UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

"Sistema para web de administración y publicación de información del Instituto Nacional de Ciencias Penales"

T E S I S P R O F E S I O N A L
QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN
P R E S E N T A :

JOSÉ ANTONIO GONZÁLEZ BALDERAS
OSCAR MONTIEL ALBORNOZ
RODRIGO TORRES DÍAZ

DIRECTOR DE TESIS: ING. CARLOS ALBERTO ROMÁN ZAMITIZ

CIUDAD UNIVERSITARIA, MÉXICO, D.F. AGOSTO DE 2004

Contenido

Introducción..... 1
CAPÍTULO 1. Descripción del problema..... 2
1.1 Situación original..... 2
1.2 Administrador de contenidos..... 3
1.3 Justificación del sistema..... 4
1.4 Descripción de los requerimientos..... 5
1.5 Usuarios del sistema..... 6
CAPÍTULO 2. Herramientas de análisis de diseño..... 7
2.1 Metodología..... 7
2.1.1 La metodología orientada a objetos..... 7
2.1.2 conceptos de la metodología orientada a objetos..... 7
2.1.3 Ventajas de la metodología orientada a objetos..... 11
2.2 UML el Lenguaje Unificado de Modelado..... 13
2.2.1 Los principales beneficios de UML..... 14
2.2.2 UML, ¿método o lenguaje de modelado?..... 14
2.2.3 Elementos del lenguaje UML..... 15
2.2.4 UML y los procesos de desarrollo..... 16
2.2.5 Diagrama de caso de uso (USE-CASE)..... 18
2.2.6 Diagrama de actividades..... 20
2.2.7 Diagrama de secuencias..... 23
2.2.8 Diagrama de colaboración..... 24
2.2.9 Diagrama de clases..... 25
2.2.10 Diagrama de estados..... 27
2.2.11 Diagrama de componentes..... 28
2.2.12 Diagrama de distribución..... 29
2.2.13 Notas y comentarios en UML..... 30
CAPÍTULO 3. Análisis del sitio web del INACIPE..... 31
3.1 Descripción de actores..... 31
3.2 Casos de usos generales..... 32
3.3 Caso de uso: Acceso al sistema..... 32
3.3.1 Diagrama del caso de uso de acceso al sistema..... 32
3.3.2 Escenario del caso de uso de acceso al sistema..... 33
3.4 Caso de uso: Acervo bibliográfico..... 38
3.4.1 Diagrama del caso de uso de acervo bibliográfico..... 38
3.4.2 Escenario del caso de uso del acervo bibliográfico..... 39
3.5 Caso de uso:Convocatorias..... 44
3.5.1 Diagrama de caso de uso de convocatorias..... 44
3.5.2 Escenario del caso de uso de convocatorias..... 45
3.6 Caso de uso: Eventos..... 57
3.6.1 Diagrama de caso de uso de eventos..... 57
3.6.2 Escenario de caso de uso de eventos..... 58

3.7 Caso de uso: Administración de usuarios..... 70
3.7.1 Diagrama de caso de uso de administración de usuarios..... 70
3.7.2 Escenario del caso de uso de administración de usuarios..... 71
CAPÍTULO 4. Diseño del sitio web de INACIPE..... 79
4.1 Diseño de la base de datos..... 79
4.2 Acceso al sistema..... 81
4.2.1 Diagrama de clases del login..... 81
4.2.2 Diagrama de clases del caso de logout..... 83
4.2.3 Diagrama de secuencia y de colaboración del caso de uso login..... 84
4.2.4 Diagrama de secuencia y de colaboración del caso de uso logout..... 87
4.3 Acervo bibliográfico..... 90
4.3.1 Diagrama de clases de Consulta acervo bibliográfico..... 90
4.3.2 Diagrama de clases de lista acervo bibliográfico..... 92
4.3.3Diagrama de secuencia y colaboración de consulta del acervo bibliográfico..... 94
4.3.4Diagrama de secuencia y colaboración de lista del acervo bibliográfico 97
4.4 Convocatorias..... 100
4.4.1 Diagrama de clases de cambia estatus a convocatorias publicadas..... 100
4.4.2 Diagrama de clases de consulta información completa de convocatorias..... 102
4.4.3 Diagrama de clases de lista convocatorias..... 104
4.4.4 Diagrama de clases de modifica datos de convocatorias publicadas..... 106
4.4.5 Diagrama de clases de publicación de convocatorias..... 108
4.4.6 Diagrama de secuencia y de colaboración de cambia estatus a convocatorias publicadas..... 110
4.4.7 Diagrama de secuencia y de colaboración de consulta información completa de convocatorias..... 113
4.4.8 Diagramas de secuencia y de colaboración de lista convocatorias..... 116
4.4.9 Diagrama de secuencia y de colaboración de modifica datos de convocatorias publicadas..... 119
4.4.10 Diagrama de secuencia de publica convocatorias..... 123
4.5 Eventos..... 126
4.5.1 Diagrama de clases de cambia estatus a eventos publicados..... 126
4.5.2 Diagrama de clases de consulta de información completa del evento... 128
4.5.3 Diagrama de clases de lista eventos..... 130
4.5.4 Diagrama de clases de modifica datos a eventos publicados..... 132
4.5.5 Diagrama de clases de publica eventos..... 134
4.5.6 Diagrama de secuencia y de colaboración de cambia estatus a eventos publicados..... 136
4.5.7 Diagrama de secuencia y de colaboración de consulta información completa del evento..... 139
4.5.8 Diagrama de secuencia y de colaboración de lista de eventos..... 142
4.5.9 Diagrama de secuencia y de colaboración de modifica datos a eventos publicados..... 145
4.5.10 Diagrama de secuencia y de colaboración de publica eventos..... 149

4.6 Administración de usuarios..... 152
4.6.1 Diagrama de clases de baja de usuarios..... 152
4.6.2 Diagrama de clases de alta de usuarios..... 154
4.6.3 Diagrama de clases de lista de usuarios..... 156
4.6.4 Diagrama de clases de modifica usuarios..... 158
4.6.5 Diagrama de secuencia y de colaboración de baja de usuarios..... 160
4.6.6Diagrama de secuencia y de colaboración de alta de usuarios..... 163
4.6.7 Diagrama de secuencia y de colaboración de lista de usuarios..... 166
4.6.8 Diagrama de secuencia y de colaboración de modifica de usuarios..... 169
CAPÍTULO 5. Arquitectura tecnológica..... 172
5.1 Sistemas operativos..... 172
5.1.1 Unix..... 172
5.1.2 Solaris..... 173
5.2 Patrones..... 173
5.2.1 Modelo vista controlador (Model View Controller MVC))..... 174
5.3 Lenguaje de programación..... 175
5.3.1 Java..... 175
5.3.2 La Máquina Virtual de Java (Java Virtual Machine)..... 176
5.3.3 Tipos de programas en Java..... 177
5.3.4 Java 2 Enterprise Edition (J2EE)..... 180
5.3.5 Enterprise Java Beans (EJB)..... 181
5.3.6 La arquitectura EJB..... 182
5.3.7 RMI..... 190
5.4 Manejador de base de datos..... 192
5.4.1 Acceso a Bases de Datos en Java (Java Data Base Connectivity JDBC)..... 192
5.4.2 Postgres..... 192
5.5 Servidores de aplicaciones..... 195
5.5.1 Orion..... 198
5.6 Hardware..... 199
5.6.1 Características Técnicas del Hardware..... 200
CAPÍTULO 6. Desarrollo del sitio web del INACIPE..... 206
6.1 Implantación de la base de datos..... 206
6.2 Desarrollo de interfaces..... 212
6.3 Desarrollo de clases..... 225
6.4 Diseño gráfico de interfaces..... 228
6.4.1 Características y especificaciones de diseño de interfaz para la página web del Instituto..... 228
6.4.2 Herramientas de diseño y construcción utilizadas para la página web del Instituto..... 230
CAPÍTULO 7. Pruebas e implantación del sistema..... 234
7.1 Esquema de pruebas..... 234
7.1.1 Reporte de errores..... 235
7.2 Pruebas realizadas al sistema..... 236
7.2.1 Acceso al sistema..... 237

7.2.2 Acervo bibliográfico..... 239
7.2.3 Eventos..... 242
7.2.4 Convocatorias..... 246
7.2.5 Administración de usuarios..... 251
7.3 Instalación en servidores de producción..... 258
7.3.1 Instalación de la base de datos..... 258
7.3.2 Instalación en el servidor de aplicaciones..... 264
7.3.3 Instalación en el servidor web..... 265
Conclusiones..... 267
Bibliografía..... 269

CAPÍTULO 1. Descripción del problema

1.1 Situación original

El Instituto Nacional de Ciencias Penales es un centro académico que forma, actualiza y especializa a los agentes del Ministerio Público de la Federación, a los peritos profesionales y a otros servidores públicos interesados en la procuración e impartición de justicia, la seguridad pública, la ejecución de sanciones, la criminología, la criminalística, la victimología y, en general, en las ciencias penales. Asimismo, imparte estudios de posgrado dirigidos a las personas interesadas en estas ciencias, realiza investigaciones y promueve actividades de difusión.

En el rubro de publicaciones se busca fortalecer la cultura de la legalidad mediante la difusión permanente de temas especializados en Ciencias Penales. Dar a conocer los resultados de la labor de investigadores nacionales y extranjeros. El Instituto cuenta con un amplio trabajo editorial al publicar libros y revistas especializados, lo que incrementa la información bibliográfica en las áreas de las Ciencias Penales.

Con esta política editorial todas las obras y revistas publicadas por el Instituto se ponen al alcance de las áreas de seguridad pública, de procuración, de administración de justicia y de ejecución de sanciones, así como de investigadores, profesores, especialistas y técnicos que se desempeñan en el ámbito de las Ciencias Penales, además del público en general. En este sentido la biblioteca (Celestino Porte Petit) pone a disposición un número importante de ejemplares con las obras publicadas por la institución o por diversos autores independientes.

Con la finalidad de facilitar el acceso al acervo bibliográfico, la institución requiere de un medio de publicación que permita a los usuarios de la biblioteca conocer las obras que se tienen disponibles evitando el traslado hasta las instalaciones sin la certeza de encontrar el libro o la revista que se busca.

Las convocatorias generadas por el Instituto (Maestría, Doctorado, Especialización, Cursos de actualización Seminarios, Talleres, Coloquios, Congresos, Jornadas, Diplomados) requerían de una constante actualización y en ocasiones éstas se publicaban con muy poco tiempo antes de comenzar el evento. Debido a esto se necesitaba de un método más rápido de realizar estas operaciones.

Originalmente la Dirección de Servicios Bibliográficos e Informáticos del Instituto publicaba en una página Web estática la información generada.

Las desventajas de una página de contenidos estáticos son:

- El tiempo para la modificación del contenido es mayor
- Pueden haber errores en los estándares definidos para la publicación de la información
- Dependencia de personal

1.2 Administrador de contenidos

El Administrador de contenidos combina, en forma excepcional, la capacidad de administración de contenidos Web, fácil de utilizar, junto con las capacidades de procesos integrados para manejar la creación, aprobación y publicación de los contenidos y de los complejos documentos que se encuentran en diversos sitios Web y en varios formatos e idiomas. El Administrador puede controlar vastas cantidades de contenidos Web dinámicos a través de los sitios distribuidos globalmente, y proporciona capacidades integradas de administración de procesos para garantizar una publicación segura y exacta de tales contenidos en línea. La facilidad de su uso les permite a los usuarios publicar la información rápidamente mientras se le otorga a los Webmasters el control necesario para crear, generalizar y administrar sus sitios.

Al simplificar y perfeccionar la publicación del contenido en línea, el Administrador de contenidos proporciona las herramientas para responder rápidamente a las demandas cambiantes del mercado. La arquitectura orientada a los eventos enlaza los contenidos con los eventos comerciales y asegura que los visitantes del sitio web puedan acceder oportunamente a los contenidos exactos y relevantes, de este modo se fortalece el valor de todas las iniciativas en línea.

1.3 Justificación del sistema

Aprovechar la tecnología informática de punta en coordinación con las necesidades de la Dirección de Servicios Bibliográficos e Informáticos del Instituto Nacional de Ciencias Penales (INACIPE).

Además de coordinar el establecimiento, mantenimiento, servicio y actualización para el acceso y difusión de la información que genere el Instituto.

La página web del Instituto se deberá de construir en un esquema de 4 capas:

- Primera capa. Servicios para el cliente: Páginas Web mostradas en un navegador (browser) para presentación y solicitud de información.
- Segunda capa. Servicios Web: desarrollo de los componentes encargados de la construcción del contenido de las páginas Web y recepción de solicitudes de los clientes Web.
- Tercera capa. Servicios transaccionales: desarrollo de componentes con la funcionalidad y comportamiento transaccional del sistema; conexión y solicitud de altas, bajas cambios y consultas de información a la Base de Datos.
- Cuarta capa. Administración de la información a través del DBMS y programación de triggers y stored procedures.

1.4 Descripción de los requerimientos

A continuación se describen las tareas que se automatizarán dentro de las áreas involucradas y una breve descripción de éstas.

Acervo bibliográfico

Se refiere a la consulta por Internet del acervo de la biblioteca. Las consultas se harán de acuerdo a los siguientes datos:

Nombre	Título Autor Materia	Libro Revista Tesis Video Todos

Como resultado de la consulta se mostrará la ficha bibliográfica del libro, revista, tesis o video.

Convocatorias

Se refiere al mecanismo de publicación de información sobre las diferentes actividades programadas dentro del Instituto, como inscripción a cursos de maestrías, doctorados, talleres y seminarios, en las que pueden participar el público en general. Se deberá realizar un ambiente de administración mediante Web para agregar, modificar, eliminar y consultar la presentación de las convocatorias.

Eventos

Se refiere a la publicación en Internet de los sucesos que se realizan en el Instituto. Se deberá realizar un ambiente de administración mediante Web para dar de alta imágenes de nuevos eventos, eliminar imágenes de eventos y modificar eventos ya publicados (sustitución).

1.5 Usuarios del sistema

DSBeI

Es la Dirección de Servicios Bibliográficos e Informáticos y esta representada por el Director y la Subdirección de Sistemas.

Usuario de Internet

Es la persona que consulta la página del Instituto.

Posgrado

Es encargado de revisar y corregir la correcta publicación del acervo bibliográfico.

CAPÍTULO 2. Herramientas de análisis y diseño

2.1 Metodología

2.1.1 La metodología orientada a objetos

La mayor parte de los métodos consisten, al menos en principio, en un lenguaje y en un proceso para modelar. El lenguaje de modelado es la notación (principalmente gráfica) de que se valen los métodos para expresar los diseños. El proceso es la orientación que nos dan sobre los pasos a seguir para hacer el diseño.

La metodología orientada a objetos presenta características que lo hacen idóneo para el análisis, diseño y programación de sistemas; sin embargo, el análisis de requisitos, que es la relación entre la asignación de software al nivel del sistema y el diseño del software, se quedó atrás por lo que empezaron a surgir diferentes métodos de análisis y diseño orientado a objetos, entre los que destacan los métodos Booch, OOSE (Object Oriented Software Engineering) y OMT (Object Modeling Technique). Para poner fin a la "guerra de métodos" que se presentó en ese momento, se creó el Lenguaje Unificado de Modelado (UML).

2.1.2 Conceptos de la metodología orientada a objetos

La metodología orientada a objetos ha derivado de las metodologías anteriores a ésta, así como los métodos de diseño estructurado realizados que guían a los desarrolladores que tratan de construir sistemas complejos utilizando algoritmos como sus bloques fundamentales de construcción, similarmente los métodos de diseño orientado a objetos han evolucionado para ayudar a los desarrolladores a explotar el poder de los lenguajes de programación basados en objetos y orientados a objetos, utilizando las clases y objetos como bloques de construcción básicos.

Actualmente el modelo de objetos ha sido influenciado por un número de factores no sólo de la Programación Orientada a Objetos (Object Oriented Programming, OOP por sus siglas en inglés). Además, el modelo de objetos ha probado ser un concepto uniforme en las ciencias de la computación, aplicable no sólo a los lenguajes de programación sino también al diseño de interfaces de usuario, bases de datos y arquitectura de computadoras por completo. La razón de ello es, simplemente, que una orientación a objetos nos ayuda a hacer frente a la inherente complejidad de muchos tipos de sistemas.

Un objeto "es cualquier cosa, real o abstracta, acerca de la cual almacenamos datos y los métodos que controlan dichos datos". También se puede definir a un objeto como "una entidad tangible que muestra alguna conducta bien definida".

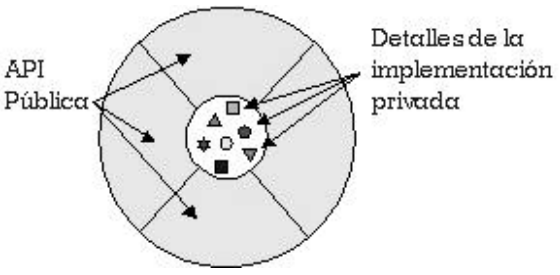
Los objetos tienen una cierta "integridad" la cual no deberá ser violada. En particular, un objeto puede solamente cambiar estado, conducta, ser manipulado o estar en relación con otros objetos de manera apropiada a este objeto.

Una clase es una plantilla para objetos múltiples con características similares. Las clases comprenden todas esas características de un conjunto particular de objetos. Cuando se escribe un programa en lenguaje orientado a objetos, no se definen objetos verdaderos sino se definen clases de objetos.

Una instancia de una clase es otro término para un objeto real. Si la clase es la representación general de un objeto, una instancia es su representación concreta. A menudo se utiliza indistintamente la palabra objeto o instancia para referirse a un objeto.

En los lenguajes orientados a objetos, cada clase está compuesta de dos cualidades: atributos (estado) y métodos (comportamiento o conducta). Los atributos son las características individuales que diferencian a un objeto de otro (ambos de la misma clase) y determinan la apariencia, estado u otras cualidades de ese objeto. Los atributos de un objeto incluyen información sobre su estado.

Los métodos de una clase determinan el comportamiento o conducta que requiere esa clase para que sus instancias puedan cambiar su estado interno o cuando dichas instancias son llamadas para realizar algo por otra clase o instancia. El comportamiento es la única manera en que las instancias pueden hacerse algo a sí mismas o tener que hacerles algo. Los atributos se encuentran en la parte interna mientras que los métodos se encuentran en la parte externa del objeto.



Representación visual de un objeto como componente de software

Para definir el comportamiento de un objeto, se crean métodos, los cuales tienen una apariencia y un comportamiento igual al de las funciones en otros lenguajes de programación como los lenguajes estructurados, pero estos métodos se definen dentro de una clase. Los métodos no siempre afectan a un solo objeto; los objetos también se comunican entre sí mediante el uso de métodos. Una clase u objeto puede llamar métodos en otra clase u objeto para avisar sobre los cambios en el ambiente o para solicitarle a ese objeto que cambie su estado.

Cualquier cosa que un objeto no sabe, o no puede hacer, es excluida del objeto. Además, como se puede observar en la figura, las variables del objeto se localizan en el centro o núcleo del objeto. Los métodos rodean y esconden el núcleo del objeto de otros objetos en el programa. Al empaquetamiento de las variables de un objeto con la protección de sus métodos se le llama encapsulamiento. Típicamente, el encapsulamiento es utilizado para esconder detalles de la puesta en práctica no importantes de otros objetos.

Entonces, los detalles de la puesta en práctica pueden cambiar en cualquier tiempo sin afectar otras partes del programa.

La imagen conceptual de un objeto (un núcleo de variables empaquetadas en una membrana protectora de métodos) es una representación ideal de un objeto y es el ideal por el que los diseñadores de sistemas orientados a objetos luchan. Sin embargo, no lo es todo, a menudo, por razones de eficiencia o la puesta en práctica, un objeto puede querer exponer algunas de sus variables o esconder algunos de sus métodos.

El encapsulamiento de variables y métodos en un componente de software ordenado es, una idea poderosa que provee dos principales beneficios a los desarrolladores de software:

> **Modularidad:** Esto es, el código fuente de un objeto puede ser escrito, así como darle mantenimiento, independientemente del código fuente de otros objetos. Además, un objeto puede ser transferido alrededor del sistema sin alterar su estado y conducta.
> **Ocultamiento de la información:** Es decir, un objeto tiene una "interfaz publica" que otros objetos pueden utilizar para comunicarse con él. Pero el objeto puede mantener información y métodos privados que pueden ser cambiados en cualquier tiempo sin afectar a los otros objetos que dependan de ello.

Los objetos proveen el beneficio de la modularidad y el ocultamiento de la información. Las clases proveen el beneficio de la reutilización. Los programadores de software utilizan la misma clase, y por lo tanto el mismo código, una y otra vez para crear muchos objetos.

En las implantaciones orientadas a objetos se percibe un objeto como un paquete de datos y procedimientos que se pueden llevar a cabo con estos datos. Esto encapsula los datos y los procedimientos. La realidad es diferente: los atributos se relacionan al objeto o instancia y los

métodos a la clase. ¿Por qué se hace así? Los atributos son variables comunes en cada objeto de una clase y cada uno de ellos puede tener un valor asociado, para cada variable, diferente al que tienen para esa misma variable los demás objetos.

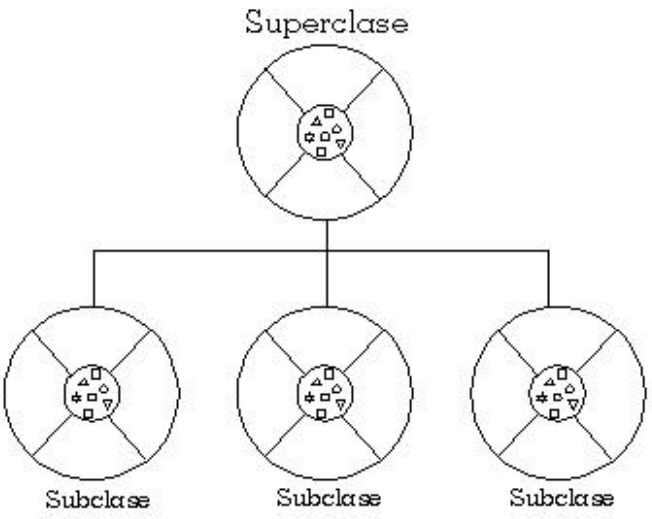
Los métodos, por su parte, pertenecen a la clase y no se almacenan en cada objeto, puesto que sería un desperdicio almacenar el mismo procedimiento varias veces y ello va contra el principio de reutilización de código.

Otro concepto muy importante en la metodología orientada a objetos es el de herencia.

La herencia es un mecanismo poderoso con el cual se puede definir una clase en términos de otra clase; lo que significa que cuando se escribe una clase, sólo se tiene que especificar la diferencia de esa clase con otra, con lo cual, la herencia dará acceso automático a la información contenida en esa otra clase.

Con la herencia, todas las clases están arregladas dentro de una jerarquía estricta. Cada clase tiene una superclase (la clase superior en la jerarquía) y puede tener una o más subclases (las clases que se encuentran debajo de esa clase en la jerarquía). Se dice que las clases inferiores en la jerarquía (clases hijas) heredan de las clases más altas (clases padres).

Las subclases heredan todos los métodos y variables de las superclases. Es decir, si en alguna clase, la superclase define un comportamiento que la clase hija necesita, no se tendrá que redefinir o copiar ese código de la clase padre.



De esta manera, se puede pensar en una jerarquía de clase como la definición de conceptos demasiado abstractos en lo alto de la jerarquía y esas ideas se convierten en algo más concreto conforme se desciende por la cadena de la superclase.

Sin embargo, las clases hijas no están limitadas al estado y conducta provistos por sus superclases; pueden agregar variables y métodos además de los que ya heredan de sus clases padres. Las clases hijas pueden, también, sobrescribir los métodos que heredan por implementaciones especializadas para esos métodos. De igual manera, no hay limitación a un sólo nivel de herencia por lo que se tiene un árbol de herencia en el que se puede heredar varios niveles hacia abajo y mientras más niveles desciendan una clase, más especializada será su conducta.

La herencia presenta los siguientes beneficios:

a) Las subclases proveen conductas especializadas sobre la base de elementos comunes provistos por la superclase.

b) A través del uso de herencia, los programadores pueden reutilizar el código de la superclase muchas veces.

c) Los programadores pueden implementar superclases llamadas clases abstractas que definen conductas "genéricas".

d) Las superclases abstractas definen, y pueden implementar parcialmente, la conducta pero gran parte de la clase no está definida ni implementada. Otros programadores concluirán esos detalles con subclases especializadas.

2.1.3 Ventajas de la metodología orientada a objetos

En síntesis, algunas ventajas que presenta son:

Reutilización. Las clases están diseñadas para que se reutilicen en muchos sistemas. Para maximizar la reutilización, las clases se construyen de manera que se puedan adaptar a los otros sistemas. Un objetivo fundamental de las técnicas orientadas a objetos es lograr la reutilización masiva al construir el software.

Estabilidad. Las clases diseñadas para una reutilización repetida se vuelven estables, de la misma manera que los microprocesadores y otros chips se hacen estables.

El diseñador piensa en términos del comportamiento de objetos y no en detalles de bajo nivel. El encapsulamiento oculta los detalles y hace que las clases complejas sean fáciles de utilizar.

Se construyen clases cada vez más complejas y clases a partir de otras clases, las cuales a su vez se integran mediante clases. Esto permite construir componentes de

software complejos, que a su vez se convierten en bloques de construcción de software más complejo.

Calidad. Los diseños suelen tener mayor calidad, puesto que se integran a partir de componentes probados que han sido verificados y pulidos varias veces.

Un diseño más rápido. Las aplicaciones se crean a partir de componentes ya existentes. Muchos de los componentes están construidos de modo que se pueden adaptar para un diseño particular.

Integridad. Las estructuras de datos (los objetos) sólo se pueden utilizar con métodos específicos. Esto tiene particular importancia en los sistemas cliente-servidor y los sistemas distribuidos, en los que usuarios desconocidos podrían intentar ingresar al sistema.

Mantenimiento más sencillo. El programador encargado del mantenimiento puede cambiar un método de clase a la vez. Cada clase efectúa sus funciones independientemente de las demás.

Independencia del diseño. Las clases están diseñadas para ser independientes del ambiente de plataformas, hardware y software. Utilizan solicitudes y respuestas con formato estándar, lo cual permite ser utilizadas en múltiples sistemas operativos, controladores de bases de datos, controladores de red, interfaces de usuario gráficas, etc. Esto permite que el creador del software no tenga que preocuparse por el ambiente o esperar a que éste se especifique.

Interacción. El software de varios proveedores puede funcionar como conjunto. Un proveedor utiliza clases de otros. Existe una forma estándar de localizar clases e interactuar con ellas. El software desarrollado de manera independiente en lugares ajenos debe poder funcionar en forma conjunta y aparecer como una sola unidad ante el usuario.

