Universidad Autonóma Metropolitana
Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería
Ingeniería en Computación

PROYECTO TERMINAL EN INGENIERÍA EN COMPUTACIÓN II

Reporte Final

Reingeniería del sistema de consulta y diagnóstico médico veterinario SICODIVE

Guillermo Monroy Rodríguez

Asesora: Dra. Ma. Lizbeth Gallardo López
Profesor Investigador Asociado
en la Universidad Autónoma Metropolitana,
Azcapotzalco, Departamento de Sistemas.

Trimestre 12-I
28 de Abril de 2012

Índice

1. Introducción **2**
1.1. Objetivo general del proyecto 3
1.2. Objetivos particulares 3

2. Antecedentes **4**

3. Justificación del proyecto **5**

4. Modelo cíclico **5**

5. Reingeniería de SICODIVE **7**
5.1. Análisis de Inventario 7
5.2. Reestructuración de documentos 11
5.3. Ingeniería inversa 12
5.4. Reestructuración de código 12
5.5. Reestructuración de datos 12
5.6. Ingeniería directa 15

6. Resultados del proyecto **41**

7. Conclusiones y perspectivas **43**

1. Introducción

El Instituto de Ingeniería de software (*SEI*) define que el término Reingeniería como la transformación sistemática de un sistema existente para mejorar su calidad en sus operaciones, su funcionalidad, su rendimiento y su evolución [1]. Una organización decide realizar una reingeniería evaluando el costo-beneficio de llevarla a cabo; los criterios considerados en la toma de decisión incluye: el costo monetario, el esfuerzo requerido, los riesgos para la misma organización [1].

[1]Sistema heredado se refiere a la noción de Legacy Information System o LIS. Se llama *legacy* a los sistemas antiguos que funcionan en una organización y que no pueden ser remplazados por diversos motivos, entre ellos se encuentran: es demasiado caro para ser reemplazado, el sistema hace lo que la empresa requiere.

El presente proyecto toma como sistema heredado a SICODIVE[2] para realizar una reingeniería, empleando el modelo cíclico [?, somerville], el cual consiste en 6 etapas:1) análisis de inventario, 2) restructuración de la documentación, 3) ingeniería inversa, 4) reestructuración del código, 5) restructuración de datos y 6) ingeniería directa. La reingeniería tiene por objetivos proporcionar extensibilidad y seguridad al sistema siguiendo las tendencias actuales (el uso de *frameworks*) en los sistemas de información.

Como resultado de la reingeniría se obtuvieron las siguientes mejoras en el sistema: 1) la arquitectura, originalmente basada en *MVC*[3], ha pasado a una arquitectura basada en capas de procesamiento[4], 2) se modificó el modelo de datos para que soporte: bitácoras, control de usuarios y securización del acceso a datos, así como el acceso de los usuarios. El sistema resultante de la reingeniería se le denomina SICODIVEv2.

1.1. Objetivo general del proyecto

Realizar la reingeniería de SICODIVE con el propósito de proporcionar extensibilidad y seguridad al sistema.

1.2. Objetivos particulares

1. Analizar el sistema heredado SICODIVE.
2. Rediseñar el sistema heredado; es decir, cambiar la arquitectura y la tecnología de implementación.
3. Evaluar y escoger la tecnología de información más adecuada para la construcción del sistema rediseñado SICODIVEv2.
4. Implementar el sistema SICODIVEv2.
5. Documentar el sistema SICODIVEv2.
6. Realizar y documentar pruebas del sistema SICODIVEv2.

[2]Sistema que implementa la solución para las necesidades del cliente poseedor de una clínica veterinaria

[3]Patrón que separa en Modelo, Vista y Controlador

[4]n-Layers

2. Antecedentes

SICODIVE es un sistema WEB que apoya al médico en: 1) la gestión de consultas médicas y 2) el diagnóstico de sus pacientes mediante la sugerencia de tratamientos. El sistema permite que el médico realice ambas actividades. SICODIVE fue diseñado para implemenar las siguientes funcionalidades:

1. Gestión de Elixir: Realiza operaciones *CRUD*[5] con los Elixires de flores de Bach[6].

2. Sugerencia de Tratamientos: Realiza una búsqueda para encontrar uno o varios elixires que coincidan con los síntomas elegidos por el médico veterinario.

3. Gestión de Expedientes: realiza las operaciones *CRUD* sobre las consultas médicas previas y la consulta actual del paciente.

4. Gestión de Pacientes: Realiza operaciones *CRUD* sobre la información del paciente.

5. Gestión de Usuarios: Realiza operaciones *CRUD* sobre la información de los usuarios y asigna uno o varios roles para cada usuario.

6. Gestión de Citas: Realiza una agenda para cada médico veterinario, de tal forma que las citas médicas no se traslapen.

SICODIVE fue diseñado y construido por un equipo de estudiantes de la Licenciatura Ingeniería en Computación de la Universidad Autónoma Metropolitana - Azcapotzalco, a saber: Carlos Alejandro Vega Quiroz, Jorge Alberto Bautista Hernández y Guillermo Monroy Rodríguez. El sistema se desarrolló en el marco de la UEA Ingeniería del software, atendiendo los requerimientos de una pequeña clínica veterinaria. El proyecto se desarrolló en 12 semanas, de las cuales 5 fueron empleadas para la implementación; lo anterior implicó que las funcionalidades gestión de elixir y gestión de citas

[5] *CRUD* son las siglas en inglés para las operaciones de base de datos Create, Retrieve, Update y Delete.

[6] Las Flores de Bach son una serie de esencias naturales utilizadas para tratar diversas situaciones emocionales, como miedos, soledad, desesperación, estrés, depresión y obsesiones. Fueron descubiertas por Edward Bach entre los años 1926 y 1934.

no fueran implementadas, aunque a nivel de diseño se cuenta con la especificación.

3. Justificación del proyecto

Las motivaciones de nuestro proyecto terminal son:

1. Experimentar un proceso de reingeniería, concretamente queremos realizar:

   a) cambios en la arquitectura para que los módulos del sistema mantengan un bajo acoplamiento;

   b) agregar funcionalidades de tal manera que el sistema se adapte a las necesidades actuales de sus usuarios;

   c) corregir los errores del diseño original, evitar hardcoding[7], evitar manejo inadecuado de excepciones, evitar acoplamiento entre clases y/o capas de software.

2. Establecerlo como caso de estudio para otros estudiantes de la Licenciatura Ingeniería en Computación, donde se observe claramente el análisis de un sistema de software heredado. El tema de reingeniería no es abordado en las UEA's de la Licenciatura; sin embargo, es una necesidad en el ambiente laboral.

3. Adquirir la experiencia de analizar un sistema heredado con miras a diseñar en un futuro sistemas que cumplan cabalmente los criterios de calidad *FURPS*.

4. Modelo cíclico

Para realizar la reingeniería de SICODIVE se aplicó el modelo cíclico (ver figura 1) porque es un modelo claro y sencillo. Este modelo define la seis actividades siguientes:

[7]Hardcoding se refiere a la práctica de desarrollo de software donde se incrustan cadenas de configuración, entradas para el programa o sistema directamente en el código fuente del sistema que se compila u otro objeto ejecutable en vez de tomar esos datos de alguna fuente externa tal como un archivo de configuración, una base de datos, etc.

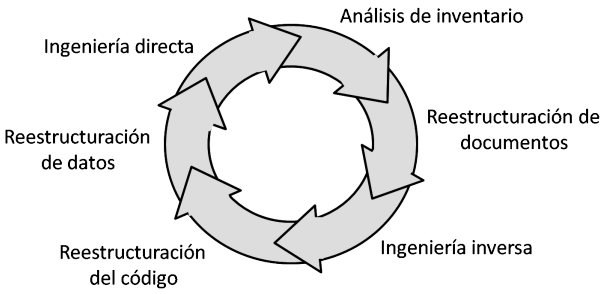


Figura 1 *Modelo cíclico de reingeniería*

- Análisis de inventario. Consiste en elaborar una lista con los componentes que conforman el sistema. Los componentes candidatos a la reingeniería aparecen cuando se ordena esta información en función de su importancia para la organización, la mantenibilidad del sistema actual, su logevidad, entre otros criterios.

- Restructuración de documentos. En esta actividad se plantean tres opciones: 1) conservar la documentación original del sistema heredado, 2) documentar únicamente los cambios que se realizaron o 3) realizar una documentación completa en el caso de que el sistema sea crítico para la organización.

- Ingeniería inversa. Consiste en comprender el funcionamiento interno del sistema con el fin de volverse expertos en él o alguna de sus partes.

- Reestructuración del código. Consiste en analizar el código fuente en busca de su reutilización; para luego reestructurarlo manualmente ó a través de una herramienta automática; posteriormente, se comprueba que no existan anomalías; y finalmente, se debe generar o actualizar la documentación del código.

- Reestructuración de datos. Consiste en analizar la arquitectura de datos actual; para luego extenderlo con el o los modelos de datos que responderán mejor a las necesidades del sistema. Por ejemplo: se identifican los estructuras de datos y, a continuación, se comprueba que dicha estructura contenga íntegramente la información requerida.

- Ingeniería directa. También se le denomina renovación o reclamación [6]. Consiste no solamente en recuperar la información de diseño de un software ya existente; además, utiliza esta información en un esfuerzo por mejorar su calidad global. En la mayoría de los casos, el software procedente de una reingeniería vuelve a implementar las funcionalidades del sistema existente, y añade nuevas funcionalidades o mejora algunos de los criterios de calidad (*FURPS*).

5. Reingeniería de SICODIVE

5.1. Análisis de Inventario

Presentamos a continuación el análisis de inventario. Primero se describe al componente candidato a ser redefinido y posteriormente se justifica la razón por la cual conveniene redefinirlo.

Arquitectura

La arquitectura de SICODIVE está basada en el patrón de diseño *MVC* (ver figura 2) y se encuentra organizada por: *a*) un modelo de datos, *b*) un gestor o controlador de peticiones, y *c*) una capa de presentación.

