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

Proyecto Terminal:
SISTEMA DE GESTIÓN DE EVENTOS ACADÉMICOS

DOCUMENTO DE DISEÑO

Alumno:
Manuel Alejandro Cobos Lomelí
Matrícula: 205305635

Asesor:
Dra. María Lizbeth Gallardo López
Profesor investigador
Departamento de Sistemas

1.Especificación del sistema

1.1. Requerimientos del Proyecto

En este esquema se describen los requerimientos obtenidos del análisis del proyecto, los cuales se encuentran dentro de la fase de inicio del Proceso Unificado

Requisitos	Solución actual	prioridad	Comentarios adicionales
Gestionar uno o varios eventos académicos	Se crea una aplicación Web específica para el evento; en ocasiones se recurre a crear un sitio Web (donde solo se muestra la información) específico para el evento.	Alta	Buscamos tener una sola aplicación que permita gestionar distintos eventos, inclusive al mismo tiempo.
Gestionar las actividades de distintos eventos.	En ocasiones todo se organiza en hojas de calculo y en documentos de texto, para luego encargar a alguien “vaciar” la información en la aplicación o en el sitio Web, específicos del evento.	Alta	Buscamos repartir el trabajo y que existan responsables por cada actividad definida en un evento; la persona responsable deberá encargarse de registrar la información correspondiente.
Gestionar la información del evento	Tenemos dos modalidades para la gestión de la información: 1) reportes, cuando el evento está en proceso y 2) reportes, cuando el evento ya concluyó. Este trabajo se realiza a partir de los registros (guardados en los distintos documentos); o a partir de la base datos (cuando existe).	Alta	Buscamos poder contar con reportes a partir de que el evento sea creado en el sistema, y aún después de terminado. Durante el evento, los reportes permitirán retroalimentar a los organizadores. Al final del evento, los reportes permitirán tener una clara idea del éxito del evento.

Tabla 1. Requisitos del proyecto.

1.1.1. Requerimientos funcionales

Para la configuración inicial de eventos, el sistema debe permitir:

1. Alta, baja, cambio y consulta de un evento.
2. Definir a los integrantes del comité organizador, así como sus roles dentro del evento.
3. Determinar las actividades que se realizarán en el marco del evento.
4. Definir el número (cupo) de participantes que asistirán al evento.

Para la Gestión de las actividades del evento**,** el sistema debe de permitir:

1. Altas, bajas, cambios, consultas y reportes de las distintas actividades para el evento.
2. Confirmar las actividades definidas en el sistema.

Para la Gestión de la información del evento el sistema de permitir:

1. Obtener, distintos reportes, sobre la información del evento, correspondiente a las actividades registradas y el comité organizador.
2. Exportar los reportes en diferentes formatos, a saber: excel, pdf, etc.

1.1.2. Requerimientos no funcionales

1. Proporcionar acceso a la base de datos del sistema, para que quien gestiona la página de la División pueda obtener los datos (incluida la URL) de los eventos activos en el sistema "Eventos Académicos" así como su descripción general.
2. Que la base de datos esté centralizada y pueda ser accedida de forma remota, para que con ello se tenga la información necesaria para gestionar de manera global los eventos, reportes, usuarios y permisos correspondientes.
3. Que el sistema soporte la concurrencia de los usuarios.
4. Que las plantillas de presentación sean definidas bajo estándares de usabilidad.
5. Que el sistema implemente el modelo MVC[1] para su mejor mantenibilidad y extensibilidad con miras en cambios futuros.
6. El sistema deberá contar con un mecanismo de parametrización para la instalación, despliegue y configuración ya sea manual o asistido por el propio sistema.
7. El sistema terminado deberá ser empaquetado para su mejor distribución, manejo de versiones y licenciamiento de uso.

2. Casos de uso

Esta etapa consta de los siguientes esquemas: diagrama de casos de uso general, así como sus escenarios correspondientes, caso de uso actividades académicas

1 Modelo-Vista-Controlador

así como sus escenarios correspondientes.

2.1. Caso de uso general

La figura 1 muestra el caso de uso general señalando las acciones que el sistema es capaz de realizar, a saber: generar eventos académicos los cuales contarán con actividades que podrán variar dependiendo del evento que organice un comité determinado.

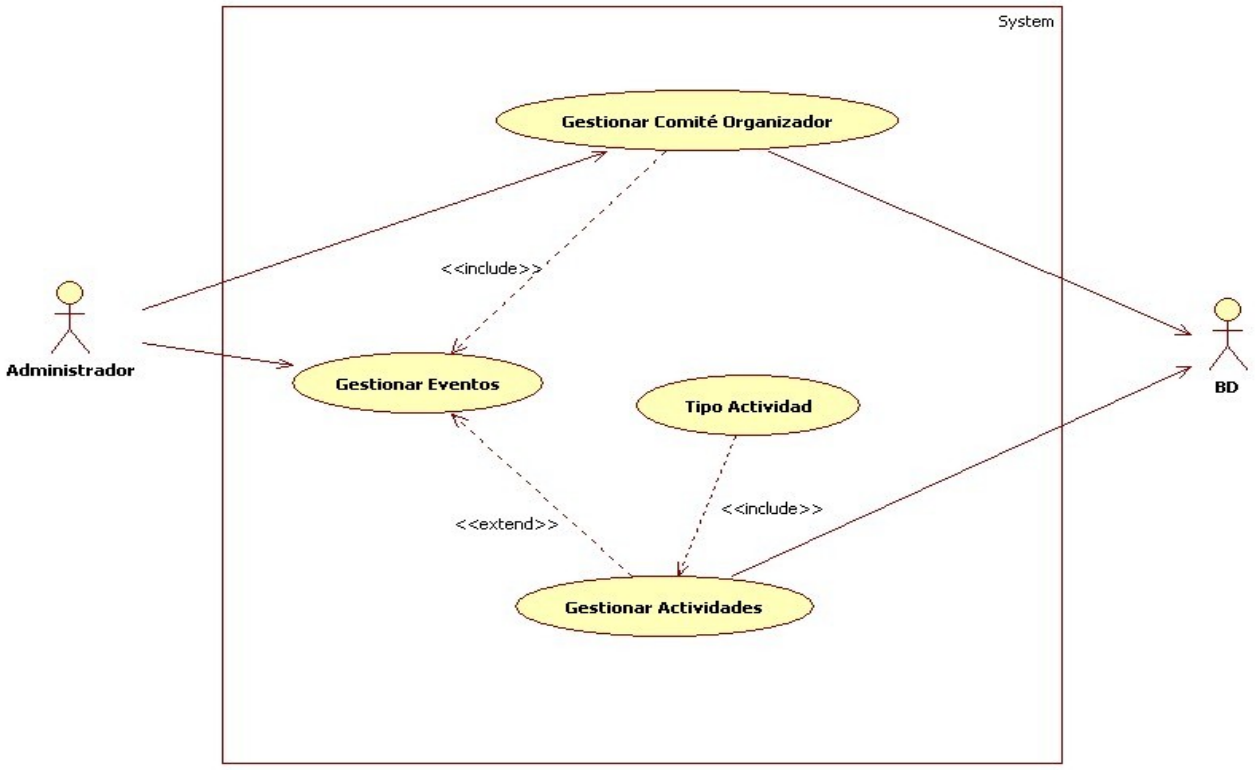


Figura 1. Diagrama de casos de uso general

2.1.1 Caso de uso: Gestionar Eventos.

El administrador tendrá la facultad de generar, consultar, modificar o borrar un evento académico en el sistema. A continuación se describe el escenario de uso de generar un evento académico, para revisar los otros escenarios consultar el documento de diseño.

Caso de uso:	Generar evento
Descripción:	El usuario tendrá la posibilidad de generar un evento académico, insertando en el sistema los datos correspondientes a la actividad que desea generar, esto es, podrá ingresar el nombre del evento, una descripción del mismo, la fecha de creación, una autorización y el tipo de evento.

2.1.2 Caso de uso: Gestionar Actividades.

El administrador tendrá la facultad de generar, consultar, modificar o borrar una actividad académica en el sistema. A continuación se describe el escenario de uso de consultar actividad.

Caso de uso:	Consultar Actividad
Descripción	El usuario tendrá la posibilidad de consultar una actividad asociada a un evento académico, esto se llevará a cabo seleccionando un evento académico y éste mostrará las actividades que corresponden al evento académico; por lo cual, al seleccionar la clave de la actividad, se mostrará la actividad académica particular, mostrando: su nombre, a quien va dirigido, las bases de la actividad, una breve descripción, un objetivo, el tipo de actividad, cupo, cortesías, costo interno y externo, autorización, fecha de inicio y de fin así como la fecha de creación.

2.2 Caso de uso: Eventos Académicos

En la figura 2 se muestra el caso de uso para eventos, señalando las acciones que el sistema es capaz de realizar, a saber: crear eventos, modificar eventos, borrar eventos o consultar eventos académicos.

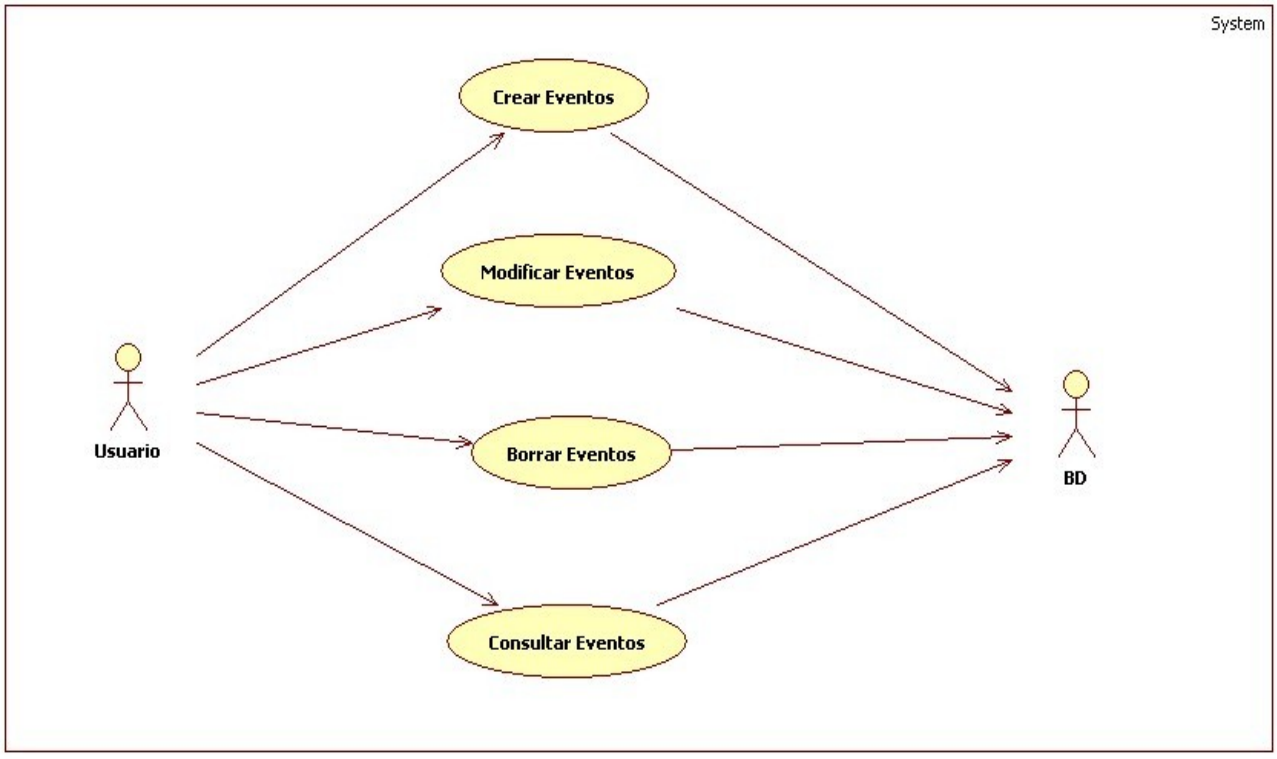


Figura 2. Caso de uso Eventos Académicos

2.2.1. Caso de uso: Crear Eventos

El administrador tendrá la facultad de crear un evento académico en el sistema. A continuación se describe el escenario de uso de crear eventos.

Caso de uso:	Crear Eventos
Descripción	El administrador tendrá la posibilidad de crear un evento académico, esto se llevará a cabo ingresando ciertos datos, a saber: nombre del evento, una breve descripción, una autorización, la fecha de creación y el tipo de evento que se creara.

2.2.2. Caso de uso: Modificar Eventos

El administrador tendrá la facultad de modificar un evento académico en el sistema. A continuación se describe el escenario de uso de `modificar eventos`.

Caso de uso:	Modificar Eventos
Descripción	El administrador tendrá la posibilidad de modificar un evento académico, esto se llevará a cabo seleccionando un evento académico por medio de la clave, acto seguido, se mostrarán las propiedades que componen al evento académico, ésto es: su nombre, la descripción del evento, si está autorizado o no, la fecha de creación y el tipo de evento. Al estar plasmadas sus propiedades se llevará a cabo la modificación del o de los campos que se quieran modificar ejecutando la acción correspondiente (actulizar_evento.action) y mostrando el resultado de la modificación en pantalla.

2.2.3. Caso de uso: Borrar Eventos

El administrador tendrá la facultad de borrar un evento académico en el sistema. A continuación se describe el escenario de uso de borrar eventos.

Caso de uso:	Borrar Eventos
Descripción	El administrador tendrá la posibilidad de borrar un evento académico, esto se llevará a cabo seleccionando un evento académico por medio de la clave, acto seguido, se le preguntará al administrador si quiere eliminar el evento y al confirmar la acción se borrara el evento y se mostrarán en la pantalla los eventos restantes.

2.2.4. Caso de uso: Consultar Eventos

El administrador tendrá la facultad de consultar un evento académico en el sistema. A continuación se describe el escenario de uso de consultar eventos.

Caso de uso:	Consultar Eventos
Descripción	El administrador podrá consultar un evento académico, esto se llevará a cabo seleccionando un evento académico por medio de la clave, acto seguido, se mostrarán las propiedades que componente al evento académico, ésto es: su nombre, la descripción del evento, si está autorizado o no, la fecha de creación y el tipo de evento.

2.3. Caso de uso: Actividades Académicas

El diagrama de este caso de uso es similar al de eventos académicos y al igual que el caso de uso de comité organizador, dichos casos de uso utilizan el mismo esquema de acciones tanto para la inserción, modificación, consulta y eliminación ya sea de actividades como de un comité organizador.

3. Modelo de dominio

La figura 3 presenta el modelo de dominio con las principales entidades que conforman un evento académico, así como las relaciones que guardan entre ellas.

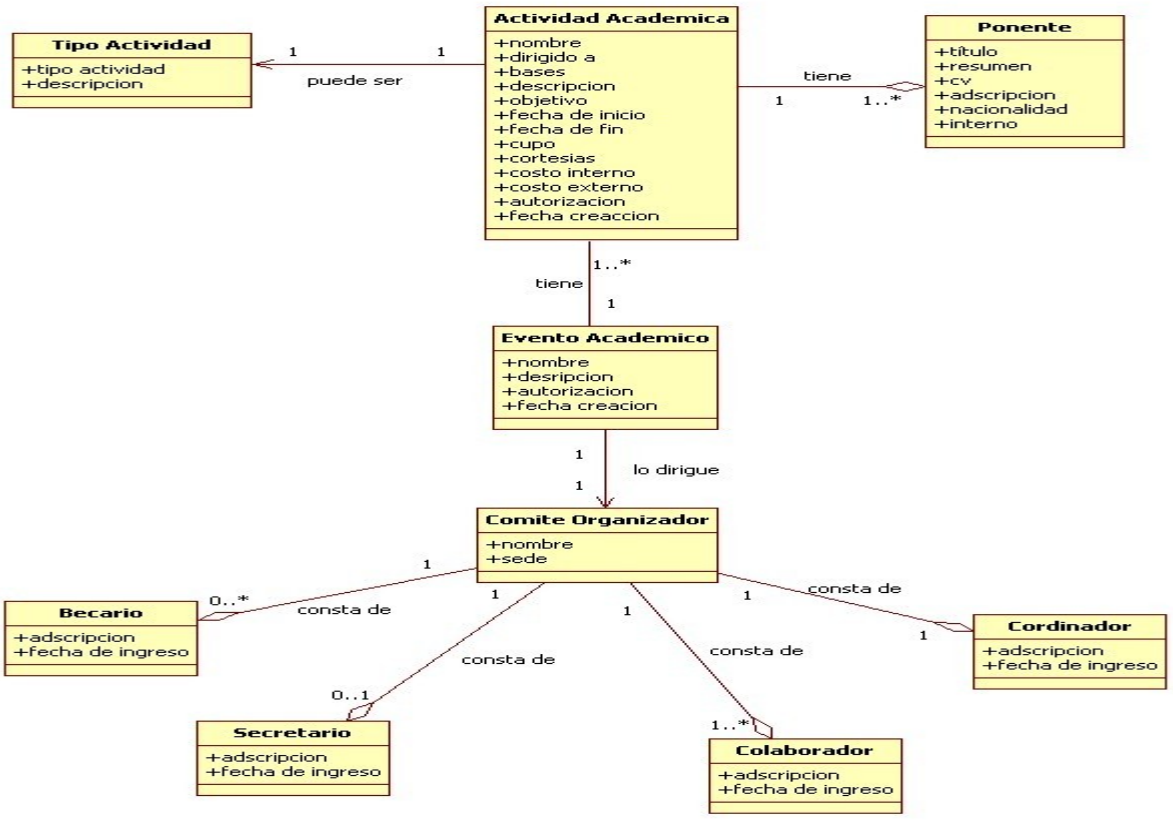


Figura 3. Diseño del Modelo de Dominio

Tanto el diagrama de casos de uso general como el modelo de dominio, se elaboraron en la fase de inicio que marca el Proceso Unificado.

4. Diagramas de secuencia

A continuación se explican los diagramas de secuencia correspondientes al caso de uso actividades académicas (sección 2.3).

La figura 4 muestra la operación de `consultar` por parte del administrador, seleccionando el `ID` (clave) de la actividad del evento al que corresponda.

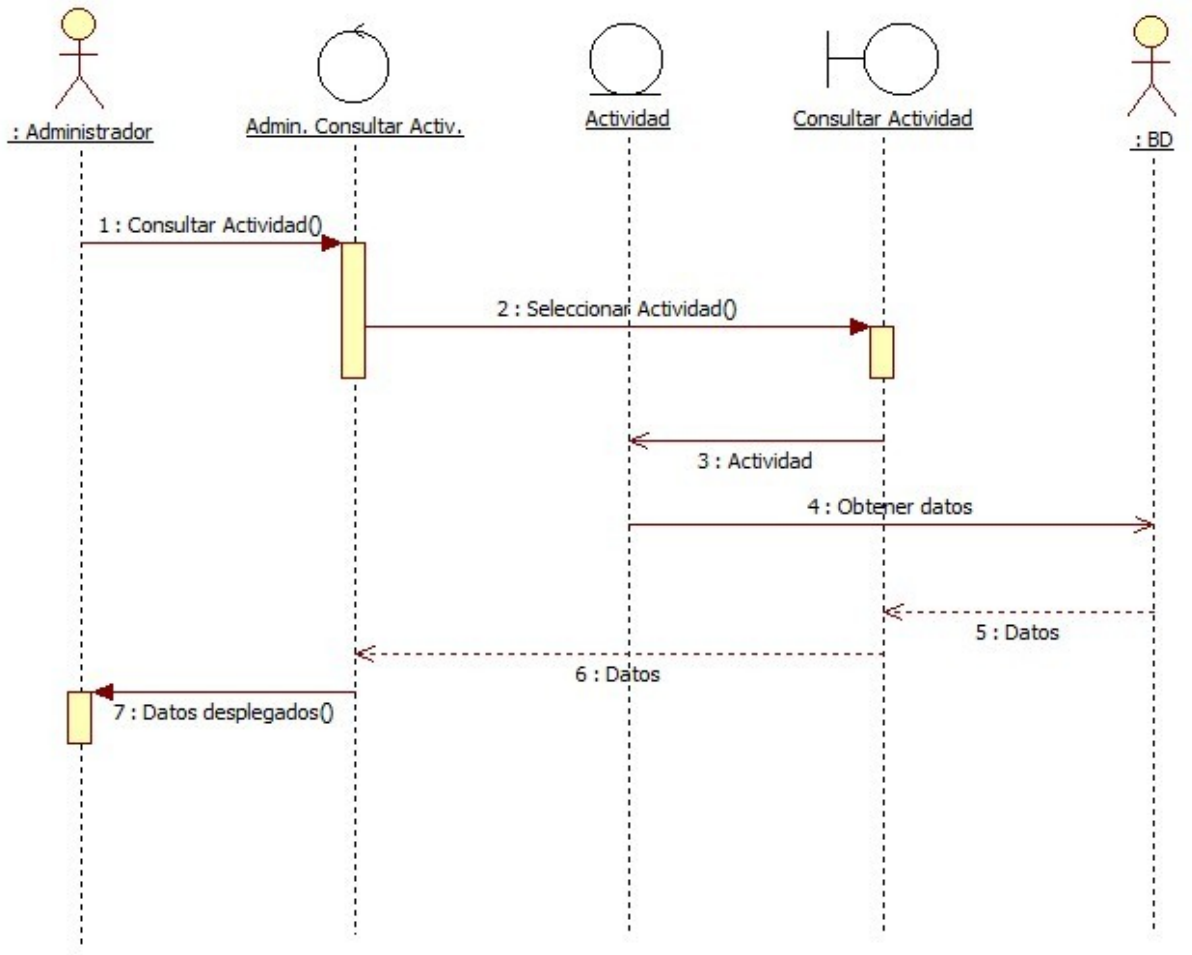


Figura 4. Diagrama de secuencia Consultar Actividad

La figura 5 muestra la secuencia de pasos que se lleva a cabo para `crear` una nueva actividad académica. El administrador ingresará los datos correspondientes. Si los datos son correctos, se crea un nuevo objeto actividad el cual se guardará en la base de datos; de lo contrario, se le notificará al usuario que los datos están mal.

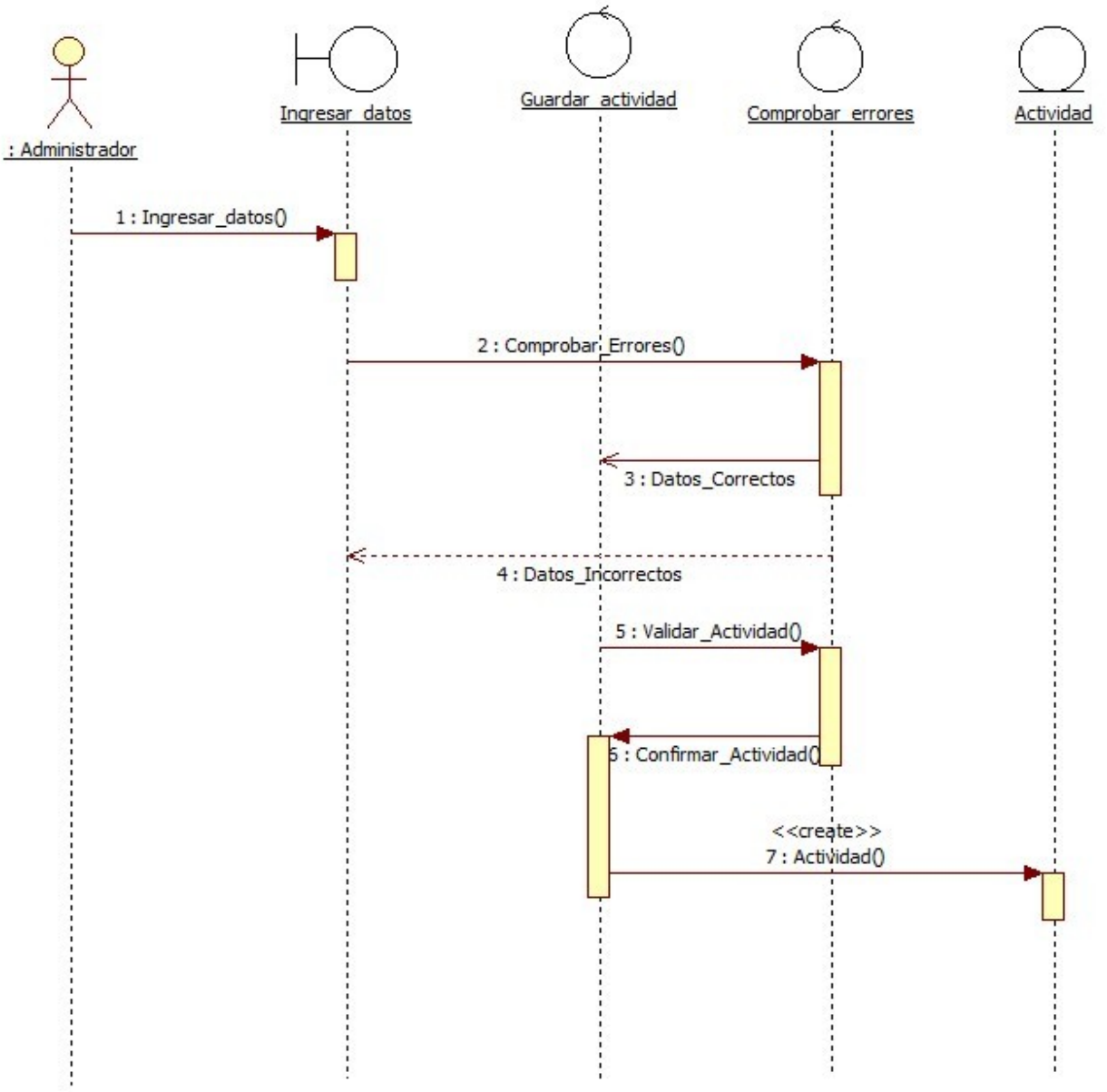


Figura 5. Diagrama de secuencia Crear Actividad

La figura 6 muestra el diagrama de secuencia para modificar una actividad académica. Se iniciará con la `consulta` de la actividad sobre la pantalla; después, el administrador realizará la o las modificaciones pertinentes; al seccionar la opción `modificar` se actualizarán los datos; finalmente, la actividad será mostrada en la pantalla con la información modificada.

