHAR2(80)</module-
data>
                <module-data name="jdbc-type">VARCHAR</module-
data>
                <module-data name="primary-key">false</module-data>
                <module-data name="auto-primary-key">false</module-
data>
                <module-data name="foreign-key">false</module-data>
                <module-data name="validation-
depends">maxlength</module-data>
                <module-data name="validation-xml">&lt;arg1
key=&quot;${var:maxlength}&quot; nombre=&quot;maxlength&quot;
resource=&quot;false&quot;/&gt;
&lt;var&gt;
  &lt;var-nombre&gt;maxlength&lt;/var-nombre&gt;
  &lt;var-valor&gt;80&lt;/var-valor&gt;
&lt;/var&gt;</module-data>
            </module-data>
        </module>
        <module name="session">
            <module-data name="name">SessionPrueba</module-data>
            <module-data name="description">SessionPrueba</module-data>
            <module-data name="root-
package">com.tool.genj.session</module-data>
            <ref-name>SessionPrueba</ref-name>
        </module>
```

</genj-detalles>

El método que se usa para la generación y escritura del archivo es el siguiente:

```
public boolean guardar() {
        actualizarRelaciones();
        actualizarRelacionesForaneas();
        if (!agregarEntidadesASessionBeans()) {
                return false;
        }
        try {
                DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
                DocumentBuilder builder = dbf.newDocumentBuilder();
                Document doc = builder.newDocument();
                Element detalles = doc.createElement("genj-detalles");
                doc.appendChild(detalles);
                raiz.getXML(detalles);
                String XMLDoc = outXML(doc);
                String nombreArchivo = archivo.getName();
                if (nombreArchivo != null) {
                        if (nombreArchivo.indexOf(".xml") == -1) {
                                archivo = new File(archivo.getAbsolutePath() + ".xml");
                        }
                }
                FileWriter fw = new FileWriter(archivo);
                fw.write(XMLDoc);
                fw.close();
                lblNombreArchivo.setText("Archivo : " + archivo.getName());
```

```
        lblNombreArchivo.setToolTipText(archivo.getAbsolutePath());

                } catch (ParserConfigurationException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                }

                setIndicadorGuardar(false);
                return true;
        }
```

El método “***raiz.getXML(detalles”;*** obtiene todos los detalles de la aplicación a ser generada, luego se la codifica a utf8.

```
        public static String outXML(Document doc) {
                try {
                        DOMSource domSource = new DOMSource(doc);
                        StringWriter sw = new StringWriter();
                        StreamResult streamResult = new StreamResult(sw);
                        TransformerFactory tf = TransformerFactory.newInstance();
                        Transformer serializer = tf.newTransformer();
                        serializer.setOutputProperty(OutputKeys.ENCODING,   "utf-
8");
                        serializer.setOutputProperty(OutputKeys.INDENT, "yes");
                        serializer.transform(domSource, streamResult);
                        return sw.toString();
                } catch (Exception e) {
                        e.printStackTrace();
                }
                return null;
```

```
}
```

Al momento de generar la aplicación debemos especificar el directorio donde se va a generar:

En este momento todo el proceso se delega a la clase Generador. A través de esta clase se empieza la generación del código para la aplicación, la cual es desconectada de la base de datos. Todos los detalles se encuentran en el archivo xml que se generó.

```
public static void main(String args[]) {
        try {
                long secs = System.currentTimeMillis();
                Log.setLogger(logger);
                if (args.length < 1) {
                        log("[Error] Uso : Generador <directorio-salida>
<archivo> <sobreescribir>");
                        return;
                }
                Boolean sobreescribir = null;
                if (args.length > 2) {
                        sobreescribir = new Boolean(args[2]);
                }
                if (args.length > 3) {
                        salirAlTerminar = new
Boolean(args[3]).booleanValue();
                }
                Generador genj = new Generador(args);
                if (GuiGenj.guiGenj == null) {
                        GuiGenj.guiGenj = new GuiGenj();
                        if (salirAlTerminar) {
                                GuiGenj.guiGenj.setVisible(false);
                                GuiGenj.guiGenj.setLogger(new LogConsola(
                                        GuiGenj.guiGenj.txtAConsola));
                        }
                        GuiGenj.guiGenj.cargarArchivoAplicacion(new
File(args[1]));
                        GuiGenj.guiGenj.guardar();
                }

                Plantilla plantilla =
GuiGenj.guiGenj.raiz.config.plantillaSeleccionada;
```

```
                    log.info("Generate using template: " +
plantilla.getPlantillaDir());
                    MotorPlantilla generador = null;
                    try {
                        generador = (MotorPlantilla)
Class.forName(plantilla.getClaseGenerador()).newInstance();
                    } catch (Exception e) {
                        log("Error! en la plantilla '" +
plantilla.getClaseGenerador() + "'  (" + e + ").");
                        try {
                            generador = (MotorPlantilla)
Class.forName(Plantilla.DEFAULT_ENGINE_CLASS).newInstance();
                            if (sobreescribir != null)

        generador.setSobreEscribir(sobreescribir);
                        } catch (Exception e1) {
                            log("No se pudo cargar la plantilla (" + e1 + ").
");
                            System.exit(1);
                        }
                    }

                    int totalArchivosNuevos =
generador.procesar(genj.getTemplateFiles(plantilla.getPlantillaDir().getAbsolute
Path()), dirSalida);

                    secs = System.currentTimeMillis() - secs;
                    secs = secs / 1000;
                    log("\nTotal Archivos nuevos : " + totalArchivosNuevos);

                    log("\nCopiando librerias nesesarias..");
                    try {
                        File directorioLibrerias = new File(args[0] +
File.separator + "lib");
                        if (!directorioLibrerias.exists()) {
                            if (!directorioLibrerias.mkdir()) {
                                log("No se pudo crear el directotio : " +
directorioLibrerias);
                            }
                        }

                        HashMap failedCopies =
CopiarLibs.copiarlibs(args[0]
                                + File.separator
                                +
GuiGenj.guiGenj.raiz.paths.getConfigOutput()
                                + File.separator + "lib.xml", args[0] +
File.separator
                                + "lib");
                        if (failedCopies == null) {
```

```
                                log("\nError al copiar las librerias necesarias.");
                        } else if (failedCopies.size() > 0) {
                                log("\nNo se copiaron todos los jars, a\u00f1adir manualmente los siguients jars:\n");
                                boolean j2eeJarPerdido = false;
                                for (Iterator iterator = failedCopies.keySet().iterator(); iterator.hasNext();) {
                                        String fileName = (String) iterator.next();
                                        String url = (String) failedCopies.get(fileName);
                                        log(" - " + fileName + " : " + url);
                                        if ("j2ee.jar".equalsIgnoreCase(fileName)) {
                                                j2eeJarPerdido = true;
                                        }
                                        log("");
                                }
                                if (j2eeJarPerdido) {
                                        log("j2ee.jar no se encuentra");
                                }
                        } else {
                                log("\nSe termino de copiar.");
                        }

                } catch (Exception e) {
                        log("\n\nError al copiar las librerias.\n");
                        e.printStackTrace();
                }
                log("se termino en " + secs + " sec.");
        } catch (InterruptedException e) {
                try {
                        log("Interrumpido por el usuario!");
                        if (salirAlTerminar) {
                                System.exit(-1);
                        }
                } catch (InterruptedException e1) {
                }
        } finally {
                GuiGenj.terminarGeneracion();
                if (salirAlTerminar) {
                        System.exit(0);
                }
        }
}
```

Dentro de esta clase existe un método para escribir el código generado en el directorio especificado:

```
        /**
         *Crea la estructura de directorios de la aplicación generada
         * @param pathPlantilla
         * @throws InterruptedException
         */
        public static void createOutputStructure(String pathPlantilla)
                        throws InterruptedException {
                try {
                        pathPlantilla = new File(pathPlantilla).getCanonicalPath();
                        pathPlantilla = pathPlantilla.replace('\\', '/');
                        File file = null;
                        EstructuraDirectorios paths = (EstructuraDirectorios)
GuiGenj.getObjetosDelArbol(EstructuraDirectorios.class).get(0);
                        Detalles config = (Detalles)
GuiGenj.getObjetosDelArbol(Detalles.class).get(0);
                        IteradorDirectorios iterator = new
IteradorDirectorios(pathPlantilla, true, true);
                        while ((file = iterator.getNext()) != null) {
                          boolean copyFile = false;
                          //no copia  CVS
                          String fullFilename = file.getCanonicalPath();
                          int lastDirPos =
fullFilename.lastIndexOf(System.getProperty("file.separator"));
                          if
(CVS_DIR.equals(file.getCanonicalPath().substring(fullFilename.length() -
CVS_DIR.length(), fullFilename.length())) ||
                                  CVS_DIR.equals(fullFilename.substring(lastDirPos -
CVS_DIR.length(), lastDirPos))) {
                            continue;
                          }
                           if ("readme.txt".equals(file.getName())) {
                              continue;
                           }
                          String fileOut = dirSalida.replace('\\', '/');
                          String path = file.getCanonicalPath().replace('\\', '/');


                          if (path.indexOf(pathPlantilla) == 0) {
                                  path = path.substring(pathPlantilla.length());

                            if
(path.startsWith(EstructuraDirectorios.CONF_GENERAL_DIR)) {

                                path = paths.getConfigOutput() +
path.substring(EstructuraDirectorios.CONF_GENERAL_DIR.length());
```

```
                    copyFile = true;
                }
                else if
(path.startsWith(EstructuraDirectorios.CONF_STRUTS_DIR)) {

                    path = paths.getConfigOutput() +
path.substring(EstructuraDirectorios.CONF_STRUTS_DIR.length());
                    copyFile = true;
                }

                else if
(path.startsWith(EstructuraDirectorios.JAVA_WEB_STRUTS_DIR)) {
                    path = paths.getJspOutput() +
path.substring(EstructuraDirectorios.JAVA_WEB_STRUTS_DIR.length());
                    if (config.verificaCapaWeb("struts").booleanValue()) {
                      copyFile = true;
                    }
                }

                else if
(path.startsWith(EstructuraDirectorios.JAVA_STRUTS_DIR)) {
                    path = paths.getWebOutput() +
path.substring(EstructuraDirectorios.JAVA_STRUTS_DIR.length());
                    if (config.verificaCapaWeb("struts").booleanValue()) {
                      copyFile = true;
                    }
                }

                else if
(path.startsWith(EstructuraDirectorios.JAVA_EJB3_DIR)) {
                    path = paths.getEjbOutput() +
path.substring(EstructuraDirectorios.JAVA_EJB3_DIR.length());
                    if (config.verificaCapaNegocios("ejb
3").booleanValue()) {
                      copyFile = true;
                    }
                }


                else if
(path.startsWith(EstructuraDirectorios.JAVA_SERVICE_DIR)) {
                    path = paths.getServiceOutput() +
path.substring(EstructuraDirectorios.JAVA_SERVICE_DIR.length());

                    copyFile = true;
                }

                else if
(path.startsWith(EstructuraDirectorios.JAVA_TEST_DIR)) {
```

```
                        path = paths.getTestOutput() +
path.substring(EstructuraDirectorios.JAVA_TEST_DIR.length());

                        copyFile = true;
                    }

                    else if ((path.indexOf("build.bat") != -1) ||
((path.indexOf("deploy.bat") != -1))) {
                        copyFile = true;
                    }
                }

                // En linux deberia empezar con / no con .
                if (!path.startsWith("/")) {
                    path = "/" + path;
                }

                if (copyFile) {
                    fileOut += path;
                    path = dirSalida + path;
                    if (!file.isDirectory()) {
                        String name = file.getName();
                        path = path.substring(0, (path.length() -
name.length()));
                    }

                    new File(path).mkdirs();
                    if (!file.isDirectory()) {
                        byte array[] = new byte[1024];
                        int size = 0;
                        try {
                            BufferedInputStream in = new
BufferedInputStream(new FileInputStream(file));
                            BufferedOutputStream out = new
BufferedOutputStream(new FileOutputStream(fileOut));

                            while ((size = in.read(array)) != -1)
                                out.write(array, 0, size);
                            in.close();
                            out.flush();
                            out.close();
                        } catch (Exception exc) {
                            log("[Error] Fallo la copia : " + fileOut);
                            log(exc.getMessage());
                        }
                    }
                }
            }
        } catch (Exception exc) {
            log.error("Error mientras se copiaban los archivo: ", exc);
```

```
                    log(exc.getMessage());
                } }
```

Durante todo el proceso en la parte inferior se muestra un log de las acciones de genj.

La clase "***LogConsola"*** maneja el log de genj:

```
    public static final HashMap<LogLevel, SimpleAttributeSet> mapaEstilos = new HashMap();

    static SimpleAttributeSet BLACK = new SimpleAttributeSet();

    static SimpleAttributeSet YELLOW = new SimpleAttributeSet();

    static SimpleAttributeSet RED = new SimpleAttributeSet();

    static {
        //Seteo coleres dependiendo del nivel de debug
        mapaEstilos.put(LogLevel.INFO, BLACK);
        mapaEstilos.put(LogLevel.DEBUG, YELLOW);
        mapaEstilos.put(LogLevel.ERROR, RED);

        StyleConstants.setForeground(BLACK, Color.BLACK);
        StyleConstants.setFontFamily(BLACK, "Lucida");
        StyleConstants.setFontSize(BLACK, 10);