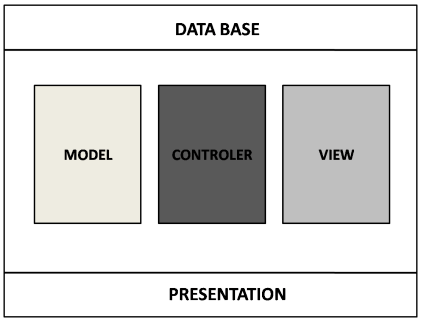


Figura 2 *Modelo-Vista-Controlador*

La arquitectura de SICODIVE, es candidata a reingeniería. Algunos módulos que componen el sistema están acoplados y tienen baja cohesión; es decir, las responsabilidades entre los módulos no están bien delimitadas; por

ejemplo, el modelo (*Model* de *MVC*) incluye operaciones sobre la presentación (*Viewer*); por lo tanto, se sugiere cambiar del modelo *MVC* a un modelo de 3 capas (*3-Layers*):

1. Capa de acceso a datos (*Data Access Layer - DAL*).
2. Capa de negocio (*Bussiness Layer - BL*).
3. Capa de presentación (*Presentation Layer - PL*).

Con el uso de estas capas aprovechamos los beneficios de las frameworks: *Hibernate* para ORM[8] y *Struts*[9] para la presentación.

La arquitectura 3-capas mejorará la arquitectura basada en el modelo *MVC* de SICODIVE al forzar al desarrollador a definir mejor las responsabilidades de cada conjunto de clases, lo que nos permite tener un bajo acoplamiento y una alta cohesión entre ellas.

Componentes

Los componentes de SICODIVE son: 1) `Dashboard`, 2) `Login`, 3) `Validacion`, 4) `baseDatos` y 5) `Sistema`.

1. `Dashboard`: En este paquete se encuentran los servlets que hacen el manejo de las peticiones al sistema SICODIVE (desde el cliente al servidor de aplicaciones), a saber:
   - *a*) ManejadorDashboard
   - *b*) ManejadorCitas
   - *c*) ManejadorElixir
   - *d*) ManejadorExpediente
   - *e*) ManejadorPaciente
   - *f*) ManejadorSintoma
   - *g*) ManejadorTratamiento
   - *h*) ManejadorUsuario

[8] Object-Relational Mapping

[9] *Apache Struts* está basado en el patrón *MVC* y es empleado en conbinación con con otras tecnologías de ORM [2] para dar una clara separación entre capas del sistema

Cada uno de estos sevlets realiza las acciones necesarias para satisfacer los requerimientos plasmados en los casos de uso de SICODIVE (para mayor detalle consultar la documentación original del sistema).

2. `Login`: Está conformado de dos servlets para manejar las peticiones de ingreso al sistema SICODIVE, así como de finalización de la sesión.

3. `Validacion`: Este paquete contiene clases que realizan las operaciones necesarias para las validaciones de los campos que se ingresan en los formularios WEB.

4. `baseDatos`: Este paquete contiene las clases correspondientes a cada entidad del modelo de datos; estas clases realizan las operaciones *CRUD* sobre los datos a través de los manejadores contenidos en los paquetes `Dashboard` y `Login`.

5. Sistema: Este paquete se conforma de las clases que contienen los datos, es decir, son las entidades de dominio del sistema a ser empleadas por los demás paquetes.

Los componentes antes descritos son candidatos a reingeniería, porque las responsabilidades de cada servlet se encuentran acopladas con las operaciones que afectan el modelo de datos; por ejemplo, las operaciones de validación de datos entre componentes se realizan en el mismo servlet; sería deseable tener un componenete cuya responsabilidad sea la de validar los datos antes de llegar al servlet. El control de transiciones y peticiones, así como el control de las respuestas hacia la presentación del sistema, se realiza en el servlet; sería pertinente tener otro componente dedicado a controlar las transiciones.

Funcionalidades

Las funcionalidades de SICODIVE son:

1. Sugerencia de Tratamiento. Se sugiere un conjunto de tratamientos en base a los síntomas que se le indican al sistema SICODIVE.

2. Gestión de Expedientes. Gestiona (alta, consulta y actualización) de expedientes para pacientes y agrega nuevas consultas médicas a un expediente existente.

3. Gestión de Pacientes. Gestión (alta, actualización y consulta) de pacientes.

4. Gestión de Usuarios. Gestión (alta, baja, actualización y consulta) de los usuarios; además asocia uno o más roles (predefinido como catálogo dentro de la base de datos: 1-Administrador, 2-Médico y 3-Paciente) del sistema.

5. Gestión de Citas (no implementada). Debe gestionar citas (altas, bajas, actualizaciones y consultas) apoyándose de un calendario y el horario disponible para cada médico.

6. Gestión de Elixires (no implementada). Debe gestionar los elixires (altas, bajas, actualizaciones y consultas) del catálogo de elixires con sus respectivos síntomas asociados, así como las instrucciones del modo de empleo.

Las funcionalidades del sistema son candidatas a reingeniería con el propósito de proporcionar al sistema las características de mantenibilidad y seguridad; para lograrlo algunas de estas funcionalidades serán extendidas, algunas serán eliminadas y otras más serán agregadas.

Modelo de datos

La base de datos relacional de SICODIVE consta de las siguientes tablas: *a*) Rol, *b*) Cita, *c*) Paciente, *d*) Usuario, *e*) Clinica, *f*) Consulta, *g*) Sintoma, *h*) Receta, *i*) SintomaPresente y *j*) Elixir.

Este modelo de datos (ver figura 3) es candidato a reingeniería porque necesitará soportar las nuevas funcionalidades, así como alojar los datos de auditoría (registro de acceso de usuarios, restricción de intentos de acceso, parametrización del sistema, configuración dinámica del sistema), e inteligencia de negocio.

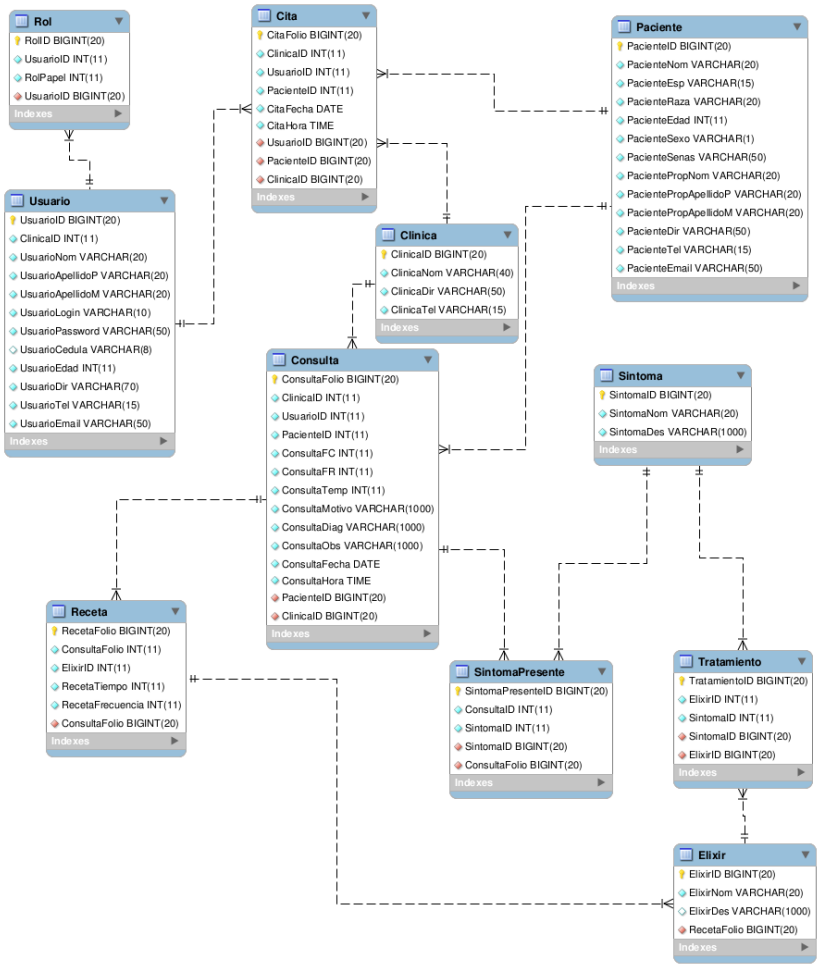


Figura 3 *Base de datos relacional de SICODIVE*

5.2. Reestructuración de documentos

La documentación de SICODIVE se ha consevado (ver Anexo: *Documentación SICODIVE*) y se agregaron los artefactos referidos en la sección 5.6 para el caso de uso de ingreso al sistema, el cual no fue planteado en SICODIVE; y adicionalmente, se especifican operaciones relacionadas con la seguridad del manejo de sesiones para los usuarios como son: generación de contraseña aleatoria en el alta de usuarios y modificación de contraseña en el primer ingreso.

5.3. Ingeniería inversa

No se realizó como tal la ingeniería inversa sobre las funcionalidades del sistema SICODIVE porque se cuenta con su documentación, a saber: documento de visión, documento de requerimientos, documento de diseño y manual de usuario; además se cuenta con la experiencia de haber implementado parte del mismo sistema, por lo tanto se conoce el funcionamiento, la tecnología y el entorno de desarrollo empleados, así como los casos de pruebas y la puesta en producción de SICODIVE.

Es conveniente destacar que SICODIVE está implementado mediante el uso de *JSP*, Servlets y el manejador de base de datos *MySQL* (versión 5) proporcionado por *Apache Foundations*.

5.4. Reestructuración de código

No se realizó reestructuración de código porque al cambiar la arquitectura y emplear las frameworks de *Hibernate* y *Struts*, el código de SICODIVE fue inutilizable. En efecto, este sistema tiene clases que hacen las veces de entidades contenedoras de los datos que se extraen de la base de datos; sin embargo, la conexión y las operaciones (inserción, consulta, borrado o actualizado) se realizan empleando una clase encargada de gestionar las conexiones y trasacciones correspondientes. En la arquitectura de 3 capas, el framework *Hibernate* es el encargado de hacer un mapeo de los datos contenidos en tablas de la base de datos a objetos; además, maneja las sesiones, transacciones, así como las operaciones *CRUD* de manera cuasi-trasparente para el desarrollador.