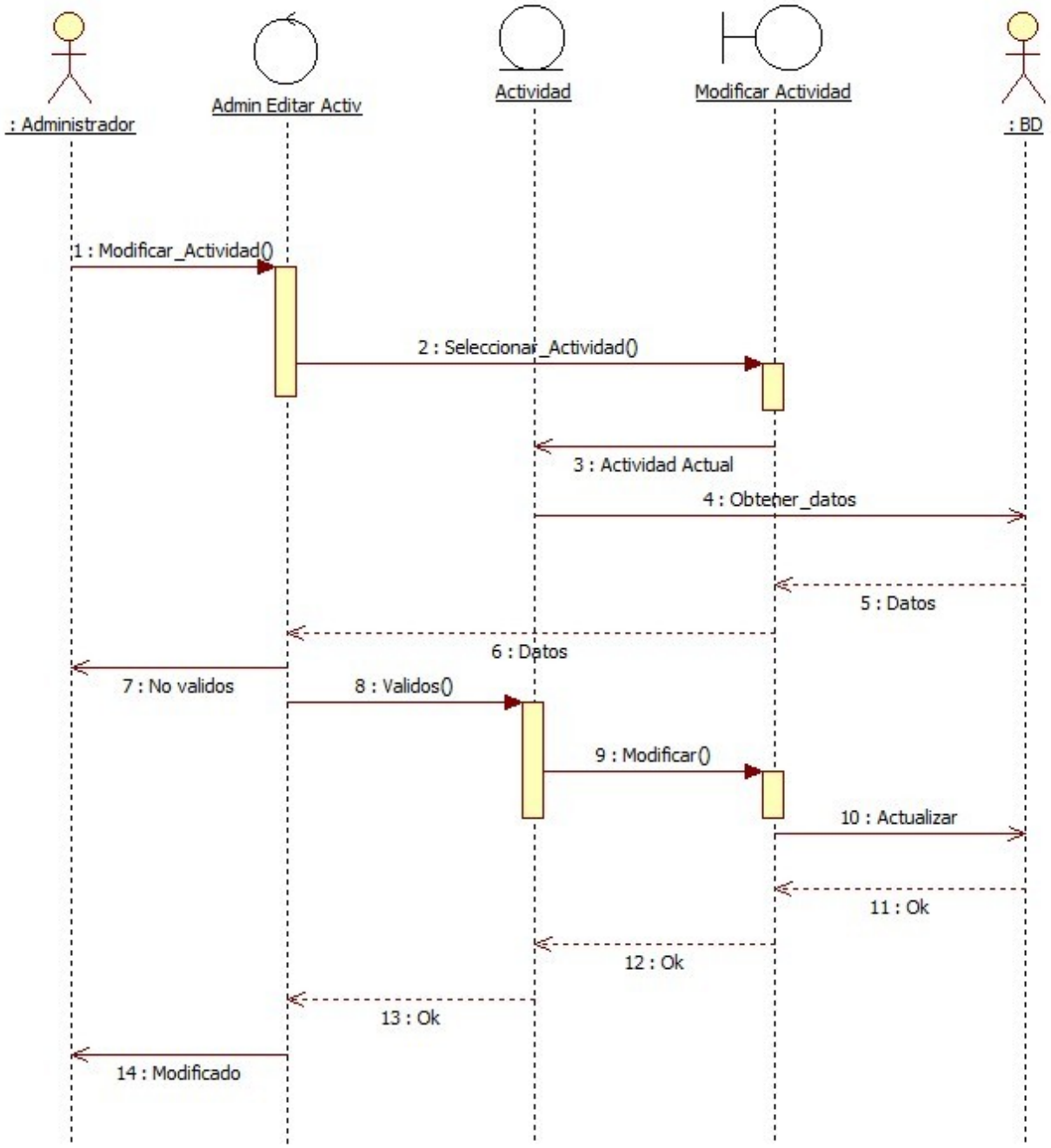


Figura 6. Diagrama de secuencia Modificar Actividad

La figura 7 muestra el diagrama de secuencia para `borrar` una actividad académica. El administrador seleccionará `borrar` una actividad académica; el sistema comprobará la actividad y preguntará al administrador si desea eliminar la actividad actual, si la respuesta es afirmativa, se le notificará al administrador y la actividad quedará eliminada del evento al que pertenecía, así como del sistema.

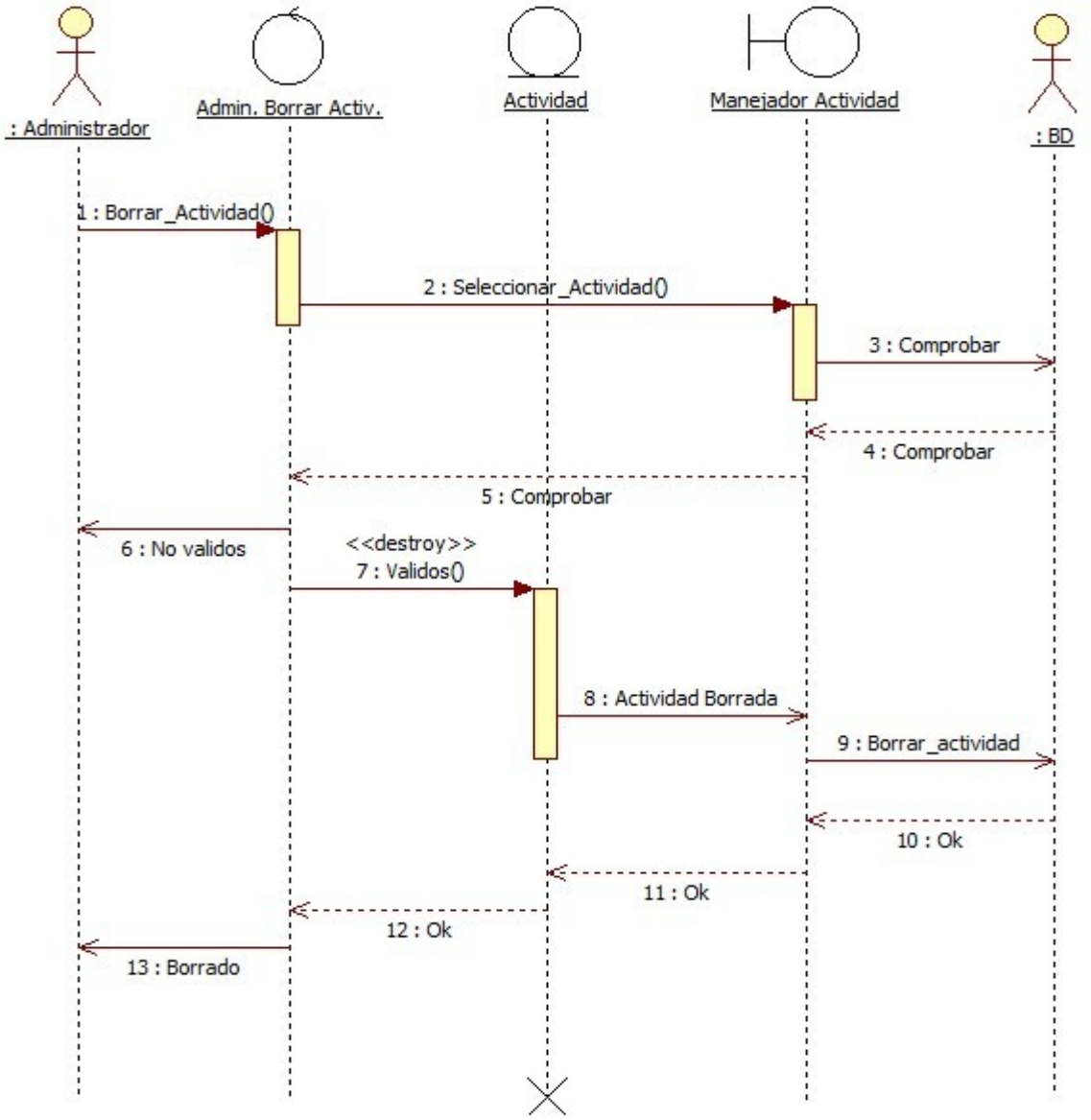


Figura 7. Diagrama de secuencia Borrar Actividad

Es importante señalar que los diagramas de secuencia para eventos académicos y comité organizador siguen el mismo patrón de secuencia en cuanto a la creación, modificación, consulta y borrado que hemos presentado para gestión de actividades; por lo tanto, no los esquematizaremos.

5. Base de datos

El esquema de la base de datos que el sistema utiliza fue elaborada por el alumno Guillermo Monroy Rodríguez, quien como parte de su reporte de servicio social, se encargó de generar dicha estructura. La figura 8 muestra las tablas que requerimos emplear para los módulos que se han desarrollado en este proyecto terminal.

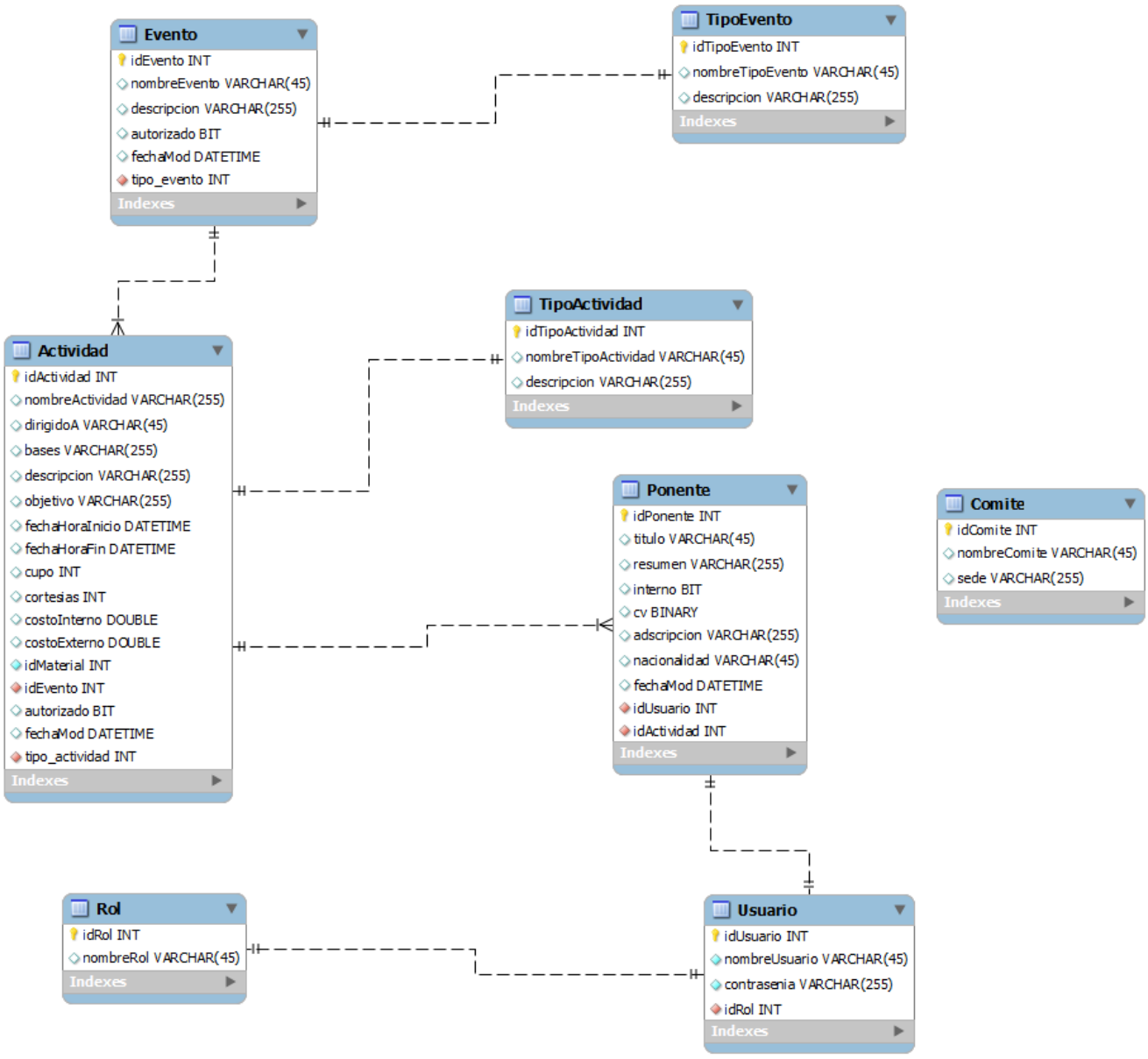


Figura 8.Diagrama Entidad-Relación Base de Datos SIEA

6. Diagrama de clases

6.1. Clases Action

Las clases encargadas de llevar a cabo la lógica de negocio y responder a una petición estarán contenidas en la carpeta *mx.uam.azc.cbi.action* (figura 9), dichas clases heredan de la clase base `com.opensymphony.xwork2.ActionSupport`, la cual usa el nombre de la clase con sufijo `Action` (por convención) para llevar a cabo el mapeo de peticiones URL y así saber cual clase `Action` debe ser llamada para dicha petición.

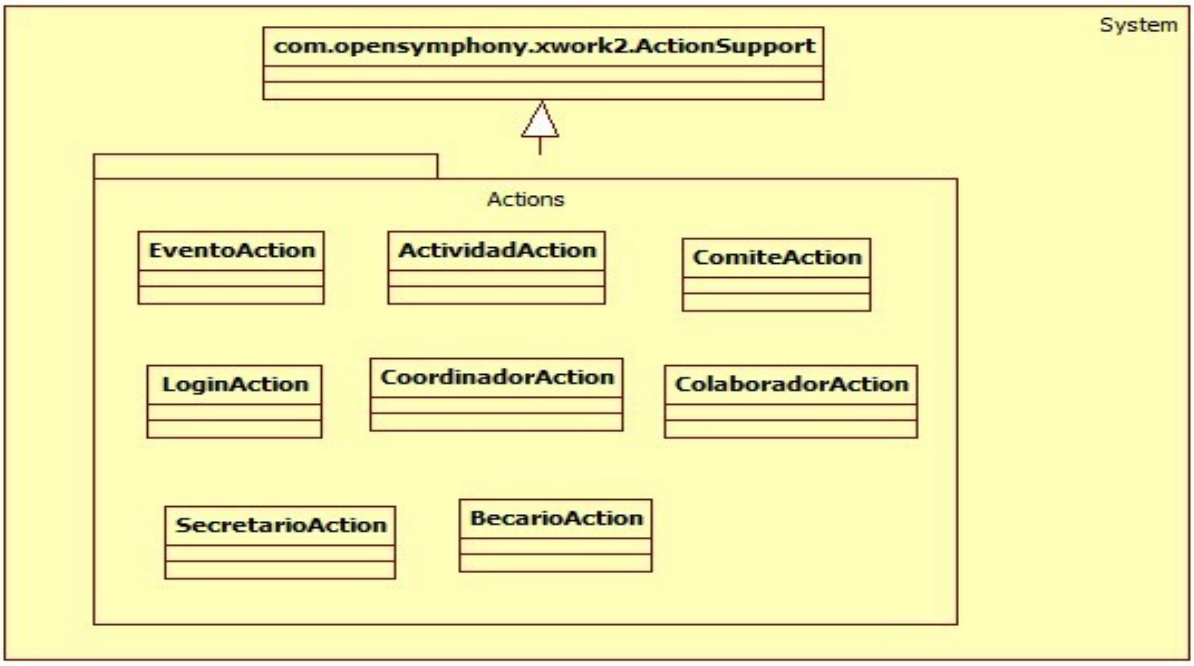


Figura 9. Diagrama de clases carpeta *mx.uam.azc.cbi.actions*

6.2. Clases Lógica

En el paquete *mx.uam.azc.cbi.logic*, se definen las interfaces que funcionan como administradores de la parte transaccional de los datos en el sistema. La figura 10 muestra el diagrama de clases que componen el paquete de la lógica de negocio junto con las clases que implementan estas interfaces.

En la lógica de negocios se definen métodos que administran la conexión a bases de datos, las transacciones, los mapeos de entidades relacionadas con la base de datos y servicios. Se usan interfaces como aislantes de la capa de presentación hacia la lógica del negocio, estas interfaces integran la capa de presentación a la capa de modelo.

Las interfaces administran responsabilidades que cubren las necesidades que el sistema necesita; ésto es, deben cubrir necesidades tales como: ingresar un evento, modificarlo, borrarlo y consultarlo. Estas interfaces manejan los datos fundamentales que conforman al módulo que se quiere generar.

Los clases de servicios administrativos declaradas en el paquete *mx.uam.azc.cbi.logica* son utilizadas por las clases de implementación contenidas en el paquete *mx.uam.azc.cbi.logica.local*, estas clases heredan todas las propiedades y métodos definidos en las clases administrativas, pero en la implementación se define la funcionalidad de cada uno de esos métodos.

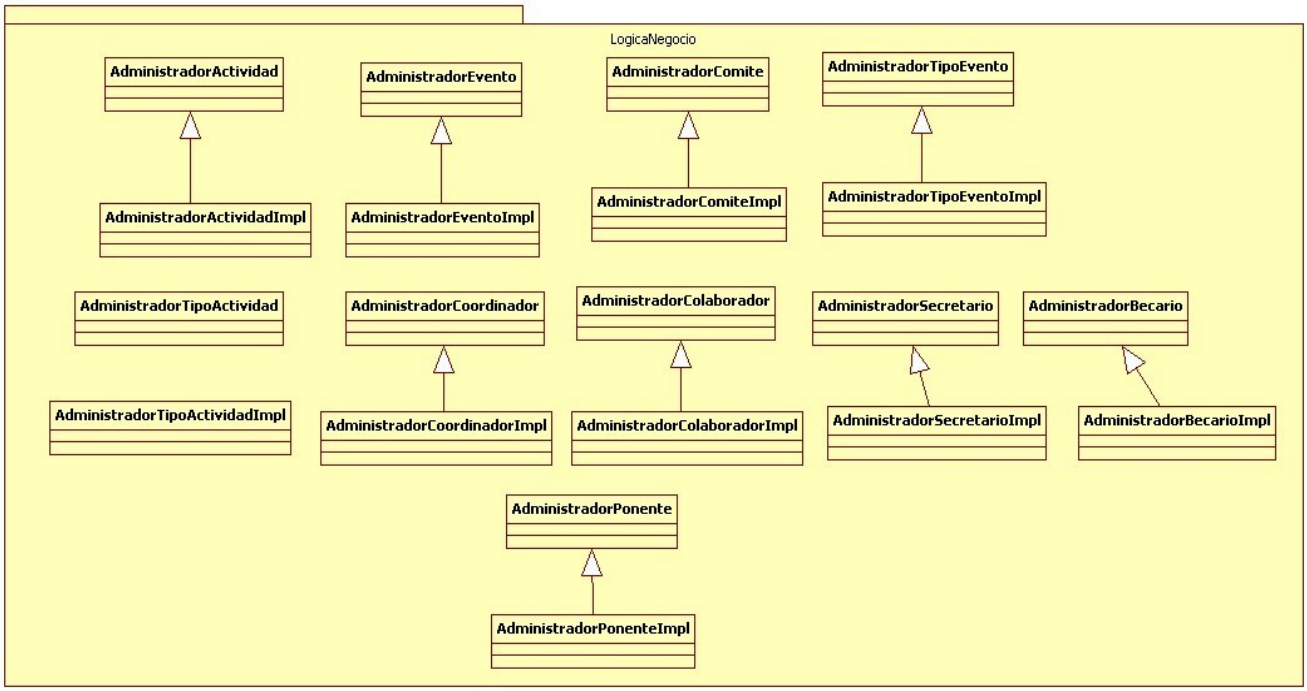


Figura 10. Diagrama de clases de Lógica de Negocio

6.3. Diagrama de clases datos

En la capa de datos se gestionan las clases que se utilizarán para el mapeo de los datos; estas clases mejoran el manejo de los datos para las clases *`Action`* y para las *`Interfaces`*; estas últimas ejecutan las operaciones necesarias que se pueden realizar en la base de datos, tales como inserción, borrado, actualización y consulta de los datos. La figura 11 muestra el diagrama de clases que componen el paquete de la capa de datos.

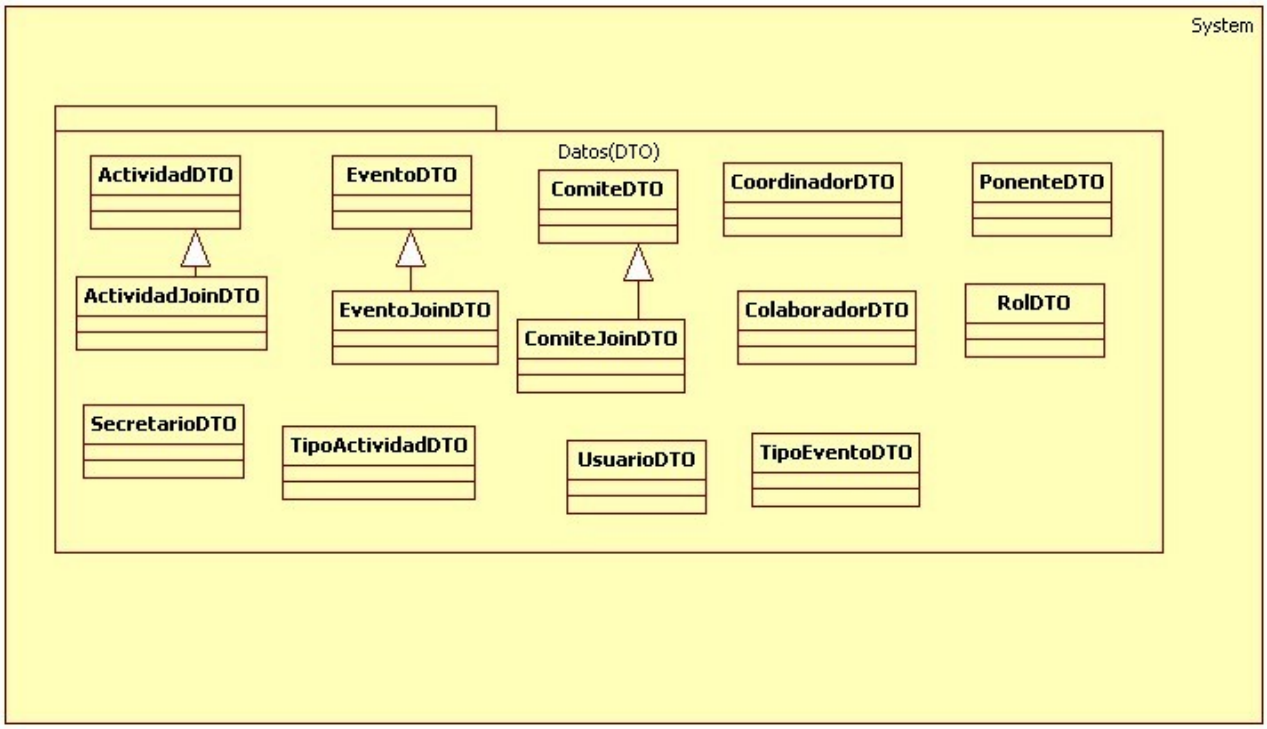


Figura 11. Diagrama de clases capa de datos.

Para el manejo de los datos en el sistema, la aplicación usa una relación con objetos aplicando el mapeo objeto-relación (OR Mapping - `Object-Relational Mapping`) el cual es una técnica utilizada en aplicaciones java.

Los objetos DTO (figura 11) actúan como entidades JPA que utilizan métodos `get()` y `set()` para el manejo de los datos utilizando un mecanismo de persistencia de mapeo orientado a los campos de la base de datos relacionado con las propiedades de los objetos DTO.

7. Arquitectura del sistema

La figura 12 muestra los elementos del "*Sistema de Gestión de Eventos Académicos*", definidos a partir del patrón de diseño MVC *(Modelo-Vista-Controlador); el* web *Framework Apache Struts2* nos permite implementar el modelo MVC de nuestra aplicación, así como definir el dominio donde ella residirá. En el bloque Controlador, el sistema atiende las peticiones del cliente por medio de interceptores y acciones. El bloque Modelo, contiene la lógica del negocio constituida por los datos referentes a los eventos, actividades y el comité organizador. El bloque Vista plasma los resultados generados por la interacción del Controlador y el Modelo en páginas jsp[2] y se presenta al cliente como resultado de la petición que realizó.

2 JavaServer Pages.

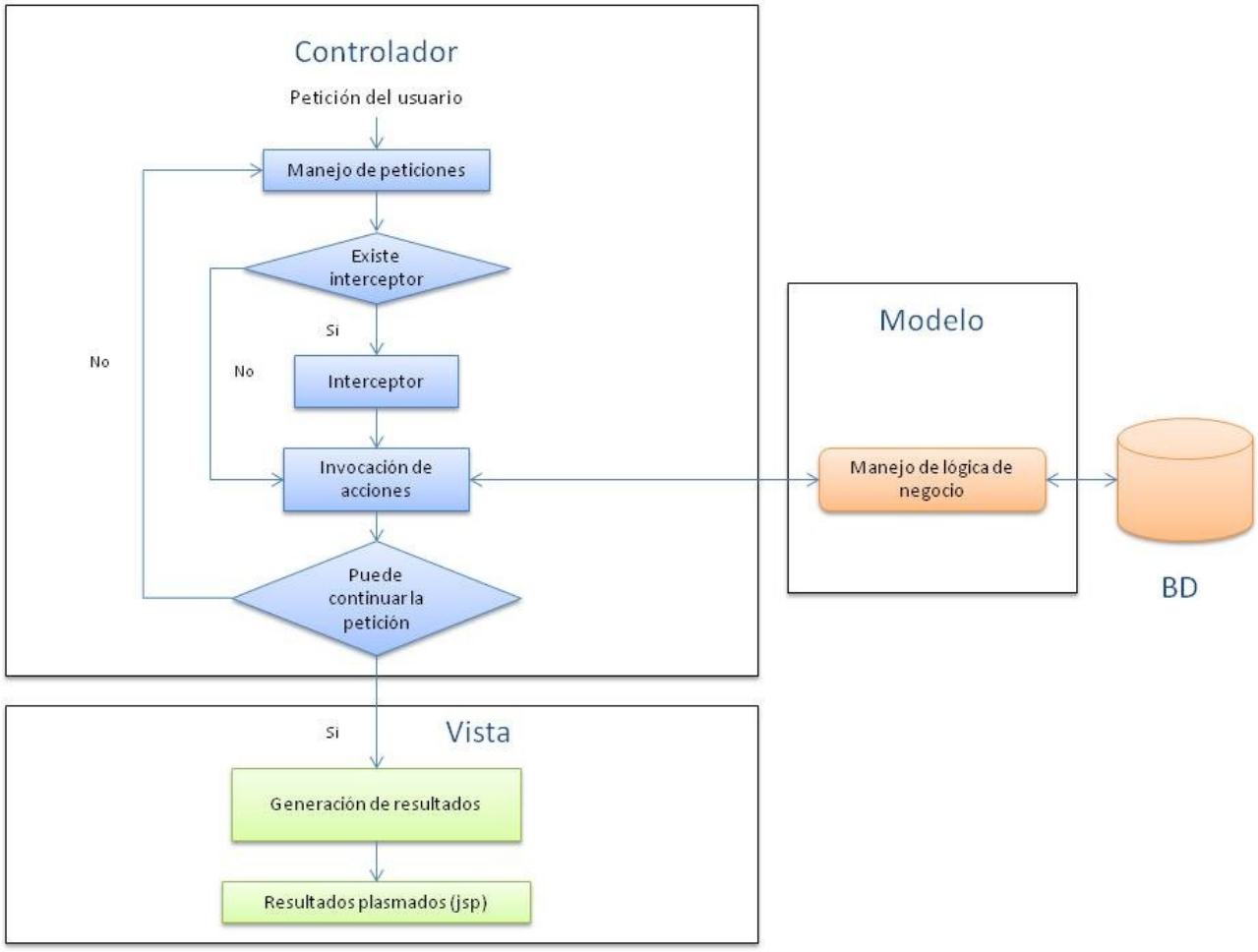


Figura 12. Arquitectura del sistema SIEA

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

Proyecto Terminal:
SISTEMA DE GESTION DE EVENTOS ACADÉMICOS

MANUAL DE USUARIO

Alumno:
Manuel Alejandro Cobos Lomelí
Matrícula: 205305635

Asesor:
Dra. María Lizbeth Gallardo López
Profesor investigador
Departamento de Sistemas

INTRODUCCIÓN

El presente documento esquematiza la utilización del SIstema de Gestión de Eventos Académicos (SGEA) por el usuario primario. Se mencionarán las diferentes pantallas y los resultados que se generan entre cada petición y acción que se lleve a cabo en el sistema al momento de interactuar con el mismo.

