UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

“SISTEMA DE INFORMACIÓN DEL CENTRO DE DOCENCIA DE LA FACULTAD DE INGENIERÍA"

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A N:

JENNIFER CHIMAL MORENO
NATH-YELI VELAZQUEZ FERRER

DIRECTOR DE TESIS:

ING. MARÍA DEL ROSARIO BARRAGÁN PAZ

Ciudad Universitaria México, D.F. 2007

Agradecimientos:

Agradezco a Dios y a la virgen de Guadalupe sobre todas las cosas,
ya que siempre me han guiado y me han permitido lograr poco a poco
lo que me he propuesto.

También agradezco a mi mamá, ya que siempre ha procurado ser padre
y madre para poder sacarnos adelante, mediante sacrificio y dedicación;
aunque ella y mi hermano, tal vez no lo saben, han sido mi motor
para seguir adelante.

Agradezco a mi papá y abuelito Alfredo Ferrer, ya que sé que aunque están
en el cielo siempre me envían la fuerza necesaria para levantarme en momentos
difíciles. Sé que si estuvieran aquí estarían orgullosos de mi.

Agradezco a mi amada Facultad de Ingeniería, la cual me ha formado como profesionista
y me ha hecho sentir orgullosa cada momento que puedo decir: "Estudié en la Facultad
de Ingeniería de la UNAM"

Agradezco a la Unidad de Servicios de Cómputo Académico, ya que me permitió ser parte de
esa gran familia y compartió conmigo el conocimiento para ser un mejor ingeniero,
pero sobre todo un mejor ser humano. En especial, agradezco a la Ing. Rosario Barragán Paz,
ya que me dio la oportunidad de pertenecer a su grupo de trabajo y sobre todo siempre demostró
confianza en el trabajo realizado por mis compañeros y por mi.

Un agradecimiento a mis profesores, ya que siempre mostraron interés en mi educación, además de otorgarme

su amistad sincera, en especial a: "Erik Castañeda de Isla Puga", siempre has sido mi modelo a seguir en el ámbito

profesional y humano. Un agradecimiento también a nuestro profesor y sinodal "Carlos Román Zamitiz", por todo el apoyo en el desarrollo y revisión de esta Tesis.

Agradezco a mi compañera de Tesis Jennifer Chimal Moreno, alias "la má", ya que además de ser una buena amiga me ayudo en el desarrollo de éste trabajo, hemos sido un buen equipo. Te quiero mucho.

Agradezco a todos mis amigos, ya que en cada etapa de mi vida me han apoyado incondicionalmente y

también gracias a ellos he logrado conseguir cada una de mis metas.

Finalmente, pero no menos importante, agradezco a mi novio Luis Daniel, ya que me ha dado todo el

apoyo, comprensión y amor existente en su corazón, ha sabido ser mi perfecto complemento y sobre todo

una parte esencial en mi familia. Te amo.

Con amor.

Nath

A MI MADRE

Evangelina Moreno Blancarte

Por estar conmigo y apoyarme en ser mejor cada vez pero sobre todo por creer en mí. Sin ti no hubiera sido posible llegar hasta donde estoy ahora.

Gracias por ser mi amiga, te quiero mucho "May".

A MIS HERMANAS

Chela y Penélope.

Por ser mis cómplices en tantas aventuras, por todos esos buenos momentos que me han hecho la persona que soy y darme ese consejo certero cuando más lo necesitaba.

A MI MEJOR Y GRAN AMIGA

Edith

Por todo tu cariño y apoyo, por ayudarme a comprender que la vida no es tan fácil pero que esforzándote se pueden alcanzar tus sueños. Gracias por ser mi estrella, te quiero como no tienes idea.

A CHINO

Mi monstruo

Por apoyarme en todas mis locuras, por inspirarme a ser mejor cada día y enseñarme a disfrutar cada instante en la vida. Por ser mi mejor amigo y darme tanto cariño.
Eres lo mejor que me pudo pasar.
Te amo.

A MI COMPAÑERA DE TESIS

Nath

Por tu dedicación y apoyo. Porque sin ti no hubiera podido realizar este trabajo de tesis. Por todo lo que me enseñaste. Por escucharme en tantos momentos difíciles en mi vida.

A MIS MUGRES

Mary, Güero y Karlitos

Por darme tantos momentos divertidos,
porque a pesar de no haber compartido un pasado común
me han dado su apoyo y cariño.
Los quiero mucho.

A MI DIRECTORA DE TESIS

Chary

Gracias por creer en nosotras y darnos todo tu apoyo para la realización de este proyecto.

A NUESTRO GRAN APOYO

Carlos Roman Zamitiz

Gracias por todo ese apoyo incondicional, fuiste la inspiración
para la realización de este proyecto de tesis,
gracias por tantas enseñanzas y por dedicarnos gran
parte de tu tiempo.

A UNICA

Por darme la oportunidad de aprender gran parte de lo que sé.

JENNIFER

Introducción

El Centro de Docencia “Ing. Gilberto Borja Navarrete”, ubicado en la Facultad de Ingeniería UNAM; tiene por objetivos propiciar elementos teóricos, metodológicos y prácticos que contribuyan al mejoramiento de la labor docente del profesor a través de la revisión teórica de los principios y fundamentos de la práctica educativa y del análisis y reflexión de su propia experiencia en un marco de competitividad y desarrollo; además de propiciar la actualización de los profesores en el conocimiento científico y tecnológico de los distintos campos disciplinarios de la ingeniería. Busca ofrecer un espacio idóneo a los profesores para intercambiar ideas, impulsar su creatividad y propiciar la reflexión sobre su práctica docente.

Dada la necesidad de difusión de las actividades propias al Centro de Docencia “Ing. Gilberto Borja Navarrete**”**, es imprescindible contar con una herramienta capaz de permitir al docente consultar temarios, listas de cursos disponibles y el detalle de los mismos; así como administrar la información de profesores inscritos a la capacitación académica a través de consultas por el tipo de actividad, actualmente se cuentan con los siguientes tipos:

- Cursos
- Eventos
- Seminarios

Las búsquedas de actividades se podrán realizar por los siguientes rubros: disciplina, dependencia, fecha de inicio, división, nombre de actividad y tipo de evento. Existirán tres roles de usuario, los cuales se definen a continuación:

- Administrador del Centro de Docencia: Será el encargado del registro de Divisiones (con sus respectivos jefes y responsables), actividades y todos los catálogos necesarios.

- Responsable de la división: Registrará actividades, realizará altas y bajas de profesores, así como la inscripción y evaluación de los mismos a las diversas actividades.

- Jefe de cada división: Consultará el aprovechamiento de los profesores, así como el histórico de su desempeño en las diversas actividades.

Objetivos

Objetivo General

Crear un sistema para administrar la información concerniente a los cursos y actividades desarrolladas en el centro de docencia “Ing. Gilberto Borja Navarrete” de la Facultad de Ingeniería.

Objetivos Específicos

- Desarrollar o crear un módulo para la administración del sistema.
- Acceder vía web al sitio.
- Contar con cuentas de acceso mediante un único usuario y password para cada jefe y responsable de división.
- Visualizar la información de cursos en la parte administrativa y secretarías.
- Desarrollar la documentación necesaria (manual de operación, manual de usuario).
- Almacenar información requerida de profesores para revisar su resumen de actividades (inscritas, acreditadas, no acreditadas).
- Difundir las actividades del Centro de Docencia (cursos, talleres, seminarios)
- Registrar la lista de lo cursos recibidos.
- Garantizar la búsqueda de los diferentes cursos impartidos en el Centro de Docencia.

Capítulo 1

MARCO DE REFERENCIA

1.1 Centro de Docencia “Ing. Gilberto Borja Navarrete”

El Centro de Docencia "Ing. Gilberto Borja Navarrete", llamado así en honor a su generoso donante, distinguido egresado de esta institución, constructor del México actual y permanente benefactor de las más nobles causas universitarias; es una gran aportación a la nuestra Facultad de Ingeniería, debido a su importante función de enriquecer a la planta docente, más allá del marco técnico.

Hace más de un año se inauguró formalmente contando con la presencia del Rector Dr. Juan Ramón de la fuente, del Ing. Gilberto Borja Navarrete y del Director MC. Gerardo Ferrando Bravo, a quién agradecemos su sensibilidad para captar y resolver las necesidades del Centro de Docencia así como su respaldo, colaboración y decidida ayuda.

1.1.1 Objetivos

Los objetivos del Centro de Docencia son:

a) Propiciar elementos teóricos, metodológicos y prácticos que contribuyan al desarrollo profesional en la labor docente del profesor a través de la revisión teórica de los principios y fundamentos de la práctica educativa y del análisis y reflexión de su propia experiencia en un marco de competitividad y desarrollo.
b) Dotar al profesor de conocimientos avanzados respecto al proceso educativo, técnicas didácticas, tecnologías de información, comunicación y desarrollo humano, a través de conferencias, seminarios, sesiones de reflexión y reuniones de intercambio de experiencias.
c) Propiciar la actualización de los profesores en el conocimiento científico y tecnológico de los distintos campos disciplinarios de la ingeniería, de acuerdo con los planes y programas de estudio establecidos, apoyando la detección de necesidades y la instrumentación de actividades de formación.
d) Coadyuvar con la planta docente en la orientación y preparación de su material didáctico, aprovechando las tecnologías más modernas.
e) Ofrecer asesoría didáctica para la instrumentación de tecnologías educativas emergentes. Mantener la autenticidad, evitando modificaciones no autorizadas.

1.1.2 Ubicación

El centro de Docencia Ing. Gilberto Borja Navarrete se encuentra ubicado en la División de Ciencias Básicas de la Facultad de ingeniería, UNAM.

1.1.3 Actividades

- Capacitar profesores a través de un programa de Mejora Continua.
- Elaboración de material didáctico
- Desarrollo de herramientas multimedia
- Tecnología educativa
- Desarrollo humano
- Cómputo académico
- Mejorar el nivel académico con un programa de Formación y Desarrollo para el ejercicio profesional de la docencia de ingeniería
- Impartir y recibir videoconferencias
- Desarrollar investigación educativa
- Difundir los avances del proceso enseñanza-aprendizaje
- Proporcionar asesoría para:
    - Práctica docente
    - Usos de nuevas tecnologías
    - Elaboración de material académico
    - Presentaciones multimedia
    - Notas de clase
    - Publicaciones, libros y apuntes

Unidad de Servicios de Cómputo Académico (UNICA)

Política de calidad

La Unidad de Servicios de Cómputo Académico (UNICA), tiene como objetivo principal cumplir con los requisitos de sus clientes en el área de cómputo, teniendo como meta elevar la calidad de sus productos y servicios, para ello se compromete en un proceso de mejora continua.

Misión

La misión de la Unidad de Servicios de Cómputo Académico de la Facultad de Ingeniería, es la de proporcionar eficaz y eficientemente en el ámbito institucional, los servicios de cómputo y el apoyo en actividades relacionadas a coadyuvar el proceso integral de la formación académica de la Facultad de Ingeniería.

Visión

La proyección de la Unidad de Servicios de Cómputo Académico al año 2010, es continuar siendo una Unidad líder en la prestación de servicios de cómputo de vanguardia a la Facultad de Ingeniería, al entorno universitario y a la sociedad en general.

- Contando con la organización, administración y recursos adecuados.
- Líderes en la formación, capacitación y difusión de la cultura informática.
- Contando con las herramientas y convenios adecuados para el desarrollo y la investigación informática.
- Contando con la infraestructura de red de cómputo moderna y tecnología de punta, brindando servicios de calidad y alta disponibilidad en tecnologías de información y comunicación
- Contando con los servicios y procesos de atención sistematizados y actuales en apoyo a los eventos de seguridad informática.
- Contando con la infraestructura adecuada y mecanismos para la actualización continua del equipo de cómputo.

Valores

El ambiente de trabajo de los integrantes de la Unidad de Servicios de Cómputo Académico se basa en un clima de cordialidad, respeto, honestidad, responsabilidad, ética y compromiso.

Objetivos

Proporcionar a nivel institucional, los servicios de apoyo en cómputo que los alumnos de la Facultad de Ingeniería requieren para la realización y cumplimiento eficaz de sus tareas sustantivas.

Formar recursos humanos de calidad, tanto en el área de cómputo como en el desempeño de la vida profesional.
Ofrecer a la comunidad de la Facultad de Ingeniería capacitación en lo relativo a tópicos de cómputo. Ésta capacitación se divide en: cursos para el personal académico, cursos complementarios para alumnos y cursos de superación para el personal administrativo.

Mantenerse a la vanguardia en todo lo relativo al cómputo a través de la investigación e incorporar a la actividad de la Facultad de Ingeniería nuevas tendencias de cómputo.

Apoyar a la Secretaría General en las actividades que involucren institucionalmente a la Facultad de Ingeniería.

Capítulo 2

REQUERIMIENTOS DEL SISTEMA

Se requiere desarrollar un sistema, para administrar la información concerniente a los cursos y actividades desarrolladas en el Centro de Docencia de la Facultad de Ingeniería, que permita facilitar la difusión de actividades y beneficios que éste proporciona. La aplicación deberá ser creada para poder ejecutarse en ambiente cliente – servidor, con el fin de poder acceder a ella desde cualquier computadora.

Los requerimientos entregados inicialmente, se pueden observar en el anexo 1 de éste documento.

2.1 Propuesta de desarrollo

Se utilizará el Lenguaje Unificado de Modelado (UML), para desarrollar una serie de diagramas que permitan ver el sistema desde diferentes perspectivas, esto se llevará a cabo en una secuencia de iteraciones, hasta llegar a la versión final del sistema. Cada iteración comienza con una primera aproximación del análisis, diseño e implementación, para identificar los riesgos del diseño, primero se identifican los riesgos y después se prueba la aplicación para que dichos riesgos se reduzcan.

Una vez obtenido el modelo, aplicaremos la tecnología JSP (Java Servlets Page) y JavaBeans para la implementación del sistema de información, así como JDBC (Java DataBase Connectivity) para acceder a la base de datos.

Se utilizará el manejador de Bases de Datos Postgresql para el almacenado y consulta de la información del sistema, además de utilizar el servidor web Tomcat, que contendrá los servlets y jsp's necesarios para su buen funcionamiento.

Los requerimientos generales son los siguientes:

- *Registrar*: Solo a los usuarios que podrán tener acceso al ingreso de nuevas actividades.
- *Establecer*: Un módulo que garantice la seguridad al sistema.
- *Configurar*: Los módulos, funcionalidad y acciones a realizar en la aplicación.
- *Clasificar*: La información en catálogos, para tener una mejor organización de la misma.
- *Crear*: Flexibilidad en la integración de los datos de las actividades.

- *Ordenar*: Información de tal manera que no provoque problemas para hacer uso de ella.

Requerimientos de entrada. (E)

E1. Uno o más usuarios dentro de la aplicación.

Requerimientos de funcionalidad (F)

F1. Módulo de administración

- Módulo de catálogos
- Módulo de configuración
- Módulo que permite el registro de menús del sistema
- Módulo que registra los roles en el sistema
- Módulo de registro de usuarios en el sistema

Requerimientos de salida (S)

S1. Pantallas de Catálogos
S2. Pantalla de módulos
S4. Pantalla de sistema
S5. Pantalla de roles
S6. Pantalla de usuarios

Requerimientos de seguridad (X)

Los usuarios tendrán acceso a los diferentes módulos de acuerdo a su perfil definido.

Requerimientos de software, hardware y comunicaciones (H)

La máquina en donde se instale el servidor deberá tener una dirección IP estática. También cabe resaltar que en caso de utilizar la plataforma de sistema operativo Windows, deberá ser configurado un sistema de archivos del tipo NTFS.

Máquina virtual requerida: Sun Java 2 SDK 1.4.2_04 o posterior.

Servidor de base de datos: PostgreSQL

Servidor de web: Apache Tomcat

Explorador web: Microsoft Internet Explorer, versión 6.0 o posterior. Adicionalmente, el plug-in de Java para Internet Explorer, Java Runtime Environment (JRE) 1.4.1 es requerido para que el usuario final pueda ejecutar la aplicación, además de permitir el acceso vía web a la consola de administración de Apache Tomcat.

Memoria RAM disponible: 512 MB de RAM como mínimo, 1 GB o más es lo más recomendable para un mejor desempeño.

Espacio en disco duro: 80 GB como mínimo, esto debido a las instalaciones correspondientes y los datos almacenados en la base de datos.

Procesador: Pentium IV como mínimo.

2.2 Antecedentes de la programación orientada a objetos.

La programación orientada a objetos, es una extensión natural de la actual tecnología de programación, y representa un enfoque nuevo y distinto al tradicional. Al igual que cualquier otro programa, el diseño de un programa orientado a objetos tiene lugar durante la fase de diseño del ciclo de vida de desarrollo de software. El diseño de un programa orientado a objetos, es único en el sentido de que se organiza en función de los objetos que manipulará. De hecho, probablemente la parte más difícil de la creación de software orientado a objetos es identificar las clases necesarias ye el modo en que interactúan entre sí.

Desgraciadamente, no hay reglas fáciles para determinar las clases de un programa dado. El proceso es algo impreciso, y por esta causa han surgido numerosos métodos que proporcionan reglas para la identificación de clases y las relaciones que existen entre ellas.

El principio básico de la programación orientada a objetos, es que un sistema de software se ve como una secuencia de transformaciones en un conjunto de objetos. El término objeto, es una persona, un lugar o una cosa. La mayoría de los objetos del mundo real tienen atributos (características que los describen), y también comportamiento. El comportamiento es el conjunto de acciones que puede realizar un objeto.

Los principios en los que se apoyan las tecnologías orientadas a objetos son:

- Objetos como instancia de una clase
- Métodos
- Mensajes

Las características que ayudan a definir un objeto son:

- Encapsulamiento
- Modularidad
- Abstracción
- Polimorfismo

Las clases se organizan para modelar el mundo real en las siguientes relaciones:

- Herencia
- Agregación
- Asociación
- Uso

2.2.1 Análisis y diseño orientado a objetos.

Un problema de programación se describe, con un conjunto de especificaciones, de estás se desprende el proceso de análisis, diseño y programación, a continuación se describen de manera general estos procesos:

- ***Análisis Orientado a Objetos (AOO):*** Durante esta fase, se piensa en las especificaciones en términos intuitivos y con independencia del lenguaje de programación y el equipo que será utilizado. La etapa crítica de esta actividad es deducir los tipos de objetos del mundo real que están implicados y obtener los atributos de estos objetos determinando su comportamiento e iteraciones.

- ***Diseño Orientado a Objetos (DOO):*** En esta segunda fase, se comienza a crear un modelo de computadora basado en el análisis que realice la tarea deseada. En esta etapa se piensan en objetos del mundo real que pueden ser representados como objetos del mundo informático. Se deben especificar los objetos con mayor precisión, especificando en detalle lo que los objetos conocen y lo que pueden realizar, a su vez, describe con cierta prudencia las interacciones entre ellos mismos. En la fase de diseño, se pueden encontrar atributos útiles adicionales y comportamiento de los objetos que no aparecieron en la fase de análisis o no estaban definidos con claridad.

- La diferencia entre AOO y DOO no esclara, y es difícil definir la transición entre ambas. De hecho, las fases AOO y DOO no representan sentido estricto de dos etapas, y a veces se funden en una sola.

- ***Programación Orientada a Objetos (POO):*** Es la fase posterior a la fase de diseño, también conocida como Fase de Implementación. Esta fase consiste en traducir el diseño previamente realizado, en código real de un lenguaje de programación OO.

El proceso de desarrollo orientado a objetos supone, en síntesis, la construcción de un modelo del mundo real que se pueda traducir posteriormente en un código real escrito en un lenguaje de programación OO. En realidad las tres fases anteriores interactúan entre sí.

2.2.2 Notaciones orientadas a objetos.

El mejor sistema para modelar el mundo real con objetos de un modo práctico, es disponer de una notación gráfica consistente y eficiente. Cada metodología de análisis y diseño posee su propia notación. En este contexto, las metodologías más populares son las siguientes:

- Metodología de Grady Booch para la descripción de conjuntos de objetos y sus relaciones.
- Técnica de modelado orientada a objetos de James Rumbaugh (OMT: Object-Modeling Technique).
- Aproximación de Ivar Jacobson (OOSE: Object- Oriented Software Engineering) mediante la metodología de casos de uso (use case).

2.2.3 Orientación a objetos.

Puede describirse como el conjunto de disciplinas (ingeniería) que desarrollan y modelan software que facilitan la construcción de sistemas complejos a partir de componentes.

Proporciona conceptos y herramientas con las cuales se modela y representa el mundo real tal fielmente como sea posible.

Los conceptos y herramientas orientados a objetos son tecnologías que permiten que los problemas del mundo real sean expresados de modo fácil y natural. Las técnicas orientadas a objetos proporcionan mejoras y metodologías para construir sistemas de software complejos a partir de unidades de software modularizado y reutilizable.

La orientación a objetos, trata de cumplir las necesidades de los usuarios finales, así como las propias de los desarrolladores de productos de software. Estas tareas se realizan mediante el modelado del mundo real. El soporte fundamental e el *modelo objeto.* Los cuatro elementos (propiedades) más importantes de este modelo son:

- Abstracción
- Encapsulamiento
- Modularidad
- Jerarquía

Como sugiere Booch, si alguno de estos elementos no existe se dice que el modelo no es orientado a objetos.

2.2.3.1 Abstracción

La abstracción es uno de los medios más importantes, mediante el cual nos enfrentamos con la complejidad inherente al software. La abstracción es la propiedad que permite representar las características esenciales de un objeto, sin preocuparse de las restantes características.

Una abstracción se centra en la vista externa de un objeto, de modo que sirva para separar el comportamiento esencial de un objeto de su implementación. Definir una abstracción significa describir una entidad del mundo real, no importa lo compleja que pueda ser y, a continuación, utilizar esta descripción en un programa.

El elemento clave de la programación orientada a objetos es la clase, en este contexto, una clase se puede definir como una descripción abstracta de un grupo de objetos.

2.2.3.2 Encapsulamiento

Es la propiedad que permite asegurar que el contenido de la información de un objeto está oculta, es decir, se realiza un empaquetamiento de las variables de un objeto con la protección de sus métodos. El encapsulamiento es utilizado para esconder detalles de la puesta en práctica no importantes de otros objetos. Entonces, los detalles pueden cambiar en cualquier tiempo sin afectar otras partes del programa.

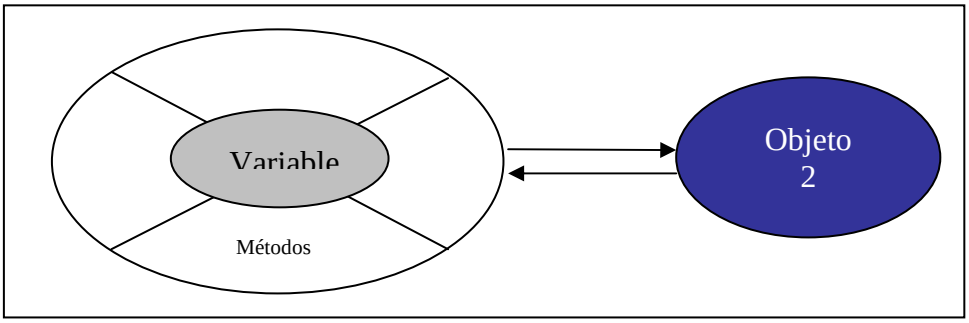


Fig. 2.1. Encapsulamiento de variables en métodos getter y setter.

2.2.3.3 Modularidad

Es la propiedad que permite subdividir una aplicación en partes más pequeñas (módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes. La modularidad consiste en dividir un programa en módulos que se puedan compilar por separado, pero que tengan conexiones con otros módulos.

2.2.3.3.1 La estructura de un módulo

Un módulo se caracteriza fundamentalmente por su interfaz y por su implementación. El módulo se define como un conjunto de acciones denominadas *funciones* o *submódulos* que comparten un conjunto de datos comunes implantados estáticamente llamados *atributos*, asociados a definiciones lógicas de tipos. Las acciones o funciones de un módulo que pueden ser llamadas desde otra aplicación o módulo se denominan *primitivas* o *puntos de entrada del módulo*.

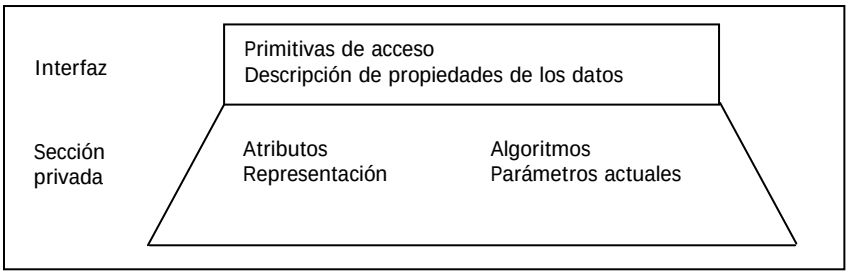


Fig. 2.2. Estructura de un módulo

2.2.3.4 Herencia

La herencia es la propiedad que permite a los objetos ser construidos a partir de otros objetos; es la capacidad de un objeto para utilizar las estructuras de datos y los métodos previstos en ascendientes. El objetivo final, es la reutilización de código anteriormente ya desarrollado.

La herencia supone una clase base y una jerarquía de clases que contienen las *clases derivadas* de la clase base. Las clases derivadas pueden heredar el código y los datos de su clase base, añadiendo su propio código especial y datos a ellas, incluso cambiar aquellos elementos de la clase base que necesite sean diferentes.

Las clases derivadas heredan características de su clase base, pero añaden otras características propias nuevas.

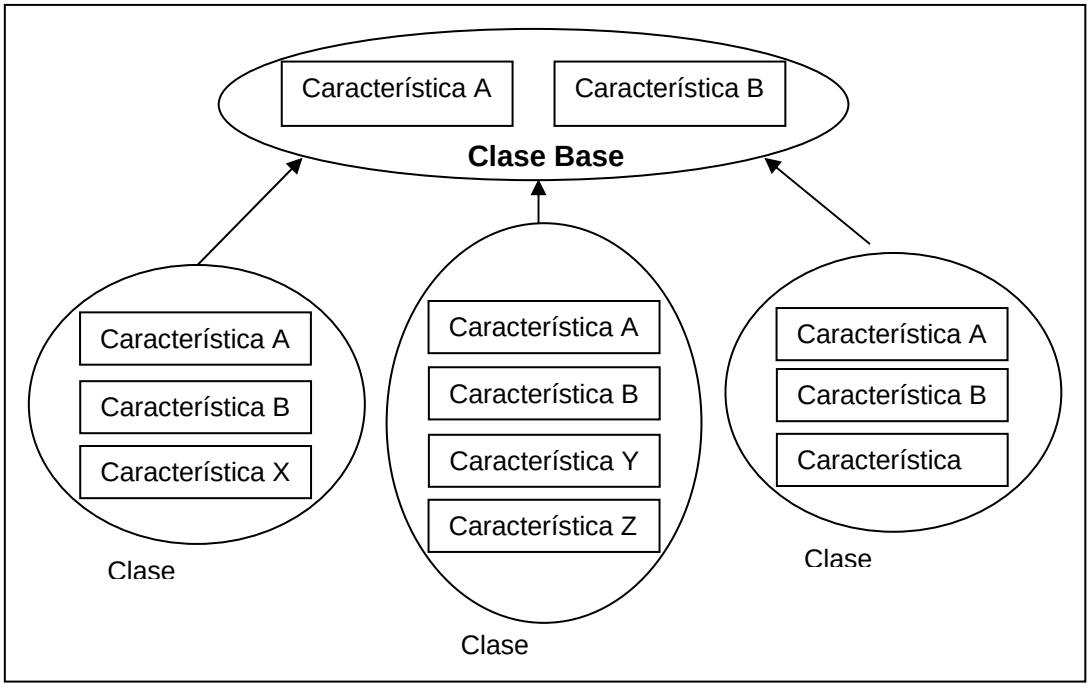


Fig. 2.3. Diagrama general de Herencia

Las clases derivadas se crean en un proceso de definición de nuevos tipos y reutilización del código anteriormente desarrollado en la definición de su clase

base. Este proceso se denomina *programación por herencia*. Las clases que heredan propiedades de una clase base pueden a su vez servir como definiciones base de otras clases. Las jerarquías de clases se organizan en forma de árbol.

A continuación se muestra un ejemplo gráfico de herencia, en el que la clase Base será la clase "Animal", la cuál es la más general de todas las clases mostradas en el diagrama. A medida que el diagrama va descendiendo, se va particularizando el tipo de animal al que se hace referencia, llegando a definirse completamente.

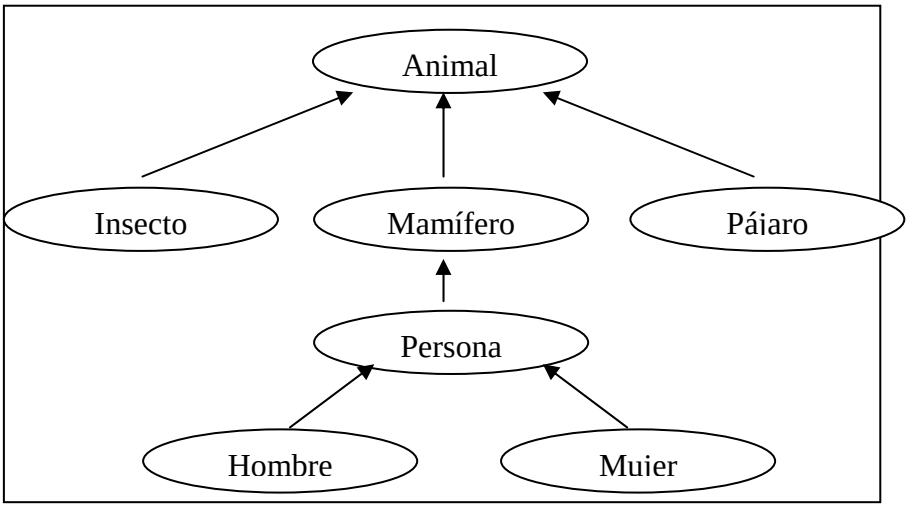


Fig. 2.4. Ejemplo práctico de Herencia

2.2.3.5 Polimorfismo

Un lenguaje orientado a objetos debe soportar el polimorfismo, lo que significa que clases diferentes pueden tener comportamientos diferentes para el mismo método.

El polimorfismo es más una característica de los comportamientos que de los objetos. Esta propiedad no suele ser considerada como fundamental en los diferentes modelos de objetos propuestos, pero dada su importancia

2.2.3.6 Sobrecarga

La sobrecarga es una propiedad que describe una característica adecuada que utiliza el mismo nombre de operación para representar operaciones similares que se comportan de modo diferente cuando se aplican a clases diferentes. Por consiguiente, los nombres de las operaciones se pueden sobrecargar, esto es, las operaciones se definen en clases diferentes y pueden tener nombres idénticos, aunque su código programado puede diferir.

Si los nombres de una operación se utilizan para nuevas definiciones en clases de una jerarquía, la operación a nivel inferior se dice que anula la operación a un nivel más alto.

Actualmente la sobrecarga se aplica sólo a operaciones. Aunque es posible extender la propiedad a atributos y relaciones específicas del modelo propuesto.

La sobrecarga no es una propiedad específica de los lenguajes orientados a objetos. Los lenguajes de programación convencionales soportan sobrecarga para algunas de las operaciones sobre algunos tipos de datos, como enteros, reales y caracteres. Los sistemas orientados a objetos ahondan un poco más en la sobrecarga y la hacen disponible para operaciones sobre cualquier tipo objeto.

2.2.4 Tipos abstractos de datos y clases.

Un tipo abstracto de dato (TAD), es un tipo de dato definido por el programador. Es posible implementar el TAD considerando los valores que se almacenan en las variables y qué operaciones están disponibles para manipular estas variables. En esencia, un TAD es un tipo de datos que consta de datos (Estructuras de datos propias) y operaciones que se pueden realizar sobre esos datos.

Un TAD se compone de *estructuras de datos* y los *procedimientos* o *funciones* que manipulan esas estructuras de datos.

TAD = Representación (datos) + Operaciones (funciones y procedimientos)

Una clase es una caracterización abstracta de un conjunto de objetos; todos los objetos similares pertenecen a una clase determinada. De modo más formal, una clase define variables (datos) y métodos (operaciones) comunes a un conjunto de objetos. En realidad, una clase es un prototipo o generador de un conjunto de objetos.

Una clase bien diseñada especifica un *tipo abstracto de dato* (TAD). Un tipo de dato es abstracto si las operaciones de alto nivel adecuadas a los tipos de datos están aisladas de los detalles de la implementación asociados con el tipo de dato.

La clase es el bloque de construcción fundamental de un lenguaje de programación orientado a objetos. Una clase es un TAD que posee un conjunto de transformaciones permitidas para dicho TAD; puede definir también su interfaz a otras clases o funciones, descubriendo para ello que parte de su descripción interna de datos o conjunto de transformaciones permitidas, pueden hacerse públicos. La regla por defecto, es que nada de una clase es pública, a menos que se declare explícitamente por el desarrollador de software que definió la clase.

El entorno orientado a objetos, oculta los detalles de implementación de un objeto. Es la propiedad conocida como *ocultación de la información.* La parte que no está oculta de un objeto, es su *interfaz público,* que consta de los mensajes que se pueden enviar al objeto. Los mensajes representan operaciones de alto nivel. El término encapsulamiento se utiliza también para enfatizar un aspecto específico de un tipo abstracto de datos. Un TAD combina métodos y representación interna.

Un objeto es una *instancia* de una clase que encapsula operaciones y representación. Este encapsulamiento contrasta con la separación tradicional de operaciones (funciones) y representación (datos).

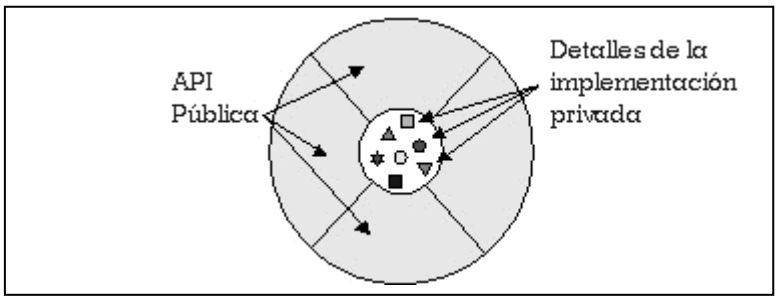


Fig. 2.5. Representación visual de un objeto como componente de software

2.3 Introducción al Lenguaje Unificado de Modelado UML

La notación UML se deriva y unifica las tres metodologías de análisis y diseño OO más extendidas:

1. Metodología de Grady Booch
2. Técnica de modelado orientada a objetos de James Rumbaugh
3. Aproximación de Ivar Jacobson

El desarrollo de UML comenzó a finales de 1994 cuando Grady Booch y Jim Rumbaugh de Rational Software Corporation empezaron a unificar sus métodos. A finales de 1995, Ivar Jacobson y su compañía Objectory se incorporaron a Rational en su unificación, aportando el método OOSE.

De las tres metodologías de partida, las de Booch y Rumbaugh pueden ser descritas como centradas en objetos, ya que sus aproximaciones se enfocan hacia el modelado de los objetos que componen el sistema, su relación y colaboración. Por otro lado, la metodología de Jacobson es más centrada a usuario, ya que todo en su método se deriva de los escenarios de uso. UML se ha ido fomentando y aceptando como estándar desde la formación de OMG, que es también el origen de CORBA, el estándar líder en la industria para la programación de objetos distribuidos.

UML es el primer método en publicar un meta-modelo en su propia notación, incluyendo la notación para la mayoría de la información de requisitos, análisis y diseño. Se trata pues de un meta-modelo auto-referencial (cualquier lenguaje de modelado de propósito general debería ser capaz de modelarse a sí mismo).

Los principales beneficios de UML son:

- Mejores tiempos totales de desarrollo (de 50% o más).
- Modelar sistemas (y no sólo de software) utilizando conceptos orientados a objetos.
- Establecer conceptos y artefactos ejecutables.
- Encaminar el desarrollo del escalamiento en sistemas complejos de misión crítica.