5.5. Reestructuración de datos

Empleamos *ORM*, el cual facilita el manejo de transacciones y operaciones atómicas sobre la base de datos. La implementación de ORM [2] la realizamos a través de la framework *Hibernate* para la tecnología *JAVA*. Una ventaja de *Hibernate* es la posibilidad de cambiar de manejador de base de datos (actualmente *MySQL 5.5 Server*) con solo modificar los archivos de configuración correspondientes sin necesidad de recompilar la aplicación.

Las nuevas funcionalidades, el criterio de securización, parametrización y modularidad que se requiere para el sistema SICODIVEv2, el modelo de datos es el mostrado en la figura 4.

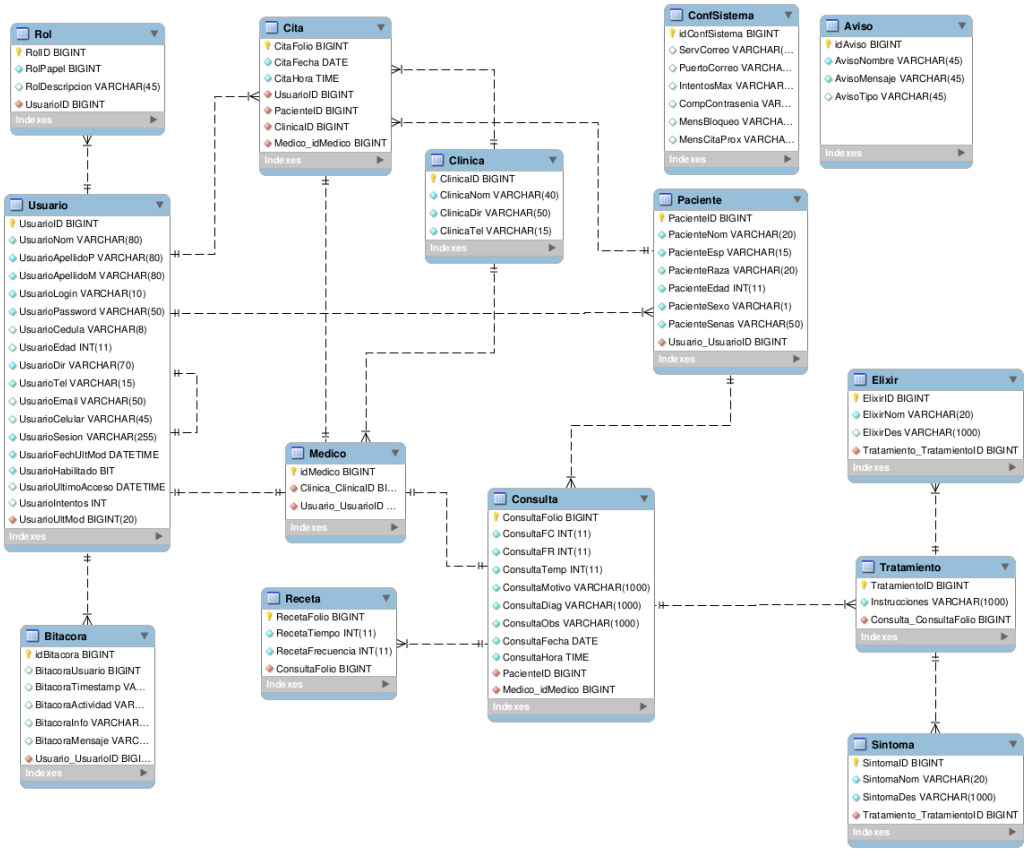


Figura 4 *Base de datos relacional de SICODIVEv2*

Las siguientes tablas fueron creadas para alojar los parámetros de configuración del sistema.

1. Bitacora
2. Aviso
3. ConfSistema
4. Controlador

Las siguientes tablas fueron modificadas en sus atributos

1. Usuario

2. Rol

La información contenida en las tablas de las dos listas antes mencionadas permiten al sistema:

a) el control de acceso de los usuarios, *b)* generar contraseña aleatoria tras reestablecerla por un usuario con rol de administrador, *c)* bloquear acceso de usuarios que ya se encuentran autenticados en el sistema, *d)* bloquear accesos de usuarios tras n intentos fallidos (los intentos fallidos son un parametro contenido en la tabla `ConfSistema`), *e)* tras dar de alta a un nuevo usuario se genera una contraseña de uso único que se deberá, obligatoriamente, cambiar en el primer acceso del usuario, *f)* llevar bitacora de actividades, *g)* configurar mensajes o avisos del sistema para los usuarios, *h)* configurar parametros de correo electrónico para envío de avisos y mensajes y *i)* gestionar los roles de usuarios del sistema.

Se eliminó la tabla `SintomaPresente` dado que los datos que contenía se pueden obtener a partir de las nuevas relaciones de las siguientes tablas:

1. Consulta
2. Tratamiento
3. Sintoma
4. Elixir

En el modelo anterior se tenía la llave primaria de los registros de la tabla `Clinica` como atributo en las tablas que se relacionaban con ella; si en alguna operación se eliminaran los registros de la tabla `Clinica`, se perdería la integridad relacional de las tablas que tuviesen dicho atributo. Para resolver este problema se creó la tabla `Medico` que sirve como relación de las tablas que se enumeran a continuación:

1. Usuario
2. Citas
3. Consulta
4. Clinica

En la tabla `Paciente` se eliminaron algunos atributos ya que éstos eran redundantes con los contenidos en la tabla `Usuario` en el modelo anterior. La tabla `Paciente` ahora tiene una relación con la tabla `Usuario`.

5.6. Ingeniería directa

En esta etapa de la metodología se realizaron las siguientes actividades: *1)* Reestructuración de funcionalidades, *2)* Definión de la arquitectura, *3)* Rediseño (diagrama de casos de uso y diagrama de clases) e *4)* Implementación del sistema SICODIVEv2.

Reestructuración de funcionalidades

SICODIVEv2 está conformado por las siguientes funcionalidades que son análogas a las de SICODIVE:

1. Gestión de Elixires: El sistema debe contar con un catálogo de elixires que serán relacionados con síntomas y tratamientos.

2. Gestión de Citas: El sistema debe contar con un gestor de citas que realiza las operaciones *CRUD* relacionadas con pacientes y médico.

3. Gestión de Expedientes: El sistema debe gestionar los expedientes de cada paciente. El expediente incluye: consultas médicas, datos relacionados con el médico y las citas programadas, así como el historial médico del paciente denominado Expediente.

4. Gestión de Pacientes: El sistema debe ser capaz de realizar las operaciones *CRUD* sobre la información del paciente.

Las siguientes funcionalidades fueron reestructuradas, creadas o extendidas para el sistema SICODIVEv2:

1. Gestión de Tratamientos: El sistema debe contar con un catálogo de tratamientos (Elixires de flores de Bach) relacionados con los síntomas típicos en animales. Este módulo debe incluir un mecanismo de filtrado sobre los elixires, a partir de los síntomas identificados por el médico veterinario, luego de la auscultación de su paciente. Esta funcionalidad extiende la funcionalidad original del SICODIVE descrita en la documentación original `Sugerencia de tratamiento`.

2. Gestión de Usuarios: El sistema debe gestionar la información con operaciones *CRUD* de todos sus usuarios. La funcionalidad extiende la funcionalidad de SICODIVE para mejorar la seguridad del sistema a

través de: *a)* control de acceso de los usuarios, *b)* generación de contraseña aleatoria tras reestablecerla por un usuario con rol de administrador, *c)* bloqueo de acceso con credenciales de usuarios que se encuentren autenticados en el sistema, *d)* bloqueo de accesos de usuarios tras n intentos fallidos(los intentos fallidos son un parámetro configurable del sistema), *e)* registro de actividades de cada usuario en una bitácora, *f)* asociación de los roles a los usuarios del sistema.

3. Gestión de roles: A través del manejo de roles se definen las acciones permitidas en el sistema; algunos ejemplos de roles son: paciente, médico, recepcionista, administrador. Esta funcionalidad extiende la de SICODIVE para que los roles puedan ser gestionados y manejados por asignación.

4. Gestión de configuración de parametros del sistema. Mediante la modificación de los parámetros del sistema se modifica parte del comportamiento del sistema, a saber: el servidor de correo electrónico y la conexión al manejador de base de datos; ésto sin necesidad de recompilar y desplegar nuevamente la aplicación. Ésta es una nueva funcionalidad del sistema.

5. Gestión de bitácoras de uso: El sistema debe registrar bitácoras referentes al uso del sistema (excepciones, interrupciones de servicio, errores, etc.). A largo plazo estas bitácoras servirán para realizar análisis estadísticos el cual pueda apoyar en la toma de decisiones del negocio (business intelligence[10]). Ésta es una nueva funcionalidad del sistema.

[10]Se denomina inteligencia empresarial, inteligencia de negocios o BI (del inglés business intelligence) al conjunto de estrategias y herramientas enfocadas a la administración y creación de conocimiento mediante el análisis de datos existentes en una organización o empresa.

Definición de la arquitectura

La arquitectura del sistema SICODIVEv2 es de 3 capas (*n-Layers*, ver figura 5). Esta arquitectura favorece los criterios cualitativos *FURPS*; proporciona un mejor control durante el desarrollo y un mejor control de fallos del sistema estando en producción. Otra ventaja de esta arquitectura es que podremos hacer mantenimiento o adicionar funcionalidades de una manera más clara no solo para el desarrollador, sino también para analistas de software; reduciendo costos y recursos [5]. A continuación se describen las capas de la arquitectura de SICODIVEv2:

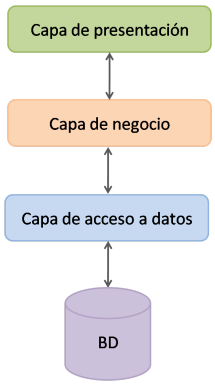


Figura 5 *Arquitectura de 3 Capas*

1. Base de datos. Se creó la base de datos descrita en la sección *Reestructuración de datos*. La nueva base de datos proporciona mayor flexibilidad, rendimiento y facilita su mantenimiento.