Computación Cliente-Servidor. En los sistemas cliente-servidor, las clases en el software cliente deben enviar solicitudes a las clases en el software servidor y recibir respuestas.

Una clase servidor puede ser utilizada por clientes diferentes. Estos clientes sólo pueden tener acceso a los datos del servidor a través de los métodos de la clase. Por lo tanto los datos están protegidos contra su corrupción.

Computación de distribución masiva. Las redes a nivel mundial utilizarán directorios de software de objetos accesibles. El diseño orientado a objetos es la clave para la computación de distribución masiva. Las clases de una máquina interactúan con las de algún otro lugar sin saber donde residen tales clases. Ellas reciben y envían mensajes orientados a objetos en formato estándar.

Migración. Las aplicaciones ya existentes, sean orientadas a objetos o no, pueden preservarse si se ajustan a un contenedor orientado a objetos, de modo que la comunicación con ella sea a través de mensajes estándar orientados a objetos.

Mejores herramientas CASE. Las herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadora) utilizarán las técnicas gráficas para el diseño de las clases y de la interacción entre ellas, para el uso de los objetos existentes adaptados a nuevas aplicaciones. Las herramientas deben facilitar el modelado en términos de eventos, formas de activación, estados de objetos, etc. Las herramientas OO del CASE deben generar un código tan pronto se definan las clases y permitir al diseñador utilizar y probar los métodos recién creados. Las herramientas se deben diseñar de manera que apoyen el máximo de creatividad y una continua afinación del diseño durante la construcción.

2.2 UML el Lenguaje Unificado de Modelado

En todas las disciplinas de la Ingeniería se hace evidente la importancia de los modelos ya que describen el aspecto y la conducta de "algo". Ese "algo" puede existir, estar en un estado de desarrollo o estar, todavía, en un estado de planeación. Es en este momento cuando los diseñadores del modelo deben investigar los requerimientos del producto terminado y dichos requerimientos pueden incluir áreas tales como funcionalidad, desempeño y confiabilidad. Además, a menudo, el modelo es dividido en un número de vistas, cada una de las cuales describe un aspecto específico del producto o sistema en construcción.

El Lenguaje Unificado de Modelado (UML Unified Modeling Language) es un lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software. UML entrega una forma de modelar cosas conceptuales como lo son procesos de negocio y funciones de sistema, además de cosas concretas como lo son escribir clases en un lenguaje determinado, esquemas de base de datos y componentes de software reusables.

UML es una técnica para la especificación de sistemas en todas sus fases. Nació en 1994 cubriendo los aspectos principales de todos los métodos de diseño antecesores y, precisamente, los padres de UML son Grady Booch, autor del método Booch; James Rumbaugh, autor del método OMT e Ivar Jacobson, autor de los métodos OOSE y Objectory. La versión 1.0 de UML fue liberada en Enero de 1997 y ha sido utilizado con éxito en sistemas construidos para toda clase de industrias alrededor del mundo: hospitales, bancos, comunicaciones, aeronáutica, finanzas, etc.

2.2.1 Los principales beneficios de UML

- Mejorar los 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.

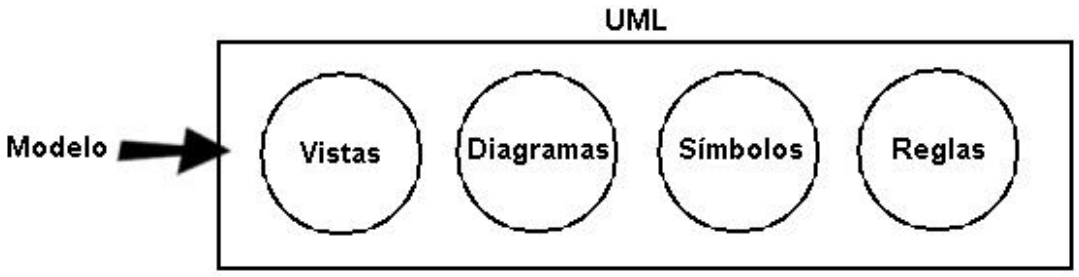
2.2.2 UML, ¿método o lenguaje de modelado?

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 (los símbolos utilizados en los modelos) y un conjunto de mecanismos generales o reglas que indican cómo utilizar los elementos. Las reglas son sintácticas, semánticas y pragmáticas.

2.2.3 Elementos del lenguaje UML



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: diagramas de caso de uso, de clases, de objetos, de estados, de secuencia, de colaboración, de actividad, de componentes y de distribución.

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.

2.2.4 UML y los procesos de desarrollo

El lenguaje UML, estandariza los artefactos las herramientas y la notación, pero no define un proceso oficial de desarrollo debido a que la esencia de un proceso apropiado admite mucha variación y depende de las habilidades del personal, de la naturaleza del problema. De las herramientas y de muchos otros factores.

En un esquema a macro nivel los pasos principales en la presentación de una aplicación son:

- ✦ Planeación y elaboración

  En esta fase se tiene la concepción inicial, la investigación de alternativas, la planeación, la especificación de requerimientos.

  Análisis de Requerimientos

  UML tiene casos de uso (use-cases) para capturar los requerimientos del cliente. A través del modelado de caso 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.

- Construcción

Es la creación del sistema.

Desarrollo Iterativo

El ciclo de vida iterativo se basa en el perfeccionamiento secuencial de un sistema a través de múltiples ciclos de desarrollo de análisis, diseño, implementación y pruebas. En cada ciclo se aborda un conjunto relativamente pequeño de requerimientos. El sistema va creciendo con cada ciclo que se concluye, reduciendo la complejidad y obteniendo una retroalimentación en una etapa temprana sin esperar a que se termine todo el desarrollo.

Programación

En esta fase las clases del diseño son convertidas a código en un lenguaje de programación orientado a objetos.

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.

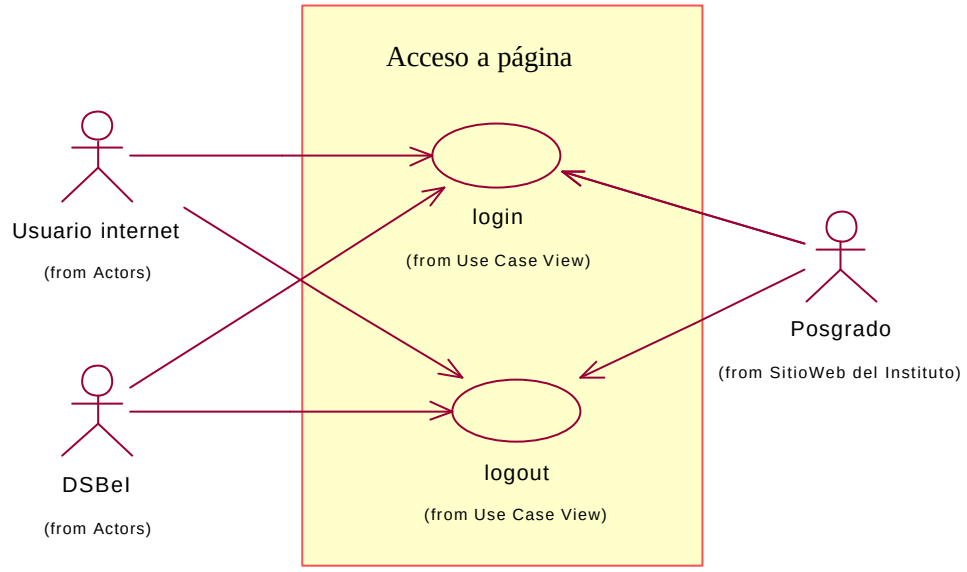
- Aplicación

Es la transición de la implementación del sistema a su uso.

2.2.5 Diagrama de caso de uso (USE-CASE)

Un modelo de caso de uso es descrito en UML como un diagrama de caso de uso (diagrama use-case) y dicho modelo puede ser dividido en varios diagramas de caso de uso. Un diagrama de caso de uso contiene elementos de modelo para el sistema, los actores y los casos de uso y muestra las diferentes relaciones tales como generalización, asociación y dependencia entre estos elementos.

Los elementos de un diagrama de caso de uso son:



= Sistema

Un sistema en un diagrama de caso de uso es descrito como una caja; el nombre del sistema aparece arriba o dentro de la caja. Ésta también contiene los símbolos para los casos de uso del sistema.

= Actores

Un Actor es una entidad externa del sistema que define un conjunto coherente de roles los cuales pueden ser desempeñados por los usuarios. Una actor se considera que juega un rol diferente con relación a cada use case con el cual actúa. Es importante destacar el uso de la palabra rol, pues con esto se especifica que un Actor no necesariamente representa a una persona en particular, sino más bien la labor que realiza frente al sistema. El actor envía a o recibe del sistema unos mensajes o intercambia información con el sistema.

En pocas palabras, el actor lleva a cabo los casos de uso. Un actor puede ser una persona u otro sistema que se comunica con el sistema a modelar.

= Notación del Actor

Usuario internet

El icono estándar de un estereotipo de actor es la figura del "Mono de palo", con el nombre del actor debajo de la figura.

Encontrando a los actores de un diagrama de caso de uso.

Es posible obtener a los actores de un diagrama de caso de uso a través de:

= Caso de uso

Un caso de uso representa la funcionalidad completa tal y como la percibe un actor. Un caso de uso en UML es definido como un conjunto de secuencias de acciones que un sistema ejecuta y que permite un resultado observable de valores para un actor en particular.

Gráficamente se representan con una elipse y tiene las siguientes características:

= Un caso de uso siempre es iniciado por un actor.

= Un caso de uso provee valores a un actor.

= Un caso de uso es completo.

El proceso para encontrar casos de uso inicia encontrando al actor o actores previamente definidos. Por cada actor identificado, hay que realizar las siguientes preguntas:

= ¿Qué funciones del sistema requiere el actor? ¿Qué necesita hacer el actor?

= ¿El actor necesita leer, crear, destruir, modificar o almacenar algún tipo de información en el sistema?

= ¿El actor debe ser notificado de eventos en el sistema o viceversa? ¿Qué representan esos eventos en términos de funcionalidad?

= ¿El trabajo diario del actor podría ser simplificado o hecho más eficientemente a través de nuevas funciones en el sistema? (Comúnmente, acciones actuales del actor que no estén automatizadas)

Otras preguntas que nos ayudan a encontrar casos de uso pero que no involucran actores son:

- ¿Qué entradas/salidas necesita el sistema? ¿De dónde vienen esas entradas o hacia donde van las salidas?
- ¿Cuáles son los mayores problemas de la implementación actual del sistema?

2.2.6 Diagrama de actividades

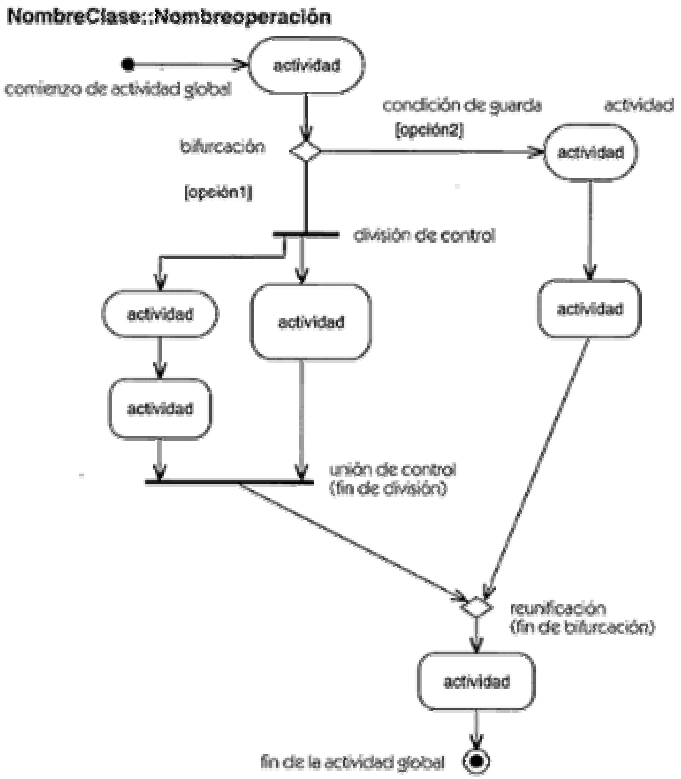
Un estado de actividad representa un paso en el flujo de trabajo o la ejecución de una operación. Las actividades se enlazan por transiciones automáticas. Cuando una actividad termina se desencadena el paso a la siguiente actividad.

Los diagrama de actividades son útiles para entender el comportamiento de alto nivel de la ejecución de un sistema, sin profundizar en los detalles internos de los mensajes. Los parámetros de entrada y salida de una acción se pueden mostrar usando las relaciones de flujo que conectan la acción y un estado de flujo de objeto.

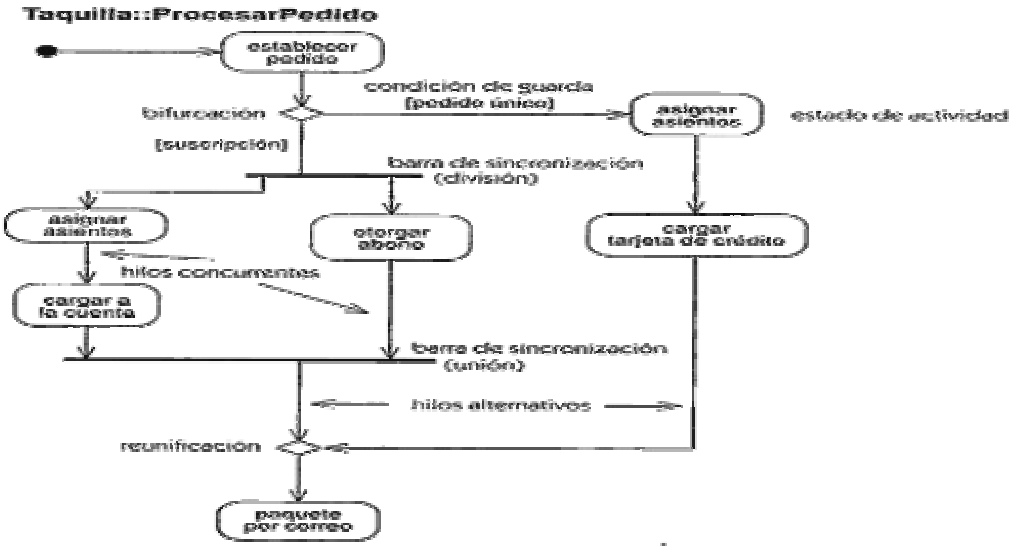
Un grafo de actividades contiene estados de actividad que representa la ejecución de una secuencia en un procedimiento, o el funcionamiento de una actividad en un flujo de trabajo. En vez de esperar un evento, como en un estado de espera normal, un estado de actividad espera la terminación de su cómputo. Cuando la actividad termina, entonces la ejecución procede al siguiente estado de actividad dentro del diagrama. Una transición de terminación es activada en un diagrama de actividades cuando se completa la actividad precedente. Los estados de actividad no tienen transiciones con eventos explícitos, peor pueden ser abortados por transiciones en estados que los incluyen.

Un diagrama de actividades puede contener bifurcaciones, así como divisiones de control en hilos concurrentes. Los hilos concurrentes representan actividades que se pueden realizar al mismo tiempo por los diversos objetos o personas. La concurrencia se representa a partir de la agregación, en la cual cada objeto tiene su propio hilo. Las actividades concurrentes se pueden realizar simultáneamente o en cualquier orden. Un diagrama de actividades es como un organigrama tradicional, excepto que permite el control de concurrencia además del control secuencial.

Un estado de actividad se representa como una caja con los extremos redondeados que contiene una descripción de actividad. Las transacciones simples de terminación se muestran como flechas. Las ramas se muestran como condiciones diamantes con múltiples flechas de salida etiquetadas. Una división o una unión de control se representa con múltiples flechas que entran o salen de la barra gruesa de sincronización.



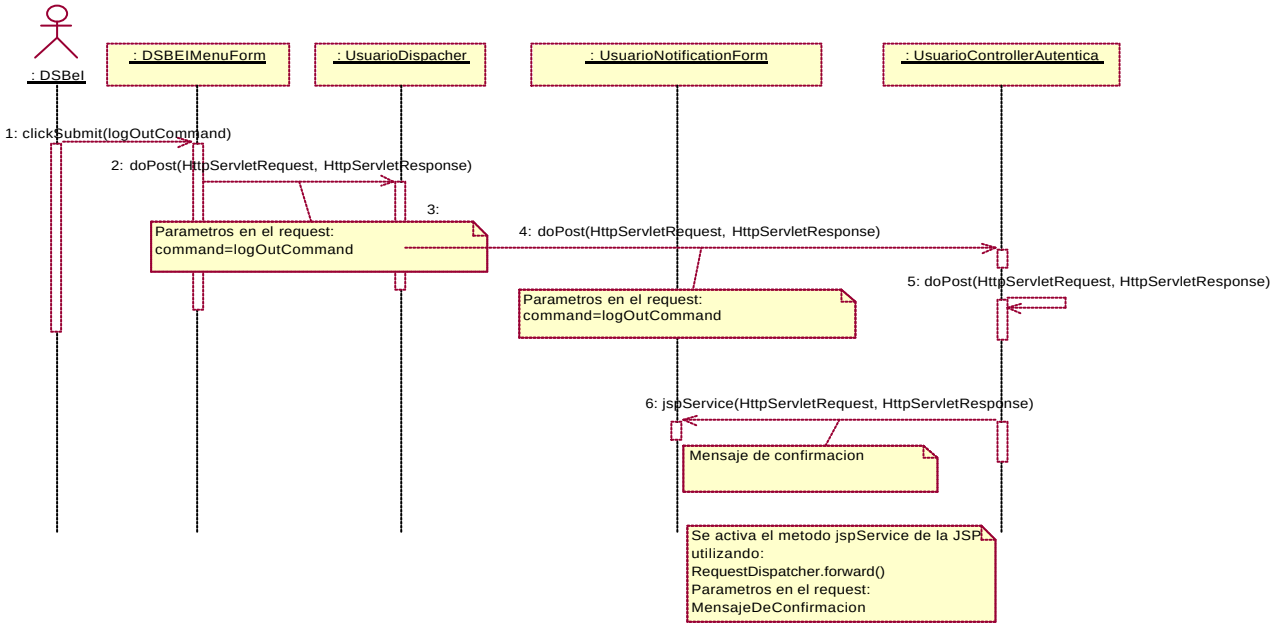
Ejemplo:


Taquilla::ProcesarPedido
establecer pedido
condición de guarda [pedido único]
bifurcación
asignar asientos
estado de actividad
[suscripción]
barra de sincronización (división)
asignar asientos
otorgar abono
cargar tarjeta de crédito
hilos concurrentes
cargar a la cuenta
barra de sincronización (unión)
hilos alternativos
reunificación
paquete por correo

2.2.7 Diagrama de secuencia

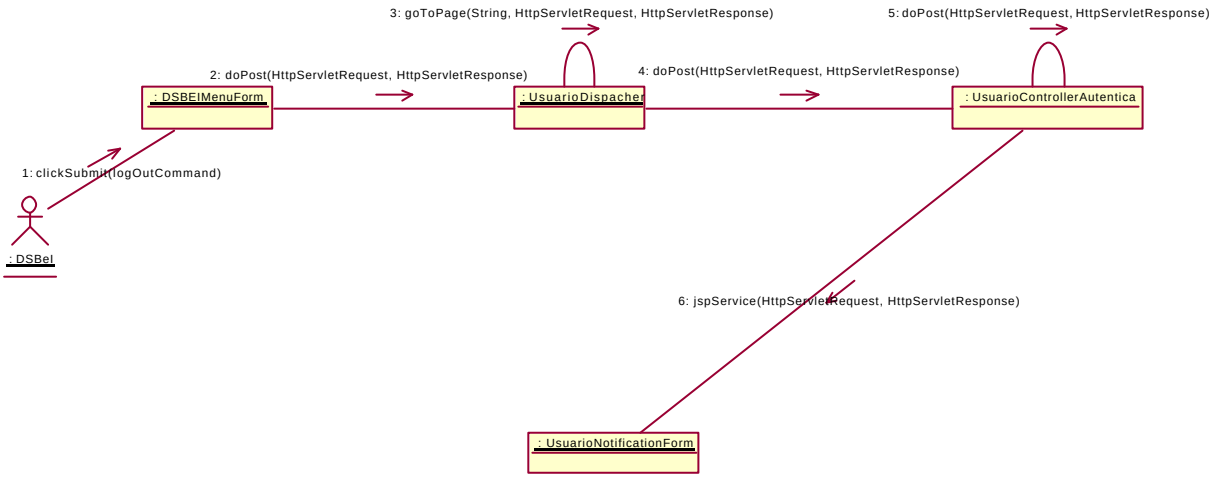
Este diagrama muestra la interacción de los objetos entre ellos. Es importante comentar que hasta este momento no se han considerado objetos técnicos. En UML, durante el análisis de los requerimientos y el análisis, no se consideran objetos técnicos que definan detalles y soluciones en el sistema de software, tales como objetos para interfaces de usuario, bases de datos, comunicaciones, etc. Todos esos objetos se consideran hasta el diseño del sistema.

En este diagrama se observa que sólo se consideran algunos objetos y es importante aclarar que estos no serán todos los objetos a considerar dentro del sistema, ya que todavía es posible agregar nuevos objetos que no se habían considerado en el dominio del análisis así como los objetos técnicos, como se mencionó anteriormente. Los objetos considerados se representan en rectángulos con el nombre subrayado y cada uno cuenta con su línea de vida vertical que muestra la vida del objeto.



2.2.8 Diagrama de colaboración

Así mismo, se cuenta con el diagrama de colaboración, el cual se centra tanto en las interacciones y las ligas entre un conjunto de objetos colaborando entre ellos (una liga es una instancia de una asociación). Ambos, el diagrama de secuencia y el diagrama de colaboración, muestran interacciones, pero el diagrama de secuencia se centra en el tiempo mientras que el diagrama de colaboración se centra en el espacio. Las ligas muestran los objetos actuales y cómo ellos se relacionan unos con otros. Así como los diagramas de secuencia, los diagramas de colaboración pueden ser utilizados para ilustrar la ejecución de una operación, una ejecución de un use-case o simplemente un escenario de interacción dentro del sistema. En este diagrama también se representa a los objetos en cajas rectangulares y con el nombre subrayado. Las ligas se dibujan con líneas y se puede agregar una etiqueta para un mensaje y un número que define la secuencia de las ligas.



2.2.9 Diagrama de clases

Para la realización del diagrama de clases se toman como base los diagramas de secuencia y de colaboración por lo que se manejarán los objetos que ahí se consideraron pero ahora a nivel de clases. Además, se pueden agregar nuevas clases que no se habían considerado y este paso deberá ser realizado por expertos en el dominio del problema. Para poder definir las clases, UML sugiere seis características selectivas que debe utilizar el analista para considerar una clase candidato en el modelo de análisis:

Información retenida. La clase será útil durante el análisis sólo si la información sobre el mismo ha de ser almacenada, transformada, analizada o manejada en algún otro modo. La información puede referirse a conceptos que deberán estar siempre registrados en el sistema, eventos o transacciones que ocurren en un momento específico.

Sistema externo. Si se tiene un sistema externo a este sistema, entonces es de interés en la etapa de modelado. Los sistemas externos deberán ser vistos como clases que el sistema contendrá o con los cuales interactuará.

Patrones, librerías de clases o componentes. Si se tienen patrones, librerías de clases o componentes, generalmente éstos son clases candidatos.

Dispositivos que el sistema maneja. Dispositivos técnicos que maneja el sistema se convertirán en clases que manejarán esos dispositivos.

Partes organizacionales. Especialmente en modelos de negocio, todas las partes que representan a la organización, serán clases candidatos.

Roles de actores. Los roles de actores serán vistos como clases, por ejemplo, usuario, operador del sistema, administrador, cliente, etc.

Gráficamente, las clases se representan en una caja rectangular dividida en 3 compartimentos: en la parte superior se encuentra el nombre de la clase, en la parte media se encuentran los atributos y en la parte inferior se encuentran las operaciones o métodos de la clase.

Las clases se relacionan entre sí a través de las relaciones que son las líneas rectas entre dos clases y constan de roles y cardinalidad. Los roles son las frases que se encuentran en la relación y sirven para definir la cardinalidad de la relación, siempre considerando la unidad en la clase origen. Además, tanto los atributos como los métodos cuentan con una sintaxis. Para los atributos la sintaxis es la siguiente:

visibilidad nombre : tipo = valor_inicial {lista_de_posibles_valores}

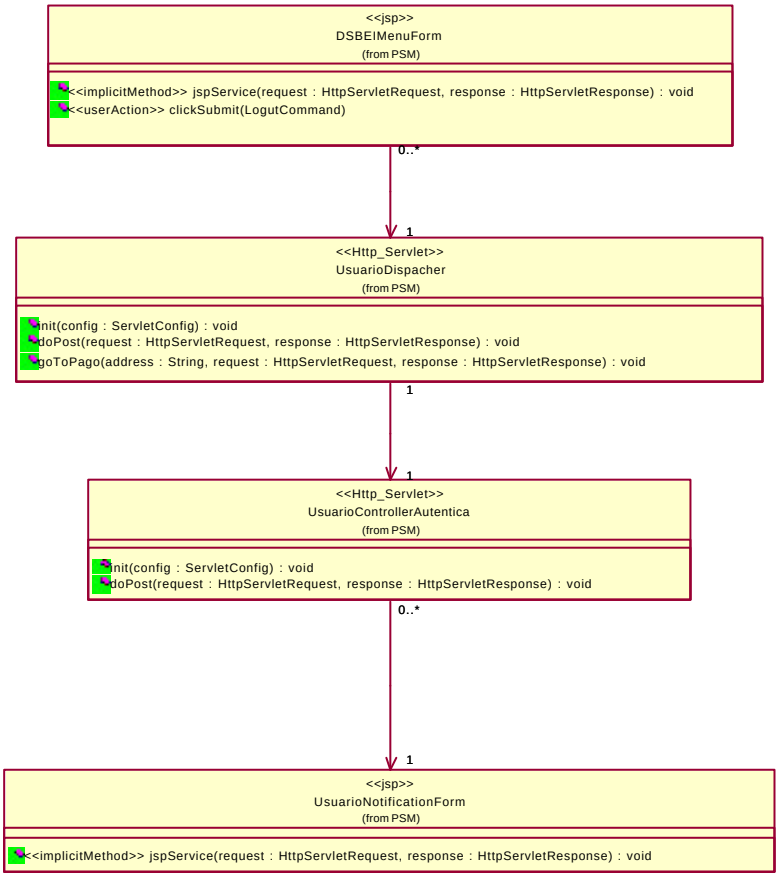
y para los métodos, la sintaxis es la siguiente:

visibilidad nombre (lista_parámetros) : tipo_de_retorno {lista_de_posibles_valores}

donde la lista de parámetros consta de la siguiente sintaxis:

nombre : tipo = valor_default

donde los parámetros van separados por comas y la visibilidad tanto para los atributos como para los métodos, puede ser público o privado y se puede representar gráficamente con un signo + para público y un signo – para privado.