- Crear un lenguaje de modelado utilizado tanto por humanos como por máquinas.
- Mejor soporte a la planeación y al control de proyectos.
- Alta reutilización y minimización de costos

2.3.1 Lenguaje Unificado de Modelado (UML, Unified Modeling Language)

UML es un lenguaje para hacer modelos y es independiente de los métodos de análisis y diseño. Existen diferencias importantes entre un método y un lenguaje de modelado. Un *método* es una manera explícita de estructurar el pensamiento y las acciones de cada individuo. Además, el método le dice al usuario qué hacer, cómo hacerlo, cuándo hacerlo y por qué hacerlo; mientras que el lenguaje de modelado carece de estas instrucciones. Los métodos contienen modelos y esos modelos son utilizados para describir algo y comunicar los resultados del uso del método.

Un modelo es expresado en un *lenguaje de modelado*. Un lenguaje de modelado consiste de vistas, diagramas, elementos de modelo y un conjunto de mecanismos generales o reglas que indican cómo utilizar los elementos. Las reglas son sintácticas, semánticas y pragmáticas|

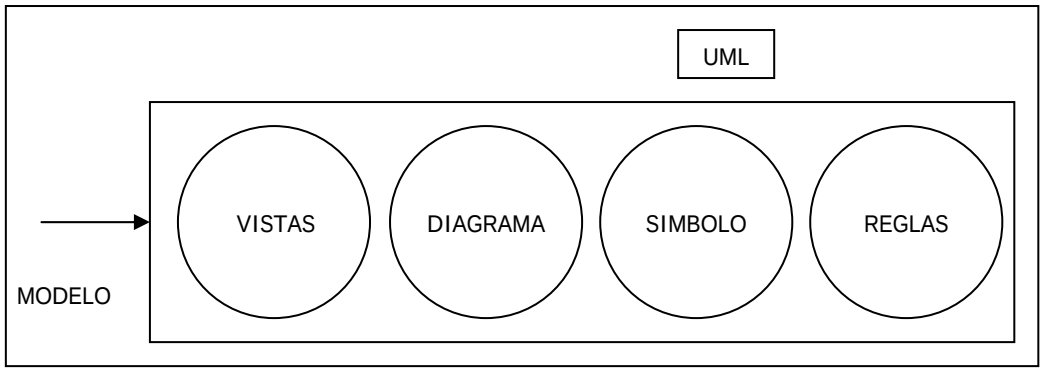


Fig. 2.6. Componentes del modelado de un sistema

Vistas: Las vistas muestran diferentes aspectos del sistema modelado. Una vista no es una gráfica, pero sí una abstracción que consiste en un número de diagramas y todos esos diagramas juntos muestran una "fotografía" completa del sistema. Las vistas también ligan el lenguaje de modelado a los métodos o procesos elegidos para el desarrollo. Las diferentes vistas que UML tiene son:

- *Vista Use-Case:* Una vista que muestra la funcionalidad del sistema como la perciben los actores externos.
- *Vista Lógica:* Muestra cómo se diseña la funcionalidad dentro del sistema, en términos de la estructura estática y la conducta dinámica del sistema.
- *Vista de Componentes: Muestra* la organización de los componentes de código.
- *Vista Concurrente:* Muestra la concurrencia en el sistema, direccionando los problemas con la comunicación y sincronización que están presentes en un sistema concurrente.

- *Vista de Distribución:* muestra la distribución del sistema en la arquitectura física con computadoras y dispositivos llamados *nodos*.

Diagramas: Los diagramas son las gráficas que describen el contenido de una vista. UML tiene nueve tipos de diagramas que son utilizados en combinación para proveer todas las vistas de un sistema. Se dispone de dos tipos diferentes de diagramas los que dan una vista estática del sistema y los que dan una visión dinámica.

Los diagramas estáticos son:

- *Diagrama de clases*: muestra las clases, interfaces, colaboraciones y sus relaciones. Son los más comunes y dan una vista estática del proyecto.
- *Diagrama de objetos*: Es un diagrama de instancias de las clases mostradas en el diagrama de clases. Muestra las instancias y como se relacionan entre ellas. Se da una visión de casos reales.
- *Diagrama de componentes*: Muestran la organización de los componentes del sistema. Un componente se corresponde con una o varias clases, interfaces o colaboraciones.
- *Diagrama de despliegue*: Muestra los nodos y sus relaciones. Un nodo es un conjunto de componentes. Se utiliza para reducir la complejidad de los diagramas de clases y componentes de un gran sistema. Sirve como resumen e índice.
- *Diagrama de casos de uso*: Muestran los casos de uso, actores y sus relaciones. Muestra quien puede hacer que y relaciones existen entre acciones (casos de uso). Son muy importantes para modelar y organizar el comportamiento del sistema.

Los diagramas dinámicos son:

- *Diagrama de secuencia, Diagrama de colaboración*: Muestran a los diferentes objetos y las relaciones que pueden tener entre ellos, los mensajes que se envían entre ellos. Son dos diagramas diferentes, que se puede pasar de uno a otro sin perdida de información, pero que nos dan puntos de vista diferentes del sistema. En resumen, cualquiera de los dos es un Diagrama de Interacción.
- *Diagrama de estados*: muestra los estados, eventos, transiciones y actividades de los diferentes objetos. Son útiles en sistemas que reaccionen a eventos.
- *Diagrama de actividades*: Es un caso especial del diagrama de estados. Muestra el flujo entre los objetos. Se utilizan para modelar el funcionamiento del sistema y el flujo de control entre objetos.

El número de diagramas es muy alto, en la mayoría de los casos excesivos, y UML permite definir solo los necesarios, ya que no todos son necesarios en todos los proyectos.

Símbolos o elementos de modelo: Los conceptos utilizados en los diagramas son los elementos de modelo que representan conceptos comunes orientados a objetos, tales como clases, objetos y mensajes, y las relaciones entre estos

conceptos incluyendo la asociación, dependencia y generalización. Un elemento de modelo es utilizado en varios diagramas diferentes, pero siempre tiene el mismo significado y simbología.

Reglas o mecanismos generales: Proveen comentarios extras, información o semántica acerca del elemento de modelo; además proveen mecanismos de extensión para adaptar o extender UML a un método o proceso específico, organización o usuario.

2.3.2 Diagramas en UML

La explicación se basará en los diagramas, en lugar de en vistas o anotación, ya que son estos la esencia de UML. Cada diagrama usa la anotación pertinente y la suma de estos diagramas crean las diferentes vistas.

2.3.2.1 Diagrama de Casos de Uso (Use-Case)

Se emplean para visualizar el comportamiento del sistema, una parte de el o de una sola clase. De forma que se pueda conocer como responde esa parte del sistema. El diagrama de uso es muy útil para definir como debería ser el comportamiento de una parte del sistema, ya que solo especifica como deben comportarse y no como están implementadas las partes que define. El diagrama también puede ser utilizado para que los expertos de dominio se comuniquen con los informáticos sin llegar a niveles de complejidad. Un caso de uso especifica un requerimiento funcional.

En el diagrama nos encontramos con diferentes figuras que pueden mantener diversas relaciones entre ellas:

- Casos de uso: representado por una elipse, cada caso de uso contiene un nombre, que indique su funcionalidad. Los casos de uso pueden tener relaciones con otros casos de uso. Sus relaciones son:
  - Include: Representado por una flecha.
  - Extends: Una relación de un Caso de Uso A hacia un caso de uso B indica que el caso de uso B implementa la funcionalidad del caso de uso A.
  - Generalization: Es la típica relación de herencia.
- Actores: se representan por un muñeco. Sus relaciones son:
  - Communicates: Comunica un actor con un caso de uso, o con otro actor.
- Parte del sistema (System boundary): Representado por un cuadro, identifica las diferentes partes del sistema y contiene los casos de uso que la forman.

Podemos emplear el diagrama de dos formas diferentes, para modelar el contexto de un sistema, y para modelar los requisitos del sistema.

2.3.2.1.1 Modelado del contexto

Se debe modelar la relación del sistema con los elementos externos, ya que son estos elementos los que forman el contexto del sistema.

Los pasos a seguir son:

- Identificar los actores que interactúan con el sistema.
- Organizar a los actores.
- Especificar sus vías de comunicación con el sistema.

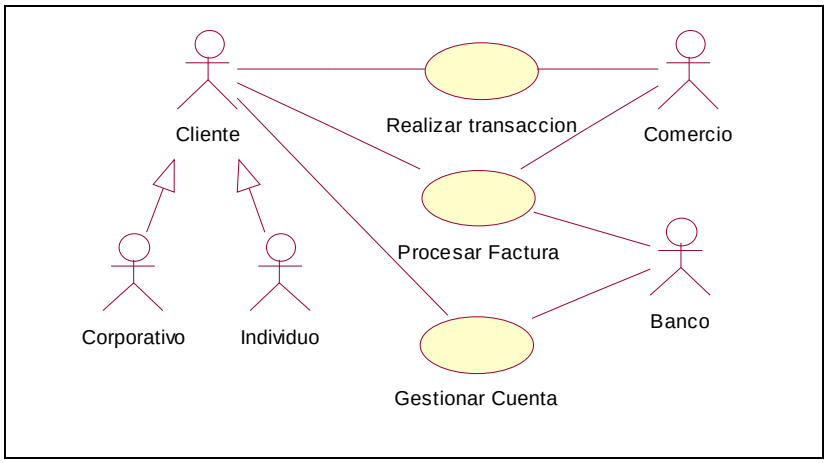


Fig. 2.7. Diagrama de modelado de contexto

2.3.2.1.2 Modelado de requisitos

La función principal, o la mas conocida del diagrama de casos de uso es documentar los requisitos del sistema, o de una parte de el.

Los requisitos establecen un contrato entre el sistema y su exterior, definen lo que se espera que realice el sistema, sin definir su funcionamiento interno. Es el paso siguiente al modelado del contexto, no indica relaciones entre autores, tan solo indica cuales deben ser las funcionalidades (requisitos) del sistema. Se incorporan los casos de uso necesarios que no son visibles desde los usuarios del sistema.

- Para modelar los requisitos es recomendable:
- Establecer su contexto, para lo que también podemos usar un diagrama de casos de uso.
- Identificar las necesidades de los elementos del contexto (Actores).
- Nombrar esas necesidades, y darles forma de caso de uso.
- Identificar que casos de uso pueden ser especializaciones de otros, o buscar especializaciones comunes para los casos de uso ya encontrados.

2.3.2.2 Diagrama de clases

Forma parte de la vista estática del sistema. En el diagrama de clases como ya hemos comentado será donde definiremos las características de cada una de las clases, interfaces, colaboraciones y relaciones de dependencia y generalización. Es decir, es donde daremos rienda suelta a nuestros conocimientos de diseño orientado a objetos, definiendo las clases e implementando las ya típicas relaciones de herencia y agregación.

En el diagrama de clases debemos definir a estas y a sus relaciones.

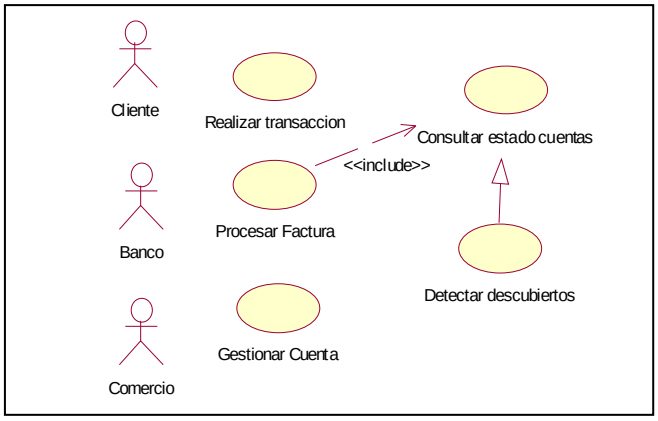


Fig. 2.8. Diagrama de clases

2.3.2.2.1 La clase

Una clase esta representada por un rectángulo que dispone de tres apartados, el primero para indicar el nombre, el segundo para los atributos y el tercero para los métodos.

Cada clase debe tener un nombre único, que las diferencie de las otras.

Un atributo representa alguna propiedad de la clase que se encuentra en todas las instancias de la clase. Los atributos pueden representarse solo mostrando su nombre, mostrando su nombre y su tipo, e incluso su valor por defecto.

Un método u operación es la implementación de un servicio de la clase, que muestra un comportamiento común a todos los objetos. En resumen es una función que le indica a las instancias de la clase que hagan algo.

Para separar las grandes listas de atributos y de métodos se pueden utilizar estereotipos.

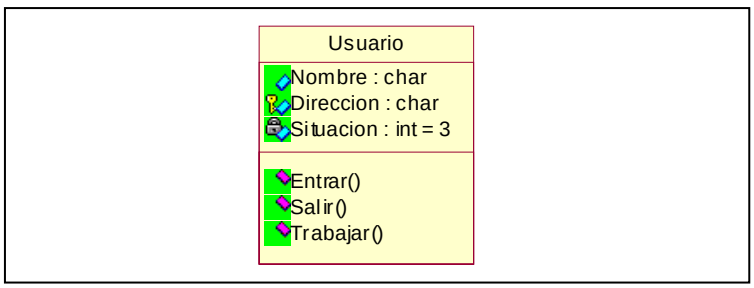


Fig. 2.9. Representación de una clase (Nombre, atributos y métodos)

2.3.2.2.2 Relaciones entre clases

Existen tres relaciones diferentes entre clases, *Dependencias*, *Generalización* y *Asociación*. En las relaciones se habla de una clase destino y de una clase origen. El origen es desde la que se realiza la acción de relacionar. Es decir desde la que parte la flecha, el destino es la que recibe la flecha. Las relaciones se pueden modificar con estereotipos o con restricciones.

2.3.2.2.2.1 Dependencias

Es una relación de uso, es decir una clase usa a otra, que la necesita para su cometido. Se representa con una flecha discontinua va desde la clase utilizadora a la clase utilizada. Con la dependencia mostramos que un cambio en la clase utilizada puede afectar al funcionamiento de la clase utilizadora, pero no al contrario. Aunque las dependencias se pueden crear tal cual, es decir sin ningún estereotipo, UML permite dar mas significado a las dependencias, es decir concretar más, mediante el uso de estereotipos.

- Estereotipos de relación Clase-objeto.

  - Bind: La clase utilizada es una plantilla, y necesita de parámetros para ser utilizada, con Bind se indica que la clase se instancia con los parámetros pasándole datos reales para sus parámetros.
  - Derive: Se utiliza al indicar relaciones entre dos atributos, indica que el valor de un atributo depende directamente del valor de otro. Es decir el atributo edad depende directamente del atributo Fecha nacimiento.
  - Friend: Especifica una visibilidad especial sobre la clase relacionada. Es decir podrá ver las interioridades de la clase destino.
  - InstanceOF: Indica que el objeto origen es una instancia del destino.
  - Instantiate: indica que el origen crea instancias del destino.
  - Powertype: indica que el destino es un contenedor de objetos del origen, o de sus hijos.
  - Refine: se utiliza para indicar que una clase es la misma que otra, pero mas refinada, es decir dos vistas de la misma clase, la destino con mayor detalle.

2.3.2.2.2.2 Generalización

Es la herencia, donde tenemos una o varias clases padre o superclase o madre, y una clase hija o subclase. UML soporta tanto herencia simple como herencia múltiple. Aunque la representación común es suficiente en el 99.73% de los casos UML nos permite modificar la relación de Generalización con un estereotipo y dos restricciones.

- Estereotipo de generalización.

- *Implementation*: El hijo hereda la implementación del padre, sin publicar ni soportar sus interfaces.

- Restricciones de generalización.

  - *Complete*: La generalización ya no permite más hijos.
  - *Incomplete*: Podemos incorporar más hijos a la generalización.
  - *Disjoint*: solo puede tener un tipo en tiempo de ejecución, una instancia del padre solo podrá ser de un tipo de hijo.

    - *Overlapping*: puede cambiar de tipo durante su vida, una instancia del padre puede ir cambiando de tipo entre los de sus hijos.

2.3.2.2.2.3 Asociación

Especifica que los objetos de una clase están relacionados con los elementos de otra clase. Se representa mediante una línea continua, que une las dos clases. Podemos indicar el nombre, multiplicidad en los extremos, su rol, y agregación.

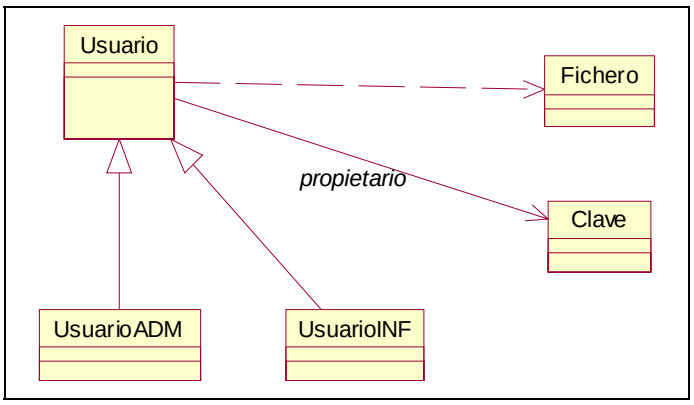


Fig. 2.10. Diagrama de asociación

En este diagrama se han creado cuatro clases. La clase principal es Usuario, que tiene dos clases hijas UsuarioADM y UsuarioINF. El usuario mantiene una relación de asociación con la clase Clave, se indica que es propietario de una clave, o de un número indeterminado de ellas. Se le crea también una relación de dependencia con la clase Perfil, es decir las instancias de usuario contendrán como miembro una instancia de Perfil.

2.3.2.3 Diagramas de objetos

Forma parte de la vista estática del sistema. En este diagrama se modelan las instancias de las clases del diagrama de clases. Muestra a los objetos y sus relaciones, pero en un momento concreto del sistema. Estos diagramas contienen objetos y enlaces. En los diagramas de objetos también se pueden incorporar clases, para mostrar la clase de la que es un objeto representado.

En este diagrama se muestra un estado del diagrama de eventos. Para realizar el diagrama de objetos primero se debe decidir que situación queremos representar del sistema. Es decir si disponemos de un sistema de mensajería, deberemos decidir que representaremos el sistema con dos mensajes entrantes, los dos para diferentes departamentos, dejando un departamento inactivo. Para el siguiente diagrama de clases:

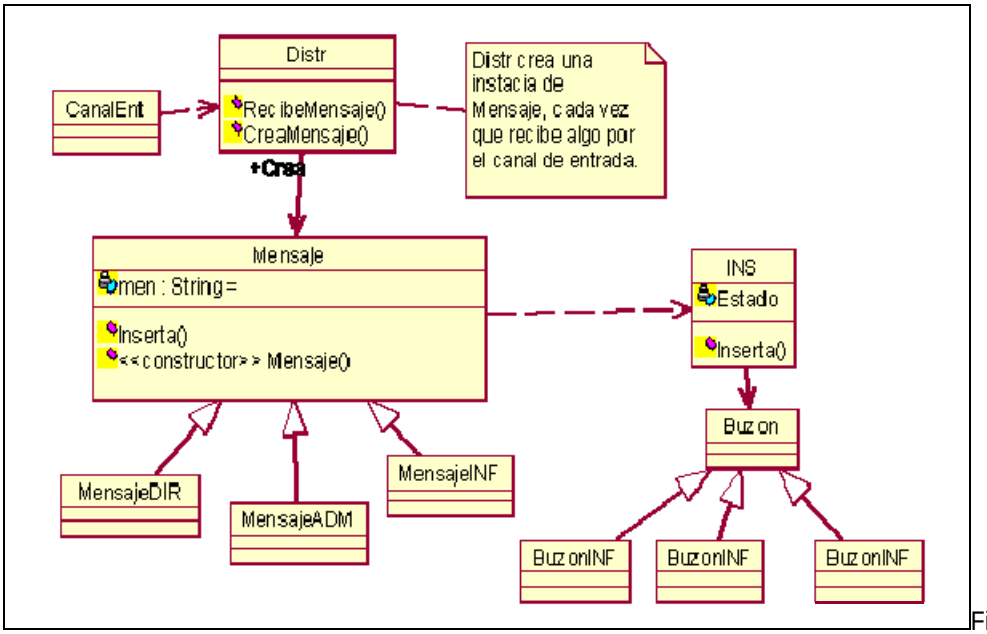


Fig. 2.11. Diagrama de clases

Tendríamos un diagrama de objetos con dos instancias de Mensaje, mas concretamente con una instancia de MensajeDIR y otra de MensajeADM, con todos sus atributos valorados. También tendríamos una instancia de cada una de las otras clases que deban tener instancia. Como CanalEnt, INS, Distr, y el Buzón correspondiente a la instancia de mensaje que se este instanciando. En la instancia de la clase INS se deberá mostrar en su miembro Estado, que esta ocupado realizando una inserción.

2.3.2.4 Diagrama de componentes

Se utilizan para modelar la vista estática de un sistema. Muestra la organización y las dependencias entre un conjunto de componentes. No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes. Cada diagrama describe un apartado del sistema.

En el situaremos librerías, tablas archivos, ejecutables y documentos que formen parte del sistema.

Uno de los usos principales es que puede servir para ver que componentes pueden compartirse entre sistemas o entre diferentes partes de un sistema.

Todo objeto UML puede ser modificado mediante estereotipos, los estándares que define UML son:

- Executable
- Library
- Table
- File
- Document

Podemos modelar diferentes partes de nuestro sistema, y modelar diferentes entidades que no tiene nada que ver entre ellas.

- Ejecutables y bibliotecas
- Tablas
- API
- Código fuente
- Hojas HTML

2.3.2.4.1 Ejecutables

Nos facilita la distribución de ejecutables a los clientes. Documenta sus necesidades y dependencias. Si disponemos de un ejecutable que solo se necesita a él mismo para funcionar no necesitaremos el diagrama de componentes.

Los pasos a seguir para modelar, a priori no a posteriori, son:

- Identificar los componentes, las particiones del sistema, cuales son factibles de ser reutilizadas. Agruparlos por nodos y realizar un diagrama por cada nodo que se quiera modelar.
- Identificar cada componente con su estereotipo correspondiente.
- Considerar las relaciones entre componentes.

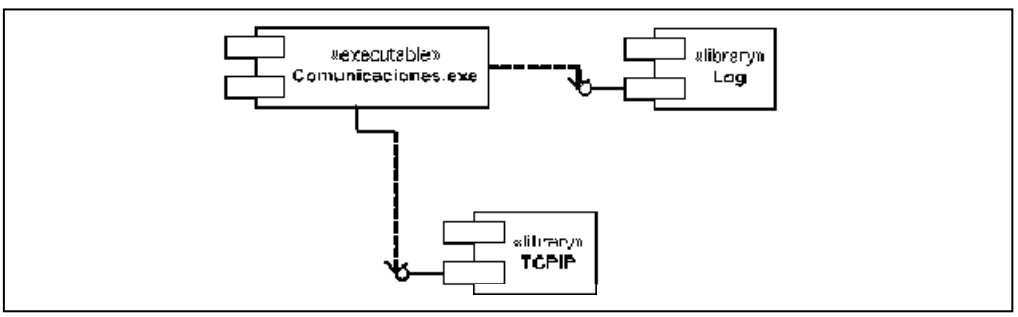


Fig. 2.12. Diagrama de componentes

2.3.2.4.2 Código fuente

Se utiliza para documentar las dependencias de los diferentes ficheros de código fuente. Un ejecutable, o librería es una combinación de estos ficheros, y al mostrar la dependencia entre ellos obtenemos una visión de las partes necesarias para la creación del ejecutable o librería.

Al tener documentadas las relaciones se pueden realizar cambios en el código de un archivo teniendo en cuenta donde se utiliza, y que otros ficheros pueden verse afectados por su modificación.

2.3.2.5 Diagramas de despliegue

En el diagrama de despliegue se indica la situación física de los componentes lógicos desarrollados. Es decir se sitúa el software en el hardware que lo contiene. Cada Hardware se representa como un nodo.

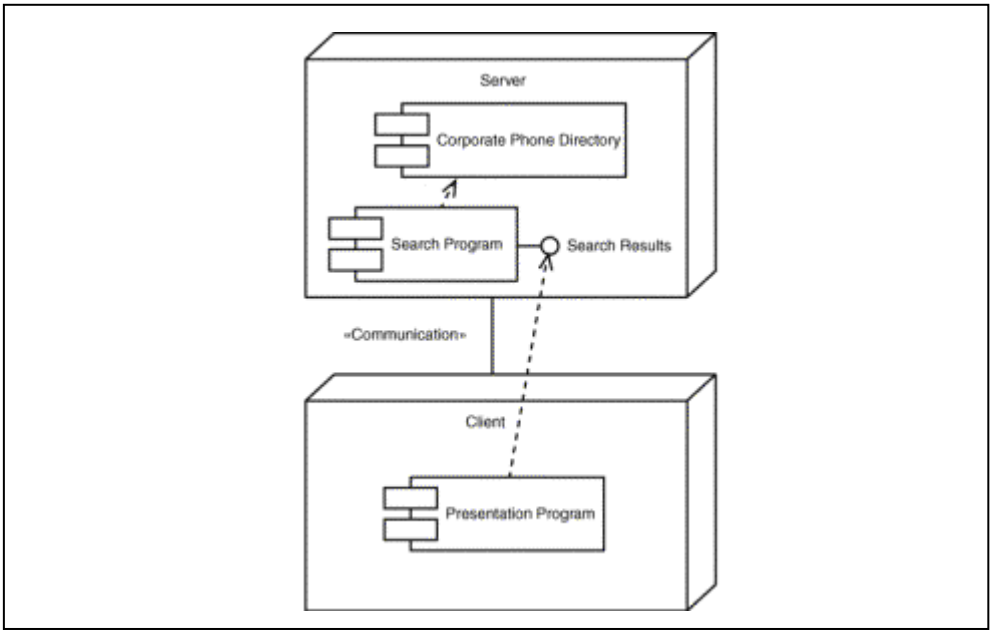


Fig. 2.13. Diagrama de despliegue

Un nodo se representa como un cubo, un nodo es un elemento donde se ejecutan los componentes, representan el despliegue físico de estos componentes.

Aquí tenemos dos nodos, el cliente y el servidor, cada uno de ellos contiene componentes. El componente del cliente utiliza un interface de uno de los componentes del servidor. Se muestra la relación existente entre los dos Nodos. Esta relación podríamos asociarle un estereotipo para indicar que tipo de conexión disponemos entre el cliente y el servidor, así como modificar su cardinalidad, para indicar que soportamos diversos clientes.

Como los componentes pueden residir en mas de un nodo podemos situar el componente de forma independiente, sin que pertenezca a ningún nodo, y relacionarlo con los nodos en los que se sitúa.

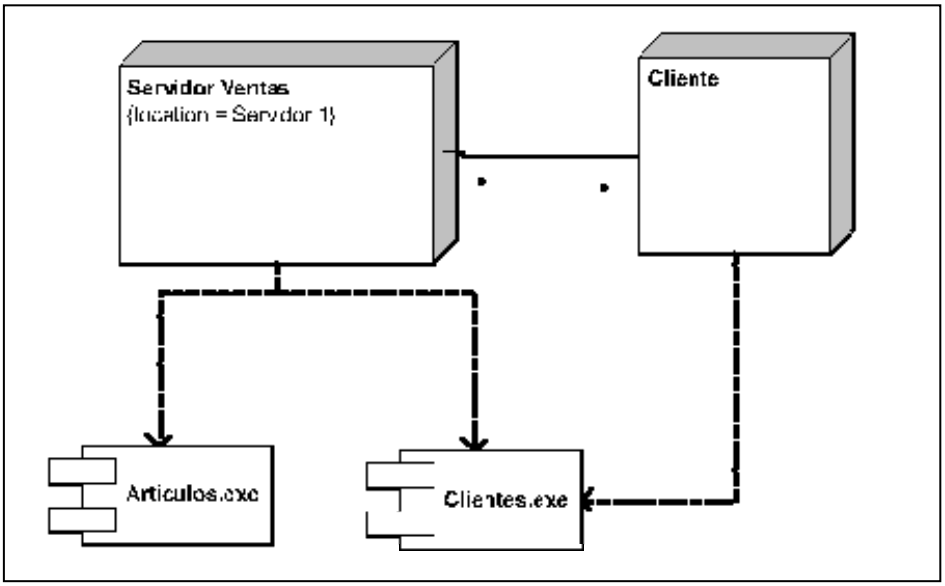


Fig. 2.14. Diagrama de componentes

2.3.2.6 Diagramas secuencia

El diagrama de secuencia forma parte del modelado dinámico del sistema. Se modelan las llamadas entre clases desde un punto concreto del sistema. Es útil para observar la vida de los objetos en sistema, identificar llamadas a realizar o posibles errores del modelado estático, que imposibiliten el flujo de información o de llamadas entre los componentes del sistema.

En el diagrama de secuencia se muestra el orden de las llamadas en el sistema. Se utiliza un diagrama para cada llamada a representar. Es imposible representar en un solo diagrama de secuencia todas las secuencias posibles del sistema, por ello se escoge un punto de partida. El diagrama se forma con los objetos que forman parte de la secuencia, estos se sitúan en la parte superior de la pantalla, normalmente en la izquierda se sitúa al que inicia la acción. De estos objetos sale una línea que indica su vida en el sistema. Esta línea simple se convierte en una línea gruesa cuando representa que el objeto tiene el foco del sistema, es decir cuando el esta activo.

2.3.3 Fases del desarrollo de un sistema

Las fases del desarrollo de sistemas que soporta UML son: *Análisis de requerimientos*, *Análisis*, *Diseño*, *Programación* y *Pruebas*.

Análisis de Requerimientos

UML tiene casos de uso (use-cases) para capturar los requerimientos del cliente. A través del modelado de casos de uso, los actores externos que tienen interés en el sistema son modelados con la funcionalidad que ellos requieren del sistema (los casos de uso). Los actores y los casos de uso son modelados con relaciones y tienen asociaciones entre ellos o éstas son divididas en jerarquías. Los actores y casos de uso son descritos en un diagrama use-case. Cada use-case es descrito en

texto y especifica los requerimientos del cliente: lo que él (o ella) espera del sistema sin considerar la funcionalidad que se implementará. Un análisis de requerimientos puede ser realizado también para procesos de negocios, no solamente para sistemas de software.

Análisis

La fase de análisis abarca las abstracciones primarias (clases y objetos) y mecanismos que están presentes en el dominio del problema. Las clases que se modelan son identificadas, con sus relaciones y descritas en un diagrama de clases. Las colaboraciones entre las clases para ejecutar los casos de uso también se consideran en esta fase a través de los modelos dinámicos en UML. Es importante notar que sólo se consideran clases que están en el dominio del problema (conceptos del mundo real) y todavía no se consideran clases que definen detalles y soluciones en el sistema de software, tales como clases para interfaces de usuario, bases de datos, comunicaciones, concurrencia, etc.

Diseño

En la fase de diseño, el resultado del análisis es expandido a una solución técnica. Se agregan nuevas clases que proveen de la infraestructura técnica: interfaces de usuario, manejo de bases de datos para almacenar objetos en una base de datos, comunicaciones con otros sistemas, etc. Las clases de dominio del problema del análisis son agregadas en esta fase. El diseño resulta en especificaciones detalladas para la fase de programación.

Programación

En esta fase las clases del diseño son convertidas a código en un lenguaje de programación orientado a objetos. Cuando se crean los modelos de análisis y diseño en UML, lo más aconsejable es trasladar mentalmente esos modelos a código.

Pruebas

Normalmente, un sistema es tratado en pruebas de unidades, pruebas de integración, pruebas de sistema, pruebas de aceptación, etc. Las pruebas de unidades se realizan a clases individuales o a un grupo de clases y son típicamente ejecutadas por el programador. Las pruebas de integración integran componentes y clases en orden para verificar que se ejecutan como se especificó. Las pruebas de sistema ven al sistema como una "caja negra" y validan que el sistema tenga la funcionalidad final que le usuario final espera. Las pruebas de aceptación conducidas por el cliente verifican que el sistema satisface los requerimientos y son similares a las pruebas de sistema.

2.4 Introducción a Java

El 15 de Enero de 1991, un grupo de desarrollo de Sun Microsystems Bill Joy, Andy Bechtolsheim, Wayne Rosing, Mike Sheridan, James Gosling y Patrick Naughton se reúnen en forma independiente y secreta a Sun (proyecto Stealth) con la finalidad de discutir el rumbo de la computación y sus actuales tendencias. Uno de los acuerdos de dicha reunión fue la conclusión de que una de las tendencias futuras del cómputo, sería el acercamiento de los sistemas digitales y la electrónica de consumo. Por lo tanto, el grupo marcó como objetivo el desarrollo de un ambiente ó entorno único que pudiera ser empleado por los dispositivos electrónicos de consumo.

El 1 de Febrero de 1991 inician los trabajos del proyecto Stealth que más tarde se llamaría Proyecto Green. El trabajo se divide: Naughton dedicado al sistema gráfico Aspen, Gosling dedicado a identificar el lenguaje de programación a utilizar en el proyecto. Después de varias modificaciones infructuosas al lenguaje C++, se llego a la creación de un nuevo lenguaje: Oak, el cual debía ser independiente de la plataforma, robusto, sencillo e interpretado, el cual daría soporte a la gran variedad de dispositivos electrónicos.

Oak contaba con similitudes de C, C++ y Objective C no ligado a un tipo de CPU en particular. Más tarde se le cambiaría el nombre de Oak a Java. Se supone que le pusieron ese nombre mientras tomaban café (Java es también el nombre de un tipo de café, originario del este de Asia, de la isla del mismo nombre), aunque hay algunos que afirman que el nombre deriva de las siglas de James Gosling, Arthur Van Hoff, y Andy Bechtolsheim.

En Marzo de 1993 el NCSA (National Center for Supercomputing Applications) liberaba Mosaic, una aplicación que permitía a los usuarios acceder a Internet de forma gráfica, pudiendo acceder a cientos de sitios, Internet crecía a pasos enormes.

En Junio de 1994 Palrang comienza el proyecto Live Oak con el objetivo de usar Oak para construir un sistema operativo y estudiar las posibilidades de negocio de Internet. Mientras Arthur van Hoff implementa el compilador de Oak en lenguaje Oak, reemplazando la versión de Gosling que se había escrito en C.

En Septiembre de 1994 Naughton y Jonathon Payne comienzan a escribir un navegador web similar a Mosaic escrito en Java, WebRunner (por la película Blade Runner) al que después se llamaría HotJava.

En Mayo de 1995 se hace la demostración a los ejecutivos de Sun del HotJava, se demuestra el potencial del lenguaje y se acepta el proyecto. Netscape anuncia su intención de utilizar la licencia de Java para uso en el navegador. En septiembre del mismo año, se concreta una alianza con Toshiba para el desarrollo de productos en recuperación remota de información, los cuales incorporan Java, posteriormente, Oracle anuncia WebSystem para la Web, que incluye Java. En diciembre del mismo año Sun y Netscape anuncian JavaScript, un lenguaje basado en Java para la web. Microsoft planea hacer uso de la licencia Java e incluye VBScript.

En Enero de 1996, Sun crea JavaSoft para desarrollar la nueva tecnología y ese mismo mes aparece la versión 1.0 del JDK.

2.4.1 Los JDK/SDK de Java

- Java 1
  - Java 1.0 Lanzada en Enero de 1996. Es la primera versión pública. Debido a la rapidez de su publicación, esta versión contiene diversos errores.
  - Java 1.1 Lanzada en Marzo de 1997. Mejoras de rendimiento en la JVM, nuevo modelo de eventos en AWT, clases anidadas, serialización de objetos, API de JavaBeans, archivos jar, internacionalización, API Reflection (Reflexión), JDBC (Java Data base Connectivity), RMI (Remote Method Invocation). Se añade la firma del código y la autentificación. Es la primera versión lo suficientemente estable y robusta.