2. Capa de acceso a datos. Se creó la capa que tiene como responsabilidad manejar la persistencia de los datos, manejar las sesiones y transacciones respectivas; así como, el manejar las excepciones dando marcha atrás (*rollback*) a las operaciones de la transacción. Esta capa permite proteger el acceso no autorizado a la base de datos; además, mantiene comunicación con la capa de negocio y con el modelo de datos que persiste en la base de datos.

3. Capa de negocios. Se creó la capa de negocio responsable de llevar a cabo los procesos que satisfacen las funcionalidades y reglas de negocio

definidas en la sección *Reestructuración de funcionalidades*. Esta capa mantiene comunicación con la capa de acceso a datos y la capa de presentación.

4. Capa de presentación. Se creó la capa de presentación responsable de llevar el flujo entre las páginas WEB dinámicas de manera segura (haciendo manejo de sesiones, validación de operaciones, uso del protocolo *HTTPS*, entre otras). Esta capa mantiene comunicación con la capa de negocios y con el usuario del sistema.

Es importante destacar que SICODIVEv2 hace uso de *PKI*[11] para asegurar el canal de comunicación entre el servidor Web y el cliente [4] (navegador de páginas WEB) empleando el protocolo *HTTPS*[12]. Además, mediante el uso del framework *Struts* como controlador del flujo de la aplicación en la capa de presentación se garantiza la validación de los datos enviados por los formularios WEB desde la capa de presentación.

[11] *Public key infrastructure*
[12] *Hypertext Transfer Protocol Secure*

Rediseño de SICODIVE

A continuación se mostrará el modelo de casos de uso general de SICOVIDEv2, y sobre el CU6 - Gestionar Usuarios se describen los subcasos de uso de texto correspondientes; también se describe el CU0 - Ingresar al sistema y se explica su diagrama de robustez. Estos artefactos se presentan como ejemplo y constituyen la base para el resto de los casos de uso, ya que todos los gestores realizan las mismas operaciones: alta, baja, consulta y actualización sobre la entidad correspondiente al gestor.

Modelo de casos de uso

Para cada una de las funcionalidades, expuestas en la sección *5.6 Reestructuración de funcionalidades*, se crearon los casos de uso (ver figura 6) que atiende a los requerimientos correspondientes.

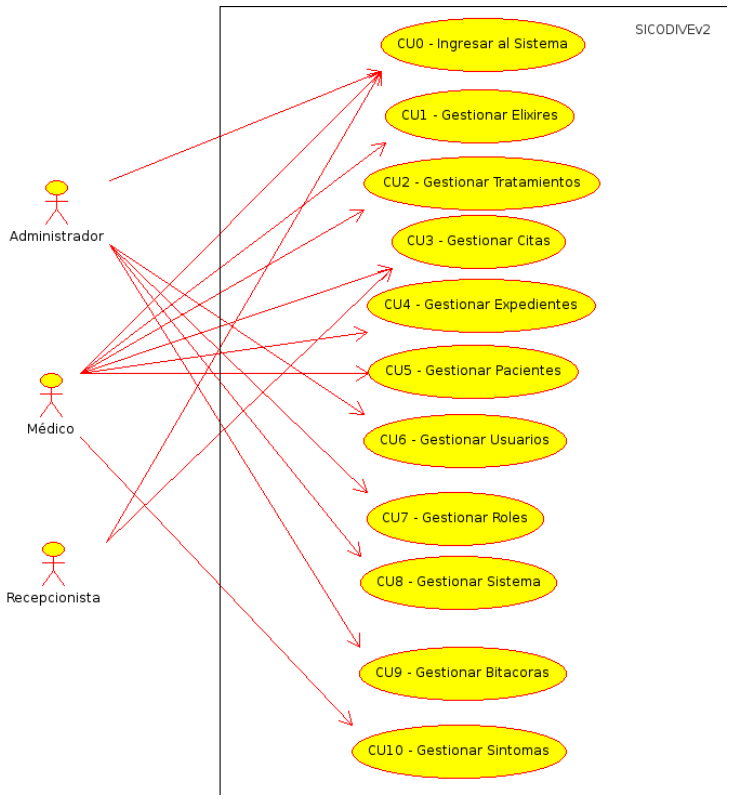


Figura 6 *Casos de Uso - SICODIVEv2*

Casos de uso (de texto)

Caso de uso 0: Ingresar al sistema
Actor principal: Administrador
Personal involucrado e intereses:

- Médico: quiere tener acceso al sistema de manera segura.
- Recepcionista: quiere tener acceso al sistema de manera segura.
- Administrador: quiere tener acceso al sistema de manera segura.

Precondiciones:

1. Que el usuario no este autenticado en el sistema.

Postcondiciones:

1. El usuario accede al sistema de manera segura.

Escenario principal de éxito:

1. El usuario ingresa sus datos en el sistema.
2. El sistema valida que no falten datos obligatorios.
3. El sistema busca que los datos proporcionados por el usuario sean correctos con los registrados.
4. El sistema registra al usuario y le da acceso a la página de bienvenida del sistema.
5. El usuario visualiza la página de bienvenida de SICODIVEv2.

Flujos alternativos

2a El usuario ingresa n veces los datos erroneamente.

- *a*) El sistema registra el intento y en el caso de ser n (parámetros de intentos máximos) bloquea el acceso para el usuario.
- *b*) El sistema notifica al usuario que su cuenta ha sido bloqueada.

2b Alguno de los datos introducidos por el administrador fueron incorrectos o no fueron ingresados.

- *a*) El sistema avisa al usuario del error y regresa al paso 1.

4c El usuario tiene una sesión registrada y abierta.

- *a*) El sistema le avisa al usuario que la sesión está siendo empleada.
- *b*) El sistema regresa a la al paso 2.

4d El usuario ingresa por primera vez al sistema.

- *a*) El sistema dirige al usuario para que cambie su contraseña.
- *b*) El usuario cambia su contraseña.
- *c*) El sistema da acceso a la página de bienvenida.

Frecuencia

1. Frecuente.

Caso de uso CU6: Gestionar Usuarios El caso de uso CU6 se encuentra conformado por los siguientes subcasos de uso:

CU6.1 Añadir usuario

CU6.2 Modificar usuario

CU6.3 Borra usuario

CU6.4 Buscar usuario

Casos de uso 6.1: Añadir usuario

Actor principal: Administrador
Personal involucrado e intereses:

- Médico: quiere tener acceso al sistema.
- Recepcionista: quiere tener acceso al sistema.
- Administrador: quiere que los usuario sean correctamente añadidos al sistema.

Precondiciones:

1. Que el Administrador este identificado y autenticado en el sistema.

Postcondiciones:

1. Se crea un nuevo usuario del sistema.

Escenario principal de éxito:

1. El administrador introduce los datos del nuevo usuario del sistema.
2. El sistema valida que los datos obligatorios estén presentes en el formulario.
3. El sistema almacena los datos y avisa al Administrador que se añadió un nuevo usuario correctamente.

Flujos alternativos

2a Alguno de los datos introducidos por el administrador fueron incorrectos o no fueron ingresados.

a) El sistema avisa al usuario del error y regresa al paso 1.

Frecuencia

1. Esporádica.

Casos de uso 6.2: Eliminar usuario

Actor principal: Administrador
Personal involucrado e intereses:

1. Administrador: quiere eliminar un usuario del sistema.

Precondiciones:

1. Que el Administrador esté identificado y autenticado en el sistema.
2. El usuario a eliminar debe estar registrado en el sistema.

Poscondiciones:

1. Un usuario se elimina del sistema.

Escenario Principal de éxito:

1. El Administrador busca e identifica en el sistema al usuario que desea eliminar.
2. El Administrador proporciona el identificador del usuario a eliminar.
3. El Sistema muestra la información del usuario del sistema que se desea eliminar.
4. El Administrador confirma la eliminación del usuario del sistema.
5. El Sistema elimina al usuario del sistema y avisa al Administrador que la operación fue exitosa.

Flujos alternativos

3a El usuario a eliminar no esta registrado en el sistema.

a) El Sistema avisa al Administrador y regresa al paso 1.

Frecuencia

1. Esporádica.

Casos de uso 6.3: Modificar usuario

Actor principal: Administrador
Personal involucrado e intereses:

1. Administrador: quiere modificar los datos de un usuario del sistema.

Precondiciones:

1. Que el Administrador esté identificado y autenticado en el sistema.
2. El usuario a modificar debe estar registrado en el sistema.

Poscondiciones:

1. Se modifican los datos de un usuario del sistema.

Escenario Principal de éxito:

1. El Administrador busca e identifica en el sistema al usuario que desea modificar.
2. El Administrador proporciona el identificador del usuario a modificar.
3. El Sistema muestra la información del usuario del sistema que se desea modificar.
4. El Administrador modifica los datos deseados del usuario del sistema.
5. El sistema valida que los datos sean correctos.
6. El Administrador confirma las modificaciones de los datos del usuario del sistema.
7. El Sistema modifica los datos del usuario y avisa al Administrador que la operación fue exitosa.

Flujos alternativos

5a Que los datos no sean correctos

a) El Sistema avisa al Administrador y regresa al paso 1.

Frecuencia

1. Esporádica.

Diagrama de robustez. Describiremos el caso de uso *CU0 - Ingresar al Sistema* apoyándonos en su diagrama de robustez (ver figura 7).