1. Ejecución de la aplicación

Para comenzar el usuario teclea la siguiente dirección en el navegador web de su preferencia: http://localhost:8080/SIEA/sistema/login_forma_login.action (donde localhost se refiere a la dirección ip del servidor). La figura 1 muestra la página de inicio del sistema, en esta página se debe ingresar un usuario y contraseña para pasar al menú correspondiente al rol del usario que quiere acceder.

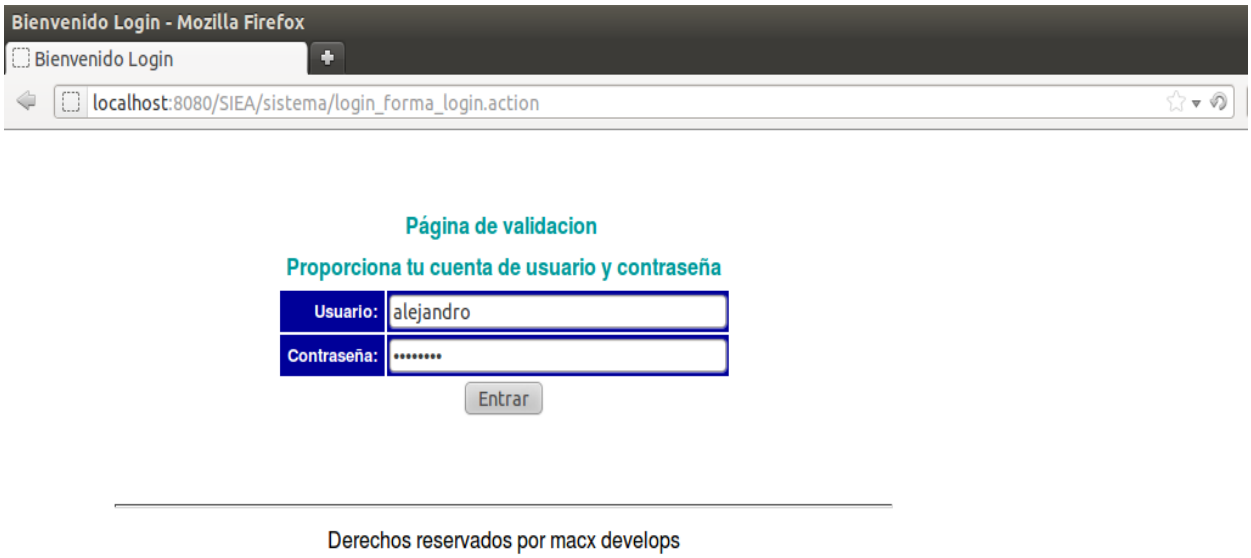


Figura 1. Página de inicio

Como resultado de la validación de un usuario administrador se desplegará el menú principal (ver figura 2). En la página desplegada el usuario cuenta con las opciones para visualizar los “`eventos académicos`”, el “`comité organizador`”, "`registro`" y “`salir`” del sistema. Cabe mencionar que la opción de "`registro`" aun no tiene funcionalidad

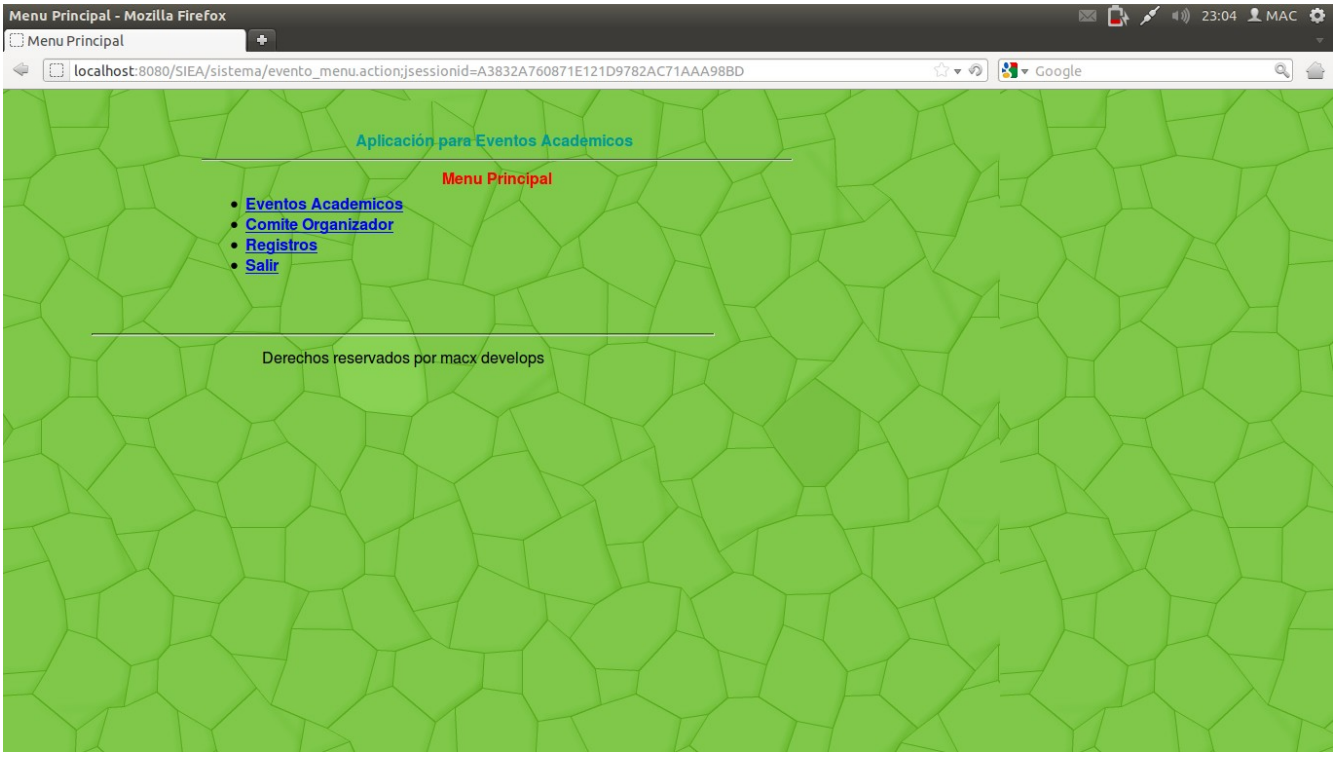


Figura 2. Página de menú principal

Al ser seleccionada la opción de "Eventos Académicos" (ver figura 3) el usuario podrá visualizar los eventos académicos que han sido generados en el sistema y que están vigentes.

Figura 3. Desplegado de eventos académicos

El usuario tendrá las opciones de consultar, borrar o insertar un evento académico. La opción de consultar el evento se lleva a cabo seleccionando el `Id` del evento académico que se quiere consultar (ver figura 4), la visualización muestra las características particulares del evento académico, como son: `nombre`, `fecha de inicio`, `tipo de evento`, `autorización`, una breve `descripción` y las `actividades` (si es que las hay) que contiene dicho evento.

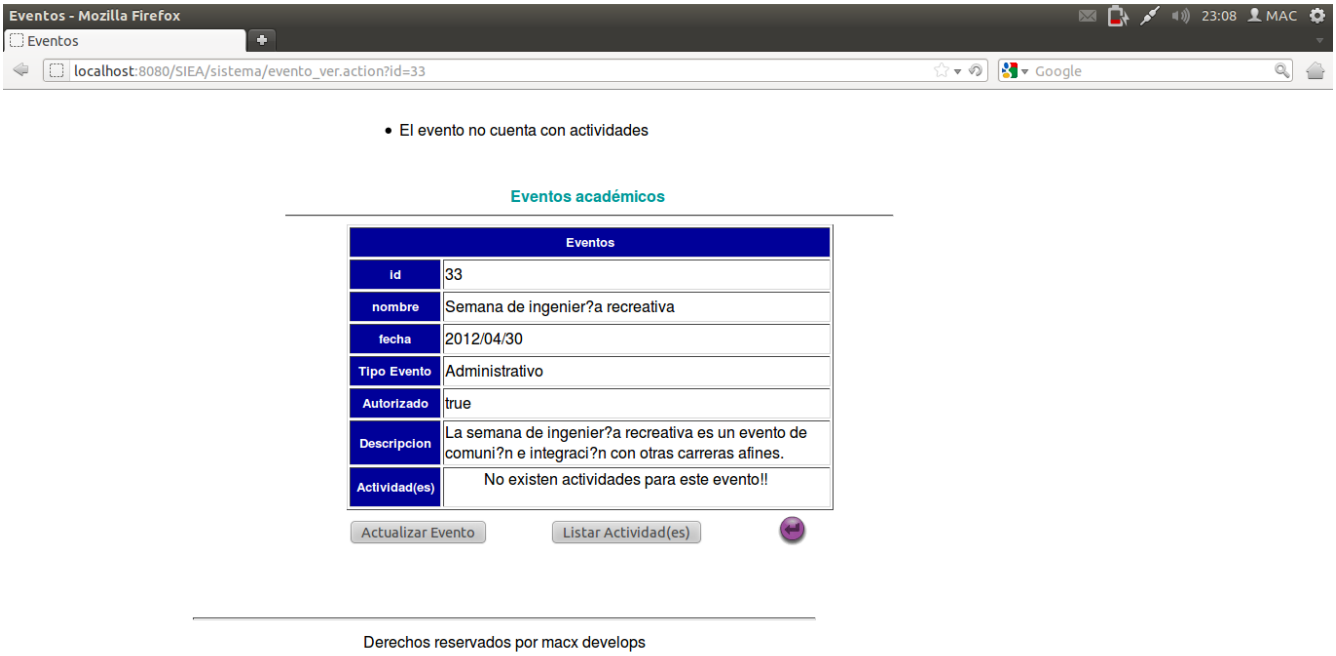


Figura 4. Desplegado de un evento académico

Observe que esta misma pantalla contiene, a su vez, el botón de "`Actualizar Evento`" y el botón "`Listar Actividades`". El primer botón nos permite modificar el evento seleccionado (ver figura 5); y el segundo botón nos lleva al listado de actividades que se llevarán a cabo en el evento en cuestión.

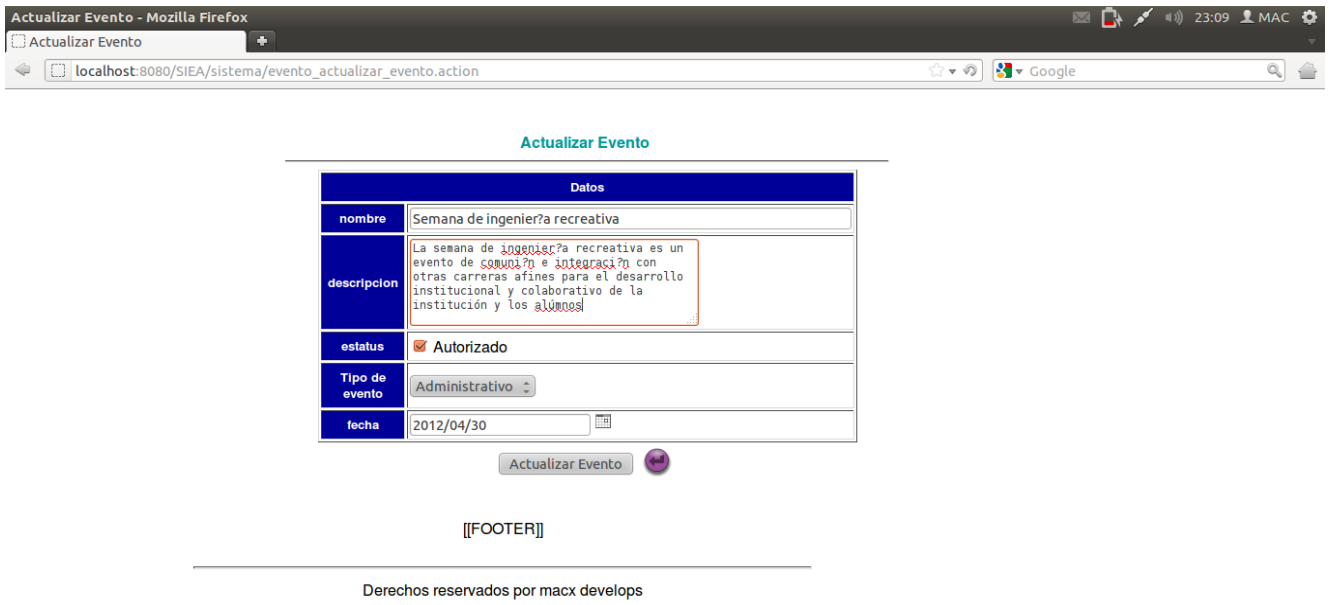


Figura 5. Modificación de un evento académico.

Al dar click en el botón de insertar nuevo evento (ver figura 3), el sistema desplegara la página correspondiente (ver figura 6) que contendrá un formulario para ingresar un evento académico con las siguientes características: `nombre`, `descripción`, `estatus`, `tipo de evento` y `fecha`. Como resultado de la inserción se desplegará el nuevo evento generado (ver figura 7).

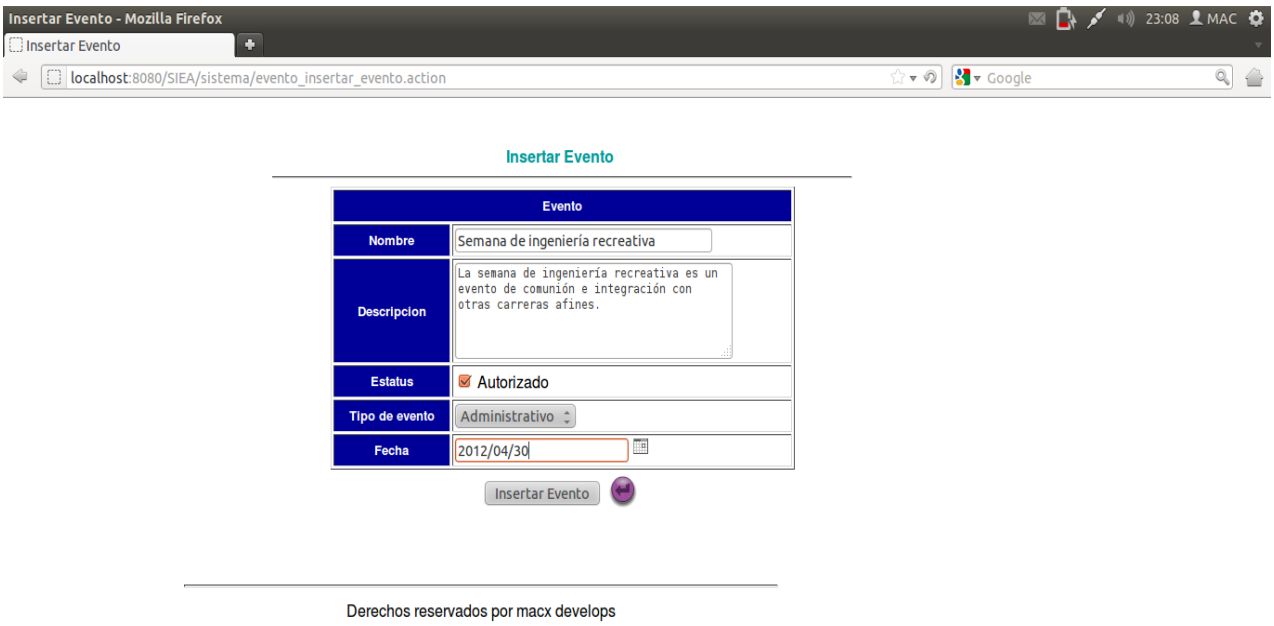


Figura 6. Formulario de nuevo evento academico

Figura 7. Nuevo evento “Semana de ingeniería recreativa”

Por último, para poder borrar algún evento académico se debera dar click en la imagen de “x” (ver figura 8), esto conlleva a que por medio del `id` del evento en cuestión se borre por completo dicho evento y el resultado se reflejará en la pantalla que despliega los eventos academicos activos (ver figura 9).

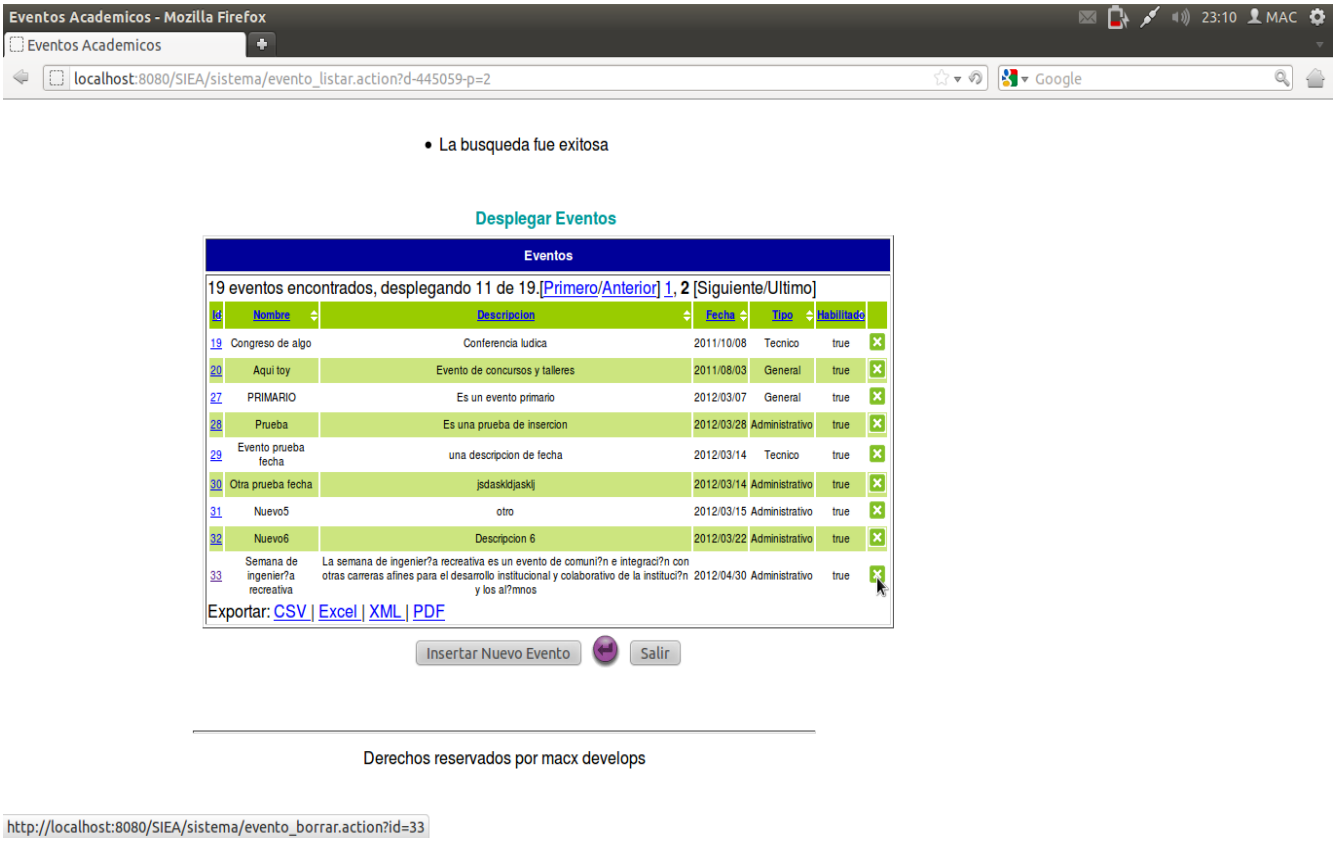


Figura 8. Borrado de eventos académicos

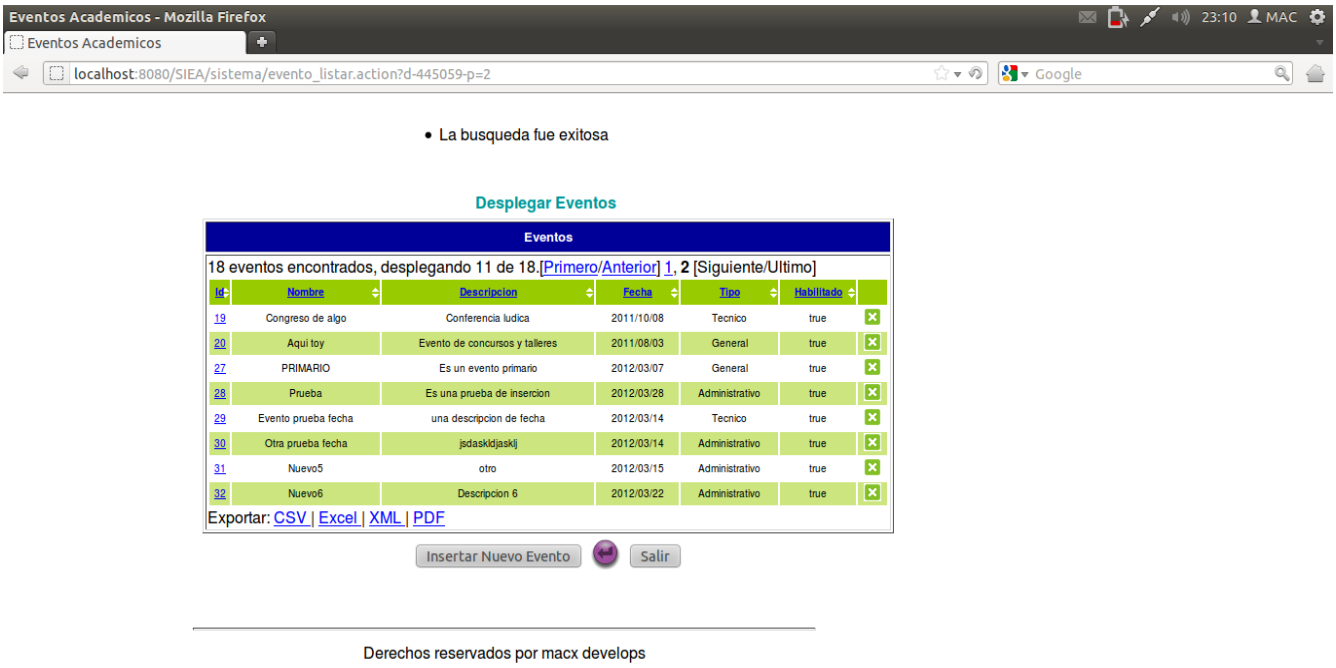


Figura 9. Listado de eventos academicos

Para visualizar una actividad académica el usuario en primera instancia, tendrá que consultar un evento académico en particular (ver figura 4) ya que las actividades están siempre relacionadas con un evento académico.

Si en el evento no hay actividades el usuario tendrá la posibilidad de generar una nueva actividad la cual corresponderá al evento académico en particular (ver figura 10).

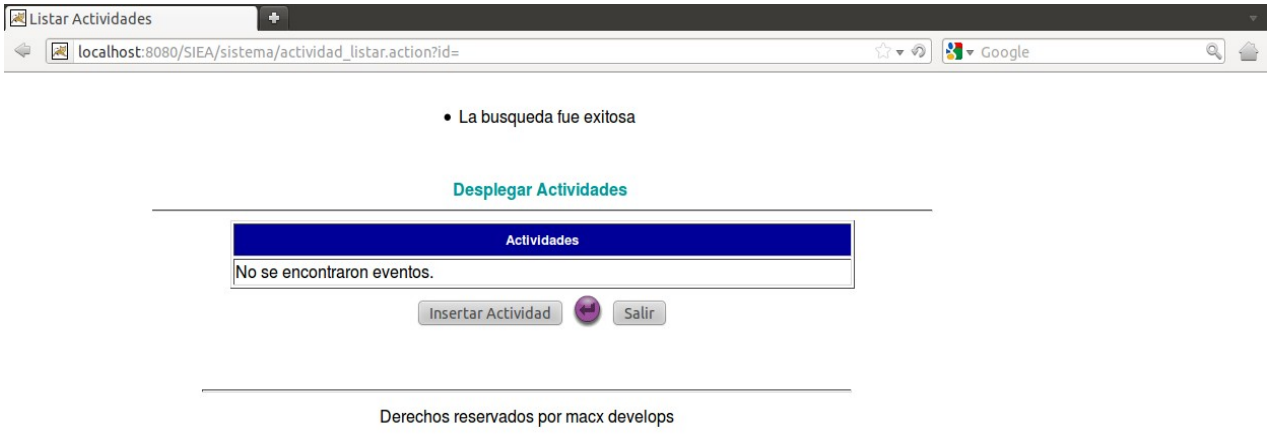


Figura 10. Desplegado de actividades académicas.

Al dar click en "`Insertar Actividad`" el usuario podrá ingresar los datos correspondientes de la nueva actividad (ver figura 11).

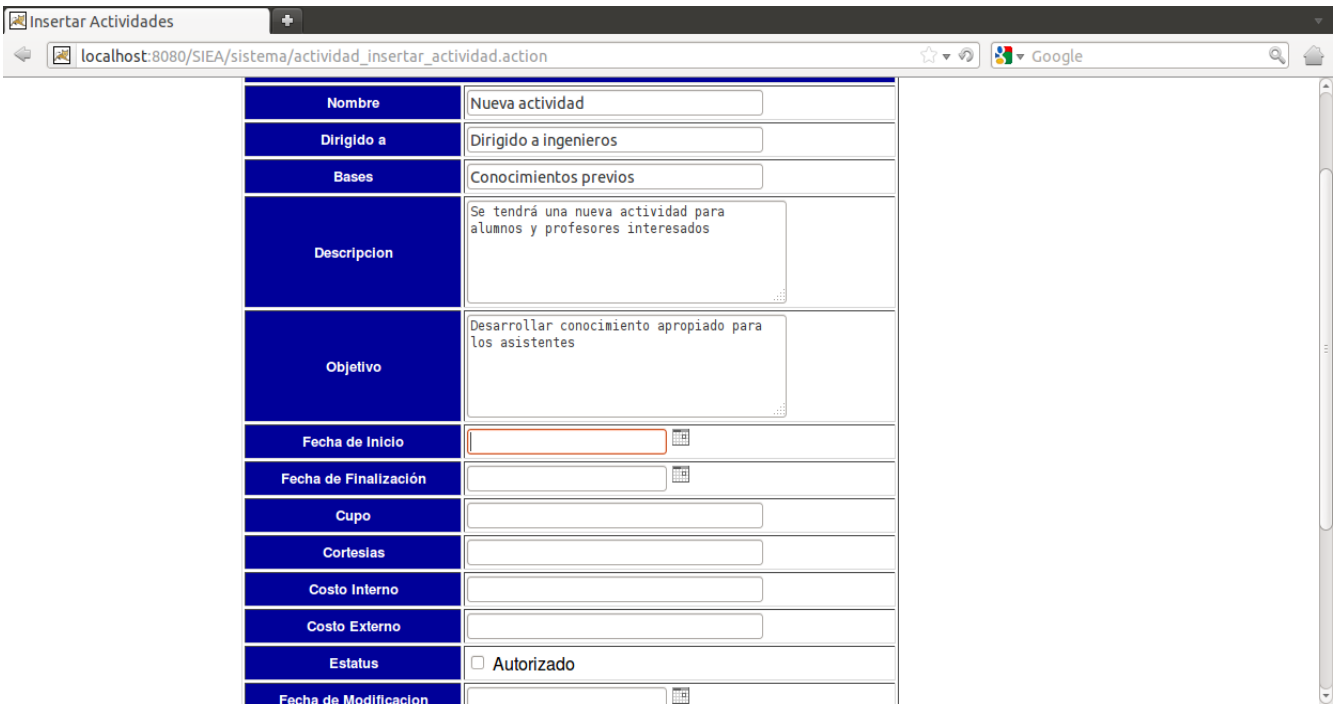


Figura 11. Inserción de actividad académica

Al termino del ingreso de los datos el usuario visualizará la nueva actividad y los datos que fueron ingresados (ver figura 12).