- Java 2
  - Java 1.2 Publicada en Diciembre de 1998: Swing, Java2D, CORBA, Collections. Se producen notables mejoras a todos los niveles. Para enfatizar esto Sun lo renombra como "Java 2". El JDK (Java Development Kit) se renombra como SDK (Software Development Kit).
  - Java 1.3 Publicada en Abril del 2000. Orientada sobre todo a la resolución de errores y a la mejora del rendimiento; se producen algunos cambios menores como la inclusión de JNDI (Java Naming and Directory Interface) y la API Java Sound. También incluye un nuevo compilador de alto rendimiento JIT (Just In Time).
  - Java 1.4 Publicada en el 2002 También conocido como Merlin, mejora notablemente el rendimiento y añade entre otros soporte de expresiones regulares, una nueva API de entrada/salida de bajo nivel (NIO, New I/O), clases para el trabajo con Collections, procesado de XML; y mejoras de seguridad como el soporte para la criptografía mediante las Java Cryptography Extension (JCE), la inclusión de la Java Secure Socket Extension (JSSE) y el Java Authentication and Authorization Service (JAAS).

2.4.2 Características del lenguaje Java

Los siguientes puntos muestran las principales características del lenguaje de programación Java.

- Simple. Java ofrece toda la funcionalidad de un lenguaje de programación simplificando algunas tareas que en otros lenguajes requieren un esfuerzo adicional por parte del programador. Un ejemplo de lo anterior es el manejo de la memoria. La maquina virtual de Java (JVM) se encarga de su administración haciendo uso de un hilo de ejecución de baja prioridad encargado de detectar y liberar bloques de memoria libres de referencias (Garbage Collector). Al eliminar características como la aritmética de punteros, los registros struct, operaciones de reserva de memoria (malloc() en C), liberación de memoria (como free()en C), existencia de referencias en lugar de punteros, etc., reduce considerablemente los errores de programación generados al delegar la administración de la memoria al programador.

- Orientado a objetos. Java es un lenguaje totalmente orientado a objetos: encapsulación, herencia, polimorfismo, etc. Todos los programas en Java son clases, y su representación en memoria son las instancias de una clase. Java

incorpora la sobrecarga y sobre escritura de métodos. No soporta herencia múltiple, sin embargo, incorpora la implementación de interfaces.

- Distribuido. Java cuenta con capacidades de interconexión TCP/IP. Existen librerías para interactuar con protocolos como *http* y *ftp*. Esto permite a los programadores acceder a la información a través de la red con facilidad. Java proporciona las herramientas y librerías necesarias para la ejecución de sistemas distribuidos mediante Java RMI, JNDI, RMI-IIOP , JNI.

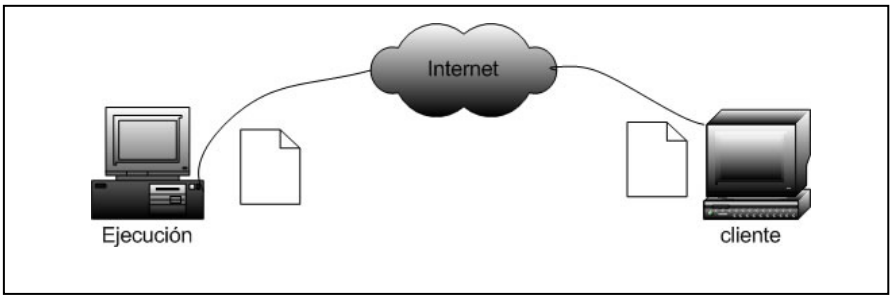


Fig. 2.15. Ejecución de aplicaciones distribuidas

- Ejecución de aplicaciones distribuidas.

- Robusto. Java proporciona diversos mecanismos con la finalidad de minimizar los errores de una aplicación en tiempo de compilación y en tiempo de ejecución, estos son algunos de ellos:

    - Tiempo de compilación.
        * Comprobación de tipos de datos
        * Requiere declaración explicita de métodos

    - Tiempo de ejecución
        * Verificación de la integridad del archivo de bytecodes
        * Manejo de memoria
        * Operaciones de comprobación de límites de índices en arreglos evitando la posibilidad de sobrescribir o corromper memoria como resultado de punteros que señalan a zonas erróneas.
        * Manejo de Excepciones

- Portable. Java es un lenguaje multiplataforma en el sentido de que una aplicación no requiere ser re-compilada ó modificada para que pueda ser ejecutada en cualquier plataforma para la cual exista una maquina virtual (JVM).

- Seguro. Antes de ejecutar un programa en Java, el código generado pasa por diversas pruebas con la finalidad de detectar posibles anomalías ó alteraciones de la integridad del código en bytecodes a interpretar:

    - El código no debe producir desbordamiento de operandos en la pila.
    - El tipo de los parámetros de todos los códigos de operación deben ser conocidos y correctos.
    - No debe existir alguna conversión ilegal de datos, por ejemplo, convertir enteros en punteros.
    - El acceso a los atributos de un objeto debe ser legal : public, private, protected
    - El código debe cumplir con las reglas de acceso y seguridad establecidas.

  De forma similar, en tiempo de compilación, existen mecanismos que evitan la generación de código inseguro ó vulnerable, En Java no existen instrucciones para la manipulación de la memoria eliminando la posibilidad de acceder a áreas ajenas ó la posibilidad de desbordamientos.

  En cuanto a código proveniente de una fuente remota (red), Java cuenta con mecanismos de seguridad. Algunos de ellos:

    - Esquema de seguridad empleado por los Applets (Sandbox) ,
    - Verificación de una llave digital por parte del cargador de clases antes de instanciar cualquier clase, etc.
    - Separación de las clases provenientes de una fuente externa y las locales con la finalidad de prevenir el indebido reemplazo de una clase externa por una local . (El cargador primero carga las clases locales y después las remotas).

    - Un aspecto considerado inseguro por los programadores es la posibilidad de la generación del código fuente a partir del archivo en bytecodes (.class) mediante el comando *javap.*

    - Multihilos. En java es posible ejecutar varias aplicaciones a la vez mediante el empleo de hilos de ejecución, lo que permite incrementar y optimizar el desempeño de una aplicación, al permitir la distribución de las tareas a realizar asociadas a hilos de ejecución debidamente organizadas y sincronizadas. La programación con multihilos es fundamental para el desarrollo de interfaces gráficas (swing), aplicaciones distribuidas, etc.

    - Para el modulo de Operación y control del pago del SIAR, esta característica es importante, como se verá mas adelante, debido principalmente a la cantidad de tareas que se requieren ejecutar al calcular una nómina.

- Dinámico. Java es dinámico debido a que la carga de las clases se realiza cuando estas son requeridas, ya sea de forma local ó de algún punto de la red empleando algún protocolo de comunicación.

2.4.2.1 La Máquina Virtual de Java (JVM)

La máquina virtual de Java es la responsable de la interpretación y ejecución de los programas Java y de varias de las características del lenguaje descritas anteriormente como la portabilidad, la eficiencia y la seguridad.

Al ejecutar un programa en Java, las instrucciones o bytecodes que lo componen, no son ejecutadas directamente en el hardware de la computadora, sino que son pasadas a un elemento de software intermedio que es el encargado de que dichas instrucciones sean ejecutadas por el hardware. Es decir, el código no se ejecuta directamente sobre un procesador físico, sino sobre un procesador virtual Java.

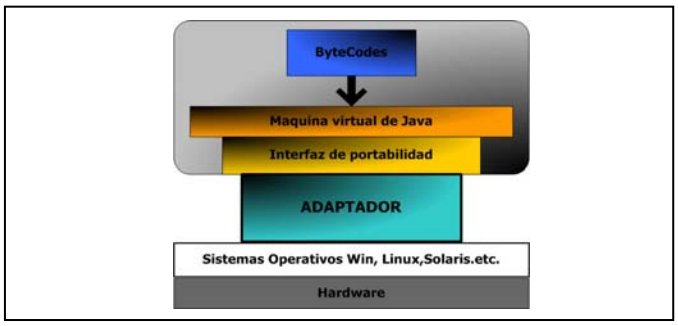


Fig. 2.16. Máquina virtual de java implementada para diversas plataformas

La figura muestra 2 capas adicionales: El *adaptador,* el cual es dependiente de la plataforma, *y la interfaz de portabilidad* independiente de la plataforma. Esta última es la encargada de ocultar las particularidades de cada plataforma para las aplicaciones Java. Al incorporar una nueva plataforma solo se requiere generar el adaptador específico de la nueva plataforma.

La máquina virtual de java implementada para diversas plataformas.

2.4.2.2 Arquitectura de la Máquina Virtual de Java (JVM)

El término máquina virtual se refiere a la especificación abstracta de una máquina de software para ejecutar programas escritos en Java. Esta especificación define elementos como el formato de los archivos de clases de Java (*.class*), así como la semántica de cada una de las instrucciones que componen el conjunto de instrucciones de la máquina virtual. A las implementaciones de esta especificación se les conocen como Ambientes en Tiempo de Ejecución Java (Java RunTime Environment JRE).

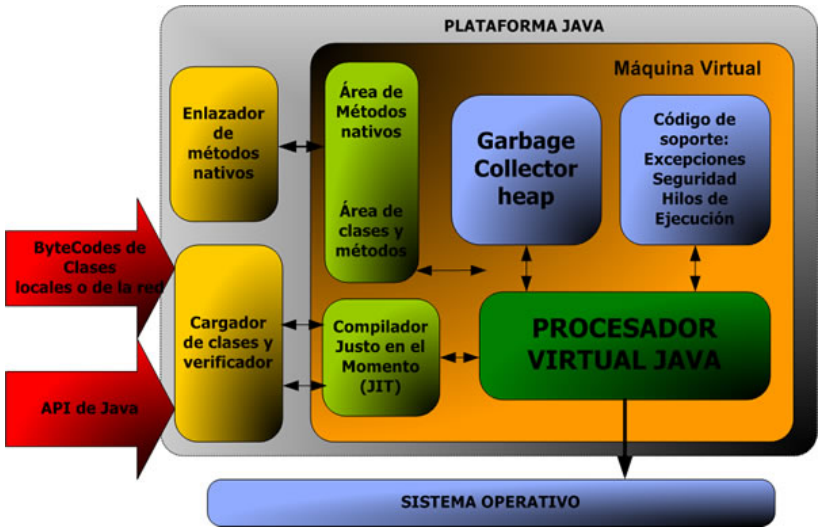


Fig. 2.17. Arquitectura de la Máquina Virtual de Java

Arquitectura JRE. Se muestran los componentes del ambiente de ejecución Java:

- Procesador Virtual Java. Es el encargado de ejecutar las instrucciones contenidas en los archivos .class (bytecodes). En versiones iníciales, este procesador consistía de un intérprete de códigos de operación. En versiones recientes, se utiliza la tecnología *Just in Time code generation* (JIT) en donde las instrucciones son convertidas a código nativo la primer vez que se ejecuta el código, lo cual incrementa el rendimiento de la aplicación.

- Muchas de las instrucciones del procesador virtual Java son similares a las instrucciones de un procesador común como los Intel, por ejemplo, instrucciones aritméticas, de acceso a memoria, etc., pero con mayor complejidad. Una de las características más significativas del conjunto de instrucciones del procesador virtual Java, es que están basadas en la pila y utilizan "posiciones de memoria" numeradas, en lugar de registros, debido a que la máquina virtual debe trabajar con diversas arquitecturas de procesadores.

- Verificador de Java. Agrega los elementos de seguridad descritos anteriormente, con la finalidad de minimizar comportamientos inciertos, ejecución de código alterado, etc. Una vez obtenido el código en bytecodes la maquina virtual se encarga de ejecutar dicho verificador.

- Administrador de memoria. Para llevar a acabo la administración de memoria, se emplean algoritmos para la detección de los objetos candidatos para su recolección mediante el seguimiento de sus referencias:

  - Algoritmo *contabilizador de referencias*. A cada objeto se le asocia un contador que indica el número de referencias hacia este. Cuando dicho contador llega a ser 0, el objeto es candidato a liberar su memoria asociada.
  - *Marcar e intercambiar (mark and sweep)*. Consiste en almacenar los objetos en un espacio de memoria heap en el que se marca de forma

periódica a todos los objetos que no tengan alguna referencia hacia ellos. Adicionalmente existe otra área de memoria a la que se mueven todos aquellos objetos que no han sido marcados en forma periódica. El recolector de basura se encarga de actualizar las referencias de este segundo heap. De esta forma, se tiene un heap únicamente con los objetos que se están utilizando, por lo que el primer heap es candidato para su recolección.

- Manejo de excepciones. Es la forma en la que Java indica que algo extraño e inesperado a sucedido. Las excepciones son generadas y lanzadas por la aplicación, para ello se emplea la instrucción de la JVM *atrhow.* Las excepciones son manejadas por *handlers* encargados de ejecutar las instrucciones especificadas cuando se origina la excepción. Si no existe un handler asociado, se emplea el handler del sistema: imprimir el mensaje notificando la excepción y termina la ejecución.

- Soporte para métodos nativos. Una aplicación en Java puede contener métodos implementados en otros lenguajes de programación como C y C++. Este código es incluido en forma dinámica en tiempo de ejecución. Una vez que se enlaza el módulo que contiene el código que implementa dicho método, el procesador virtual atrapa las llamadas a éste y se encarga de invocarlo. Este proceso incluye la modificación de los argumentos de la llamada, para adecuarlos al formato que requiere el código nativo, así como transferirle el control de la ejecución.
- Interfaz multihilos. Para dar soporte a la capacidad multihilos de java, se cuenta con la posibilidad de simular más de un procesador virtual, cada uno con sus propios elementos en el que se ejecutan los diferentes flujos. Los procesadores virtuales pueden ser simulados por software o implementados mediante llamadas al sistema operativo.

- Carga dinámica de clases. Como se mencionó anteriormente, una aplicación Java esta 100% organizada en clases. Al ejecutarse, se realiza la carga de clases conforme a la demanda, es decir, las clases se cargan en memoria hasta que esta es requerida. Existen 2 formas de cargar clases: la carga de las clases del sistema (clases estándar de java), y la segunda forma es mediante una instancia de la clase java.lang.ClassLoader.

A pesar de todas las mejoras, optimizaciones e integración de nuevas tecnologías para los lenguajes basados en maquinas virtuales, no se ha alcanzado un rendimiento alto en comparación con los lenguajes compilados específicos de la plataforma, esto debido principalmente a la integración de una capa intermedia de software entre el código generado y el hardware de la máquina.

El código Java, compilado *justo en el momento*, tampoco puede competir con el código completamente compilado como el de C, debido a que los compiladores son diseñados para obtener todas las ventajas posibles de su arquitectura, mientras que Java JIT cuenta con las restricciones que impone el diseño de instrucciones del procesador virtual Java.

Aun con lo anterior, lo cierto es que Java a revolucionado en gran parte la computación de hoy en día con las características que se han descrito en esta

sección las cuales son de vital importancia en el desarrollo de los sistemas que demandan las necesidades actuales, y mas aun las aplicaciones empresariales: portabilidad, escalabilidad, seguridad.

2.5 Patrón MVC

La arquitectura MVC (*Model/View/Controller*) fue introducida como parte de la versión Smalltalk-80 del lenguaje de programación Smalltalk. Fue diseñada para reducir el esfuerzo de programación necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos. 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.

En la figura siguiente, 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.

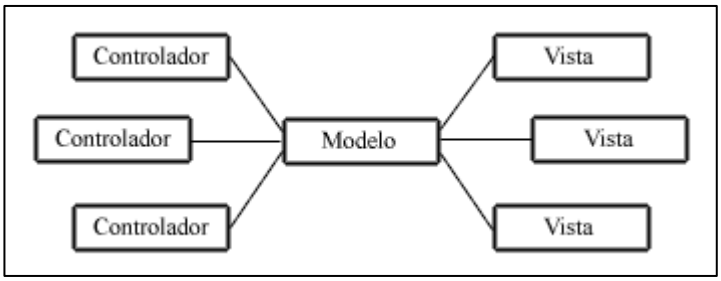


Fig. 2.18. Arquitectura Modelo-Vista-Controlador

Este modelo de arquitectura presenta varias ventajas:

- Hay una clara separación entre los componentes de un programa; lo cual nos permite implementarlos por separado
- 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.

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. Este escenario contrasta con la aproximación monolítica típica de muchos programas Java. Todos tienen un *Frame* que contiene todos los elementos, un controlador de eventos, un montón de cálculos y la presentación del resultado. Ante esta perspectiva, hacer un cambio aquí no es nada trivial.

2.5.1 Definición de las partes

El *Modelo* es el objeto que representa los datos del programa. Maneja los datos y controla todas sus transformaciones. 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.

La *Vista* es el objeto que maneja la presentación visual de los datos representados por el Modelo. Genera una representación visual del Modelo y muestra los datos al usuario. Interactúa con el Modelo a través de 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. Interactúa con el Modelo a través de una referencia al propio Modelo.

Observador y observable

El lenguaje de programación Java proporciona soporte para la arquitectura MVC mediante dos clases:

- Observador: Es cualquier objeto que desee ser notificado cuando el estado de otro objeto sea alterado
- Observable: Es cualquier objeto cuyo estado puede representar interés y sobre el cual otro objeto ha demostrado ese interés

Estas dos clases se pueden utilizar para muchas más cosas que la implementación de la arquitectura MVC. Serán útiles en cualquier sistema en que se necesite que algunos objetos sean notificados cuando ocurran cambios en otros objetos.

El Modelo es un subtipo de Observable y la Vista es un subtipo de Observador. Estas dos clases manejan adecuadamente la función de notificación de cambios que necesita la arquitectura MVC. Proporcionan el mecanismo por el cual las Vistas pueden ser notificadas automáticamente de los cambios producidos en el Modelo. Referencias al objeto Modelo tanto en el Controlador como en la Vista permiten acceder a los datos de ese objeto Modelo.

En el caso de Java, las partes del Patrón MVC, están conformadas por los siguientes elementos de programación.

- Modelo: Definido por los Java Beans
- Vista: Definida por los JSP
- Controlador: Está constituido por Servlets o Clases Java

A continuación se definen cada uno de éstos elementos:

JSP

Java Server Pages (JSP) es la tecnología para generar páginas web de forma dinámica en el servidor, desarrollado por Sun Microsystems, basado en scripts que

utilizan una variante del lenguaje java. Este tipo de tecnología nos permite mezclar tags de HTML estático con tags HTML generado dinámicamente, la parte dinámica está escrita en Java, permite el uso de servlets.
Ventajas de JSP
Podemos crear aplicaciones web que se ejecuten en varios servidores web, de múltiples plataformas, ya que Java es en esencia un lenguaje multiplataforma.
Las páginas JSP están compuestas de código HTML/XML mezclado con etiquetas especiales para programar scripts ejecutables en el servidor en sintaxis Java. Por lo tanto, las JSP podremos escribirlas con nuestro editor HTML/XML habitual.

El motor JSP

El motor de las páginas JSP está basado en los servlets, que son programas en Java destinados a ejecutarse en el servidor, aunque el número de desarrolladores que pueden afrontar la programación de JSP es mucho mayor, debido a que la programación de los servlets es más complicada.

En JSP creamos páginas de manera parecida a como se crean en ASP o PHP -otras dos tecnologías ejecutables en el servidor-. Se generan archivos que incluyen, dentro de la estructura de etiquetas HTML, las sentencias Java a ejecutar en el servidor. Antes de que sean funcionales los archivos, el motor JSP lleva a cabo una fase de traducción de esa página en un servlet, implementado en un archivo class (Byte codes de Java). Esta fase de traducción se lleva a cabo habitualmente cuando se recibe la primera solicitud de la página jsp, aunque existe la opción de precompilar el código para evitar ese tiempo de espera la primera vez que un cliente solicita la página.

JavaBeans

Un JavaBean o bean es un componente hecho en software que se puede reutilizar y que puede ser manipulado visualmente por una herramienta de programación en lenguaje Java.

Para ello, se define un interfaz para el momento del diseño (design time) que permite a la herramienta de programación o IDE, interrogar (query) al componente y conocer las propiedades (properties) que define y los tipos de sucesos (events) que puede generar en respuesta a diversas acciones.

Aunque los beans individuales pueden variar ampliamente en funcionalidad desde los más simples a los más complejos, todos ellos comparten las siguientes características:

- Introspection: Permite analizar a la herramienta de programación o IDE como trabaja el bean
- Customization: El programador puede alterar la apariencia y la conducta del bean.
- Events: Informa al IDE de los sucesos que puede generar en respuesta a las acciones del usuario o del sistema, y también los sucesos que puede manejar.
- Properties: Permite cambiar los valores de las propiedades del bean para personalizarlo (customization).
- Persistence: Se puede guardar el estado de los beans que han sido personalizados por el programador, cambiando los valores de sus propiedades.

- En general, un bean es una clase que obedece ciertas reglas:
- Un bean tiene que tener un constructor por defecto (sin argumentos)
- Un bean tiene que tener persistencia, es decir, implementar el interface *Serializable*.
- Un bean tiene que tener introspección (instrospection). Los IDE reconocen ciertas pautas de diseño, nombres de las funciones miembros o métodos y definiciones de las clases, que permiten a la herramienta de programación mirar dentro del bean y conocer sus propiedades y su conducta.

Propiedades

Una propiedad es un atributo del JavaBean que afecta a su apariencia o a su conducta. Por ejemplo, un botón puede tener las siguintes propiedades: el tamaño, la posición, el título, el color de fondo, el color del texto, si está o no habilitado, etc.

Las propiedades de un bean pueden examinarse y modificarse mediante métodos o funciones miembro, que acceden a dicha propiedad, y pueden ser de dos tipos:

- getter method: lee el valor de la propiedad
- setter method: cambia el valor de la propiedad.

Un IDE que cumpla con las especificaciones de los JavaBeans sabe como analizar un bean y conocer sus propiedades. Además, crea una representación visual para cada uno de los tipos de propiedades, denominada editor de propiedades, para que el programador pueda modificarlas fácilmente en el momento del diseño.

Cuando un programador, coge un bean de la paleta de componentes y lo deposita en un panel, el IDE muestra el bean sobre el panel. Cuando seleccionamos el bean aparece una hoja de propiedades, que es una lista de las propiedades del bean, con sus editores asociados para cada una de ellas.

El IDE llama a los métodos o funciones miembro que empiezan por get, para mostrar en los editores los valores de las propiedades. Si el programador cambia el valor de una propiedad se llama a un método cuyo nombre empieza por set, para actualizar el valor de dicha propiedad y que puede o no afectar al aspecto visual del bean en el momento del diseño.

Las especificaciones JavaBeans definen un conjunto de convenciones (design patterns) que el IDE usa para inferir qué métodos corresponden a propiedades.

```
public void setNombrePropiedad(TipoPropiedad valor)
public TipoPropiedad getNombrePropiedad( )
```

Cuando el IDE carga un bean, usa el mecanismo denominado *reflection* para examinar todos los métodos, fijándose en aquellos que empiezan por set y get. El IDE añade las propiedades que encuentra a la hoja de propiedades para que el programador personalice el bean.

Servlets

Un servlet de forma intuitiva se puede definir como un programa independiente de plataforma que aporta la misma funcionalidad a la programación en el lado del servidor que tradicionalmente han realizado la interfaz CGI.Con respecto a esta tecnología aporta numerosas ventajas que citaremos a continuación:

- Independencia de la plataforma. Esto proporciona un menor esfuerzo de codificación con respecto a soluciones dependientes del servidor web y de la plataforma como ISAPI o NSAPI.
- Ejecución en paralelo de múltiples peticiones por una sola instancia del servlet. Tradicionalmente en los programas CGI se ejecuta un proceso distinto para cada petición lo que conlleva una gradual degradación del rendimiento y una necesidad de recursos muy elevada. En un servlet todas las peticiones se atienden en el mismo proceso por distintos hilos y una vez que se ha cargado el servlet este permanece en memoria hasta que se reinicie el servidor o hasta que se le diga lo contrario con lo cual las subsiguientes peticiones son mas rápidas al encontrarse el programa ya cargado en memoria.

Estructura de un servlet

El API Servlet consiste básicamente en dos paquetes:

- javax.servlet En este paquete se definen 6 interfaces y 3 clases para la implementación de servlets genéricos, sin especificación de protocolo. Hoy en día no tienen utilidad práctica más que para servir de base en la jerarquía de clases de los servlets. Conforme pase el tiempo se supone que constituirán la base para la implementación de otros protocolos distintos de http.
- javax.servlet.http Ofrece la implementación especifica de servlets para el protocolo http.

En estos paquetes se definen todas las clases e interfaces necesarias para la escritura de applets. Al utilizar los servlets para gestionar conexiones http usaremos las clases del paquete javax.servlet.http.

2.6 Postgresql

PostgreSQL es un Sistema de Gestión de Bases de Datos Objeto-Relacionales (ORDBMS) que ha sido desarrollado de varias formas desde 1977. Comenzó como un proyecto denominado *Ingres* en la Universidad Berkeley de California. *Ingres* fue más tarde desarrollado comercialmente por la *Relational Technologies/Ingres Corporation*.

En 1986 otro equipo dirigido por *Michael Stonebraker* de Berkeley continuó el desarrollo del código de *Ingres* para crear un sistema de bases de datos objeto-relacionales llamado *Postgres*. En 1996, debido a un nuevo esfuerzo de código abierto y a la incrementada funcionalidad del software, *Postgres* fue renombrado a *PostgreSQL*, tras un breve periplo como *Postgres95*. El proyecto *PostgreSQL* sigue

actualmente un activo proceso de desarrollo a nivel mundial gracias a un equipo de desarrolladores y contribuidores de código abierto.

Características PostgreSQL

PostgreSQL está ampliamente considerado como el sistema de bases de datos de código abierto más avanzado del mundo.

PostgreSQL dispone de las siguientes características:

- Transacciones
- Subselects
- Triggers
- Vistas
- Integridad referencial de claves externas
- Un sofisticado sistema de bloqueos

Además, también presenta algunas funcionalidades que no encontramos en otras bases de datos comerciales, como tipos de datos definibles por el usuario, herencia, reglas, y control de concurrencia multi-version que permite reducir el bloqueo de conexión.

A continuación se presenta una comparación de PostgreSQL con los demás motores:

- Es posible instalarlo un número ilimitado de veces sin temor a sobrepasar la cantidad de licencias, la principal preocupación de muchos proveedores de bases de datos comerciales.
- Velocidad y rendimiento
- Flexibilidad para extenderse según se requiera
- Diseño escalable
- Mínimos requerimientos de administración
- Sigue estándares ANSI
- Excelente Atomicidad, Consistencia, Aislamiento y Durabilidad (Prueba del ACID)
- El siguiente cuadro ilustra el comportamiento de PostgreSQL contra otros motores de datos.

	MySQL	Microsoft	Oracle	PostgreSQL
Conexiones simultáneas (instalación default)	101	1000	41	32
Columnas en tabla	2819	1024	1000	1600
Máximo tamaño de renglón en tabla (omitiendo blobs)	65534	8036	255000	103275
Tamaño del renglón en tabla sin nulos (omitiendo blobs)	65502	8036	255000	103275
Tamaño query	1048574	16777216	16777216	16777216
Valor FALSE	0			0
Valor TRUE	1			1

Capítulo 3

Diseño del Sistema

3.1 Planeación

La planeación del sistema está representada mediante un diagrama de Gantt (ver Anexo 2), en él se detallan las actividades más sobresalientes del diseño e implementación del sistema.

3.2 Análisis

Después del análisis de requerimientos de acuerdo a lo solicitado en el Anexo 1, llegamos a la conclusión de que era indispensable que la aplicación estuviera disponible en web y para consultar las actividades en el Centro de Docencia desde cualquier computadora y la mejor opción para el desarrollo de acuerdo a las necesidades presentadas fue J2EE.

En nuestra aplicación se aplicarán las ventajas de J2EE en cuestión de portabilidad, ya que soporta múltiples sistemas operativos. Al ser una plataforma basada en el lenguaje Java, es posible desarrollar arquitecturas basadas en J2EE utilizando cualquier sistema operativo donde se pueda ejecutar una máquina virtual Java.

Una gran ventaja de J2EE es que posee soluciones libres, por lo que es posible crear arquitectura completas basadas única y exclusivamente en productos de software libre.

El almacenamiento, consultas y modificación de información se procesará en los servlets correspondientes a las diferentes tablas de la Base de Datos, de esta manera, sólo se regresa una respuesta a los JSP; por otro lado, la conexión a dicha Base de Datos se realiza mediante un controlador JDBC, por lo que no es necesario configurar en la computadora cliente una fuente de datos.

A continuación se presentan los casos de uso definidos en la etapa de análisis, cada uno de estos casos, se presentará en la siguiente sección con sus respectivos diagramas de secuencia.

Los casos de uso por actor son los siguientes:

UCA) Administrador del sistema

Actor: Administrador del Sistema.

Casos de Uso: Autenticación, Alta Dependencia, Alta Área, Alta División (Alta Responsable de División), Alta Jefe de división, Alta Actividad, Alta Actividad Con Plantilla, Alta Nombramiento, Modificación Dependencia, Modificación División (Modificación Responsable de División), Modificación Jefe de División, Modificación Actividad, Baja Dependencia, Baja Actividad.

Propósito: El propósito de éste caso de uso es mostrar las acciones al que tiene acceso el administrador del sistema.

UCB) Responsable de división

Actor: Responsable de División.

Casos de Uso: Autenticación, Alta Actividad, Control de Profesor (Alta Profesor, Modificar Profesor, Inscribir Profesor a Actividad, Calificar Profesor.

Propósito: El propósito de éste caso de uso es mostrar las acciones al que tiene acceso el responsable de división.

UCC) Jefe de división

Actor: Jefe de División.

Casos de Uso: Autenticación, Consultar Información Profesor (Datos, Actividades Inscritas, Actividades Acreditadas, Actividades No Acreditadas).

Propósito: El propósito de éste caso de uso es mostrar las acciones al que tiene acceso el jefe de división.

UCD) Profesor

Actor: Profesor.

Casos de Uso: Búsqueda de Actividades por:

- Dependencia
- Fecha
- División
- Disciplina
- Curso
- Evento

- Mostrar Todo

Propósito: El propósito de éste caso de uso es mostrar las acciones al que tiene acceso el profesor.

UC1) Autenticación

Caso de Uso: **Autenticación**

Actor: Administrador, Responsable de División, Jefe de División.

Precondición:
- El usuario debe contar con una clave y password válido.
- El usuario debe accesar al sistema

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	Acceder a la página inicial del sistema.	2	Mostrar la página de acceso, donde se encuentra la clave y password.	
3	Captura la clave y password de acceso. Acepta los datos.	4	Despliega la página principal del sistema.	E1, E2

Excepciones:

ID	Nombre	Acción
E1	Datos inválidos	Mensaje de alerta indicando que los datos son erróneos y permite nuevamente su captura.
E2	Usuario no registrado.	Mensaje de error de usuario no registrado.

Poscondiciones: Regresa a pantalla inicial.

UC2) Alta de dependencia

Caso de Uso: **Alta de dependencia.**

Propósito: El propósito de éste caso de uso, es mostrar el flujo al dar de alta una dependencia en el sistema SICED.

Actor: Administrador del sistema

Precondición:

- El usuario debe estar dentro del sistema.
- El usuario debe tener una clave de usuario, que le es proporcionada cuando es dado de alta en el sistema.
- El usuario debe registrar los datos de l dependencia a ingresar.

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	Llenar formato “Alta dependencia”	2	Verificación de los datos Registro de la dependenci a.	E1,E2, E3

Excepciones:

ID	Nombre	Acción
E1	Falta de Campos	Mensaje de alerta indicando que falta el llenado de campos
E2	Datos Incongruentes	Mensaje de error en el llenado del formato.
E3	La dependencia ya existe.	Mensaje de aviso de aviso: dependencia ya existente

Poscondiciones: Regresa a pantalla “Alta dependencia”.

UC3) Alta de área

Caso de Uso: **Alta de área.**

Propósito: Mostrar el flujo al dar de alta un área dentro del sistema SICED

Actor: Administrador

Precondición:

- El usuario debe estar dentro del sistema.
- El usuario debe tener una clave de usuario, que le es proporcionada cuando es dado de alta en el sistema.
- El usuario debe registrar los datos del área a ingresar.

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	Llenar formato “Alta Área”	2	Verificación de los datos Registro del Área.	E1,E2

Excepciones:

ID	Nombre	Acción
E1	Falta de Campos	Mensaje de alerta indicando que falta el llenado de campos
E2	Datos Incongruentes	Mensaje de error en el llenado del formato.

Poscondiciones: Regresa a pantalla “Alta Área”.

UC4) Alta de división

Caso de Uso: **Alta de División**

Propósito: Mostrar el flujo al dar de alta una división dentro del sistema SICED

Actor: Administrador

Precondición:
- El usuario debe estar dentro del sistema.
- El usuario debe tener una clave de usuario, que le es proporcionada cuando es dado de alta en el sistema.
- El usuario debe registrar los datos de la división a ingresar

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	Llenar formato “Alta División”	2	Verificación de los datos Registro de la División	E1, E2, E3

Excepciones:

ID	Nombre	Acción
E1	Falta de Campos	Mensaje de alerta indicando que falta el llenado de campos
E2	Datos Incongruentes	Mensaje de error en el llenado del formato.
E3	La división ya existe.	Mensaje de aviso de división ya registrada.

Poscondiciones: Regresa a pantalla "Alta división".

UC4.1) Alta responsable de división

Caso de Uso: **Alta responsable de división**

Actor: Administrador

Propósito: Mostrar el flujo al dar de alta un responsable de división dentro del sistema SICED

Precondición:

- El usuario debe estar dentro del sistema
- El usuario debe tener una clave de usuario, que le es proporcionada cuando es dado de alta en el sistema
- El usuario debe registrar los datos del responsable de división en la forma alta división

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	Llenar formato "Alta Responsable de División"	2	Verificación de los datos Registro de la División.	E1, E2, E3, E4

Excepciones:

ID	Nombre	Acción
E1	Falta de Campos	Mensaje de alerta indicando que falta el llenado de campos
E2	Datos Incongruentes	Mensaje de error en el llenado del formato
E3	El responsable de división ya existe.	Mensaje de aviso del responsable de división ya está registrado.
E4	Las contraseñas no coinciden.	Mensaje de aviso de confirmación de contraseña no coincide.

Poscondiciones: Regresa a pantalla "Alta responsable de división"

UC5) Alta de jefe de división

Caso de Uso: **alta jefe de división**

Actor: Administrador

Propósito: Mostrar el flujo al dar de alta un jefe de división dentro del sistema SICED

Precondición:
- El usuario debe estar dentro del sistema.
- El usuario debe tener una clave de usuario, que le es proporcionada cuando es dado de alta en el sistema.
- Se debe tener registrada al menos una división.
- El usuario debe registrar los datos del jefe de división a ingresar.

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El administrador elige la opción para Alta de Jefe de División	2	Despliega la pantalla correspondiente y solicita el nombre de la división.	E1
3	Llenar formato "Alta Jefe de División"	4	Verificación de los datos del Jefe de División.	E2, E3

Excepciones:

ID	Nombre	Acción
E1	El responsable de división ya existe.	Mensaje de aviso de que todas las divisiones tienen un jefe asignado.
E2	Datos Incongruentes	Mensaje de error en el llenado del formato
E1	Falta de Campos	Mensaje de alerta indicando que falta el llenado de campos

Poscondiciones: Regresa a pantalla alta jefe de división.

UC6) Alta de actividades

Caso de Uso: **Alta de Actividad**

Actor: Administrador

Propósito: Mostrar el flujo al dar de alta una actividad dentro del sistema SICED

Precondición:

- El usuario debe estar dentro del sistema
- El usuario debe tener una clave de usuario, que le es proporcionada cuando es dado de alta en el sistema
- Se debe tener registrada al menos una dependencia
- Se debe tener registrada al menos una división
- El usuario debe registrar los datos de la actividad a ingresar

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	Llenar formato “Alta Actividad”	2	Verificación de los datos de Alta de Actividad	E1,E2, E3

Excepciones:

ID	Nombre	Acción
E1	Falta de Campos	Mensaje de alerta indicando que falta el llenado de campos
E2	Datos Incongruentes	Mensaje de error en el llenado del formato
E3	El responsable de división ya existe.	Mensaje de aviso de que todas las divisiones tienen un jefe asignado.