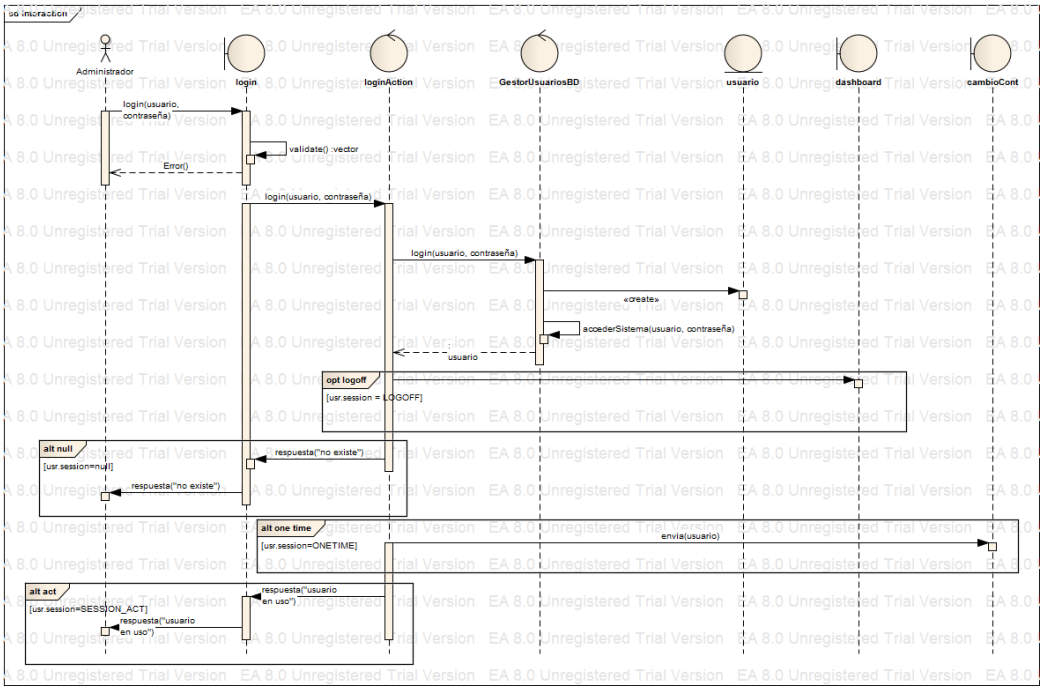


Figura 7 *Diagrama de robustez - accesso usuario*

Para el caso de uso *CU0 - Ingresar al Sistema*, el usuario proporciona su nombre de usuario y contraseña en el formulario WEB `login`, éste a su vez emplea un *Bean* denominado `LoginForm` (que es un componente de *Struts*); `LoginForm` valida los datos ingresados en el formulario, para el caso de que no se proporcionen los dos campos obligatorios, la petición o mensaje no será enviado al siguiente componente (`LoginAction`) y se notifica al componente `login` mediante un mensaje los campos que fueron dejados vacíos; para el

caso contrario los datos son enviados hacia el controlador `LoginAction`; los datos serán enviados, a su vez, a la *BL* (*Bussiness Layer*) mediante el controlador `GestorUsuarios`; para enviar los datos se crea un objeto entidad `usuario` y como atributos de la clase se asignan los valores provenientes del `LoginForm`; en el `GestorUsuarios` se llama a su método `accederSistema()` y éste emplea las reglas de negocio para el ingreso. De manera excluyente se realizará alguna de las siguientes acciones:

a) Si la sesión del usuario tiene el valor "LOGOFF", se le permite el acceso al `dashboard`.

b) Si la sesión del usuario tiene el valor "NULL", se regresa a **login** el mensaje de que el usuario no existe o que los datos son incorrectos.

c) Si la sesión del usuario tiene el valor "ONETIME", se le permite el acceso a `cambioCont` para que cambie la contraseña ya que es el primer ingreso que realiza al sistema.

d) Si la sesión del usuario tiene el valor "SESSION_ACT", se regresa a `login` el mensaje de que la sesión de usuario se encuentra en uso.

Gracias a la arquitectura *MVC* de Struts podemos tener un control del flujo de las páginas dinámicas en la capa de presentación. El componente controlador, `LoginAction` para nuestro caso, hace uso de la capa de negocio de nuestro sistema. La capa de negocio a su vez hace uso de la capa de acceso a datos empleando la clase `UsuarioDAO`.

Implementación

La implementación del sistema SICODIVEv2 se codificó en el entorno de desarrollo para *Java Netbeans 6.9*; SICODIVEv2 es un proyecto Web con los frameworks: *Struts* y *Hibernate* [3] empleando el manejador de base de datos *MySQL Server* sobre un sistema operativo GNU/Linux Debian 6.

La organización de los paquetes obedece a separar, tomando como base al paquete **com.olimpo**, los componentes en capas como se describe a continuación: *1)* Capa de Acceso a Datos, *2)* Capa de Negocio y *3)* Capa de Presentación. La estructura de la implementación del proyecto se muestra en la figura 8.

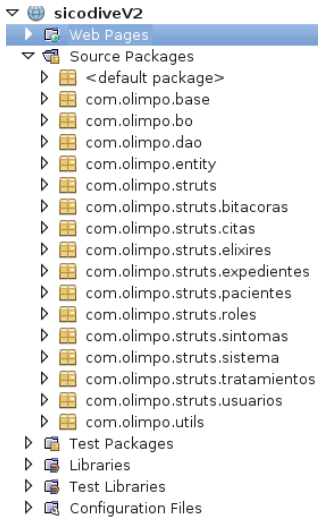


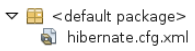
Figura 8 *Estructura de paquetes de SICOVIDEv2*

Estructura de paquetes de la implementación

A continuación describiremos los paquetes que conforman el proyecto.

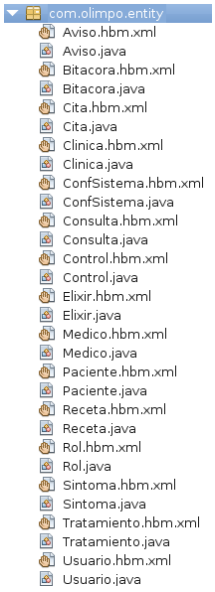
1. Capa de Acceso a Datos (*DAL*). Paquetes:

   a) **default package**. Este paquete contiene el archivo de configuración `hibernate.cfg.xml`; este archivo contiene los datos de conexión, el dialecto (lenguaje de consultas a un motor específico de base de datos, en este caso `MySQLDialect`, cuyos datos de conexión son protegidos por el *framework*; para mayor detalle sobre la protección de datos referirse a [4]), el usuario y su contraseña, así com algunos parámetros de configuración especiales (pool de conexiones, inicialización de los esquemas del mapeo en el manejador de la base de datos, etc) y los mapeos *ORM* a las tablas de la base de datos.



   b) **com.olimpo.entity**. Este paquete contiene los archivos *XML* y *Java* para cada tabla en la base de datos lo que constituye una `entidad` con la que la capa de acceso a datos podrá realizar las

operaciones de persistencia que correspondan; por ejemplo, para Usuario se tiene un archivo `Usuario.java` y su correspodiente `Usuario.hbm.xml`, con ellos *hibernate* sabe donde realizar las operaciones que se le indiquen sobre los objetos de tipo `Usuario` en las tablas de la base de datos.

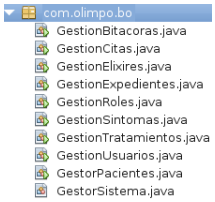


c) **com.olimpo.base**. Este paquete contiene dos clases de *Java*: `BaseDAO y BaseInterface`; la clase `BaseInterface` es una interfaz de *Java* que es implementada por `BaseDAO` y tiene los métodos para realizar la conexión con la base de datos a través de *hibernate*; también tiene las operaciones *CRUD* implementadas de manera genérica, ello para poder ser empleadas (usando herencia) por las clases que forman propiamente la capa de acceso a datos.

d) **com.olimpo.dao**. Este paquete contiene una clase *DAO* (*Data Access Object*) para cada entidad de persistencia de nuestro modelo de datos. Cada clase hereda de la clase `BaseDAO` (del paquete **com.olimpo.base**). Las clases tiene la siguiente denominación `«Entidad»DAO.java` y cada una de ellas sabe como realizar las

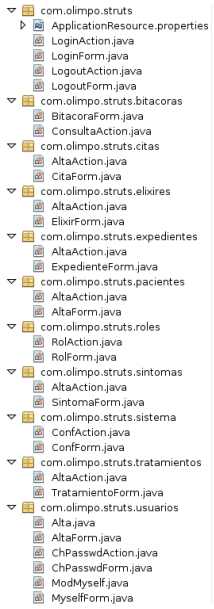
operaciones *CRUD* sobre la entidad de que se trate, así como el manejo de sesión y transacciones con la base de datos a través de *hinernate*.

2. Capa de Negocio (*BL*). Este paquete contiene una clase gestor que implementa las reglas de negocio del sistema y usa las operaciones de persistencia implementadas en la capa de acceso a datos. La nomenclatura de las clases es `Gestor«Entidad».java`. Se creó un gestor por cada caso de uso del sistema para dejar bien definidas las resposabilidades de cada uno de los gestores.

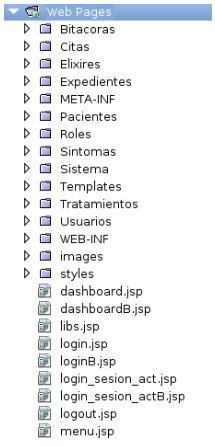


3. Capa de Presentación (*PL*). La capa de presentación se conforma por:

   a) **Paquetes *Java***. Estos paquetes estan separados en entidades; es decir, tomando com base **com.olimpo.struts** se tiene un paquete para cada entidad que se procesa en el sistema. Cada paquete contiene los `ActionForm` y `Action` que conforman en conjunto con los *JSPs* la estructura *MVC* de *Struts*.



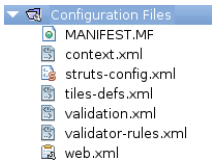
b) **Java Server Pages**. En esta estructura de directorios se encuentran los *JSPs* separados en entidades; así mismo dentro de cada directorio se encuentra un directorio *Template* para las plantillas que hacen uso de los *Tiles* [13] de Struts.



c) **Archivos de configuración de *Struts***. Los archivos de configuración se encuentran separados por el entorno de desarrollo en

[13] `Tiles` es un *framework* de *Apache* que es usado para realizar plantillas para la presentación de *JSPs*

la carpeta denominada `Configuration Files`.