Figura 12. Desplegado nueva actividad académica

Para poder actualizar la actividad, el usuario seleccionará el id de la actividad y se desplegará la actividad a modificar. Se ingresará algún dato que falte o se modificará algún dato que ya este plasmado, y para hacer la modificación se dará

click en el botón de actualizar (ver figura 13) dando como resultado la visualización de la actividad ya con los cambios reflejados al instante (ver figura 14).

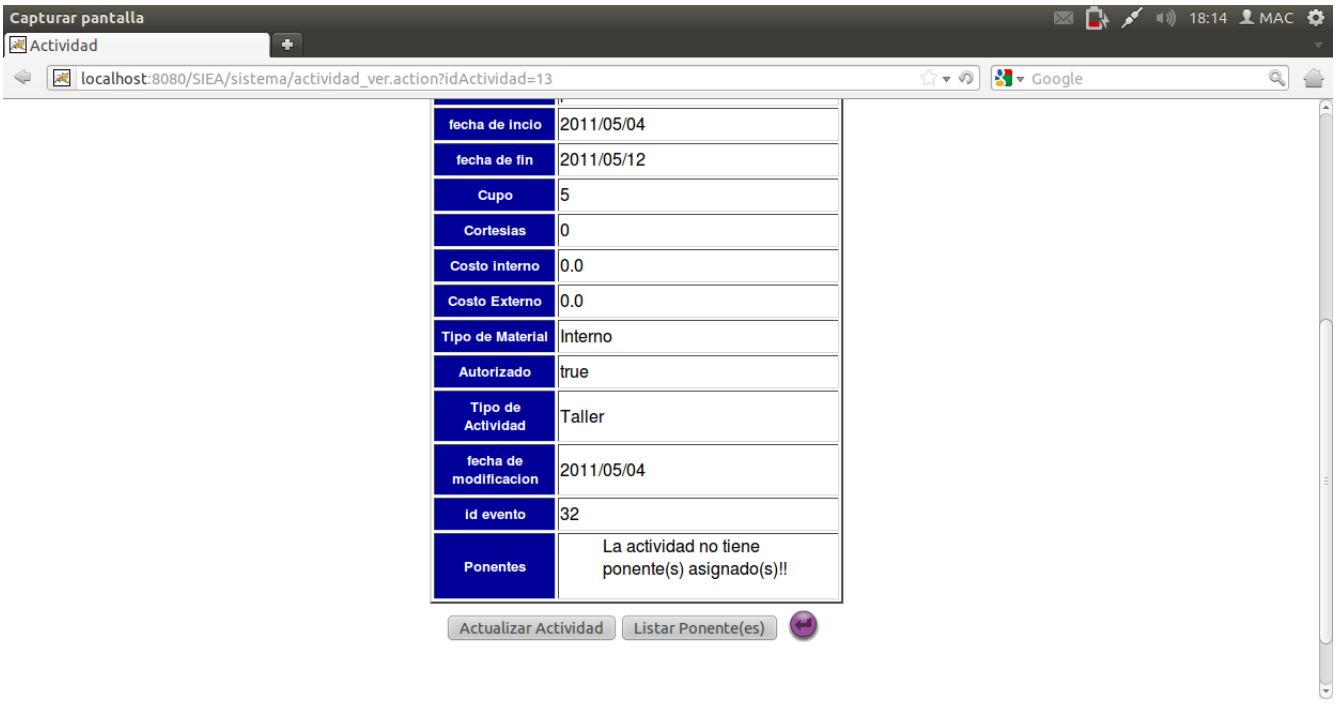


Figura 13. Actualizar actividad académica

Figura 14. Actividad academica actualizada

Por último el usuario tendrá la facultad de borrar una actividad dando click en la imagen de “x”, esto realizará el borrado de la actividad (ver figura 10), dejando

de corresponder a el evento en el que se quiso ingresar dicha actividad (ver figura 4).

Para el modulo de comité organizador se llevan a cabo las mismas acciones y la secuencia correspondiente, esto es, se puede consultar, crear, actualizar y borrar un comité organizador, todo esto con sus respectivas propiedades.

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

Proyecto Terminal:
SISTEMA DE GESTION DE EVENTOS ACADÉMICOS

MANUAL TÉCNICO

Alumno:
Manuel Alejandro Cobos Lomelí
Matrícula: 205305635

Asesor:
Dra. María Lizbeth Gallardo López
Profesor investigador
Departamento de Sistemas

Introducción

El presente documento presenta las especificaciones técnicas del SIstema de Gestión de Eventos Académicos (SGEA), a saber: hardware, el software y la instalación.

Hardware

La aplicación SIEA se montó sobre una máquina virtual Oracle VM VirtualBox con sistema operativo Ubuntu 11.10 con las siguientes caracteristicas:

1. Procesador Intel Core i5-2410M a 2.30GHz.
2. Sistema operativo Ubuntu 11.10 de 32 bits.
3. Disco duro virtual de 8 GB.
4. Memoria base de 1024 MB.

La aplicación se montará en un servidor HP proliant G6 para que pueda ser usada internamente en la unidad Azcapotzalco; y con sistema operativo Solaris 10

Software

La aplicación se desarrolló bajo el siguiente software:

1 Manejador de base de datos: pgAdminIII, PostgreSQL Tools.
2 Lenguaje de programación: Java Enterprise Edition.
  2.1 IDE: Eclipse Ganymede.
  2.2 Framework: Struts2, Spring y Hibernate 3.
3 Sistema operativo: Windows 7 Professional.

Estructura

La estrutura de la aplicación está desarrollada dentro del paquete SIEA el cual corresponde al nombre del proyecto requerido por el IDE eclipse. El empaquetamiento base mx.uam.azc.cbi funge como el dominio de la aplicación y se divide de la siguiente forma:

1. mx.uam.azc.cbi.actions: En este paquete se encuentran las clases que funcionan como controladores de la lógica del negocio y son las acciones que se llevan a cabo al momento de ejecutar una petición en el sistema, estas clases utilizan el sufijo Action como convención para llevar a cabo el mapeo de peticiones URL y así saber que clase `Action` debe ser llamada para dicha petición. Las clases controladoras se corresponden a los módulos: ActividadAction, EventoAction, ComiteAction, ColaboradorAction, BecarioAction, PonenteAction, SecreatrioAction, CoordinadorAction.
2. mx.uam.azc.cbi.datos: En este paquete se encuentran las clases que se utilizan para la persistencia de los datos; estas clases se utilizan para hacer el mapeo de la base de datos para crear objetos entidad, reconocidas por el

sufijo DTO, algunas de ellas son: ActividadJoinDTO, ComiteDTO, EventoJoinDTO, TipoActividadDTO y TipoEventoDTO.

3. mx.uam.azc.cbi.logica: En este paquete se definen las interfaces que contienen los métodos que llevan a cabo las transacciones requeridas en una acción particular, por ejemplo: consultas, inserciones, modificaciones o eliminaciones de los datos que pertenecen a los objetos entity.
4. mx.uam.azc.cbi.logica.local: En este paquete se implementaron las interfaces definidas en el paquete mx.uam.azc.cbi.logica, estas clases fueron sobreescritas de manera que pudieran utilizar a las clases definidas en el paquete mx.uam.azc.cbi.datos y hacer las transacciones pertinentes utilizando las funcionalidades de Hibernate v3, a saber: consultas, inserciones, actualizaciones y borrado de datos.
5. mx.uam.azc.cbi.test: Este paquete está constituido de la clase Test donde se llevaron a cabo pruebas de: i) conexión a la base de datos, ii) inserción, consulta, borrado y actualización de datos en la BD, y iii) comunicación con las clases que implementan las interfaces del paquete mx.uam.azc.cbi. logica.
6. mx.uam.azc.cbi: En este paquete se define la clase InicializadorCatalogos la cual implementa la interfaz ServletContextListener; esta clase contiene el contexto de la aplicación.

Para la configuración de la conexión con la base de datos, los servicios de gestión de las transacciones y lógica de negocios, así como la definición de los jsp que se utilizaron para plasmar los resultados de las acciones se definen a continuación.

1. sistema.xml: Este archivo de configuración contiene la declaración de los jsp en donde se plasman los datos generados cuando una o varias acciones son generadas a partir de una petición de un usuario sobre el sistema.
2. spring-services.xml: En este archivo se configura la lógica de negocios en base a las clases que implementan las interfaces contenidas en el paquete mx.uam.azc.cbi.logica. La declaración de los servicios (componentes) de gestión de actividades, eventos y comité organizador se define por medio de las etiquetas `<bean>`.
3. spring-dev.xml: En este archivo se encuentra configurada la información del ambiente de desarrollo y la declaracion del componente dataSource como servicio de conexión a BD. También se declara el componente para el manejo de transacciones en java, así como el manejo de hibernate tanto para sesiones en la base de datos como para definir las clases que servirán para el mapeo de las tablas de la BD hacia el sistema.
4. queries.hbm.xml: En este archivo se definen los querys que usarán los servicios de gestión al momento de hacer una transacción con la base de datos; los querys se encuentra contenido dentro de las etiquetas <querys> las cuales son funcionalidades del framewok Hibernate 3.

La aplicación cuenta con otro paquete, el cual tiene como propósito contener las páginas Jsp que sirven para la vista en la aplicación. Los paquetes se encuentran en la carpeta WebContent->WebInf->paginas y se enlistan a continuación:

1. Actividad
2. Becario
3. Colaborador
4. Comite
5. Coordinador
6. Evento
7. Login
8. Ponente
9. Secretario

Dentro de WebContent->WebInf se encuentra el archivo de configuración web.xml que define el listener y la ubicacion del contexto de los servicios de gestión de la lógica del negocio, también se definen los filtros de peticiones del framework Struts.

Proceso de instalación

Para instalar la aplicación dentro del IDE de programación Eclipse Ganymede se llevan a cabo los siguientes pasos:

1. Seleccionar el paquete SIEA->Export->Export. En la ventana (Figura 1) se mostrara el tipo de archivo al cual se exportará la aplicación.

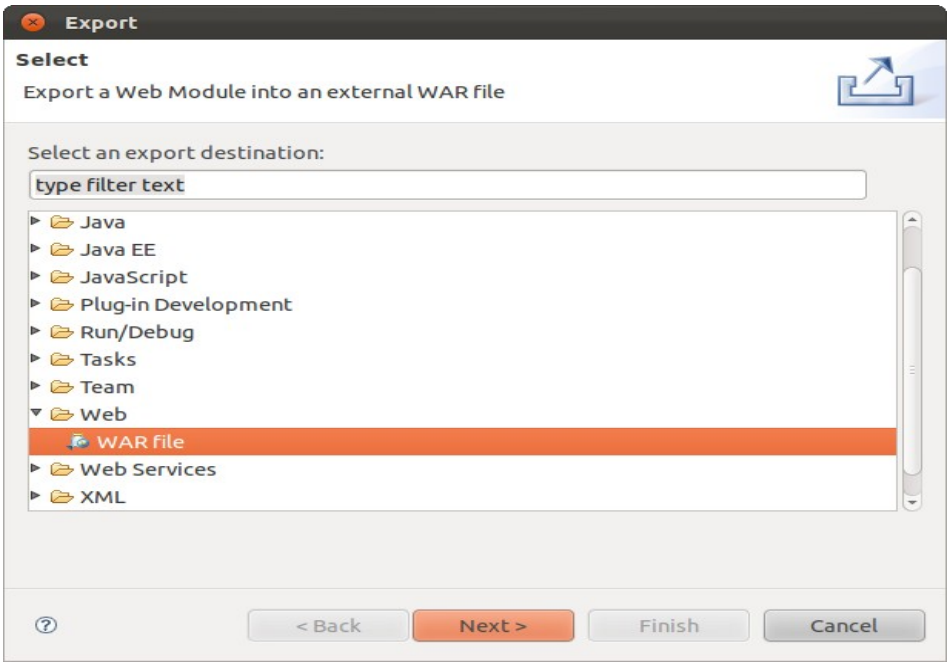


Figura 1

2. Se seleccionará la ubicación en donde estará contenida la aplicación pulsando Browse... de la opción Destination (Figura 2).

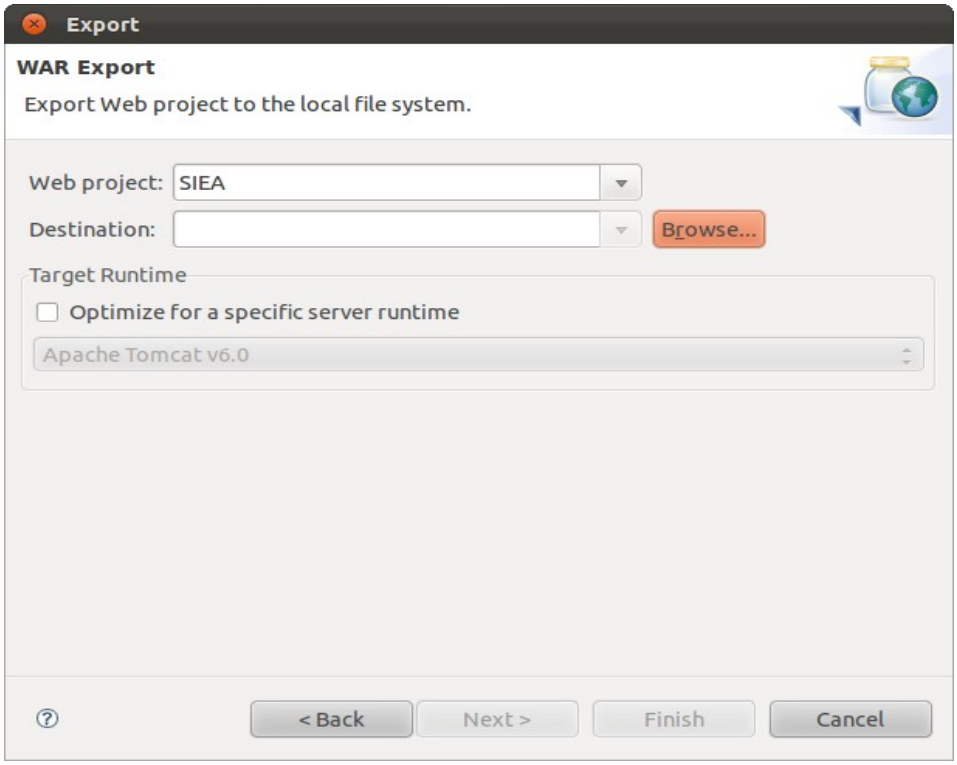


Figura 2

3. La ubicación en donde estará la aplicación es la carpeta de instalación del servidor apache-tomcat.xxx, dentro de esta carpeta se encuentra la carpeta webapps en donde se guardará la aplicación (Figura 3).

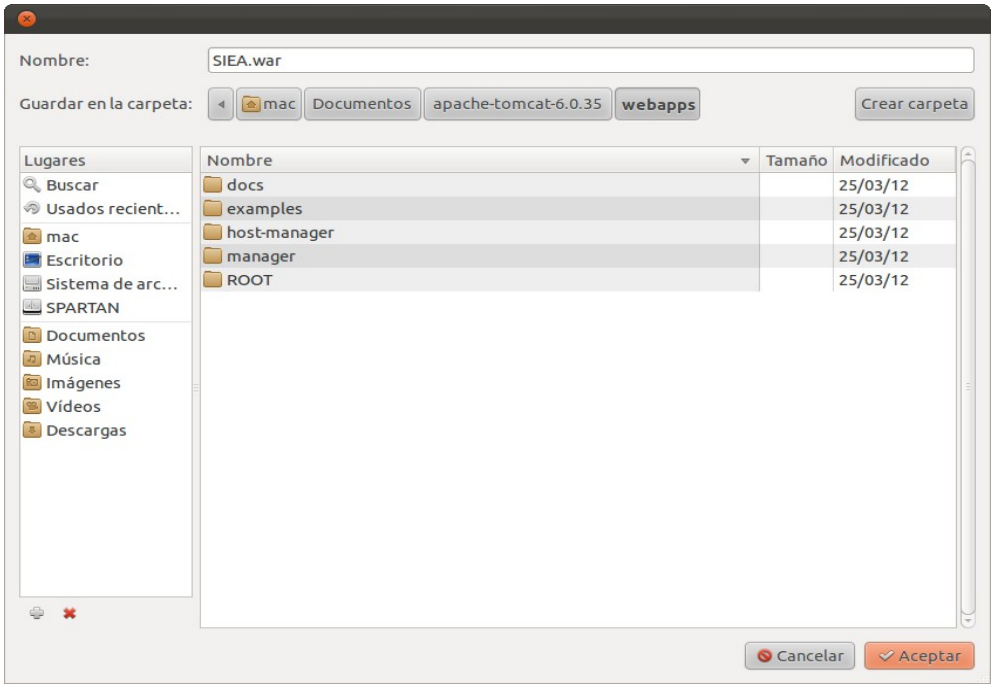


Figura 3

4. Pulsar aceptar y se mostrará la ventana con la ruta en donde estará la aplicación; pulcar finalizar y como paso siguiente se inicializará el servidor

apache-tomcat (Figura 4).

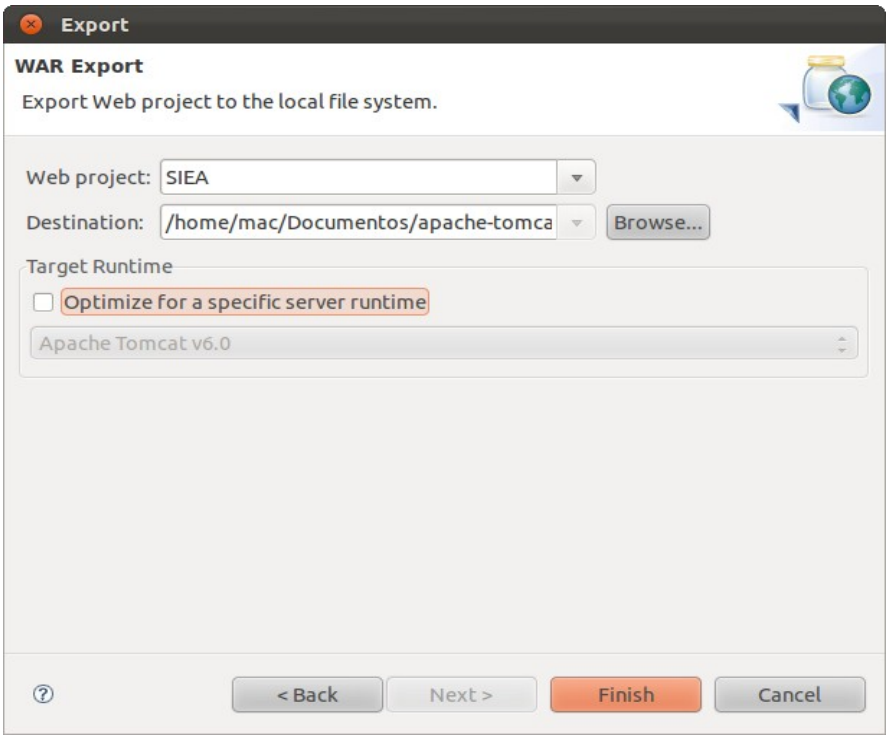


Figura 4

5. El siguiente paso es inicializar el servidor apache-tomcat, ésto se llevará a cabo por medio de la terminal del sistema; luego habrá que dirigirse a la carpeta apache-tomcat.xxx->bin en la cual se encuentra el archivo stratup.sh que sirve para arrancar el servidor de apacheTomcat (Figura 5).

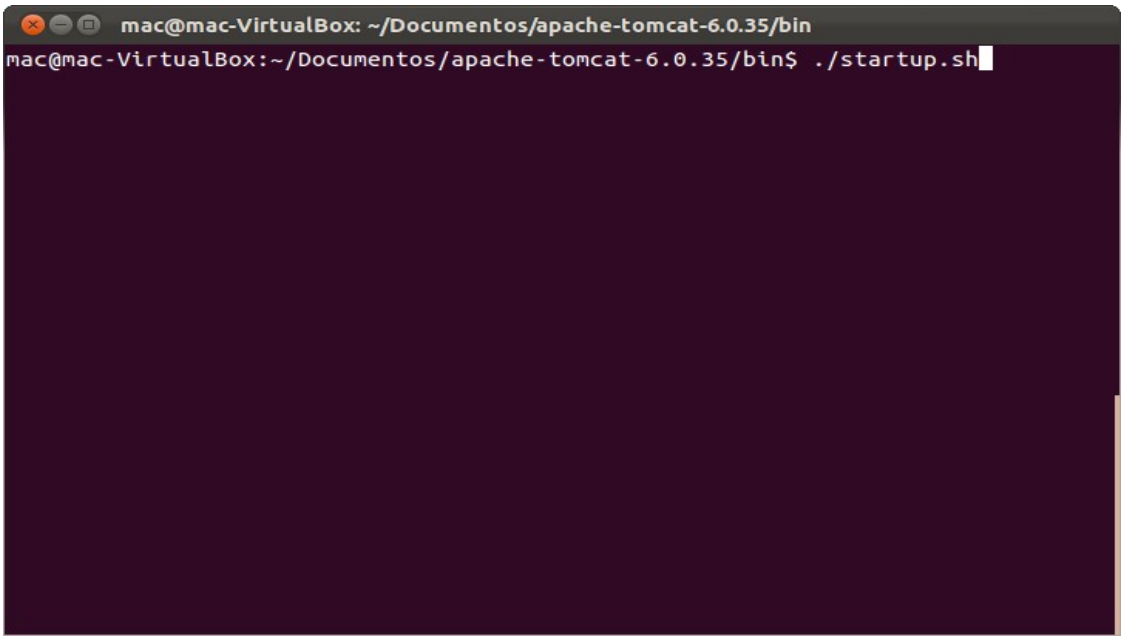


Figura 5

6. Ya inicializado el servidor se mostrará la siguiente respuesta en la terminal, lo que verifica que el servidor arrancó correctamente (Figura 6) y se podrá

verificar que el servidor está activo (Figura 7).

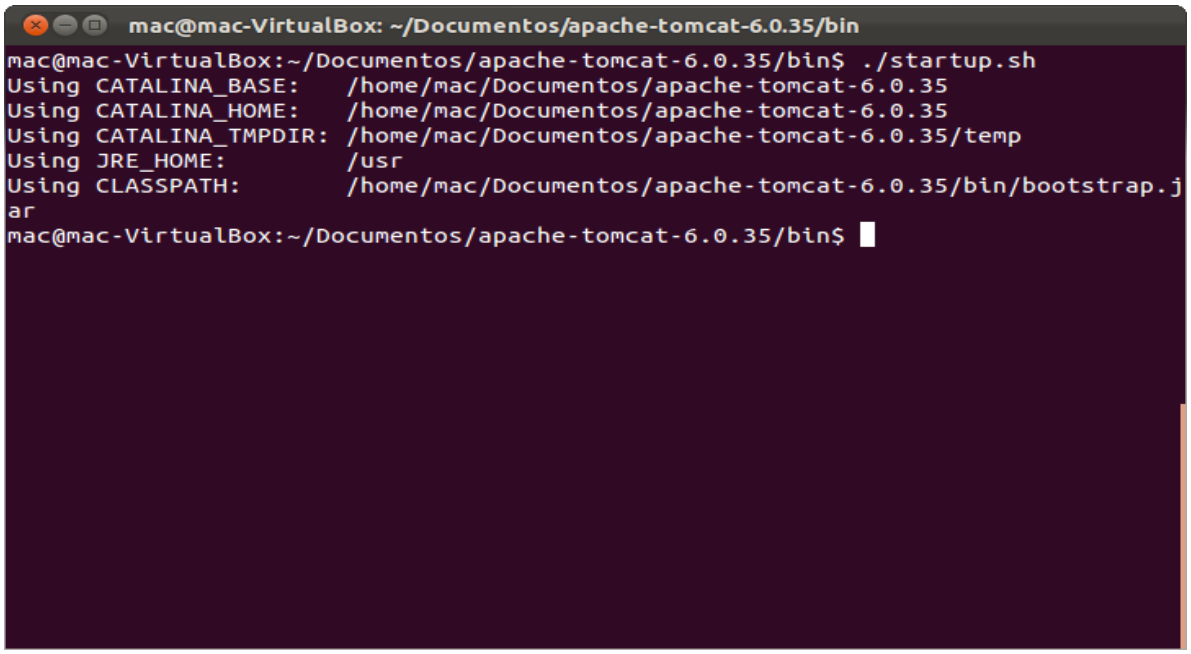


Figura 6

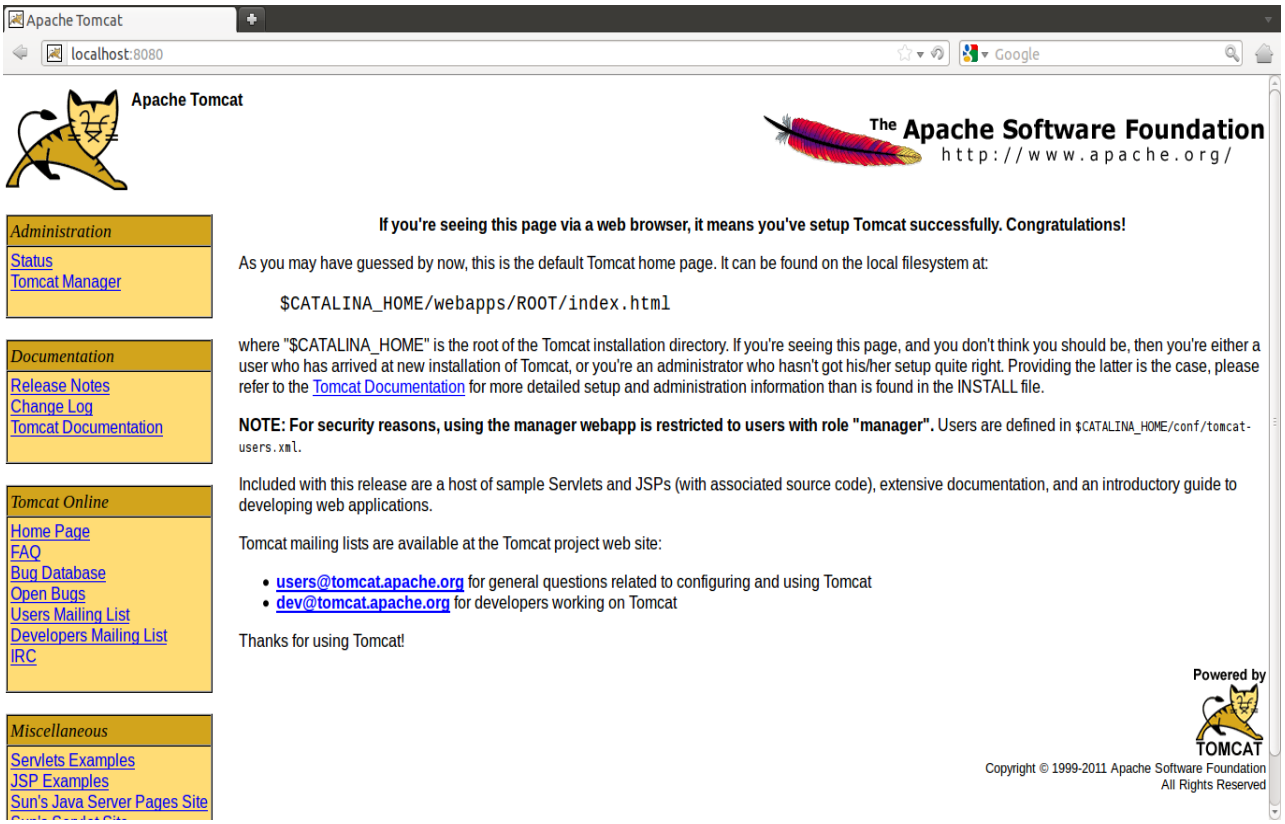


Figura 7

7. Por último se muestra la aplicación en funcionamiento y lista para ser ejecutada (Figura 8).


Bienvenido Login
localhost:8080/SIEA/sistema/login_forma_login.action
Google
Página de validacion
Proporciona tu cuenta de usuario y contraseña
Usuario:
Contraseña:
Entrar
Derechos reservados por macx develops

Figura 8

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

Proyecto Terminal:
SISTEMA DE GESTION DE EVENTOS ACADÉMICOS

Alumno:
Manuel Alejandro Cobos Lomelí
Matrícula: 205305635

Asesor:
Dra. María Lizbeth Gallardo López
Profesor investigador
Departamento de Sistemas

Índice de contenido

INTRODUCCIÓN

Este proyecto terminal es motivado por un proyecto realizado en el año 2007, denominado “SISTEMA DE INSCRIPCIÓN PARA LA SEMANA DE INGENIERÍA EN COMPUTACIÓN 2007”. Este nuevo proyecto busca generalizarse para cualquier evento académico. Inicialmente el proyecto se plantea para la División de Ciencias Básicas e Ingeniería, sin embargo, podría ser útil para cualquier División de la UAM-A.