Poscondiciones: Regresa a pantalla alta actividad

UC7) Alta de nombramientos

Caso de Uso: **Alta de Nombramiento**

Propósito: Mostrar el flujo al dar de alta un nombramiento dentro del sistema SICED

Actor: Administrador

Precondición:
- El usuario debe estar dentro del sistema
- El usuario debe tener una clave de usuario, que le es proporcionada cuando es dado de alta en el sistema
- El usuario debe registrar los datos del nombramiento a ingresar

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	Se llena formato “Alta Nombramientos”.	2	Verificación de los datos Registro del Nombramiento.	E1, E2

Excepciones:

ID	Nombre	Acción
E1	Falta de Campos	Mensaje de alerta indicando que falta el llenado de campos.
E2	Datos Incongruentes	Mensaje de error en el llenado del formato.

Poscondiciones: Regresa a pantalla “Alta Nombramiento”

UC8) Modificar dependencia

Caso de Uso: **Modificar Dependencia**

Propósito: Mostrar el flujo al modificar los datos de una dependencia dentro del sistema SICED.

Actor: Administrador.

Precondición:

- El usuario debe estar dentro del sistema
- El usuario debe tener una clave de usuario, que le es proporcionada cuando es dado de alta en el sistema
- Se debe tener registrada al menos una dependencia
- El usuario debe buscar la dependencia e ingresar los nuevos datos del registro a modificar

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El actor selecciona la opción “Modificar Dependencia”	2	Seleccionar la dependencia que se desea modificar.	E1
3	El sistema regresa un formulario editable con los datos actuales de la dependencia.	4	Se introducen los nuevos datos de la dependencia y se guardan.	E2, E3
5	Informa que la información ha sido guardada exitosamente.			

Excepciones:

ID	Nombre	Acción
E1	No existen dependencias registradas.	Mensaje de alerta indicando que no hay dependencias ingresadas.
E2	Falta de Campos	Mensaje de alerta indicando que falta el llenado de campos.
E3	Datos Incongruentes	Mensaje de error en el llenado del formato.

Poscondiciones: Regresa a pantalla “Modificar Dependencia”

UC9) Modificar divisiones

Caso de Uso: **Modificar División**

Propósito: Mostrar el flujo al modificar los datos de una división dentro del sistema SICED

Actor: Administrador.

Precondición:
- El usuario debe estar dentro del sistema.
- El usuario debe tener una clave de usuario, que le es proporcionada cuando es dado de alta en el sistema.
- El usuario debe buscar la división e ingresar los nuevos datos del registro a modificar.
- Se debe tener registrada al menos una división.
- Se debe contar con la contraseña del responsable de la división ya que se necesita confirmar, en caso contrario se deberá ingresar una nueva contraseña.

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El actor selecciona la opción “Modificar División”	2	Seleccionar la división que se desea modificar.	E1
3	El sistema regresa un formulario editable con los datos actuales de la división.	4	Se introducen los nuevos datos de la división y se guardan.	E2, E3, E4
5	Informa que la información ha sido guardada exitosamente.			

Excepciones:

ID	Nombre	Acción
E1	No existen divisiones registradas.	Mensaje de alerta indicando que no hay divisiones ingresadas.
E2	Falta de Campos	Mensaje de alerta indicando que falta el llenado de campos.
E3	Datos Incongruentes	Mensaje de error en el llenado del formato.
E4	Las contraseñas no	Mensaje de aviso de

	coinciden.	confirmación de contraseña no coincide.

Poscondiciones: Regresa a pantalla "Modificar División".

UC10) Modificar jefes de división

Caso de Uso: **Modificar Jefe de División.**

Propósito: Mostrar el flujo al modificar los datos del jefe de división dentro del sistema SICED.

Actor: Administrador.

Precondición:

- El usuario debe estar dentro del sistema.
- El usuario debe tener una clave de usuario, que le es proporcionada cuando es dado de alta en el sistema.
- El usuario debe buscar la división e ingresar los nuevos datos del registro a modificar.
- Se debe tener registrado al menos un jefe de división.
- Se debe contar con la contraseña del jefe de la división ya que se necesita confirmar, en caso contrario se deberá ingresar una nueva contraseña.

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El actor selecciona la opción "Modificar Jefe de División"	2	Seleccionar la división a la que pertenece el jefe de división que se desea modificar.	E1
3	El sistema regresa un formulario editable con los datos actuales del jefe de división.	4	Se introducen los nuevos datos del jefe de división y se guardan.	E2, E3, E
5	Informa que la información ha sido guardada exitosamente.			

Excepciones:

ID	Nombre	Acción
E1	No existen jefes de división registrados.	Mensaje de alerta indicando que no hay jefes de división ingresados.
E2	Falta de Campos	Mensaje de alerta indicando que falta el llenado de campos.
E3	Datos Incongruentes	Mensaje de error en el llenado del formato.
E4	Las contraseñas no coinciden.	Mensaje de aviso de confirmación de contraseña no coincide.

Poscondiciones: Regresa a pantalla "Modificar Jefe de División".

UC11) Modificar actividades

Caso de Uso: **Modificar actividad.**

Propósito: Mostrar el flujo al modificar los datos de la actividad dentro del sistema SICED.

Actor: Administrador

Precondición:
- El usuario debe estar dentro del sistema.
- El usuario debe tener una clave de usuario, que le es proporcionada cuando es dado de alta en el sistema.
- Se debe tener registrada al menos una actividad.
- El usuario debe buscar la actividad e ingresar los nuevos datos del registro a modificar.

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El actor selecciona la opción "Modificar Actividades"	2	Seleccionar el tipo y nombre de actividad que se desea modificar.	E1, E2
3	El sistema regresa un formulario editable con los datos actuales de la actividad.	4	Se introducen los nuevos datos de la actividad y se guardan.	E3, E4
5	Informa que la			

	información ha sido guardada exitosamente.			

Excepciones:

ID	Nombre	Acción
E1	No existen actividades registradas.	Mensaje de alerta indicando que no hay actividades ingresadas.
E2	No se eligió tipo y nombre de actividad.	Mensaje de alerta indicando que se debe elegir el tipo y el nombre de la actividad a modificar.
E3	Falta de Campos	Mensaje de alerta indicando que falta el llenado de campos.
	Datos Incongruentes	Mensaje de error en el llenado del formato.

Poscondiciones: Regresa a pantalla “Modificar Actividades”.

UC12) Alta actividad con plantilla

Caso de Uso: **Alta Actividad con Plantilla Establecida.**

Propósito: Mostrar el flujo al ingresar la actividad preguardada dentro del sistema SICED.

Actor: Administrador

Precondición:
- El usuario debe estar dentro del sistema.
- El usuario debe tener una clave de usuario, que le es proporcionada cuando es dado de alta en el sistema.
- Se debe tener registrada al menos una actividad.
- El usuario debe buscar la actividad e ingresar los datos del registro a ingresar.

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El actor selecciona la opción "Plantillas Establecidas"	2	Seleccionar el tipo y nombre de actividad que se desea ingresar.	E1
3	El sistema regresa un formulario editable con los datos comunes de la plantilla de actividad.	4	Se introducen los nuevos datos de la actividad y se guardan.	E2, E3
5	Informa que la información ha sido guardada exitosamente.			

Excepciones:

ID	Nombre	Acción
E1	No se eligió tipo y nombre de actividad.	Mensaje de alerta indicando que se debe elegir el tipo y el nombre de la actividad a modificar.
E2	Falta de Campos	Mensaje de alerta indicando que falta el llenado de campos.
E3	Datos Incongruentes	Mensaje de error en el llenado del formato.

Poscondiciones: Regresa a pantalla "Modificar Actividades".

UC13) Eliminar dependencia

Caso de Uso: **Eliminar Dependencia.**

Propósito: Mostrar el flujo al eliminar una dependencia dentro del sistema SICED.

Actor: Administrador

Precondición:
- El usuario debe estar dentro del sistema.
- El usuario debe tener una clave de usuario, que le es proporcionada cuando es dado de alta en el sistema.
- Se debe tener registrada al menos una dependencia.

- El usuario debe buscar la dependencia a eliminar.

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El actor selecciona la opción “Eliminar Dependencia”	2	Seleccionar la dependencia que se desea eliminar.	E1
3	Informa que la dependencia ha sido eliminada exitosamente.			

Excepciones:

ID	Nombre	Acción
E1	No existen dependencias registradas.	Mensaje de alerta indicando que no hay dependencias ingresadas.

Poscondiciones: Regresa a pantalla “Eliminar Dependencia”

UC14) Eliminar actividad

Caso de Uso: **Eliminar Actividad**

Propósito: Mostrar el flujo al eliminar una actividad dentro del sistema SICED

Actor: Administrador

Precondición:
- El usuario debe estar dentro del sistema.
- El usuario debe tener una clave de usuario, que le es proporcionada cuando es dado de alta en el sistema.
- Se debe tener registrada al menos una actividad.
- El usuario debe buscar la actividad a eliminar.

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El actor selecciona la opción “Eliminar Actividad”	2	Seleccionar el tipo y nombre de actividad que se desea eliminar.	E1, E2
3	Informa que la información ha sido eliminada exitosamente.			

Excepciones:

ID	Nombre	Acción
E1	No existen actividades registradas.	Mensaje de alerta indicando que no hay actividades ingresadas.
E2	No se eligió tipo y nombre de actividad.	Mensaje de alerta indicando que se debe elegir el tipo y el nombre de la actividad a eliminar.

Poscondiciones: Regresa a pantalla “Eliminar Actividad”.

UC15) Alta de profesor

Caso de Uso: **Alta de profesor**

Propósito: El propósito de éste caso de uso, es mostrar el flujo realizado al dar de alta a un profesor en la división

Actor: Responsable de división

Precondiciones: El sistema verificó que el usuario tiene los permisos necesarios para acceder a este caso de uso:

- Haber dado de alta el catálogo de nombramientos
- Haber dado de alta el catálogo de áreas

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El Actor selecciona la opción “alta de profesor”	2	Introduce los datos del profesor	E1, E2
3	Informa que la información ha sido insertada exitosamente			

Excepciones:

ID	Nombre	Acción
E1	No se llenaron todos los datos del profesor	Mensaje de alerta indicando que hacen falta datos del profesor
E2	El profesor ya se había registrado	Se envía el mensaje: Profesor ya existente

Poscondiciones: Regresa a pantalla “Alta de profesor”.

UC16) Modificar datos del profesor

Caso de Uso: **Modificar datos del profesor**

Propósito: El propósito de éste caso de uso, es mostrar el flujo realizado al modificar los datos personales y laborales de un profesor registrado en la división

Actor: Responsable de división

Precondiciones: El sistema verificó que el usuario tiene los permisos necesarios para acceder a este caso de uso

Haber dado de alta al menos un profesor en la división

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El Actor selecciona la opción “modificar profesor”	2	Seleccionar el profesor al que le desea modificar los datos	
3	Editar los datos actuales del profesor obtenidos del sistema			E1, E2
4	Informa que la información ha sido actualizada exitosamente			

Excepciones:

ID	Nombre	Acción
E1	No se llenaron todos los datos del profesor	Mensaje de alerta indicando que hacen falta datos del profesor
E2	No se tienen profesores registrados en la división	Aparece la leyenda: No se han dado de alta profesores para esta división

Poscondiciones: Regresa a pantalla “Modificar profesor”.

UC17) Inscribir profesor a actividad

Caso de Uso: **Inscribir profesor**

Propósito: El propósito de éste caso de uso, es mostrar el flujo para inscribir a un profesor a una actividad

Actor: Responsable de división

Precondiciones: El sistema verificó que el usuario tiene los permisos necesarios para acceder a este caso de uso

- Tener actividades disponibles para inscripción
- Haber dado de alta al menos un profesor en la división

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El Actor selecciona la opción "inscribir profesor"	2	Seleccionar el tipo de actividad al que desea inscribir al profesor	
3	Seleccionar el nombre de la actividad a la que desea inscribir al profesor	4	Seleccionar al profesor que desea inscribir a la actividad	E1
5	Informa que la información ha sido actualizada exitosamente			

Excepciones:

ID	Nombre	Acción
E1	No se tienen profesores registrados en la división	Aparece la leyenda: No se han dado de alta profesores para esta división

Poscondiciones: Regresa a pantalla "Inscribir profesor".

UC18) Calificar profesor

Caso de Uso: **Evaluar profesor**

Propósito: El propósito de éste caso de uso, es mostrar el flujo para calificar a un profesor que se inscribió en una actividad

Actor: Responsable de división

Precondiciones: El sistema verificó que el usuario tiene los permisos necesarios para acceder a este caso de uso

Tener actividades disponibles para inscripción
Haber dado de alta al menos un profesor en la división

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El Actor selecciona la opción “calificar profesor”	2	Selecciona el tipo de actividad que desea calificar	
3	Seleccionar el nombre de la actividad a calificar	4	Seleccionar al profesor que requiere calificar	E1
5	Registrar los criterios a evaluar	6	Informa que el profesor se ha calificado	

Variantes de acuerdo a la calificación obtenida:

- **Profesor acreditado**: Si la calificación es mayor a 8, aparecerá un campo para registrar el folio de la constancia que se le entregará al profesor
- **Profesor no acreditado**: Si la calificación registrada en el paso anterior es menor a 8, aparecerán tres criterios de las causas por la que el profesor reprobó, se podrán elegir todas o al menos una.

Excepciones:

ID	Nombre	Acción
E1	No se tienen profesores registrados en la división	Aparece la leyenda: No se han dado de alta profesores para esta división

Poscondiciones: Regresa a pantalla “Calificar profesor”.

UC19) Eliminar profesor

Caso de Uso: **Eliminar profesor**

Propósito: El propósito de éste caso de uso, es mostrar el flujo para eliminar un profesor de una división

Actor: Responsable de división

Precondiciones: El sistema verificó que el usuario tiene los permisos necesarios para acceder a este caso de uso

Haber dado de alta al menos un profesor en la división

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El Actor selecciona la opción “eliminar profesor”	2	Seleccionar de la lista el profesor que se desea eliminar	E1
3	Informa que el profesor se ha eliminado satisfactoriamente			

Excepciones:

ID	Nombre	Acción
E1	No se tienen profesores registrados en la división	Aparece la leyenda: No se han dado de alta profesores para esta división

Poscondiciones: Regresa a pantalla “Eliminar profesor”.

UC20) Revisar datos de un profesor

Caso de Uso: **Consultar datos del profesor**

Propósito: El propósito de éste caso de uso, es mostrar el flujo para consultar los datos de un profesor adscrito a una división

Actor: Jefe de división

Precondiciones: El sistema verificó que el usuario tiene los permisos necesarios para acceder a este caso de uso

Haber dado de alta al menos un profesor en la división

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El Actor selecciona el profesor del cual desea conocer la información	2	Indicar que se desea conocer la información personal del profesor seleccionado en el paso 1	E1
3	Despliega los datos del profesor seleccionado			

Excepciones:

ID	Nombre	Acción
E1	No se tienen profesores registrados en la división	Aparece la leyenda: No se han dado de alta profesores para esta división

Poscondiciones: Seleccionar a otro profesor
Consultar actividades inscritas, acreditadas y no acreditadas del profesor.

UC21) Consultar actividades a las que se ha inscrito un profesor

Caso de Uso: **Consultar actividades inscritas**

Propósito: El propósito de éste caso de uso, es mostrar el flujo para consultar las actividades a las que un profesor de cierta división se ha inscrito

Actor: Jefe de división

Precondiciones: El sistema verificó que el usuario tiene los permisos necesarios para acceder a este caso de uso

Haber dado de alta al menos un profesor en la división
Que el profesor seleccionado se encuentre inscrito a alguna actividad

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El Actor selecciona el profesor del cual desea conocer la información	2	Indicar que desea conocer las actividades a las que el profesor seleccionado en el paso 1 se ha inscrito	E1, E2
3	Despliega las diferentes actividades en las que el profesor se ha inscrito, con su respectivo detalle			

Excepciones:

ID	Nombre	Acción
E1	No se tienen profesores registrados en la división	Aparece la leyenda: No se han dado de alta profesores para esta división
E2	El profesor no se ha inscrito a ninguna actividad	Aparece la leyenda: El profesor no se ha inscrito a ninguna actividad

Poscondiciones: Seleccionar a otro profesor
Consultar datos personales, actividades acreditadas y no acreditadas del profesor.

UC22) Consultar actividades que ha acreditado un profesor

Caso de Uso: **Consultar actividades acreditadas**

Propósito: El propósito de éste caso de uso, es mostrar el flujo para consultar las actividades que un profesor de cierta división ha acreditado

Actor: Jefe de división

Precondiciones: El sistema verificó que el usuario tiene los permisos necesarios para acceder a este caso de uso

Haber dado de alta al menos un profesor en la división
Que el profesor seleccionado haya acreditado alguna actividad

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El Actor selecciona el profesor del cual desea conocer la información	2	Indicar que desea conocer las actividades acreditadas por el profesor que seleccionó en el paso 1	E1, E2
3	Despliega las diferentes actividades que acreditó el profesor con el detalle de las mismas			

Excepciones:

ID	Nombre	Acción
E1	No se tienen profesores registrados en la división	Aparece la leyenda: No se han dado de alta profesores para esta división
E2	El profesor no ha acreditado actividades	Aparece la leyenda: El profesor no ha acreditado ninguna actividad

Poscondiciones: Seleccionar a otro profesor
Consultar datos personales, actividades inscritas y no acreditadas del profesor.

UC23) Consultar actividades que no ha acreditado un profesor

Caso de Uso: **Consultar actividades no acreditadas**

Propósito: El propósito de éste caso de uso, es mostrar el flujo para consultar las actividades que un profesor de cierta división no ha acreditado

Actor: Jefe de división

Precondiciones: El sistema verificó que el usuario tiene los permisos necesarios para acceder a este caso de uso

Haber dado de alta al menos un profesor en la división
Que el profesor seleccionado no haya acreditado alguna actividad

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El Actor selecciona el profesor del cual desea conocer la información	2	Indicar que desea conocer las actividades no acreditadas por el profesor que seleccionó en el paso 1	E1, E2
3	Despliega las diferentes actividades que reprobó el profesor con el detalle de las mismas			

Excepciones:

ID	Nombre	Acción
E1	No se tienen profesores registrados en la división	Aparece la leyenda: No se han dado de alta profesores para esta división
E2	El profesor no ha reprobado actividades	Aparece la leyenda: El profesor no ha reprobado ninguna actividad

Poscondiciones: Seleccionar a otro profesor
Consultar datos personales, actividades inscritas y acreditadas del profesor.

UC24) Consultar datos de actividades disponibles para inscripción

Caso de Uso: **Consultar actividades disponibles**

Propósito: El propósito de éste caso de uso, es mostrar el flujo al buscar una actividad disponible para inscripción

Actor: Profesor

Precondiciones: Haber dado de alta al menos una actividad en el sistema

Flujo:

Paso	Actor	Paso	Sistema	Excepción
1	El Actor ingresa a la página del Centro de Docencia	2	Ingresa a la opción buscar de dicha página	
3	Despliega la página que contiene los diferentes criterios de búsqueda	4	El Actor selecciona el criterio de búsqueda	
5	Aparecen todas las actividades que cumplen el criterio de búsqueda	6	El Actor selecciona la actividad que le interesa para conocer sus detalles	E1
7	Despliega una página con los detalles de la actividad			

Excepciones:

ID	Nombre	Acción
E1	No encontrar actividades que cumpla el criterio de búsqueda seleccionado	Aparece la leyenda: No existen actividades con esas características

Poscondiciones: Regresar a la pantalla: "Buscar actividades"

3.3 Modelado del sistema

El modelado del sistema es una visión general de la construcción, en la cual se detallan ciertos aspectos como son:

- La construcción: Se muestran las clases implicadas en el sistema y los paquetes en que está divido el sistema
- El comportamiento: En dónde se muestra la funcionalidad que tendrá el sistema
- La interacción: Enfatiza sobre el flujo de control y de datos entre los elementos del sistema modelado.
- Es importante definir un modelo del sistema, ya que esto ayudará al programador a tener una idea más clara de los requerimientos y de las labores a realizar.

3.3.1 Diagramas UML

UML se usa para definir un sistema de software; para detallar los artefactos en el sistema; para documentar y construir, es el lenguaje en el que está descrito el modelo.

A continuación se mostrarán los diagramas UML para el sistema SICED, conforme a la clasificación mencionada anteriormente.

Construcción

Diagrama de clases: Incluye mucha información de las clases, tal como la relación entre un objeto y otro, la herencia de propiedades de otro objeto, el conjunto de atributos y métodos que son utilizados en la clase, etc.

Diagrama de paquetes: Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema.

En las siguientes páginas se muestran los diagramas descritos anteriormente, cabe señalar que cada clase y paquete se describirán con más detalle en secciones posteriores.

Diagrama de clases

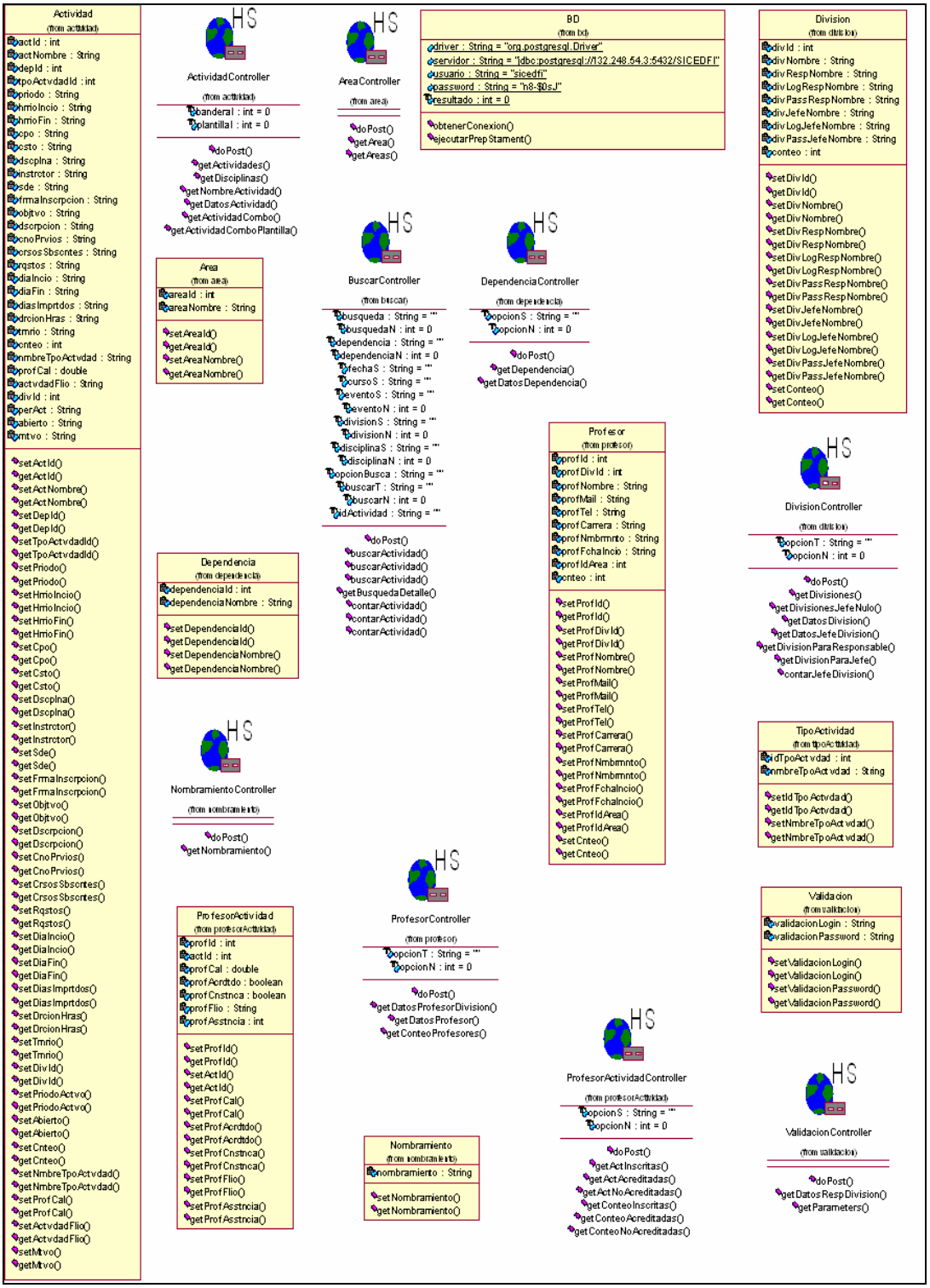


Fig. 3.1 Diagrama de Clases

Diagrama de paquetes

Las clases se encuentran organizadas en diversos paquetes de acuerdo a la información que están manejando; esto se muestra en el siguiente diagrama:

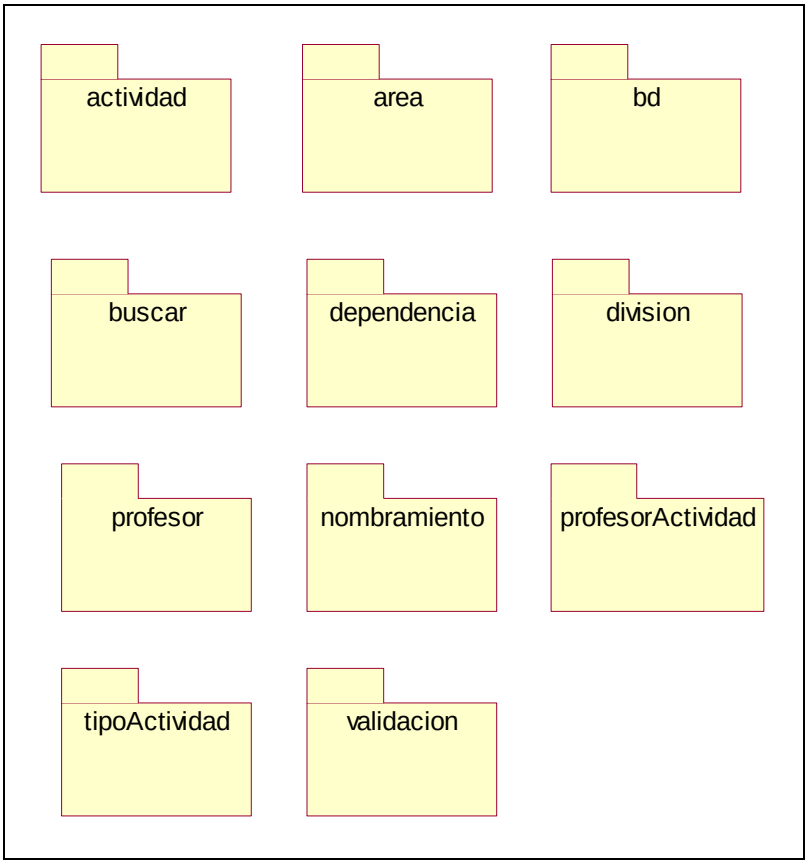


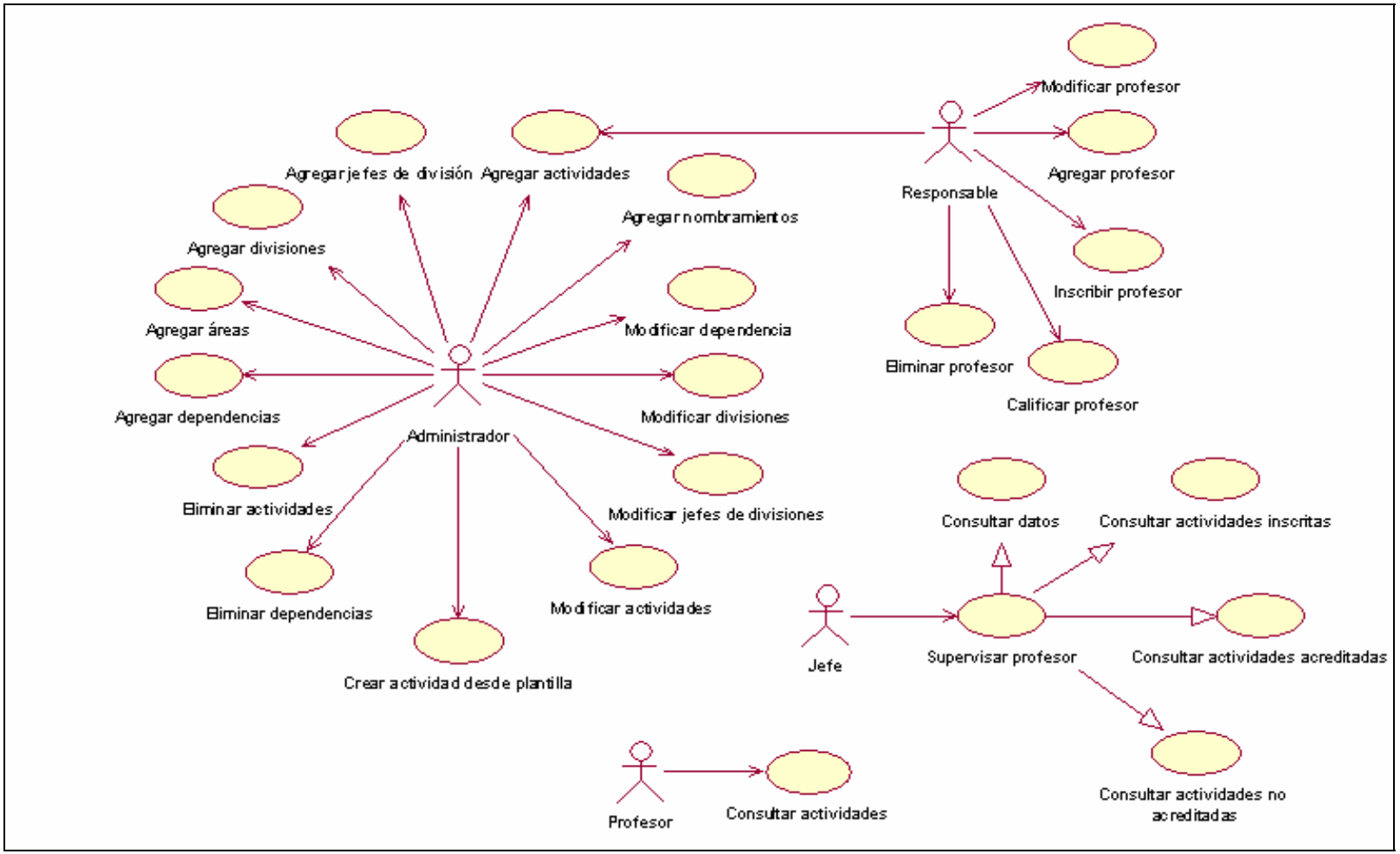
Fig. 3.2 Diagrama de Paquetes

Como se muestra en éste diagrama, el sistema se compone de once paquetes, los cuales contienen los archivos .class generados. Cada una de las clases contenidas en estos paquetes, maneja información relacionada con una tabla en la base de datos del SICED, y existe un paquete especial, bd, el que contiene la clase que maneja la información de conexión a la base de datos.

Comportamiento

Diagrama de casos de uso: Es una especie de diagrama de comportamiento, describe la funcionalidad del sistema de una manera general. A continuación se muestra el diagrama general de casos de uso. Es importante señalar que en la siguiente sección se mostrarán cada uno de los casos de uso involucrados en el diagrama. En éste diagrama están involucrados cuatro actores principales: Administrador, responsable de división, jefe de división y profesores; estos actores corresponden a los tres niveles de seguridad implementados en el sistema y un usuario no validado, respectivamente; por lo tanto, dependiendo de la validación del usuario, éste sólo podrá acceder a ciertos casos de uso.

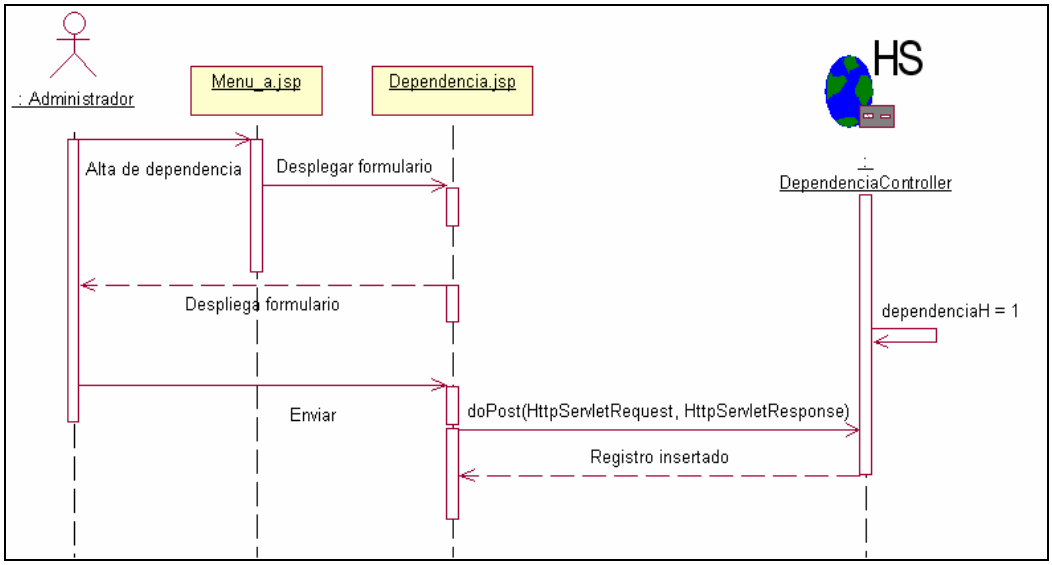
Diagrama de casos de uso.



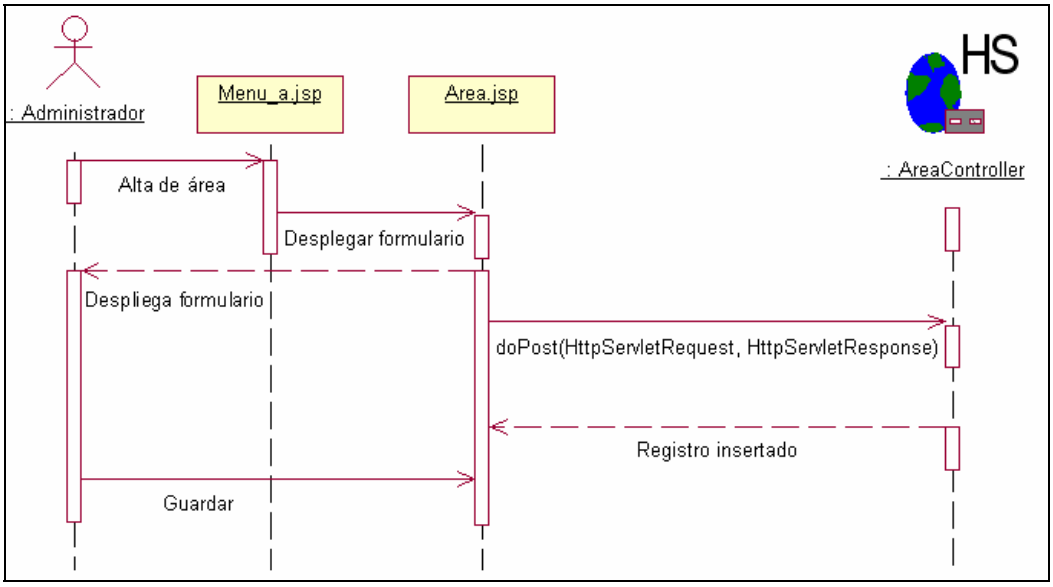
Interacción

Diagramas de secuencia: Muestra la interacción entre objetos. A continuación se mostrarán los diagramas de secuencia utilizados en éste sistema. Cabe señalar que sólo se están mostrando los flujos de trabajo normales, ya que en la descripción de casos de definieron los flujos alternos y de error.