Descripción de un flujo en código

El usuario ingresa en el sistema proporcionando su nombre de usuario y contraseña. El código de `loginB.jsp`, que es el cuerpo de la página dinámica de acceso a SICODIVEv2, es el encargado de desplegar en el navegador de páginas Web el formulario de ingreso. Los tags a destacar del *JSP* son el `bean:message` se encarga de traer la cadena de texto que se desplegará como etiqueta de la caja de entrada `hmtl:text` y el `html:errors` recogerá los errores en la validación una vez pasado por el `LoginForm` encargado de realizar las validaciones.

```
<html:form action="login.do" focus="nombre">
  <table align="center">
  <tr align="center">
  <td><H3><bean:message key="login.message" /></H3></td>
  </tr>
  <tr align="center">
    <td>
    <table align="center">
    <tr>
    <td>
      <logic:present name="loginError">
      <font style="color:red;text-align:center" size=1>
      <bean:message key="login.error" />
      </font>
      </logic:present>
      <logic:present name="sesion_act">
      <font style="color:red;text-align:center" size=1>
      <bean:message key="login.uso" />
      </font>
      </logic:present>
    </td>
    </tr>
    <tr>
    <td align="center" rowspan="2">
    <html:img srcKey="image.login" width="50%" height="50%" />
    </td>
    <td align="right" valign="bottom">
    <bean:message key="login.username"/>
    <html:text property="nombre" size="15" maxlength="15" />
    <td>
    <html:errors property="nombre" />
    </td>
    </td>
    </tr>
    <tr>
    <td align="right">
    <bean:message key="login.password"/>
    <html:password property="password" size="15" maxlength="15" redisplay="false"/>
    <td>
    <html:errors property="password" />
```

```
    </td>
    </tr>
    <tr>
    <td colspan="3" align="center">
    <html:submit >
      <bean:message key="login.button.signon"/>
    </html:submit>
    </td>
    </tr>
    </table>
    </td>
    </tr>
    </table>
  </html:form>
 </body>
</html>
```

Código 1: *loginB.jsp*

```
package com.olimpo.struts;
import com.olimpo.bo.GestionUsuarios;
import com.olimpo.entity.Usuario;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

public class LoginAction extends org.apache.struts.action.Action {

    private static final String SUCCESS = "success";
    private static final String FAILURE = "failure";
    private static final String SESION_ACT = "sesion-act";
    private static final String ONE_TIME_PASSWD = "one-time";
    private static final String LOGOFF = "logoff";

    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws Exception {
        String usrName = ((LoginForm) form).getNombre();
        String pass = ((LoginForm) form).getPassword();
        GestionUsuarios gestor = new GestionUsuarios();
        Usuario usr = gestor.accesoSistema(usrName, pass);
        // Cuando el usuario y contrasea no son correctos.
        if (usr == null) {
            request.getSession().setAttribute("loginError",
                new String("El usuario o contrasea son incorrectos"));
            return mapping.findForward(FAILURE);
        } else if (usr.getUsuarioSesion().equalsIgnoreCase(LOGOFF)) {
            usr.setUsuarioSesion("SESION_ACT");
            gestor.actualizaUsuario(usr);
            request.getSession().setAttribute("usuario", usr);
            return mapping.findForward(SUCCESS);
        } else if (usr.getUsuarioSesion().equals("ONE_TIME") &&
         usr.getUsuarioPassword().equals(usr.getUsuarioOneTimePassw())) {
            request.getSession().setAttribute("usuario", usr);
            return mapping.findForward(ONE_TIME_PASSWD);
        } else if(usr.getUsuarioSesion().equals("SESION_ACT")){
            request.getSession().setAttribute("sesion_act",
                new String("El usuario se encuentra en uso"));
            return mapping.findForward(SESION_ACT);
        }
        return mapping.findForward(FAILURE);
    }
}
```

Código 2: *LoginAction.java*

La clase `LoginForm` es la encargada de validar los campos del formulario `«usuario»` y `«contraseña»` existan. Si no existen es mostrada la pantalla del error correspondiente. Otra de las responsabilidades de `LoginForm` es la de contener los campos que se declaran para el formulario en `login.jsp`.

```
public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
   ActionErrors errors = new ActionErrors();
   if ((getNombre() == null) || (getNombre().length() < 1)) {
       errors.add("nombre", new ActionMessage("error.nombre.required"));
   }
   if ((getPassword() == null) || (getPassword().length() < 1)) {
       errors.add("password", new ActionMessage("error.password.required"));
   }
   return errors;
}
```

Código 3: *LoginForm.java*

El `LoginForm` llega al `LoginAction` y le son extraídos los parámetros: nombre del usuario y la contraseña, para posteriormente, pasarlos al Gestor de usuarios. En el Action se revisa: que el usuario exista y que los datos coincidan con los registrados en la base de datos; también se revisa que el usuario no esté siendo empleado y se revisa si es la primera vez que accede al sistema ó, si no existe. De acuerdo a la respuesta del gestor de usuarios se sigue un flujo diferente (veáse el diagrama de robustez). La respuesta del Action es enviada al *Struts* y él revisa el `struts-config.xml` para saber que flujo debe seguir a partir de dicha respuesta.

```
<struts-config>
    <form-beans>
.
.
.
       <form-bean name="LoginForm" type="com.olimpo.struts.LoginForm"/>
.
.
.
</form-beans>

<global-forwards>
    <forward name="login" path="/login" />
</global-forwards>

<action-mappings>
.
.
.
  <action path="/login"
              type="com.olimpo.struts.LoginAction"
              name="LoginForm"
              scope="session"
              validate="true"
              input="/login.jsp">

       <forward name="success" path="/dashboard.jsp" />
       <forward name="failure" path="/login.jsp" />
       <forward name="sesion-act" path="/login.jsp"/>
        <forward name="one-time" path="/Usuarios/chPasswd.jsp"/>
   </action>
.
```

```
.
.
    </action-mappings>
.
.
.
<struts-config>
```

Código 4: *LoginAction.java*

```
:
:
public class GestionUsuarios {

.
.
.
    private static final String LOGOFF="LOGOFF";
    private static final String ONE_TIME="ONE_TIME";
    private static final String SESION_ACT="SESION_ACT";
.
.
.

    public Usuario accesoSistema(String login, String password) throws Exception {

        ConfSistemaDAO confSys = new ConfSistemaDAO();
        ConfSistema configuracion = confSys.obtenerTodos().get(0);
        if (configuracion == null) {
            throw new Exception("Error: La configuracin del sistema no ha sido establecida.");
        }
        ControlDAO controlDao = new ControlDAO();
        Control ctrl = new Control();
        ctrl.setNombreUsuario(login);

        List<Control> listaControl = controlDao.encontrarPorEjemplo(ctrl,
             new String[]{"idControl", "sesion", "intentosFallidos",
                          "fechaUltimoIntento", "usuarioBloqueado"});
        if (listaControl.size() > 0) {
            ctrl = listaControl.get(0);
            ctrl.setFechaUltimoIntento(new Date());
            controlDao.actualiza(ctrl);
        } else {
            ctrl.setFechaUltimoIntento(new Date());
            ctrl = controlDao.guarda(ctrl);
        }
        UsuarioDAO usrDao = new UsuarioDAO();
        Usuario usr = new Usuario();
        usr.setUsuarioLogin(login);
        usr.setUsuarioPassword(password);
        List<Usuario> lista = usrDao.encontrarPorEjemplo(usr,
             new String[]{"usuarioNom", "usuarioApellidoP", "usuarioApellidoM",
                    "usuarioDir", "usuarioTel", "usuarioSesion", "usuarioFechUltMod",
             "usuarioHabilitado", "usuarioUltMod"});

// ACCESO CORRECTO AL SISTEMA
        //Si se encontraro usuario con el password y contrasea

        if (lista == null) {
            return null;
        }
        if (lista.isEmpty()){
            return null;
        }

        if (lista.size() > 0) {
            if (!ctrl.isUsuarioBloqueado()) {
                usr = lista.get(0);
                if (usr.getUsuarioSesion() != null &&
             usr.getUsuarioSesion().equals(LOGOFF)) {
                    usr.setUsuarioUltimoAcceso(new Date());
```

```
                usr = usrDao.actualiza(usr);
                ctrl.setIntentosFallidos(0);
                ctrl.setUsuarioBloqueado(false);
                controlDao.actualiza(ctrl);
            }
        } else if (!usr.getUsuarioSesion().equals(SESION_ACT)) {
            int i = ctrl.getIntentosFallidos() == null ? 0 : ctrl.getIntentosFallidos();
            if (i < Integer.parseInt(configuracion.getIntentosMax())) {
                ctrl.setIntentosFallidos(i + 1);
                controlDao.actualiza(ctrl);
            } else {
                ctrl.setUsuarioBloqueado(true);
                controlDao.actualiza(ctrl);
            }
        }
    }
    return usr;
}

.
.
.

}
```

Código 5: *LoginAction.java*

En las secciones anteriores se ha mostrado el rediseño y la implementación completa para el flujo que inicia con el ingreso al sistema y es continuada por la gestión de usuarios (casos de uso CU0 - Ingreso al sistema y CU6 - Gestión de Usuarios). Con este flujo hemos demostrado la interacción entre las capas *(DAL, BL y PL)* de la arquitectura. Es importante destacar que la implementación de la capa de acceso a datos, así como la capa de negocio ha sido concluida. Los componentes de la capa DAL fueron probados simulando la capa superior, a saber: la capa de negocio; así también, la capa de negocio fue probada simulando la capa de presentación; se realizaron las pruebas unitarias para asegurar que la responsabilidad de cada componente de las capa se cumpliera cabalmente; el apartado Casos de prueba presenta algunos ejemplos. Con respecto a la capa de presentación se implementó el controlador del *MVC* (Action) para los casos de uso siguientes: *i*) CU0 - Ingreso al sistema, *ii*) CU6 - Gestión de Usuarios, *iii*) CU7- Gestinar Roles y *iv*) CU8 - Gestión Sistema. Para el resto de los casos de uso solo se crearon las interfaces necesarias (*JSPs, Tiles, ActionForm, Action* y su mapeo en el `struts-config`) para la comunicación con las capas inferiores; queda pendiente implementar las acciones especificas en el `Action y ActionForm` para completar la presentación. Por ejemplo, para la gestión de bitácoras se creó el paquete `com.olimpo.struts.bitacoras` que contiene los `Action y ActionForm` (de *Struts*) que junto con los *JSPs* y el *template* de `Tiles` son mapeados por el `struts-config.xml` para tener la 3-tupla (patrón de diseño *MVC*).