        StyleConstants.setForeground(YELLOW, Color.YELLOW);
        StyleConstants.setFontFamily(YELLOW, "Lucida");
        StyleConstants.setFontSize(YELLOW, 10);

        StyleConstants.setForeground(RED, Color.RED);
        StyleConstants.setFontFamily(RED, "Lucida");
        StyleConstants.setFontSize(RED, 10);
    }

    private boolean deColor;

    private JTextComponent consolaGenerica;

    /**
     * Constructor de la clase
     * @param console
     */
    public LogConsola(JTextArea console) {
        setconsolaGenerica(console);
        setDeColor(false);
    }
```

```
	/**
	 * Constructor de la clase con soporte para color
	 *
	 * @param console
	 */
	public LogConsola(JTextPane console) {
		setconsolaGenerica(console);
		setDeColor(true);
	}

	/**
	 * Log en la consola
	 * @param mensaje de log
	 */
	public void log(String mensaje) {
		log(mensaje, LogLevel.INFO);
	}

	/**
	 * Log a la consola deacuerdo a un nivel de log
	 * @param mensaje
	 * @param nivel
	 */
	public void log(String mensaje, LogLevel nivel) {
		if (estaEnColor()) {
			insertarTexto(mensaje, getEstiloPorNivel(nivel));
		} else
			insertarTexto(mensaje, BLACK);
		insertarTexto("\n", BLACK);
		setFinalSeleccion();
	}

	private SimpleAttributeSet getEstiloPorNivel(LogLevel nivel) {
		SimpleAttributeSet c = (SimpleAttributeSet)
LogConsola.mapaEstilos.get(nivel);
		SimpleAttributeSet ret = BLACK;
		if (c != null)
			ret = c;
		return ret;
	}

	public boolean estaEnColor() {
		return deColor;
	}

	public void setDeColor(boolean deColor) {
		this.deColor = deColor;
	}
```

```
	public JTextComponent getconsolaGenerica() {
		return consolaGenerica;
	}

	public void setconsolaGenerica(JTextComponent consolaGenerica) {
		this.consolaGenerica = consolaGenerica;
	}

	protected void insertarTexto(String texto, AttributeSet set) {
		try {
			consolaGenerica.getDocument().insertString(
					consolaGenerica.getDocument().getLength(),
texto, set);
		} catch (BadLocationException e) {
			e.printStackTrace();
		}
	}

	protected void setFinalSeleccion() {

	consolaGenerica.setSelectionStart(consolaGenerica.getDocument()
				.getLength());

	consolaGenerica.setSelectionEnd(consolaGenerica.getDocument()
				.getLength());
	}

	/**
	 * Log con Nivel de Error en la consola
	 * @param mensaje
	 */
	public void error(String mensaje) {
		log(mensaje, LogLevel.ERROR);
	}

	/**
	 * Log con nivel de debug en la consola
	 * @param mensaje
	 */
	public void debug(String mensaje) {
		log(mensaje, LogLevel.DEBUG);
	}
```

## 3.2 **PRUEBAS**

Siguiendo los pasos indicados por la metodología que seleccionamos entramos a la fase de ***Transmisión***; dentro de la cual se realizaran pruebas a fin de poder construir un producto de calidad.

Genj es una aplicación mono usuario, para la cual se realizaron varias pruebas de manejo de errores, cambios de configuración, generación de código y funcionalidad; dichas pruebas fueron realizadas por un usuario que se dedica a la construcción de aplicaciones java y que tiene un promedio de experiencia con el lenguaje de un año[3].

### 3.2.1. PRUEBAS DE FUNCIONALIDAD

Comentario [J1]:

*3.2.1.1 Manejo de errores*

-Borramos el archivo de configuración genj.xml:

No se puede iniciar genj sin el archivo de configuración, en cuyo caso se despliega el siguiente mensaje de error:


**Figura 3.15 Error producido por el archivo de configuración.**

-Modificamos la estructura del archivo de configuración genj.xml:

Si el archivo de configuración no cumple la estructura establecida, se presenta el siguiente mensaje de error:

[3] bitpipe , Systems Testing, http://www.bitpipe.com/rlist/term/Systems-Testing.html, Ultimo Acceso: Junio 2006.

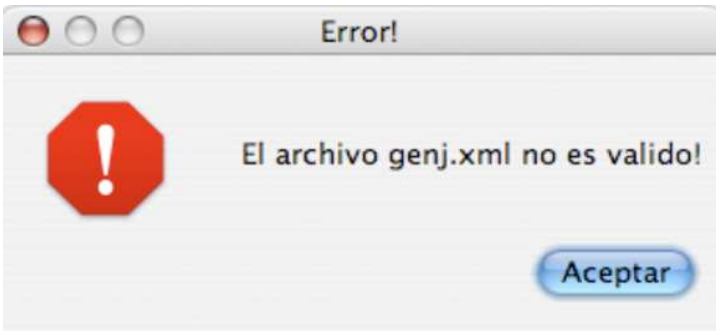



**Figura 3.16 Error producido por modificar el archivo de configuración.**

-Quitamos del archivo de configuración el tag bases-soportadas, en el que se especifica a que bases de datos se va a poder conectar.

Con el archivo sin modificar tenemos estos resultados:

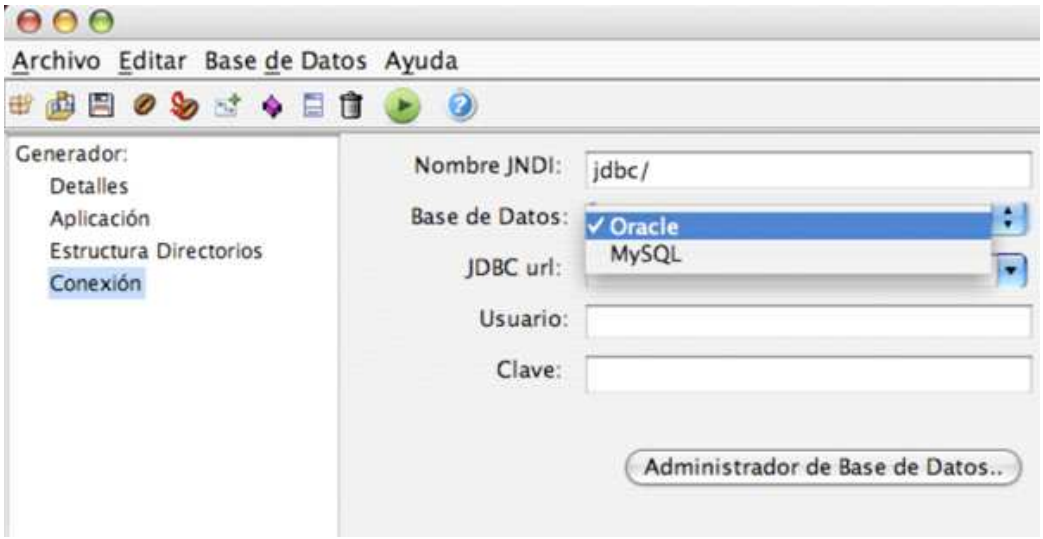



**Figura 3.17 Archivo de configuración no modificado**

Cuando modificamos el archivo de configuración genj.xml, **Genj** se inicia sin problemas, pero en el diálogo de conexión a la base de datos no se despliega la información relacionada al dbms correctamente.


**Figura 3.18 Error producido por la ausencia del archivo de configuración.**

-Borramos el archivo: "/plantillas/genj/plantilla.xml"

Cuando el archivo esta presente, el resultado es el siguiente:

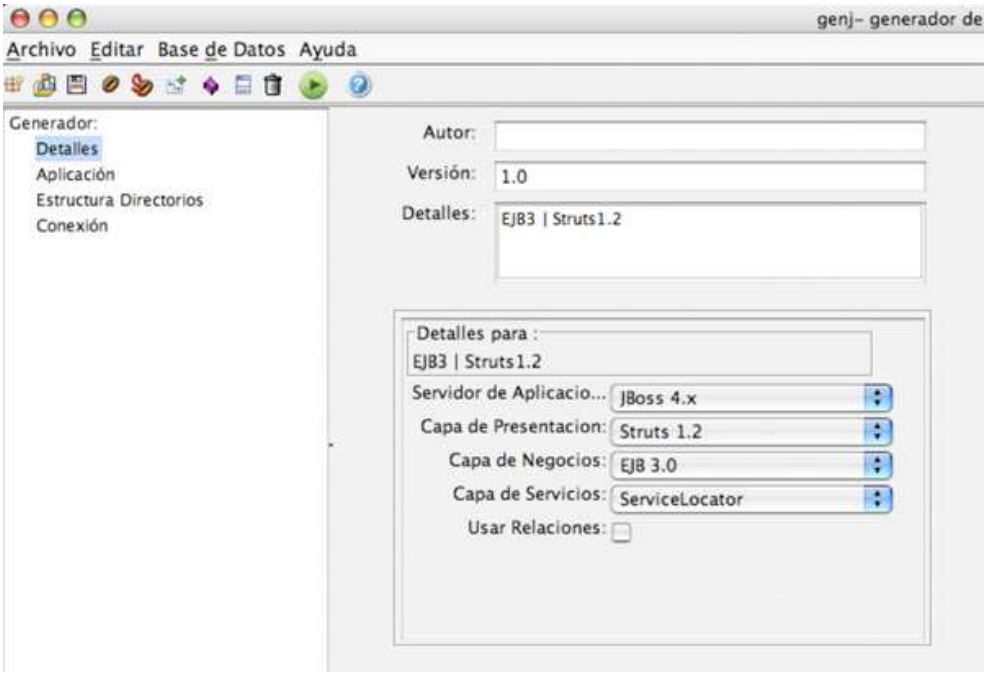



**Figura 3.19 Archivo de plantillas presente.**

Los detalles de generación se muestran en la pantalla anterior.

Al borrar el archivo no se muestra ningún detalle:

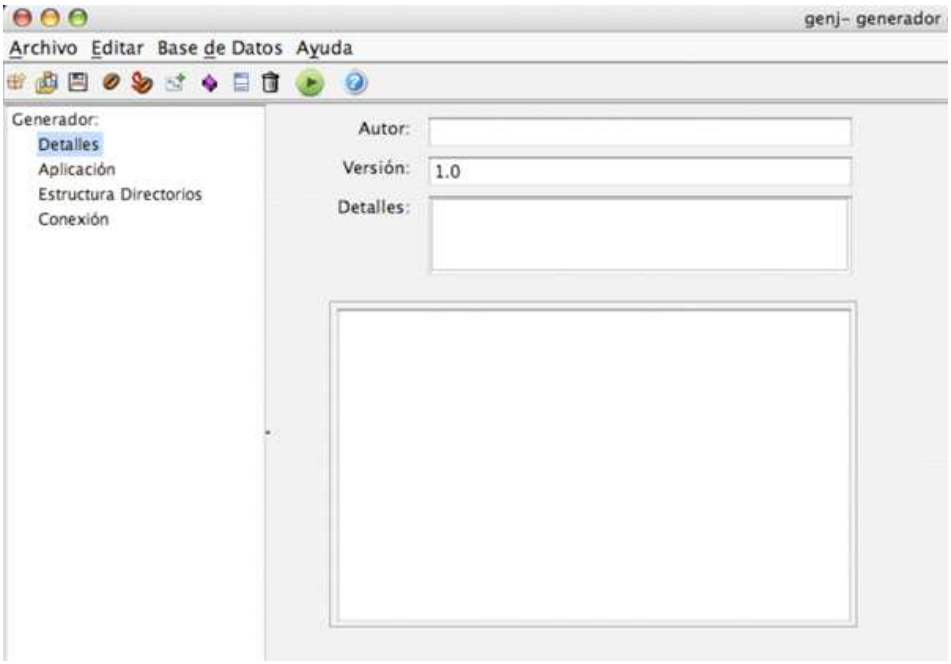



**Figura 3.20 Archivo de Plantillas Borrado**

-Modificamos la estructura del archivo "plantilla.xml" el cual contiene información referente a las plantillas para generación de código y se presenta el siguiente mensaje de error:


**Figura 3.21 Modificación de la plantilla de generación.**

-Paramos el servidor de base de datos al que nos conectamos y se presenta el siguiente mensaje:

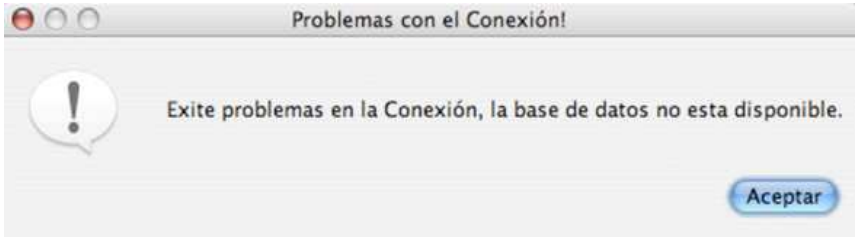



**Figura 3.22 Error producido debido a que no está en funcionamiento el dbms.**

-Durante una conexión ya establecida detenemos el servidor de base de datos e intentamos obtener el listado de las tablas, se muestra el siguiente error:

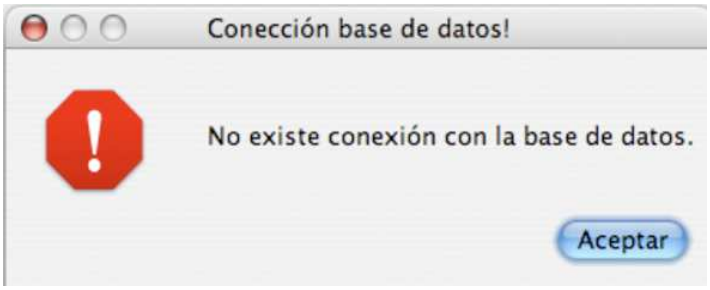



**Figura 3.23 Parar el servicio del dbms accidentalmente.**

-No se ha creado ningún entity bean y se intenta asociar a un session bean, se despliega el siguiente mensaje de error:


**Figura 3.24 Orden de generación errónea**

-No se ha creado ningún entity bean y se intenta agregar una relación, se muestra el siguiente mensaje:

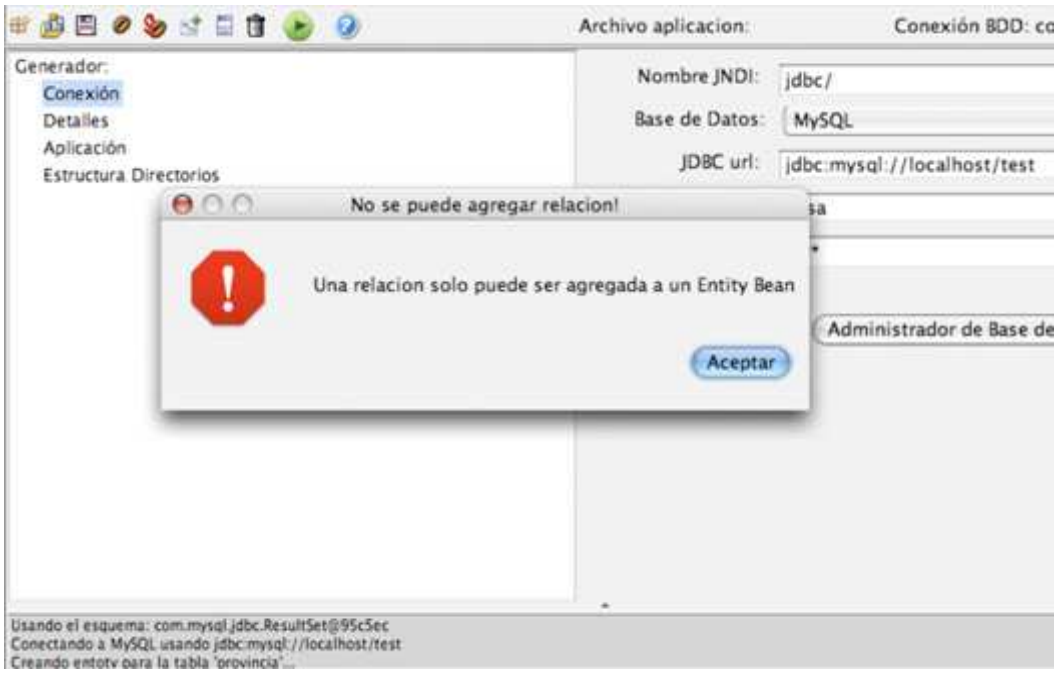



**Figura 3.25 Agregar una relación sin tener entidades creadas.**

De igual manera si intentamos agregar un campo cuando aun no se ha creado ningún entity nos despliega este error:


**Figura 3.26 Agregar campos sin tener entidades**

Si se intenta generar la aplicación sin guardarla, o cuando aún no se ha seleccionado una aplicación existente se despliega el siguiente error:

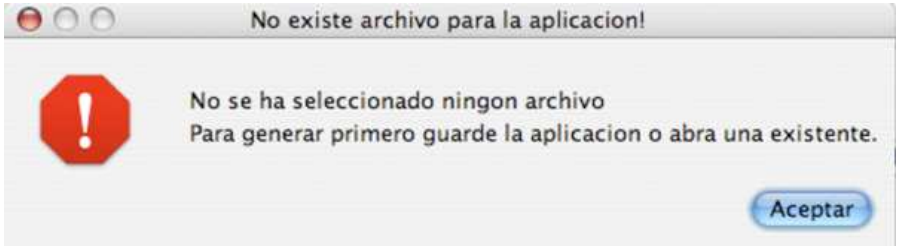



**Figura 3.27 Intentar generar una aplicación sin haber guardado los cambios.**

Cuando se intenta abrir un archivo de una aplicación que esta incorrecto se presenta el siguiente mensaje:

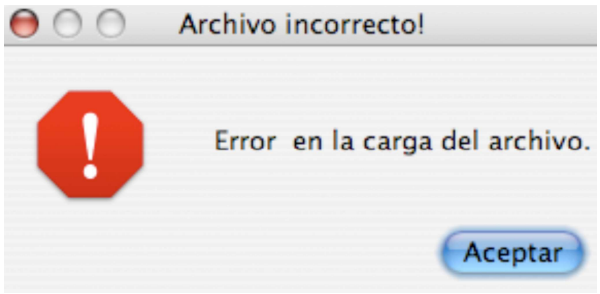



**Figura 3.28 Archivo de aplicación incorrecto.**

Cuando intentamos añadir un nuevo driver y seleccionamos un archivo que no contenga un driver, se muestra el siguiente mensaje:

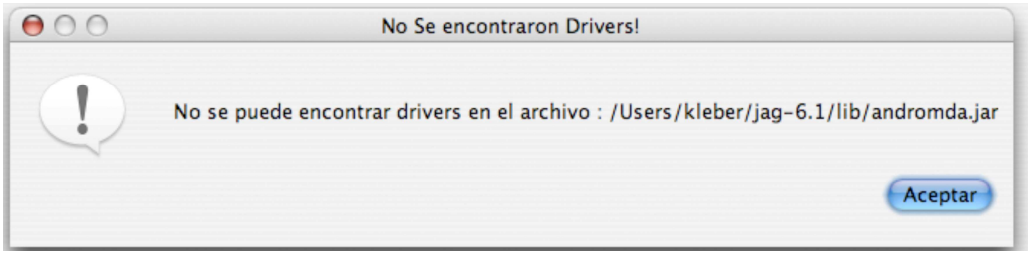



**Figura 3.29 Error producido por seleccionar un driver incorrecto.**

*3.2.1.2 Funcionalidad:*

Para esta prueba utilizamos una base de datos de prueba que contiene cuatro tablas cuya definición es la siguiente:

```
create table t_usuario (
      id int(11)  not null,
      nombre text,
      fecha_ingreso date,
      primary key (id)
) type=innodb;

create table t_direccion (
      id int(11)  not null,
      id_usuario int(11),
      codigo_postal text,
      numero_casa text,
      calle text,
      index user_ind (id_usuario),
      primary key (id),
      foreign key (id_usuario) references t_usuario(id) on delete cascade
) type=innodb;

create table t_proyecto (
   id int(11) not null,
   nombre text,
   primary key(id)
) type=innodb;

create table t_usuario_proyecto (
   id_usuario int(11),
   id_proyecto int(11),
         primary key (id_usuario, project_id),
         foreign key(id_usuario) references t_usuario(id),
         foreign key(id_proyecto) references t_proyecto(id)
) type=innodb;
```

Los detalles de conexión:


**Figura 3.30 Ejemplo detalles conexión.**

Detalles de la aplicación:

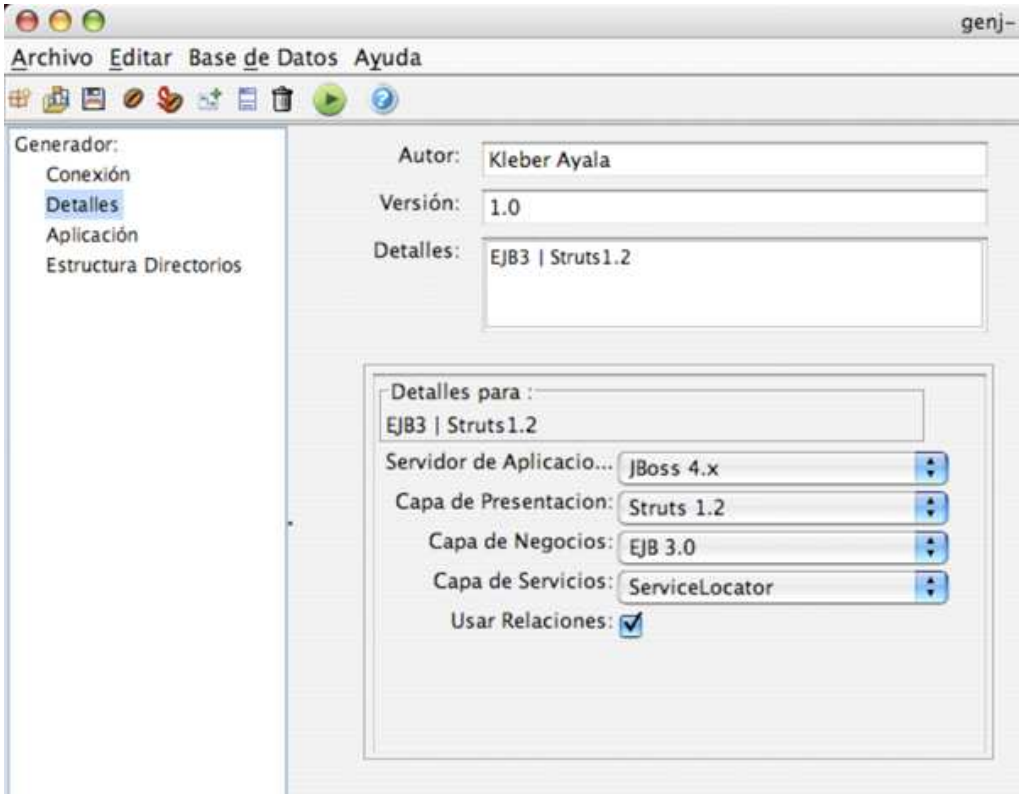



**Figura 3.31 Ejemplo detalles de aplicación.**

Datos generales de la aplicación: Nombre, Versión, Descripción, Paquete Raíz.

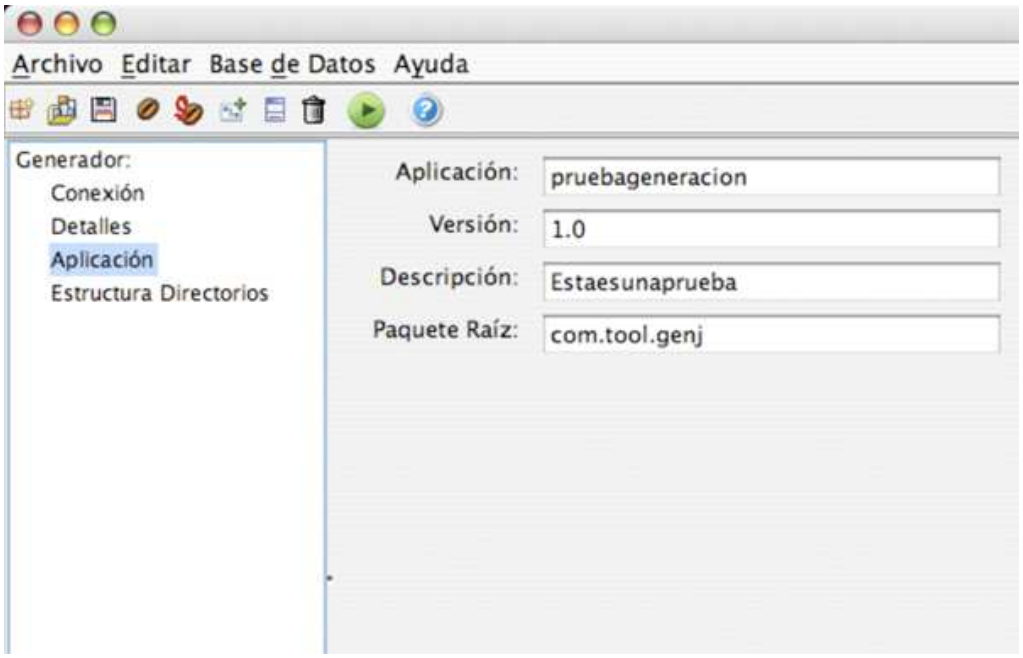



**Figura 3.32 Ejemplo datos generales de la aplicación.**

Estructura de directorios para la generación de código: Define la estructura para cada capa que será guardada bajo la jerarquía de directorios aquí definida.


**Figura 3.33 Ejemplo detalles conexión.**

Seleccionamos las tablas que se van a ser mapeadas a cada entity bean.

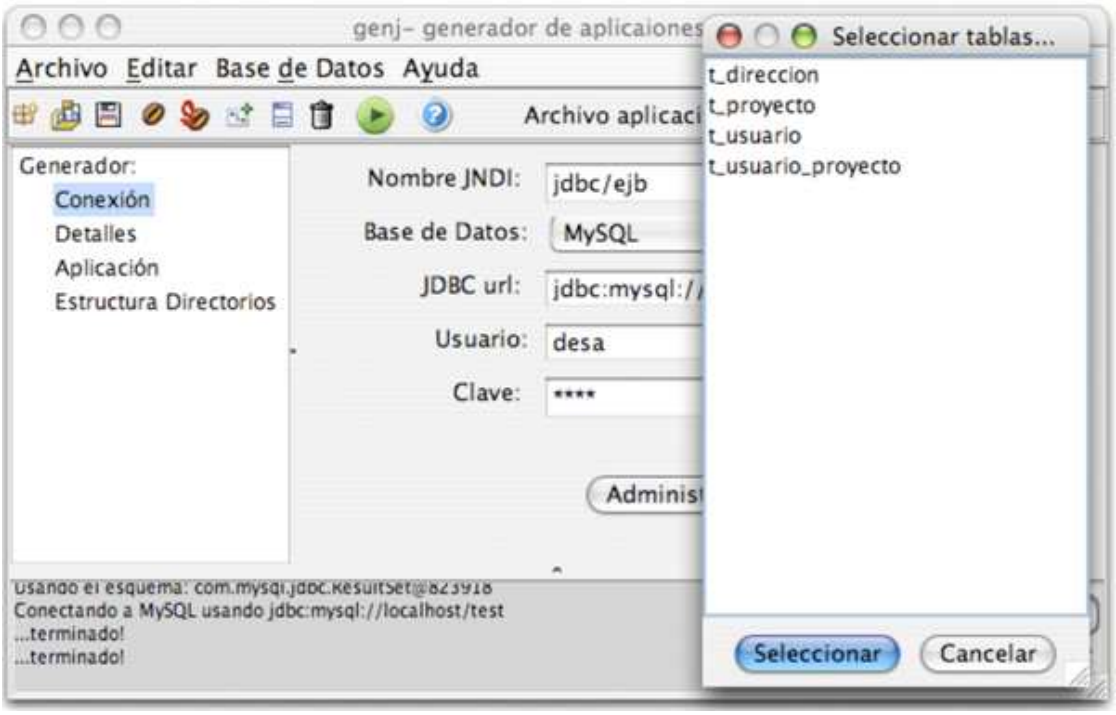



**Figura 3.34 Ejemplo mapeo de tablas a entidades.**

Luego de agregar las entidades session bean, los mismos que deberán estar relacionados a una entidad previamente agregada:

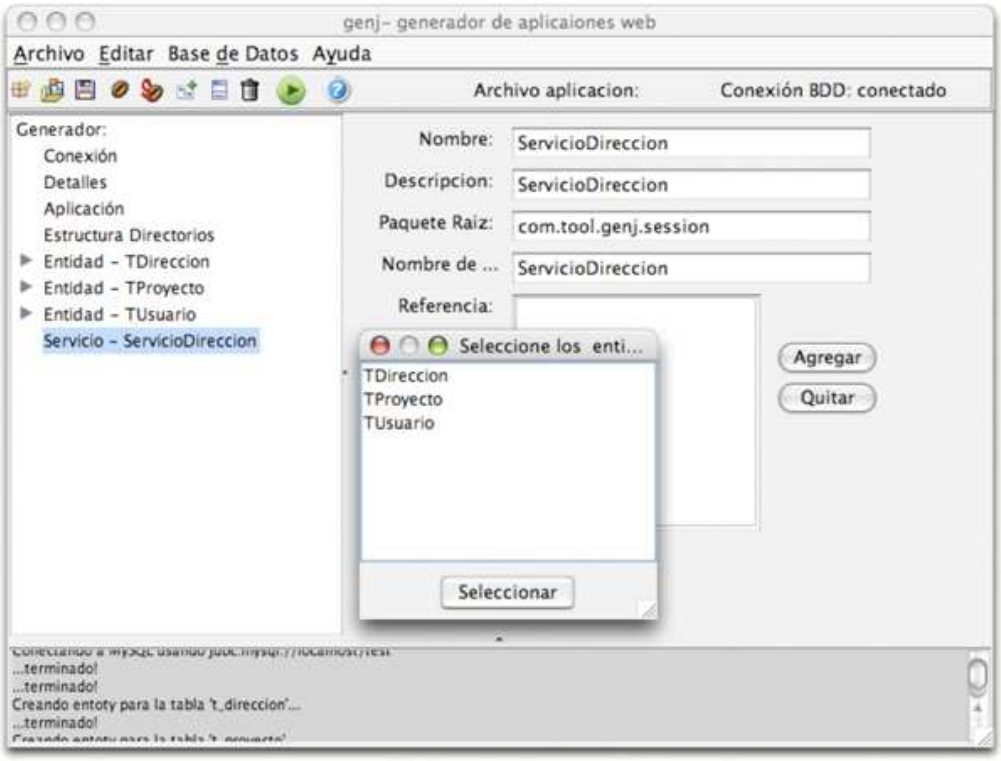



**Figura 3.35 Agregar Session Bean**

Procedemos a la generación del código y obtenemos el siguiente resultado.

Podemos observar la estructura de directorios que se ha generado la cual representa las diferentes capas para el modelo MVC.

- **Modelo**. Dentro de la carpeta "java-ejb" podemos encontrar entity y session bean, los mismos que se encargan de representar el modelo de datos.
- **Vista**. Bajo la carpeta "java-web" y "web" podemos ver que se han creado las carpetas actions y forms las cuales contienen clases y páginas jsp que son las encargadas de desplegar la información.
- **Controlador**. Clases contenidas dentro de la carpeta "java-service" son encargadas de interpretar las órdenes del usuario actuando directamente con el modelo.

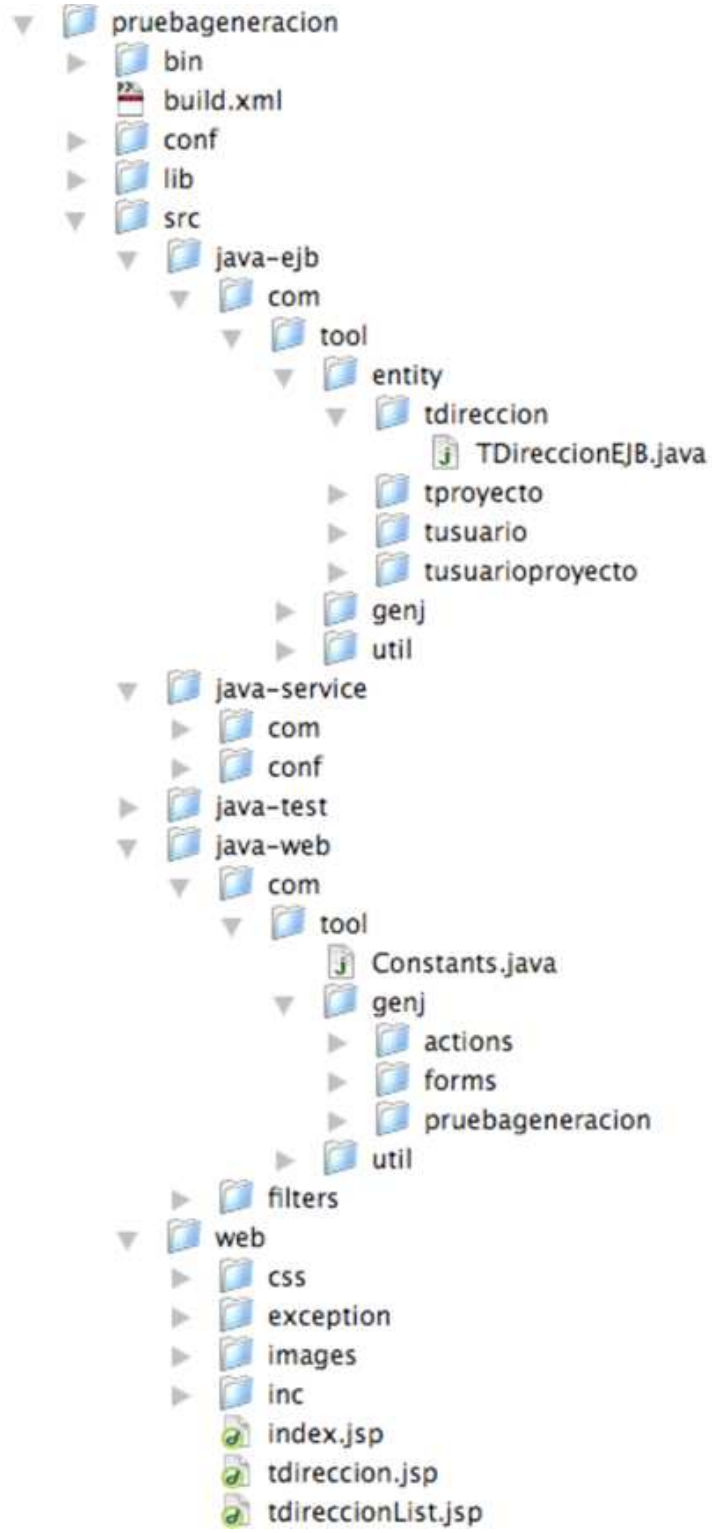



**Figura 3.36 Resultado de la generación de código.**

# CAPITULO 4. CASO DE ESTUDIO

Luego de haber concluido la implementación y las pruebas respectivas de la herramienta, el presente capítulo estará dedicado a la demostración práctica del uso de la herramienta para la generación de las operaciones básicas de una aplicación web.