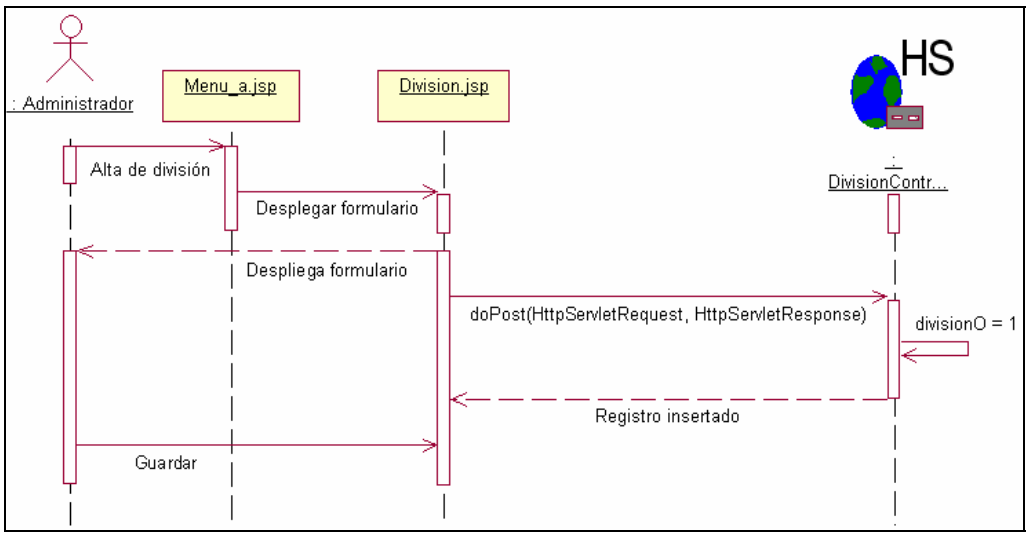
a) Alta de dependencia



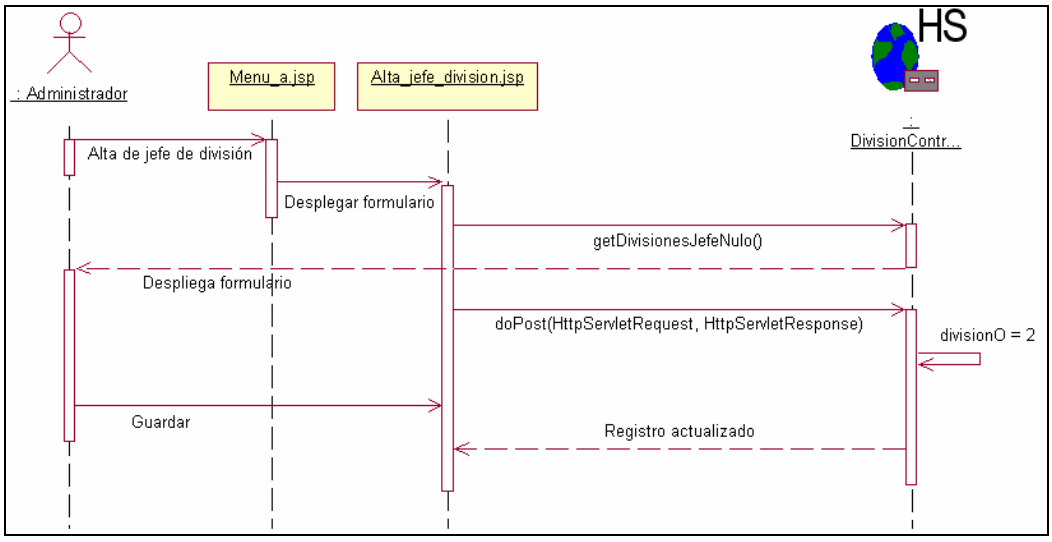
b) Alta de áreas



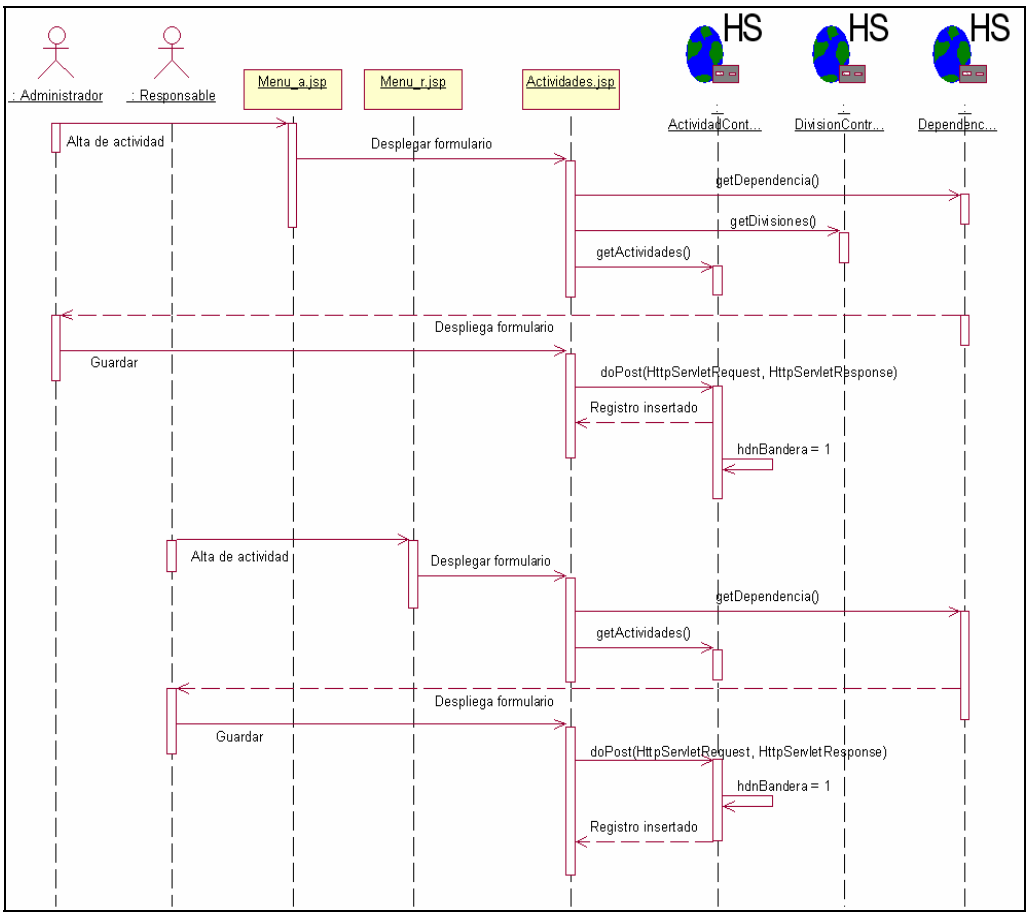
c) Alta de divisiones



d) Alta de jefe de división



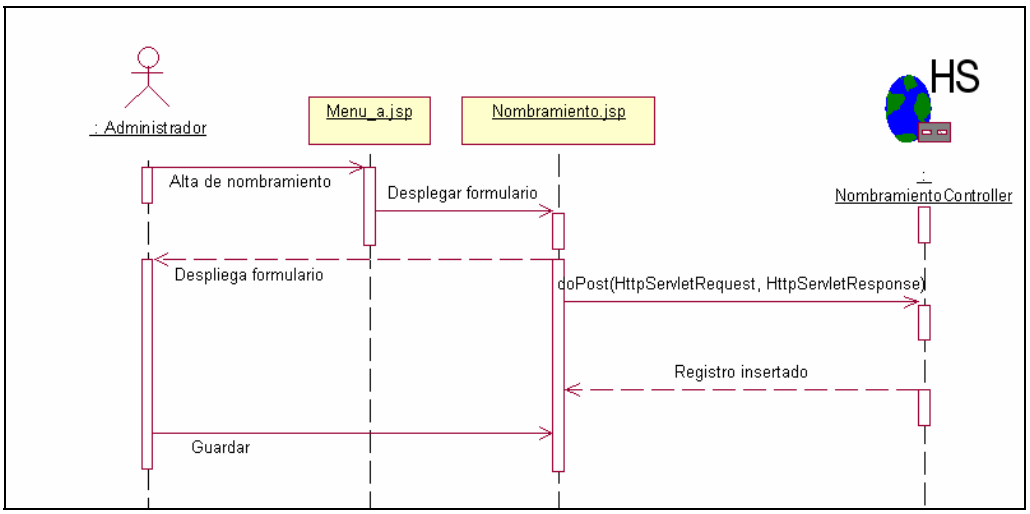
e) Alta de actividad



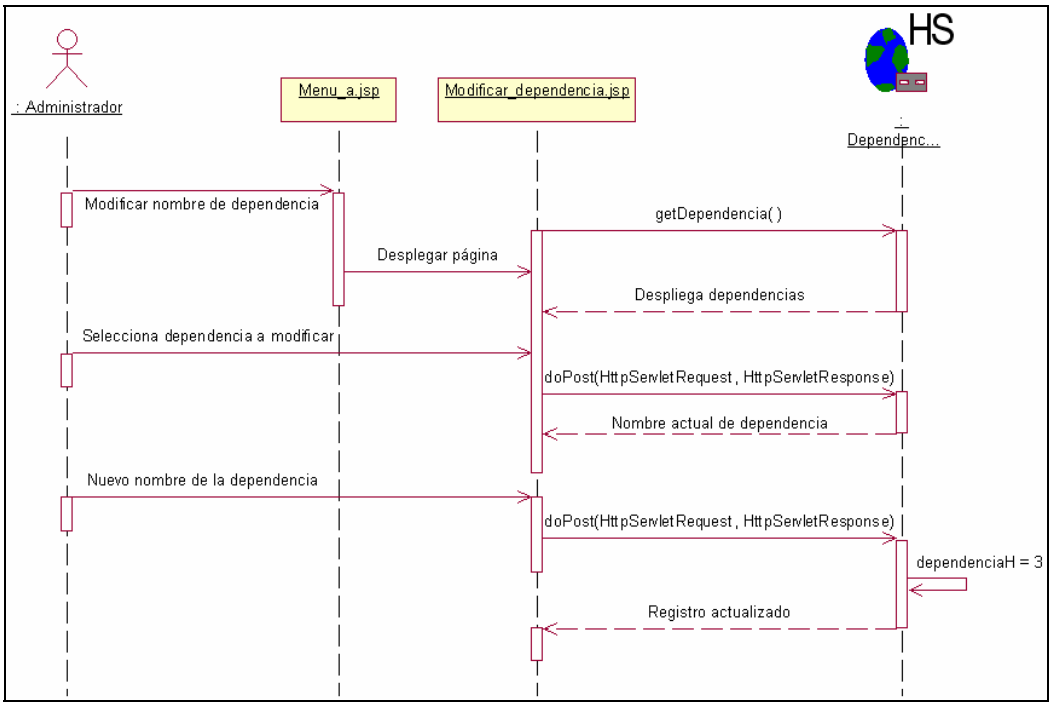
Se puede mostrar en éste diagrama de secuencia que en la operación **Alta de actividad**, tienen acceso dos actores: Administrador y responsable. Estos dos actores tienen casi la misma funcionalidad, sólo que el segundo no tiene la opción para modificar la división de la actividad que va a introducir, ya que ésta siempre estará por defecto con respecto a la división a la que pertenece el responsable.

Esta es la única operación que puede ser accedida por dos actores.

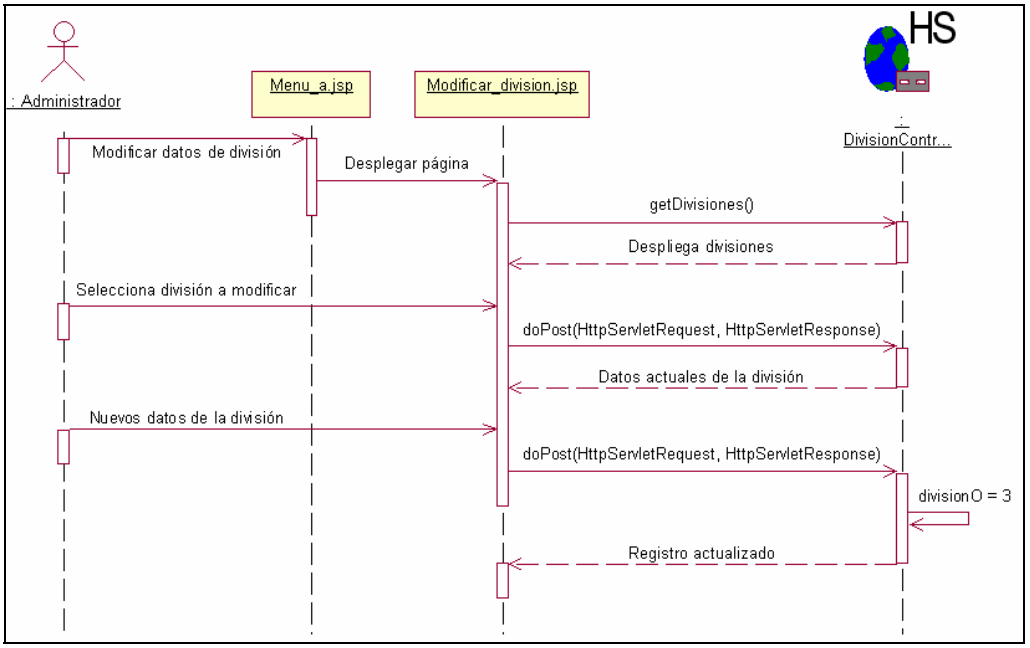
f) Alta de nombramiento



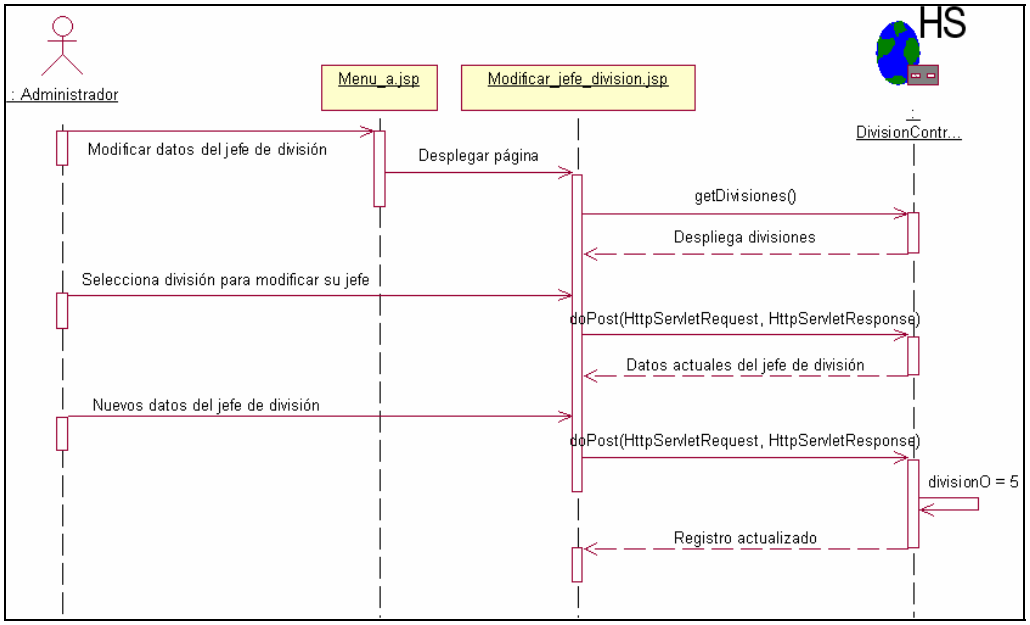
g) Modificar dependencia



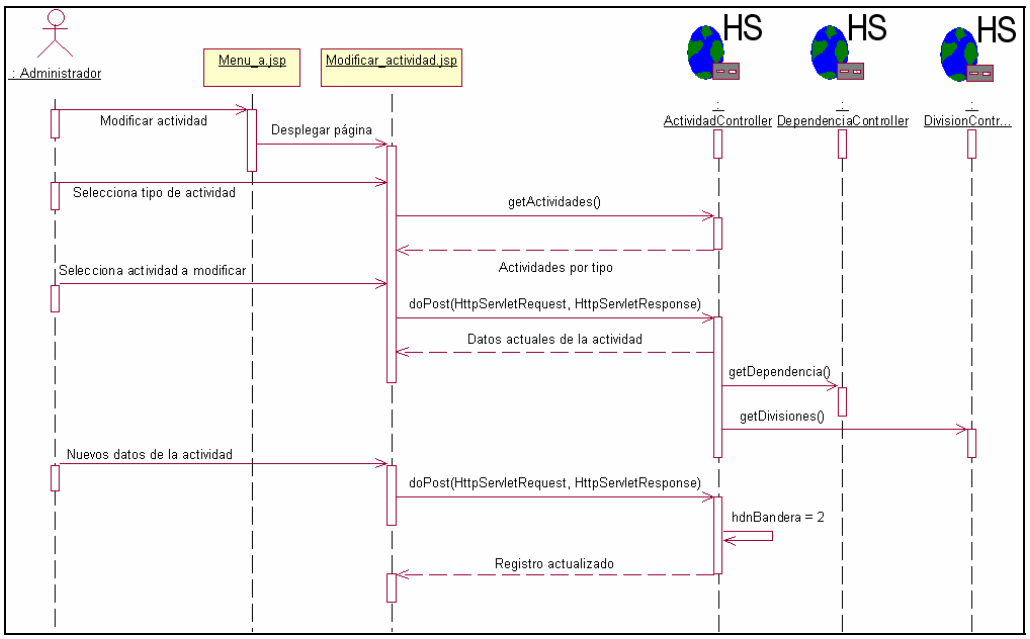
h) Modificar división



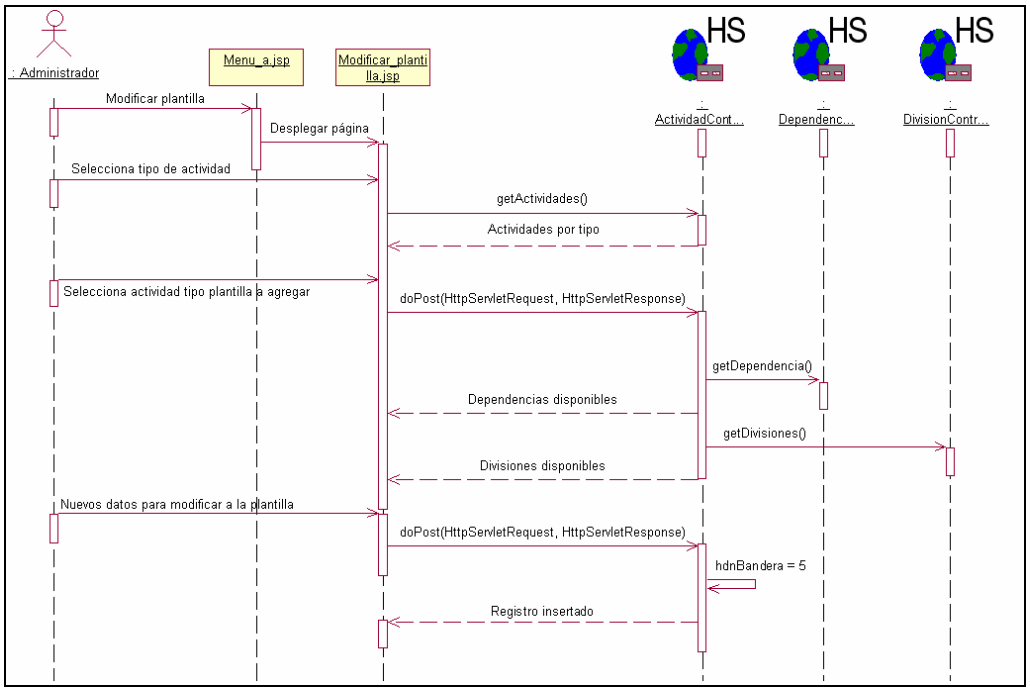
i) Modificar jefe de división



j) Modificar actividades

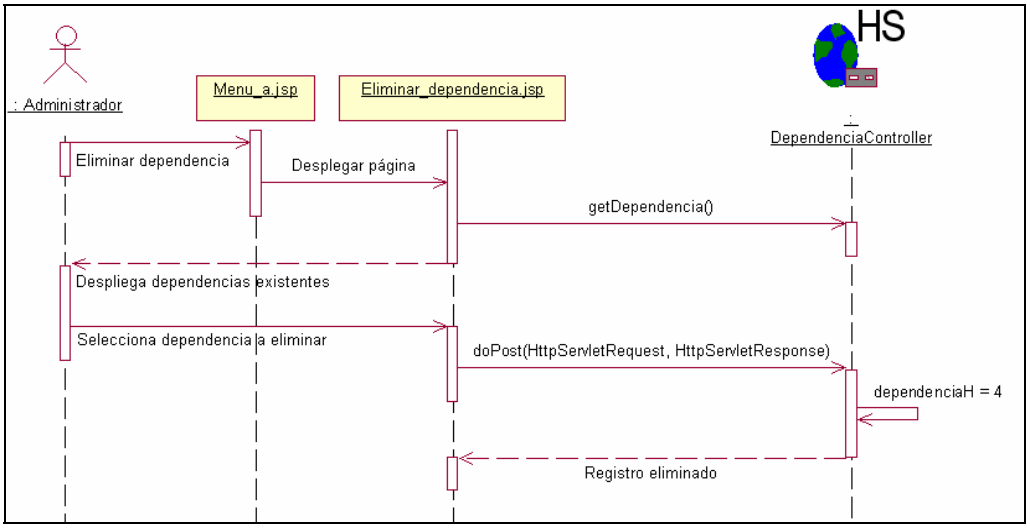


k) Modificar plantilla

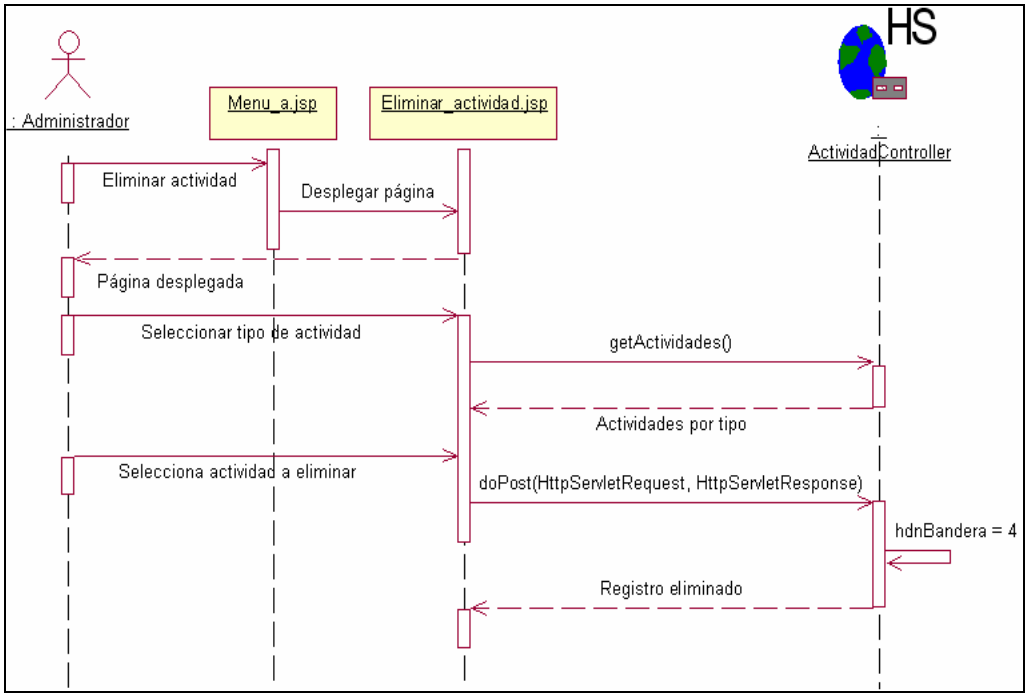


Esta opción permite insertar una actividad nueva pero con datos que pueden ser tomados de otra, por ejemplo; un curso de java en el que no varía el temario, objetivo, entre otros; pero puede variar el día de la semana en el que se imparte, las horas de duración, el horario, instructor, etc.