La idea de este sistema surge para ayudar a gestionar de manera sencilla la información de cualquier evento organizado por la UAM, específicamente de la División de Ciencias Básicas e Ingeniería. Entre las actividades que se han identificado en este tipo de eventos se tienen: talleres, conferencias (de vinculación y magistrales), concursos, actividades lúdicas y definir el comité organizador. Hemos delimitado, para este proyecto terminal, las tres primeras.

Por supuesto que esta propuesta contempla, para el mediano plazo, la integración de dos módulos más, a saber: concursos y actividades lúdicas. Con estos módulos obtendríamos un sistema de gestión para eventos académicos suficiente para el tipo de eventos que se realizan en la División de Ciencias Básicas e Ingeniería.

OBJETIVO GENERAL

Desarrollar una herramienta (aplicación Web) de gestión para las actividades que se requieren en un evento académico de la UAM Azcapotzalco.

Objetivos particulares

1. Definir la arquitectura del sistema de gestión para eventos académicos.
2. Diseñar los módulos para la gestión de la información de Talleres, Conferencias y Comité Organizador.
3. Implementar los módulos anteriormente mencionados.
4. Probar la funcionalidad de los módulos
5. Generar la documentación correspondiente: reporte final, manual de usuario, manual técnico y documento de diseño.

Metodología empleada en el proyecto

La metodología empleada en la elaboración de este proyecto fue el Proceso Unificado (RUP) (Figura 1)[7]. Es un modelo de software que permite el desarrollo de software a gran escala, mediante un proceso continuo de pruebas y retroalimentación, garantizando el cumplimiento de ciertos estándares de calidad. Aunque con el inconveniente de generar mayor complejidad en los controles de administración del mismo. Sin embargo, los beneficios obtenidos recompensan el esfuerzo invertido en este aspecto.

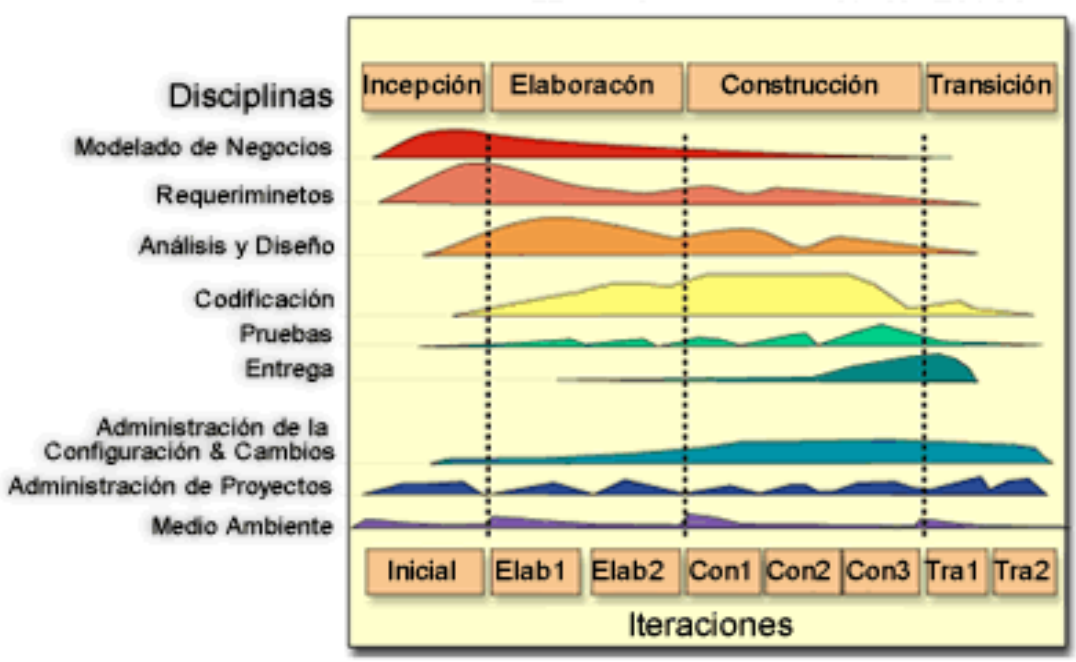


Figura 1. Diagrama del Proceso Unificado

El Proceso Unificado se basa en componentes (*component-based*), lo que significa que el sistema en construcción está hecho de componentes de software interconectados por medio de interfaces bien definidas (*well-defined interfaces*).

El Proceso Unificado usa el Lenguaje de Modelado Unificado (UML) en la preparación de todos los planos del sistema. De hecho, UML es una parte integral del Proceso Unificado.

Los aspectos distintivos del Proceso Unificado están capturados en tres conceptos clave: dirigido por casos de uso (*use-case driven*), centrado en la arquitectura (*architecture-centric*), iterativo e incremental. Esto es lo que hace único al Proceso Unificado.

DESARROLLO DEL PROYECTO

Requerimientos del Proyecto

En este esquema se describen los requerimientos obtenidos del análisis del proyecto, los cuales se encuentran dentro de la fase de inicio del Proceso Unificado

Requisitos	Solución actual	prioridad	Comentarios adicionales
Gestionar uno o varios eventos académicos	Se crea una aplicación Web específica para el evento; en ocasiones se recurre a crear un sitio Web (donde	Alta	Buscamos tener una sola aplicación que permita gestionar distintos eventos,

	solo se muestra la información) específico para el evento.		inclusive al mismo tiempo.
Gestionar las actividades de distintos eventos.	En ocasiones todo se organiza en hojas de calculo y en documentos de texto, para luego encargar a alguien “vaciar” la información en la aplicación o en el sitio Web, específicos del evento.	Alta	Buscamos repartir el trabajo y que existan responsables por cada actividad definida en un evento; la persona responsable deberá encargarse de registrar la información correspondiente.
Gestionar la información del evento	Tenemos dos modalidades para la gestión de la información: 1) reportes, cuando el evento está en proceso y 2) reportes, cuando el evento ya concluyó. Este trabajo se realiza a partir de los registros (guardados en los distintos documentos); o a partir de la base datos (cuando existe).	Alta	Buscamos poder contar con reportes a partir de que el evento sea creado en el sistema, y aún después de terminado. Durante el evento, los reportes permitirán retroalimentar a los organizadores. Al final del evento, los reportes permitirán tener una clara idea del éxito del evento.

Tabla 1. Requisitos del proyecto.

Requerimientos funcionales

Para la configuración inicial de eventos, el sistema debe permitir:

1. Alta, baja, cambio y consulta de un evento.
2. Definir a los integrantes del comité organizador, así como sus roles dentro del evento.
3. Determinar las actividades que se realizarán en el marco del evento.
4. Definir el número(cupo) de participantes que asistirán al evento.

Para la Gestión de las actividades del evento**,** el sistema debe de permitir:

1. Altas, bajas, cambios, consultas y reportes de las distintas actividades para el evento.
2. Confirmar las actividades definidas en el sistema.

Para la Gestión de la información del evento el sistema de permitir:

1. Obtener, distintos reportes, sobre la información del evento, correspondiente a las actividades registradas y el comité organizador.
2. Exportar los reportes en diferentes formatos, a saber: excel, pdf, etc.

Requerimientos no funcionales

1. Proporcionar acceso a la base de datos del sistema, para que quien gestiona la página de la División pueda obtener los datos (incluida la URL) de los eventos activos en el sistema "Eventos Académicos" así como su descripción general.
2. Que la base de datos esté centralizada y pueda ser accedida de forma remota, para que con ello se tenga la información necesaria para gestionar de manera global los eventos, reportes, usuarios y permisos correspondientes.
3. Que el sistema soporte la concurrencia de los usuarios.
4. Que las plantillas de presentación sean definidas bajo estándares de usabilidad.
5. Que el sistema implemente el modelo MVC[1] para su mejor mantenibilidad y extensibilidad con miras en cambios futuros.
6. El sistema deberá contar con un mecanismo de parametrización para la instalación, despliegue y configuración ya sea manual o asistido por el propio sistema.
7. El sistema terminado deberá ser empaquetado para su mejor distribución, manejo de versiones y licenciamiento de uso.

DISEÑO

Esta etapa consta de los siguientes esquemas: diagrama de casos de uso general, así como sus escenarios correspondientes, diagrama de dominio, diagramas de secuencia [8] y la estructura de la base de datos.

Casos de uso y diagrama de dominio

La figura 2 muestra el caso de uso general señalando las acciones que el sistema es capaz de realizar, a saber: generar eventos académicos los cuales contarán con actividades que podrán variar dependiendo del evento que organice un comité determinado.

1 Modelo-Vista-Controlador

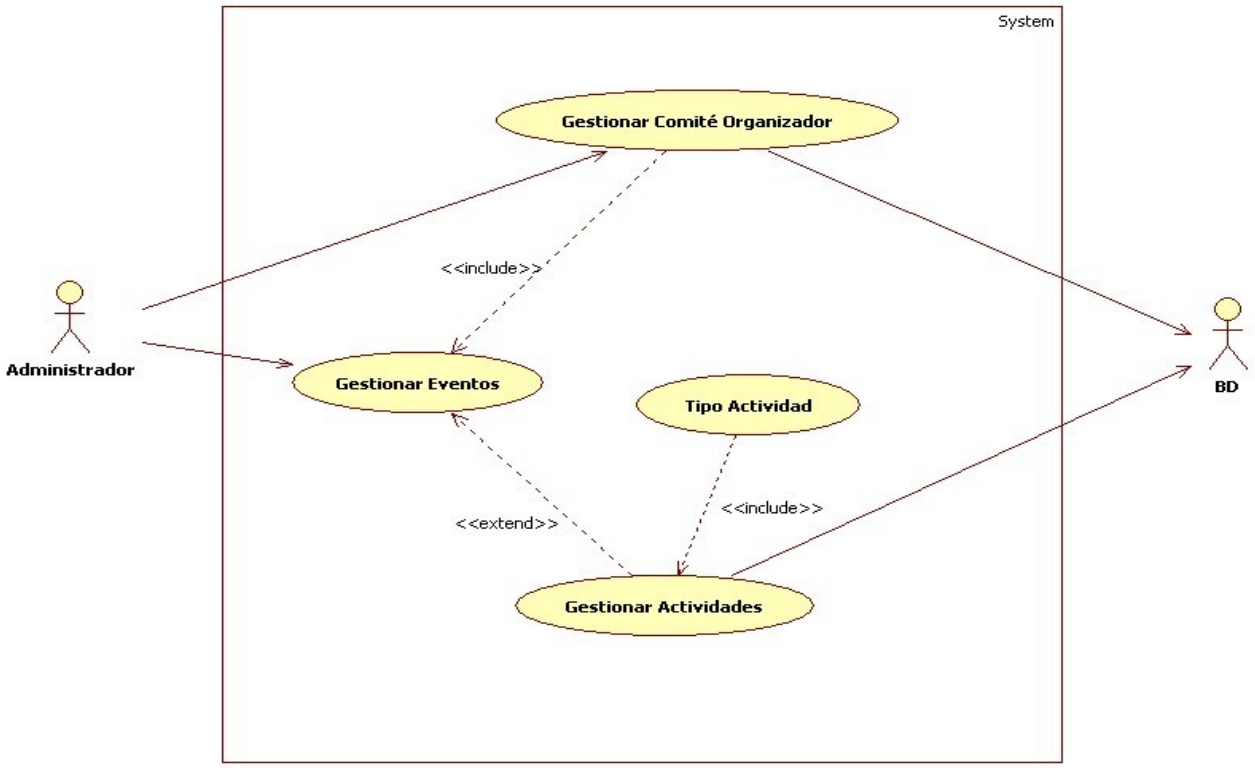


Figura 2. Diagrama de casos de uso general

Caso de uso: Gestionar Eventos

El administrador tendrá la facultad de generar, consultar, modificar o borrar un evento académico en el sistema. A continuación se describe el escenario de uso de `generar` un evento académico, para revisar los otros escenarios consultar el documento de diseño.

Caso de uso:	Generar evento
Descripción:	El usuario tendrá la posibilidad de generar un evento académico, insertando en el sistema los datos correspondientes a la actividad que desea generar, esto es, podrá ingresar el nombre del evento, una descripción del mismo, la fecha de creación, una autorización y el tipo de evento.

Caso de uso: Gestionar Actividades

El administrador tendrá la facultad de generar, consultar, modificar o borrar una actividad académica en el sistema. A continuación se describe el escenario de uso de `consultar` actividad, para revisar los otros escenarios consultar el documento de diseño.

Caso de uso:	Consultar Actividad
Descripción	El usuario tendrá la posibilidad de consultar una actividad asociada a un evento académico, esto se llevará a cabo seleccionando un evento académico y éste mostrará las actividades que corresponden al evento académico; por lo cual, al seleccionar la clave de la actividad, se mostrará la actividad académica particular, mostrando: su nombre, a quien va dirigido, las bases de la actividad, una breve descripción, un objetivo, el tipo de actividad, cupo, cortesías, costo interno y externo, autorización, fecha de inicio y de fin así como la fecha de creación.

La figura 3 presenta el modelo de dominio con las principales entidades que conforman un evento académico, así como las relaciones que guardan entre ellas.

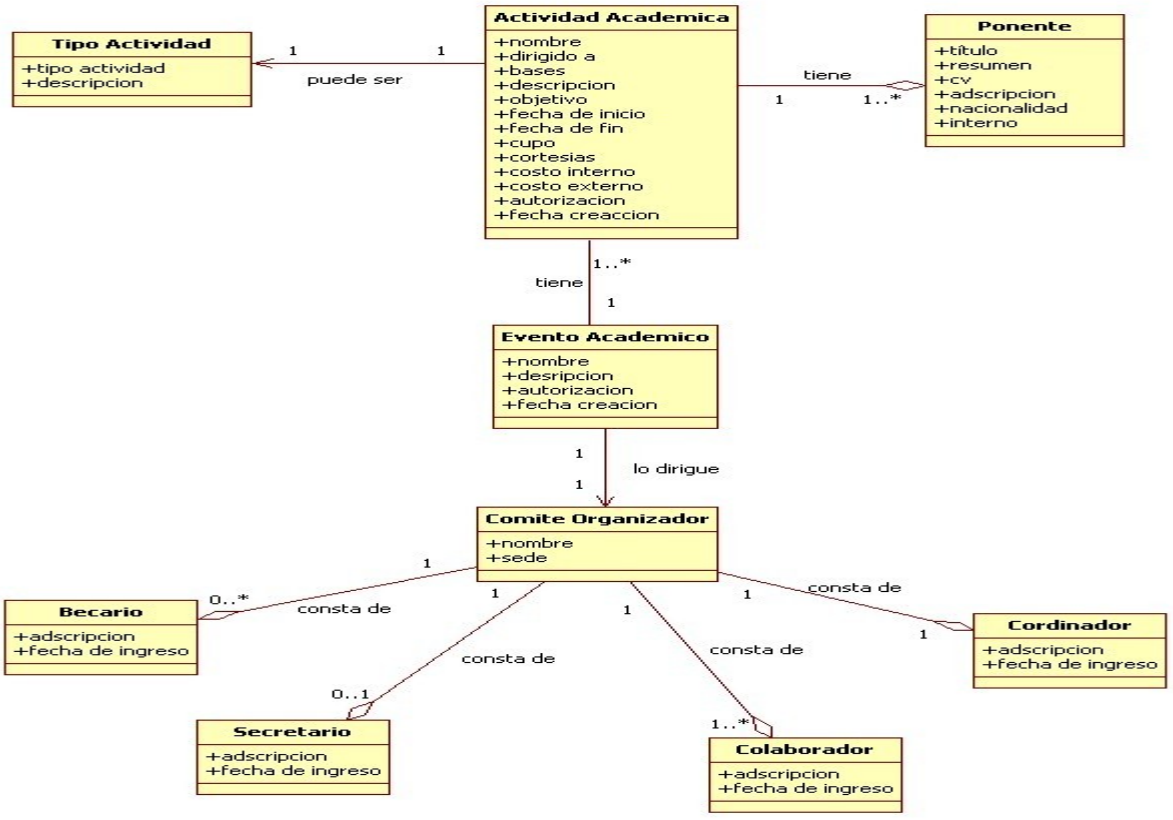


Figura 3. Diseño del Modelo de Dominio

Tanto el diagrama de casos de uso general como el modelo de dominio, se elaboraron en la fase de inicio que marca el Proceso Unificado.

Diagramas de secuencia

A continuación se explican algunos de los diagramas de secuencia correspondientes a los escenarios de uso descritos en la sección 3.1.

La figura 4 muestra la operación de `consultar` por parte del administrador, seleccionando el `ID` (clave) de la actividad del evento al que corresponda.

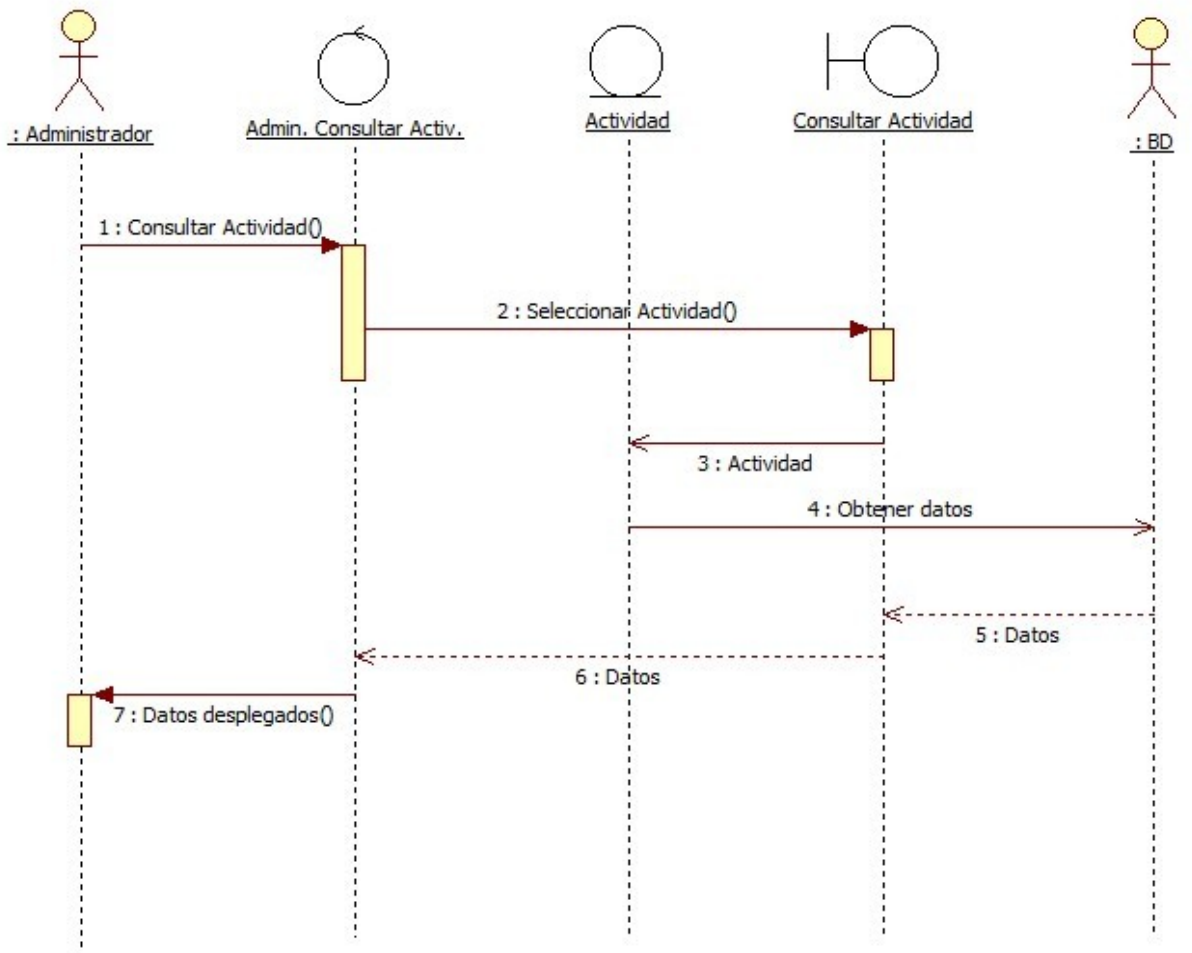


Figura 4. Diagrama de secuencia Consultar Actividad

La figura 5 muestra la secuencia de pasos que se lleva a cabo para `crear` una nueva actividad académica. El administrador ingresará los datos correspondientes. Si los datos son correctos, se crea un nuevo objeto actividad el cual se guardará en la base de datos; de lo contrario, se le notificará al usuario que los datos están mal.

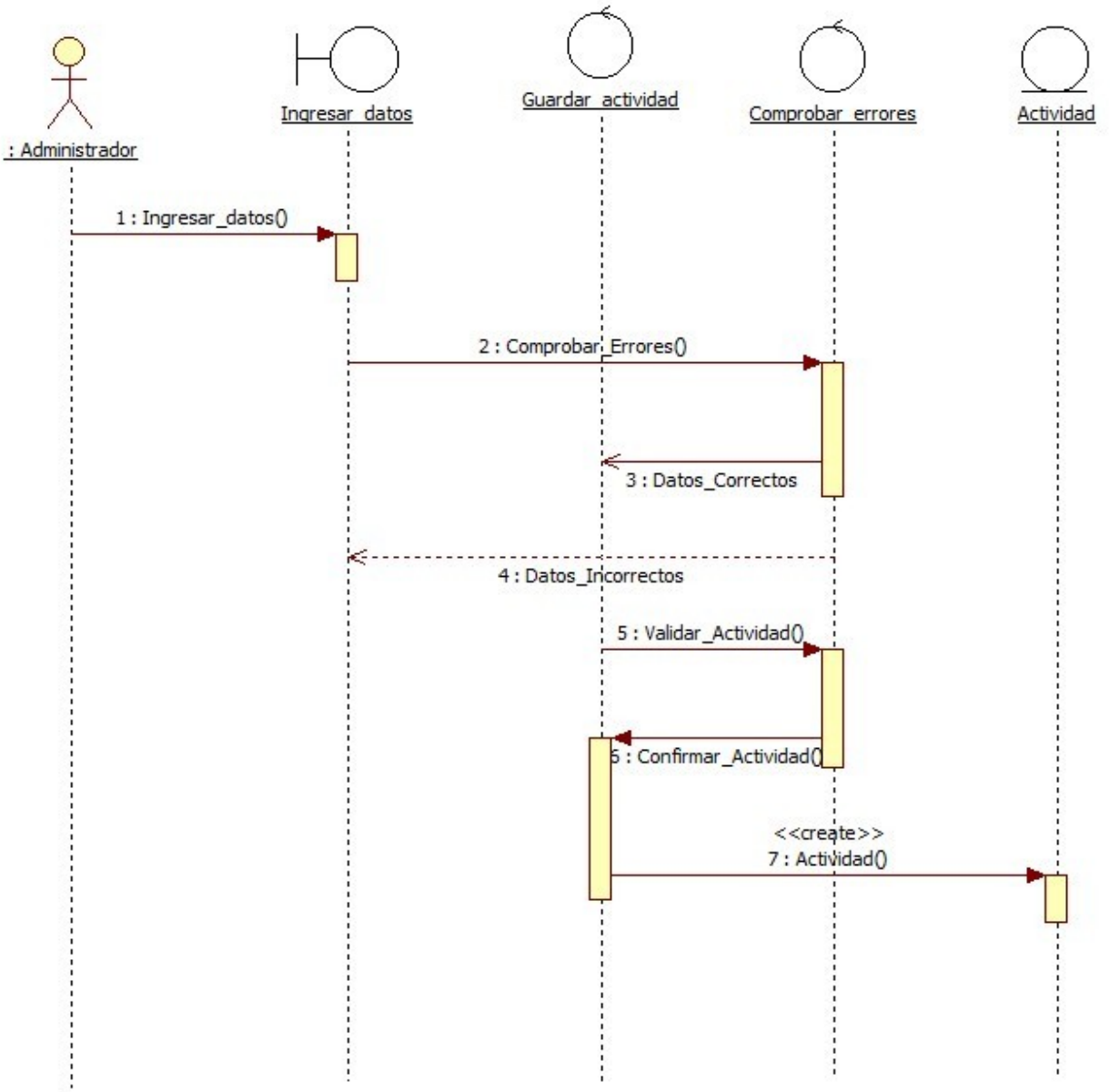


Figura 5. Diagrama de secuencia Crear Actividad

La figura 6 muestra el diagrama de secuencia para modificar una actividad académica. Se iniciará con la `consulta` de la actividad sobre la pantalla; después, el administrador realizará la o las modificaciones pertinentes; al seccionar la opción `modificar` se actualizarán los datos; finalmente, la actividad será mostrada en la pantalla con la información modificada.