El siguiente apartado presenta la especificación del ambiente de pruebas que incluye, tanto características del servidor de base de datos, como los modelos lógico y físico de la base de datos que vamos a utilizar como caso de estudio; adicionalmente, se describirán las situaciones prácticas que hemos seleccionado, las mismas que darán la pauta para la realización de las pruebas.

## 4.1 DEFINICIÓN DE AMBIENTE DE PRUEBA.

La herramienta está diseñada para la generación de código fuente. A continuación se describirá las características básicas para la ejecución del programa.

### 4.1.1 CARACTERÍSTICAS DEL COMPUTADOR

#### *4.1.1.1 Hardware*

La Tabla 4.1 muestra las principales características de hardware.

| **Característica** | **Descripción** |
|---|---|
| **Procesador** | **Pentium 4, 1.8ghz o superior.** |
| **Memoria** | **512 mb o superior** |

**Tabla 4.1 Configuración de Hardware**

#### *4.1.1.2 Software*

El computador requiere tener instalado una base de datos con un proveedor compatible con java, además se requiere instalar la máquina virtual de java.

La Tabla 4.2 muestra las principales características de software.

| **Característica** | **Descripción** |
|---|---|
| **Máquina Virtual de Java** | **j2se Versión 1.5.** |
| **DBMS** | **MySQL Versión 4 o superior.** |
| **Servidor de Aplicaciones** | **Jboss 4.04** |

**Tabla 4.2 Configuración Software**

### 4.1.2 ESPECIFICACIÓN DE BASE DE DATOS

Este apartado presentará los modelos, tanto lógico como físico de la base de datos seleccionada para el presente caso de estudio práctico de uso de la herramienta.

La base de datos que seleccionamos se refiere un registro básico de libros y autores. La Figura 4.1 muestra el modelo lógico.

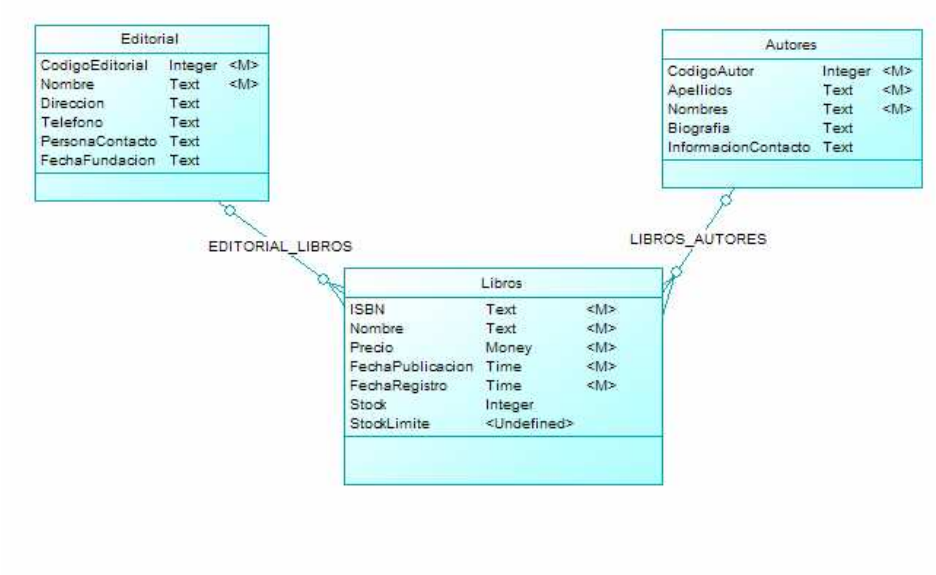



**Figura 4.1 Modelo Lógico de Base de Datos**

Como se puede observar en la Figura 4.1, la base consta de tres tablas que permiten llevar registro básico de una librería. Una estructura para guarda libros y su stock, otra estructura para guardar autores, de manera que un autor puede tener varios libros y finalmente una estructura para guardar las editoriales; una editorial puede tener varios libros.

La Figura 4.2 presenta el modelo físico de la base

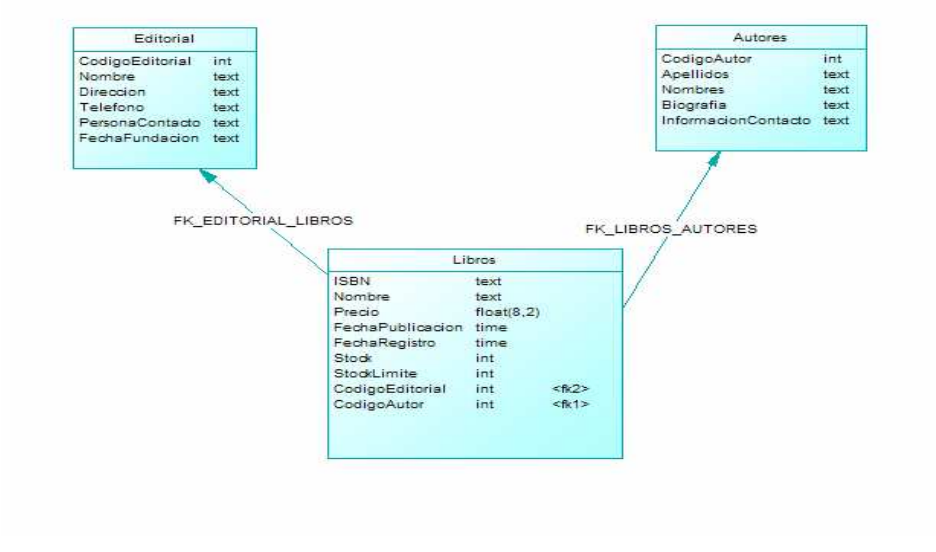



**Figura 4.2 Modelo Físico de Base de Datos**

En la Figura 4.2 se observan además, las claves foráneas de cada tabla que constan como columnas adicionales con las respectivas relaciones entre tablas.

## 4.2 USO DE LA HERRAMIENTA.

### 4.1.1 GENERACIÓN DE CÓDIGO

Para generar la aplicación, primero hay que crear la base de datos en el DBMS, luego iniciamos la herramienta generadora de código e ingresamos los detalles de la conexión a la base de datos:

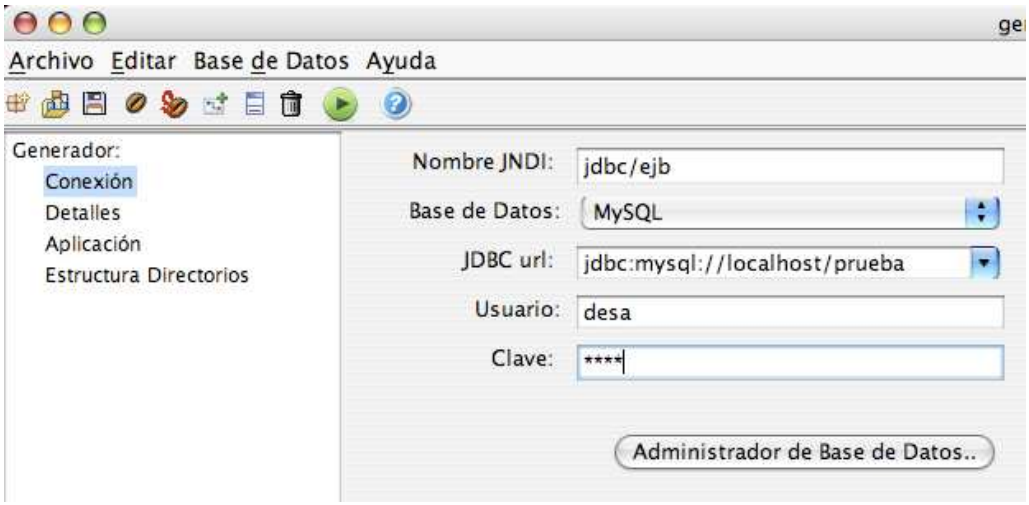



**Figura 4.3 Pantalla para especificar los detalles de conexión del dbms.**

De igual manera el resto de detalles relacionados a la aplicación:

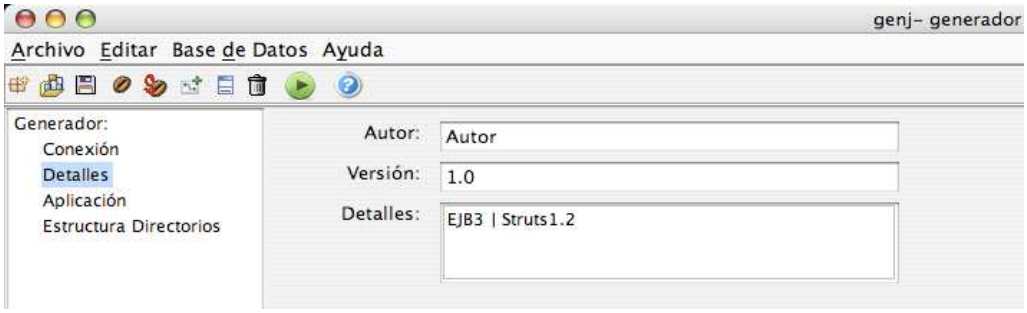



**Figura 4.4 Pantalla para especificar los detalles generales de la aplicación.**

Detalles de la aplicación:


**Figura 4.5 Pantalla para especificar los detalles de la aplicación.**

Estructura de directorios:

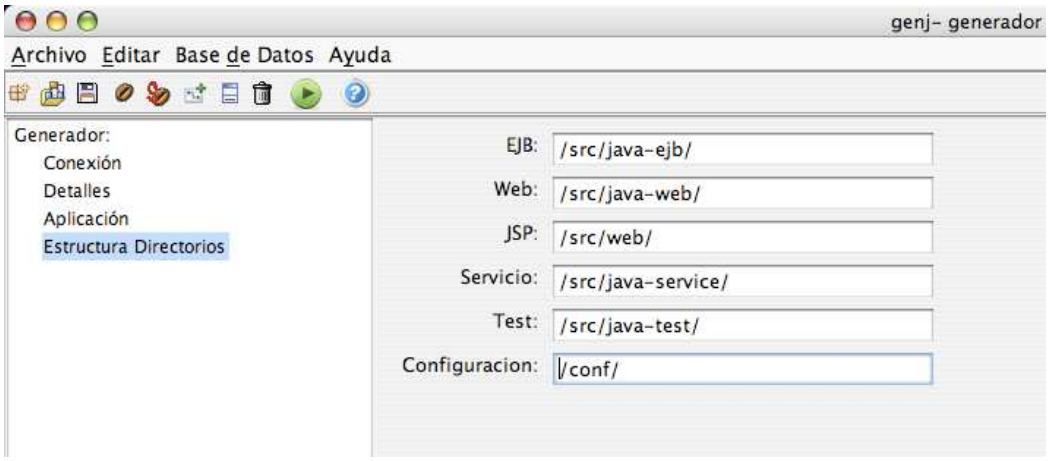



**Figura 4.6 Pantalla para especificar la estructura de directorios.**

Establecemos la conexión a la base de datos y especificamos si queremos mostrar tablas vistas o sinónimos:


**Figura 4.7 Conexión con un dbms.**

En el log de la aplicación se muestra el siguiente mensaje:

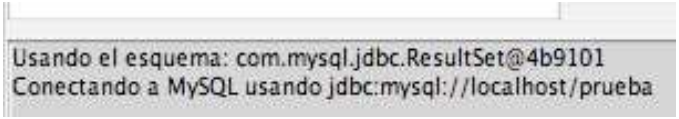



**Figura 4.8 Pantalla para especificar los detalles de la aplicación.**

Seleccionamos las tablas que deseamos como entitys en nuestra aplicación:


**Figura 4.9 Selección de tablas para generación de código.**

Especificamos los detalles de los entitys asociados a cada tabla:

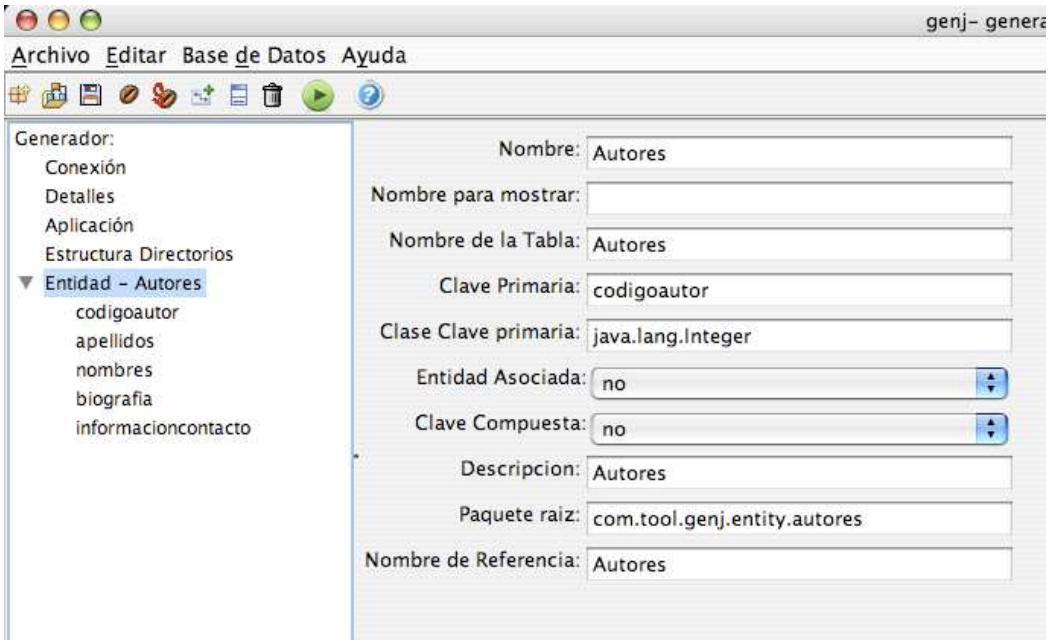



**Figura 4.10 Especificar propiedades para cada entity.**

De igual manera los detalles de los campos de cada entity:

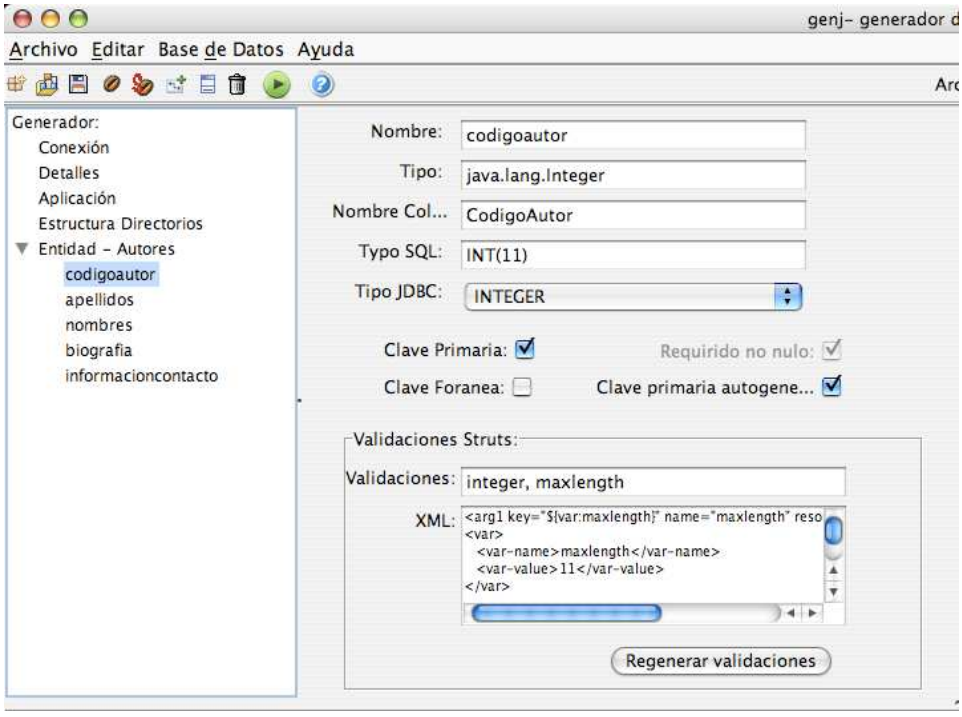



**Figura 4.11 Propiedades particulares de cada campo que contiene una entidad.**

Para cada campo es necesario definir las propiedades mostradas en la Figura 4.11, además se puede incluir un código xml para que genere una validación de dicho campo a nivel del cliente.

Continuamos agregando los entitys:

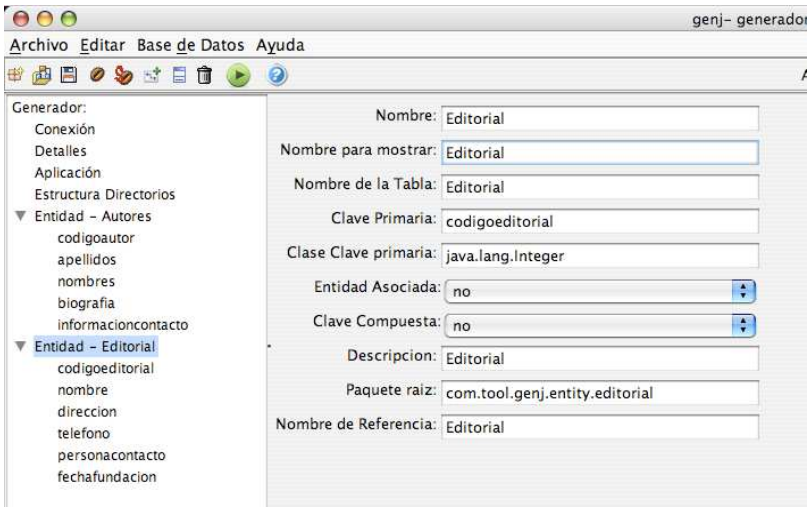



**Figura 4.12 Agregar la entidad editorial.**

A continuación definiremos los detalles para cada campo que pertenece a la entidad editorial.

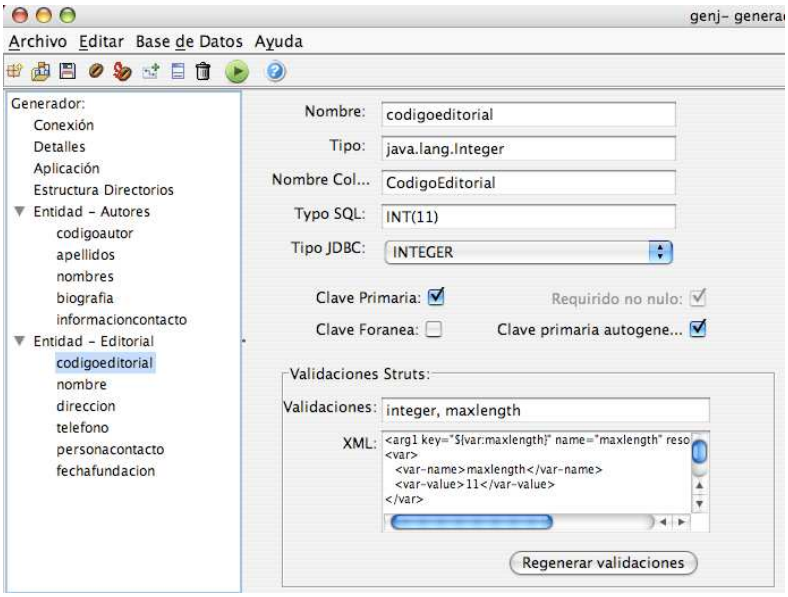



**Figura 4.13 Definición del campo "codigoeditorial" para la entidad "Editorial"**

Finalmente agregamos la entidad Libros y definiremos las propiedades para cada campo de la entidad.

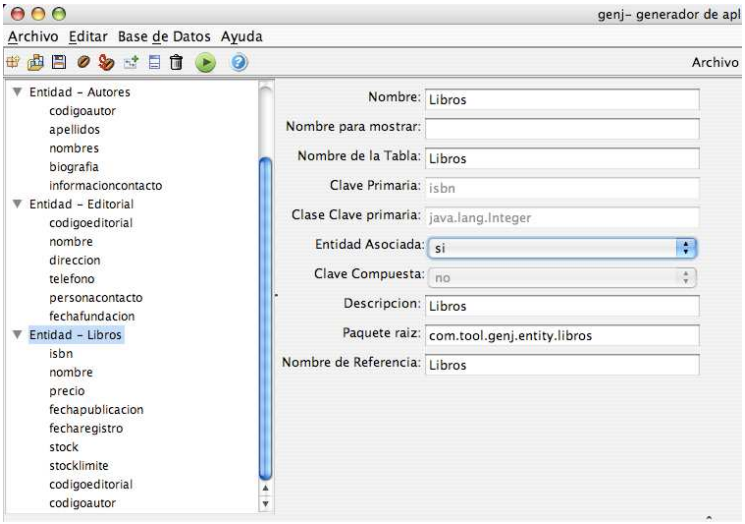



**Figura 4.14 Agregar la entidad editorial.**

Según el modelo de datos que definimos observamos una relación entre las tablas editorial y libros, la cuál debe ser representada:

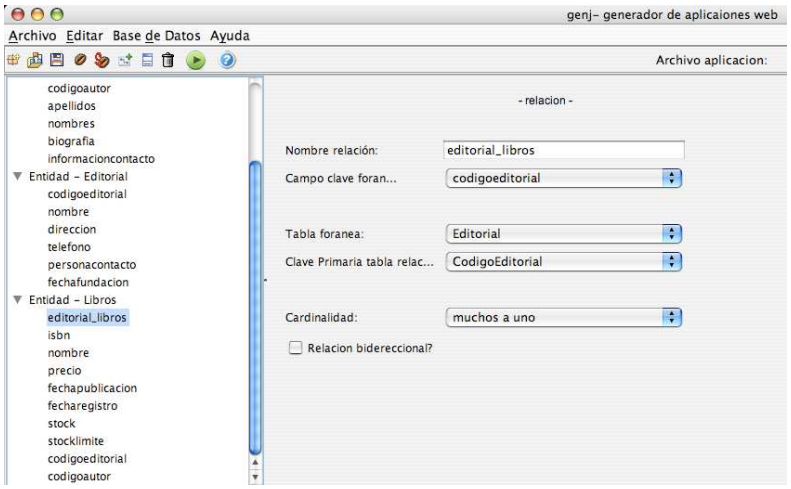



**Figura 4.15 Relación editorial - libros.**

Otra relación representada en el modelo es la tabla libros con la tabla autores, la cuál es representada de la siguiente manera.

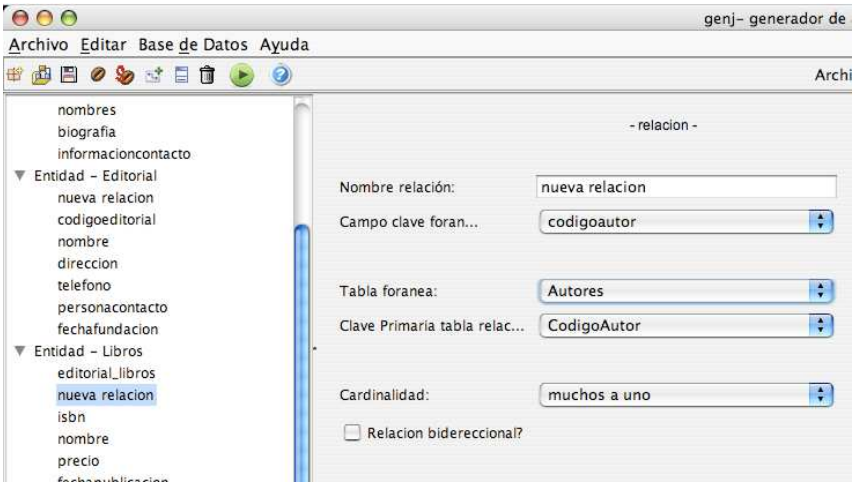



**Figura 4.16 Relación libros - autores.**

Para proceder con la generación del código guardamos la aplicación:

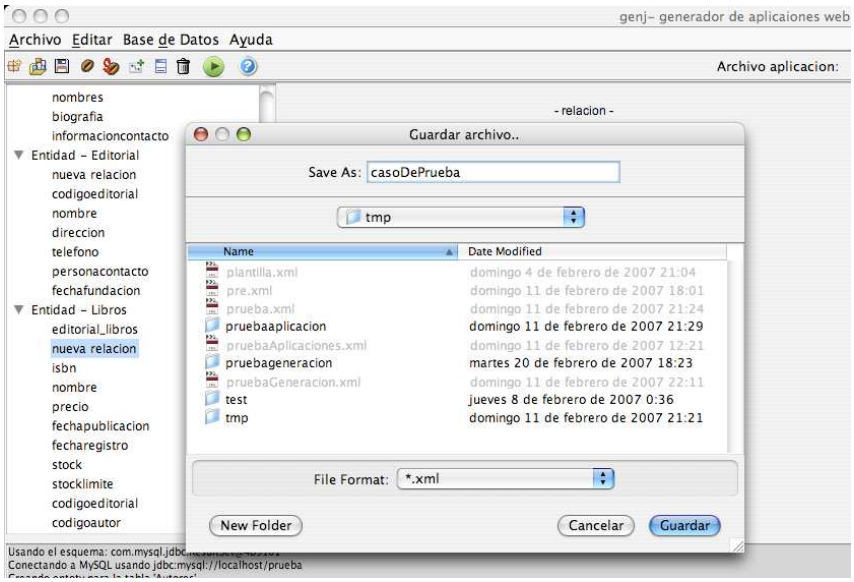