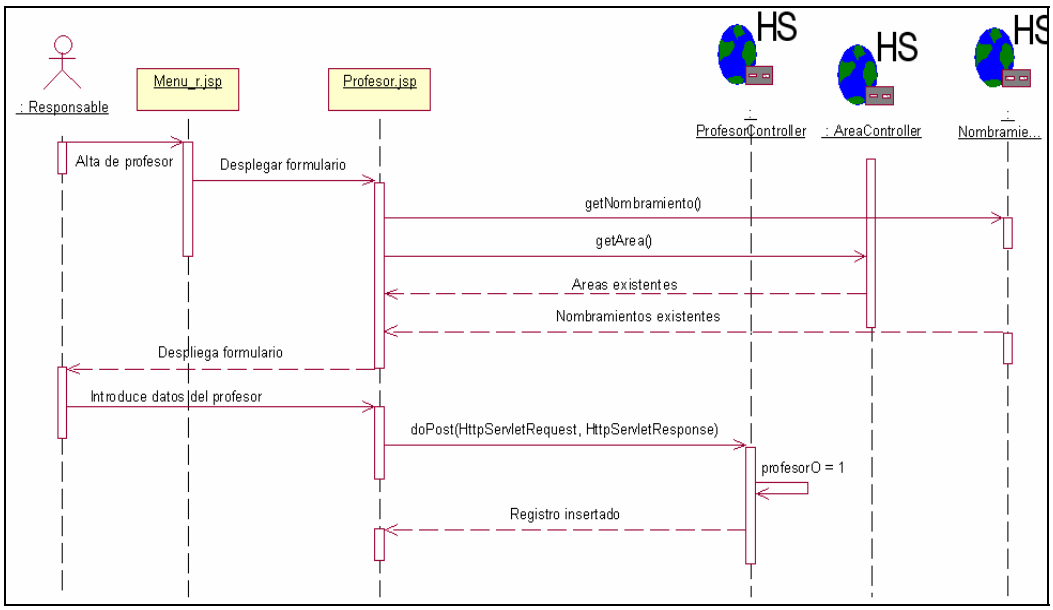
l) Eliminar dependencias



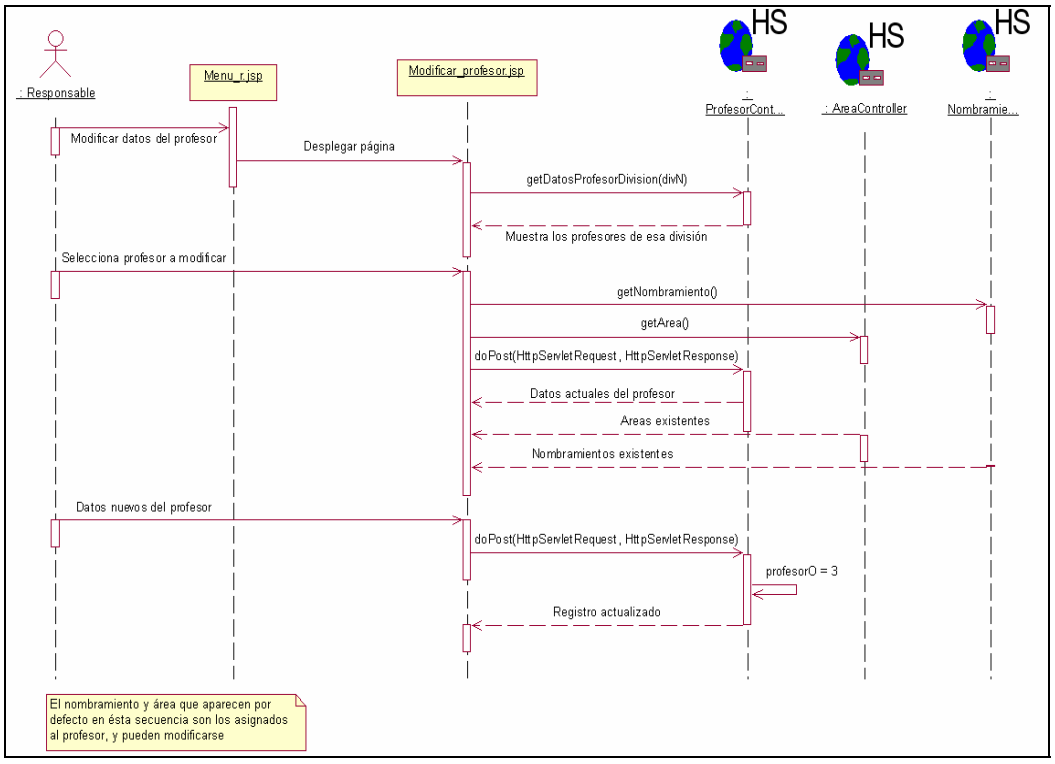
m) Eliminar actividad



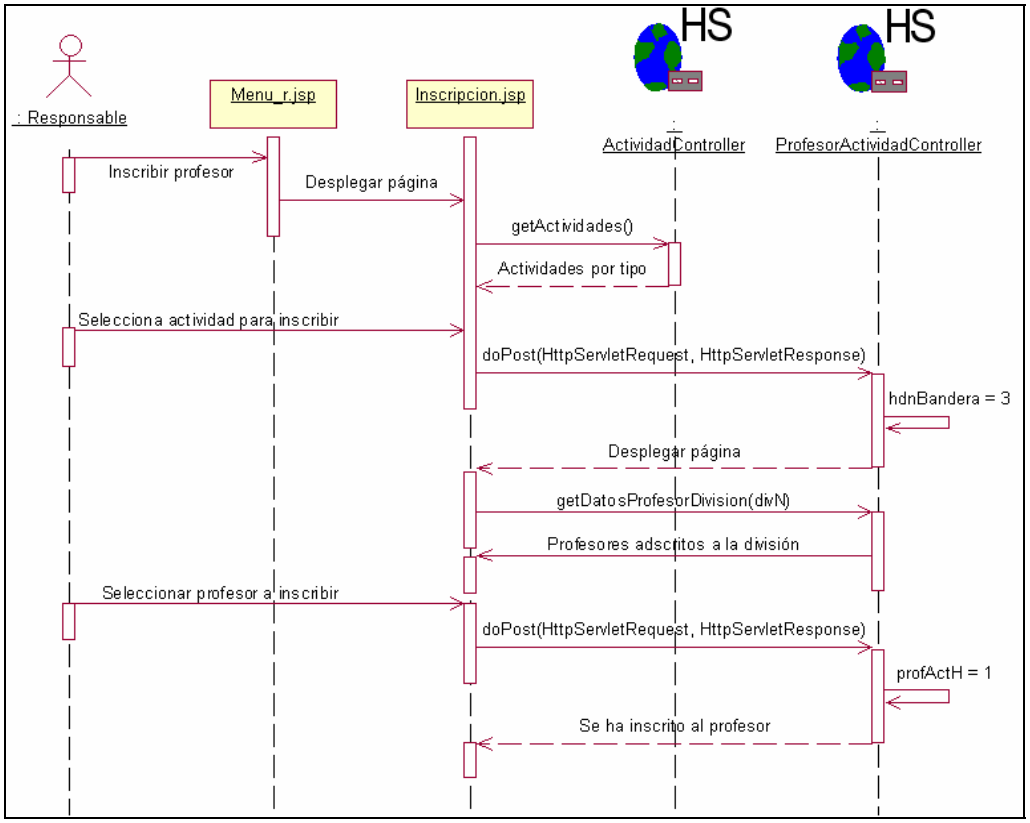
n) Alta de profesor



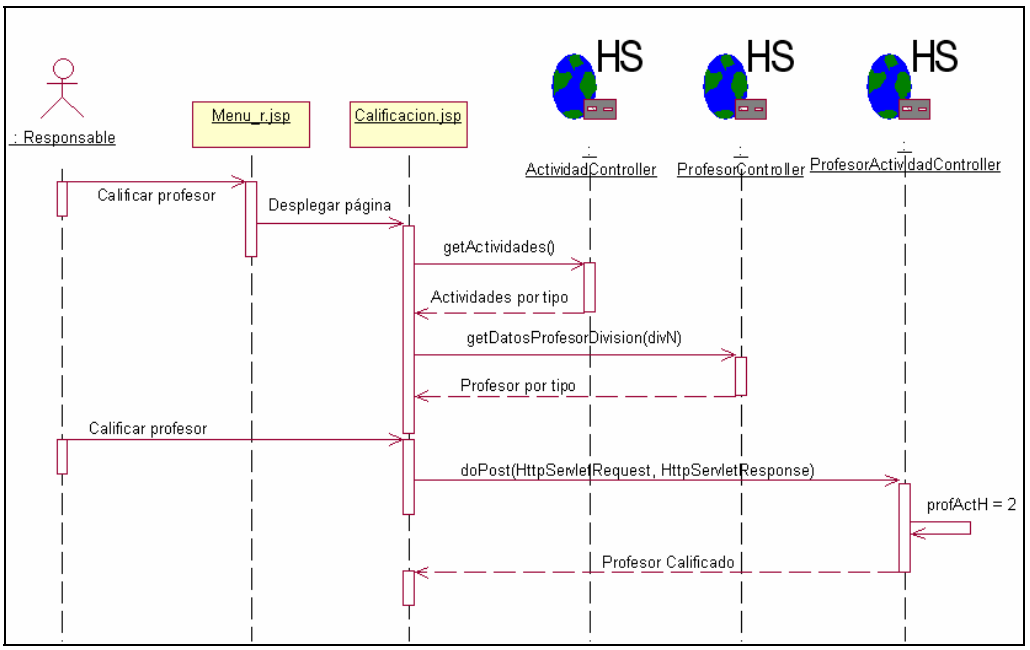
o) Modificar datos del profesor



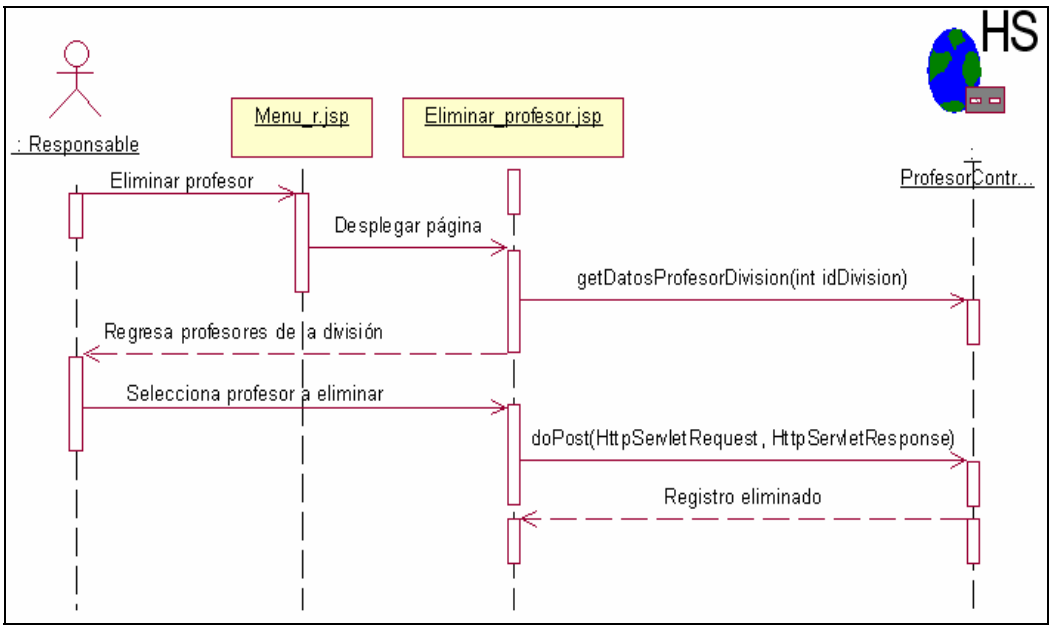
p) Inscribir profesor a actividad



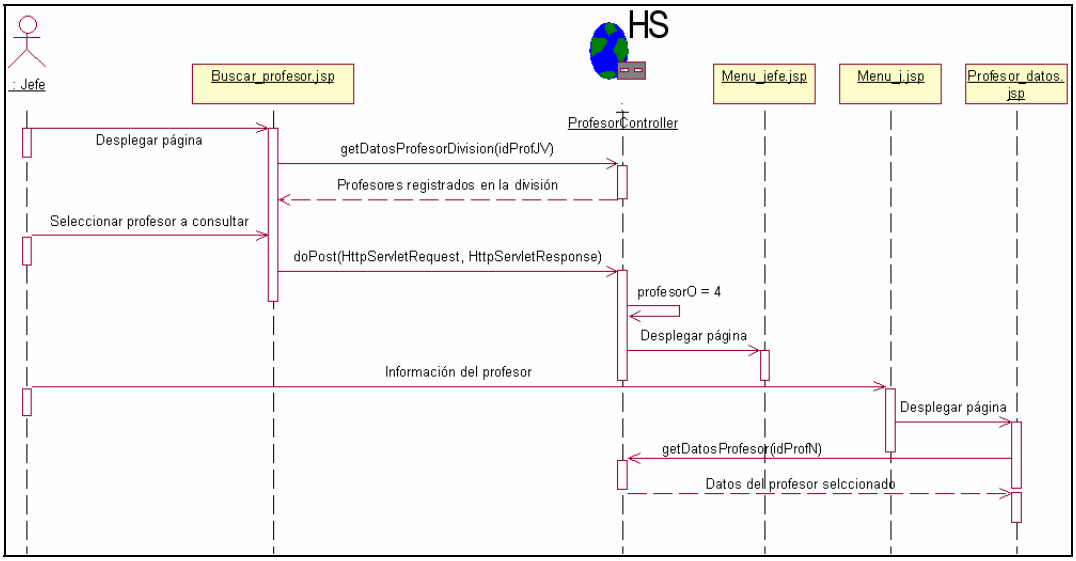
q) Calificar profesor



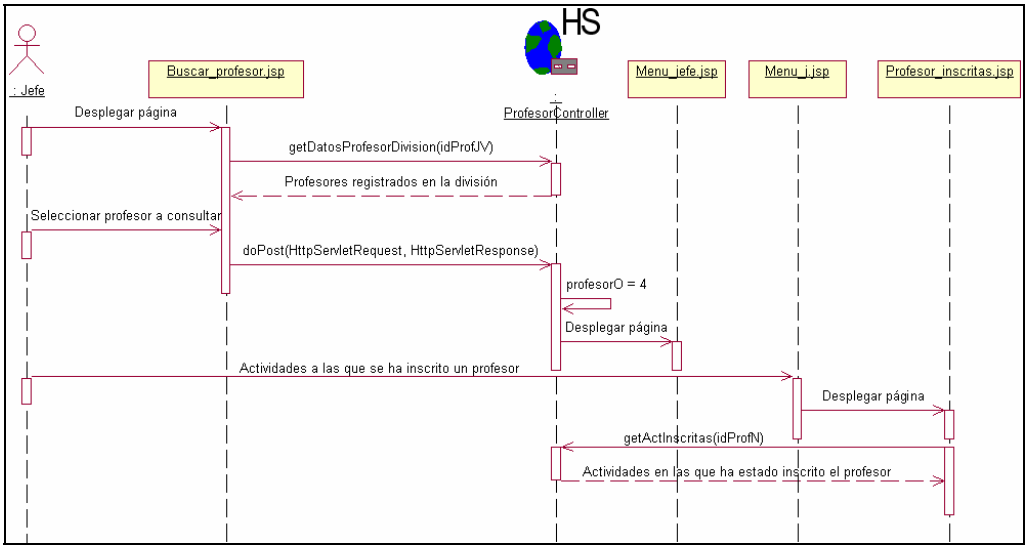
r) Eliminar profesor



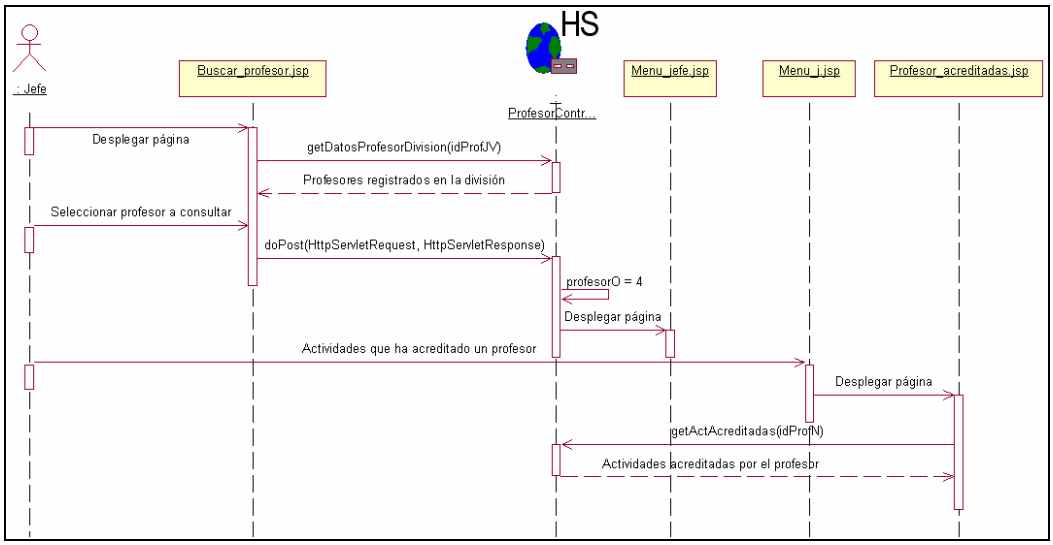
s) Revisar datos de un profesor



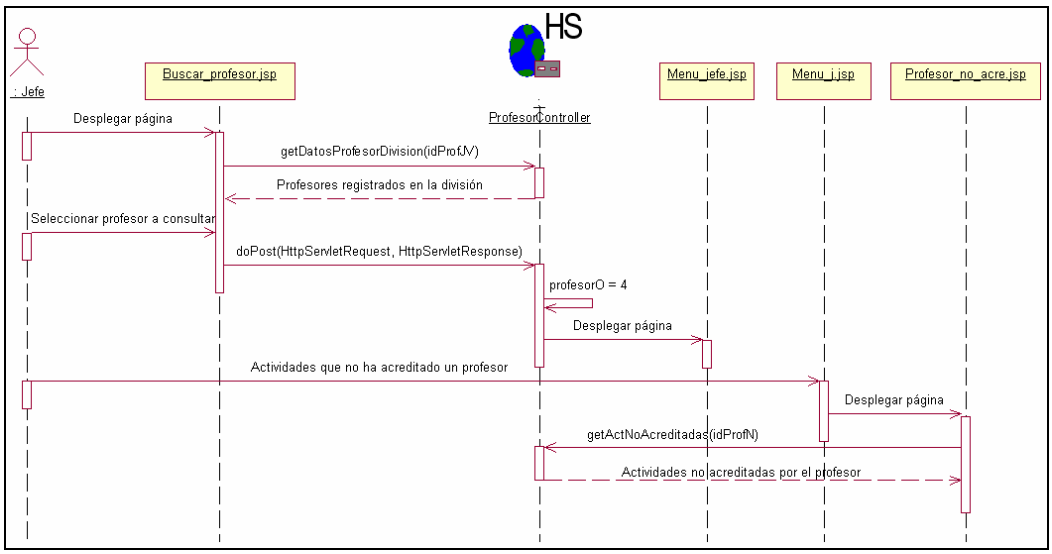
t) Revisar actividad en las que se inscribió un profesor



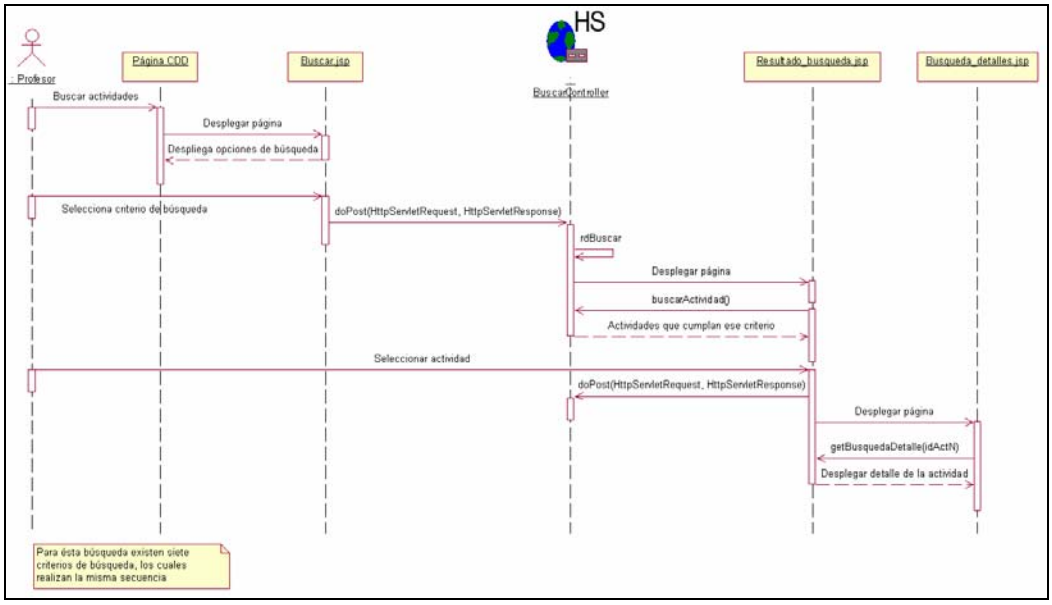
u) Consultar actividades que ha acreditado un profesor



v) Consultar actividades que no ha acreditado un profesor



w) Consultar datos de actividades disponibles para inscripción

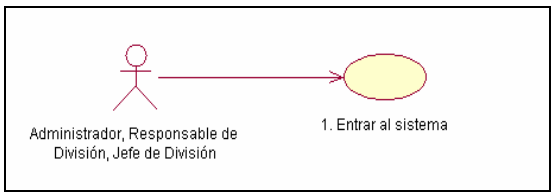


En ésta última secuencia, se puede observar que para cualquier criterio utilizado, se utiliza el mismo flujo, sólo varían los parámetros utilizados al realizar la consulta a la Base de Datos. Es importante recordar que para ésta secuencia el profesor no requiere autenticarse, ya que es de acceso público.

3.4 Implementación del modelo del sistema

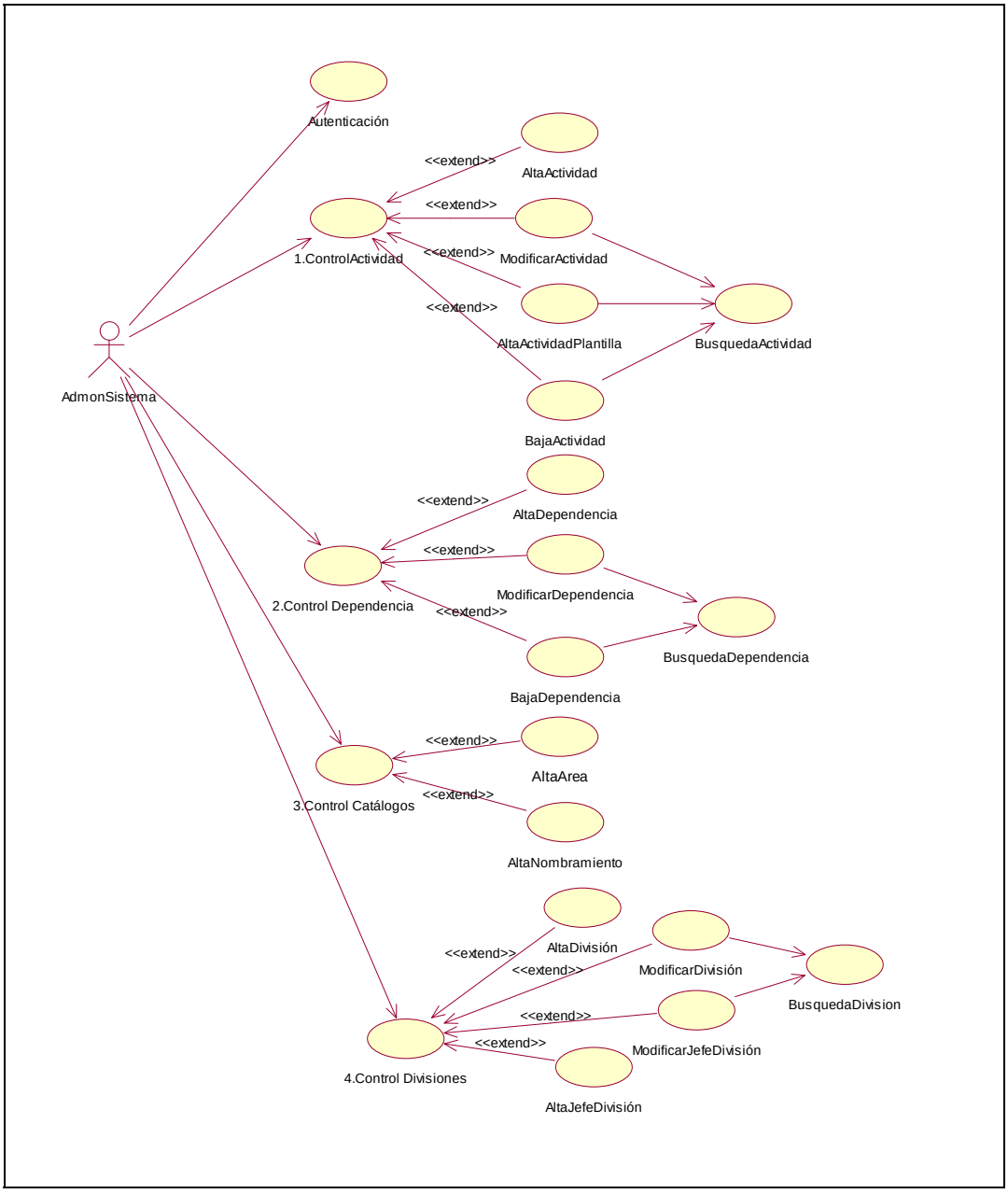
A continuación se presentan los correspondientes diagramas de los casos de uso vistos en la parte de análisis del sistema.

UC1) Autenticación

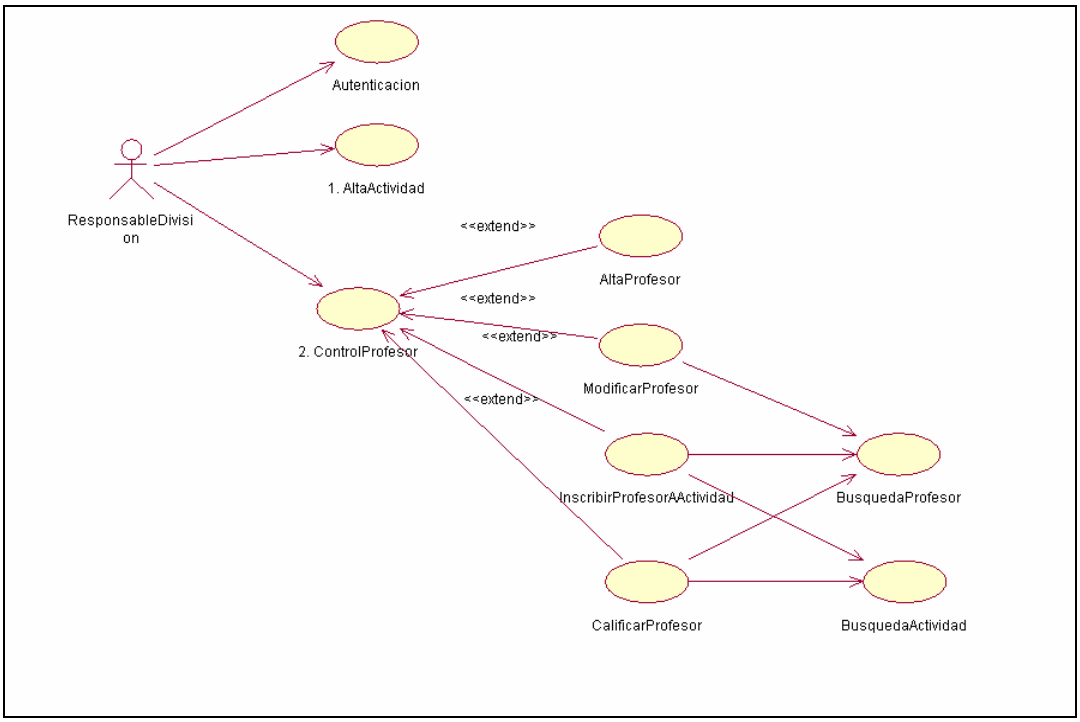


Este es el caso de uso más importante, ya que es donde está registrada la seguridad del sistema. Es importante notar, que para los casos de uso siguientes se utilizara se utilizará la misma pantalla de autenticación y el controlador indicado será capaz de diferenciar el rol del usuario que está ingresando al sistema.

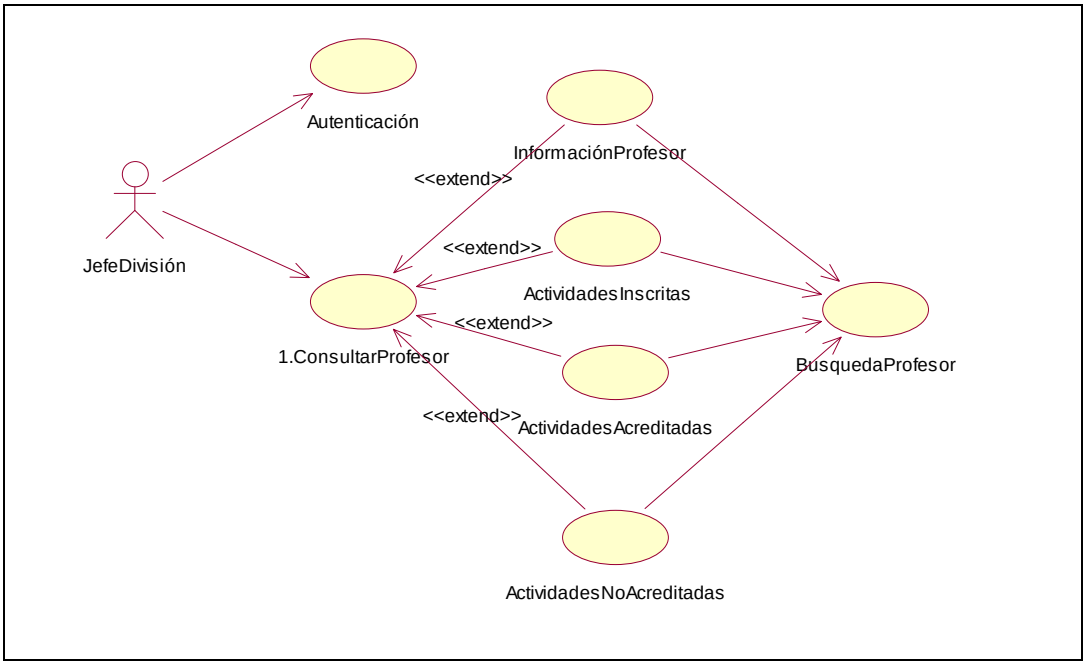
UCA) Administrador



UCB) Responsable de división

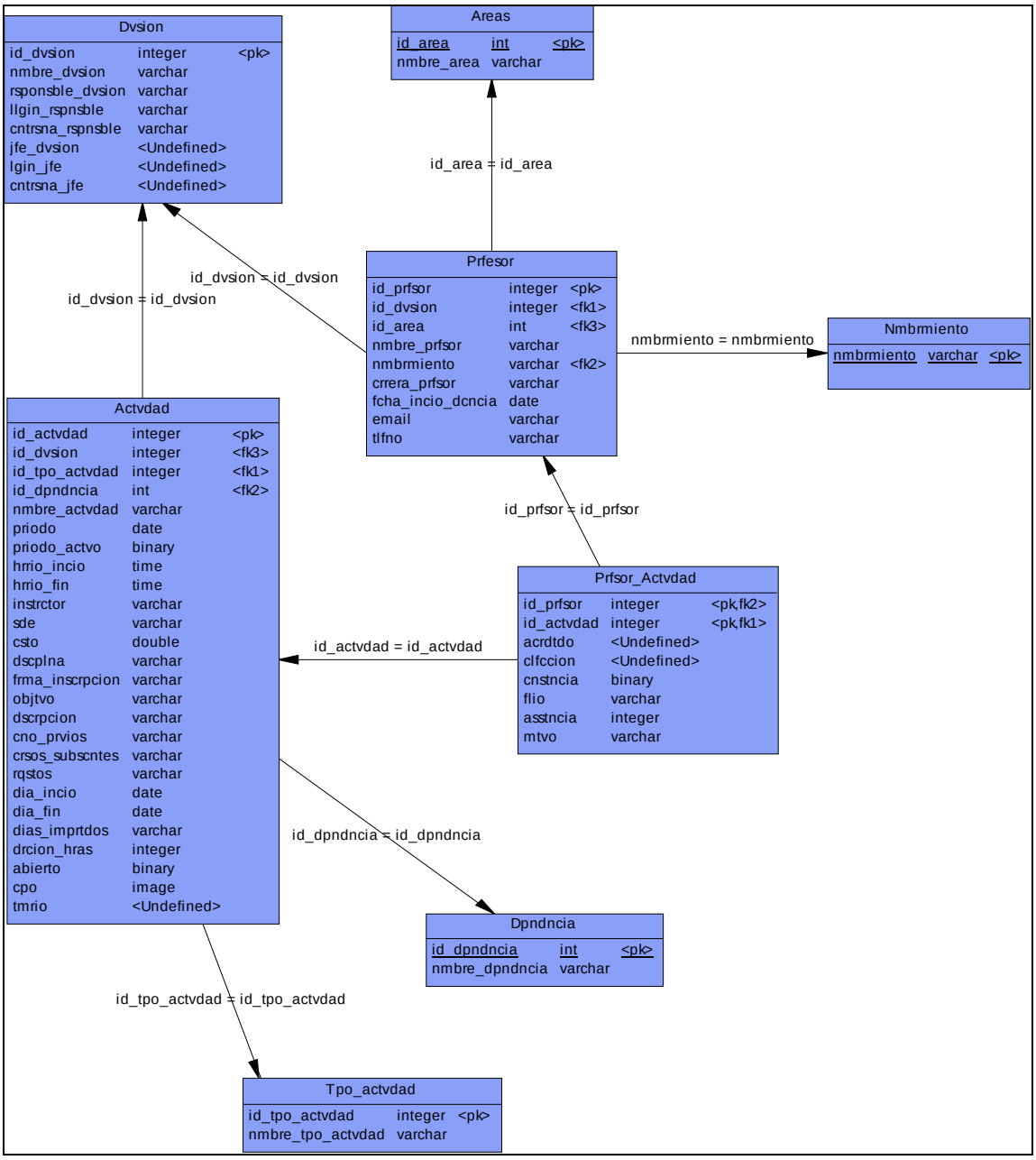


UCC) Jefe de división



3.5 Esquema de la base de datos

La Base de datos del sistema, se encuentra alojada en el servidor del centro de docencia; en éste servidor se realizó la instalación de postgresql. Se realizó la normalización de las tablas para un mejor desempeño. A continuación se presenta el diagrama entidad relación de la base de datos diseñada.



3.5.1 Diccionario de datos

El diccionario de datos, es una descripción de los datos almacenados en una Base de datos; los datos fundamentales que debe tener son: Nombre del dato, descripción, tipo y longitud. Esta información será de gran ayuda en la etapa de mantenimiento del sistema, ya que dará una visión más clara de para que se utiliza cada tabla y el significado de cada uno de sus atributos.

Actvdad				
Nombre	**Referencia**	**Tipo**	**Longitud**	**Descripción**
id_actvdad	PK	Integer		Identificador numérico de la actividad
id_dvsion	FK referencia de tabla **Dvsion**	Integer		Identificador numérico de la división que promueve la actividad
id_tpo_actvdad	FK referencia de tabla **Tpo_actvdad**	Integer		Identificador del tipo de actividad (curso, seminario, taller, etc.)
id_dpndncia	FK referencia de tabla **Dpndncia**	Integer		Identificador de la dependencia que organiza la actividad
nmbre_actvdad		Varchar	200	Nombre de la actividad
priodo		Varchar	10	Período en el que la actividad estuvo vigente
priodo_actvo		Boolean		Para conocer si se trata del período actual
hrrio_incio		Time		Hora de inicio de la actividad
hrrio_fin		Time		Hora de finalización de la actividad

instrctor		Varchar	255	Nombre del instructor
sde		Varchar	150	Lugar en el que será impartido el curso
csto		Double		Costo de la actividad en caso de que no sea aprobada
dscplna		Varchar	100	Disciplina a la que pertenece la actividad (cómputo, matemáticas, docencia, etc.)
frma_inscrpcion		Varchar	30	Forma en la que puede ser realizada la inscripción a la actividad (telefónica, presencial)
objtvo		Varchar	255	Objetivo de la actividad
dscrpcion		Varchar	180	Descripción de la actividad
cno_prvios		Varchar	80	Conocimiento previo sugerido para cursar la actividad
crsos_sbscntes		Varchar	255	Actividades dan continuidad al la actividad que se desea cursar
rqstos		Varchar	70	Requisitos para mínimos para cursar la actividad
Dia_incio		Date		Fecha de inicio del curso
Dia_fin		Date		Fecha de termino del curso

dias_imprtdos		Varchar	50	Días de la semana que se imparte la actividad
drcion_hras		Integer		Duración de la actividad en horas
abierto		Boolean		Estado de la actividad (abierto o cerrado por la cantidad de asistentes)
cpo		Integer		Cupo máximo de asistentes a la actividad
tmrio		Texto		Temario de la actividad

Areas				
Nombre	**Referencia**	**Tipo**	**Longitud**	**Descripción**
id_area	PK	Integer		Identificador numérico de el área
nmbre_area		Varchar	100	Nombre del área a la que puede pertenecer el asistente (posgrado, biblioteca, etc.)

Dpndncia				
Nombre	**Referencia**	**Tipo**	**Longitud**	**Descripción**
Id_dpndncia	PK	Integer		Identificador numérico de la dependencia
nmbre_dpndncia		Varchar	150	Nombre de la dependencia que promueve la actividad

Dvsion				
Nombre	**Referencia**	**Tipo**	**Longitud**	**Descripción**
Id_dvsion	PK	Integer		Identificador numérico de la división
nmbre_dvsion		Varchar	120	Nombre de la división
rspnsble_dvsion		Varchar	150	Nombre del responsable de la división
lgin_rspnsble		Varchar	15	Login del responsable de la división
cntrsna_rspnsble		Varchar	10	Contraseña del responsable de la división
jfe_dvsion		Varchar	150	Nombre del jefe de la división
lgin_jfe		Varchar	15	Login del jefe de división
cntrsna_jfe		Varchar	10	Contraseña del jefe de división

Nmbrmiento				
Nombre	**Referencia**	**Tipo**	**Longitud**	**Descripción**
nmbrmiento	PK	Varchar	200	Nombramiento del profesor

Prfsor				
Nombre	**Referencia**	**Tipo**	**Longitud**	**Descripción**
Id_prfsor	PK	Integer		Identificador numérico del profesor
Id_dvsion	FK referencia de tabla **Dvsion**	Integer		Identificador numérico de la división a la que pertenece el profesor
id_area	FK referencia de tabla **Areas**	Integer		Identificador numérico del área correspondiente al profesor

nmbre_prfsor		Varchar	150	Nombre del profesor
nmbrmnto_prfsor	FK referencia de tabla **Nmbrmnto**	Varchar	70	Nombramiento del profesor
crrera_prfsor		Varchar	100	Carrera profesional del profesor
fcha_incio_dcncia		Date		Fecha en la que inició la docencia en la Facultad
email		Varchar	100	Correo electrónico del profesor
tlfno		Varchar	20	Teléfono personal del profesor

Prfsor_actvdad				
Nombre	**Referencia**	**Tipo**	**Longitud**	**Descripción**
Id_prfsor	PK, FK referencia de tabla **Prfsor**	Integer		Identificador numérico del profesor
id_actvdad	PK, FK referencia de tabla **Actvdad**	Integer		Identificador numérico de la actividad a la que se inscribió el profesor
acrdtdo		Boolean		Estado de la acreditación del profesor en la actividad
clfccion		Double		Calificación del profesor en la actividad
cnstncia		Boolean		Estado de la entrega o no de constancia de asistencia a la actividad
Flio		Varchar	18	Folio de la constancia en caso de haberla obtenido

asstncia		Integer		Porcentaje de asistencia del profesor a la actividad
mtvo		Varchar	30	Lista de motivos de no acreditación

Tpo_actvdad				
Nombre	**Referencia**	**Tipo**	**Longitud**	**Descripción**
id_tpo_actvdad	PK	Integer		Identificador numérico del tipo de actividad
nmbre_tpo_actvdad		Varchar	25	Descripción del tipo de actividad

Capítulo 4

Metodología aplicada

Con apoyo de la herramienta Rational Rose se obtuvieron los diagramas desarrollados en el capítulo 3, lo cuál nos llevó a utilizar el Modelo Vista Controlador (MVC) como mejor alternativa. Este modelo es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se utiliza frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página.

En el sistema desarrollado para el Centro de Docencia, aplicamos el patrón mencionado. Se tienen especificadas las capas de la siguiente manera:

Modelo: JavaBeans que están ligados a los campos que se manejan en la Base de Datos. En el sistema presentado se tienen por ejemplo las clases siguientes:

- Dependencia
- Division
- Profesor

Cada una de éstas clases está vinculada con una tabla en la Base de datos y tiene asignados los atributos como propiedades de la misma, pudiendo acceder a ellos por medio de los métodos get() y set() correspondientes.

Vista: Archivos JSP que permiten la comunicación entre el usuario y el sistema. En estos archivos sólo se maneja el despliegue de datos y formularios para la introducción de datos al sistema.

Controlador: Servlets que tienen contenida la lógica de programación y comunicación a la Base de Datos. Estas clases son las que tienen el sufijo Controller, ejemplo de ellas son:

- DependenciaController
- DivisionController
- ProfesorController

A continuación se muestra un diagrama del flujo de la información del sistema.

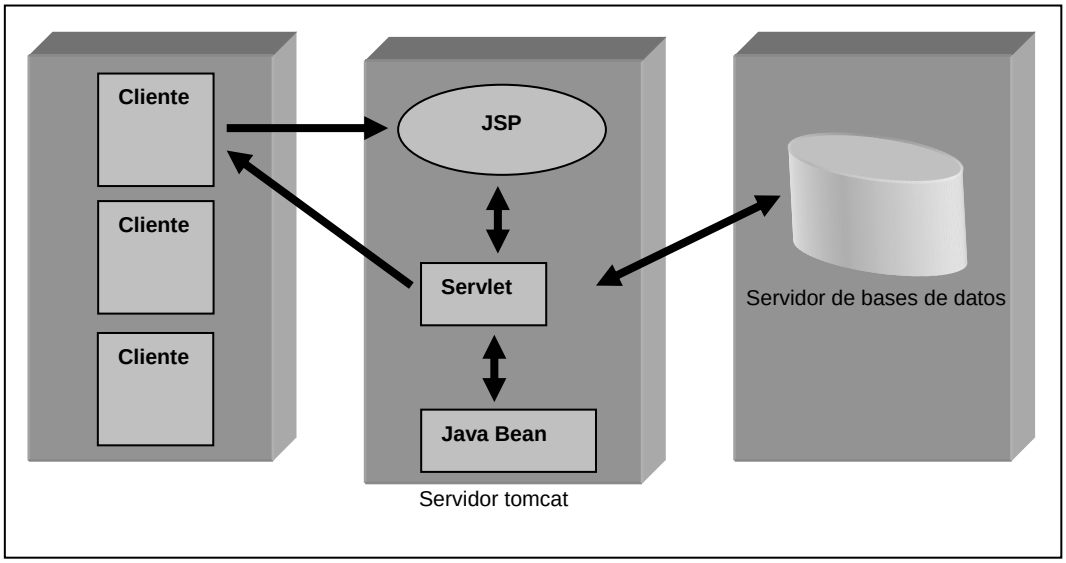


Fig. 4.1. Arquitectura aplicada al sistema

Al utilizar un servidor web, en éste caso Apache Tomcat, es importante tener un archivo descriptor de la aplicación; esto con la finalidad de que el servidor tenga en cuenta hacia donde se deben redirigir las peticiones, cuál será la página de entrada, entre otras cosas. Dicho archivo tiene por nombre **web.xml**, y se encuentra en el directorio WEB – INF de nuestra aplicación.