2.2.10 Diagrama de estados

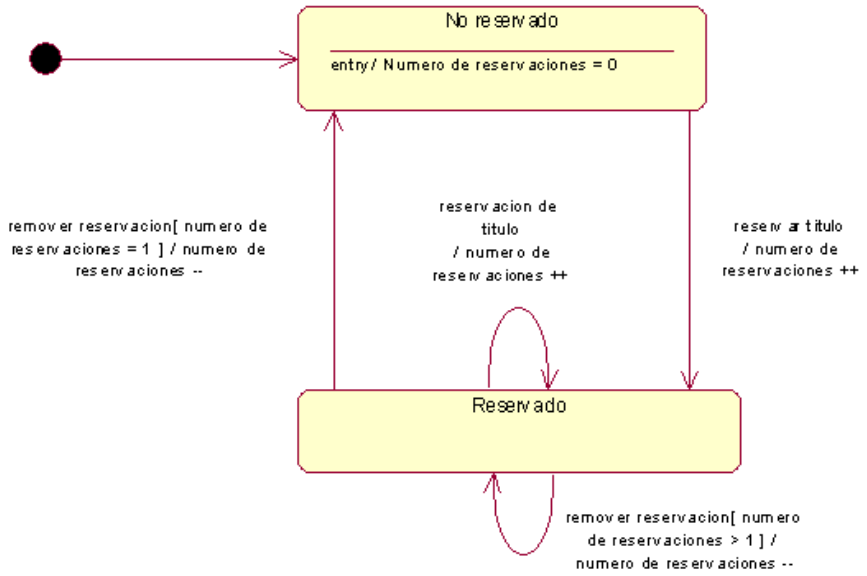
El diagrama de estados captura el ciclo de vida de los objetos, subsistemas y sistemas. Dicho diagrama determina los estados que un objeto puede tener y cómo los eventos afectan esos estados a través del tiempo. Un diagrama de estado debe abarcar todas las clases que tengan estados y conducta definidos claramente.

Todos los objetos tienen un estado y éste es el resultado de actividades previas ejecutadas por el objeto. Ese estado está determinado por los valores de los atributos de este objeto y sus relaciones con otros objetos. Una clase puede tener un atributo que especifique el estado, o el estado puede ser determinado por los valores de los atributos "normales" del objeto.

Gráficamente, los estados se representan en rectángulos con esquinas redondeadas y las líneas entre dos estados se llaman transiciones. Las transiciones constan de una sintaxis, la cual es la siguiente:

nombre_evento (parámetros) [condición] / acción_o_consecuencia

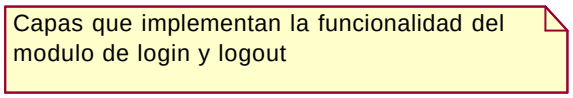
Los parámetros van separados por comas. La condición es una expresión que tiene que considerarse para que el evento se genere y la acción o consecuencia es una expresión que se debe efectuar después del evento y puede ser un incremento o un decremento.

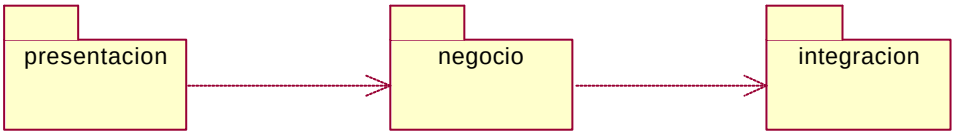


2.2.11 Diagrama de componentes

El diagrama de componentes describe componentes de software y sus dependencias con otros componentes, representando la estructura del código. Los componentes de software pueden ser: componentes de código, componentes binarios que son los generados por la compilación de los componentes de código y los componentes ejecutables.

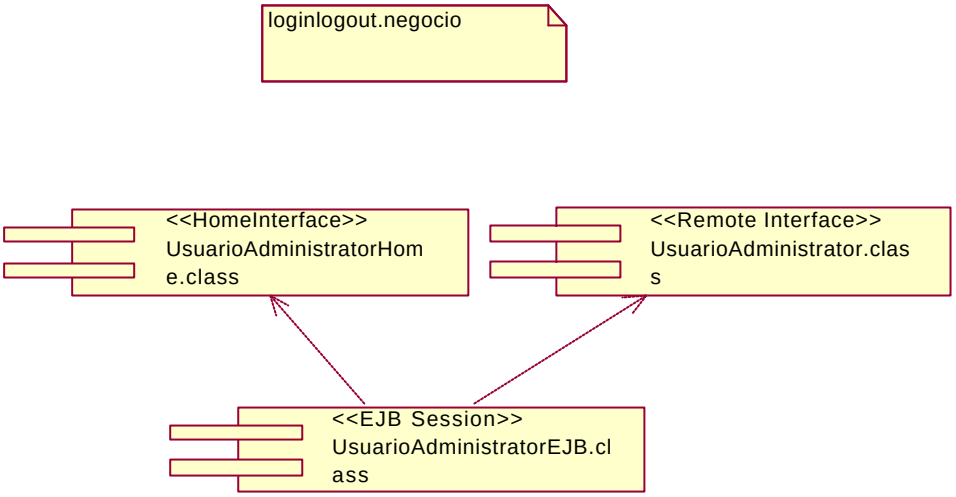
En este diagrama se pueden manejar paquetes, que son contenedores de clases utilizados para mantener el espacio de nombres de clases dividido en compartimentos, de manera que se utilizan para representar subsistemas del sistema en el mundo físico. Cada paquete se liga con otros a través de dependencias, que se representan con flechas de líneas discontinuas que van del componente dependiente al componente del cual depende.





Cada paquete debe tener un diagrama de componentes para representar las clases que contiene internamente, similar a hacer un acercamiento o "zoom" al interior de cada uno de los paquetes.

Los componentes de software se representan con un rectángulo que contiene dos pequeños rectángulos y un ovalo en su extremo izquierdo, o simplemente el rectángulo con los dos pequeños rectángulos.



2.2.12 Diagrama de distribución

En UML se definen los diagramas de distribución para apoyar las especificaciones de la arquitectura física y lógica de los sistemas. En estos diagramas se identifican físicamente los nodos y las conexiones que muestran la arquitectura del sistema donde se ejecuta cada componente.

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.

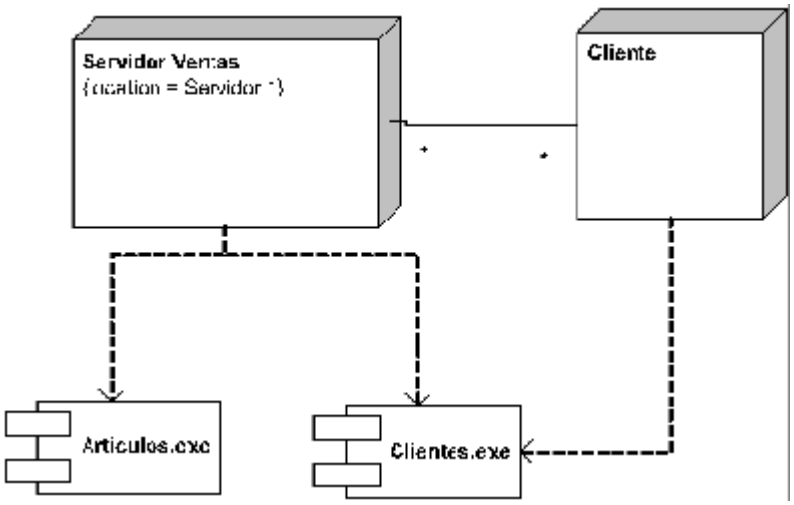
Cada Hardware se representa como un nodo.

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

En la figura siguiente se muestran dos nodos, el cliente y el servidor, cada uno de ellos contiene componentes. El componente del cliente utiliza una interfaz 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.



2.2.13 Notas y comentarios en UML

Una nota es un comentario del diagrama. No ejerce influencia semántica sobre los elementos y puede ser colocada en cualquier parte de un diagrama de UML. El símbolo empleado para indicar una nota es un rectángulo con la esquina superior derecha doblada hacia el interior.

Capas que implementan la funcionalidad del módulo de publicaciones

CAPÍTULO 3. Análisis del sitio web del INACIPE

3.1 Descripción de actores

DSBeI

Es la Dirección de Servicios Bibliográficos e Informáticos y esta representada por el Director y la Subdirección de Sistemas.

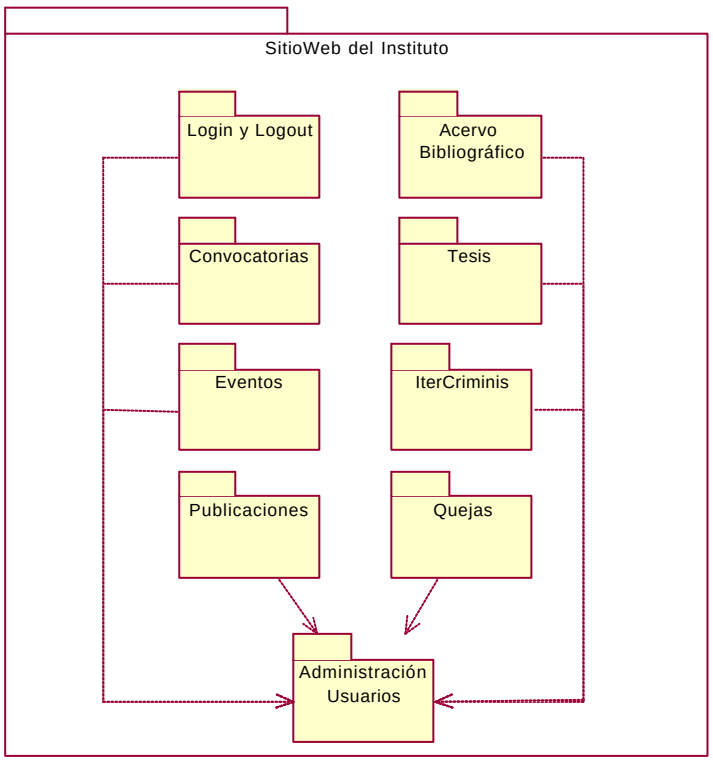
Usuario de Internet

Es la persona que consulta la página del Instituto.

Posgrado

Es encargado de revisar y corregir la correcta publicación del acervo bibliográfico.

Sitio web del Instituto



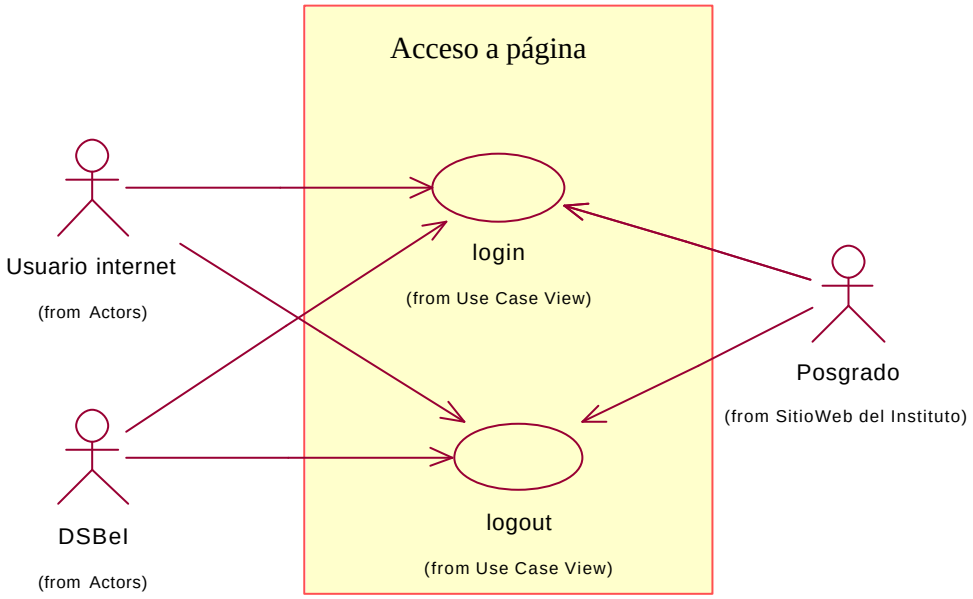
3.2 Casos de usos generales

Se presentan los diagramas de caso de uso que fueron analizados para su desarrollo:

- Acceso a la página
- Acervo Bibliográfico
- Convocatorias
- Eventos
- Administración de usuarios

3.3 Caso de uso: Acceso al sistema

3.3.1 Diagrama del caso de uso de acceso al sistema



3.3.2 Escenario del caso de uso de acceso al sistema

Usuario accesa al sistema (Login) y obtiene sesión.

Breve descripción:

Este caso de uso indica el momento en el que un usuario ingresa al área privada del sistema.

Justo en el momento en el que un usuario accesa al sistema éste deberá otorgarle al usuario una sesión.

En todos los casos el proceso es el mismo, por lo tanto sólo se indica una ocasión.

La sesión que se otorgará al usuario tiene un tiempo de vida limitado (por cuestiones de seguridad).

Actores:

DSBel, Usuario Internet y Posgrado

Extends o Include:

Requerimientos especiales:

Se debe tener el plugin de flash instalado en la computadora donde el usuario intentará accesar al sistema.

Flujo de eventos:

- Precondiciones
  - Estar en la forma de acceso al sistema
- Flujo ideal
  - El usuario digita su identificador en un campo de entrada
  - El usuario digita su contraseña en un campo de entrada
  - El usuario indica que desea ingresar al sistema
  - El servidor realiza una conexión con la BD
  - El sistema recupera la información recuperada desde la forma que digitó el usuario

- Se verifica que exista un usuario con el identificador y contraseña
- Se recupera de la base de datos la información de las actividades que puede hacer un usuario (para posteriormente ser solicitadas cuando se pida un servicio).
- El servidor cierra la conexión con la BD
- Una vez encontrado el usuario se crea una sesión para ese usuario
- Se redirecciona al usuario con alguna forma inicial de acciones

? Flujos alternativos

= No hay conexión a la BD

- El usuario digita su identificador en un campo de entrada
- El usuario digita su contraseña en un campo de entrada
- El usuario indica que desea accesar al sistema a través de un componente visual que indique "Acceso al sistema"
- Se intenta hacer una conexión con la BD, pero no logra completarse
- Se redirecciona al usuario a alguna forma y se le indica con un mensaje la razón por la que no pudo ingresar al sistema

= Login y password incorrecto

- El servidor realiza una conexión con la BD
- El sistema recupera la información capturada en la forma
- Se verifica el identificador y contraseña
- El identificador o la contraseña son inválidos
- El servidor cierra la conexión con la BD
- Una vez encontrado el usuario se crea una sesión para ese usuario
- Se redirecciona al usuario a alguna forma y se le indica con un mensaje la razón por la que no pudo ingresar al sistema

= No se cuenta con el plugin de flash instalado

- El usuario al solicitar la página donde debe digitar su identificador y su contraseña se le mostrará un mensaje indicándole que no cuenta con el plugin de flash y se le dará la opción a redireccionarse al lugar en donde puede descargar el plugin

? Post condición

- Una vez que el usuario haya ingresado con éxito al sistema verá las opciones que puede realizar
- Esta pantalla será la pantalla inicial de actividades del usuario

Frecuencia:

Muy frecuente

Notas:

El web server, el app server y la BD deben estar en ejecución.

En caso de instalar el plugin de flash este proceso se efectúa una única ocasión (para navegadores inferiores a la versión 5).

DSBel/Usuario Internet logout

Breve descripción:

Con este caso de uso el usuario termina su sesión con el sistema.

Extends o include:

Requerimientos especiales:

Ninguno

Flujo de eventos:

- Precondiciones
  - El usuario se encuentra en sesión, en cualquier lugar del sitio
- Flujo ideal
  - El usuario selecciona la opción para terminar de usar la parte administrativa de la página web del Instituto
  - El sistema verifica la sesión del usuario
  - El sistema toma la sesión del usuario que ha indicado salir del sistema
  - Termina los recursos empleados por este usuario en la sesión
  - Vuelve la sesión inactiva
  - Redirecciona al usuario a una forma que le indica que ha salido del sistema
- Flujos alternativos
  - Ninguno
- Post-condiciones
  - Una vez que haya salido el usuario del sistema no habrá manera de obtener todas las funcionalidades que podía hacer justo antes de indicar que saldría del sistema. Si desea hacer uso nuevamente de alguna funcionalidad deberá ingresar nuevamente su identificador y su contraseña en el sistema y ejecutar el caso de uso “hacer login”

Frecuencia:

Muy frecuente

Notas:

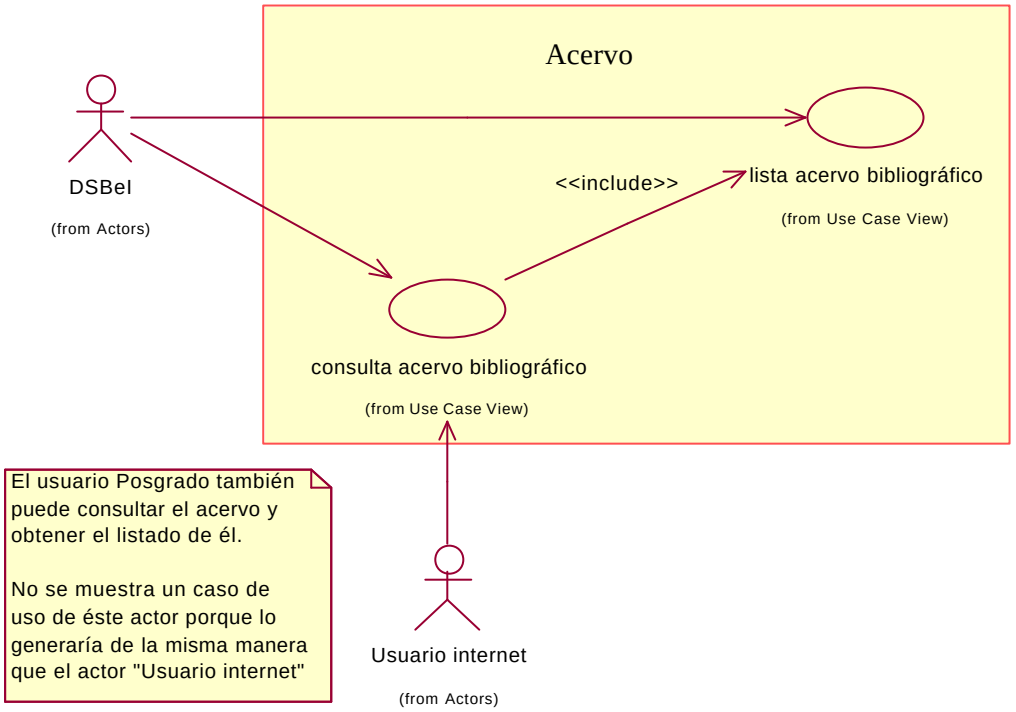
El servidor Web, el app server deben estar ejecutándose.

Debe existir en todas las pantallas un enlace que permita a un usuario salir de sesión.

El usuario debe de acostumbrarse a cerrar su sesión cuando ya no vaya a efectuar operaciones en la página.

3.4 Caso de uso: Acervo bibliográfico

3.4.1 Diagrama del caso de uso de acervo bibliográfico



3.4.2 Escenario del caso de uso de acervo bibliográfico

DSBeI lista el acervo bibliográfico (derivado de una consulta)

Breve descripción:

Este caso de uso permite al DSBeI obtener un listado de todos los acervos bibliográficos resultado de una consulta, almacenados en la base de datos en un listado paginado completo.

El caso de uso inicia cuando al usuario se le presenta una forma con opciones de búsqueda y el usuario digita la información que quiere obtener. El sistema deberá entregarle al usuario un listado con la información obtenida de la base de datos.

El listado deberá mostrar los resultados con algún orden (posiblemente en orden alfabético de algún campo).

Extends o include:

Requerimientos especiales:

Ninguno

Flujo de eventos:

? Precondiciones
    - Estar en sesión y que ésta no haya expirado
    - El usuario DSBeI debe estar en su forma inicial (después de hacer login)
? Flujo ideal
    - El DSBeI solicita listar los acervos bibliográficos
    - El DSBeI digita datos de consulta en una forma de consultas
    - El DSBeI indica continuar
    - El servidor verifica que el usuario tenga una sesión válida
    - Se establece una conexión con la BD
    - Se ejecuta la consulta con los datos proporcionados desde la forma y los resultados se recuperan
    - Se cierra la conexión con BD

- Los resultados son formateados
- Se envían al usuario los resultados de la consulta de manera paginada

? Flujos alternativos

= No hay conexión a la BD

- El DSBeI solicita listar los acervos bibliográficos
- El DSBeI digita datos de consulta en una forma de consultas
- El DSBeI indica continuar
- Se intenta hacer una conexión con la BD, pero es fallida
- Se redirecciona al usuario a una forma de mensaje y se le indica la realización fallida de la operación

= La sesión ha expirado

- El DSBeI solicita listar los acervos bibliográficos
- El DSBeI digita datos de consulta en una forma de consultas
- El DSBeI indica continuar
- El sistema detecta que la sesión expiró y lo redirecciona a una forma
- En esa forma se le indica con un mensaje el estado de su petición y se le dá opción de llegar a la forma de login nuevamente

? Post condición

- Si se efectuó satisfactoriamente este caso de uso el usuario estará viendo un listado paginado de todos los contenidos almacenados en la base de datos con el patrón de búsqueda que introdujo en un formato paginado
- Después del listado deberá dársele al usuario la facilidad de generar una nueva consulta a partir de esta forma pero no filtrada, la búsqueda será una nueva búsqueda

Frecuencia:

Frecuente

Notas:

Cuando el usuario tenga el listado de los acervos bibliográficos en su pantalla podrá ejecutar otros casos de uso.

El webserver, app server y la base de datos deben estar ejecutándose.

DSBeI consulta el acervo bibliográfico

Breve descripción:

En este caso de uso el usuario DSBeI a partir del listado obtenido en el caso de Uso "Lista acervo bibliográfico" puede seleccionar un acervo y enseguida se mostrará el detalle (datos particulares) del acervo bibliográfico seleccionado.

Extends o include:

Include Caso de Uso: DSBeI lista acervo bibliográfico.

Actores:

DSBeI, Usuario Internet.

Requerimientos especiales:

Ninguno

Flujo de eventos:

? Precondiciones

- Estar en sesión y que ésta esté vigente
- Estar en la pantalla obtenida después de ejecutar el caso de uso "DSBeI lista acervo bibliográfico"

? Flujo ideal

- El usuario DSBeI selecciona un elemento del listado que se encuentra en la pantalla de acervo bibliográfico
- El servidor verifica que el usuario tenga una sesión válida
- Se establece una conexión con la BD
- Se recupera el identificador del acervo seleccionado desde el listado
- Se obtienen los resultados de la BD
- Se formatean los resultados
- Se cierra la BD

- Se redirecciona al usuario una forma con la información bibliográfica que correspondió a la selección

? Flujos alternativos

= No hay conexión a la BD

- Aunque es muy difícil que se dé este caso (dado que para llegar al listado es porque previamente se estableció una conexión con la base de datos se indica de todos modos): El usuario selecciona un elemento que se encuentra en el listado del acervo bibliográfico
- El sistema intenta establecer una conexión con la base de datos, pero el intento es fallido
- Se indica al usuario la realización de la operación

= La sesión ha expirado

- El usuario selecciona un elemento que se encuentra en el listado del acervo bibliográfico
- El sistema detecta que la sesión ha expirado
- Se redirecciona al usuario a la página de login, para que proporcione sus datos de acceso y obtenga con ello una nueva sesión

? Post-condiciones

- Si el usuario ha completado todos los pasos del flujo ideal estará entonces viendo en su pantalla una descripción particular del acervo bibliográfico seleccionado

Frecuencia

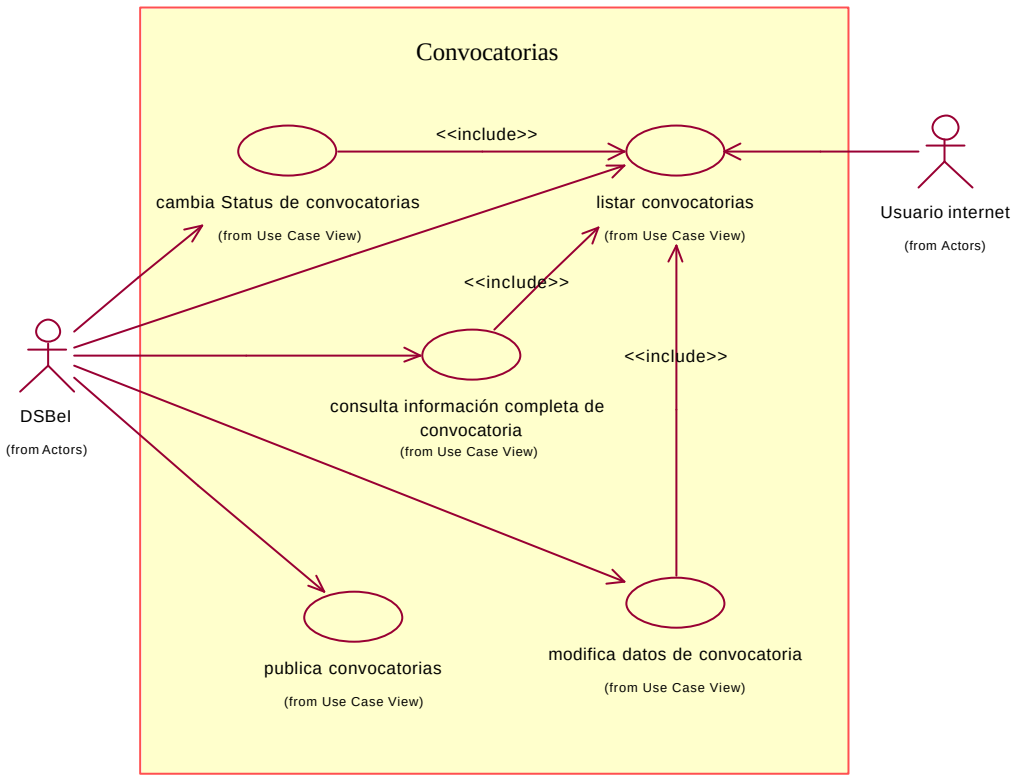
Frecuente

Notas:

El web server, app server y la base de datos deben estar ejecutándose.

3.5 Caso de uso: Convocatorias

3.5.1 Diagrama del caso de uso de convocatorias



3.5.2 Escenario del caso de uso convocatorias

DSBeI publica convocatorias

Breve descripción:

Este caso de uso permite al usuario DSBeI publicar convocatorias (imagen de la convocatoria y otros datos asociados a la convocatoria).

Desde el menú inicial de acciones del usuario se debe mostrar una manera para que éste pueda accesar al área de Convocatorias y desde ahí deberá poder publicar una nueva convocatoria.