Caso de prueba

Los casos de prueba que presentamos a continuación nos permiten validar el correcto funcionamiento del flujo que inicia con el ingreso al sistema (ver figura 9) y es continuada por la gestión de usuarios (casos de uso CU0 - Ingreso al sistema y CU6 - Gestión de Usuarios)

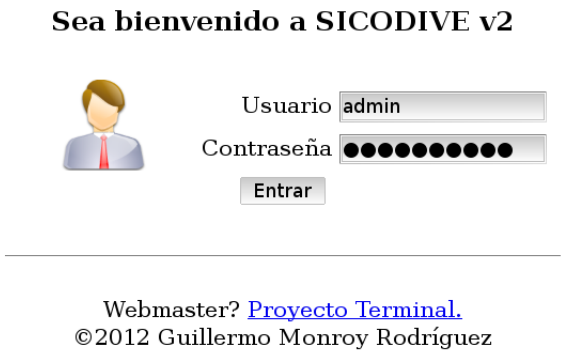


Figura 9 *pantalla para ingresar a SICODIVEv2*

I) **Caso de prueba de alta de nuevo usuario**.

(a) Ingresamos al sistema con el usuario Administrador, cuya contraseña es: Prueba@123

(b) Seleccionamos la opción `Usuarios`.

(c) Seleccionamos la opción `Alta usuario`.

(d) Ingresamos los siguientes datos en el formulario.

Campo	**Valor**
Nombre	Usuario
Apellido Paterno	de
Apellido Materno	pruebas
Usuario	pruebas
Rol	Usuario
Activo	Activo

(e) Copiamos la contraseña que se nos proporciona.

(f) Cerramos la sesión del usuario Administrador.

(g) Ingresamos al sistema con las credenciales del nuevo usuario (pruebas : «contraseña guardada»).

II) **Caso de prueba de bloqueo de usuario por accesos erroneos**.

a) Ingresamos la página de inicio del sistema.

b) Ingresamos los datos de usuario: prueba y la contraseña: 111111

c) Ingresamos los datos de usuario: prueba y la contraseña: 222222

d) Ingresamos los datos de usuario: prueba y la contraseña: 333333

e) El sistema avisa que el usuario se ha bloqueado.

III) **Caso de prueba de acceso con sesión de usuario en curso**.

a) Abrimos el navegador de páginas Web e ingresamos la ruta para SICOVIDEv2.

b) Ingresamos al sistema con el usuario Administrador, empleando la contraseña: Prueba@123

c) Abrimos otro navegador de páginas Web e ingresamos la ruta para SICOVIDEv2.

d) Ingresamos al sistema con el usuario Administrador, empleando la contraseña: Prueba@123

e) El sistema nos deberá notificar que el usuario está en uso y no nos dejará ingresar con ese usuario hasta que la sesión sea liberada.

IV) **Caso de prueba de usuario no existente**.

a) Abrimos el navegador de páginas Web e ingresamos la ruta para SICOVIDEv2.

b) Ingresamos un usuario inexistente, por ejemplo, usuarioFalso y la contraseña 123456.

c) El sistema nos avisará que el usuario no existe o los datos son erroneos.

Casos de prueba de la capa de acceso a datos

Se realizaron una serie de pruebas unitarias para validar el correcto funcionamiento de cada una de las clases de la capa de acceso a datos. Como ejemplo presentramos la prueba unitaria para la clase `UsuarioDAO` donde se realiza el alta de un usuario simulando la capa de negocio.

```
public static void main(String args[]) {
 UsuarioDAO usrdao = new UsuarioDAO();
 Usuario usr = new Usuario("Guillermo", "Monroy", "Rodrguez",
   "user", "password", "mi calle", "123213", "admin",
    new Date(), true, new Date().getTime());
 Usuario id = usrdao.guarda(usr);
 if (id != null) {
 System.out.println("Usuario guardado exitosamente:"
     + id.getUsuarioId());
 }
}
```

Código 6: *Prueba unitaria UsuariosDAO*

Consultando la base de datos podemos corrobarar que la inserción fue exitosa.

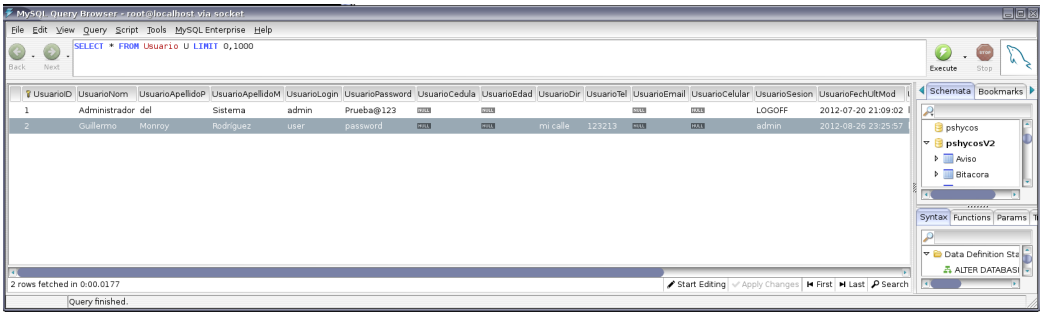


Figura 10 *Consulta de la tabla Usuarios*

Casos de prueba de la capa de negocio

Se realizaron una serie de pruebas unitarias para validar el correcto funcionamiento de cada una de las clases de la capa de negocio, denominados *gestores*. Como ejemplo presentramos la prueba unitaria para la clase `GestorUsuarios` donde se realiza la carga de configuración del sistema y posteriormente se prueba el ingreso al mismo.

```
public static void main(String[] args) {
       ConfSistema conf = new ConfSistema("127.0.0.1", "25", "3", "oh!", "Tu cuenta fue bloqueda por exceso de intentos",
        "Usted tiene cita ---");
       ConfSistemaDAO confDao = new ConfSistemaDAO();
       confDao.guarda(conf);

       try {
           GestionUsuarios gestor = new GestionUsuarios();
           Usuario usr = gestor.altaUsuario("Guillermo", "Monroy", "Rodriguez", "brujo",
```

```
"lovecraft", "", "", "LOGOFF", new Date(), true, 0);
          usr = gestor.restablecerContrasea(usr);
          System.out.println("INFO : " + usr.getUsuarioOneTimePassw());

          System.out.println(gestor.accesoSistema("brujo", "pepe"));
          System.out.println(gestor.accesoSistema("brujo", "lovecraft"));
          System.out.println(gestor.accesoSistema("brujo", "w"));
          System.out.println(gestor.accesoSistema("brujo", "dsgf"));
          System.out.println(gestor.accesoSistema("brujo", "dfds"));
          System.out.println(gestor.accesoSistema("brujo", "dfds"));
          System.out.println(gestor.accesoSistema("brujo", "lovecraft"));
      } catch (Exception ex) {
          Logger.getLogger(GestionUsuarios.class.getName()).log(Level.SEVERE, null, ex);
      }


  }
```

Código 7: *Prueba unitaria para GestorUsuarios*

Consultando la base de datos podemos corrobarar que la insersión fue exitosa.

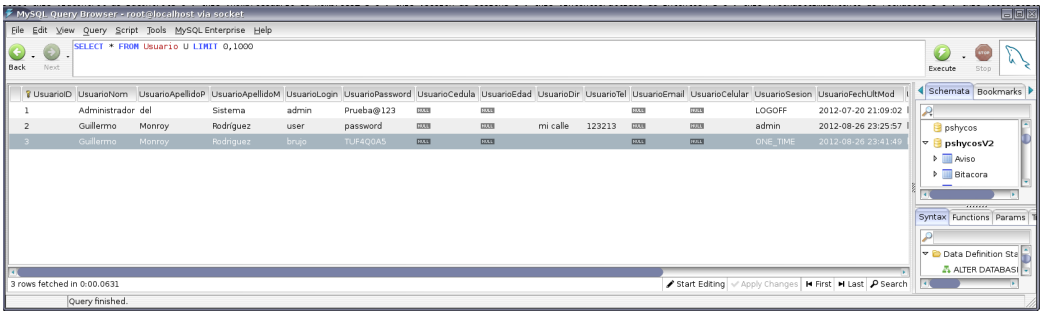


Figura 11 *Consulta de la tabla Usuarios*

Tecnología empleada: software y hardware

Hardware del ambiente de desarrollo: El hardware de la máquina que se utilizó para el desarrollo del sistema cuenta con las siguientes características:

- AMD Phenom(tm) 8450 Triple-Core Processor
- Disco duro de 160 Gbytes de capacidad de almacenamiento.
- 4Gbyte de memoria RAM DDR2.
- Monitor LCD 19 pulg. con 60 Mhz de tasa de actualización.
- Tarjeta gráfica ATI RS880 [Radeon HD 4200].

Características de la red La red donde se monto el sistema es una red local de 3 máquinas con topología de estrella, la aplicación se trata de una aplicación WEB por lo que se tendrá acceso al sistema desde cualquier computadora que tenga acceso a internet y un navegador de paginas WEB

Tecnología empleada para la realización del Proyecto.

1. **Manejador de Base de Datos.** El manejador de Base de Datos que se está utilizando para este proyecto es *MySQL 5.1.63*.
   Características del producto: *1)* Licencia GNU GPL. *2)* Motor multiplataforma.