**Figura 4.17 Guardar la aplicación previo a la generación de código.**

### 4.2.2 Resultados.

Genj crea la estructura de directorios especificada anteriormente en la configuración, y dentro de esta estructura las clases .jsp y archivos de configuración y deployment generados.

Las capas generadas estan de acorde con la arquitectura “MVC”.

**Modelo.** Conformada por los entity y session bean, encarcados de manejar el modelo de datos.

**java-ejb**

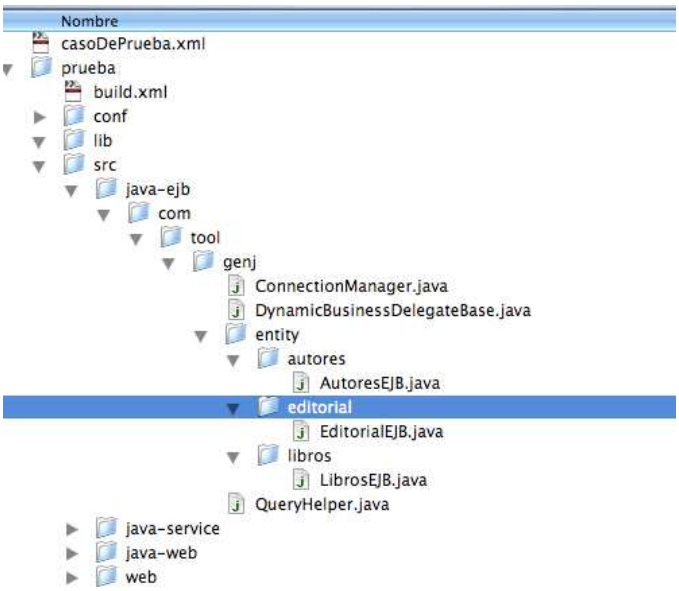



**Figura 4.18 Archivos Generados para la capa java-ejb.**

**Controlador.** Encargado de interpretar las órdenes de los usuarios y de comunicarse directamente con el modelo.

**java-service**

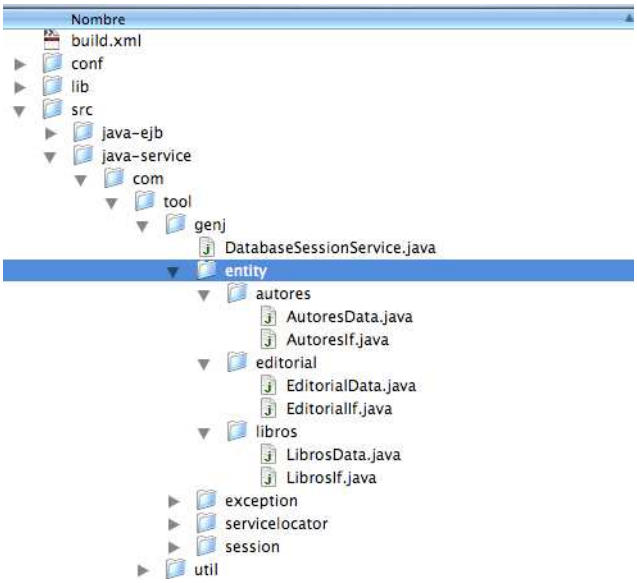



**Figura 4.19 Archivos Generados para la capa java-service.**

**Vista.** Conformado por “java-web” y “web” encargado de representar los datos de modelo; además de validar y entregar dichos datos al controlador

**java-web**


**Figura 4.20 Archivos Generados para la capa java-web.**

**web**

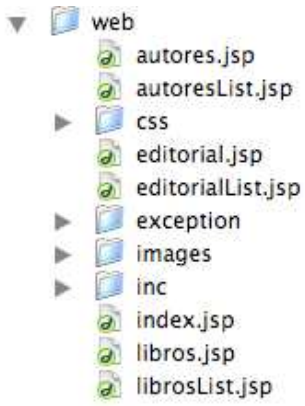
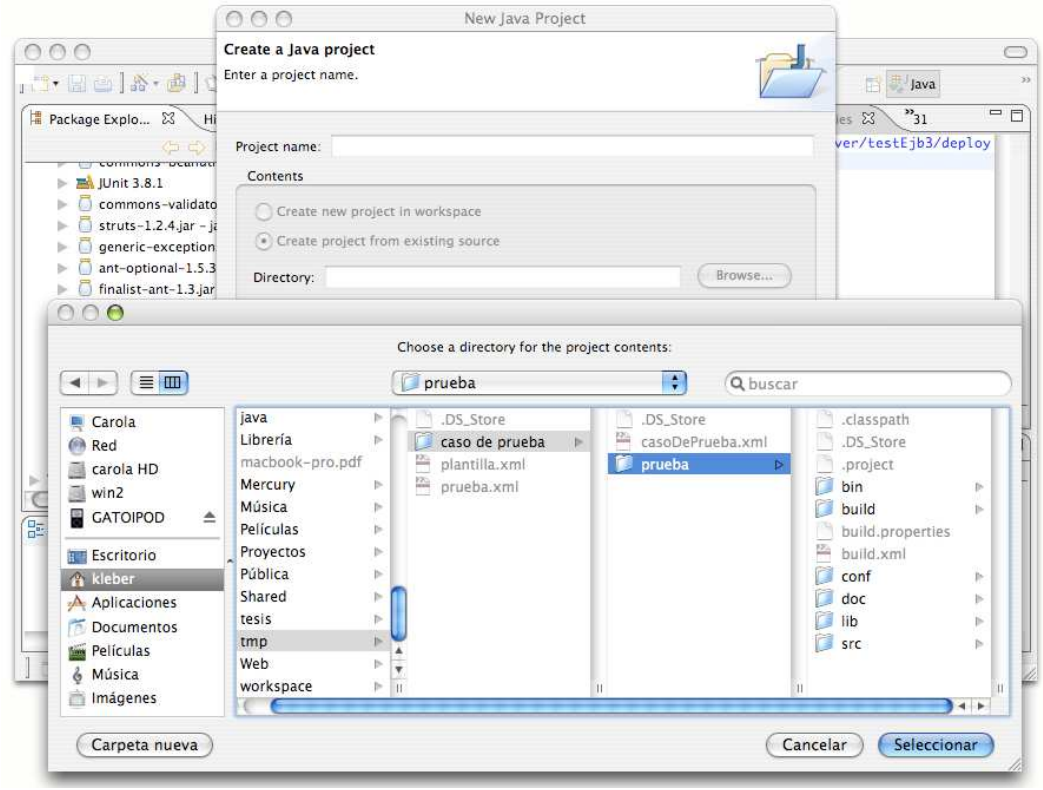



**Figura 4.21 Archivos Generados para la capa web.**

Para revisar de mejor manera el código generado utilizamos el ambiente de desarrollo de eclipse:



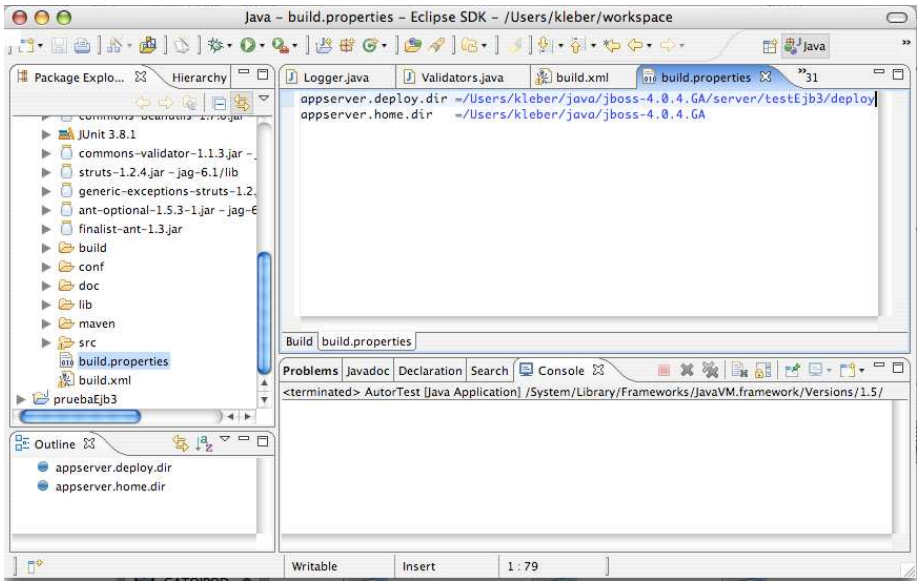



**Figura 4.22 Código generado visto desde Eclipse.**

Tenemos que modificar el archivo de propiedades **build.properties** y especificar el directorio de nuestro servidor de aplicaciones.

appserver.home.dir =/Users/kleber/java/jboss-4.0.4.GA

Para probar la aplicación ejecutándose tenemos que deployar el proyecto, este proceso compilara, empaquetará, y copiará el archivo de la aplicación al servidor de aplicaciones:

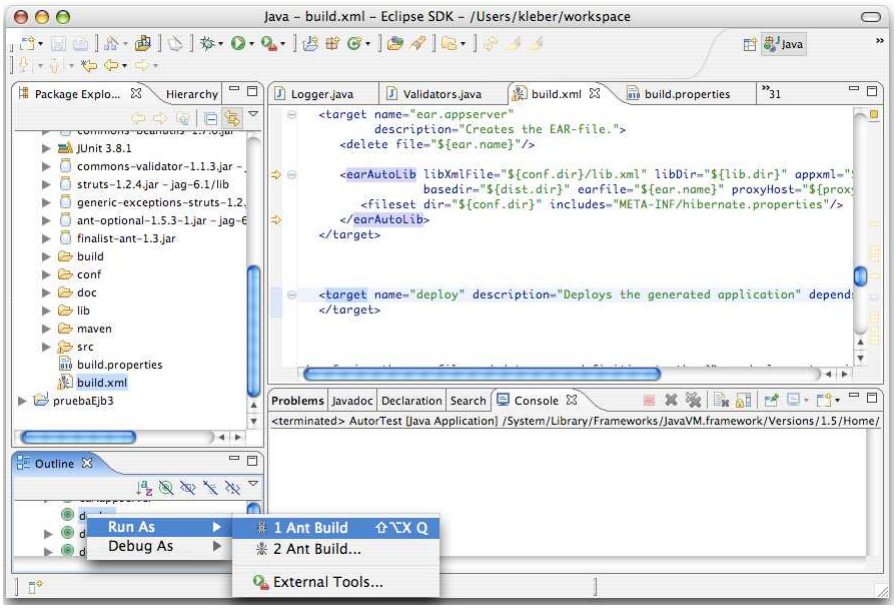



**Figura 4.23 Edición del archivo build.xml.**

En la consola de eclipse se muestra el resultado del proceso de deployment:

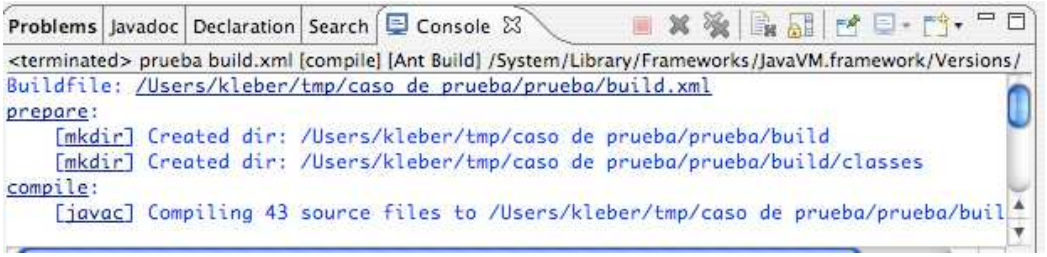



**Figura 4.24 Resultado del proceso de deployment.**

En el siguiente gráfico se muestra la estructura de directorios del servidor de aplicaciones:

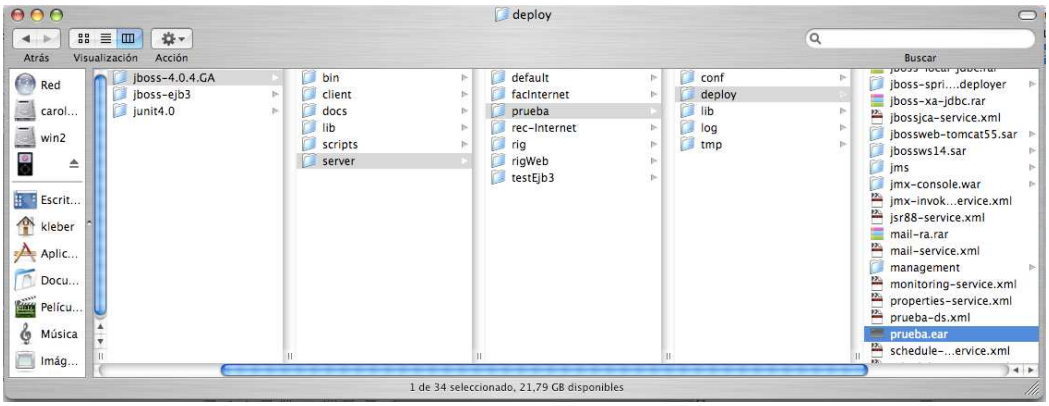



**Figura 4.25 Estructura de directorios en el servidor de aplicaciones.**

Previamente creamos un archivo de fuente de datos para conectarnos al dbms "mysql", **prueba-ds.xml** en el que especificamos los detalles de conexión a la base de datos:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- ===================================== -->
<!--  Datasource para Caso de Prueba
<!-- ===================================== -->
<datasources>
  <local-tx-datasource>
    <jndi-name>pruebaDS</jndi-name>
    <connection-url> jdbc:mysql://localhost/prueba</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>desa</user-name>
    <password>desa</password>
</local-tx-datasource>
</datasources>
```