La publicación de la convocatoria siempre incluye una imagen y una serie de datos adicionales obligatorios y opcionales y opcionalmente alguna información adicional relacionada con la convocatoria en archivo(s).

Extends o include:

Requerimientos especiales:

La imagen a publicar deberá estar en formato gif o jpg.

Generalmente esta dirección es para que la gente se ponga en contacto con el responsable de dar información sobre la convocatoria publicada.

Flujo de eventos:

? Precondiciones

- Se debe de contar con la imagen de la convocatoria
- El usuario DSBeI debe de estar en la forma de inicio de sus actividades y de ahí indicar que se publicará una convocatoria

? Flujo ideal

- DBSeI indica el número de archivos anexos que llevará la convocatoria a publicar
- El servidor valida que el usuario tenga una sesión válida
- El servidor recibe el número de archivos anexos que llevará la siguiente forma
- El servidor redirecciona la forma para agregar los datos de la convocatoria

- Ingresa los datos correspondientes a la descripción de la convocatoria (algunos datos son obligatorios y otros son opcionales)
- Selecciona la imagen que corresponde con la convocatoria a publicar
- Anexa los archivos asociados con la convocatoria
- Envía los datos
- El servidor verifica que el usuario tenga una sesión válida
- El servidor inicia una conexión con la BD
- Se guardan asociados a la convocatoria en la BD
- Se guarda la imagen de la convocatoria en el sistema de archivos
- Se guardan los archivos asociados a la convocatoria en el sistema de archivos
- Se guarda la referencia de la imagen de la convocatoria en la BD
- Se guarda la referencia de los archivos asociados en la base de datos
- Se cierra la conexión con la BD
- El servidor redirecciona una forma indicando la operación realizada

? Flujos alternativos

= No hay conexión a la BD

- El DBSel entra a la sección de publicación de convocatorias
- Ingresa los datos a la forma que se le solicitan
- Selecciona la imagen que corresponde con la convocatoria a publicar.
- Envía los datos
- El servidor no puede establecer una conexión con la BD.
- Se borra el archivo con la imagen del sistema de archivos.
- Se indica al usuario que no se realizó la operación a través de una forma

= No hay espacio en disco

- DBSel entra a la sección de publicación de convocatorias
- Ingresa los datos del formulario
- Selecciona la imagen que corresponde con la convocatoria a publicar

- Envía los datos
- El servidor abre la conexión con la BD
- Se intenta guardar el archivo en el sistema de archivos y se recibe notificación de que no hay espacio suficiente para ser guardado
- No se guarda la imagen en el sistema de archivos del servidor
- No se guarda ningún dato en la BD
- Se cierra la conexión con la BD
- Se indica al usuario que no se realizó la operación

= No hay espacio en la BD

- DBSel entra a la sección de publicación de convocatorias
- Ingresa los datos al formulario
- Selecciona la imagen que corresponde con la convocatoria a publicar
- Envía los datos
- Se abre la conexión con la BD
- Se guarda la imagen en el sistema de archivos
- Se intenta guardar los datos de la convocatoria pero se notifica que no hay espacio suficiente en la BD
- Se elimina la imagen del sistema de archivos del servidor
- Se indica al usuario que no se realizó la operación

? Post-condiciones

- Si se efectuaron satisfactoriamente los pasos del flujo ideal el evento se habrá publicado lo que implica que la imagen del cartel de la convocatoria estará guardada en el sistema de archivos, su referencia estará en la base de datos al igual que los datos asociados al evento y sus archivos anexos

Frecuencia

Frecuente

Notas:

DSBeI lista convocatorias publicadas

Breve descripción:

Este caso de uso permite al usuario DSBeI obtener un listado de las convocatorias que han sido publicadas en la base de datos.

Este listado debe estar paginado y deberá estar ordenado por alguna característica de la convocatoria.

Extends o include:

Requerimientos especiales:

Ninguno

Flujo de eventos:

? Precondiciones

- El usuario debe estar en sesión y ésta debe estar activa
- DBSeI se encuentra en la forma inicial y selecciona “listado de convocatorias”

? Flujo ideal

- Cuando el actor selecciona la opción de obtener el listado de las convocatorias se hace una petición al servidor
- El servidor verifica que el usuario tenga una sesión válida
- Se abre conexión a la BD
- Se recuperan las convocatorias publicadas
- Al momento de recuperarse cada una de las convocatorias se debe de colocar un indicativo que permita en el listado efectuar alguna acción sobre la convocatoria que se listará
- Se formatean las convocatorias
- Se redirecciona al usuario una forma en la que se muestran las convocatorias listadas

? Flujos alternativos

= No hay conexión a la BD

- DBSeI entra a la sección de publicación de convocatorias
- No se establece la conexión con la BD
- No se listan las convocatorias
- Se indica al usuario que no se realizó la operación

? Post-condiciones

- Si el usuario ha efectuado satisfactoriamente los pasos indicados en el flujo ideal el DSBeI estará viendo en su pantalla un listado con todas las convocatorias publicadas y que están almacenadas en la base de datos

Frecuencia:

Muy Frecuente

Notas

Debe estar ejecutándose el servidor web, el app server y la base de datos.

DSBeI consulta información completa de convocatorias

Breve descripción:

Este caso de uso permite al usuario DSBeI obtener la información completa que corresponde a una convocatoria. Esta información comprende la imagen de la convocatoria, sus datos asociados y los archivos anexos en caso de existir.

Extends o include:

Include caso de uso DSBeI lista convocatorias publicadas.

Requerimientos especiales:

Ninguno

Flujo de eventos:

? Precondiciones

- El usuario debe estar en sesión y ésta deberá estar vigente
- El usuario debió de haber ejecutado el caso de uso “lista convocatorias” y estar en la pantalla generada por este caso de uso

? Flujo ideal

- DBSeI elige la convocatoria de la que quiere ver los detalles y la selecciona
- El servidor verifica que el usuario tenga una sesión válida
- El servidor recupera el identificador de la convocatoria
- El servidor establece una conexión a la BD
- Se obtienen los datos de la convocatoria seleccionada de la base de datos
- El servidor cierra la conexión con la base de datos
- Se redirecciona al usuario una forma con la información de la convocatoria

? Flujos alternativos

= La sesión ha expirado

- El DSBeI elige la convocatoria que quiere ver y selecciona "Visualizar"

- El sistema detecta que la sesión expiró y lo redirecciona a una forma con mensaje y opción a abrir pantalla de login

= No hay conexión a la BD

- El DSBeI elige la convocatoria que quiere ver y selecciona "Visualizar"
- No se establece la conexión con la BD
- Se indica al usuario que no se realizó la operación

? Post-condiciones

- Si el usuario efectuó con éxito todos los pasos indicados en el flujo ideal entonces debió de tener en su pantalla la imagen completa correpondiente a la convocatoria y en cuyo caso la dirección de correo de la persona que se dio de alta en la base para poder obtener más información de la convocatoria

Frecuencia:

Frecuente

Notas:

El servidor web, el app server y la base de datos deben de estar ejecutándose.

DSBeI modifica datos de convocatorias.

Breve descripción:

Este caso de uso permite al usuario DSBeI modificar los datos de las convocatorias dados de alta en el sistema y que se publicaron en el momento en el que la convocatoria se publicó.

Extends o include:

Include, hace uso del caso de uso “listar convocatorias”.

Requerimientos especiales:

Para poder visualizar un archivo anexado a una convocatoria se deberá contar con el software que corresponda para poder ver dicho archivo (por ejemplo, para visualizar un documento anexado como Word se deberá tener instalado el editor de texto Word).

Flujo de eventos:

? Precondiciones

- Que el usuario este en sesión y que ésta esté activa
- Que el usuario haya ejecutado el caso de uso “listar convocatorias” y se encuentre en el listado de las mismas

? Flujo ideal

- DBSeI elige la convocatoria que quiere modificar y selecciona la opción de “Modificar” que se encontraría cercana a cada convocatoria del listado
- El servidor verifica que el usuario tenga una sesión válida
- El servidor establece una conexión a la BD
- El servidor recupera el identificador de la convocatoria
- Se obtienen los datos de la convocatoria seleccionada
- El servidor reenvía una forma al usuario con los datos recuperados
- Se despliega la información de la convocatoria
- DBSeI realiza los cambios de los datos en la forma
- Envía los datos
- El servidor verifica la sesión del usuario

- El servidor recupera de la forma las modificaciones realizadas
- El servidor establece una conexión con la base de datos
- El servidor guarda los datos en la BD
- El servidor cierra la conexión a la BD
- El servidor redirecciona una forma al DBSeI indicándole una confirmación de la operación realizada

? Flujos alternativos

= La sesión ha expirado

- El DSBeI elige la convocatoria que quiere modificar y selecciona "Modificar"
- El sistema detecta que la sesión expiró y lo redirecciona a la pantalla con mensaje y opción a abrir pantalla de login

= No hay conexión a la BD

- El DSBeI elige la convocatoria que quiere modificar y selecciona "Modificar"
- No se establece la conexión con la BD
- Se indica al usuario que la operación no se realizó

= No hay espacio en disco

- DBSeI elige la convocatoria que quiere modificar y selecciona "Modificar"
- Se abre conexión a la BD
- Se recupera el identificador de la convocatoria
- Se obtienen los datos de la convocatoria seleccionada
- Se despliega la información de la convocatoria
- DBSeI realiza los cambios de los datos en la forma
- Envía los datos
- Se guardan los datos en la BD
- No se puede guardar la imagen de la convocatoria en el sistema de archivos por que no hay espacio suficiente
- Se cancelan los cambios en la BD

- Se cierra la conexión a la BD
- Se indica al usuario que la operación no se realizó

? Post-condiciones

- Si el usuario ejecutó satisfactoriamente todos los pasos del flujo ideal entonces habrá afectado la base de datos con nueva información relacionada a la convocatoria que seleccionó

Frecuencia:

Poco frecuente

Notas:

El servidor Web, el app server y la base de datos deben estar ejecutándose.

DSBeI cambia status de convocatorias

Breve descripción:

Este caso permite al usuario DSBeI cambiar el status de las convocatorias como publicable o no publicable.

Una convocatoria publicable será aquella que pueda consultar desde Internet un "Usuario Internet". Una convocatoria no publicable nunca podrá verla un "Usuario Internet" y únicamente la podrá consultar el usuario DSBeI.

Extends o include:

Include hace uso del caso de uso DSBeI lista convocatorias.

Requerimientos especiales:

El usuario debe estar en sesión y ésta debe estar vigente.

El usuario debe tener el listado de las convocatorias, es decir, haber terminado el caso de uso "lista de convocatorias".

Flujo de eventos:

? Pre condiciones

- Las convocatorias listadas deben estar en la pantalla (esta pantalla de convocatorias listadas es la forma obtenida por ejecutar el caso de uso "Lista convocatorias")

? Flujo ideal

- DBSeI elige la convocatoria que quiere cambiar status
- La forma del DSBeI solicita una confirmación del cambio de status
- DBSeI confirma que sí desea cambiar el status de la convocatoria
- El servidor verifica que el usuario tenga una sesión válida
- Se recupera el identificador de la convocatoria
- El servidor establece una conexión a la BD
- Se cambia el status de la convocatoria de la BD
- Se cierra la conexión a la BD

- El servidor redirecciona al usuario DBSeI una forma de confirmación de la operación realizada

? Flujos alternativos

= Se cancela la operación

- DBSeI elige la convocatoria que quiere cambiar el status
- El sistema solicita una confirmación de cambio de status
- DBSeI cancela la operación de cambio de status
- El usuario DSBeI permanece en la forma inicial

= No se puede establecer una conexión con la BD

- DBSeI elige la convocatoria que quiere cambiar status
- El sistema solicita una confirmación del cambio de status
- DBSeI confirma que si desea cambiar el status de la convocatoria
- El servidor intenta establece una conexión a la BD pero no puede completarla
- El servidor redirecciona al usuario DBSeI a una forma en donde se le indica que la operación no pudo efectuarse.

= La sesión ha expirado

- El DSBeI elige la convocatoria que quiere eliminar y selecciona "Borrar"
- El sistema detecta que la sesión expiró y lo redirecciona a una forma con un mensaje indicándole que la sesión ha terminado y le presenta una opción para ir hacia la pantalla de login.

? Post-condiciones

- Si el usuario completó satisfactoriamente las acciones indicadas en el flujo ideal entonces la convocatoria habrá cambiado de status en la base de datos y el usuario continuará en el sistema en su pantalla inicial

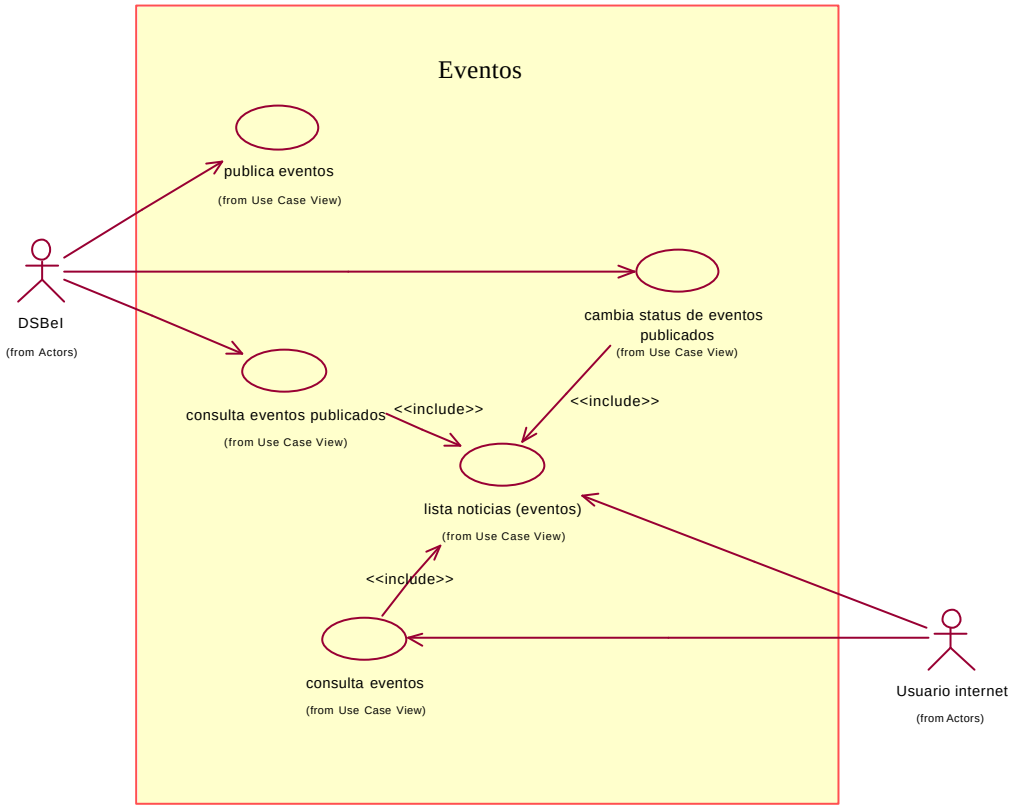
Frecuencia:

Frecuente

Notas:

3.6 Caso de uso: Eventos

3.6.1 Diagrama de caso de uso de eventos



3.6.2 Escenario del caso de uso de eventos

DSBeI publica eventos

Breve descripción:

Este caso de uso permite al usuario DSBeI publicar información relacionada con algún evento.

La información asociada a un evento consta de título, una imagen, tipo de evento, fecha de expiración y un número de archivos anexos que puede variar de 0 a n.

Cada evento tiene a su vez un status con el que se controla se despliegue a un usuario de Internet o no se muestre.

El usuario debe indicar desde la forma inicial donde selecciona su intención de publicar un evento, cuántos archivos anexos deberán publicarse para el evento que intentará realizar.

Extends o include:

Requerimientos especiales:

Si el evento que se va a publicar lleva anexa una imagen ésta deberá estar en formato gif o jpg.

Los archivos que se anexen se recomienda sean subidos en un formato común (cualquiera de Office).

Flujo de eventos:

? Precondiciones

- Se debe de contar con la imagen del evento
- El usuario DSBeI debe de estar en la forma de inicio de sus actividades y de ahí indicar que se publicará un evento

? Flujo ideal

- DBSeI indica el número de archivos anexos que llevará el evento a publicar
- El servidor valida que el usuario tenga una sesión del usuario
- El servidor recibe el número de archivos anexos que llevará la siguiente forma
- El servidor redirecciona la forma para agregar los datos del evento

- Ingresa los datos correspondientes a la descripción del evento (algunos datos son obligatorios y otros son opcionales)
- Selecciona la imagen que corresponde con el evento a publicar
- Anexa los archivos asociados con el evento
- Envía los datos
- El servidor verifica que el usuario tenga una sesión válida
- El servidor inicia una conexión con la BD
- Se guarda la imagen del evento en el sistema de archivos
- Se guardan los archivos asociados al evento en el sistema de archivos
- Se guarda la referencia de la imagen del evento en la BD
- Se guarda la referencia de los archivos asociados en la base de datos
- Se cierra la conexión con la BD
- El servidor redirecciona una forma indicando la operación realizada

? Flujos alternativos

= No hay conexión a la BD

- El DBSeI entra a la sección de publicación de eventos
- Ingresa los datos a la forma que se le solicitan
- Selecciona la imagen que corresponde con el evento a publicar
- Envía los datos
- El servidor no puede establecer una conexión con la BD
- Se borra el archivo con la imagen del sistema de archivos
- Se indica al usuario que no se realizó la operación a través de una forma

= No hay espacio en disco

- DBSeI entra a la sección de publicación de eventos
- Ingresa los datos del formulario
- Selecciona la imagen que corresponde con el evento a publicar
- Envía los datos

- El servidor abre la conexión con la BD
- Se intenta guardar el archivo en el sistema de archivos y se recibe notificación de que no hay espacio suficiente para ser guardado
- No se guarda la imagen en el sistema de archivos del servidor
- No se guarda ningún dato en la BD
- Se cierra la conexión con la BD
- Se indica al usuario que no se realizó la operación

= No hay espacio en la BD

- DBSel entra a la sección de publicación de eventos
- Ingresa los datos al formulario
- Selecciona la imagen que corresponde con el evento a publicar
- Envía los datos
- Se abre la conexión con la BD
- Se guarda la imagen en el sistema de archivos
- Se intenta guardar la los datos del evento pero se notifica que no hay espacio suficiente en la BD
- Se elimina la imagen del sistema de archivos del servidor
- Se indica al usuario que no se realizó la operación

? Post-condiciones

- Si se efectuaron satisfactoriamente los pasos del flujo ideal el evento se habrá publicado lo que implica que la imagen del cartel del evento estará guardada en el sistema de archivos, su referencia estará en la base de datos al igual que los datos asociados al evento y sus archivos anexos

Frecuencia:

Frecuente

Notas:

DSBeI lista eventos

Breve descripción:

Este caso de uso permite al usuario DSBeI obtener un listado de todos los eventos que han sido publicados en la base de datos.

Este listado debe estar paginado y deberá estar ordenado por algún atributo del evento.

Extends o include:

Requerimientos especiales:

Ninguno

Flujo de eventos:

? Precondiciones

- El usuario debe estar en sesión y ésta debe estar activa
- DBSeI se encuentra en la forma inicial y selecciona “listado de eventos”

? Flujo ideal

- Cuando el actor selecciona la opción de obtener el listado de los eventos se hace una petición al servidor
- El servidor verifica que el usuario tenga una sesión válida
- Se abre conexión a la BD
- Se recuperan los eventos publicados
- Al momento de recuperarse cada uno de los eventos se debe de colocar un indicativo que permita en el listado efectuar alguna acción sobre el evento que se listará
- Se formatean los eventos recuperados
- Se redirecciona al usuario una forma en la que se muestran los eventos listados.

? Flujos alternativos

= No hay conexión a la BD

- DBSeI entra a la sección de publicación de eventos

= No se establece la conexión con la BD

- No se listan los eventos
- Se indica al usuario que no se realizó la operación

? Post-condiciones

- Si el usuario ha efectuado satisfactoriamente los pasos indicados en el flujo ideal el DSBeI estará viendo en su pantalla un listado con todos los eventos publicados y que están almacenados en la base de datos

Frecuencia:

Muy Frecuente

Notas:

El listado de los eventos debe listar el status del evento y la ruta donde se encuentra la imagen correspondiente al evento ya que si se desea modificar esa imagen una vez que el evento se ha publicado permitirá al administrador conocer el nombre de la imagen correspondiente y subir y renombrar al archivo gráfico que sustituirá al anterior.

DSBeI consulta eventos publicados (información completa)

Breve descripción:

Este caso de uso permite al usuario DSBeI obtener la información completa que corresponde a un evento. Esta información comprende la imagen del evento, sus datos asociados, los archivos anexos en caso de existir.

Extends o include:

Include caso de uso DSBeI lista eventos publicados.

Requerimientos especiales:

Ninguno

Flujo de eventos:

? Precondiciones

- El usuario debe estar en sesión y ésta deberá estar vigente
- El usuario debió de haber ejecutado el caso de uso “lista eventos” y estar en la pantalla generada por este caso de uso

? Flujo ideal

- DBSeI elige el evento del que quiere ver los detalles y lo selecciona
- El servidor verifica que el usuario tenga una sesión válida
- El servidor recupera el identificador del evento
- El servidor establece una conexión a la BD
- Se obtienen los datos del evento seleccionado de la base de datos
- El servidor cierra la conexión con la base de datos
- Se redirecciona al usuario una forma con la información del evento

? Flujos alternativos

= La sesión ha expirado

- El DSBeI elige el evento que quiere ver y selecciona "Visualizar"
- El sistema detecta que la sesión expiró y lo redirecciona a una forma con mensaje y opción a abrir pantalla de login

= No hay conexión a la BD

- El DSBeI elige el evento que quiere ver y selecciona "Visualizar"
- No se establece la conexión con la BD
- Se indica al usuario que no se realizó la operación

? Post-condiciones

- Si el usuario efectuó con éxito todos los pasos indicados en el flujo ideal, entonces debió de tener en su pantalla la imagen completa correspondiente al evento incluyendo información para obtener informes relacionados al evento

Frecuencia:

Frecuente

Notas:

El servidor web, el app server y la base de datos deben de estar ejecutándose.

DSBeI cambia status a eventos publicados

Breve descripción:

Este caso permite al usuario DSBeI cambiar el status de los eventos como publicable o no publicable.

Un evento publicable será aquel que puede consultar desde Internet un “Usuario Internet”. Un evento no publicable nunca podrá ser visto por un “Usuario Internet” y únicamente la podrá consultar el usuario DSBeI.

Extends o include:

Include hace uso del caso de uso DSBeI lista eventos.

Requerimientos especiales:

El usuario debe estar en sesión y ésta debe estar vigente.

El usuario debe tener el listado de los eventos, es decir, haber terminado el caso de uso “lista eventos”.

Flujo de eventos:

? Pre condiciones

- Los eventos listados deben estar en la pantalla (esta pantalla de eventos listados es la forma obtenida por ejecutar el caso de uso “Lista eventos”)

? Flujo ideal

- DBSeI elige el evento al que quiere cambiar status
- La forma del DSBeI solicita una confirmación del cambio de status
- DBSeI confirma que sí desea cambiar el status del evento
- El servidor verifica que el usuario tenga una sesión válida
- Se recupera el identificador del evento
- El servidor establece una conexión a la BD
- Se cambia el status del evento de la BD
- Se cierra la conexión a la BD

- El servidor redirecciona al usuario DBSeI una forma de confirmación de la operación realizada

? Flujos alternativos

= Se cancela la operación

- DBSeI elige el evento al que quiere cambiar el status
- El sistema solicita una confirmación de cambio de status
- DBSeI cancela la operación de cambio de status
- El usuario DSBeI permanece en la forma inicial.

= No se puede establecer una conexión con la BD

- DBSeI elige el evento al que quiere cambiar status
- El sistema solicita una confirmación del cambio de status
- DBSeI confirma que si desea cambiar el status del evento
- El servidor intenta establecer una conexión a la BD pero no puede completarla
- El servidor redirecciona al usuario DBSeI a una forma en donde se le indica que la operación no pudo efectuarse

= La sesión ha expirado

- El DSBeI elige el evento al que quiere eliminar y selecciona "Borrar"
- El sistema detecta que la sesión expiró y lo redirecciona a una forma con un mensaje indicándole que la sesión ha terminado y le presenta una opción para ir hacia la pantalla de login

? Post-condiciones

- Si el usuario completó satisfactoriamente las acciones indicadas en el flujo ideal entonces el evento habrá cambiado de status en la base de datos y el usuario continuará en el sistema en su pantalla inicial

Frecuencia:

Frecuente

Notas:

Ninguna

DSBeI modifica datos de eventos publicados

Breve descripción:

Este caso de uso permite al usuario DSBeI modificar los datos de los eventos dados de alta en el sistema y que se publicaron en el momento en el que el evento se publicó.

Extends o include:

Include, hace uso del caso de uso “Lista eventos”.

Requerimientos especiales:

Para poder visualizar un archivo anexado a una convocatoria se deberá contar con el software que corresponda para poder ver dicho archivo (por ejemplo, para visualizar un documento anexado como Word se deberá tener instalado el editor de texto Word).

Flujo de eventos:

? Precondiciones

- Que el usuario este en sesión y que ésta esté activa
- Que el usuario haya ejecutado el caso de uso “Lista eventos” y se encuentre en el listado de los mismos

? Flujo ideal

- DBSeI elige el evento al cual quiere modificar su información relacionada y selecciona la opción de “Modificar” que se encontraría cercana a cada evento del listado.
- El servidor verifica que el usuario tenga una sesión válida
- El servidor establece una conexión a la BD
- El servidor recupera el identificador del evento
- Se obtienen los datos del evento seleccionado
- El servidor reenvía una forma al usuario con los datos recuperados
- Se despliega la información del evento
- DBSeI realiza los cambios de los datos en la forma
- Envía los datos

- El servidor verifica la sesión del usuario
- El servidor recupera de la forma las modificaciones realizadas
- El servidor establece una conexión con la base de datos
- El servidor guarda los datos en la BD
- El servidor cierra la conexión a la BD
- El servidor redirecciona una forma al DBSeI indicándole una confirmación de la operación realizada

? Flujos alternativos

= La sesión ha expirado

- El DSBeI elige el evento que quiere modificar y selecciona "Modificar"
- El sistema detecta que la sesión expiró y lo redirecciona a la pantalla con mensaje y opción a abrir pantalla de login

= No hay conexión a la BD

- El DSBeI elige el evento que quiere modificar y selecciona "Modificar"
- No se establece la conexión con la BD
- Se indica al usuario que la operación no se realizó

= No hay espacio en disco.