2. **Lenguaje de Programación.** El Lenguaje de programación en el que se desarrolló el proyecto es *Java* version 1.6.0_23 con paradigma orientado a objetos. La razón de usar un paradigma de programación Orientado a Objeto se resume en los siguientes puntos:

   - `Flexibilidad.` Al tener relacionados los procedimientos que manipulan los datos con los datos a tratar, cualquier cambio que se realice sobre ellos quedará reflejado automáticamente en cualquier lugar donde estos datos aparezcan.
   - `Estabilidad.` Dado que permite un tratamiento diferenciado de aquellos objetos que permanecen constantes en el tiempo sobre aquellos que cambian con frecuencia permite aislar las partes del programa que permanecen inalterables en el tiempo.
   - `Reusabilidad.` La noción de objeto permite que programas que traten las mismas estructuras de información reutilicen las definiciones de objetos empleadas en otros programas e incluso los procedimientos que los manipulan. De esta forma, el desarrollo de un programa puede llegar a ser una simple combinación de objetos ya definidos donde estos están relacionados de una manera particular.

3. **Frameworks.** En el desarrollo de SICODIVEv2 se emplearon los siguientes frameworks:

   - *Hibernate.* Se empleó el framework *Hibernate* 3.5 para realizar el mapeo *ORM* [14].

[14] Object-Relational mapping

- *Struts*. Se empleó el framework *Struts* 1.3.6 que implementa el patrón *MVC* en la capa de presentación para tener un control y modularidad entre las páginas WEB y la capa de negocio del sistema.

4. **Servidor de aplicaciones.** *Apache Tomcat* 6.0.33.0 con *JRE* 1.6.0_23.
5. **IDE de desarrollo.** *Netbeans* 6.9 con *JRE* 1.6.0_23 y *MySQL Workbench 5.2.34*.
6. **Sistema Operativo de desarrollo.** El entorno de desarrollo se montó sobre GNU/Linux Debian 6 Squeeze para arquitectura AMD 64 bits.

6. Resultados del proyecto

Los resultados del proyecto se presentan a continuación a través de la descripción los puntos débilies identificados en SICODIVE y la mejora que se implementó en SICODIVEv2

1. SICODIVE permite el ingreso simultaneo de varias personas con un mismo usuario a partir de sesiones diferentes; por lo tanto, esas personas pueden escribir datos mientras su sesión está en curso, lo cual puede corromper los datos en la base de datos

   MEJORA: SICODIVEv2 garantiza que cuando un usuario haya ingresado al sistema, no se pueda hacer uso de su cuenta en otro equipo o en otra sesión, hasta que termine la primera sesión. Por seguridad se registra en una bitácora toda sesión inciada en el sistema; este registro permite identificar intrusiones o errores en el manejo de sesiones.

2. SICODIVE no bloquea el acceso cuando se ha intentado acceder sin éxito varias veces. Ésto permite que un intruso averigue la contraseña mediante "fuerza bruta", logrando acceder al sistema y violando así la seguridad del mismo sistema.

   MEJORA: SICODIVEv2 maneja las cuentas de los usuarios empleando roles con permisos específicos sobre las funcionalidades del sistema. En particular el rol de administrador de la seguridad podrá realizar el manejo de las configuraciones y tendrá permiso de ver las bitácoras del sistema.

3. SICODIVE no previene al usuario sobre los errores en que incurre cuando utiliza el sistema.

   Mejora: SICODIVEv2 cuenta con un sistema de avisos para que el usuario esté informado de lo que está pasando durante su operación. Adicionalmente, registra en una bitácora los eventos críticos que pueden provocar que el sistema deje de funcionar.

4. SICODIVE tiene en código duro la conexión a la base de datos. Esto hace imposible cambiar la ubicación de la base de datos si se deseara migrar a otro servidor; también obliga a mantener el usuario y la contraseña que registrada en el código.

   Mejora: SICODIVEv2 emplea varios archivos de configuración donde se especifica la cadena de conexión, el nombre de usuario, la contraseña y la ubicación del servidor de la base de datos.

5. SICODIVE no maneja bitácoras.

   Mejora: SICODIVEv2 tiene definidas las tablas (a nivel de base datos) para el manejo de bitácoras donde se registra información sobre la operación de sistema; también se cuenta con tablas que permiten manejar la información de bitácoras dende se reportan alertas y errores del sistema.

6. SICODIVE no cuenta con una conexión segura entre el sistema y el cliente. Esta carencia puede generar que un intruso escuche y capture las transacciones entre el cliente y el sistema, comprometiendo así la información de los usuarios.

   Mejora: SICODIVEv2 Emplea SSL (!!!!!! definir) entre el sistema y el browser; así como entre el servidor de base de datos y la aplicación. Ésto se logra configurando el servidor de aplicaciones mediante el uso de un certificado para sitio Web, el cual puede ser autofirmado o generado por una Autoridad Certificadora, por ejemplo Verisign.

7. SICODIVE no tiene parametrizadas las leyendas y las etiquetas que aparecen en la vista del sistema, por lo que sería imposible cambiar de

idioma sin recompilar el sistema completamente.

MEJORA: SICODIVEv2 Maneja las etiquetas y leyendas que aparecen en la vista del sistema mediante archivos de propiedades; con ello garantizamos que pueda emplearse la internacionalización sin necesidad de compilar el sistema nuevamente.

7. Conclusiones y perspectivas

Siguiendo el modelo ciclíco de reingeniería se analizó el sistema WEB SICODIVE. Concretamente, se analizó la documentación existente y el código fuente de SICOVIDE. Partiendo del conocimiento del sistema se realizó una revisión operativa del sistema, se detectaron puntos débiles y se procedió a tomar nota para incluirlos en las mejoras del sistema. Se planteó una nueva arquitectura de 3 capas para darle mejoras *FURPS* al sistema; además de fortalecer el modelo de datos para que soportara los requerimientos producto de la detección de puntos débiles. La arquitectura de 3 capas se implementó usando los frameworks *Hibernate* y *Struts*. A continuación se describe la implementación de las capas.

- Capa de acceso a datos (*Data Access Layer*): Se implementó esta capa mediante el mapeo de las tablas a objetos de *Hibernate*. La base de datos se encuentra alojada en un motor de busqueda *MySQL*. Para realizar la conexión se creó un usuario de *MySQL* que tiene acceso al esquema PshycosV2 y permisos de inserción, actualización, borrado y consulta.

- Capa de negocio (*Bussiness Layer*): Se implementó usando clases que se denominan gestores, las cuales hacen llamadas a métodos de las capa de acceso a datos. Las clases implementadas cuentan con las reglas de negocio correspondientes para cada caso de uso.

- Capa de presentación (*Presentation Layer*): La capa de presentación se implementó para los casos siguientes: *i*) CU0 - Ingreso al sistema, *ii*) CU6 - Gestión de Usuarios, *iii*) CU7- Gestinar Roles y *iv*) CU8 - Gestión Sistema. La implementación se realizó empleando el framework *Struts*, *JSPs* y archivos de propiedades y configuración. Esta capa hace uso de los gestores de la capa de negocio (BL); también hace uso de

Struts cuyos elementos son: *a)* Tiles, *b)* Validators, *c)* ActionForms y *d)* Form.

La implementación de la capa de acceso a datos, así como la implementación de la capa de negocio fueron concluidas. Los componentes de la capa DAL fueron probados simulando la capa superior, a saber: la capa de negocio; también, la capa de negocio fue probada simulando la capa de presentación; se realizaron las pruebas unitarias para asegurar que la responsabilidad de cada componente de las capa se cumpliera cabalmente. Con respecto a la capa de presentación se crearon las interfaces necesarias (*JSPs*, `Tiles, ActionForm, Action` y su mapeo en el `struts-config`) para la comunicación con las capas inferiores en los casos. Como trabajo futuro habrá que implementar las acciones especificas del `Action`, a fin de completar la capa de presentación para el resto de los casos de uso; quien continúe este proyecto podrá tomar como ejemplos las implementaciones del `Action` de los casos de uso: CU0, CU6, CU7 y CU8.

Cabe mencionar que no se realizó un manual de usuario como tal, ya que no se tiene la capa de presentación completa. Sin embargo, se cuenta con el documento de pruebas de las funcionalidades a nivel de las capas *DAL y BL*, las cuales son la base para la implementación de la presentación de los casos de uso que quedaron a nivel de interfaz de presentación. Entre las ventajas que ofrece el sistema SICODIVEv2 para quien lo retome y conluya podemos citar:

1. Internacionalización a través de archivos de properties.

2. Parametrización en archivos de configuración para la conexión a la base de datos.

3. Parametrización en archivos de la configuración del servidor de aplicaciones.

4. Implementación en capas con pruebas unitarias dentro de las clases que forman las capas.

5. Uso de plantillas (`Tiles` de *Struts*) de *JSPs* para darle un orden y control a la capa de presentación.

6. Uso de validaciones antes de la petición al servidor de aplicaciones.

Consideramos que la realización de este proyecto servirá como ejemplo del proceso de reingeniería a otros estudiantes. Además, podrá servir como plataforma de ensayo para aquellos que se interesen tanto en la tecnología de desarrollo empleada, así como en la metodología de reingeniería planteada.

Referencias

[1] S.R. Tilley & D.B. Smith, Perpectives on Legacy Systems Reengineering Software Institute, Carnegie Mellon University, 1995

[2] C. Bauer & G. King, Java Persistance with Hibernate, 2a ed, Greenwich, MANNING, 2007,ch 8,9 y 10.

[3] N. Ford, Art of Java Web Development, Greenwich, MANNING , 2004, ch 6.

[4] M. Pistoia, et al., Java 2 Network Security, 2a ed, IBM- International support organization, 1999,ch 13,16

[5] I. Sommerville, Ingeniería del software, 7a ed., Madrid, España, Pearson Addison Wesley, 2005, ch 12,13,18 y 21

[6] E.J. Chifofsky & J.H. Cross, Reverse Engineering and Design Recovery: A Taxonomy, IEEE Software, 1990, pg. 13-17