Este archivo no es generado por genj, tiene que ser agregado en la carpeta deploy del proyecto en el servidor de aplicaciones.

Para poder probar los resultados es necesario ejecutar el servidor de aplicaciones:


**Figura 4.26 Ejecutar la aplicación.**

Ingresamos a nuestra aplicación de prueba ingresando la dirección que muestra la figura:


**Figura 4.27 Pantalla principal de la aplicacion.**

Del lado izquierdo de la aplicación podemos visualizar el menú de opciones con las distintas entidades que tienen nuestra aplicación, seleccionamos la opción autores.

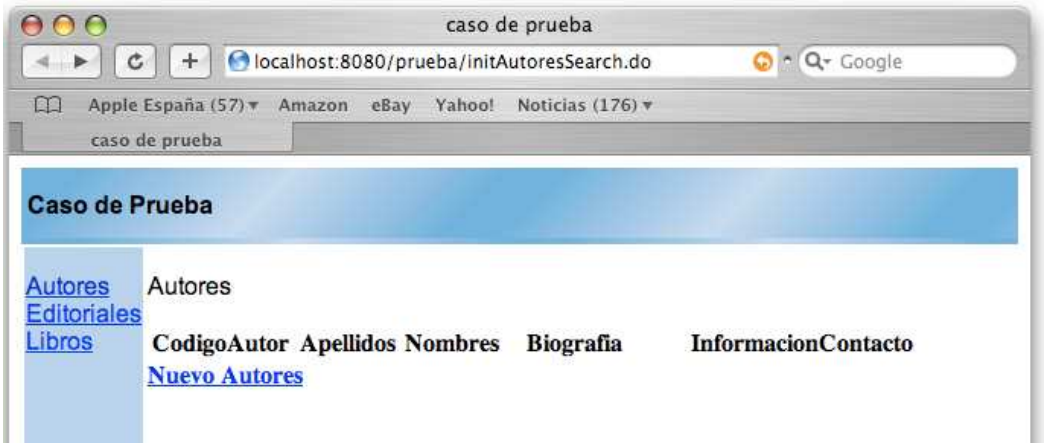



**Figura 4.28 Opción autores.**

Al momento no tenemos ingresado ningún dato en la base de datos, agregamos un nuevo Autor:


**Figura 4.29 Registrar un nuevo autor.**

Luego visualizamos nuevamente la lista de autores:


**Figura 4.30 Visualizar nuevo registro ingresado.**

De igual manera editamos un registro haciendo clic sobre el código del autor.


**Figura 4.31 Editar un registro.**

Guardamos el registro editado:

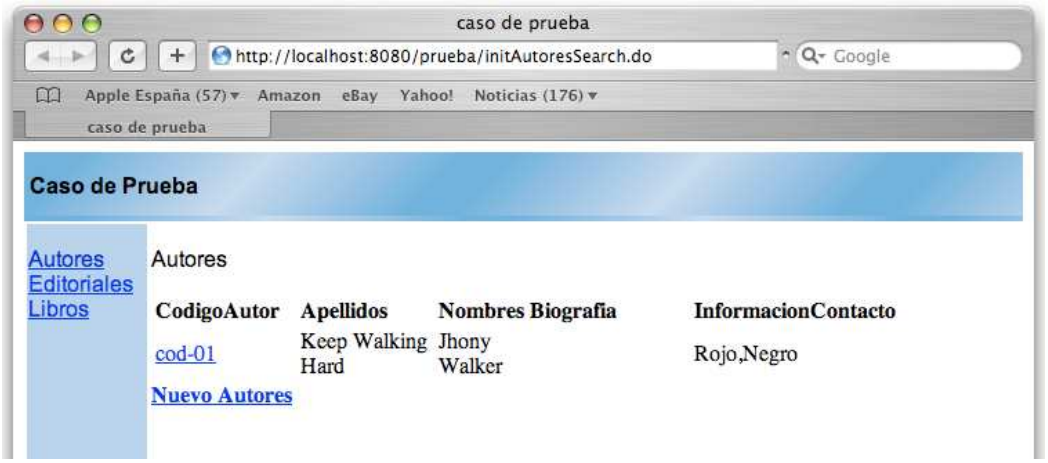



**Figura 4.32 Visualizar datos Editados.**

Para borrar el registro, seleccionamos el registro a eliminar y damos clic en borrar:


**Figura 4.33 Eliminar un registro.**

Ahora podemos ver que el registro ha sido borrado correctamente.


**Figura 4.34 Comprobar que se eliminó correctamente el registro.**

## 4.3 EVALUACIÓN DE RESULTADOS

Una vez concluidas las pruebas de utilización práctica de la herramienta, es necesario determinar los resultados obtenidos y evaluarlos en relación a la consecución de los objetivos planteados.