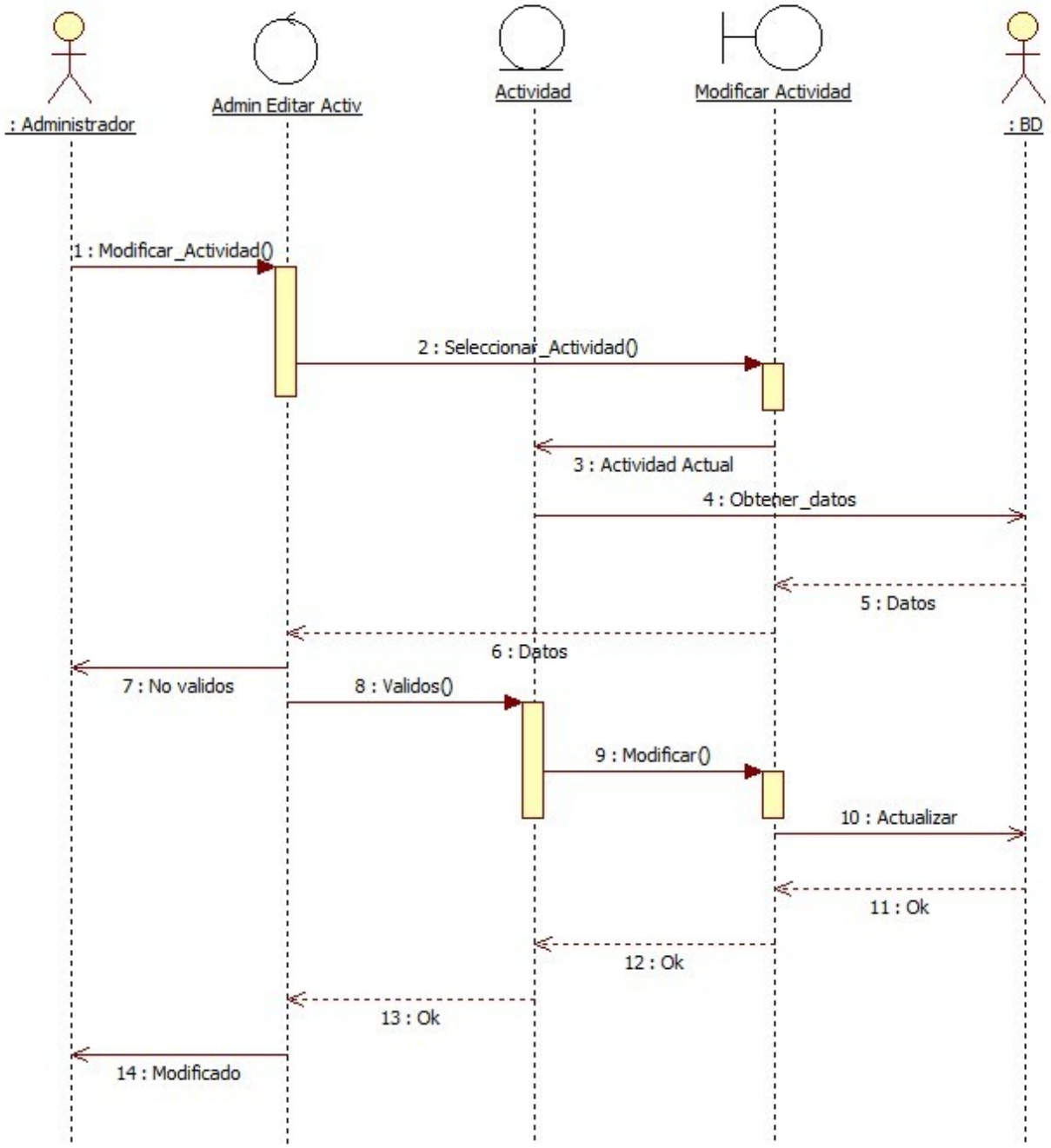


Figura 6. Diagrama de secuencia Modificar Actividad

La figura 7 muestra el diagrama de secuencia para `borrar` una actividad académica. El administrador seleccionará `borrar` una actividad académica; el sistema comprobará la actividad y preguntará al administrador si desea eliminar la actividad actual, si la respuesta es afirmativa, se le notificará al administrador y la actividad quedará eliminada del evento al que pertenecía, así como del sistema.

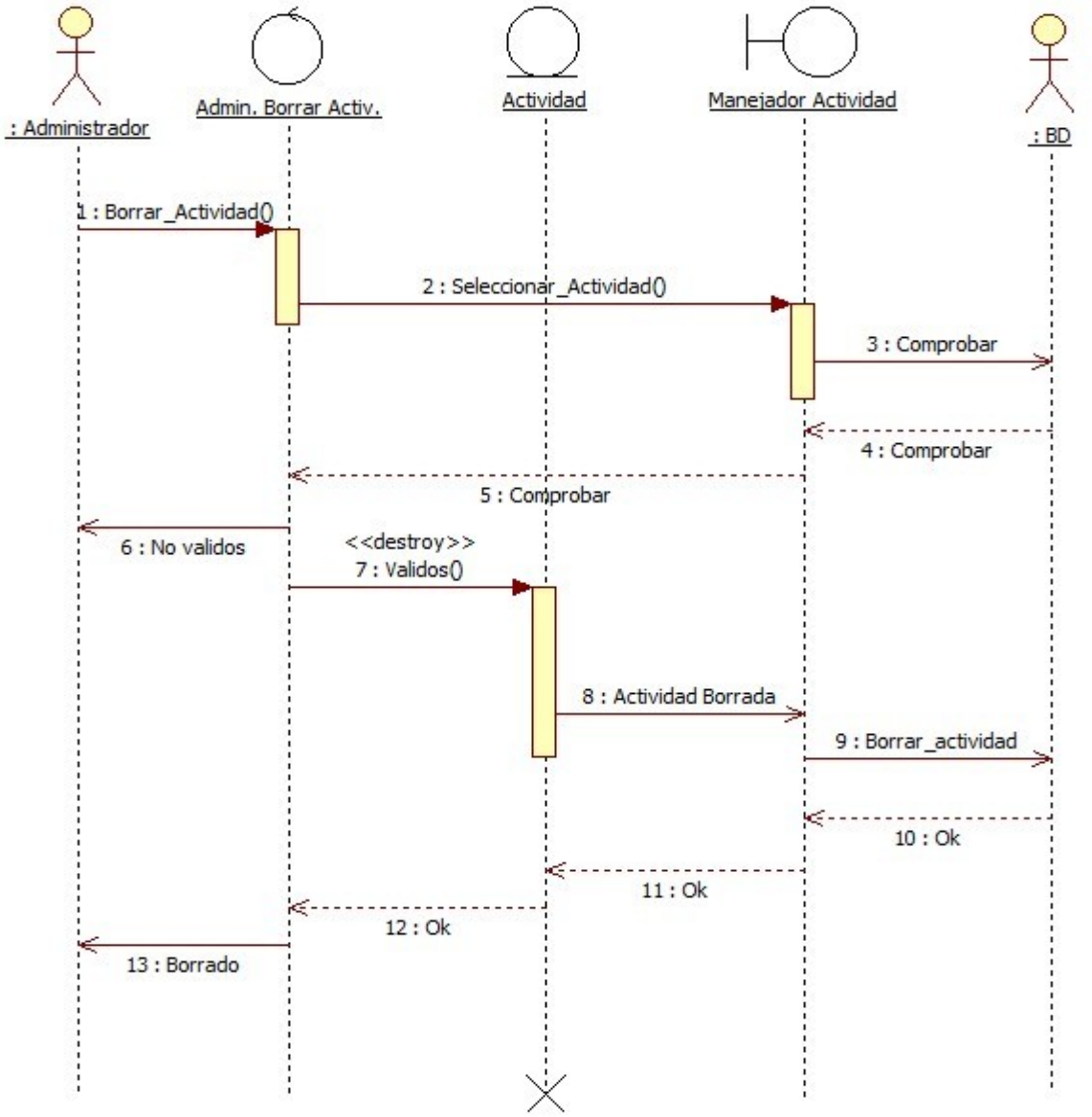


Figura 7. Diagrama de secuencia Borrar Actividad

Es importante señalar que los diagramas de secuencia para eventos académicos y comité organizador siguen el mismo patrón de secuencia en cuanto a la creación, modificación, consulta y borrado que hemos presentado para gestión de actividades; por lo tanto, no los esquematizaremos. Consultar el documento de diseño, donde se detallan más ampliamente cada uno de los módulos.

Base de datos

La estructura de la base de datos que el sistema utiliza fue elaborada por el alumno Guillermo Monroy Rodríguez, quien como parte de su reporte de servicio social [2], se encargó de generar dicha estructura. La figura 8 muestra las tablas que requerimos emplear para los módulos que se han desarrollado en este proyecto terminal.

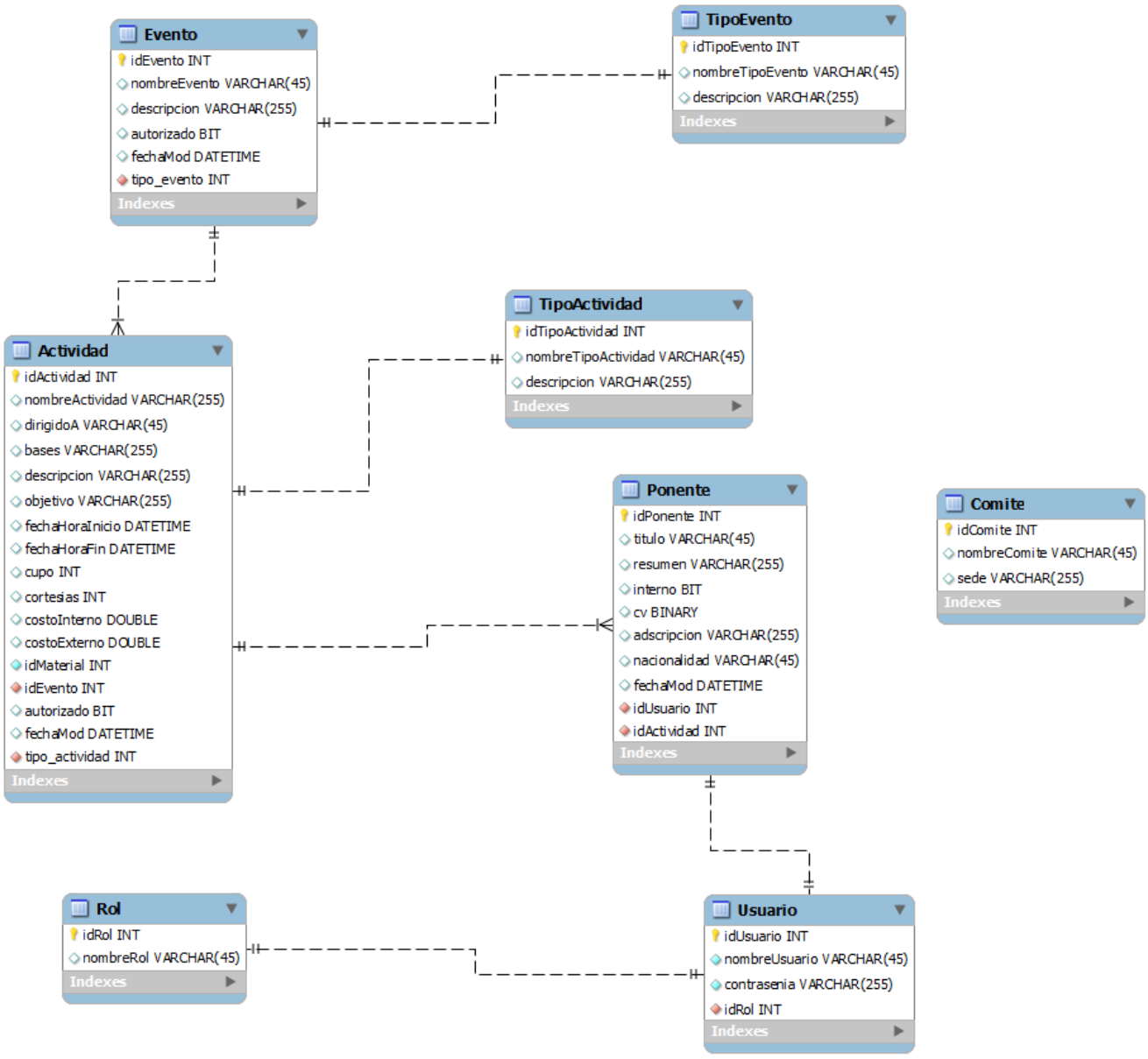


Figura 8.Diagrama Entidad-Relación Base de Datos SIEA

IMPLEMENTACIÓN

Para la implementación del sistema se utilizó el *framework Apache Struts2* [9], el cual es un *framework* para el desarrollo de aplicaciones *web* que hace que estas sean más robustas y flexibles. *Struts2* es un *framework* de presentación, dentro de las capas en las que se divide una aplicación en la arquitectura *JEE*[2], implementando el patrón de diseño MVC. El *framework* proporciona una perfecta integración con otros *framework* para la implementación de la capa del modelo como *Hibernate* [3], y *Spring* [4] además de proporcionar componentes para la capa de la vista como son los *Jsp.*

2 Java Enterprise Edition

El sistema tiene un paquete base mostrado en la figura 9, el cual se compone de:

1. El dominio del servidor donde residirá la aplicación, por ejemplo: *cbi.azc.uam.mx*.
2. El nombre único del proyecto, por ejemplo: SIEA.

Como paquete base se asignó el siguiente nombre: *mx.uam.azc.cbi*.

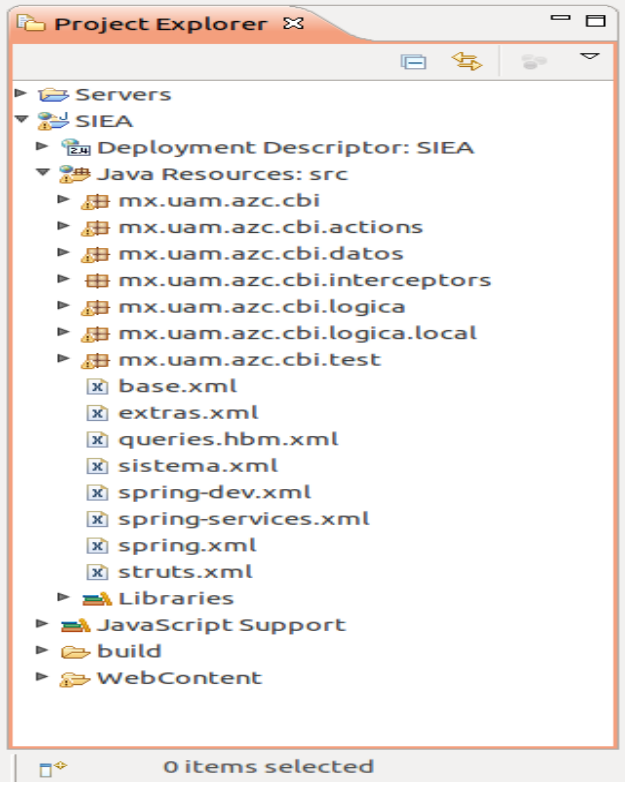


Figura 9. Esquema del paquete base del sistema

Arquitectura del sistema

La figura 10 muestra los elementos del sistema "*Gestión de eventos académicos*", definidos a partir del patrón de diseño MVC *(Modelo-Vista-Controlador); el* El web *Framework Apache Struts2* nos permite implementar el modelo MVC de nuestra aplicación, así como definir el dominio donde ella residirá. En el bloque Controlador, el sistema atiende las peticiones del cliente por medio de interceptores y acciones. El bloque Modelo, contiene la lógica del negocio constituida por los datos referentes a los eventos, actividades y el comité organizador. El bloque Vista plasma los resultados generados por la interacción del Controlador y el Modelo en páginas jsp[3] y se presenta al cliente como resultado de la petición que realizó.

3 JavaServer Pages.

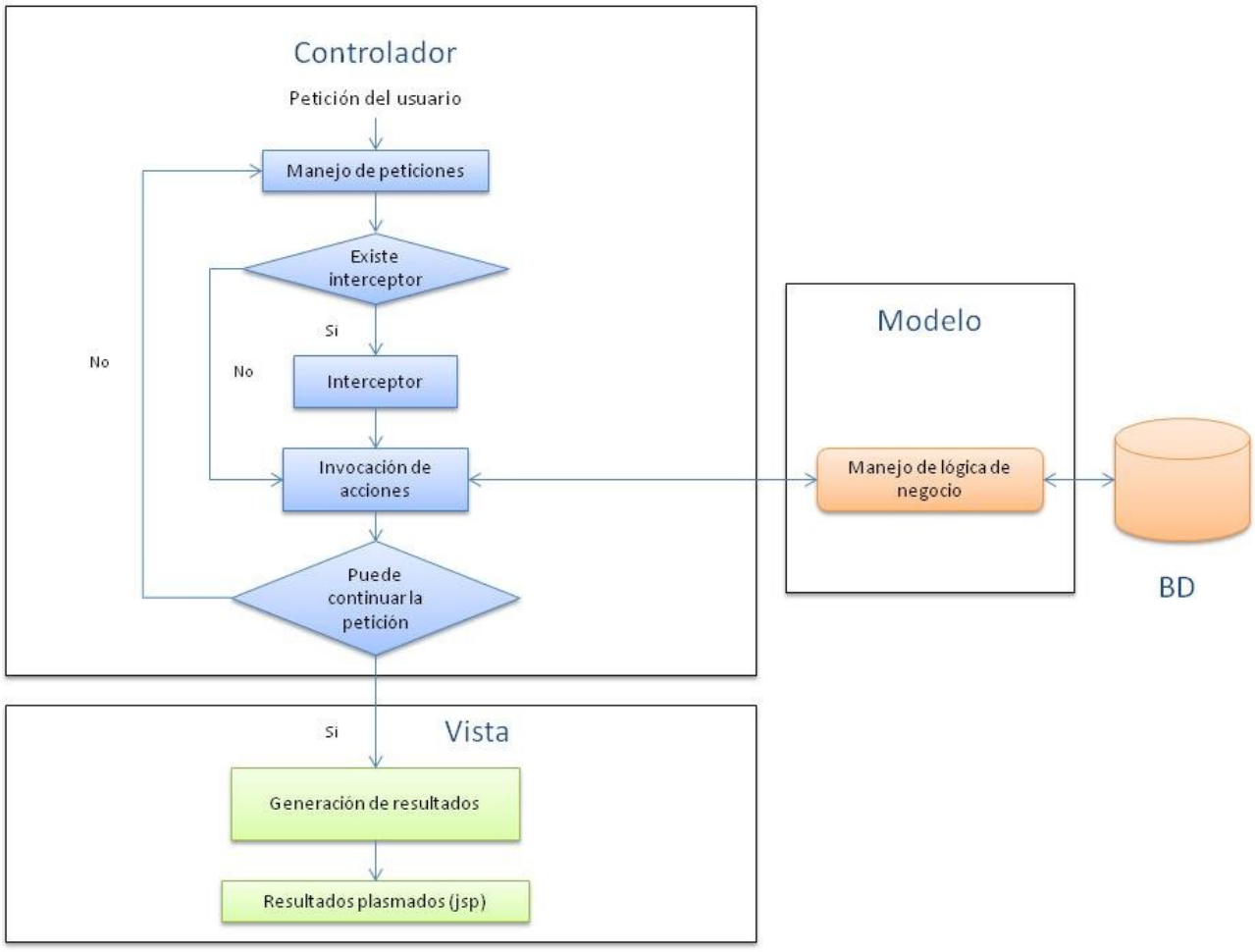


Figura 10. Arquitectura del sistema SIEA.

Implementación del framework Apache Struts2

Struts2 procesa peticiones usando tres elementos principales: `interceptores`, `acciones` y `resultados`. El `filtro` conocido como `FilterDispatcher` ejecuta todas las peticiones que involucran al *framework*, este filtro ejecuta los *Action* que son los manejadores de peticiones, como resultado de la ejecución de los *Action* se genera un objeto (*Result)* de tipo *String* que a su vez produce una respuesta para el usuario. Las clases encargadas de llevar a cabo la lógica de negocio y responder a una petición estarán contenidas en la carpeta *mx.uam.azc.cbi.action* (figura 11), dichas clases heredan de la clase base `com.opensymphony.xwork2.ActionSupport`, la cual usa el nombre de la clase con sufijo `Action` (por convención) para llevar a cabo el mapeo de peticiones URL y así saber cual clase `Action` debe ser llamada para dicha petición.

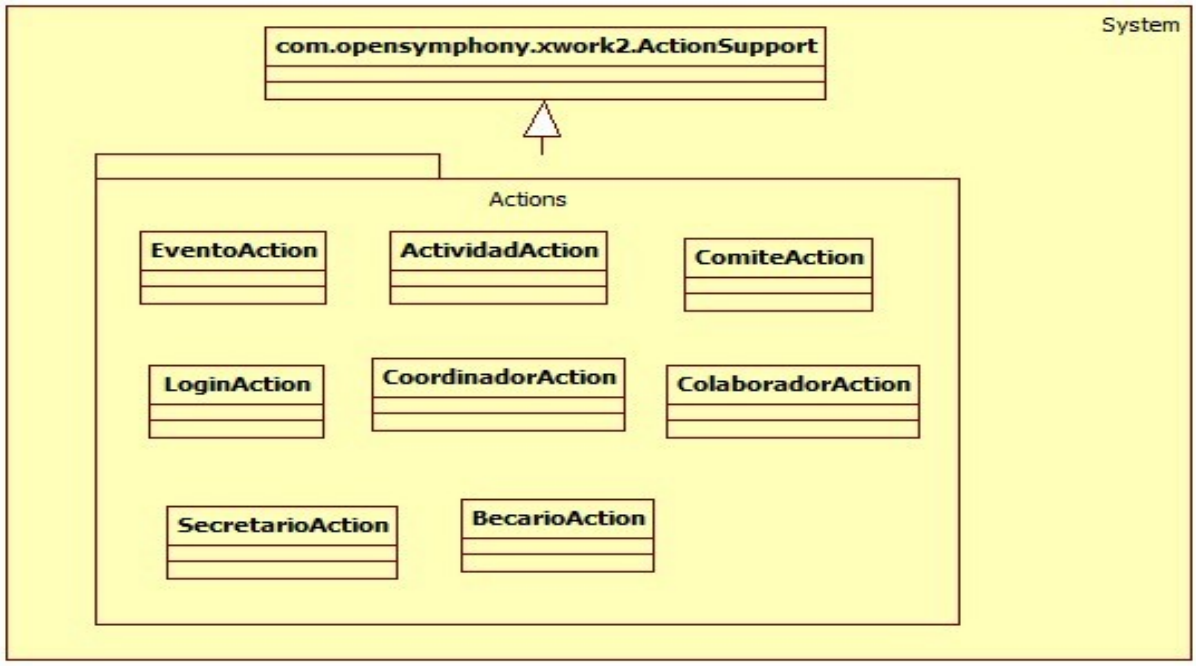


Figura 11. Diagrama de clases carpeta *mx.uam.azc.cbi.actions*

En la figura 12 se muestra la clase `EventoAction` la cual consta del método `listar()`, este método genera el `String` de retorno "`listar`" que servirá para obtener el `Result` correspondiente de la configuración del `Action`, en el archivo `sistema.xml`.

EventoAction.java

```
17 public class EventoAction extends ActionSupport implements SessionAware,
18     Preparable
19 {
20
21   private EventoJoinDTO _evento;
22   private List<EventoJoinDTO> _eventos = new ArrayList<EventoJoinDTO>();
23   private EventoDTO _eventoSave;
24   private EventoDTO _eventoUpdate;
25   private long _id;// id del evento
26   private List<ActividadJoinDTO> _actividades = new ArrayList<ActividadJoinDTO>();
27   private Map _sessionData;
28   private FiltroEventos _filtro = new FiltroEventos();
29   private AdministradorEvento _administradorEvento;
30   private AdministradorActividad _administradorActividad;
31   private AdministradorTiposEventos _administradorTiposEventos;
32
33   public String listar()
34   {
35     _sessionData.put( "idEventos", null );
36     FiltroEventos _filter = new FiltroEventos();
37     _filter.setNombre("");
38     _eventos = _administradorEvento.leerEventos( _filter );
39     addActionMessage( "La busqueda fue exitosa" );
40     return "listar";
41   }
```

Figura 12. Clase `EventoAction`

La figura 13 muestra el archivo de configuración para los `Actions`. En este archivo se listan los diferentes `Results` que se mostrarán al usuario, quien realiza una petición mediante un `Action`. El `Result` generado es plasmado en la página `listar_evento.jsp`; sin embargo, si se trata de un administrador del sistema, éste puede agregar o modificar los `Results`; o bien, redirigirlos a otras páginas jsp u otros `Actions` del sistema.

sistema.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<!--**** Aqui se declaran los interceptores, acciones y resultados*****-->
<struts>