- DBSeI elige el evento que quiere modificar y selecciona "Modificar"
- Se abre conexión a la BD
- Se recupera el identificador del evento
- Se obtienen los datos del evento seleccionado
- Se despliega la información del evento
- DBSeI realiza los cambios de los datos en la forma
- Envía los datos
- Se guardan los datos en la BD
- No se puede guardar la imagen del evento en el sistema de archivos por que no hay espacio suficiente

- Se cancelan los cambios en la BD
- Se cierra la conexión a la BD
- Se indica al usuario que la operación no se realizó

? Post-condiciones

- Si el usuario ejecutó satisfactoriamente todos los pasos del flujo ideal entonces habrá afectado la base de datos con nueva información relacionada al evento que seleccionó.

Frecuencia:

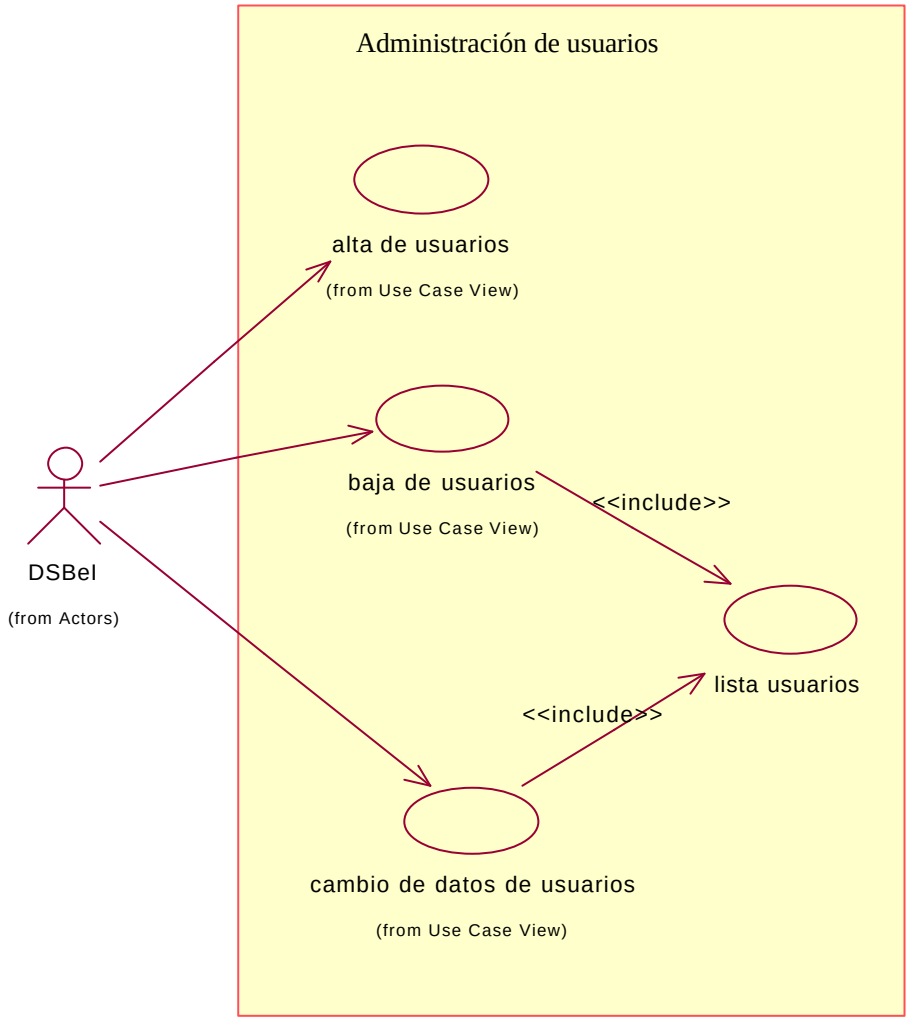
Poco frecuente

Notas:

El servidor Web, el app server y la base de datos deben estar ejecutándose.

3.7 Caso de uso: Administración de usuarios

3.7.1 Diagrama de caso de uso de administración de usuarios



3.7.2 Escenario del caso de uso de administración de usuarios

DSBeI alta de usuarios

Breve descripción:

Este caso de uso permite al administrador dar de alta a nuevos usuarios y asignarles las actividades que tendrán permitido efectuar.

Extends o include:

Requerimientos especiales:

Ninguno

Flujo de eventos:

? Pre condiciones

- El usuario DSBeI debe tener una sesión
- La sesión debe estar activa
- Que el usuario DSBeI haya seleccionado la opción para dar de alta a usuarios

? Flujo ideal

- El usuario DSBeI solicita la forma para agregar a un nuevo usuario
- El servidor verifica que el usuario tenga una sesión y que esa sesión sea válida
- El sistema redirecciona al DSBeI una forma de captura donde podrá seleccionar los distintos permisos existentes y marcará los que corresponderán al nuevo usuario incluyendo los campos para digitar la información adicional
- El DSBeI confirma los datos digitados
- El servidor verifica que el usuario tenga una sesión y que esa sesión sea válida
- Los datos son recuperados por el servidor. Entre esos datos debe recuperarse la contraseña digitada desde la forma. Esta contraseña será necesaria para que posteriormente el usuario ingrese al sistema
- El servidor establece una conexión con la base de datos
- El servidor registra los datos en la base de datos

- El servidor cierra la conexión con la base de datos
- El servidor redirecciona al usuario a una forma que le indica que el nuevo usuario se ha registrado

? Flujos alternativos

= La sesión del usuario no esta activa

- El usuario DSBeI solicita la forma para digitar los datos del nuevo usuario
- El sistema verifica que el usuario tiene una sesión válida
- El sistema lo redirecciona a una forma con un mensaje indicándole que su sesión no esta activa

= La base de datos no esta ejecutándose

- El usuario DSBeI solicita la forma para digitar los datos del nuevo usuario
- El sistema verifica que el usuario tenga una sesión activa
- El sistema redirecciona al usuario la forma de captura
- El DSBeI define en esa misma forma los permisos que tendrá el nuevo usuario
- El DSBeI confirma los datos digitados
- Los datos son enviados al servidor
- El servidor recupera los datos de la forma
- El servidor establece una conexión con la base de datos

? Post-condiciones

- Ninguna

Frecuencia:

Notas:

El servidor Web, el app server y la base de datos deben estar ejecutándose.

Se recomendaría que en el momento en el que el administrador dé de alta a un nuevo usuario éste se encuentre presente para que pueda digitar su contraseña inicial, de no ser así entonces el administrador deberá de establecer una por defecto.

Este caso de uso es muy importante y delicado, sólo deberá ser ejecutado por el administrador y por nadie más.

DSBeI lista usuarios

Breve descripción:

Este caso de uso le da capacidad al usuario DSBeI de listar a los usuarios dados de alta en el sistema. El listado se efectuará alfabéticamente y en un formato paginado.

Extends o include:

Requerimientos especiales:

Ninguno

Flujo de eventos:

? Pre condiciones

- El usuario debe tener una sesión válida
- El usuario se encuentra en su forma inicial de actividades

? Flujo ideal

- El usuario DSBeI selecciona la opción para listar a los usuarios dados de alta en el sistema
- El servidor verifica que el usuario tenga una sesión válida
- El servidor establece una conexión con la base de datos
- Recupera los datos de los usuarios dados de alta en el sistema
- Se cierra la conexión con la base de datos
- Se redirecciona al usuario una forma que contiene la información de los usuarios recuperados

? Post-condiciones

- Una vez que se han efectuado todos los pasos indicados en el flujo ideal el usuario DSBeI estará viendo una forma con los registros de los usuarios registrados en el sistema
- Esa lista estará ordenada alfabéticamente y de manera paginada

Frecuencia:

Posiblemente frecuente

Notas:

Este caso de uso permitirá efectuar otros casos de uso como:

DSBeI cambia datos de usuario.

DSBeI da de baja a usuario.

El servidor web, el app server y la base de datos deben estar ejecutándose.

DSBeI da de baja un usuario

Breve descripción:

Este caso de uso le da capacidad al usuario DSBeI de borrar del sistema a un usuario que haya dado de alta previamente.

Sólo se eliminan los usuarios que se dieron de alta a través del caso de uso “DSBeI da de alta a usuario”.

Extends o include:

Include caso de uso DSBeI lista usuarios.

Requerimientos especiales:

Ninguno

Flujo de eventos:

? Pre condiciones

- El usuario debe tener una sesión válida
- El usuario se encuentra en la forma donde se encuentran todos los usuarios listados

? Flujo ideal

- El usuario DSBeI selecciona a un usuario del listado
- Indica que desea borrarlo
- La forma le solicita confirmación de borrado
- El usuario DSBeI confirma.
- El servidor verifica que el usuario tenga una sesión válida
- El servidor recupera el identificador del usuario
- El servidor establece una conexión con la base de datos
- El servidor borra el registro del usuario
- El servidor redirecciona una forma de indicación de la operación efectuada

? Flujos alternativos

- Ninguno

? Post-condiciones

- Si el usuario ha efectuado todos los pasos indicados en el flujo ideal entonces se habrá eliminado a un usuario de la base de datos. Ese usuario no va a poder accesar nuevamente al sistema

Frecuencia:

Frecuente

Notas:

El servidor Web, el app server y la base de datos deben estar ejecutándose.

DSBeI cambios en datos de usuarios

Breve descripción:

En este caso de uso el usuario DSBeI puede modificar los datos de un usuario seleccionado previamente del listado de usuarios registrados en la base de datos.

Extends o include:

Include hace uso del caso de uso DSBeI lista usuarios.

Requerimient os especiales:

Flujo de eventos:

? Pre condiciones

- Que el usuario este en sesión y que ésta esté activa
- Que el usuario haya ejecutado el caso de uso “Lista usuarios” y se encuentre en el listado de los mismos

? Flujo ideal

- DBSeI elige al usuario al que quiere modificar y selecciona la opción de “Modificar” que se encontraría cercana a cada nombre de usuario del listado
- El servidor verifica que el usuario tenga una sesión válida
- El servidor establece una conexión a la BD
- El servidor recupera el identificador del usuario
- Se obtienen los datos del usuario seleccionado
- El servidor reenvía una forma al usuario con los datos recuperados
- Se despliega la información del usuario
- El DBSeI realiza los cambios de los datos en la forma
- Envía los datos
- El servidor verifica la sesión del usuario
- El servidor recupera de la forma las modificaciones realizadas
- El servidor establece una conexión con la base de datos

- El servidor guarda los datos en la BD
- El servidor cierra la conexión a la BD
- El servidor redirecciona una forma al DBSel indicándole una confirmación de la operación realizada

? Flujos alternativos

= La sesión ha expirado

- El DSBel elige al usuario al que quiere modificar sus datos y selecciona "Modificar"
- El sistema detecta que la sesión expiró y lo redirecciona a la pantalla con mensaje y opción a abrir pantalla de login

= No hay conexión a la BD

- El DSBel elige el usuario al que quiere modificar sus datos y selecciona "Modificar"
- No se establece la conexión con la BD
- Se indica al usuario que la operación no se realizó

? Post-condiciones

- Si el usuario ejecutó satisfactoriamente todos los pasos del flujo ideal entonces habrá afectado la base de datos con nueva información relacionada al usuario que seleccionó

Frecuencia:

Poco frecuente

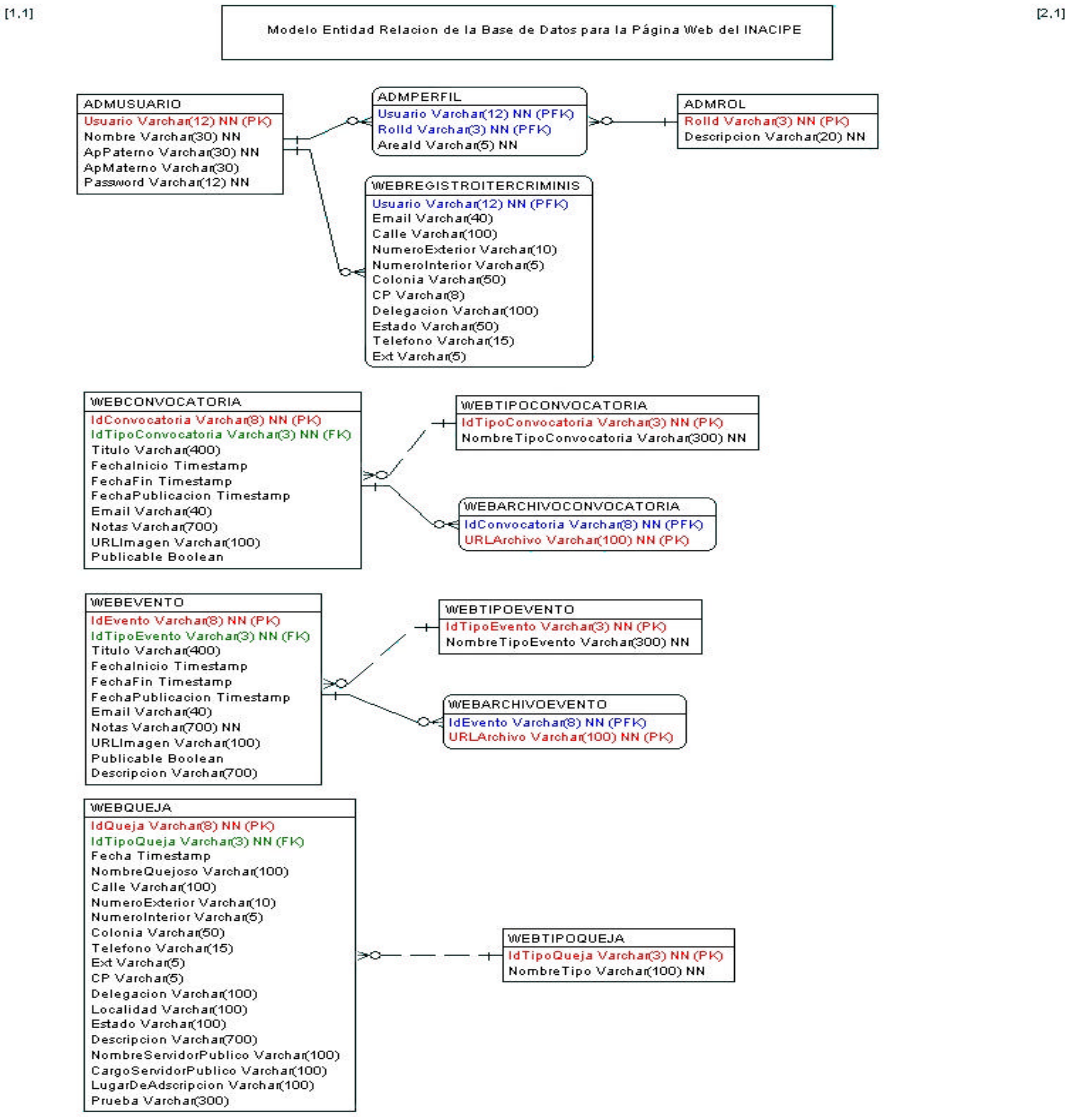
Notas:

El servidor Web, el app server y la base de datos deben estar ejecutándose.

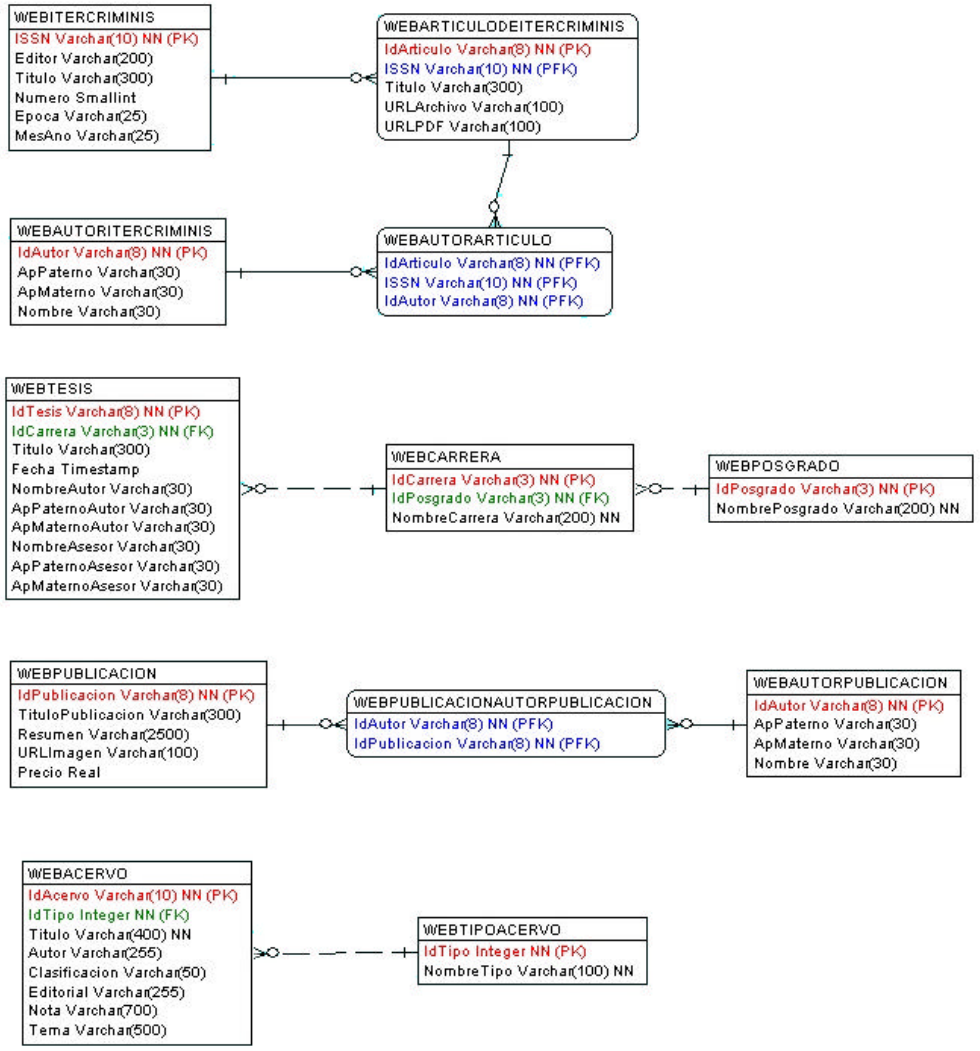
CAPÍTULO 4. Diseño del sitio web del INACIPE

4.1 Diseño de la base de datos

Se construyó un diagrama de entidad relación en, al menos, 3FN (Tercera Forma Normal), que representa el modelo de datos necesario para la implantación de la página de Internet del Instituto.



[1,2] [2,2]

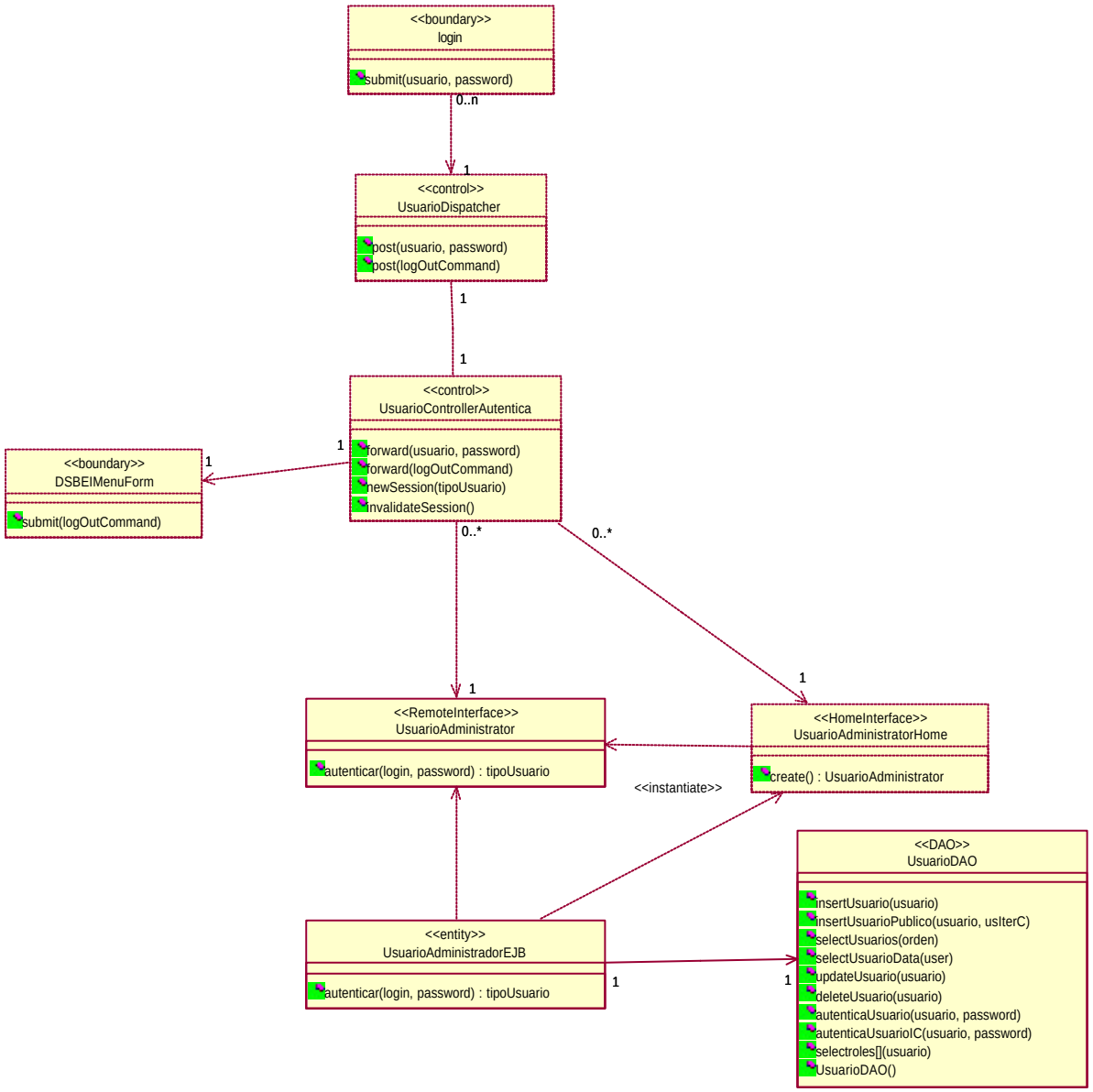

WEBITERCRIMINIS
ISSN Varchar(10) NN (PK)
Editor Varchar(200)
Titulo Varchar(300)
Numero Smallint
Epoca Varchar(25)
MesAno Varchar(25)
WEBARTICULODEITERCRIMINIS
IdArticulo Varchar(8) NN (PK)
ISSN Varchar(10) NN (PFK)
Titulo Varchar(300)
URLArchivo Varchar(100)
URLPDF Varchar(100)
WEBAUTORITERCRIMINIS
IdAutor Varchar(8) NN (PK)
ApPaterno Varchar(30)
ApMaterno Varchar(30)
Nombre Varchar(30)
WEBAUTORARTICULO
IdArticulo Varchar(8) NN (PFK)
ISSN Varchar(10) NN (PFK)
IdAutor Varchar(8) NN (PFK)
WEBTESIS
IdTesis Varchar(8) NN (PK)
IdCarrera Varchar(3) NN (FK)
Titulo Varchar(300)
Fecha Timestamp
NombreAutor Varchar(30)
ApPaternoAutor Varchar(30)
ApMaternoAutor Varchar(30)
NombreAsesor Varchar(30)
ApPaternoAsesor Varchar(30)
ApMaternoAsesor Varchar(30)
WEBCARRERA
IdCarrera Varchar(3) NN (PK)
IdPosgrado Varchar(3) NN (FK)
NombreCarrera Varchar(200) NN
WEBPOSGRADO
IdPosgrado Varchar(3) NN (PK)
NombrePosgrado Varchar(200) NN
WEBPUBLICACION
IdPublicacion Varchar(8) NN (PK)
TituloPublicacion Varchar(300)
Resumen Varchar(2500)
URLImagen Varchar(100)
Precio Real
WEBPUBLICACIONAUTORPUBLICACION
IdAutor Varchar(8) NN (PFK)
IdPublicacion Varchar(8) NN (PFK)
WEBAUTORPUBLICACION
IdAutor Varchar(8) NN (PK)
ApPaterno Varchar(30)
ApMaterno Varchar(30)
Nombre Varchar(30)
WEBACERVO
IdAcervo Varchar(10) NN (PK)
IdTipo Integer NN (FK)
Titulo Varchar(400) NN
Autor Varchar(255)
Clasificacion Varchar(50)
Editorial Varchar(255)
Nota Varchar(700)
Tema Varchar(500)
WEBTIPOACERVO
IdTipo Integer NN (PK)
NombreTipo Varchar(100) NN

4.2 Acceso al sistema

Describe los procesos y las acciones que emplea el sistema para controlar la entrada y salida de los usuarios.

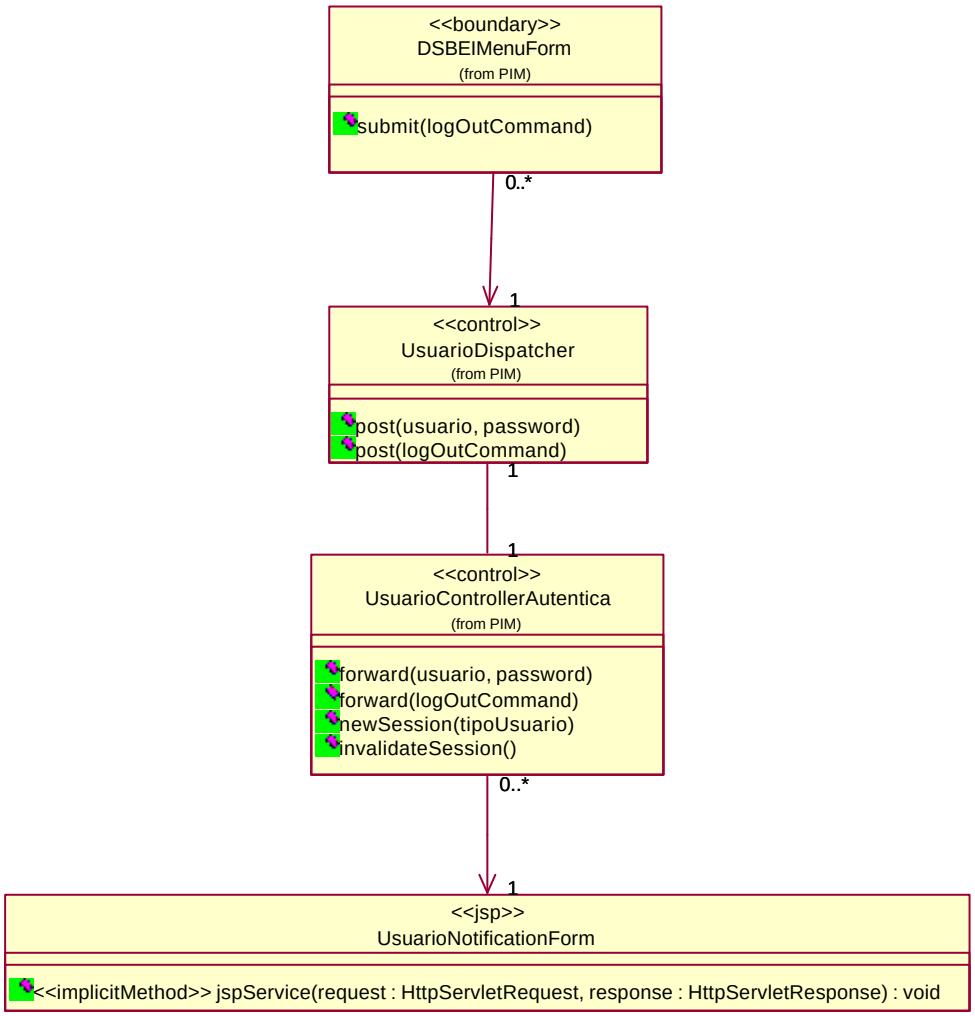
4.2.1 Diagrama de clases del login

El JSP-boundary LOGIN pasa los datos al Servlet UsuarioDispatcher, quien redirecciona al Servlet UsuarioControllerAutentica. Éste último, es cliente del EJB UsuarioAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (UsuarioAdministratorHome) y de InterfaceRemote (UsuarioAdministrator), las cuales a través de la clase DAO UsuarioDAO, en donde se encuentran las operaciones SQL y con base en el resultado de la consulta, da acceso al sistema a través del JSP DSBElMenuForm.