A continuación se evaluarán estos resultados en base a criterios como: facilidad de uso, precisión del código generado, estabilidad de la aplicación y versatilidad en la presentación de la información al usuario.

### 4.3.2 FACILIDAD DE USO

La herramienta posee una interfaz sencilla e intuitiva, que está orientada a usuarios técnicos, que con una sencilla explicación de la herramienta o leyendo la ayuda de la misma, podrían usarla sin ningún inconveniente.

### 4.3.2 PRECISION DEL CODIGO GENERADO

Los archivos generados contienen solo el código necesario para realizar las tareas de lectura, creación, edición y eliminación de datos; no contienen lógica

del negocio, esta deberá ser implementada posteriormente, de acuerdo a la necesidad de la aplicación web.

### 4.3.3 ESTABILIDAD

Durante la realización de las pruebas se detectaron varios errores, los mismos que fueron corregidos a nivel de código fuente. Después de estas correcciones la aplicación se desempeñó de forma estable. En caso de errores, el aplicativo muestra un log que facilita la retroalimentación y la depuración de errores.
Entre los errores detectados describimos los principales:

Al obtener los meta datos de la base de datos es necesario hacer una conversión de tipos sql a tipos java, esto no fue considerado en un principio, lo que provocaba que los tipos java se generen como tipo sql, la solución que se implemento fue crear equivalencias sql a java en una clase que se encargue de la conversión:

```
public String getType(Columna columna) {
        if (columna.getSqlType() == null) {
            return "";
        }
        String sqlType = columna.getSqlType().toUpperCase();
        int decimales = columna.getDecimales();
        int precision = columna.getPrecision();
        if (sqlType == null)
            return "";
        if (sqlType.equals("DATE"))
            return "java.sql.Date";
        if (sqlType.equals("BOOL"))
            return "java.lang.Boolean";
        if (sqlType.equals("FLOAT"))
            return "java.lang.Float";
        if (sqlType.equals("DOUBLE"))
            return "java.lang.Double";
        if (sqlType.equals("FLOAT(7)"))
            return "java.lang.Float";
        if (sqlType.equals("FLOAT8"))
            return "java.lang.Double";
        if (contiene(sqlType, "NUMERIC") || contiene(sqlType, "NUMERIC"))
            return "java.math.BigDecimal";
```

```
        if (sqlType.equals("BYTEA")) {

            return "java.sql.Blob";
        }
        if ((sqlType.indexOf("TIMESTAMP") != -1) ||
(sqlType.indexOf("DATETIME") != -1))
            return "java.sql.Timestamp";
        if (sqlType.equals("TIME"))
            return "java.sql.Time";
        if (contiene(sqlType, "TINYINT"))
            return "java.lang.Byte";
        if (contiene(sqlType, "SMALLINT"))
            return "java.lang.Short";
        if (contiene(sqlType, "BIGINT"))
            return "java.lang.Long";
        if (contiene(sqlType, "DECIMAL"))
            return "java.math.BigDecimal";
        if (contiene(sqlType, "BLOB"))
            return "java.sql.Blob";
        if (contiene(sqlType, "SERIAL"))
            return "java.lang.Long";
       if (contiene(sqlType, "IDENTITY"))
           return "java.lang.Long";
        if (sqlType.equals("NUMBER") || sqlType.equals("INT") ||
                sqlType.equals("YEAR") ||
                sqlType.indexOf("INT") > -1) {
            if (decimales == 0) {
                if (precision == 0) {

                    return "java.lang.Integer";
                }
                if (precision <= 2) {

                    return "java.lang.Integer";
                }
                if (precision <= 5) {
                    return "java.lang.Integer";
                }
                if (precision <= 9) {

                    return "java.lang.Integer";
                }
                if (precision <= 18) {
                    if (sqlType.indexOf("INT") != -1) {
                        return "java.lang.Integer";
                    }

                    return "java.lang.Long";
                } else {
                    return "java.math.BigDecimal";
```

```
            }
        }
        if (precision + decimales <= 12) {

            return "java.math.BigDecimal";
        }
        if (precision + decimales <= 64) {
            return "java.lang.Double";
        } else {
            return "java.math.BigDecimal";
        }
    }

    if (sqlType.indexOf("CHAR") > -1) {
        return "java.lang.String";
    }
    if (sqlType.indexOf("TEXT") > -1) {
        return "java.lang.String";
    }

    return "java.lang.String";
}
```

Antes de solucionar este error obteníamos el siguiente resultado:

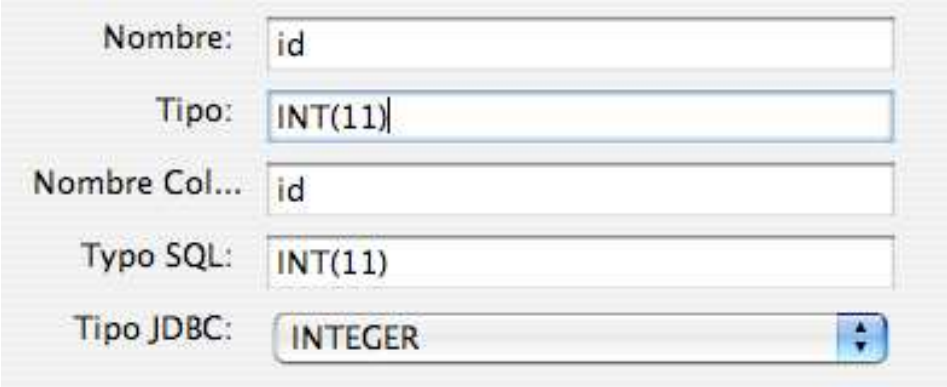



Figura 4.35 Antes de corregir error de tipo de datos.

En Tipo: INT(11), esta incorrecto, debería ser java.lang.Integer.
Luego de corregido el error se muestra correctamente:

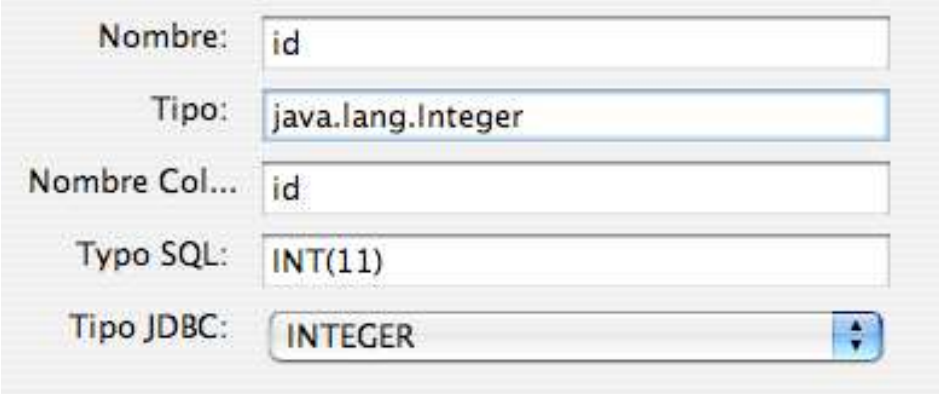



Figura 4.36 Después de corregir error de tipo de datos.

En el desarrollo inicial de la aplicación se utilizó referencias fijas a archivos y directorios utilizados para configuración de genj y generación de código, lo que provocaba que cuando la aplicación se cambiaba de directorio dejará de funcionar correctamente, la solución a este problema fue modificar el código y cambiar las referencias fijas a referencias relativas a la aplicación.

```
private static final File PLANTILLA_DEFAULT = new
File("/Users/kleber/Proyectos/genJ/templates/genj");
```

En la sección de código mostrada se muestra una referencia a un directorio, que al momento de generar la aplicación desde otra ubicación producía el siguiente error:

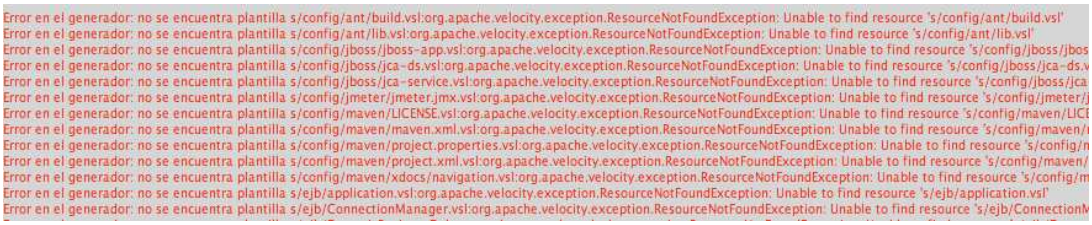



Figura 4.37 Error producido por referencias relativas

El código corregido para usar referencias relativas a alos directorios:

```
private static final File PLANTILLA_DEFAULT = new File("templates/genj");
```

### 4.3.4 PORTABILIDAD

La herramienta fue desarrollada en java, en ambientes Linux y Mac OSX, y fue probada en Windows, Linux y Mac OSX, por lo que es completamente portable a cualquiera de los sistemas operativos mencionados, siempre y cuando cumplan los requisitos mínimos de hardware y software especificados.

En cuanto al código generado, de igual manera puede ser compilado en cualquiera de los sistemas operativos indicados y con los mismos requisitos.

# CAPITULO 5. CONCLUSIONES Y RECOMENDACIONES

## 5.1 CONCLUSIONES

- Se puede ahorrar tiempo en la codificación de aplicaciones gracias al uso del generador, debido a que definiendo los parámetros adecuados genera las múltiples capas de la arquitectura planteada sin mayor esfuerzo para el programador.

- La herramienta previene errores de codificación que son causados al momento en que los programadores copian y pegan el código escrito, muchas veces sin haber echo pruebas exhaustivas sobre el mismo.

- Ayuda a implementar estándares de codificación debido a que la mayor parte del código esta generado en base a plantillas donde están definidos los estándares de desarrollo. Gracias a esto los desarrolladores obtienen código estándar, el cual representa un ejemplo claro de cómo aplicar los estándares a la hora de codificar.

- Permite aplicar una arquitectura predeterminada a un proyecto sin mayor problema debido a que genera cada capa del modelo y el paso completo de datos entre capas desde la interfaz del usuario hasta la base de datos.

## 5.2 RECOMENDACIONES

- Es recomendable extender este modelo de generación de código por plantillas hacia otras tecnologías por ejemplo .NET, debido a que el modelo es portable, lo que se requiere para extender el modelo a esta tecnología es construir nuevas plantillas con la sintaxis de codificación específica para .NET.

- Cuando se diseña un sistema es recomendable plantear un generador de código que ayude a optimizar el tiempo de los desarrolladores. Se puede construir generadores para la mayoría de arquitecturas solo es necesario definir las plantillas de generación.

- Cuando se construyen este tipo de generadores en necesario incluir en las plantillas la mayor parte de funcionalidad que requiera la aplicación. Ejemplo: manejo genérico de errores, bloques de código que permitan manejar transaccionalidad en el sistema, auditoria. Estas tareas suelen implementarse al final de la construcción de los sistemas lo que representa modificación de código que ya funciona con el peligro de modificaciones erróneas, incluyendo estos bloques desde el inicio podemos ahorrar valioso tiempo y evitar problemas al momento de cambiar el código que ya esta funcionando.

## Bibliografía


1 APACHE , Apache Struts, http://struts.apache.org/ Ultimo Acceso: Junio 2006.

2 Alan Davis, La Especificación de Requerimientos de Software, http://www.altera.com/produc Ultimo Acceso: Junio 2006

3 bitpipe , Systems Testing, http://www.bitpipe.com/rlist/term/Systems-Testing.html, Ultimo Acceso: Junio 2006.

4 Carol Silwa, ".Net vs. Java: Five Factors to Consider", http://www.computerworld.com/developmenttopics/development/story/0,10801,71221,00.htm 2007

5 IEEE Recommended Practice for Software Requirements Specification. AN-SI/IEEE std. 83(

6 José Díaz, Estándares de Codificación Java, http://www.itprofessionals.com.pe/home/index.php?option=com_content&task=view&id=39&I Junio 2006.

7 MICROSOFT, Métodos Heterodoxos en Desarrollo de Software, http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/heterodox.asp, Ultimo Ac

8 PABLO FIGUEROA, Metodología de desarrollo de software Orientado por Objetos, http://www.cs.ualberta.ca/~pfiguero/soo/metod/, Ultimo Acceso: Junio 2006.

9 RATIONAL, Rational Unified Proces, http://www.augustana.ab.ca/~mohrj/courses/2000.winter/csc220/papers/rup_best_practices/r Ultimo Acceso: Febrero 2007

10 RAHUL TYAGI, Java Programming: Explore the advanced coding features of JDK 1.5, http:// 6370-5147404.html, Ultimo Acceso: Junio 2006.

11 SUN MICROSYSTEMS, Model-View-Controller, http://java.sun.com/blueprints/patterns/MVC Junio 2006

12 SUN MICROSYSTEMS, Web-Tier Application Framework Design http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web- Junio 2006

13 SUN MICROSYSTEMS , New Features and Enhancements J2SE 5.0, http://java.sun.com/j2se/1.5.0/docs/relnotes/features.html, Ultimo Acceso: Junio 2006

14 Terry Quatrani, Introduction to the Unified Modeling Language, ftp://ftp.software.ibm.com/software/rational/web/whitepapers/2003/intro_rdn.pdf, Ultimo Acce

**ANEXO 1. Diagrama de clases definitivo.**