    <package name="sistema" namespace="/sistema" extends="base">
        <action name="evento_*"
        class="mx.uam.azc.cbi.actions.EventoAction" method="{1}">
            <interceptor-ref name="eventoStack"/>
            <result name="menu">/WEB-INF/paginas/evento/menu.jsp</result>
            <result name="listar">/WEB-INF/paginas/evento/listar_evento.jsp</result>
            <result name="ver">/WEB-INF/paginas/evento/ver_evento.jsp</result>
            <result name="insertar_evento">/WEB-INF/paginas/evento/insertar_evento.jsp</result>
            <result name="actualizar_evento">/WEB-INF/paginas/evento/actualizar_evento.jsp</result>
            <result name="listar_redirect" type="redirectAction">evento_listar</result>
            <result name="ver_redirect" type="redirectAction">evento_ver?id=${idEventos }</result>
            <result name="error" type="redirectAction">
              <param name="actionName">login_forma_login</param>
            </result>
        </action>
```

Figura 13. Archivo de configuración de `Actions sistema.xml`

En la lógica de negocios se definen métodos que administran la conexión a bases de datos, las transacciones, los mapéos de entidades relacionadas con la base de datos y servicios. Se usan interfaces como aislantes de la capa de presentación hacia la lógica del negocio, estas interfaces integran la capa de presentación a la capa de modelo.

Las interfaces administran responsabilidades que cubren las necesidades que el sistema necesita, esto es, deben cubrir necesidades tales como: ingresar un evento, modificarlo, borrarlo y consultarlo. Estas interfaces manejan los datos fundamentales que conforman al módulo que se quiere generar.

En el paquete *mx.uam.azc.cbi.logic*, se definen las interfaces que funcionan como administradores de la parte transaccional de los datos en el sistema. La figura 14 muestra el diagrama de clases que componen el paquete de la lógica de negocio junto con las clases que implementan estas interfaces.

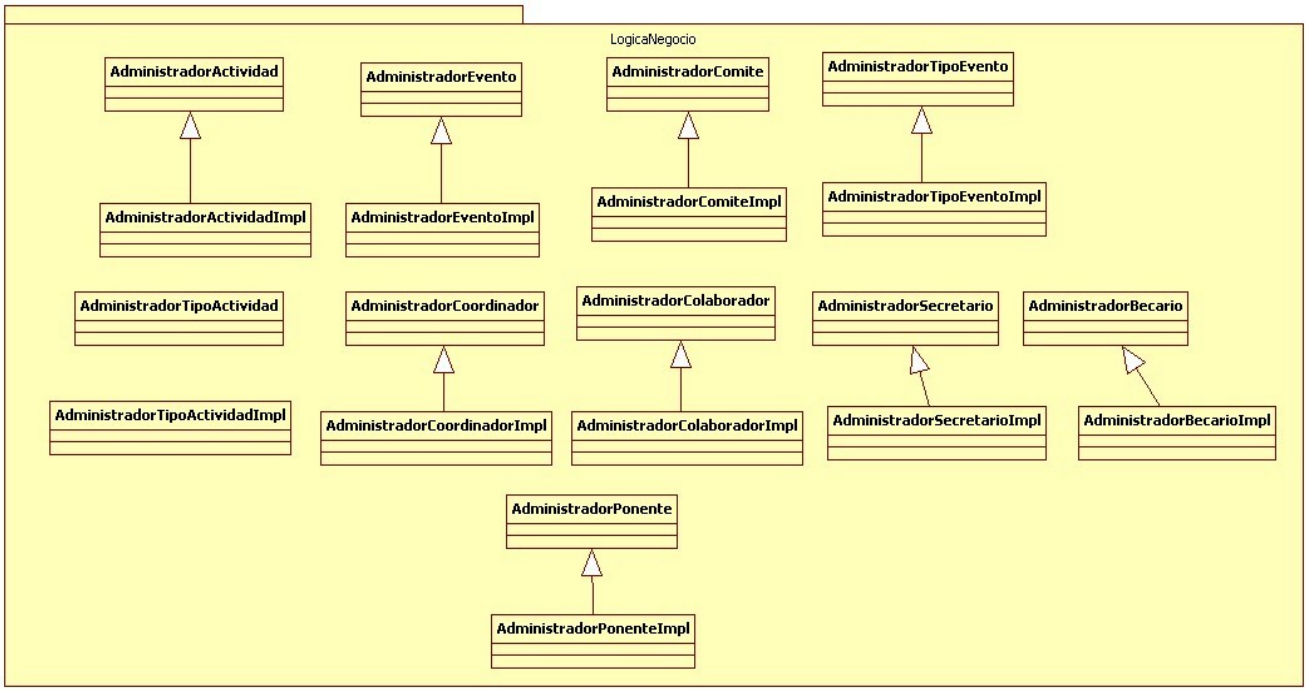


Figura 14. Diagrama de clases de Lógica de Negocio

Para la configuración de los componentes de nuestro sistema de eventos académicos, se utilizó el framework `Spring` el cual tiene soporte a varias tecnologías de conexión a bases de datos (`Hibernate` del cual se hablará más adelante), persistencia (JPA) e integración de servicios de aplicaciones java.

La figura 15 muestra la configuración de los servicios que llevarán a cabo la lógica de negocios, estos servicios hacen una integración directa con los componentes que utilizarán las conexiones a la base de datos, el mapeo de los datos y las transacciones.

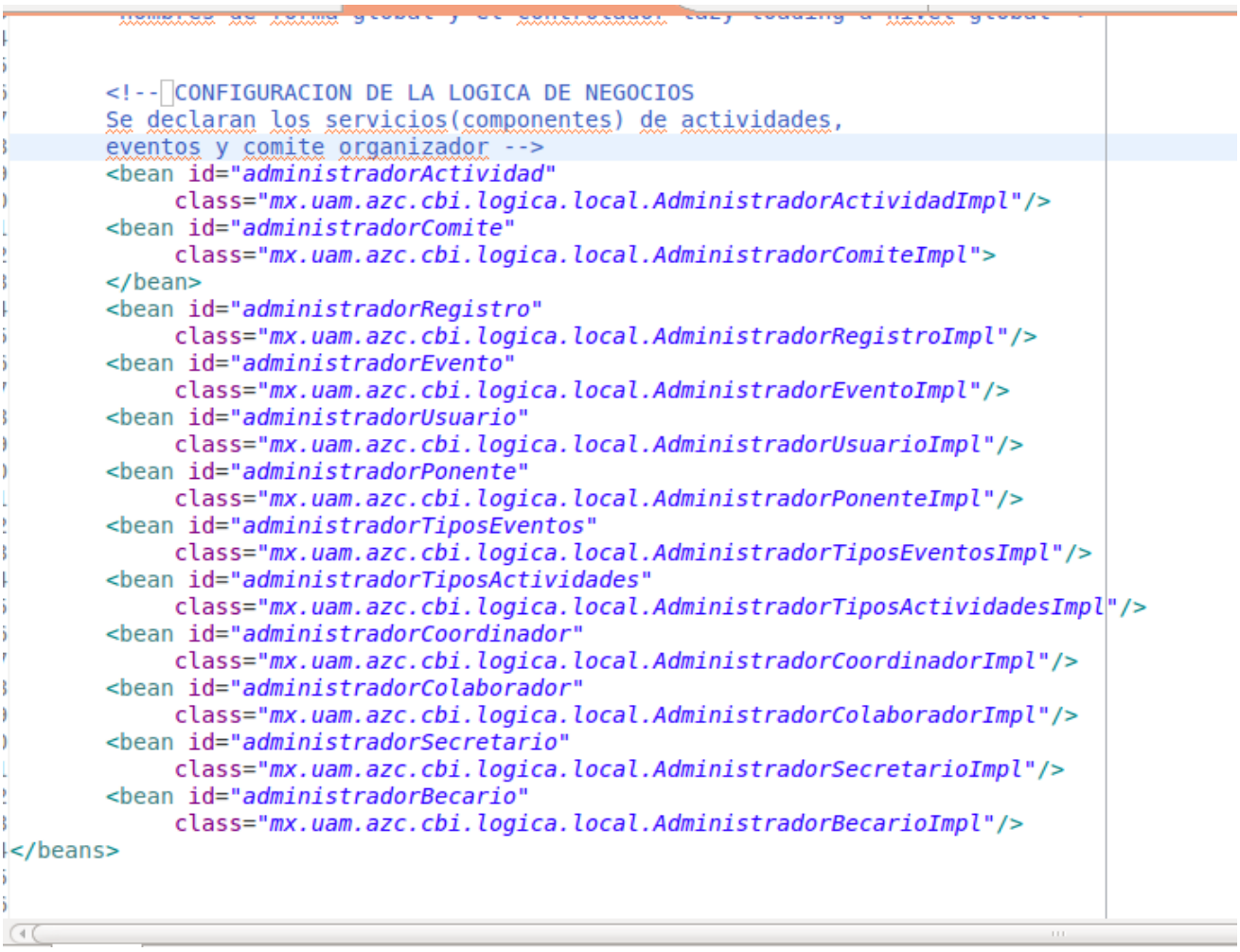
spring-dev.xml

```
<!-- INFORMACION DEL AMBIENTE DE DESARROLLO
Declaracion del componente dataSource como servicio el cual es la fabrica de conexiones a BD-->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
     <property name="driverClassName">
         <value>org.postgresql.Driver</value>
     </property>
     <property name="url">
         <value>jdbc:postgresql://localhost:5432/siea?autoReconnect=true</value>
     </property>
     <property name="maxWait" value="10000"/>
     <property name="username" value="postgres"/>
     <property name="password" value="root"/>
</bean>
<!-- Declaracion del componente para manejo de transacciones en java así como el manejo de hibernate-->
<bean id="transactionManager"
     class="org.springframework.orm.hibernate3.HibernateTransactionManager"/>
<tx:annotation-driven/><!-- se configuran y se activan los interceptores AOP de transacciones -->

<!-- Se declara el sessionFactory para crear y utilizar las sesiones a la base de datos -->
<bean id="sessionFactory"
     class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
     <property name="annotatedClasses">
         <list><!-- Son los objetos que se van a mapear -->
             <value>mx.uam.azc.cbi.datos.ComiteDTO</value>
             <value>mx.uam.azc.cbi.datos.EventoDTO</value>
             <value>mx.uam.azc.cbi.datos.ActividadDTO</value>
             <value>mx.uam.azc.cbi.datos.UsuarioDTO</value>
             <value>mx.uam.azc.cbi.datos.TipoEventoDTO</value>
             <value>mx.uam.azc.cbi.datos.EventoJoinDTO</value>
             <value>mx.uam.azc.cbi.datos.ActividadJoinDTO</value>
             <value>mx.uam.azc.cbi.datos.TipoActividadDTO</value>
             <value>mx.uam.azc.cbi.datos.PonenteDTO</value>
             <value>mx.uam.azc.cbi.datos.UsuarioDTO</value>
```

Figura 15. Archivo de configuración `spring-dev.xml`

La figura 16 muestra la declaración y configuración de los servicios, estos servicios serán implementados mediantes clases que heredan de las interfaces generadas anteriormente y que administran los servicios de comunicación y manejo de datos entre la capa de presentación y el modelo de negocios.



```
        <!--CONFIGURACION DE LA LOGICA DE NEGOCIOS
        Se declaran los servicios(componentes) de actividades,
        eventos y comite organizador -->
        <bean id="administradorActividad"
              class="mx.uam.azc.cbi.logica.local.AdministradorActividadImpl"/>
        <bean id="administradorComite"
              class="mx.uam.azc.cbi.logica.local.AdministradorComiteImpl">
        </bean>
        <bean id="administradorRegistro"
              class="mx.uam.azc.cbi.logica.local.AdministradorRegistroImpl"/>
        <bean id="administradorEvento"
              class="mx.uam.azc.cbi.logica.local.AdministradorEventoImpl"/>
        <bean id="administradorUsuario"
              class="mx.uam.azc.cbi.logica.local.AdministradorUsuarioImpl"/>
        <bean id="administradorPonente"
              class="mx.uam.azc.cbi.logica.local.AdministradorPonenteImpl"/>
        <bean id="administradorTiposEventos"
              class="mx.uam.azc.cbi.logica.local.AdministradorTiposEventosImpl"/>
        <bean id="administradorTiposActividades"
              class="mx.uam.azc.cbi.logica.local.AdministradorTiposActividadesImpl"/>
        <bean id="administradorCoordinador"
              class="mx.uam.azc.cbi.logica.local.AdministradorCoordinadorImpl"/>
        <bean id="administradorColaborador"
              class="mx.uam.azc.cbi.logica.local.AdministradorColaboradorImpl"/>
        <bean id="administradorSecretario"
              class="mx.uam.azc.cbi.logica.local.AdministradorSecretarioImpl"/>
        <bean id="administradorBecario"
              class="mx.uam.azc.cbi.logica.local.AdministradorBecarioImpl"/>
</beans>
```

Figura 16. Archivo de configuración spring-services.xml

Todos estos servicios estarán configurados dentro de un contexto de aplicación la cual hace referencia a los servicios por medio de nombres definidos por las etiquetas `<bean>`.

La figura 17 muestra la interfaz `AdministradorEvento`, la cual utiliza anotaciones transaccionales para la recuperación de datos, así como los métodos necesarios para la inserción, modificación y borrado de los eventos.

AdministradorEvento.java

```java
@Transactional( rollbackFor=Exception.class )
public interface AdministradorEvento
{

  @Transactional(propagation=Propagation.SUPPORTS, readOnly=true)
  public EventoJoinDTO leerEvento(long id);
  //////////////////////////////////////////////////////////
  @Transactional(propagation=Propagation.SUPPORTS, readOnly=true)
  public List<EventoJoinDTO> leerEventosQuery();
  //////////////////////////////////////////////////////////
  @Transactional( propagation=Propagation.SUPPORTS, readOnly = true )
  public long contarComites();
  //////////////////////////////////////////////////////////
  @Transactional( propagation=Propagation.SUPPORTS, readOnly = true )
  public List<EventoJoinDTO> leerEventos( FiltroEventos filter );
  //////////////////////////////////////////////////////////////////
  @Transactional( propagation=Propagation.SUPPORTS, readOnly = true )
  public List<EventoJoinDTO> buscarEventosCriterioNombre( FiltroEventos filter );
  //////////////////////////////////////////////////////////////////
  @Transactional( propagation=Propagation.SUPPORTS, readOnly = true )
  public List<Object[]> leerEventosActividades( );
  //////////////////////////////////////////////////////////////////////
  @Transactional( propagation=Propagation.SUPPORTS, readOnly = true )
  public List<String> leerActividadesDescripcion( );
  //////////////////////////////////////////////////////////////////////
  public long insertarEvento(EventoDTO evento);
  public void actualizarEvento(EventoDTO evento);
  public void borrarEvento(long id);

```

Figura 17. Interfaz `AdministradorEvento`.

El servicio administradorEvento utiliza la clase de implementación `AdministradorEventoImpl` (figura 18), contenida en el paquete `mx.uam.azc.cbi.logica.local`, la cual hereda todas las propiedades y métodos definidos de la interfaz `AdminitradorEvento`, pero en la implementación se define la funcionalidad de cada uno de esos métodos.

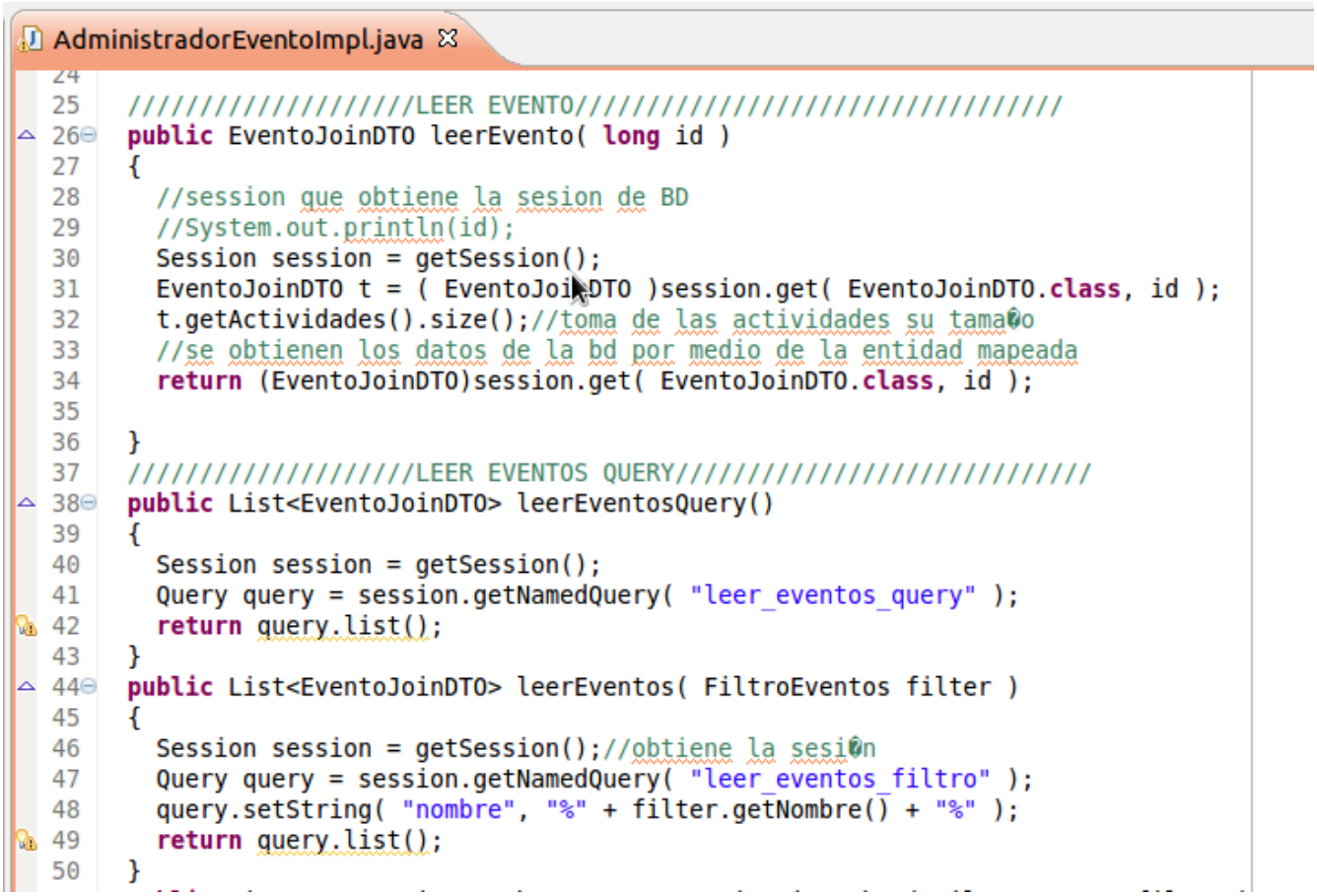


Figura 18. Clase `AdministradorEventoImpl`.

La capa modelo es capaz de gestionar los componentes que conforman a la lógica del negocio. En la capa de datos se gestionan las clases que se utilizarán para el mapeo de los datos; estas clases mejoran el manejo de los datos para las clases *Action* y para las *Interfaces;* estas últimas ejecutan las operaciones necesarias que se pueden realizar en la base de datos, tales como inserción, borrado, actualización y consulta de los datos. La figura19 muestra el diagrama de clases que componen el paquete de la capa de datos. Para consultar la implementación de las operaciones de estas clases, referirse a la documentación de Hibernate [5].

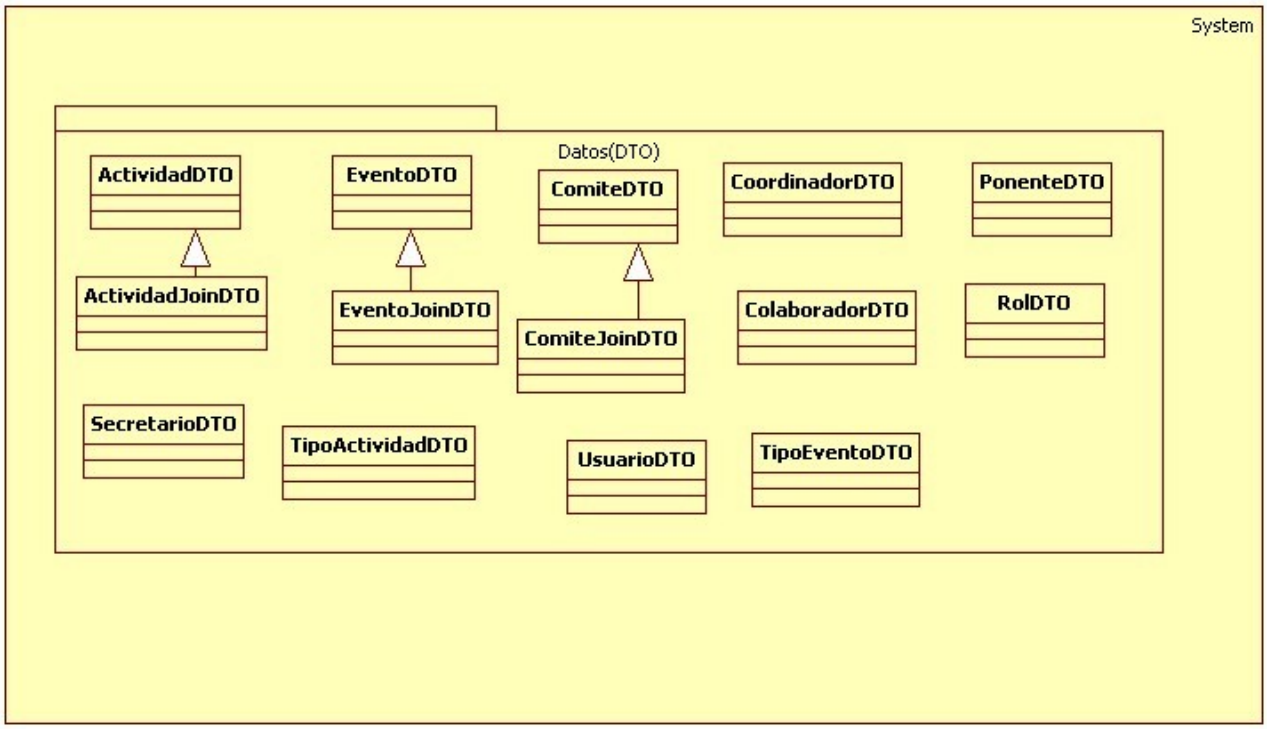


Figura 19. Diagrama de clases de capa de datos.

Para el manejo de los datos en el sistema, la aplicación usa una relación con objetos aplicando el mapeo objeto-relación (OR Mapping - `Object-Relational Mapping`) el cual es una técnica utilizada en aplicaciones java. Para el mapeo de estos objetos que se relacionan con las tablas definidas en la base de datos se utilizó el framework *`Hibernate3`* el cual soporta el mapeo de datos por JPA[4] basado en anotaciones; esta *framework* integra escalabilidad, desempeño, mecanismos de persistencia, flexibilidad y diversidad en la relación de mapeos, enlazado de componentes y reconexión con el mecanismo de persistencia [6].

La configuración de `Hibernate` y del manejo de transacciones está definido en el archivo de configuración `spring-dev.xml` en donde se declaran los objetos utilizados para el mapeo a la base de datos los cuales están definidos con un sufijo DTO[5] y el dialecto que se utilizará al momento de llevar a cabo alguna consulta a la base de datos; también se define el servicio que se encargará de las transacciones y las sesiones que se utilizarán cuando se lleven a cabo el manejo de los datos, a través de los DTO que son objetos de datos que simulan las tablas y los datos que están en la base de datos (figura 19).

Los objetos DTO actúan como entidades JPA que utilizan métodos `get()` y `set()` para el manejo de los datos utilizando un mecanismo de persistencia de mapeo orientado a los campos de la base de datos relacionado con las propiedades de los objetos DTO.

4 Java Persistence API
5 Data Transfer Object

La figura 20 muestra la clase `EventoJoinDTO` la cual corresponde a una entidad de la tabla `evento` de la base de datos del sistema.

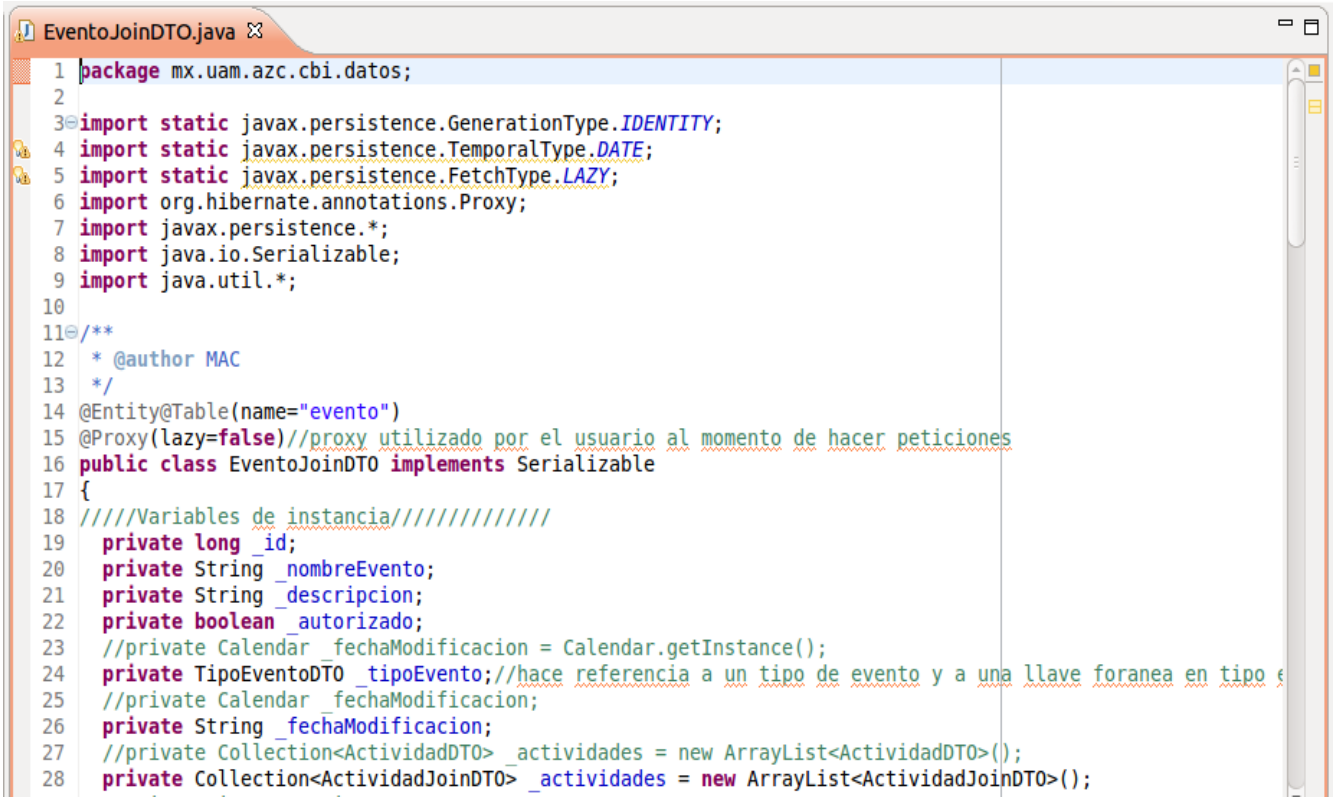


Figura 20. Clase `EventoJoinDTO`

Esta clase muestra las propiedades que serán mapeadas con la tabla `evento`, las cuales son similares en cuanto a la estructura de la tabla; es decir, que cuenta con un `id`, un `nombre`, una `descripción`, una `autorización`, el `tipo del evento`, una `fecha de modificación` y las `actividades` que constituyen al evento.

Las propiedades se manejan a través de métodos `get()` y `set()` de cada uno y estos métodos son mapeados por medio de anotación que contienen el nombre de los campos que están en la tabla que en este caso es la tabla `evento` (figura 21).

EventoJoinDTO.java

```
32  //////////////Metodos accesores/////////
33  //id autoincremental por la BD
34  @Id@Column(name="id_evento")@GeneratedValue(strategy=IDENTITY)
35    public long getId()
36  {
37    return _id;
38  }
39  //en el diagrama es nombreEvento
40  @Column(name="nombre")
41  public String getNombreEvento()
42  {
43    return _nombreEvento;
44  }
45  @Column(name="descripcion")
46  public String getDescripcion()
47  {
48    return _descripcion;
49  }
50  @Column(name="autorizado")
51  public boolean isAutorizado()
52  {
53    return _autorizado;
54  }
55  @Column(name="fecha_mod")
56  public String getFechaModificacion()
57  {
58    return _fechaModificacion;
59  }
60  @OneToOne@JoinColumn(name="tipo_evento")
61  public TipoEventoDTO getTipoEvento()
62  {
63    return _tipoEvento;
64  }
```

Figura 21. Métodos accesores de la clase `EventoJoinDTO`.

Por último la generación de los resultados se plasma en páginas jsp las cuales utilizan las etiquetas `struts-tags` que funcionan como enlazadoras de las acciones del sistema y los objetos DTO al momento de ejecutar las acciones correspondientes.

Los `actions` manejan los DTO como propiedades a ser mostradas al momento de ejecutar una acción en particular, estos DTO son manejados por medio de los administradores de servicios definidos en spring, llevando a cabo las transacciones pertinentes.

La figura 22 muestra la pagina `ver_evento.jsp` la cual desplegará en la pantalla mediante la cadena de resultado "`ver`" enviada por el método `ver()` de la acción `EventoAction` (figura 23) las propiedades que el método maneja, esto es, los eventos que se han generado en el sistema.

ver_evento.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

<html>
  <head>
    <title>Eventos</title>
  </head>
  <body>
    <table id="tabla_eventos" border="0" cellspacing="0" cellpadding="0" width="100%">
      <tr>
        <td align="center" valign="top" width="150" class="menu">
        </td>
        <td width="25">
        </td>
        <td valign="top" align="center" width="625">
          <div class="section">
            Eventos académicos
          </div>
              <hr>
            <s:form action="evento_actualizar_evento" method="post" namespace="/sistema">
              <table id="tabla_evens" align="center" width="80%" border="1">
              <tr>
                <td  class="form" align="center" colspan="4">
                  <div class="label">Eventos</div>
                </td>
              </tr>
              <tr>
                <td class="form" align="center"><div class="label">id</div></td>
                  <s:hidden name="evento.id"/>
                  <td>${ evento.id }</td>
              </tr>
              <tr>
                <td  class="form" align="center"><div class="label">nombre</div></td>
                <td>${ evento.nombreEvento }</td>
```

Figura 22. Página `ver_evento.jsp`.