<<boundary>>
login
submit(usuario, password)
0..n
1
<<control>>
UsuarioDispatcher
post(usuario, password)
post(logOutCommand)
1
1
<<control>>
UsuarioControllerAutentica
forward(usuario, password)
forward(logOutCommand)
newSession(tipoUsuario)
invalidateSession()
1
<<boundary>>
DSBEIMenuForm
submit(logOutCommand)
1
0..*
0..*
1
<<RemoteInterface>>
UsuarioAdministrator
autenticar(login, password) : tipoUsuario
1
<<HomeInterface>>
UsuarioAdministratorHome
create() : UsuarioAdministrator
<<instantiate>>
<<DAO>>
UsuarioDAO
insertUsuario(usuario)
insertUsuarioPublico(usuario, usIterC)
selectUsuarios(orden)
selectUsuarioData(user)
updateUsuario(usuario)
deleteUsuario(usuario)
autenticaUsuario(usuario, password)
autenticaUsuarioIC(usuario, password)
selectroles[](usuario)
UsuarioDAO()
<<entity>>
UsuarioAdministradorEJB
autenticar(login, password) : tipoUsuario
1
1

4.2.2 Diagrama de clases del caso de uso logout

EL JSP DSBElMenuForm pasa los datos al Servlet UsuarioDispatcher, el cual redirecciona al ServeIt UsuarioControllerAutentica y manda un mensaje de salida del sistema a través del JSP UsuarioNotificationForm.



4.2.3 Diagrama de secuencia y de colaboración del caso de uso login

El actor DSBel envía sus datos, login y password, en el mensaje submit() para tener acceso al sistema. La clase login reenvía esos datos al UsuarioDispatcher y éste a su vez los envía al controlador UsuarioControllerAutentica. Ésta misma clase invoca al método create de la interfaz Home para obtener una referencia del tipo de la interfaz Remote. Una vez hecho esto, invoca al método autenticar de la clase del Bean UsuarioAdministrador.

Como el usuario es un usuario válido para el sistema, crea una nueva sesión invocando el mensaje newSession(), para posteriormente redireccionar al usuario a la interfaz DSBElMenuForm que es cuando el usuario ya ha sido validado en el sistema y ha entrado a éste.

Diagrama de secuencia del caso de uso login

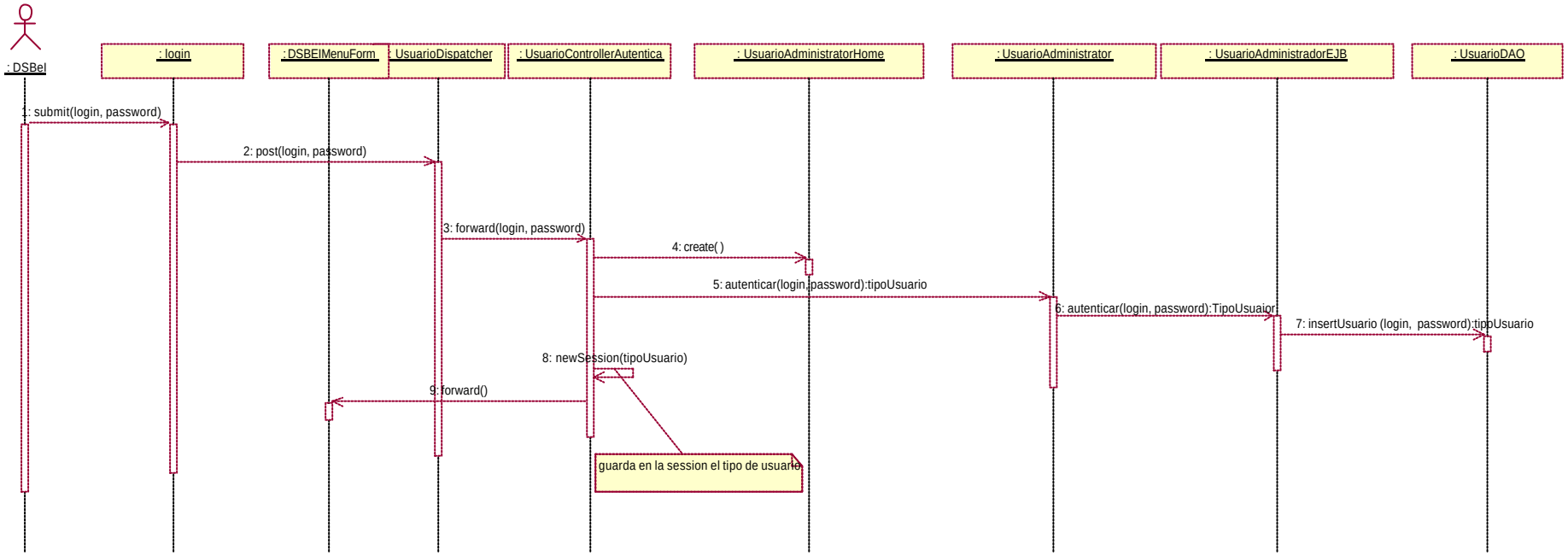
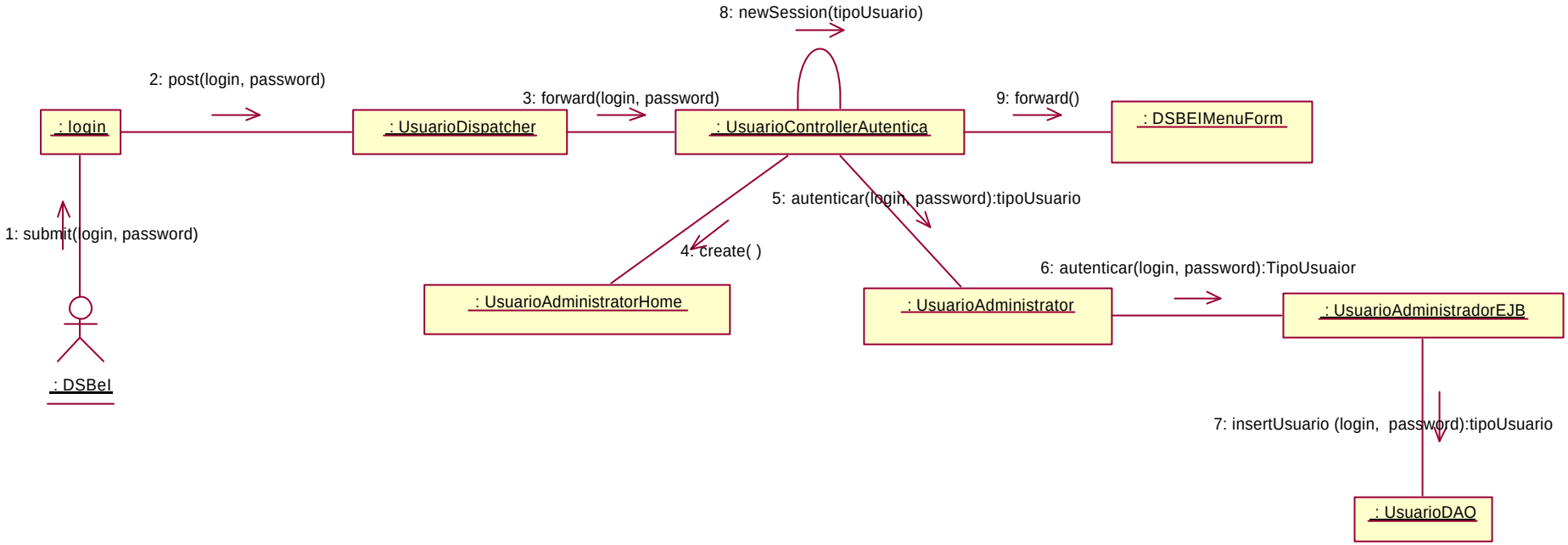


Diagrama de colaboración del caso de uso login



4.2.4 Diagrama de secuencia y colaboración del caso de uso logout

El usuario DSBei envía la solicitud de logout mediante un mensaje submit() para salir del sistema. La interfaz DSBEIMenuForm reenvía los datos a la clase UsuarioDispatcher y éste a su vez los envía al controlador UsuarioControllerAutentica. Ésta clase invoca al método invalidateSession la cual terminara la sesión del usuario y envía un mensaje informando que ha salido del sistema.

Diagrama de secuencia del caso de uso logout

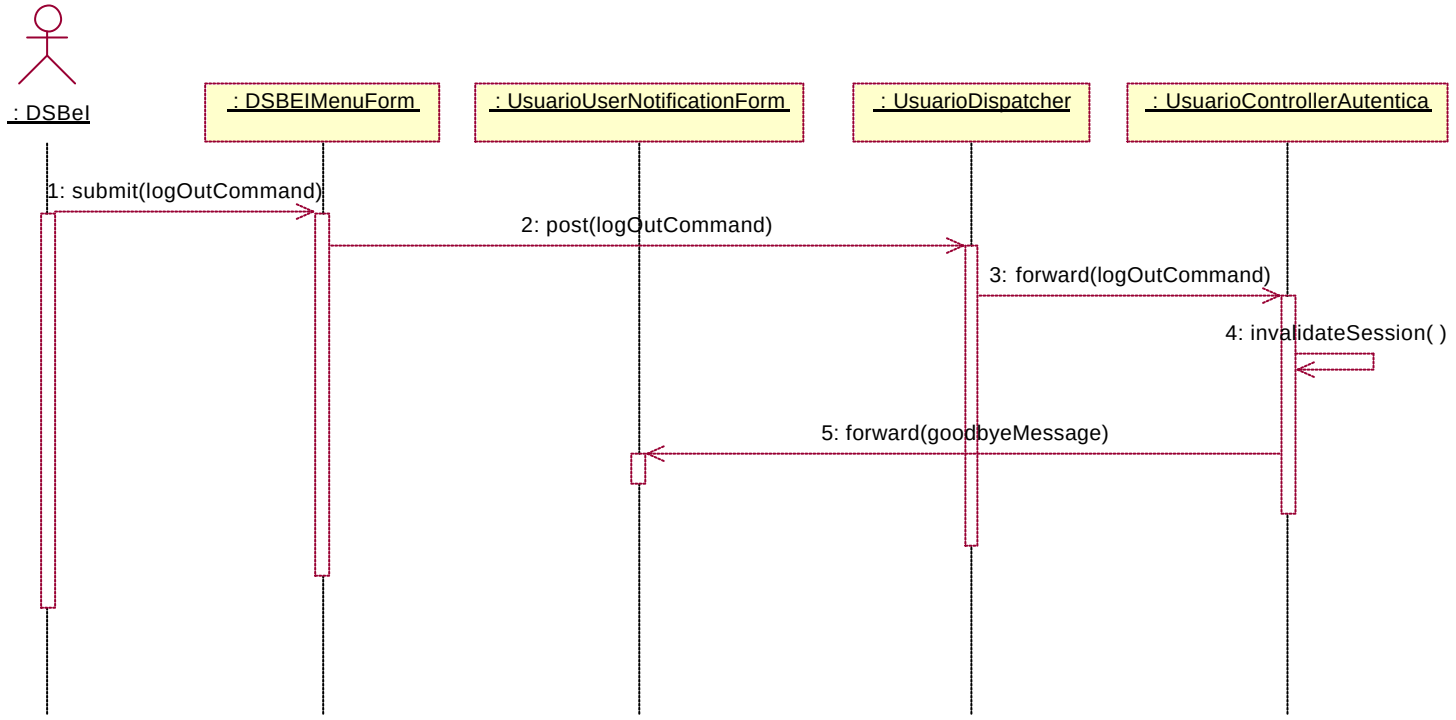
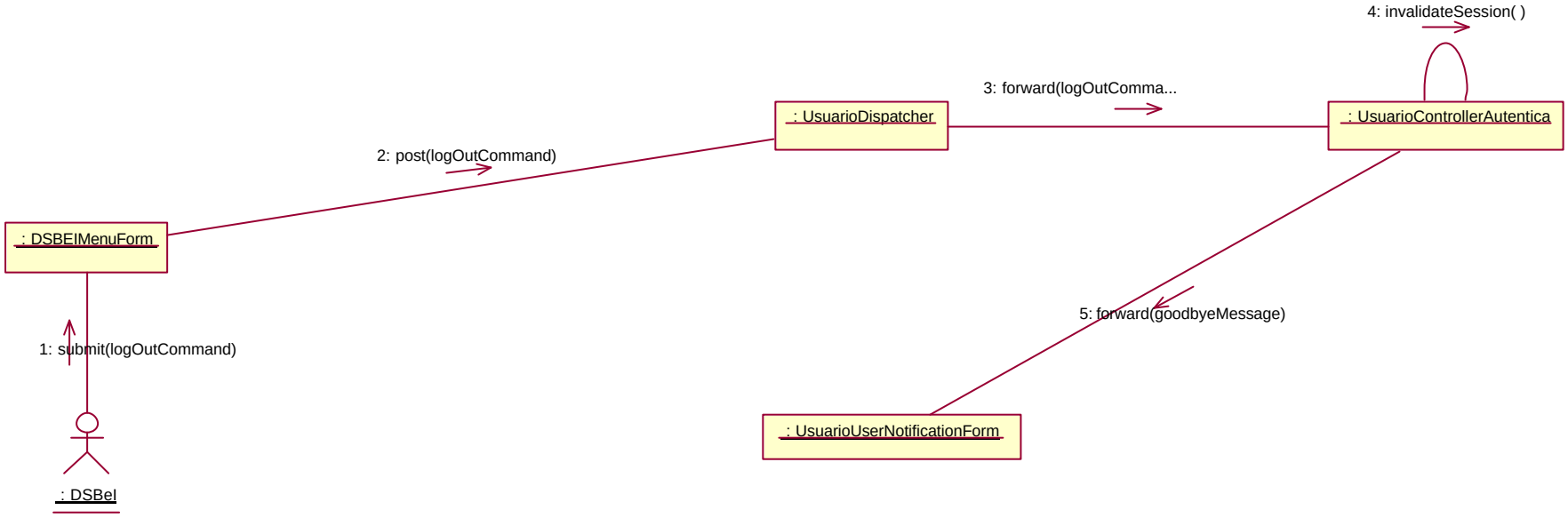


Diagrama de colaboración del caso de uso logout

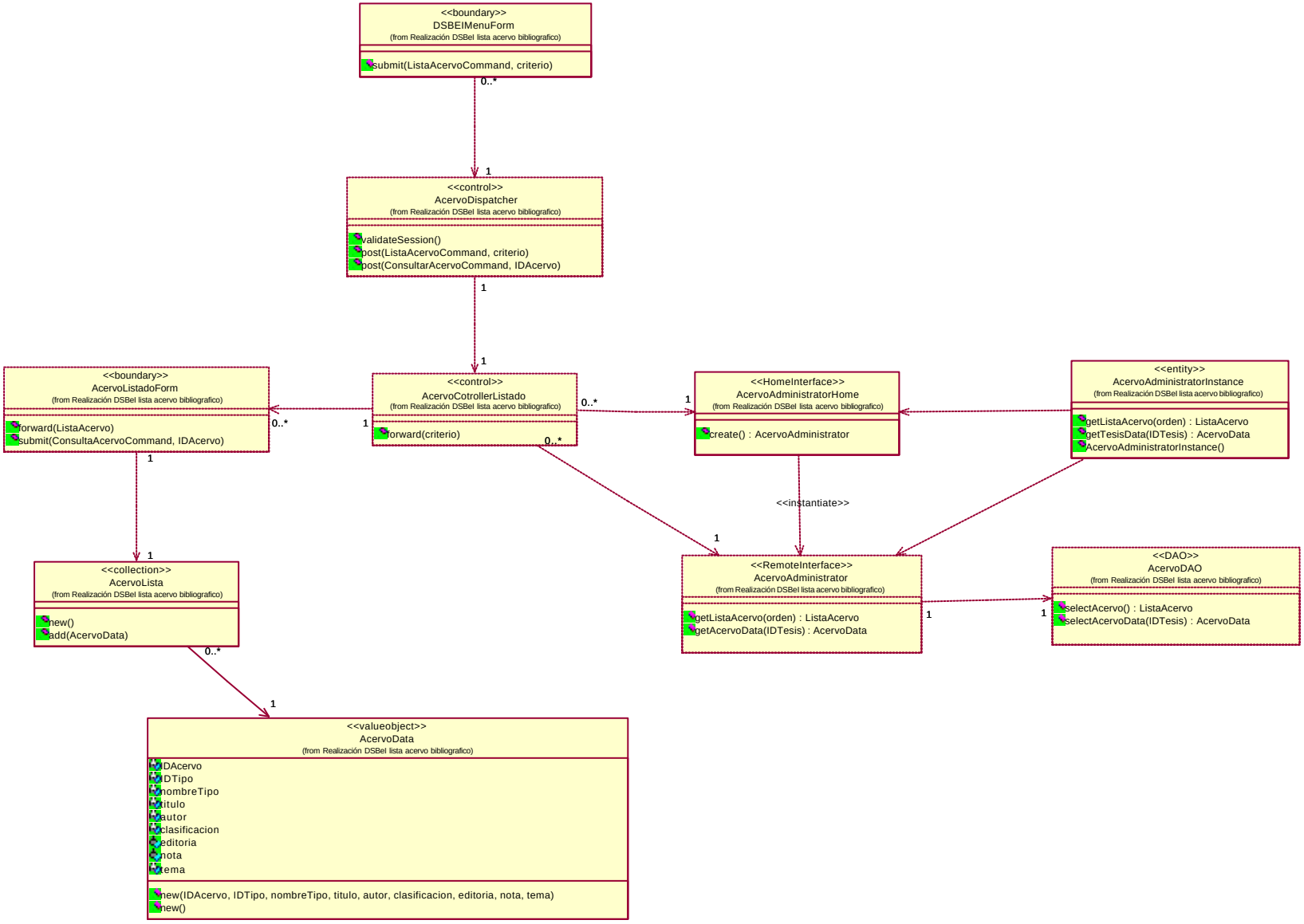


4.3 Acervo bibliográfico

Describe los procesos y operaciones que el sistema emplea para el manejo de los datos del acervo bibliográfico.

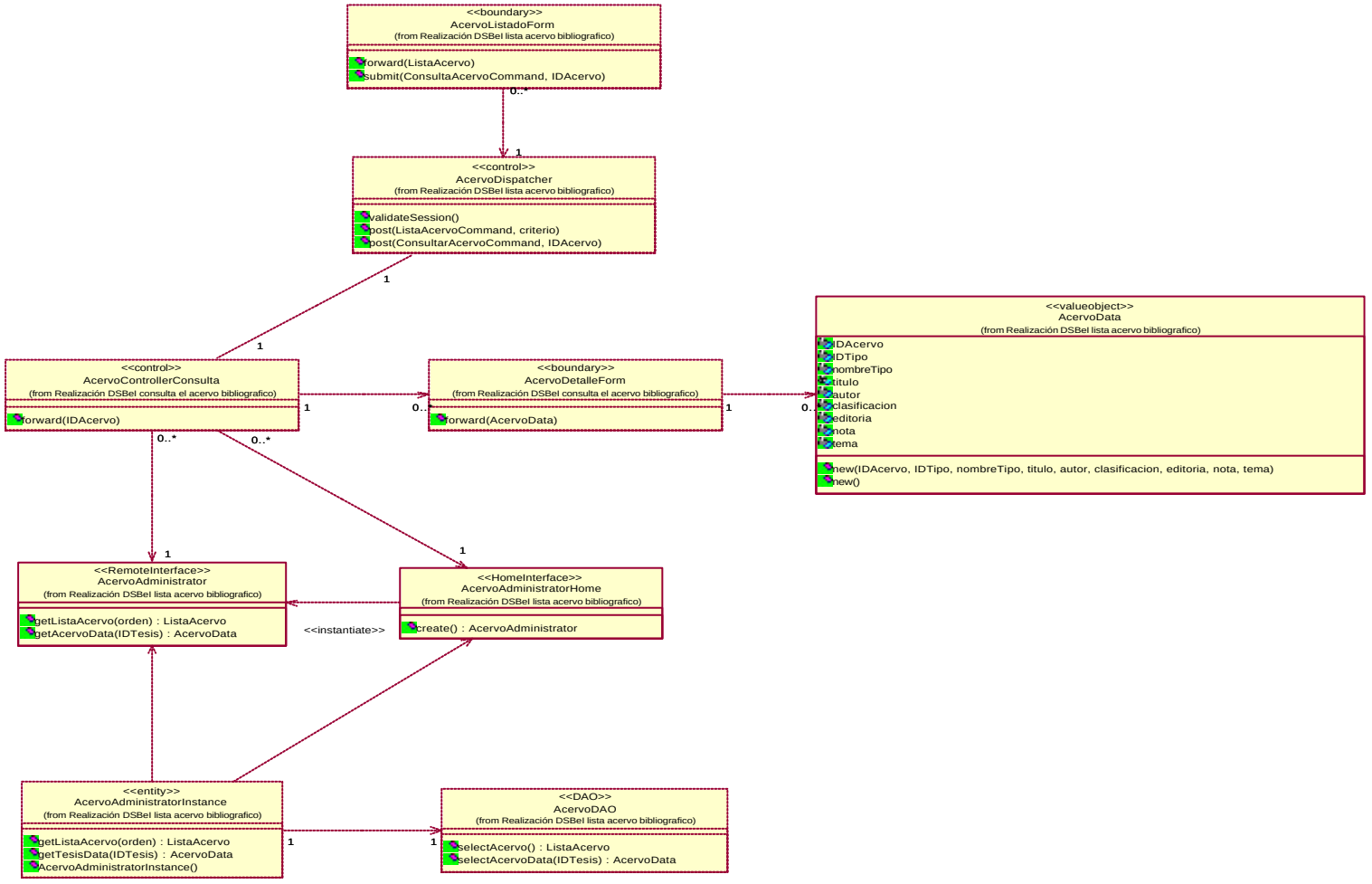
4.3.1 Diagrama de clases de Consulta acervo bibliográfico

EL JSP AcervoListadoForm pasa los datos al Servlet AcervoDispatcher, el cual redirecciona los datos al Servlet AcervoControllerConsulta. Éste último, es cliente del EJB AcervoAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (AcervoAdministratorHome) y de InterfaceRemote (AcervoAdministrator), las cuales a través de la clase DAO AcervoDAO, quien tiene las operaciones SQL, obtiene al objeto Entity AcervoData, que es empleado por el JSP AcervoControllerConsulta para mostrar los datos a través del JSP AcervoDetalleForm.


<<boundary>>
DSBElMenuForm
(from Realización DSBel lista acervo bibliografico)
submit(ListaAcervoCommand, criterio)
0..*
1
<<control>>
AcervoDispatcher
(from Realización DSBel lista acervo bibliografico)
validateSession()
post(ListaAcervoCommand, criterio)
post(ConsultarAcervoCommand, IDAcervo)
1
1
<<boundary>>
AcervoListadoForm
(from Realización DSBel lista acervo bibliografico)
forward(ListaAcervo)
submit(ConsultaAcervoCommand, IDAcervo)
0..*
1
<<control>>
AcervoCotrollerListado
(from Realización DSBel lista acervo bibliografico)
forward(criterio)
1
0..*
0..*
<<HomeInterface>>
AcervoAdministratorHome
(from Realización DSBel lista acervo bibliografico)
create() : AcervoAdministrator
1
<<entity>>
AcervoAdministratorInstance
(from Realización DSBel lista acervo bibliografico)
getListaAcervo(orden) : ListaAcervo
getTesisData(IDTesis) : AcervoData
AcervoAdministratorInstance()
<<instantiate>>
1
1
<<collection>>
AcervoLista
(from Realización DSBel lista acervo bibliografico)
new()
add(AcervoData)
0..*
1
<<RemoteInterface>>
AcervoAdministrator
(from Realización DSBel lista acervo bibliografico)
getListaAcervo(orden) : ListaAcervo
getAcervoData(IDTesis) : AcervoData
1
<<DAO>>
AcervoDAO
(from Realización DSBel lista acervo bibliografico)
selectAcervo() : ListaAcervo
selectAcervoData(IDTesis) : AcervoData
1
<<valueobject>>
AcervoData
(from Realización DSBel lista acervo bibliografico)
IDAcervo
IDTipo
nombreTipo
titulo
autor
clasificacion
editoria
nota
tema
new(IDAcervo, IDTipo, nombreTipo, titulo, autor, clasificacion, editoria, nota, tema)
new()

4.3.2 Diagrama de clases de lista acervo bibliográfico

EL JSP-boundary DSBElMenuForm pasa los datos al Servlet AcervoDispatcher, el cual redirecciona hacia otro Servlet, el AcervoControllerListado. Éste último, es cliente del EJB AcervoAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (AcervoAdministratorHome) y de InterfaceRemote (AcervoAdministrator), las cuales a través de la clase DAO AcervoDAO, quien tiene las operaciones SQL, obtiene al objeto collection AcervoLista que contiene a la clase Entity AcervoData, así, el Servlet AcervoControllerListado pasa el objeto Collection AcervoLista al JSP AcervoListadoForm, quien muestra el resultado final en AcervoListadoForm.