A continuación mostramos el archivo descriptor para nuestra aplicación:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <display-name>SICED</display-name>
    <description>
      Sistema del centro de docencia de la FI.
    </description>
    <welcome-file-list>
      <welcome-file>/Validaciones.jsp</welcome-file>
    </welcome-file-list>
    <error-page>
      <exception-type>org.postgresql.util.PSQLException</exception-type>
      <location>/Error.jsp</location>
    </error-page>
    <env-entry>
      <env-entry-name>cdd.cddUser</env-entry-name>
      <env-entry-value>siced</env-entry-value>
      <env-entry-type>java.lang.String</env-entry-type>
    </env-entry>
    <env-entry>
      <env-entry-name>cdd.cddPwd</env-entry-name>
      <env-entry-value>siced</env-entry-value>
      <env-entry-type>java.lang.String</env-entry-type>
    </env-entry>
    <servlet>
      <servlet-name>ValidacionController</servlet-name>
      <servlet-class>Tesis.validacion.ValidacionController</servlet-class>
    </servlet>
       <servlet>
      <servlet-name>ActividadController</servlet-name>
      <servlet-class>Tesis.actividad.ActividadController</servlet-class>
    </servlet>
    <servlet>
      <servlet-name>AreaController</servlet-name>
      <servlet-class>Tesis.area.AreaController</servlet-class>
    </servlet>
       <servlet>
      <servlet-name>BuscarController</servlet-name>
      <servlet-class>Tesis.buscar.BuscarController</servlet-class>
    </servlet>
       <servlet>
      <servlet-name>DependenciaController</servlet-name>
      <servlet-class>Tesis.dependencia.DependenciaController</servlet-
class>
    </servlet>

    <servlet>
      <servlet-name>DivisionController</servlet-name>
      <servlet-class>Tesis.division.DivisionController</servlet-class>
    </servlet>
    <servlet>
      <servlet-name>NombramientoController</servlet-name>
      <servlet-class>Tesis.nombramiento.NombramientoController</servlet-
class>
     </servlet>
```

```
    <servlet>
      <servlet-name>ProfesorController</servlet-name>
      <servlet-class>Tesis.profesor.ProfesorController</servlet-class>
    </servlet>
    <servlet>
      <servlet-name>ProfesorActividadController</servlet-name>
      <servlet-
class>Tesis.profesorActividad.ProfesorActividadController</servlet-class>
    </servlet>
    <servlet-mapping>
      <servlet-name>ValidacionController</servlet-name>
      <url-pattern>/Validaciones</url-pattern>
    </servlet-mapping>
       <servlet-mapping>
      <servlet-name>ActividadController</servlet-name>
      <url-pattern>/Actividades</url-pattern>
    </servlet-mapping>
       <servlet-mapping>
      <servlet-name>AreaController</servlet-name>
      <url-pattern>/Areas</url-pattern>
    </servlet-mapping>
       <servlet-mapping>
      <servlet-name>BuscarController</servlet-name>
      <url-pattern>/Buscar</url-pattern>
    </servlet-mapping>
       <servlet-mapping>
      <servlet-name>DependenciaController</servlet-name>
      <url-pattern>/Dependencias</url-pattern>
    </servlet-mapping>
       <servlet-mapping>
      <servlet-name>DivisionController</servlet-name>
      <url-pattern>/Divisiones</url-pattern>
    </servlet-mapping>
       <servlet-mapping>
      <servlet-name>NombramientoController</servlet-name>
      <url-pattern>/Nombramientos</url-pattern>
    </servlet-mapping>
       <servlet-mapping>
      <servlet-name>ProfesorController</servlet-name>
      <url-pattern>/Profesores</url-pattern>
    </servlet-mapping>
       <servlet-mapping>
      <servlet-name>ProfesorActividadController</servlet-name>
      <url-pattern>/ProfesoresActividades</url-pattern>
    </servlet-mapping>
</web-app>
```

Éste archivo, es de tipo xml, y proporciona información sobre configuración y despliegue de los componentes Web que componen una aplicación Web. Los ejemplos de los componentes Web son los parámetros de servlets, las definiciones de servlets y JSP (JavaServer Pages) y las correlaciones de URL (Uniform Resource Locators).

A continuación describimos algunas etiquetas del archivo implantado en nuestra aplicación:

- <display-name> : Nombre de la aplicación a desplegar
- <description> : Breve descripción de la aplicación
- <welcome-file-list> : Contiene una lista ordenada de los archivos de entrada
- <welcome-file> : Nombre del archivo utilizado como archivo de entrada por defecto, un ejemplo podría ser el archivo index.html

- <error-page> : Especifica el mapeo entre un código de error o tipo de excepción con la ruta o fuente en la aplicación web.
- <exception-type> : Nombre de la clase o tipo de excepción, por ejemplo: java.lang.string
- <location> : Localización de la página a desplegar cuando sucede el error, por ejemplo: /MiPaginaDeError.html
- <env-entry> : Declara una variable de entrada de ambiente, con su respectivo nombre <env-entry-name>, valor <env-entry-value> y tipo <env-entry-type>. Dichas variables pueden ser utilizadas en la aplicación, una vez que son encontradas con ciertos métodos.
- <servlet> : Contiene la declaración de datos de un servlet, tales como su nombre <servlet-name> y la clase con su ruta de empaquetado <servlet-class>.
- <servlet-mapping> : Describe el mapeo entre un servelt <servlet-name> y el URL con el que se invocado <url-pattern>.

El archivo puede contener más elementos, lo único que es importante es respetar el orden de éstos. Para conocer el orden en el que los elementos deben ir, existen archivos tipo dtd, en el caso de nuestro archivo, el archivo dtd se encuentra descrito en la línea del xml que se encuentra a continuación:

```
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
```

En ésta línea viene la dirección web donde se puede observar la estructura del archivo dtd.

Capítulo 5

Programación en Java del sistema

En éste capítulo se definirá la programación utilizada en el sistema, la cual está desarrollada en el lenguaje Java utilizando la plataforma J2EE. De ésta tecnología se tomaron dos objetos principalmente los JSP y los servlets; como se ha mencionado anteriormente, los JSP fungirán el papel de capa de presentación y los servlets manejarán toda la lógica de negocio involucrada en el sistema. En las secciones siguientes se describirán brevemente los métodos principales utilizados en los servlets, así como la interfaz gráfica generada en los JSP.

5.1 Métodos utilizados

A continuación se definirán los servlets que existen en el sistema, así como una descripción breve de los métodos que contienen.

ActividadController.java

Firma del método	Función
public ArrayList getActividades()	Regresa los tipos de actividades registrados en el sistema, actualmente sólo existen: curso, taller, conferencia y seminario.
public ArrayList getDisciplinas()	Regresa las disciplinas registradas en el sistema. Algunos ejemplos de disciplina pueden ser: Cómputo, Desarrollo, etc.
public ArrayList getNombreActividad()	Regresa todas las actividades registradas en el sistema.
public ArrayList getDatosActividad(int actividad_id)	Regresa los datos de la actividad con el identificador recibido como parámetro.
public ArrayList getActividadCombo(int id_tipo_actividad)	Regresa todas las actividades del tipo de actividad recibido como parámetro.
public ArrayList getActividadComboPlantilla(int id_tipo_actividad)	Obtiene una lista de todas las actividades registradas, pero sin mostrar las que tienen el mismo nombre pero diferentes datos, esto con la finalidad de generar la lista de platillas disponibles.

AreaController.java

Firma del método	Función
Public ArrayList getArea()	Devuelve todas las áreas existentes en el sistema.
Public ArrayList getAreas(int idArea)	Regresa los datos del área con el identificador recibido como parámetro

BuscarController.java

Firma del método	Función
Public ArrayList buscarActividad(int opcion, int id)	Regresa las actividades que cumplen con cierto criterio de búsqueda, dependiendo del parámetro opción que se indique. El parámetro opción tiene 3 valores posibles: Valor 1: Si se desean buscar todas las actividades que pertenezcan a la dependencia referenciada por el parámetro id. Valor 2: Si se desean buscar todas las actividades que sean del tipo referenciado por el parámetro id. Valor 3: Si se desean buscar todas las actividades registradas por la división referenciada por el parámetro id.
Public ArrayList buscarActividad(int opcion, String criterio)	Regresa las actividades que cumplen con cierto criterio de búsqueda, dependiendo del parámetro opción que se indique. El parámetro opción tiene 3 valores posibles: Valor 1: Si se desean buscar todas las actividades cuya fecha de inicio sea mayor a la fecha actual y sea igual a la fecha que se está recibiendo en el parámetro criterio Valor 2: Si se desean buscar todas las actividades cuya fecha de inicio sea mayor a la fecha actual y la disciplina sea igual a la que se está recibiendo en el parámetro criterio Valor 3: Si se desean buscar todas las actividades cuya fecha de inicio sea mayor a la fecha actual y el nombre de la actividad contenga al parámetro criterio; es decir, si se introduce criterio = 'Fundamentos', la búsqueda regresaría: 'Fundamentos de Java', 'Fundamentos de electrónica', etc.
public ArrayList buscarActividad()	Regresa todas las actividades cuya fecha de inicio sea mayor a la fecha actual
public ArrayList getBusquedaDetalle(int idAct)	Regresa los datos de la actividad que corresponda al identificador obtenido con el parámetro idAct.
public int contarActividad(int opcion,	Regresa el conteo de las actividades registradas de

int id)	acuerdo a los mismos criterios que el método: *public ArrayList buscarActividad(int opcion, int id)* de ésta clase.
public int contarActividad(int opcion, String fecha)	Regresa el conteo de las actividades registradas de acuerdo a los mismos criterios que el método: *public ArrayList buscarActividad(int opcion, String criterio)* de ésta clase.
public int contarActividad()	Regresa el conteo total de las actividades registradas en el sistema

DependenciaController.java

Firma del método	Función
public ArrayList getDependencia()	Regresa todas las dependencias registradas en el sistema
public ArrayList getDatosDependencia(int dependencia_id)	Regresa los datos de la dependencia referenciada al identificador obtenido como parámetro en: dependencia_id

DivisionController.java

Firma del método	Función
public ArrayList getDivisiones()	Obtiene todas las divisiones registradas en el sistema.
public ArrayList getDivisionesJefeNulo()	Obtiene todas las divisiones que no tienen asignado un jefe de división.
public ArrayList getDatosDivision(int div_id)	Regresa los datos de la división cuyo identificador sea igual al parámetro obtenido: div_id.
public ArrayList getDatosJefeDivision(int divId)	Regresa los datos del jefe de división de la división seleccionada.
public ArrayList getDivisionParaResponsable(String loginResp, String passResp)	Regresa el identificador de la división correspondiente al usuario y contraseña del responsable de división.
public ArrayList getDivisionParaJefe(String loginJefe, String passJefe)	Regresa el identificador de la división correspondiente al usuario y contraseña del jefe de división.
public int contarJefeDivision(int opcion, int idDiv)	Regresa el conteo de los jefes de división dependiendo de la opción que se pase como parámetro: Valor 1: Cuenta todos los jefes de una división. Valor 2: Cuenta cuantos jefes de división no han sido asignados a las divisiones.

NombramientoController.java

Firma del método	Función
public ArrayList getNombramiento()	Muestra todos los nombramientos existentes en el sistema.

ProfesorActividadController.java

Firma del método	Función
public ArrayList getActInscritas(int idProf)	Regresa todas las actividades en las que ha estado inscrito un profesor
public ArrayList getActAcreditadas(int idProfA)	Regresa las actividades que ha acreditado un profesor
public ArrayList getActNoAcreditadas(int idProfNA)	Regresa las actividades que un profesor no ha aprobado
public ArrayList getConteoInscritas(int idProfIns)	Regresa el conteo de las actividades en las que se ha inscrito el profesor
public ArrayList getConteoAcreditadas(int idProfAC)	Regresa el conteo de las actividades acreditadas por el profesor
public ArrayList getConteoNoAcreditadas(int idProfNoAC)	Regresa el conteo de las actividades que el profesor no acreditó

ProfesorController.java

Firma del método	Función
public ArrayList getDatosProfesorDivision(int idDivision)	Obtiene todos los profesores que pertenecen a la división con el identificador correspondiente al parámetro idDivision.
public ArrayList getDatosProfesor(int profId)	Obtiene los datos del profesor cuyo identificador es igual al parámetro profId.
public ArrayList getConteoProfesores(int idDiv)	Regresa el conteo de todos los profesores registrados en la división seleccionada.

ValidacionController.java

Firma del método	Función
public static String[] getParameters()	Método para obtener el usuario y contraseña del administrador del sistema. Estos datos se encuentran registrados en el archivo web.xml

5.2 Interfaz gráfica

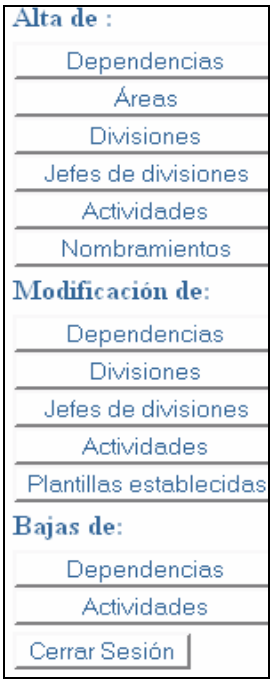
A continuación se muestran las diversas pantallas que componen la interfaz gráfica.

Pantalla de inicio

Esta pantalla es la que se despliega al consultar la aplicación, donde se necesita contar con un usuario y una contraseña para poder acceder al menú correspondiente al rol del usuario.

Menú de administrador

Este menú se despliega después que el usuario se autenticó y tiene rol de administrador.



Alta de dependencia

Analizando cada parte del menú de administrador, en la parte de Altas se tiene la siguiente pantalla de alta dependencia. Se cuenta con un solo campo de texto.

Alta División

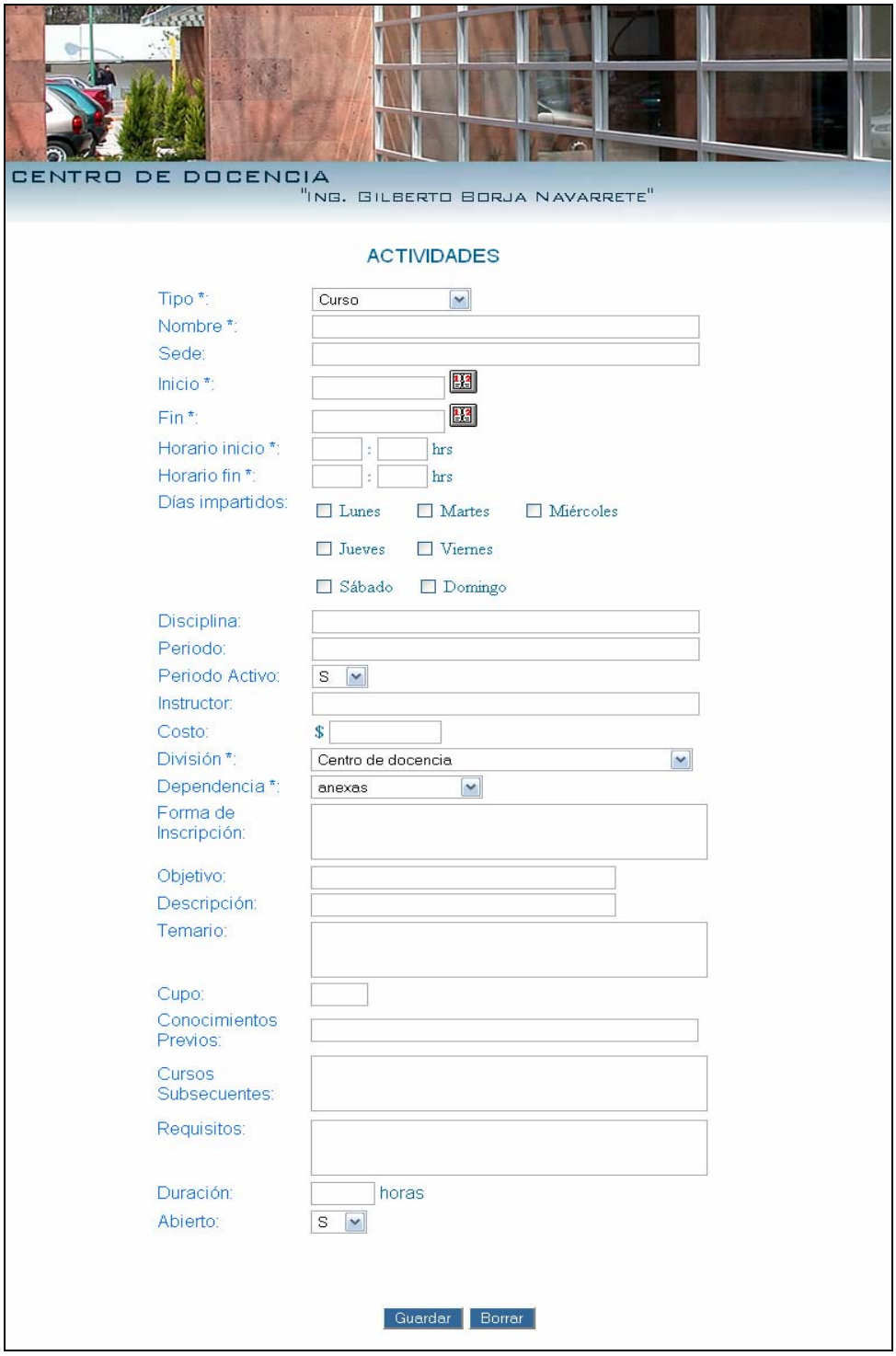
En esta pantalla se tienen los datos de la división, nombre de la división, nombre del responsable de dicha división, login y contraseña, es decir que al dar de alta una nueva división se debe de contar con los datos del responsable a cargo.

Alta de jefe de División

Una vez registrada la división y el responsable se prosigue con el alta del jefe de división que es la persona que tendrá acceso a la información del profesor así como de un historial de las actividades.

Alta de actividad

La pantalla de Alta de Actividad cuenta con diversos campos donde se detalla la descripción general de la actividad que se llevará a cabo. Maneja las fechas, requisitos, cupo y sede entre otras cosas.



Alta de nombramientos

Esta pantalla solo cuenta con un campo de texto donde se da de alta el catálogo de nombramientos de los profesores.

Modificar dependencia

Esta pantalla cuenta con dos partes, la primera de ellas solo muestra un combo donde se muestran todas las dependencias registradas en la aplicación, una vez localizada la misma se procede a realizar la búsqueda con la que se muestra la segunda parte de la pantalla con los datos de la dependencia en un campo de texto para poder ser modificado.

Modificar División

Esta pantalla cuenta con un combo de búsqueda de división para posteriormente modificar los datos, para ello se requiere contar con la contraseña del responsable de división para realizar cualquier cambio o en dado caso también se utiliza esta pantalla para modificar la contraseña del responsable.

Modificar jefe de División

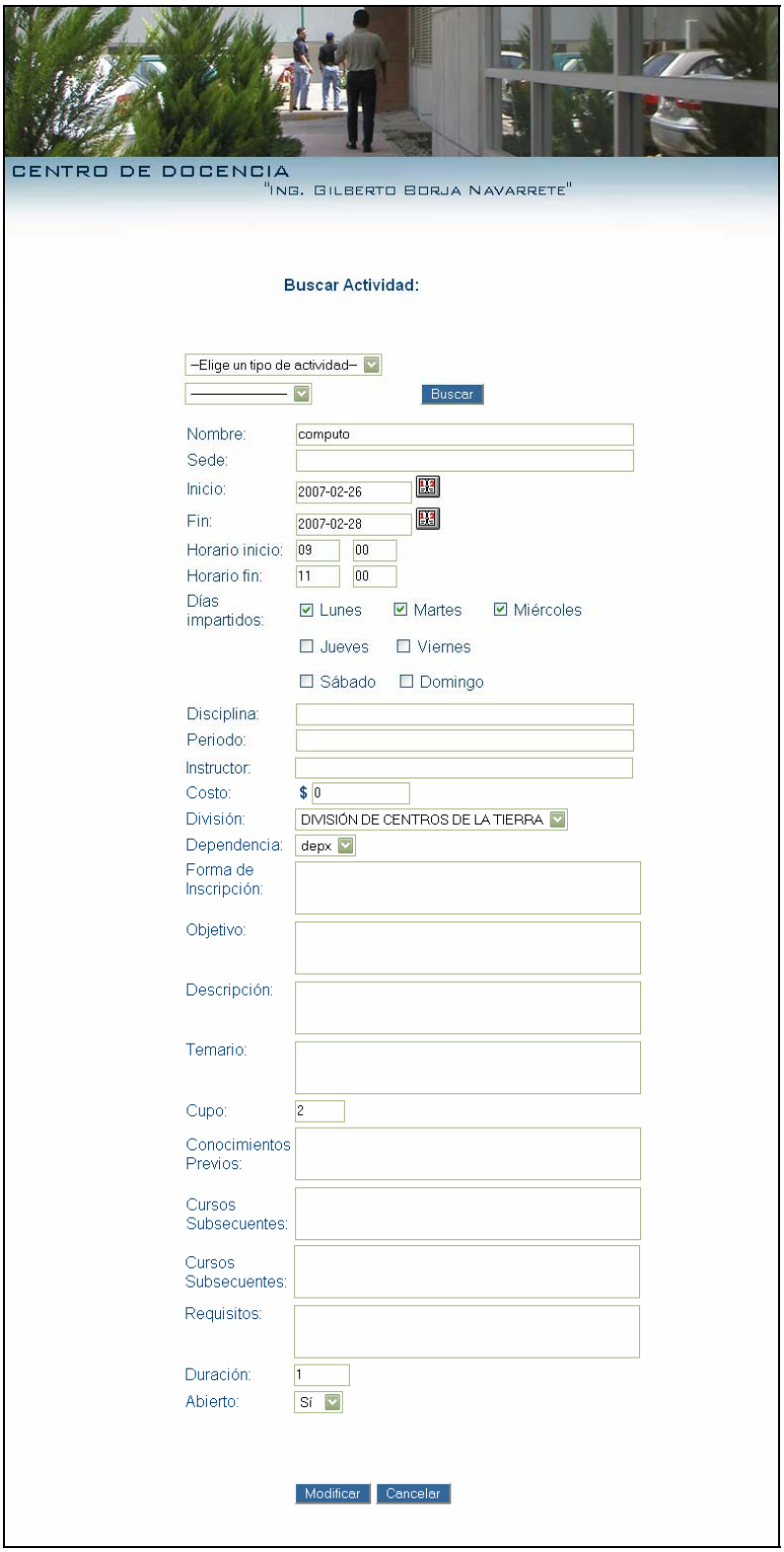
En esta pantalla se realiza la búsqueda de la división lo cual nos despliega los datos del jefe de división para poder modificar el nombre, login o contraseña del mismo. Si no se requiere modificar la contraseña se debe de contar con ella para hacer algún otro cambio.

Modificar actividad

En esta pantalla se muestra la búsqueda del tipo de actividad, el cuál nos despliega las posibles opciones en el segundo combo, una vez elegido el nombre de la actividad se realiza la búsqueda.

Modificar actividad (segunda pantalla)

Después de buscada la actividad se muestra una pantalla como la siguiente donde se despliegan todos los detalles de la actividad para así poder modificarlos.

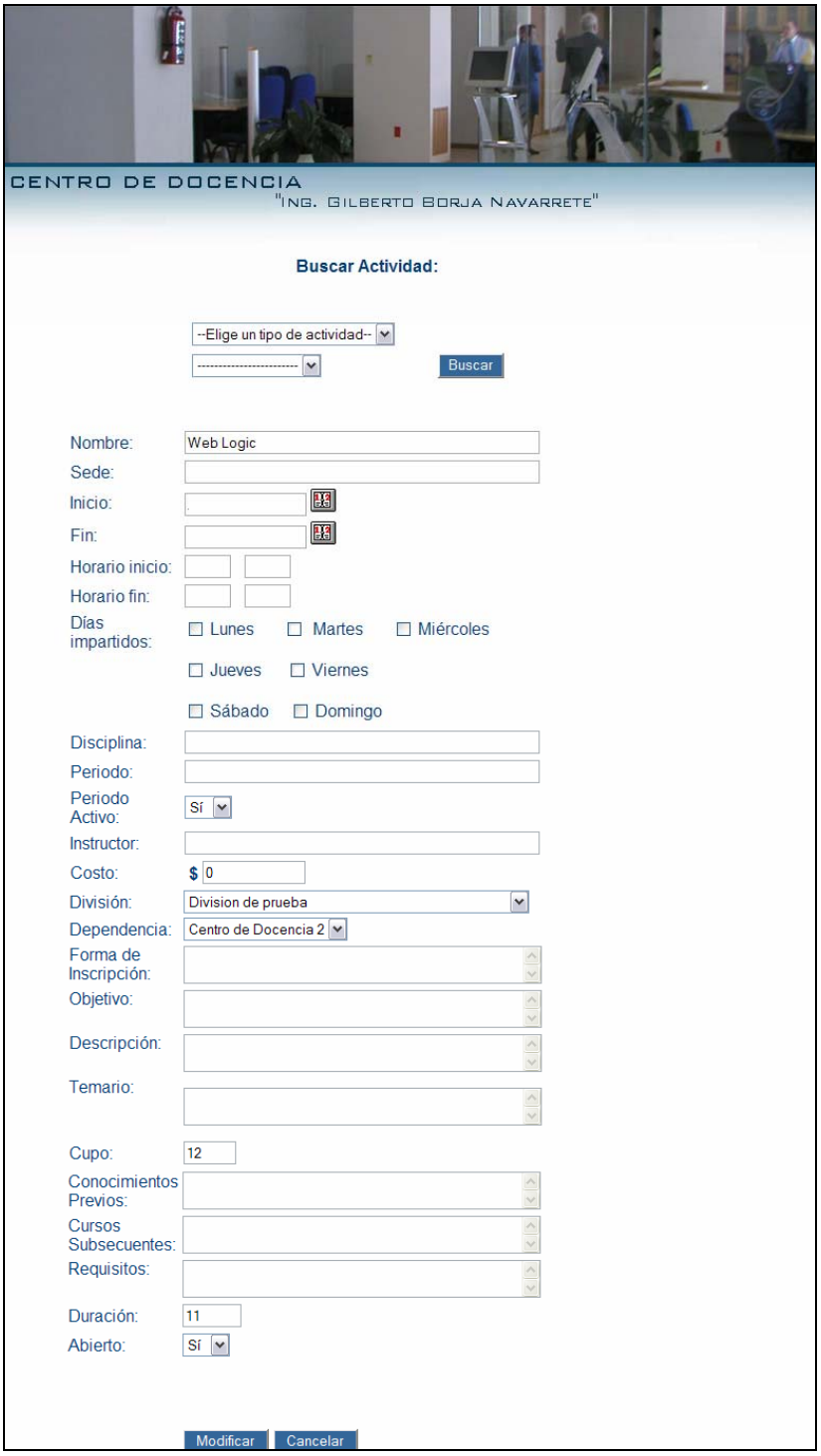


Plantillas establecidas

En esta pantalla se muestra la búsqueda del tipo de actividad, el cuál nos despliega las posibles opciones en el segundo combo las actividades existentes, una vez elegido el nombre de la actividad se realiza la búsqueda.

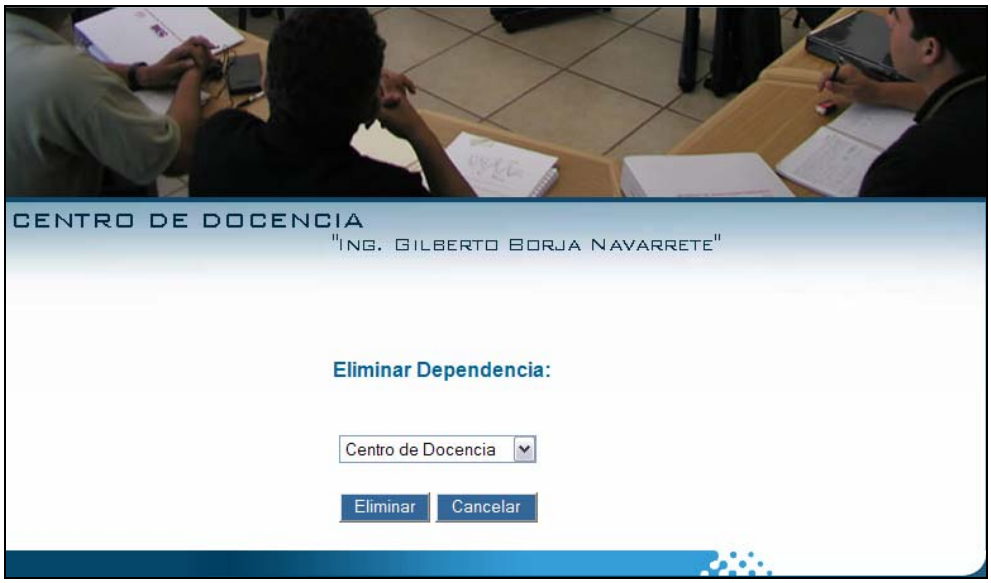
Plantillas establecidas (segunda pantalla)

Después de buscada la actividad se muestra una pantalla como la siguiente donde se despliegan los detalles básicos de la actividad para así poder insertar un nuevo registro de una actividad. Esta plantilla permite ingresar nuevas actividades de forma más rápida ya que son cargados datos genéricos, tales como: nombre, división, dependencia, cupo y duración.



Baja de Dependencia

Se tiene la siguiente pantalla de baja de dependencia. Se cuenta con un solo combo donde son cargadas las dependencias existentes.



Baja de actividad

Se tiene la siguiente pantalla de baja de actividad. Se cuenta con un combo de búsqueda de tipo de actividad y otro combo dinámico donde son cargados los nombres de las actividades correspondientes al tipo seleccionado.

Menú de responsable de División

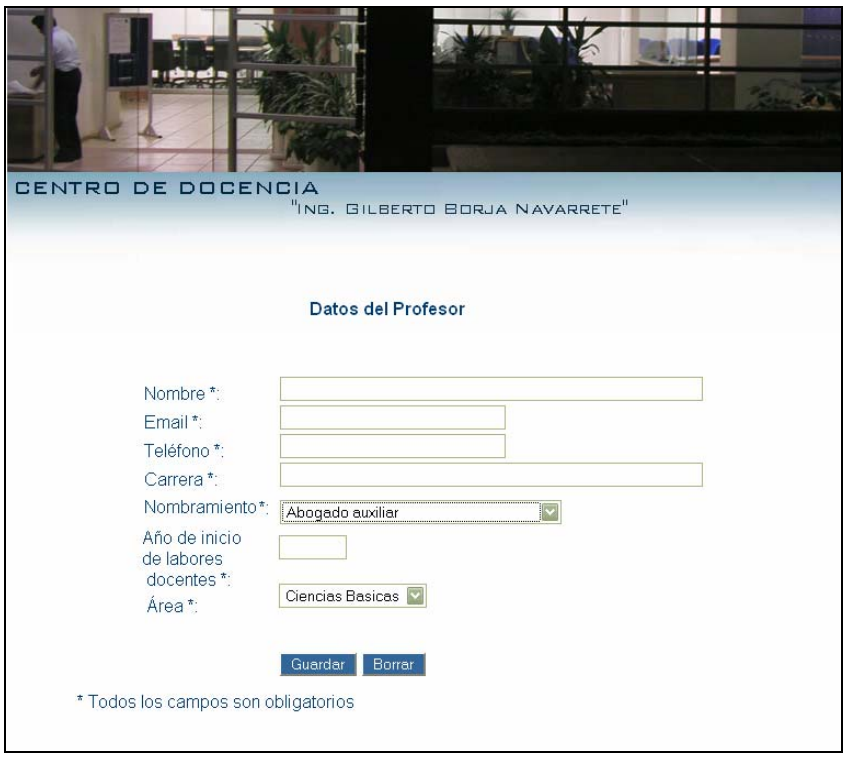
Este menú se despliega después que el usuario se autenticó y tiene rol de responsable de División.

Alta de actividad

La pantalla de Alta de Actividad es la misma pantalla que se describió con anterioridad en el rol de Administrador.

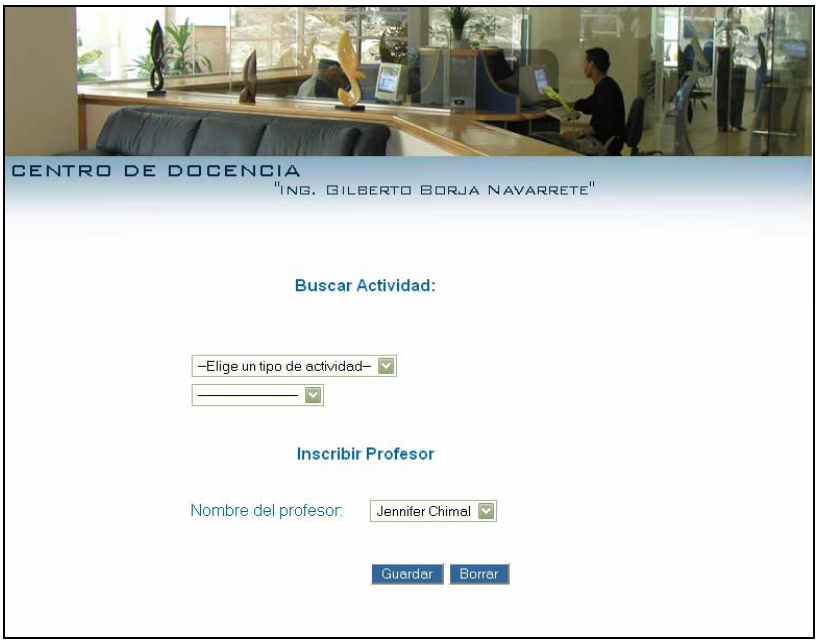
Alta de profesor

La pantalla de alta de profesor registra los datos del mismo como es nombre, email, teléfono, carrera, nombramiento, año de inicio de labores docentes y el área al que pertenece.



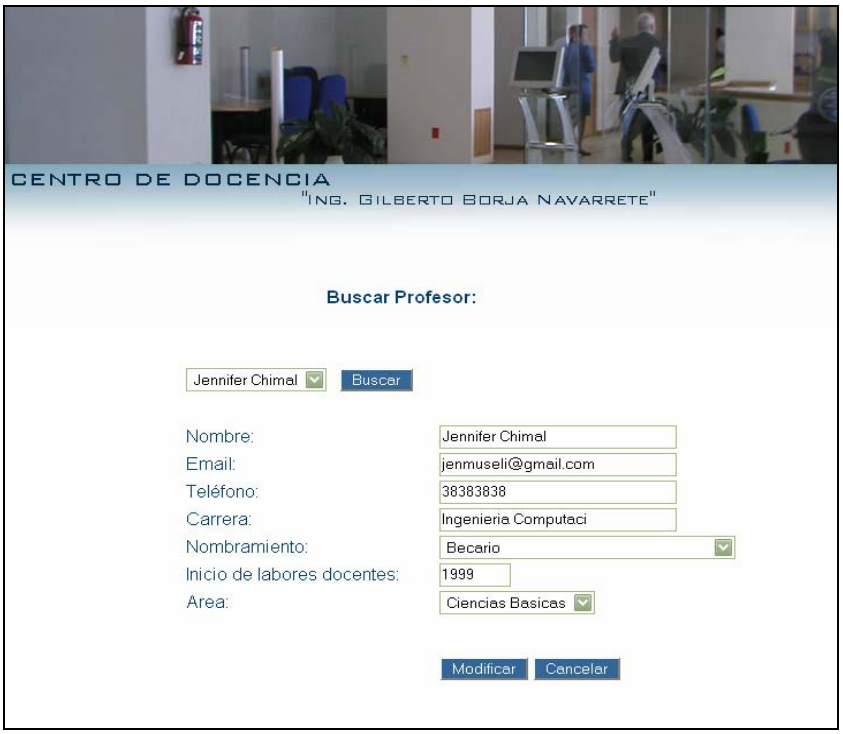
Inscribir profesor

La pantalla de inscribir profesor registra a un profesor dado de alta con anterioridad en una actividad en particular.



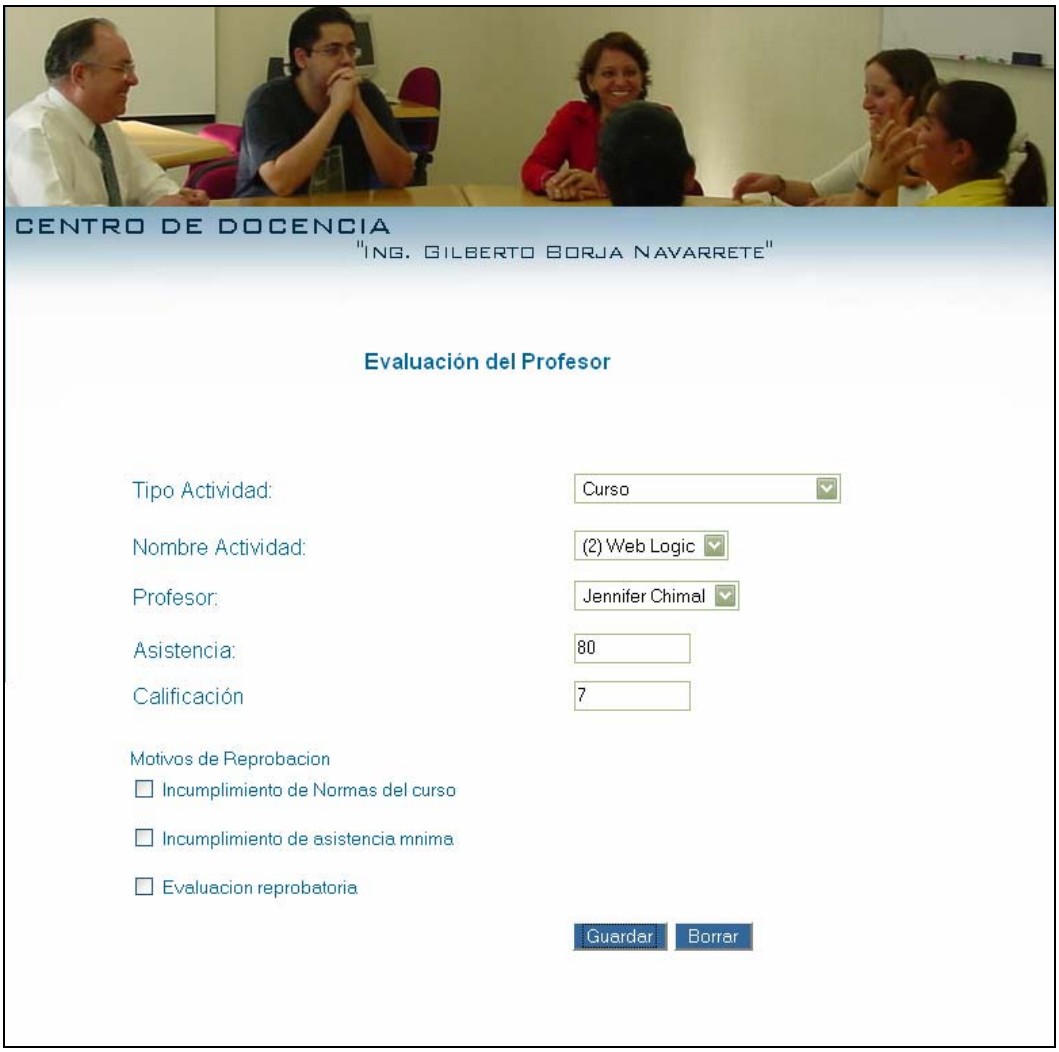
Modificar profesor

En esta pantalla se despliega una lista con el nombre de los profesores registrados en el sistema, una vez seleccionado el nombre del profesor se debe oprimir el botón “Buscar” y se despliegan los datos del profesor para su modificación.



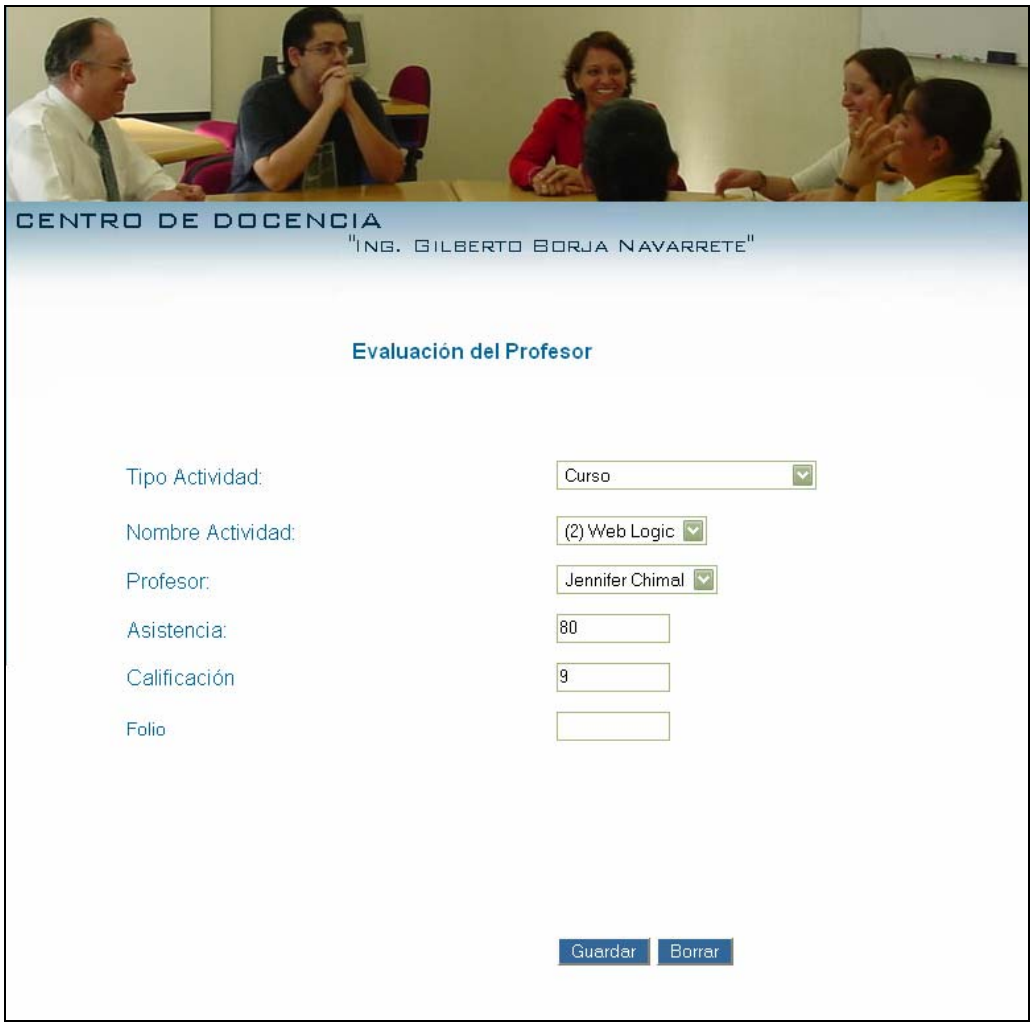
Calificar profesor

En esta pantalla se asigna la calificación del profesor inscrito con anterioridad a una actividad. Se selecciona el tipo y nombre de la actividad, el nombre del profesor a calificar. El profesor necesita tener una calificación mayor o igual a 8 y un porcentaje de asistencia mayor o igual al 80% para poder acreditar el curso, en caso habilita el apartado de “Motivos de Reprobación”.



Calificar profesor (Segunda pantalla)

Si el profesor acredita la actividad se tiene que ingresar el número de folio.



Eliminar profesor

En esta pantalla se elimina el registro de un profesor dado de alta en el sistema. Todos sus datos y registro de actividades son eliminados.

Módulo de jefe de División

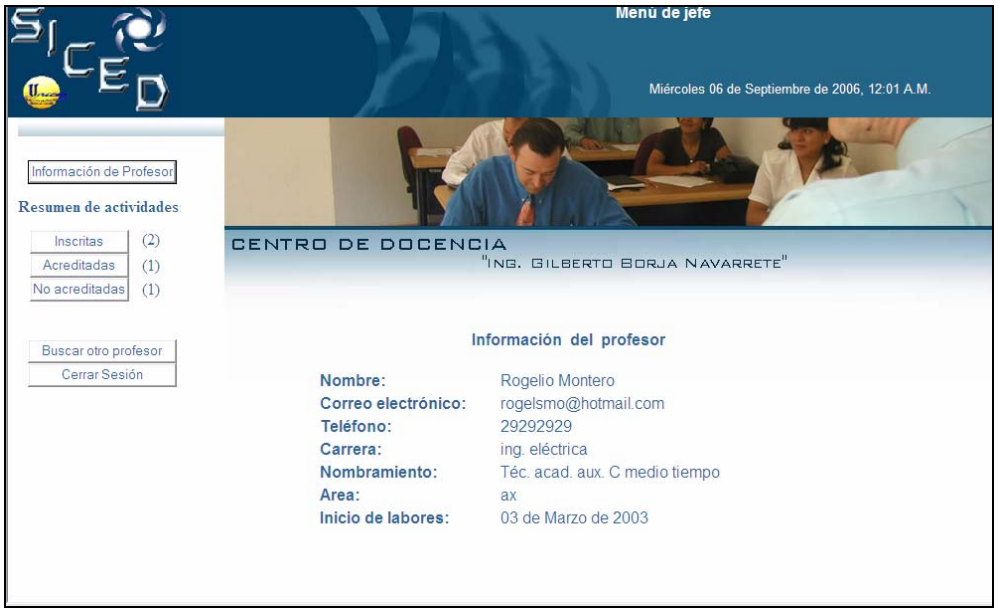
Consultar datos del profesor

Se pueden consultar los datos completos de un profesor dado de alta en el sistema y todo el historial de actividades cursadas.

Se debe seleccionar el profesor a consultar, solo se visualizan los profesores pertenecientes a la división en la que se encuentra el “Jefe de División”.

Consultar datos del profesor (Segunda pantalla)

Una vez seleccionado el profesor se muestran las actividades inscritas, acreditadas y no acreditadas. Al dar clic en algún rubro se muestra el detalle.



Búsquedas

Para consultar las actividades que cuenta el Centro de Docencia no se necesita autenticarse ya que es de uso público. Las búsquedas se pueden realizar por dependencia, fecha en que se imparte, División, disciplina, curso y evento.

Se muestran todas las actividades dependiendo la selección con un detalle del periodo, fecha, hora de inicio y fin, nombre del instructor, sede, costo, cupo y forma de inscripción. Si seleccionamos una de las actividades y damos clic en el botón “Ver Detalles” nos muestra el temario del curso.

Capítulo 6

Puesta en marcha

6.1 Garantía de calidad

La garantía de calidad es el conjunto de acciones que realizan los individuos, organizaciones y sociedad, de forma deliberada y sistemática para generar, mantener o mejorar calidad. Es una actividad esencial en cualquier empresa, ya que permite entregar al cliente un producto que cumpla con las expectativas solicitadas. En este caso, nuestro trabajo es producir un sistema que cumpla con los requerimientos del Centro de Docencia.

La garantía de calidad del software (SQA) es la guía de preceptos de gestión y de las disciplinas de diseño, para la aplicación de la Ingeniería de software.

Los factores que afectan la calidad del software se clasifica en dos grupos:

- Factores que pueden ser medidos directamente (como errores de programación, de infraestructura y más).
- Factores que solo pueden ser medidos indirectamente (como la facilidad de uso y facilidad de mantenimiento, entre otros).

Los factores de calidad del software se centran en tres aspectos importantes de un producto de software:

- Características operativas
- Capacidad de soportar los cambios
- Adaptabilidad a nuevos entornos

Estos factores se cumplieron en los siguientes puntos:

- Corrección. El grado en que un programa satisface sus especificaciones y consigue los objetivos de la misión encomendada por el cliente. Éste punto se cumplió, dado que el sistema atiende las especificaciones solicitadas en los formatos entregados al iniciar el proyecto (ver Anexo1).
- Fiabilidad. El grado en que se puede esperar que un programa lleve a cabo las funciones correspondientes con la precisión requerida. En éste aspecto, no es aplicable, dado que el sistema no realiza procesos que requieran precisión, como cálculos complejos o estadísticos.

- Eficiencia. La cantidad de recursos de computadora y de código requeridos por un programa para llevar a cabo sus funciones. Dado que el cliente no requiere tener instalado el sistema en su equipo de trabajo, se cumple con éste factor.
- Integridad. El grado en que puede controlarse el acceso al software o a los datos, por personal no autorizado. En este aspecto el sistema tiene seguridad en la autenticación y existe el rol Administrador para la creación de accesos a la aplicación y por consecuencia los datos en ella guardados.
- Facilidad de uso. El esfuerzo requerido para aprender un programa, trabajar con él, preparar su entrada e interpretar su salida. En éste sentido, el sistema es muy intuitivo, ya que todas las operaciones a realizar son realizadas en pocas etapas.
- Facilidad de Mantenimiento. El esfuerzo requerido para localizar y arreglar un error de un programa. La división del sistema en paquetes referidos a la tabla con la que están interactuando, le da ésta característica.
- Facilidad de prueba. El esfuerzo requerido para probar un programa de manera que se asegure que realiza su función requerida. Debido a que el sistema tiene facilidad de uso, se cumple con éste factor.
- Portabilidad. El esfuerzo requerido para transferir el programa desde un hardware y/o un entorno de sistemas de software a otro. Se cumple con ésta característica, ya que el sistema ha sido instalado en la plataforma Windows y Linux con el mismo comportamiento y desempeño.

Por lo descrito anteriormente, el sistema cumple con la garantía de calidad requerida para poder ser operado.

6.2 Pruebas

El objetivo de la fase de pruebas de un programa, es el detectar todo posible malfuncionamiento antes de que entre en producción. Un error detectado en el laboratorio puede ser costoso de reparar; pero siempre es peor que el error le aparezca al usuario final.

En esta idea, una batería de pruebas será de mayor calidad cuantos menos errores queden por descubrir tras haberla pasado. Y, viceversa, si un programa aún tiene muchos fallos tras haber superado una batería de pruebas, diremos que esta batería es de poca calidad.

Si se pudiera probar un programa con todos los posibles datos de entrada, tendríamos una batería de pruebas perfecta, pues no hay lugar para las sorpresas, lamentablemente, casi nunca es posible probar con todos los casos; en consecuencia necesitamos un criterio para elegir qué casos probamos.

6.2.1 Pruebas de caja negra

Consiste en tomar la especificación del programa o, en su defecto, el manual de usuario, ya que ambos determinan "lo que el programa tiene que hacer". Posteriormente, se debe hacer al menos una prueba de todas y cada una de las cosas que el programa tiene que hacer.

Es recomendable ir marcando las funcionalidades que se han probado dentro del manual; en cuanto se hayan revisado todas se puede decir que se tiene una cobertura funcional del 100%.

Es imprescindible lograr una cobertura de caja negra del 100%, pues los usuarios se quejarían con todo derecho de fallos de este calibre.

6.2.1.1 Datos de prueba

Cuando se prueban las funciones del manual hay que elegir datos concretos para ejercitar la prueba.

Es imprescindible usar datos "normales" en las pruebas, los cuales, es seguro que ingresará un usuario en la vida real, además de datos que pudieran provocar un error o inconsistencia. Un ejemplo de esto podría ser la programación de una división, en la que datos “normales” serían dos enteros (divisor y dividendo), y un dato que seguramente ocasionará un error será la división entre cero, por lo que se deben considerar todas las posibilidades.

Es importante identificar qué rangos de datos pueden alterar el comportamiento del sistema y así definir zonas de trabajo. Es imprescindible pasar pruebas con al menos un dato de cada zona, tanto si el programa debe funcionar como si debe dar un mensaje de error.

La experiencia indica, además, que suelen producirse fallos en los bordes de las zonas, por lo que se recomienda probar siempre con datos extremos.

En el caso del sistema SICED, se realizó la prueba de caja negra con éxito, ya que se tomaron cada una de las funcionalidades de los tres niveles de usuario y todas obtuvieron resultados satisfactorios.

6.2.2 Pruebas de caja blanca

Si las pruebas de caja negra arrojaron resultados satisfactorios, ahora se necesita realizar un listado del programa e ir marcando las líneas de código que se ejecutan, podremos determinar qué porcentaje de líneas han sido ejecutadas alguna vez, porcentaje que se conoce como cobertura de sentencias.

Si se logró una cobertura de caja negra del 100%, y una cobertura de sentencias del 100%, se pasa a la fase siguiente: prueba de condiciones.

En caso de que se descubran partes del código que no hayan sido ejecutadas jamás (la cobertura de sentencias es inferior al 100%), se deben realizar más pruebas buscando que se ejecuten todas y cada una de las sentencias del programa.

En el caso extremo de que no haya forma de ejecutar alguna parte del programa, es recomendable analizar si ese segmento es necesario, o se puede prescindir de él.

Es muy recomendable alcanzar una elevada cobertura de sentencias, aunque no siempre es posible por premura de tiempo o medios.

En el caso del sistema SICED, se realizaron las pruebas de caja blanca con éxito, dado que varias partes del código son reutilizadas, y se diseñaron por separado de acuerdo a su funcionalidad.

Capítulo 7

Etapa de mantenimiento del Sistema

7.1 Mantenimiento

Para poder garantizar el buen funcionamiento de la aplicación, se pretende llevar a cabo el mantenimiento de la misma, a continuación se describen los tipos de mantenimiento que se pueden aplicar.

7.1.1 Mantenimiento preventivo

Es aquél que reduce la frecuencia con que ocurren las causas de riesgo, permitiendo cierto margen de violaciones.

Para este caso, se recomienda tener un monitoreo constante de la aplicación dentro del servidor, revisando que los procesos de java, tomcat y postgresql se encuentren corriendo. En la parte del servidor se recomienda verificar el tamaño de los filesystems. Con lo que respecta a la base de datos se tiene que verificar que los table spaces no se encuentren al máximo de su capacidad. Contar con una política de seguridad de usuarios para manejar passwords con caracteres alfanuméricos de más de 6 dígitos.

Realizar backups tanto de la aplicación completa como de la base de datos, recomendado para cada semana.

7.1.2 Mantenimiento detectivo

Es aquel que no evita que ocurran las causas del riesgo sino que los detecta luego de ocurridos. En cierta forma sirven para evaluar la eficiencia de los controles preventivos.

Se deben tener ubicados los logs de fallas de la aplicación para poder verificarlos en caso de riesgo de la aplicación, se debe tener un control de accesos adecuado tanto en el servidor como en la aplicación. Verificar los logs del sistema operativo.

7.1.3 Control correctivo

Ayuda a la investigación y corrección de las causas del riesgo. La corrección adecuada puede resultar difícil e ineficiente, siendo necesaria la implantación de controles detectivos sobre los controles correctivos, debido a que la corrección de errores es en si una actividad altamente propensa a errores.

Una vez detectada la falla se procederá a realizar la corrección del mismo con un análisis pertinente.

7.2 Perspectivas a futuro

Con base en lo aprendido en el diseño, desarrollo e implementación de éste sistema; se tienen las siguientes perspectivas para la mejora y ampliación del mismo:

- Implementación de bitácoras de errores con Log4j
- Implementación de un thread que coordine el envío de correos con la información de cursos a los profesores registrados en el sistema de manera periódica
- Almacenamiento de temarios en archivos tipo Word (.doc)

Estos tres puntos se piensan desarrollar más adelante; esto con la finalidad de facilitar más la administración del sistema.

7.3 Contribución

Éste sistema contribuirá en el desempeño del Centro de docencia, ya que facilitará la difusión de actividades y la evaluación de profesores.

A su vez, proporcionará a los jefes de división datos acerca de los profesores que están adscritos a la división que tienen a su cargo, así como su desempeño en las diversas actividades a las que fueron inscritas; esto con la finalidad de facilitar una evaluación del aprovechamiento de éste beneficio en las diferentes divisiones.

Tecnológicamente, el sistema es portable, por que sin importar que en un futuro el servidor del Centro de docencia cambiara al sistema operativo Windows (ya que actualmente se encuentra en linux), podrá funcionar de igual manera, sólo tomando en cuenta que se tenga instalada la misma versión de java (jsdk).

Finalmente, se tendrá un fácil mantenimiento, dada la utilización del patrón MVC; que como se ha mencionado anteriormente, separa las capas de modelo, vista y control.

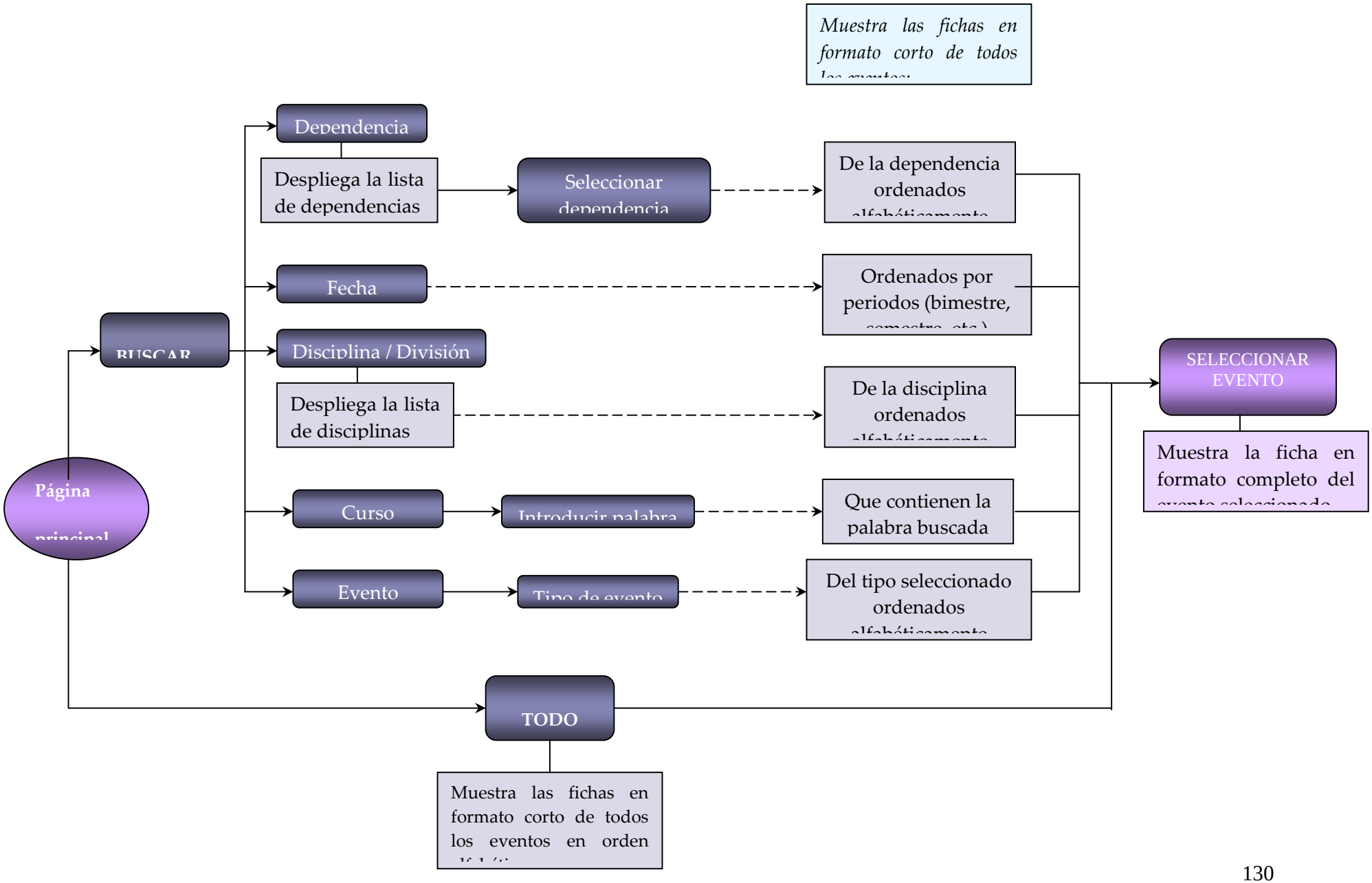
Conclusiones

Con el desarrollo del sistema, se llegaron a las siguientes conclusiones:

- El sistema cumple con todos los requerimientos registrado en el anexo 1 del presente documento.
- Superó las expectativas, ya que en los requerimientos iniciales (ver anexo 1), sólo se solicitaba la creación de una aplicación que permitiera la búsqueda de las actividades impartidas por el Centro de Docencia, así como una sección para consulta de las actividades: inscritas, acreditadas y no acreditadas por los profesores; para poder alcanzar éstas metas, se implementó un módulo de administración que permitiera registrar las actividades a mostrar; el alta, inscripción y evaluación de los profesores; todo esto aplicando seguridad por división.
- Se cumplió con el objetivo de acceder al sitio vía web, aplicando la tecnología J2EE anteriormente descrita.
- El sistema cumple con el requisito de tener cuentas únicas por usuario (responsable y jefe de división) con su respectiva contraseña, los cuáles podrán ser modificados de acuerdo a las necesidades de los administradores y usuarios del sistema.
- Se cuenta con la documentación necesaria para el fácil manejo y mantenimiento del sistema.
- La interfaz de búsqueda es de fácil acceso y permite al usuario localizar la información por diversos criterios, de acuerdo a los requerimientos del Centro de Docencia.

Anexo 1

Requerimientos del Sistema


Muestra las fichas en formato corto de todos los eventos;
Dependencia
Despliega la lista de dependencias
Seleccionar dependencia
De la dependencia ordenados
Fecha
Ordenados por periodos (bimestre,
BUSCAR
Disciplina / División
Despliega la lista de disciplinas
De la disciplina ordenados
SELECCIONAR EVENTO
Muestra la ficha en formato completo del
Página principal
Curso
Introducir palabra
Que contienen la palabra buscada
Evento
Tipo de evento
Del tipo seleccionado ordenados
TODO
Muestra las fichas en formato corto de todos los eventos en orden

/* INICIO */

CURSOS, TALLERES, CONFERENCIAS

Búsqueda:

Curso (nombre)
Dependencia
Disciplina / División
Periodo
Tipo de evento

Mostrar todo

/* Salida búsqueda por dependencia */

Atrás INICIO Búsqueda

Categoría: DEPENDENCIA / DIVISIÓN

Seleccione una dependencia:

- Universidad Nacional Autónoma de México
- Facultad de Ingeniería, UNAM
- División de Ciencias Básicas
- División de Ciencias Sociales y Humanidades
- División de Educación Continua
- Secretaría de Estudios de Posgrado
- División de Ingeniería Civil Topográfica y Geodésica
- División de Ingeniería Eléctrica
- División de Ingeniería en Ciencias de la Tierra
- División de Ingeniería Mecánica e Industrial
- Otras

/* Salida búsqueda por disciplina */

 Atrás INICIO Búsqueda

CATEGORÍA

Seleccione una disciplina:

- Cómputo
- Formación Docente
- Disciplinares
- Desarrollo humano

/* Salida búsqueda por periodo */

ANTERIOR SIGUIENTE

NOVIEMBRE – DICIEMBRE 2006 /* periodo */

CURSOS

"*Nombre del curso*"

Periodo: _____
Horario: _____
Cupo: _____
Costo: $_____
Instructor: _____
Forma de inscripción: (Telefónica, presencial y/o e-mail)
Dependencia: _____

SEMINARIOS

"*Nombre del seminario*"

Periodo: _____

Horario: _____

Cupo: _____

Costo: $_____

Instructor: _____

Forma de inscripción: (Telefónica, presencial y/o e-mail)

Dependencia: _____

 Atrás INICIO Búsqueda

/* Salida búsqueda por nombre */

Resultados para "*palabra ingresada*"

CURSOS

"*Nombre del curso*"

Periodo: _____

Horario: _____
Cupo: _____
Costo: $_____
Instructor: _____
Forma de inscripción: (Telefónica, presencial y/o e-mail)
Dependencia: _____

SEMINARIOS

"*Nombre del seminario*"
Periodo: _____
Horario: _____
Cupo: _____
Costo: $_____
Instructor: _____
Forma de inscripción: (Telefónica, presencial y/o e-mail)
Dependencia: _____

 Atrás INICIO Búsqueda

/* Salida búsqueda por tipo de evento */

/* Tipo de evento */

CURSOS

"*Nombre del curso*"
Periodo: _____
Horario: _____
Cupo: _____
Costo: $_____
Instructor: _____
Forma de inscripción: (Telefónica, presencial y/o e-mail)
Dependencia: _____

"*Nombre del curso*"
Periodo: _____
Horario: _____
Cupo: _____
Costo: $_____
Instructor: _____
Forma de inscripción: (Telefónica, presencial y/o e-mail)
Dependencia: _____

/* Fichas cortas */

NOTA: **En caso de no acreditar el curso, taller o diplomado, debe cubrir el costo del mismo:**

CURSOS

"*Nombre del curso*"
Periodo: _____
Horario: _____
Cupo: _____
Costo: $_____
Instructor: _____
Forma de inscripción: (Telefónica, presencial y/o e-mail)
Dependencia: _____

"*Nombre del curso 2*"
Periodo: _____
Horario: _____
Cupo: _____
Costo: $_____
Instructor: _____
Forma de inscripción: (Telefónica, presencial y/o e-mail)
Dependencia: _____

SEMINARIOS

"*Nombre del seminario*"
Periodo: _____
Horario: _____
Cupo: _____
Costo: $_____
Instructor: _____
Forma de inscripción: (Telefónica, presencial y/o e-mail)

Dependencia: _____

TALLERES

"*Nombre del taller*"

Periodo: _____
Horario: _____
Cupo: _____
Costo: $_____
Instructor: _____
Forma de inscripción: (Telefónica, presencial y/o e-mail)
Dependencia: _____

CONFERENCIAS

"*Nombre de la conferencia*"

Fecha: _____
Horario: _____
Cupo: _____
Costo: $_____
Ponente: _____
Forma de inscripción: (Telefónica, presencial y/o e-mail)
Dependencia: _____

MESAS REDONDAS

OTROS

/* Ficha completa: Cursos, talleres, seminarios, diplomados */

NOMBRE DEL CURSO

- **OBJETIVO:**
- **DESCRIPCIÓN Y TEMARIO:**
- **CONOCIMIENTOS PREVIOS:**
- **CURSOS SUBSECUENTES:**
- **SEDE:**
- **INSCRIPCIONES:**
- **REQUISITOS:**

Objetivo del curso

Descripción de actividades a realizar y temario.

Conocimientos previos necesarios

(Omitir si no hay cursos subsecuentes)

Lugar: Sala de Cómputo del Centro de Docencia; Auditorio Javier Barros Sierra, edificio Principal de la Facultad de Ingeniería.

Fechas de inscripciones, lugar y forma de inscripción: (Telefónica, presencial y/o e-mail).

En caso de ser profesor externo, la inscripción se valida al momento de cubrir el costo del curso.

/* Ficha completa: Conferencias, videoconferencias, seminarios, mesas redondas */

INICIA: FECHA DE INICIO

NOMBRE DE LA CONFERENCIA

- **OBJETIVO:**
- **DESCRIPCIÓN**:
- **SEDE:**
- **INSCRIPCIONES:**
- **REQUISITOS**

Objetivo

Descripción de actividades a realizar y/o tema

Lugar: Sala de Cómputo del Centro de Docencia; Auditorio Javier Barros Sierra, edificio Principal de la Facultad de Ingeniería

Fechas de inscripciones, lugar y forma de inscripción:
(Telefónica, presencial y/o e-mail) (si es necesario)

En caso de ser profesor externo, la inscripción se valida al momento de cubrir el costo de la conferencia.

INFORMACIÓN DEL PROFESOR

Información personal	Nombre completo del profesor Área (departamento, coordinación, secretaría)
Resumen de actividades Inscritas (#) Acreditadas (#) No acreditadas (#)	División base. E-mail: Teléfono:

NOMBRE DEL PROFESOR

	Actividades Inscritas
Información personal	
Resumen de actividades Inscritas (#) Acreditadas (#) No acreditadas (#)	Curso: "Nombre del curso" Periodo: _____ Horario: _____ Instructor: _____

Sede:_____

Actividades Acreditadas: #

Curso: "Nombre del curso"
Periodo: _____
Instructor: _____
Impartido por: Dependencia

Taller: "Nombre del taller"
Periodo: _____
Instructor: _____
Impartido por: Dependencia

Actividades no acreditadas: #

Curso: "Nombre del curso"
Periodo: _____
Instructor: _____
Motivo:
Incumplimiento de Normas del curso.

Incumplimiento de asistencia mínima.
Evaluación reprobatoria.
Impartido por: Dependencia

Taller: "Nombre del taller"
Periodo: _____
Instructor: _____
Motivo:
Incumplimiento de Normas del curso.
Incumplimiento de asistencia mínima.
Evaluación reprobatoria.
Impartido por: Dependencia

Actividades no asistidas

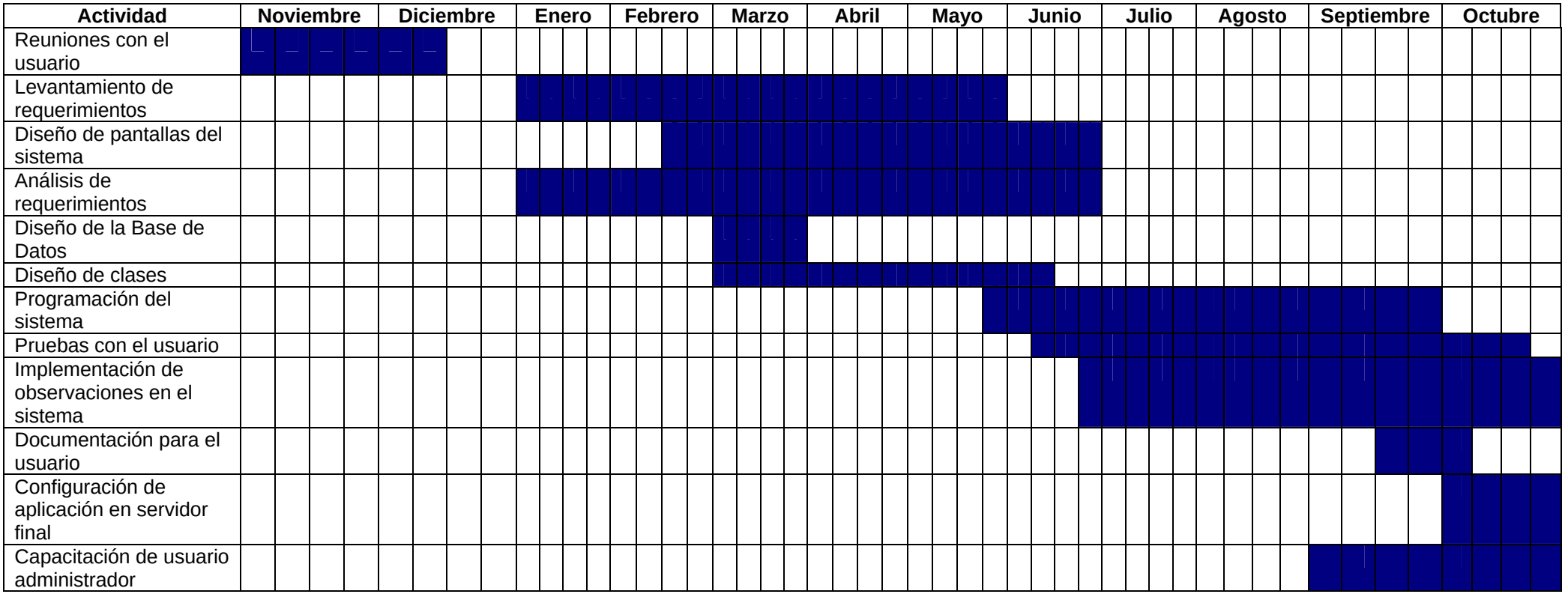
Seminario: "Nombre del seminario"
Periodo: _____
Instructor: _____
Impartido por: Dependencia

Anexo 2

Diagrama de Gantt de la Planeación del Sistema

Planeación

La planeación del sistema está representada mediante el siguiente diagrama de Gantt, en él se detallan las actividades más sobresalientes del diseño e implementación del sistema.



Bibliografía

- BOOCH, Grady. 2006. El lenguaje unificado de modelado. 2ª. edición. Pearson.
- CEBALLOS, Francisco Javier. Java 2: Lenguaje y aplicaciones. Ra-ma.
- CEBALLOS, Francisco Javier. 2006. Java 2 Curso de Programación. 3ra. edición. Madrid, España. Alfaomega Ra-Ma.
- ECKEL, Bruce. 2006. Thinking in java. 4ta. edición. Prentice Hall
- HALL, Marti. 2000. Core Servlets and Java Server Pages. Prentice Hall PTR.
- JOYANES, Luis. 1998. Programación orientada a objetos. 2ª. edición. Madrid, España. Mcgraw-Hill.
- KIMMEL, Paul. Manual de UML. Mcgraw-Hill.
- PRESSMAN, Roger. Ingeniería del software. 2ª. edición. Mcgraw-Hill.

Referencias electrónicas

- http://www.ulfix.com/index.php?option=com_content&task=view&id=37&Itemid=137
- http://programacion.com/tutorial/uml/0/
- http://manuales.dgsca.unam.mx/jsp/
- http://es.wikipedia.org/wiki/Lenguaje_Unificado_de_Modelado
- http://www.inei.gob.pe/biblioineipub/bancopub/inf/Lib5042/cap20.htm
- http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/rweb_webxf.html
- http://e-docs.bea.com/wls/docs61/webapp/web_xml.html#1026980

Glosario

AOO: Análisis orientado a objetos.

API: Es la Interfaz de Programación de Aplicaciones, del inglés *Application Programming Interface*. es un conjunto de especificaciones de comunicación entre componentes software. Se trata del conjunto de llamadas al sistema que ofrecen acceso a los servicios del sistema desde los procesos y representa un método para conseguir abstracción en la programación, generalmente (aunque no necesariamente) entre los niveles o capas inferiores y los superiores del software. Uno de los principales propósitos de una API consiste en proporcionar un conjunto de funciones de uso general, por ejemplo, para dibujar ventanas o iconos en la pantalla. De esta forma, los programadores se benefician de las ventajas de la API haciendo uso de su funcionalidad, evitándose el trabajo de programar todo desde el principio. Las APIs asimismo son abstractas: el software que proporciona una cierta API generalmente es llamado la implementación de esa API.

CORBA: (Common Object Request Broker Arquitecture).

DTD : (Document Type Definition), la definición de tipo de documento (DTD) es una descripción de estructura y sintaxis de un documento XML o SGML. Su función básica es la descripción del formato de datos, para usar un formato común y mantener la consistencia entre todos los documentos que utilicen la misma DTD.

J2EE: (Java 2 Enterprise Edition).

JDBC: Es el acrónimo de *Java Database Connectivity*, un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema de operación donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice.

JSP: Java Server Pages.

LPOO: Lenguajes de Programación Orientados a Objetos.

OMG: Grupo Manejador de Objetos (Object Management Group).

OO: Orientación a objetos.

OOSE: Ingeniería de Software Orientada a Objetos (Object-Oriented Software Engineering).

ORB: (Object Request Broker).

POO: Programación orientada a objetos.

SQA: Software Quality Assurance.

TAD: Tipo abstracto de dato.

Thread: Es un hilo de ejecución en un programa. La máquina virtual de java permite a una aplicación tener múltiples hilos de ejecución corriendo concurrentemente.

XML: (e**X**tensible **M**arkup **L**anguage), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C).