EventoAction.java

```
57  public String ver()
58  {
59   //se verifica si el id esta vacio y si lo esta se inicializa con 1
60    if( _id == 0)
61      _id = ((Long) _sessionData.get("idEventos")).longValue();
62    System.out.println("Id recibido del evento: "+ _id);
63  //se obtiene la entidad mapeada por medio del id del evento
64    try
65    {
66      _evento = _administradorEvento.leerEvento( _id);
67      System.out.println( _evento.getNombreEvento()+". que cuenta con: "+ _evento.getActividades().size()+"Actividades academicas");
68    }
69    catch(Exception ex)
70    {
71      System.out.println(ex.fillInStackTrace());
72    }
73    //se guarda el id de la session para que no se pierda y se guarda con una llave
74     _sessionData.put("idEventos",  _id );
75    if( _evento.getActividades().isEmpty())
76    {
77      addActionMessage( "El evento no cuenta con actividades" );
78    }
79    //Si tiene actividades las recuperamos y guardamos en una colecci?n de actividades
80    Collection<ActividadJoinDTO> activs =  _evento.getActividades();
81    //despu?s las metemos en un iterador de tipo ActividadJoinDTO
82    Iterator<ActividadJoinDTO> iter = activs.iterator();
83    //mientras haya actividades
84    while(iter.hasNext())
85      {
86      //se almacenan las actividades dentro de una lista de actividades para ser mostradas
87       _actividades.add( iter.next() );
88    }
89    return "ver";
90  }
```

Figura 23. Método `ver()` de la clase `EventoAction`

EJECUCIÓN DE LA APLICACIÓN

A continuación se mostrará la ejecución de la aplicación en un explorador web, esta ejecución mostrará las diferentes fases y opciones que el sistema lleva a cabo y los resultados obtenidos del desarrollo de la misma. La figura 24 muestra la ejecución del servidor `Apache Tomcat` en el IDE de desarrollo `Eclipse Ganymede`.

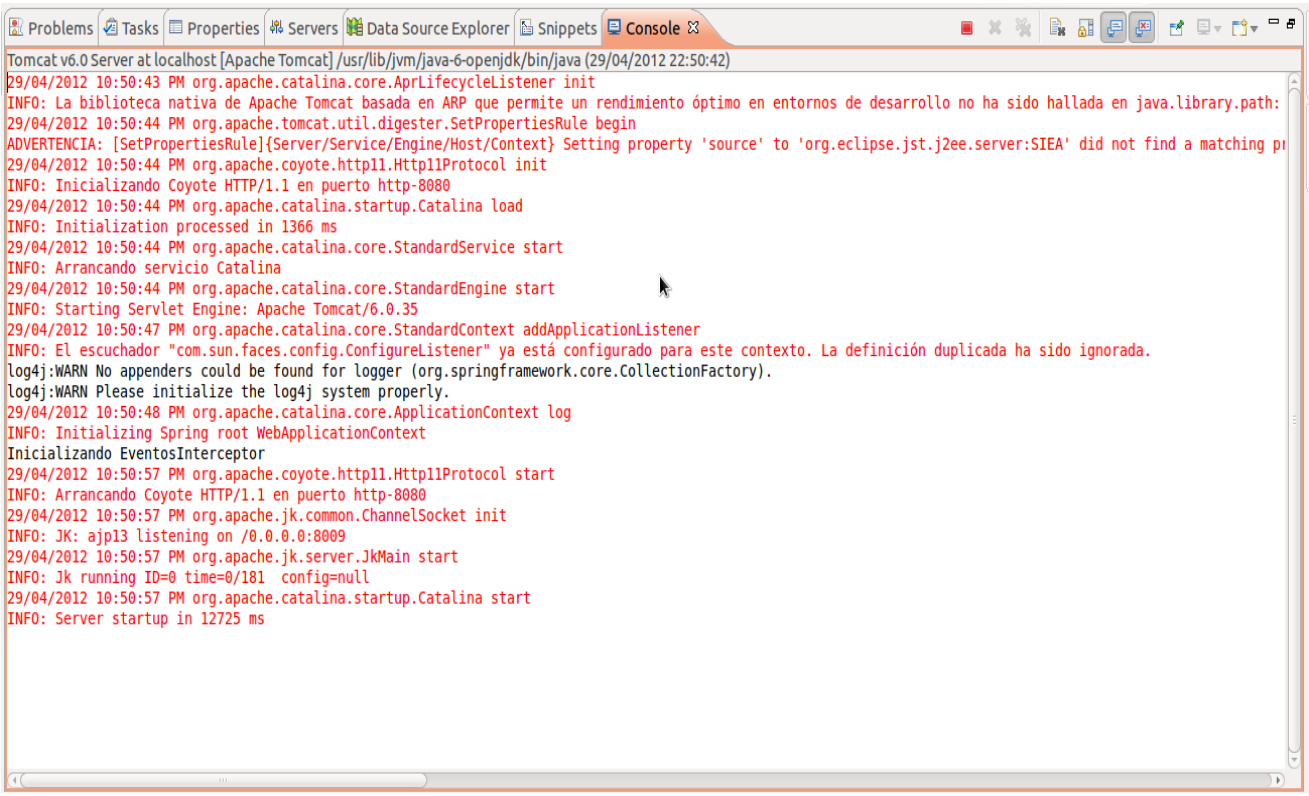


Figura 24. Inicialización del servidor `Apache Tomcat`

En la figura 25 se muestra la pagina de inicio del sistema, en esta pagina se hace la petición de la acción `forma_login.action` la cual al ser invocada despliega la pagina de inicio `forma_login.jsp` en la cual se podrá ingresar un usuario y contraseña para poder ingresar al menú correspondiente.

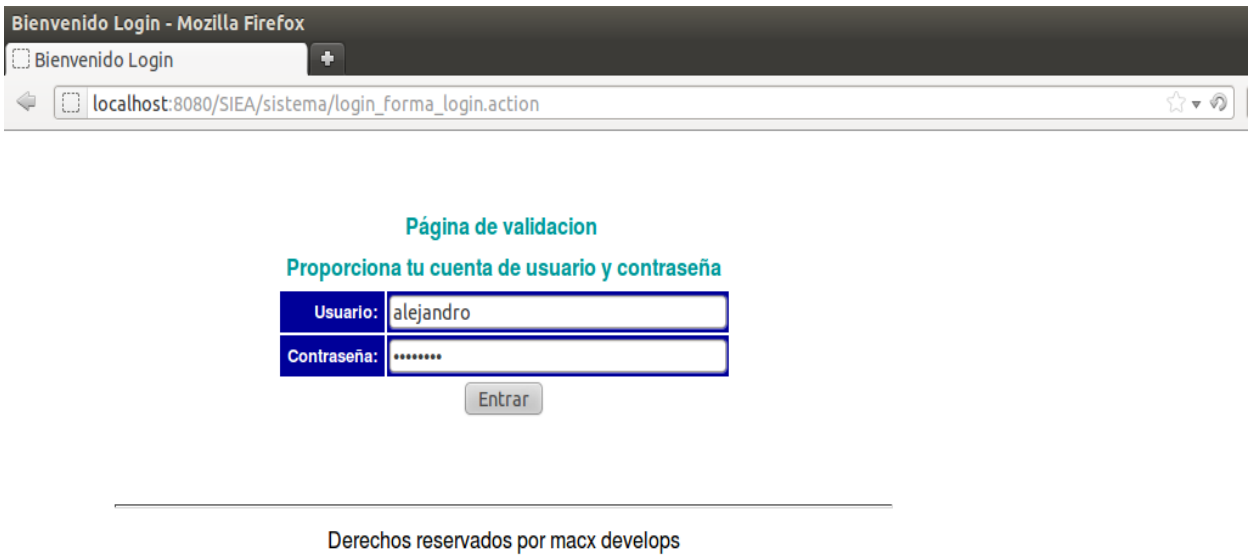


Figura 25. Página de inicio `forma_login.jsp`

Como resultado de la validación del usuario administrador, se desplegará la página de `menú` (página 26) correspondiente al método del mismo nombre del action de evento académico, en la página desplegada el usuario cuenta con las opciones para visualizar los "`eventos académicos`", el "comité `organizador`" o "`salir`" del sistema.

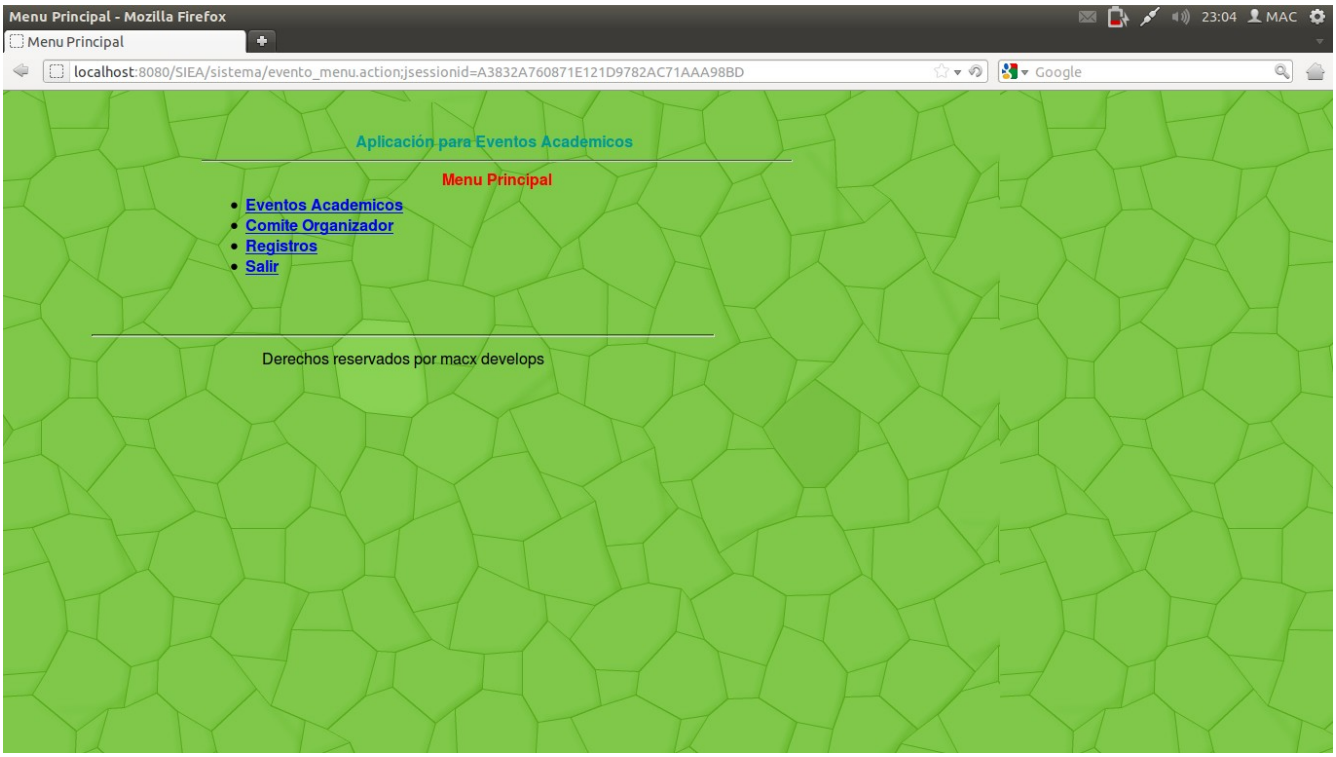


Figura 26. Página de menú principal

Al ser seleccionada la opción de "Eventos Académicos" el usuario podrá visualizar los eventos académicos (figura 27) generados en el sistema.

Figura 27. Listado (`evento_listar.action`) de eventos académicos

Estos eventos son desplegados por medio del método `listar` de la clase `EventosAction`. El usuario tendrá las opciones de consultar, borrar o insertar un evento académico. La opción de consultar el evento se lleva a cabo seleccionando el `Id` del evento académico que se quiere consultar (figura 28), la visualización muestra las características particulares del evento académico, como son: `nombre`, `fecha de inicio`, `tipo de evento`, `autorización`, una breve `descripción` y las `actividades` (si es que las hay) que contiene dicho evento.

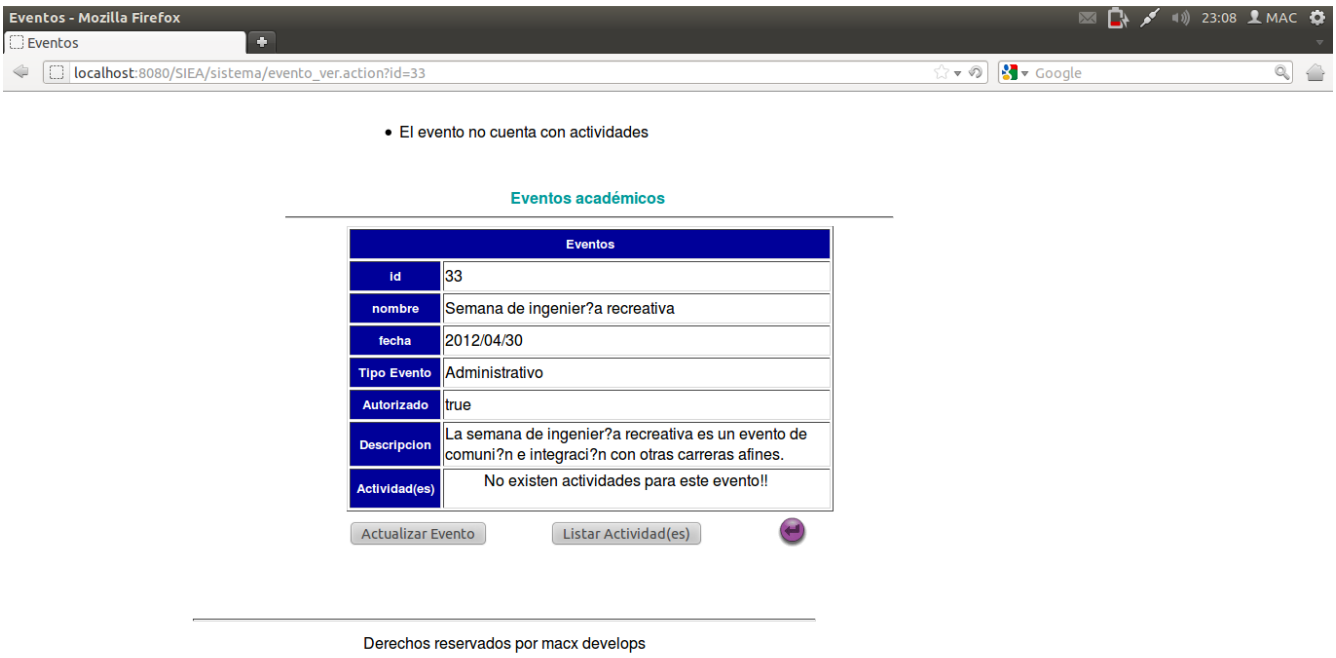


Figura 28. Método `evento_ver.action`.

Esta misma opción contiene el método que actualizara el evento académico (figura 29) y el listado de actividades que se llevarán a cabo en el evento en cuestión.

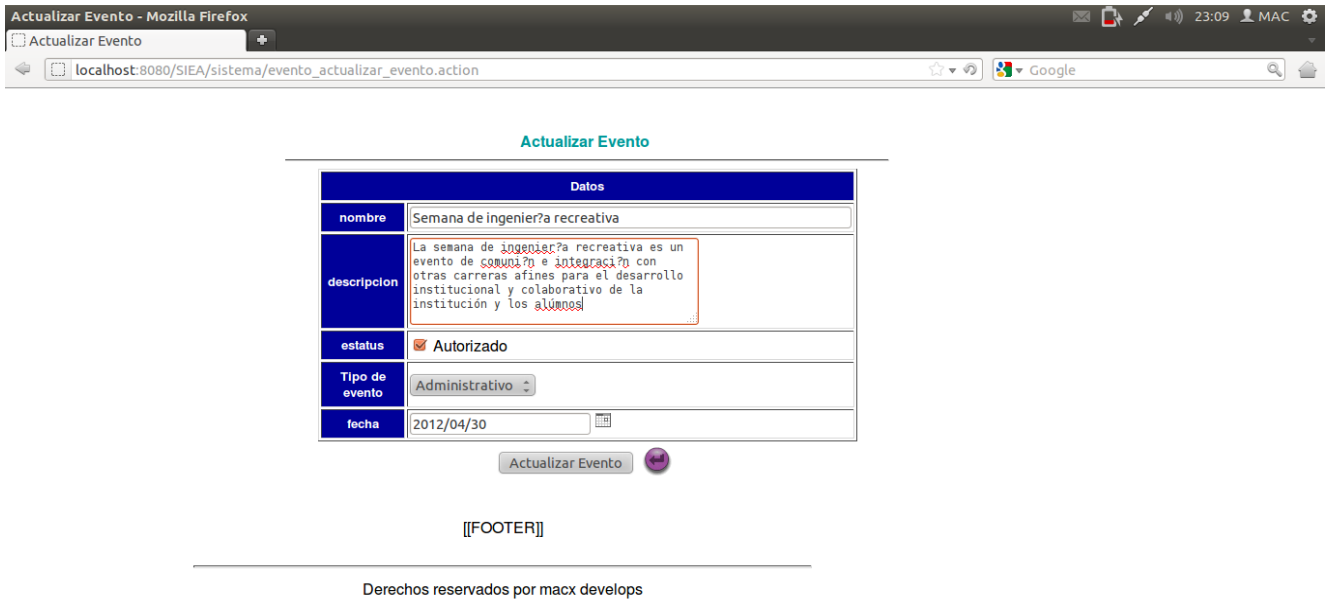


Figura 29. Método `actualizar_evento.action`.

Al dar click en el botón de insertar nuevo evento, el sistema hará referencia al método `evento_insertar_evento.action` (figura 30) dando como resultado la página correspondiente que contendrá un formulario para ingresar un evento académico con las siguientes características: `nombre`, `descripción`, `estatus`, `tipo de evento` y `fecha`. El formulario mandará los datos insertados al action `insertar.action`, el cual, a su vez, los ingresa al sistema reflejando el nuevo evento (figura 31).

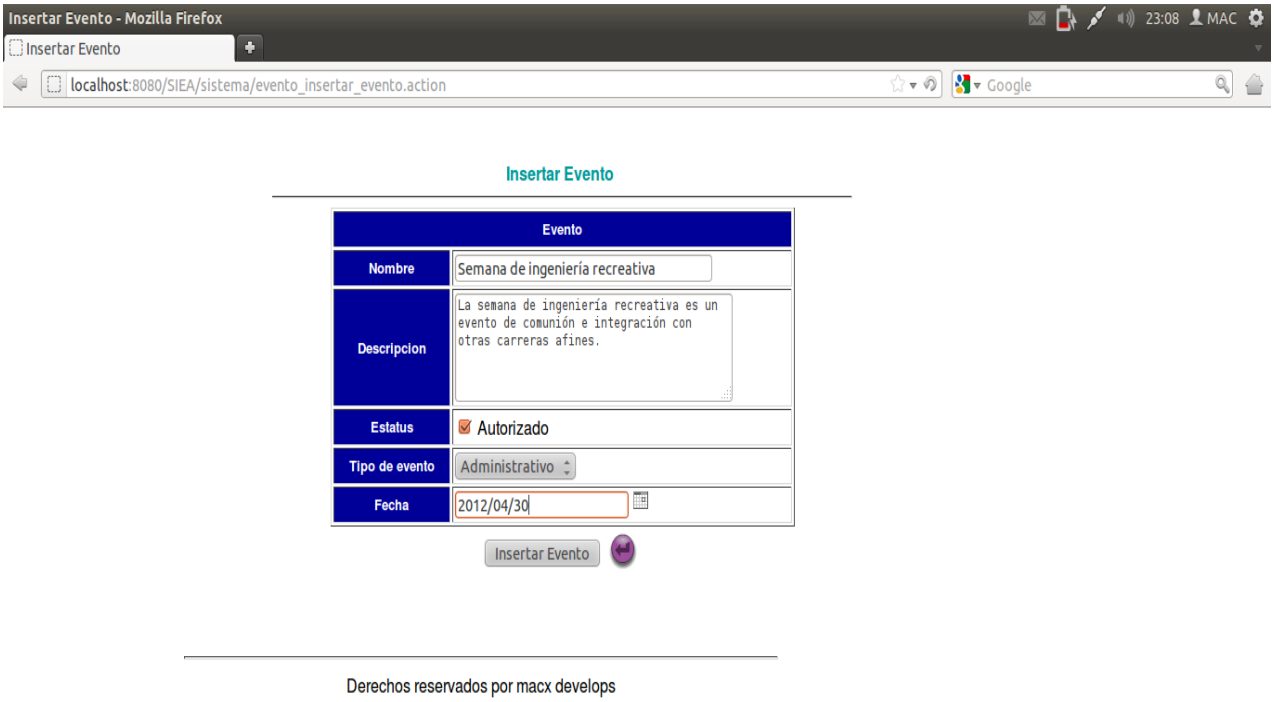


Figura 30. Método `evento_insertar_evento.action`.

Figura 31. Nuevo evento “Semana de ingeniería recreativa”

Por último, para poder borrar algún evento académico el método `borrar_evento.action` (figura 32), el cual está señalizado por la imagen de un tache borrará un evento académico por medio del `id` del evento en cuestión y se reflejará en el método `evento_listar` (figura 33)

Figura 32. Método `evento_borrar.action`.

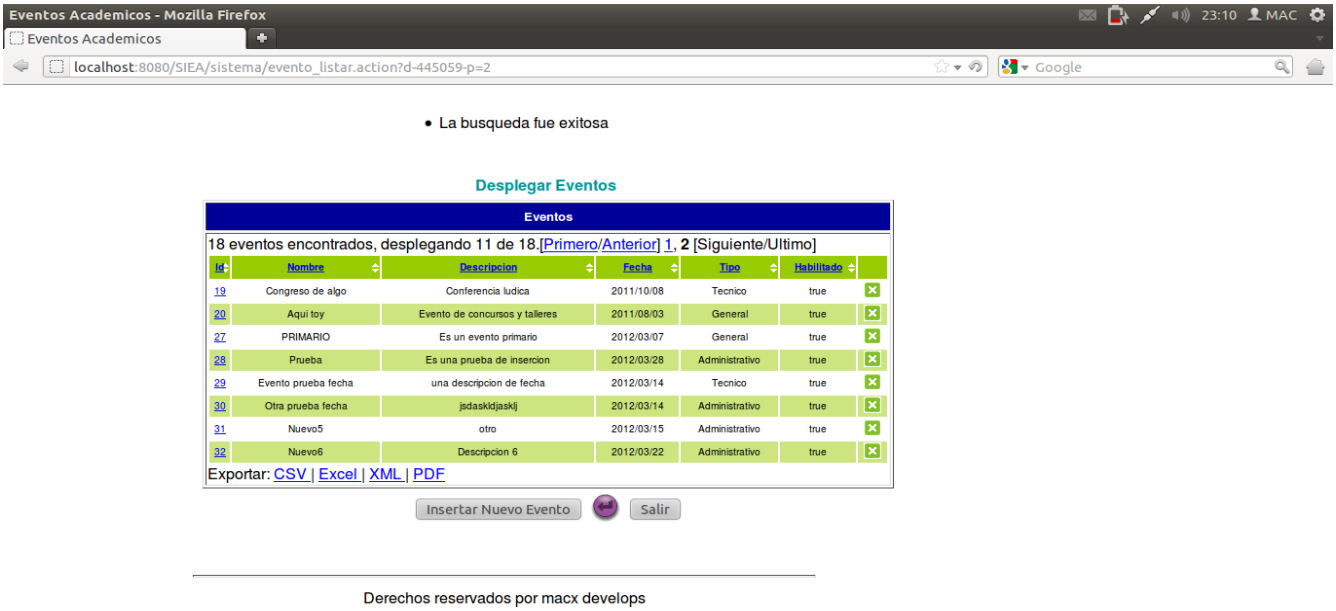


Figura 33. Método `evento_listar.action`.

Para las opciones de los módulos de gestión de Actividades Académicas y gestión del Comité Organizador se utilizan los mismos métodos de las clases `Action`; la tabla 2 enuncia la acción sobre el sistema y el método que lo materializa.

Método	Descripción
actividad_ver.action	Consulta de actividades referidas a un evento
actividad_listar.action	Listado de actividades
actividad_insertar.action	Inserción de actividades
actividad_borrar.action	Eliminación de actividad

Tabla 2. Métodos módulo Actividad.

También para la generación de ponentes con los que cuenta la actividad, el sistema está habilitado para generar esta opción de la misma manera que para los módulos antes mencionados.

CONCLUSIONES

La aplicación web "sistema de inscripción de eventos académicos", que se desarrolló como proyecto terminal, estará habilitada para poder administrar los eventos académicos que se lleven a cabo en la unidad Azcapotzalco; en primera instancia para la División de Ciencias Básicas e Ingeniería. Los eventos académicos que se propongan contendrán actividades tales como: talleres, conferencias magistrales, actividades lúdicas y concursos; además, permite definir un comité organizador, el cuál tendrá la gestión de un evento en particular. La tabla 3 muestra los objetivos específicos, así como el alcance que se obtuvo con el desarrollo de este proyecto terminal.

	Objetivos	Descripción
✔	Definir la arquitectura del sistema de gestión para eventos académicos.	Se cumplió con la definición de la arquitectura en base al modelo MVC.
✔	Diseñar los módulos para la gestión de la información de Talleres, Conferencias y Comité Organizador.	Los módulos que gestionarán la información fueron diseñados satisfactoriamente, ubicándolos cada uno en su respectivo caso de uso y diagramas en conjunto.
✔	Implementar los módulos anteriormente mencionados.	Los módulos se implementaron mediante el framework Struts2 correctamente.
✔	Probar la funcionalidad de los módulos.	Realizamos pruebas sobre las operaciones de inserción, consulta, modificación y borrado, correspondientes a cada uno de los módulos.
➢	Persistencia de los datos.	Se utilizaron archivos XML para configurar los objetos que se mapearon

		desde la base de datos.

Tabla 3. Objetivos alcanzados.

- ✔ Objetivo alcanzado.
- ➢ Objetivo alcanzado no propuesto (objetivos extras).

La realización de este proyecto me permitió reafirmar mis conocimientos del lenguaje Java, y de implementar basado en el framework Struts2; además de los frameworks que complementan a Struts como son Hibernate y el manejo de Spring.

Considero que el avance sobre la aplicación “SIEA” fue sustancial; sin embargo, aún hay funcionalidades por desarrollar, a saber:

- Generar Registro y Pre-registo de usuarios.
- Generar reportes históricos y actuales de los eventos académicos llevados a cabo.

BIBLIOGRAFIA

[1] Proyecto Terminal en Ingeniería en Computación, “SISTEMA DE INSCRIPCIÓN PARA LA SEMANA DE INGENIERÍA EN COMPUTACIÓN 2007”, Participantes: Daniel Mendoza Moreno e Ignacio Augusto Pérez Segura, Responsable: María Lizbeth Gallardo López, UAM-Azcapotzalco, División de Ciencias Básicas e Ingeniería, 07O.

[2] Monroy Rodriguez Guillermo, "Modelo de datos para el sistema de gestión de eventos académicos (SGEA)", artefacto generado como parte del proyecto de servicio social titulado "Sistema de gestión de eventos académicos", marzo de 2011.

[3] http://www.hibernate.org/docs

[4] http://www.springsource.org/spring-framework

[5] http://docs.jboss.org/hibernate/orm/4.1/devguide/en-US/html_single/

[6] http://www.hibernate.org/about/why-hibernate

[7] Ingeniería de software, Séptima Edición, Ian Sommerville, Editorial: Pearson Educación, S.A. Madrid 2005.

[8] El lenguaje unificado de modelado, Grady Booch, James Rumbaugh, Ivar Jacoboson, Editorial: Addison-Wesley, Madrid.

[9] Struts, Segunda Edición, J. Antonio Martin Sierra, Editorial: Alfaomega, México 2008