<<boundary>>
AcervoListadoForm
(from Realización DSBel lista acervo bibliografico)
forward(ListaAcervo)
submit(ConsultaAcervoCommand, IDAcervo)
<<control>>
AcervoDispatcher
(from Realización DSBel lista acervo bibliografico)
validateSession()
post(ListaAcervoCommand, criterio)
post(ConsultarAcervoCommand, IDAcervo)
<<valueobject>>
AcervoData
(from Realización DSBel lista acervo bibliografico)
IDAcervo
IDTipo
nombreTipo
titulo
autor
clasificacion
editoria
nota
tema
new(IDAcervo, IDTipo, nombreTipo, titulo, autor, clasificacion, editoria, nota, tema)
new()
<<control>>
AcervoControllerConsulta
(from Realización DSBel consulta el acervo bibliografico)
forward(IDAcervo)
<<boundary>>
AcervoDetalleForm
(from Realización DSBel consulta el acervo bibliografico)
forward(AcervoData)
<<RemoteInterface>>
AcervoAdministrator
(from Realización DSBel lista acervo bibliografico)
getListaAcervo(orden) : ListaAcervo
getAcervoData(IDTesis) : AcervoData
<<HomeInterface>>
AcervoAdministratorHome
(from Realización DSBel lista acervo bibliografico)
create() : AcervoAdministrator
<<instantiate>>
<<entity>>
AcervoAdministratorInstance
(from Realización DSBel lista acervo bibliografico)
getListaAcervo(orden) : ListaAcervo
getTesisData(IDTesis) : AcervoData
AcervoAdministratorInstance()
<<DAO>>
AcervoDAO
(from Realización DSBel lista acervo bibliografico)
selectAcervo() : ListaAcervo
selectAcervoData(IDTesis) : AcervoData
0..*
1

4.3.3 Diagrama de secuencia y de colaboración de consulta acervo bibliográfico

El actor DSBEI envía los datos de solicitud de consulta del acervo y el identificador del ejemplar a consultar en el mensaje Submit(). La clase AcervoListadoForn reenvía estos datos a la clase AcervoDispatcher, el cual ejecuta el método validateSession donde al ser valida la sesión reenvía dichos datos a la clase AcervoControllerConsulta. La clase AcervoControllerConsulta envía los datos a la clase AcervoAdministratorHome quien se encargara de crear una instancia de la clase AcervoAdministrator y ésta ejecuta el método getAcervoData de ésta manera obtendrá la información en un objeto de la clase AcervoData y se enviará a la interfaz AcervoDetalleForm para presentar la información al actor.

Diagrama de secuencia de consulta acervo bibliográfico

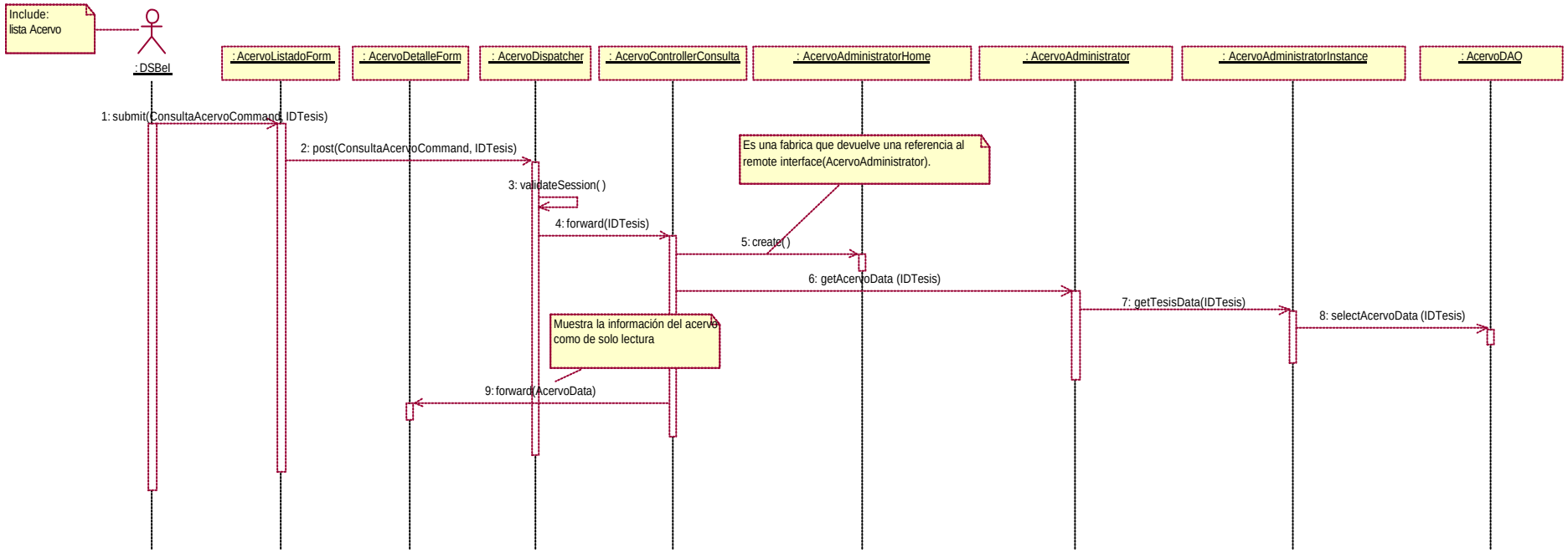
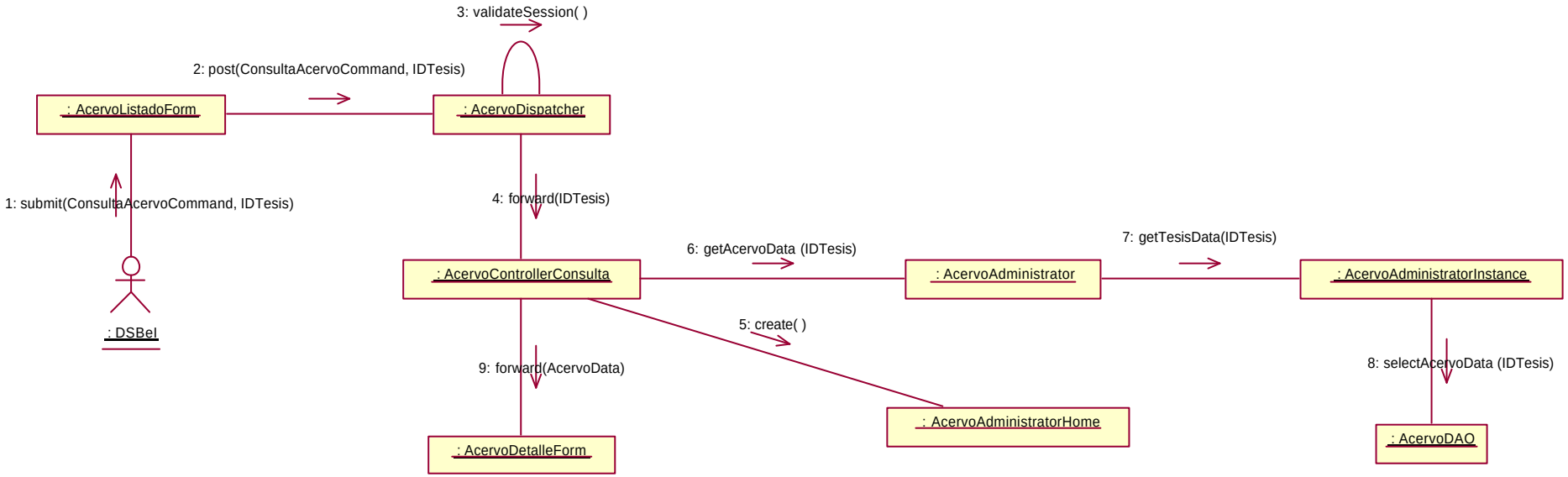


Diagrama de colaboración de consulta acervo bibliográfico



4.3.4 Diagrama de secuencia y de colaboración de lista acervo bibliográfico

EL actor DSBel envía la solicitud de listado del acervo mediante el mensaje submit desde la interfaz DSBEIMenuForm. En el mensaje se indica el criterio por el cual se realiza la selección del acervo para el listado. La clase AcervoDispatcher ejecuta el método validateSession donde al ser valida la sesión reenvía dichos datos a la clase AcervoControllerListados. La clase AcervoControllerListados envía el criterio de selección a la clase AcervoAdministratorHome quien se encargará de crear una instancia de la clase AcervoAdministrator y ésta ejecuta el método getListaAcervo con el cual obtendrá la información empleando el objeto de la clase AcervoData, quien genera un conjunto de objetos de la clase AcervoDato contenidos en una instancia de la clase AcervoLista. Finalmente el objeto de la clase AcervoLista que contiene la información obtenida de la consulta a la base de datos se envía a la clase AcervoListadoForm para presentar la información al actor.

Diagrama de secuencia de lista acervo bibliográfico

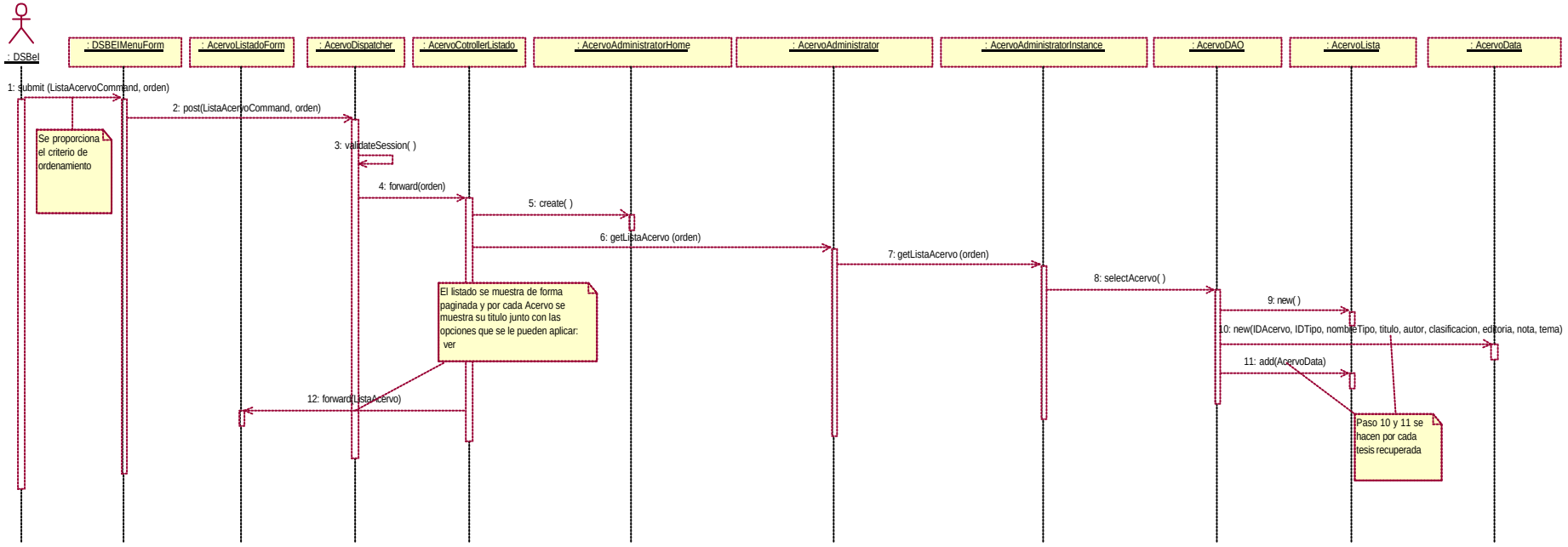
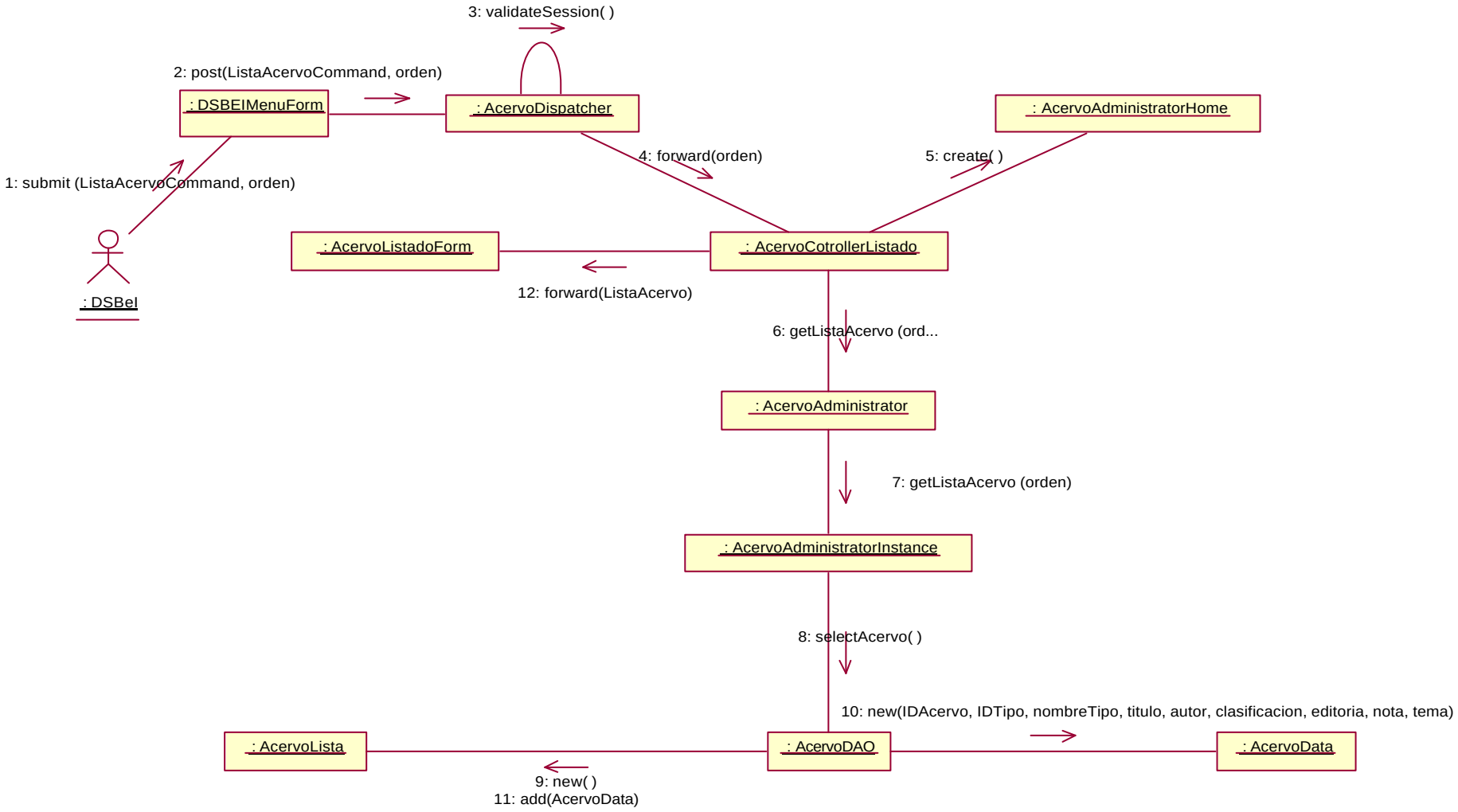


Diagrama de colaboración de lista acervo bibliográfico

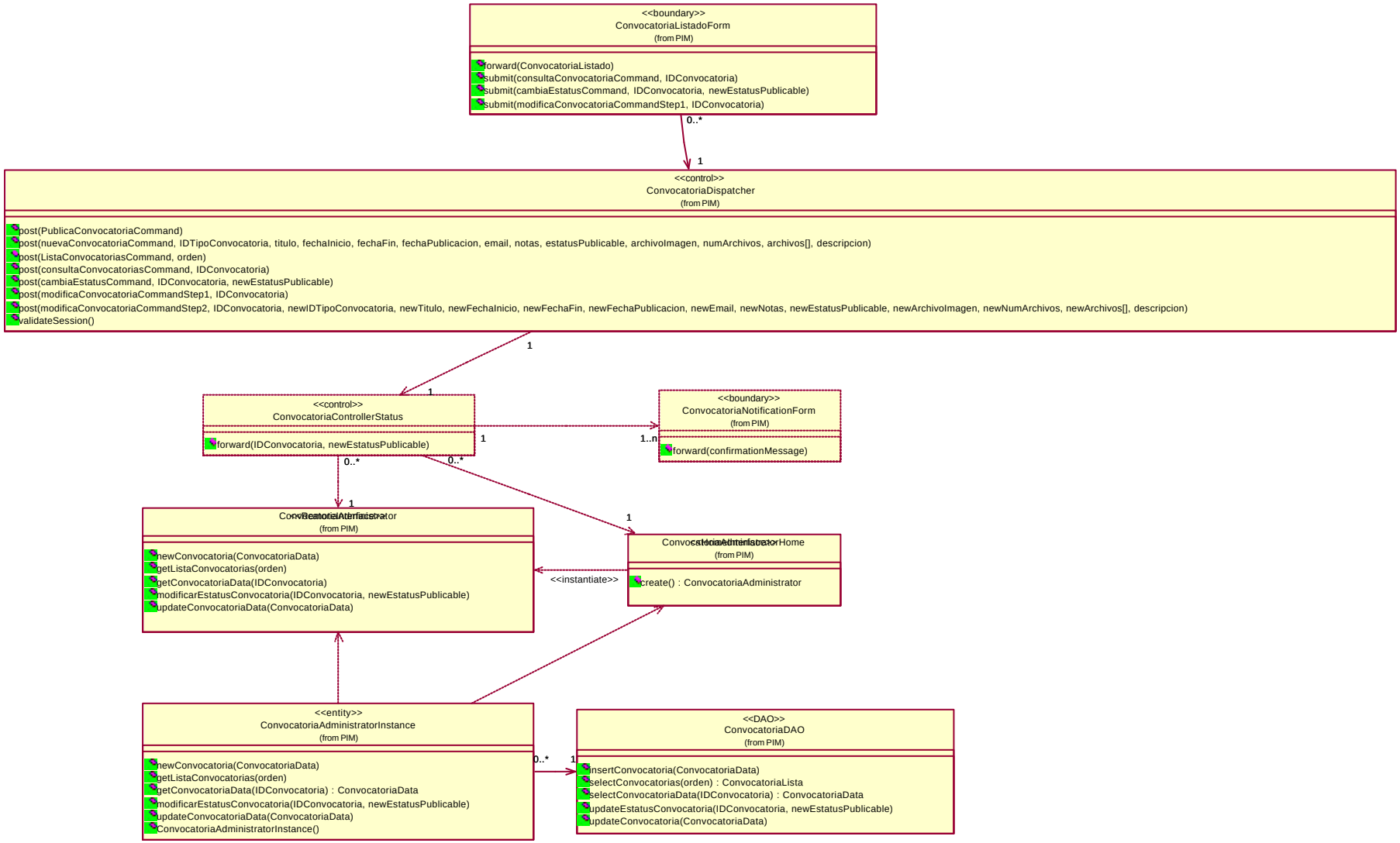


4.4 Convocatorias

Describe las operaciones y funciones con que cuenta el sistema para el manejo y la manipulación de la información de las convocatorias.

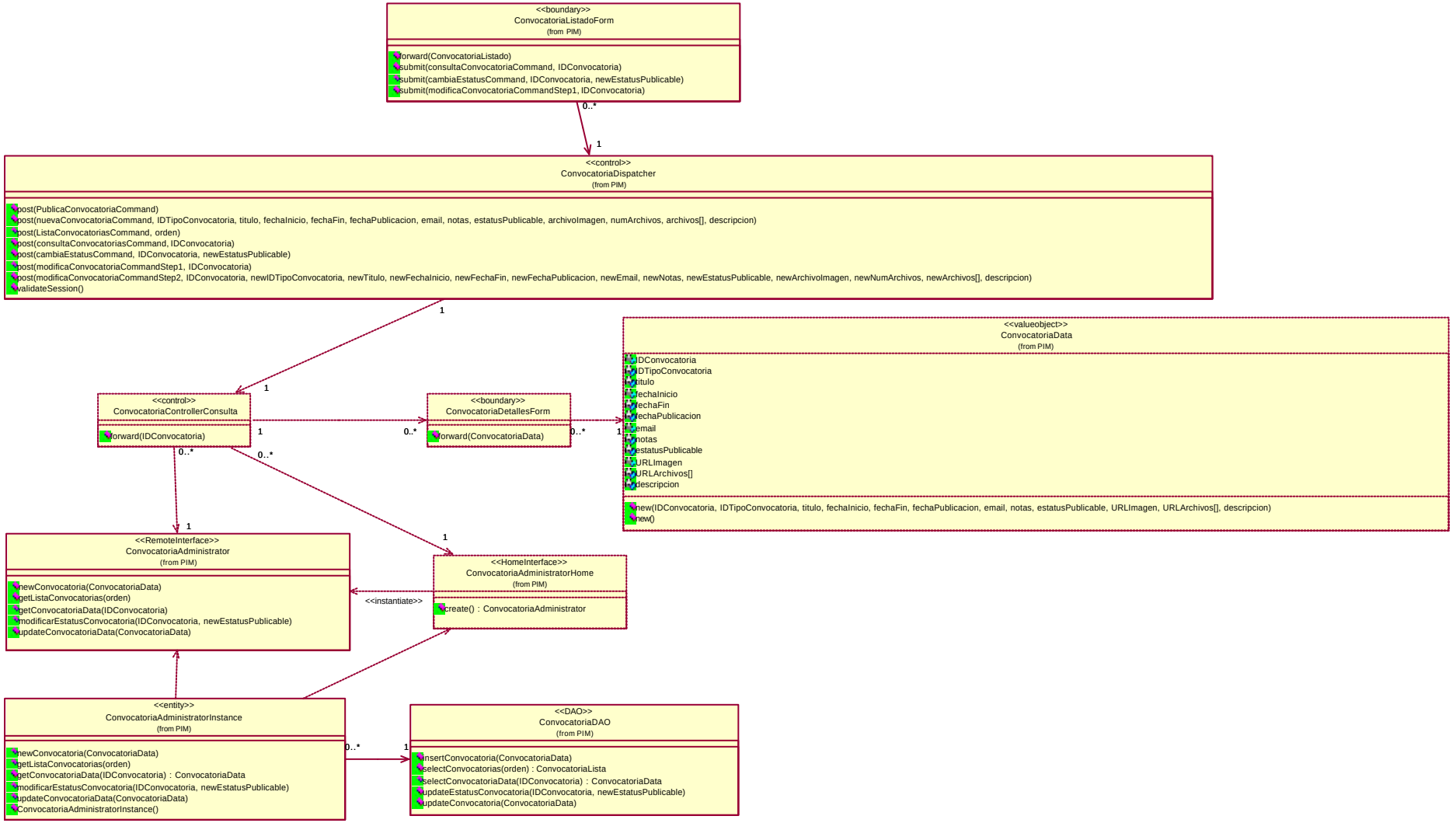
4.4.1 Diagrama de clases de cambia estatus a convocatorias publicadas

EL JSP-boundary ConvocatoriaListadoForm pasa los datos al Servlet ConvocatoriaDispatcher, éste redirecciona hacia otro Servlet, el ConvocatoriaControllerStatus., éste último, es cliente del EJB ConvocatoriaAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (ConvocatoriaAdministratorHome) y de InterfaceRemote (ConvocatoriaAdministrator), las cuales a través de la clase DAO ConvocatoriaDAO, quien tiene las operaciones SQL, se realiza el cambio del estatus en el evento. Posteriormente el Servlet ConvocatoriaControllerStatus envía una notificación de que se realizó el cambio a través del JSP ConvocatoriaNotificationForm.


<<boundary>>
ConvocatoriaListadoForm
(from PIM)
forward(ConvocatoriaListado)
submit(consultaConvocatoriaCommand, IDConvocatoria)
submit(cambiaEstatusCommand, IDConvocatoria, newEstatusPublicable)
submit(modificaConvocatoriaCommandStep1, IDConvocatoria)
0..*
1
<<control>>
ConvocatoriaDispatcher
(from PIM)
post(PublicaConvocatoriaCommand)
post(nuevaConvocatoriaCommand, IDTipoConvocatoria, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[], descripcion)
post(ListaConvocatoriasCommand, orden)
post(consultaConvocatoriasCommand, IDConvocatoria)
post(cambiaEstatusCommand, IDConvocatoria, newEstatusPublicable)
post(modificaConvocatoriaCommandStep1, IDConvocatoria)
post(modificaConvocatoriaCommandStep2, IDConvocatoria, newIDTipoConvocatoria, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[], descripcion)
validateSession()
1
1
<<control>>
ConvocatoriaControllerStatus
forward(IDConvocatoria, newEstatusPublicable)
1
<<boundary>>
ConvocatoriaNotificationForm
(from PIM)
1..n
forward(confirmationMessage)
0..*
0..*
1
1
(from PIM)
newConvocatoria(ConvocatoriaData)
getListaConvocatorias(orden)
getConvocatoriaData(IDConvocatoria)
modificarEstatusConvocatoria(IDConvocatoria, newEstatusPublicable)
updateConvocatoriaData(ConvocatoriaData)
<<instantiate>>
(from PIM)
create() : ConvocatoriaAdministrator
<<entity>>
ConvocatoriaAdministratorInstance
(from PIM)
newConvocatoria(ConvocatoriaData)
getListaConvocatorias(orden)
getConvocatoriaData(IDConvocatoria) : ConvocatoriaData
modificarEstatusConvocatoria(IDConvocatoria, newEstatusPublicable)
updateConvocatoriaData(ConvocatoriaData)
ConvocatoriaAdministratorInstance()
0..*
1
<<DAO>>
ConvocatoriaDAO
(from PIM)
insertConvocatoria(ConvocatoriaData)
selectConvocatorias(orden) : ConvocatoriaLista
selectConvocatoriaData(IDConvocatoria) : ConvocatoriaData
updateEstatusConvocatoria(IDConvocatoria, newEstatusPublicable)
updateConvocatoria(ConvocatoriaData)

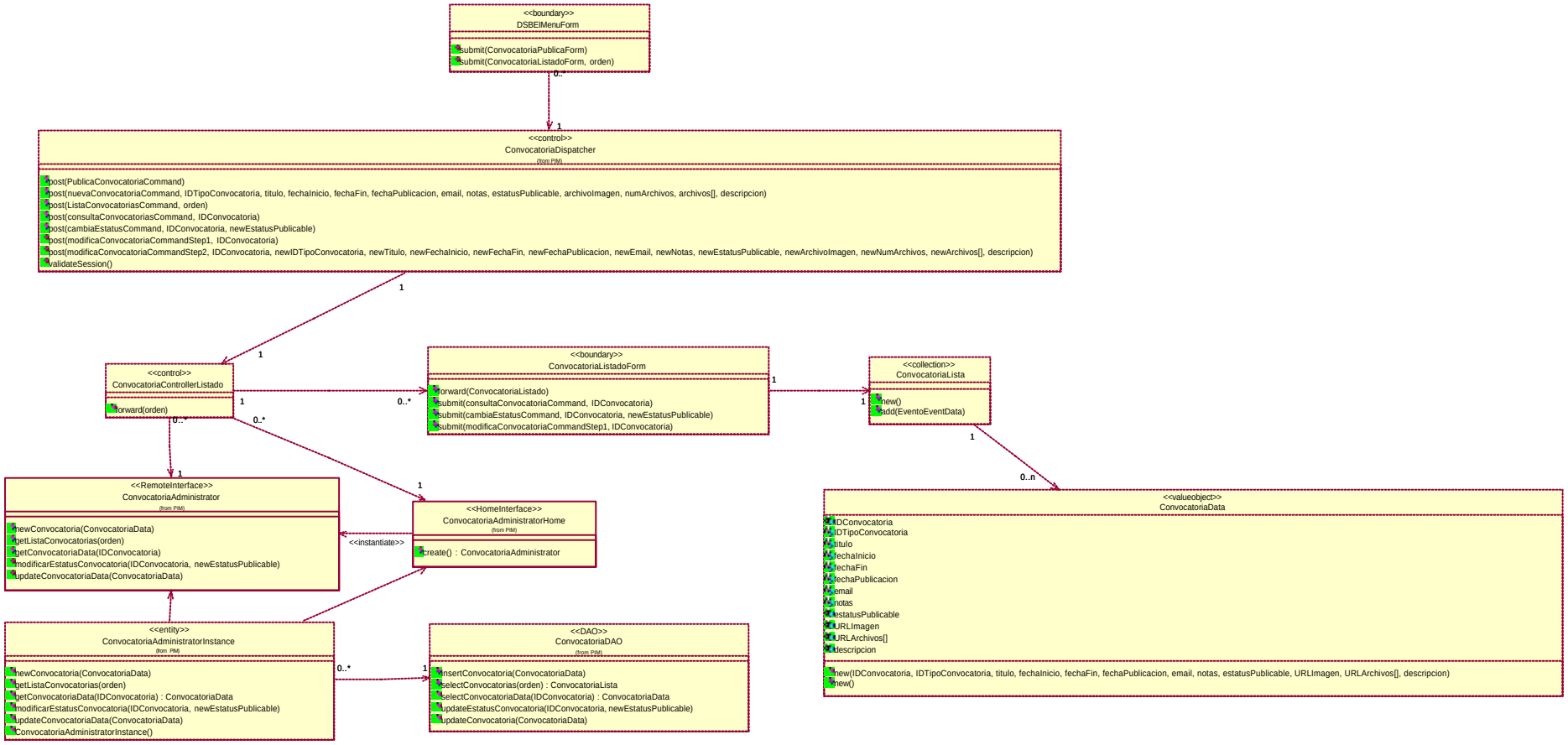
4.4.2 Diagrama de clases de consulta información completa de convocatorias

EL JSP ConvocatoriaListadoForm pasa los datos al Servlet ConvocatoriaDispatcher, el cual redirecciona los datos al Servlet ConvocatoriaControllerConsulta, éste último, es cliente del EJB ConvocatoriaAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (ConvocatoriaAdministratorHome) y de InterfaceRemote (ConvocatoriaAdministrator), las cuales a través de la clase DAO ConvocatoriaDAO, quien tiene las operaciones SQL, obtiene al objeto Entity ConvocatoriaData, con éste, el JSP ConvocatoriaControllerConsulta muestra los datos a través del JSP ConvocatoriaDetalleForm.


<<boundary>>
ConvocatoriaListadoForm
(from PIM)
forward(ConvocatoriaListado)
submit(consultaConvocatoriaCommand, IDConvocatoria)
submit(cambiaEstatusCommand, IDConvocatoria, newEstatusPublicable)
submit(modificaConvocatoriaCommandStep1, IDConvocatoria)
0..*
1
<<control>>
ConvocatoriaDispatcher
(from PIM)
post(PublicaConvocatoriaCommand)
post(nuevaConvocatoriaCommand, IDTipoConvocatoria, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[], descripcion)
post(ListaConvocatoriasCommand, orden)
post(consultaConvocatoriasCommand, IDConvocatoria)
post(cambiaEstatusCommand, IDConvocatoria, newEstatusPublicable)
post(modificaConvocatoriaCommandStep1, IDConvocatoria)
post(modificaConvocatoriaCommandStep2, IDConvocatoria, newIDTipoConvocatoria, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[], descripcion)
validateSession()
1
1
<<valueobject>>
ConvocatoriaData
(from PIM)
IDConvocatoria
IDTipoConvocatoria
titulo
fechaInicio
fechaFin
fechaPublicacion
email
notas
estatusPublicable
URLImagen
URLArchivos[]
descripcion
new(IDConvocatoria, IDTipoConvocatoria, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, URLImagen, URLArchivos[], descripcion)
new()
<<control>>
ConvocatoriaControllerConsulta
forward(IDConvocatoria)
1
0..*
<<boundary>>
ConvocatoriaDetallesForm
forward(ConvocatoriaData)
0..*
1
0..*
0..*
1
1
<<RemoteInterface>>
ConvocatoriaAdministrator
(from PIM)
newConvocatoria(ConvocatoriaData)
getListaConvocatorias(orden)
getConvocatoriaData(IDConvocatoria)
modificarEstatusConvocatoria(IDConvocatoria, newEstatusPublicable)
updateConvocatoriaData(ConvocatoriaData)
<<instantiate>>
<<HomeInterface>>
ConvocatoriaAdministratorHome
(from PIM)
create() : ConvocatoriaAdministrator
<<entity>>
ConvocatoriaAdministratorInstance
(from PIM)
newConvocatoria(ConvocatoriaData)
getListaConvocatorias(orden)
getConvocatoriaData(IDConvocatoria) : ConvocatoriaData
modificarEstatusConvocatoria(IDConvocatoria, newEstatusPublicable)
updateConvocatoriaData(ConvocatoriaData)
ConvocatoriaAdministratorInstance()
0..*
1
<<DAO>>
ConvocatoriaDAO
(from PIM)
insertConvocatoria(ConvocatoriaData)
selectConvocatorias(orden) : ConvocatoriaLista
selectConvocatoriaData(IDConvocatoria) : ConvocatoriaData
updateEstatusConvocatoria(IDConvocatoria, newEstatusPublicable)
updateConvocatoria(ConvocatoriaData)

4.4.3 Diagrama de clases de lista convocatorias

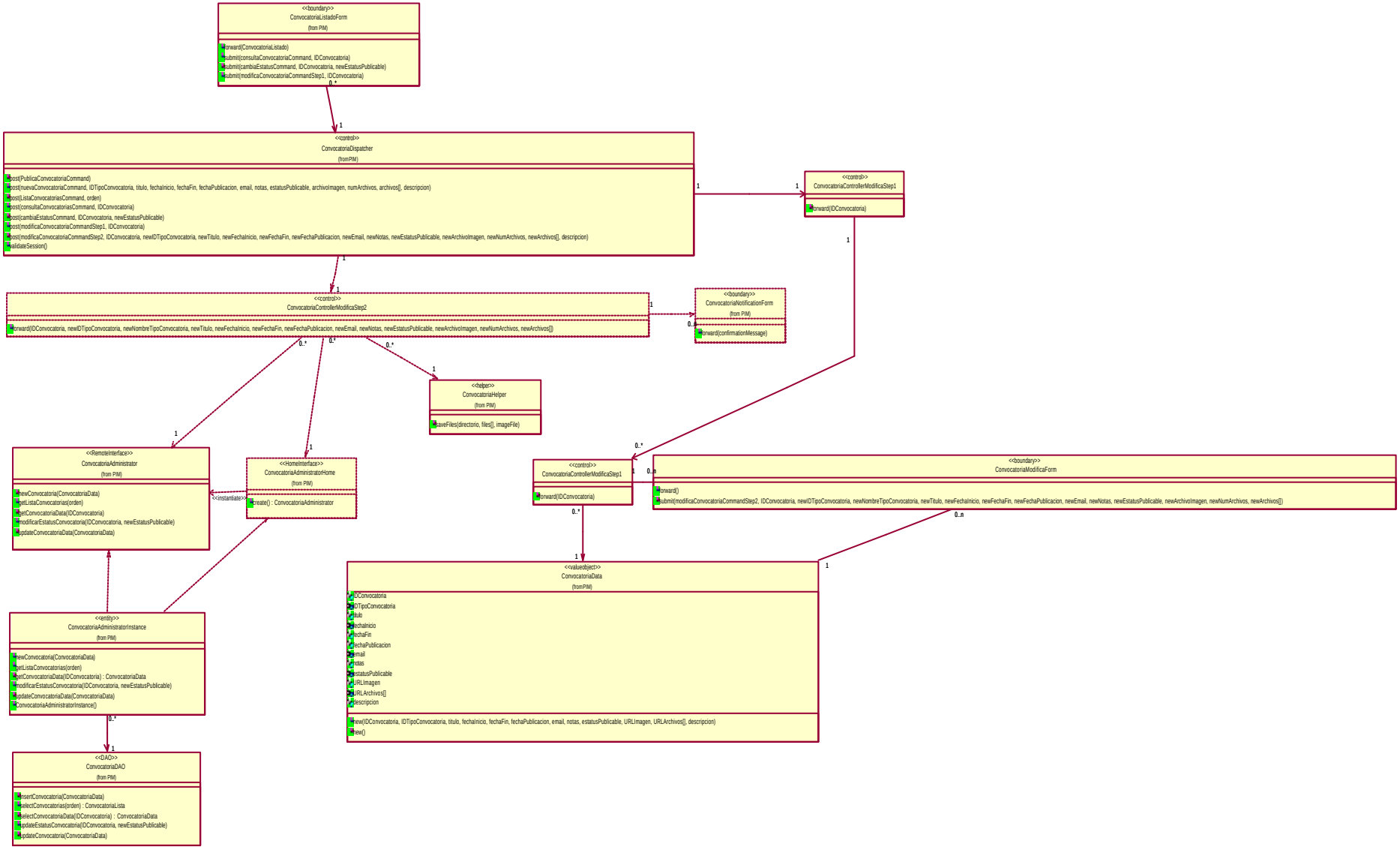
EL JSP-boundary DSBElMenuForm pasa los datos al Servlet ConvocatoriaDispatcher, éste redirecciona hacia otro Servlet, el ConvocatoriaControllerListado, éste último, es cliente del EJB ConvocatoriaAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (ConvocatoriaAdministratorHome) y de InterfaceRemote (ConvocatoriaAdministrator), las cuales a través de la clase DAO ConvocatoriaDAO, quien tiene las operaciones SQL, obtiene al objeto collection ConvocatoriaLista que contiene a la clase Entity ConvocatoriaData, así, el Servlet ConvocatoriaControllerListado pasa el objeto Collection ConvocatoriaLista al JSP ConvocatoriaListadoForm, quien finalmente muestra el listado en la interfaz.


<<boundary>>
DSBEIMenuForm
submit(ConvocatoriaPublicaForm)
submit(ConvocatoriaListadoForm, orden)
<<control>>
ConvocatoriaDispatcher
(from PIM)
post(PublicaConvocatoriaCommand)
post(nuevaConvocatoriaCommand, IDTipoConvocatoria, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[], descripcion)
post(ListaConvocatoriasCommand, orden)
post(consultaConvocatoriasCommand, IDConvocatoria)
post(cambiaEstatusCommand, IDConvocatoria, newEstatusPublicable)
post(modificaConvocatoriaCommandStep1, IDConvocatoria)
post(modificaConvocatoriaCommandStep2, IDConvocatoria, newIDTipoConvocatoria, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[], descripcion)
validateSession()
<<control>>
ConvocatoriaControllerListado
forward(orden)
<<boundary>>
ConvocatoriaListadoForm
forward(ConvocatoriaListado)
submit(consultaConvocatoriaCommand, IDConvocatoria)
submit(cambiaEstatusCommand, IDConvocatoria, newEstatusPublicable)
submit(modificaConvocatoriaCommandStep1, IDConvocatoria)
<<collection>>
ConvocatoriaLista
new()
add(EventoEventData)
<<RemoteInterface>>
ConvocatoriaAdministrator
(from PIM)
newConvocatoria(ConvocatoriaData)
getListaConvocatorias(orden)
getConvocatoriaData(IDConvocatoria)
modificarEstatusConvocatoria(IDConvocatoria, newEstatusPublicable)
updateConvocatoriaData(ConvocatoriaData)
<<HomeInterface>>
ConvocatoriaAdministratorHome
(from PIM)
create() : ConvocatoriaAdministrator
<<instantiate>>
<<valueobject>>
ConvocatoriaData
IDConvocatoria
IDTipoConvocatoria
titulo
fechaInicio
fechaFin
fechaPublicacion
email
notas
estatusPublicable
URLImagen
URLArchivos[]
descripcion
new(IDConvocatoria, IDTipoConvocatoria, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, URLImagen, URLArchivos[], descripcion)
new()
<<entity>>
ConvocatoriaAdministratorInstance
(from PIM)
newConvocatoria(ConvocatoriaData)
getListaConvocatorias(orden)
getConvocatoriaData(IDConvocatoria) : ConvocatoriaData
modificarEstatusConvocatoria(IDConvocatoria, newEstatusPublicable)
updateConvocatoriaData(ConvocatoriaData)
ConvocatoriaAdministratorInstance()
<<DAO>>
ConvocatoriaDAO
(from PIM)
insertConvocatoria(ConvocatoriaData)
selectConvocatorias(orden) : ConvocatoriaLista
selectConvocatoriaData(IDConvocatoria) : ConvocatoriaData
updateEstatusConvocatoria(IDConvocatoria, newEstatusPublicable)
updateConvocatoria(ConvocatoriaData)
0..*
1
0..n

4.4.4 Diagrama de clases de modifica datos de convocatorias publicadas

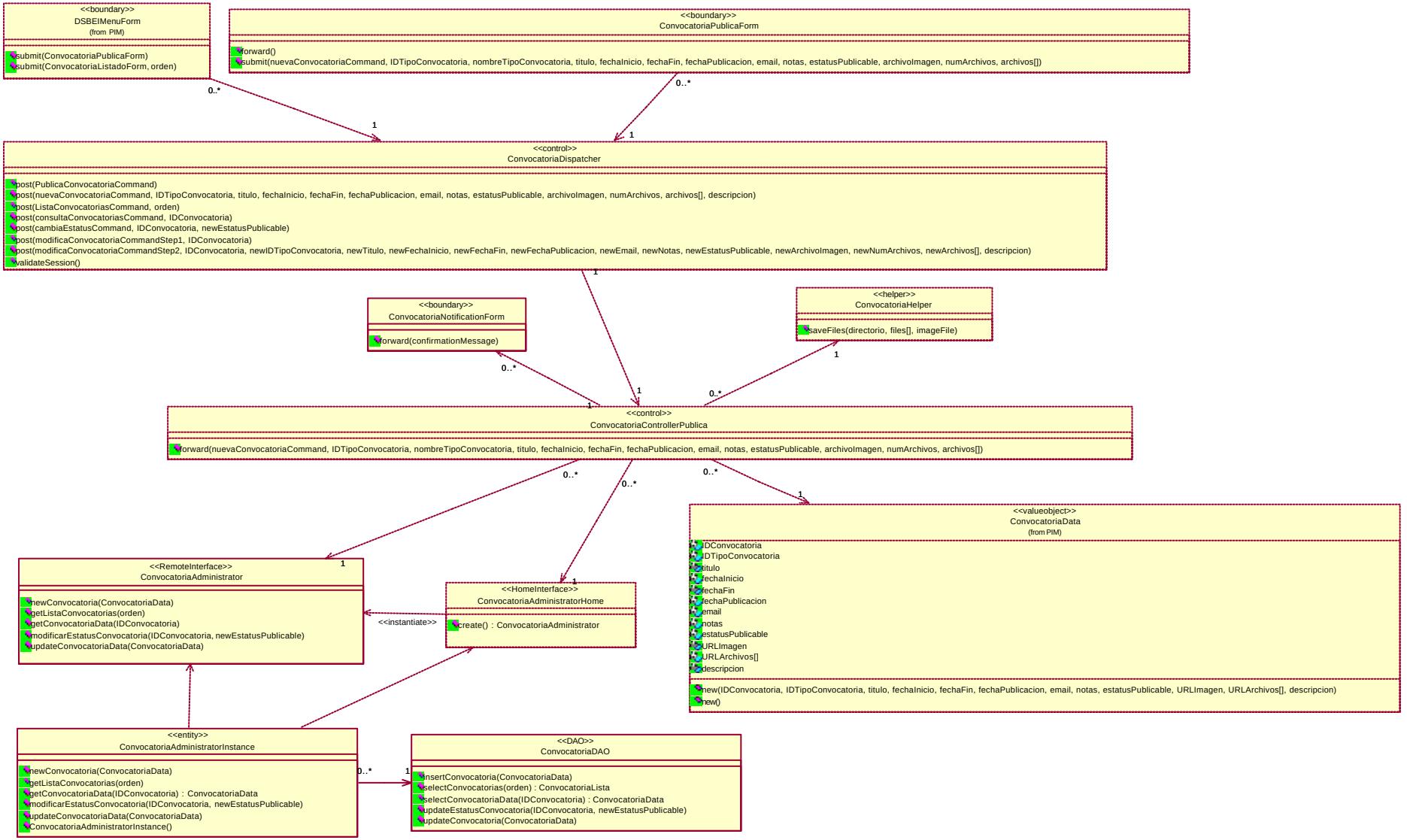
El JSP-boundary ConvocatoriaListadoForm pasa los datos al Servlet ConvocatoriaDispatcher, quien redirecciona los datos al Servlet ConvocatoriaControllerModificaStep1, éste último emplea el objeto Entity ConvocatoriaData para presentar los datos de la convocatoria en el JSP-boundary ConvocatoriaModificaForm.

Posteriormente el Servlet ConvocatoriaDispatcher envía los datos modificados de la convocatoria seleccionada al Servlet ConvocatoriaControllerModificaStep2, quien es cliente del EJB ConvocatoriaAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (ConvocatoriaAdministratorHome) y de InterfaceRemote (ConvocatoriaAdministrator), las cuales a través de la clase DAO ConvocatoriaDAO, quien tiene las operaciones SQL para realizar la actualización de los datos de la convocatoria contenidos en el objeto Entity ConvocatoriaData. El Servlet ConvocatoriaControllerModificaStep2 envía los datos adicionales de archivos contenidos en la convocatoria al Helper ConvocatoriaHelper. Posteriormente el Servlet ConvocatoriaControllerModificaStep2 envía una notificación de que se realizó el cambio a través del JSP ConvocatoriaNotificationForm.


<<boundary>>
ConvocatoriaListadoForm
(from PIM)
forward(ConvocatoriaListado)
submit(consultaConvocatoriaCommand, IDConvocatoria)
submit(cambiaEstatusCommand, IDConvocatoria, newEstatusPublicable)
submit(modificaConvocatoriaCommandStep1, IDConvocatoria)
<<control>>
ConvocatoriaDispatcher
(fromPIM)
post(PublicaConvocatoriaCommand)
post(nuevaConvocatoriaCommand, IDTipoConvocatoria, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[], descripcion)
post(ListaConvocatoriasCommand, orden)
post(consultaConvocatoriasCommand, IDConvocatoria)
post(cambiaEstatusCommand, IDConvocatoria, newEstatusPublicable)
post(modificaConvocatoriaCommandStep1, IDConvocatoria)
post(modificaConvocatoriaCommandStep2, IDConvocatoria, newIDTipoConvocatoria, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[], descripcion)
validateSession()
<<control>>
ConvocatoriaControllerModificaStep1
forward(IDConvocatoria)
<<control>>
ConvocatoriaControllerModificaStep2
forward(IDConvocatoria, newIDTipoConvocatoria, newNombreTipoConvocatoria, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[])
<<boundary>>
ConvocatoriaNotificationForm
(from PIM)
forward(confirmationMessage)
<<helper>>
ConvocatoriaHelper
(from PIM)
saveFiles(directorio, files[], imageFile)
<<RemoteInterface>>
ConvocatoriaAdministrator
(from PIM)
newConvocatoria(ConvocatoriaData)
getListaConvocatorias(orden)
getConvocatoriaData(IDConvocatoria)
modificarEstatusConvocatoria(IDConvocatoria, newEstatusPublicable)
updateConvocatoriaData(ConvocatoriaData)
<<HomeInterface>>
ConvocatoriaAdministratorHome
(from PIM)
<<instantiate>>
create() : ConvocatoriaAdministrator
<<control>>
ConvocatoriaControllerModificaStep1
forward(IDConvocatoria)
<<boundary>>
ConvocatoriaModificaForm
forward()
submit(modificaConvocatoriaCommandStep2, IDConvocatoria, newIDTipoConvocatoria, newNombreTipoConvocatoria, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[])
<<valueobject>>
ConvocatoriaData
(fromPIM)
IDConvocatoria
IDTipoConvocatoria
titulo
fechaInicio
fechaFin
fechaPublicacion
email
notas
estatusPublicable
URLImagen
URLArchivos[]
descripcion
new(IDConvocatoria, IDTipoConvocatoria, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, URLImagen, URLArchivos[], descripcion)
new()
<<entity>>
ConvocatoriaAdministratorInstance
(from PIM)
newConvocatoria(ConvocatoriaData)
getListaConvocatorias(orden)
getConvocatoriaData(IDConvocatoria) : ConvocatoriaData
modificarEstatusConvocatoria(IDConvocatoria, newEstatusPublicable)
updateConvocatoriaData(ConvocatoriaData)
ConvocatoriaAdministratorInstance()
<<DAO>>
ConvocatoriaDAO
(from PIM)
insertConvocatoria(ConvocatoriaData)
selectConvocatorias(orden) : ConvocatoriaLista
selectConvocatoriaData(IDConvocatoria) : ConvocatoriaData
updateEstatusConvocatoria(IDConvocatoria, newEstatusPublicable)
updateConvocatoria(ConvocatoriaData)

4.4.5 Diagrama de clases de publicación de convocatorias

El JSP-boundary DSBElMenuForm lanza la solicitud para publicar una convocatoria al Servlet ConvocatoriaDispatcher, el Servlet envía el JSP-boundary ConvocatoriaPublicaForm en donde se recuperan los datos de la convocatoria para posteriormente enviarlos al Servlet ConvocatoriaDispatcher, desde donde se redirecciona hacia otro Servlet, el ConvocatoriaControllerPublica, éste último, es cliente del EJB ConvocatoriaAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (ConvocatoriaAdministratorHome) y de InterfaceRemote (ConvocatoriaAdministrator), las cuales a través de la clase DAO ConvocatoriaDAO, quien tiene las operaciones SQL para realizar la inserción de los datos de la convocatoria contenidos en el objeto Entity ConvocatoriaData. El Servlet ConvocatoriaControllerPublica envía los datos adicionales de archivos contenidos en la convocatoria al Helper ConvocatoriaHelper. Posteriormente el Servlet ConvocatoriaControllerPublica envía una notificación de que se realizó la inserción a través del JSP ConvocatoriaNotificationForm.


<<boundary>>
DSBEIMenuForm
(from PIM)
submit(ConvocatoriaPublicaForm)
submit(ConvocatoriaListadoForm, orden)
<<boundary>>
ConvocatoriaPublicaForm
forward()
submit(nuevaConvocatoriaCommand, IDTipoConvocatoria, nombreTipoConvocatoria, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[])
<<control>>
ConvocatoriaDispatcher
post(PublicaConvocatoriaCommand)
post(nuevaConvocatoriaCommand, IDTipoConvocatoria, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[], descripcion)
post(ListaConvocatoriasCommand, orden)
post(consultaConvocatoriasCommand, IDConvocatoria)
post(cambiaEstatusCommand, IDConvocatoria, newEstatusPublicable)
post(modificaConvocatoriaCommandStep1, IDConvocatoria)
post(modificaConvocatoriaCommandStep2, IDConvocatoria, newIDTipoConvocatoria, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[], descripcion)
validateSession()
<<boundary>>
ConvocatoriaNotificationForm
forward(confirmationMessage)
<<helper>>
ConvocatoriaHelper
saveFiles(directorio, files[], imageFile)
<<control>>
ConvocatoriaControllerPublica
forward(nuevaConvocatoriaCommand, IDTipoConvocatoria, nombreTipoConvocatoria, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[])
<<valueobject>>
ConvocatoriaData
(from PIM)
IDConvocatoria
IDTipoConvocatoria
titulo
fechaInicio
fechaFin
fechaPublicacion
email
notas
estatusPublicable
URLImagen
URLArchivos[]
descripcion
new(IDConvocatoria, IDTipoConvocatoria, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, URLImagen, URLArchivos[], descripcion)
new()
<<RemoteInterface>>
ConvocatoriaAdministrator
newConvocatoria(ConvocatoriaData)
getListaConvocatorias(orden)
getConvocatoriaData(IDConvocatoria)
modificarEstatusConvocatoria(IDConvocatoria, newEstatusPublicable)
updateConvocatoriaData(ConvocatoriaData)
<<instantiate>>
<<HomeInterface>>
ConvocatoriaAdministratorHome
create() : ConvocatoriaAdministrator
<<entity>>
ConvocatoriaAdministratorInstance
newConvocatoria(ConvocatoriaData)
getListaConvocatorias(orden)
getConvocatoriaData(IDConvocatoria) : ConvocatoriaData
modificarEstatusConvocatoria(IDConvocatoria, newEstatusPublicable)
updateConvocatoriaData(ConvocatoriaData)
ConvocatoriaAdministratorInstance()
<<DAO>>
ConvocatoriaDAO
insertConvocatoria(ConvocatoriaData)
selectConvocatorias(orden) : ConvocatoriaLista
selectConvocatoriaData(IDConvocatoria) : ConvocatoriaData
updateEstatusConvocatoria(IDConvocatoria, newEstatusPublicable)
updateConvocatoria(ConvocatoriaData)
0..*
1

4.4.6 Diagrama de secuencia y de colaboración de cambia estatus a convocatorias publicadas

Empleando la lista de convocatorias, el actor DSBEI envía la solicitud de cambio de estatus mediante el mensaje submit(). La interfaz ConvocatoriaListadoForm reenvia los datos a la clase ConvocatoriaDispatcher. En el mensaje se envía el identificador de la convocatoria y su nuevo estatus. De la clase ConvocatoriaDispatcher se ejecuta el método validateSession donde al ser valida la sesión envía dichos datos a la clase ConvocatoriaControllerStatus. Ésta última, invoca al método create de la interfaz ConvocatoriaAdministratorHome y así obtener una referencia del tipo de la interfaz remota ConvocatoriaAdministrator. Una vez hecho esto, invoca al método modificarEstatusConvocatoria de la clase ConvocatoriaAdministrator que a su vez empleará el método modificarEstatusConvocatoria de la clase ConvocatoriaAdministratorInstance en donde se empleará el método updateEstatusConvocatoria de la clase ConvocatoriaDAO quien se ocupara de realizar el cambio del estatus, en la base de datos, a la convocatoria seleccionada.

Ya que se realizó la actualización, la clase ConvocatoriaControllerStatus envía el mensaje de confirmación a la interfaz Convocatoria NotificationForm, donde se presentará la notificación del cambio realizado al actor.

Diagrama de secuencia de cambia estatus a convocatorias publicadas

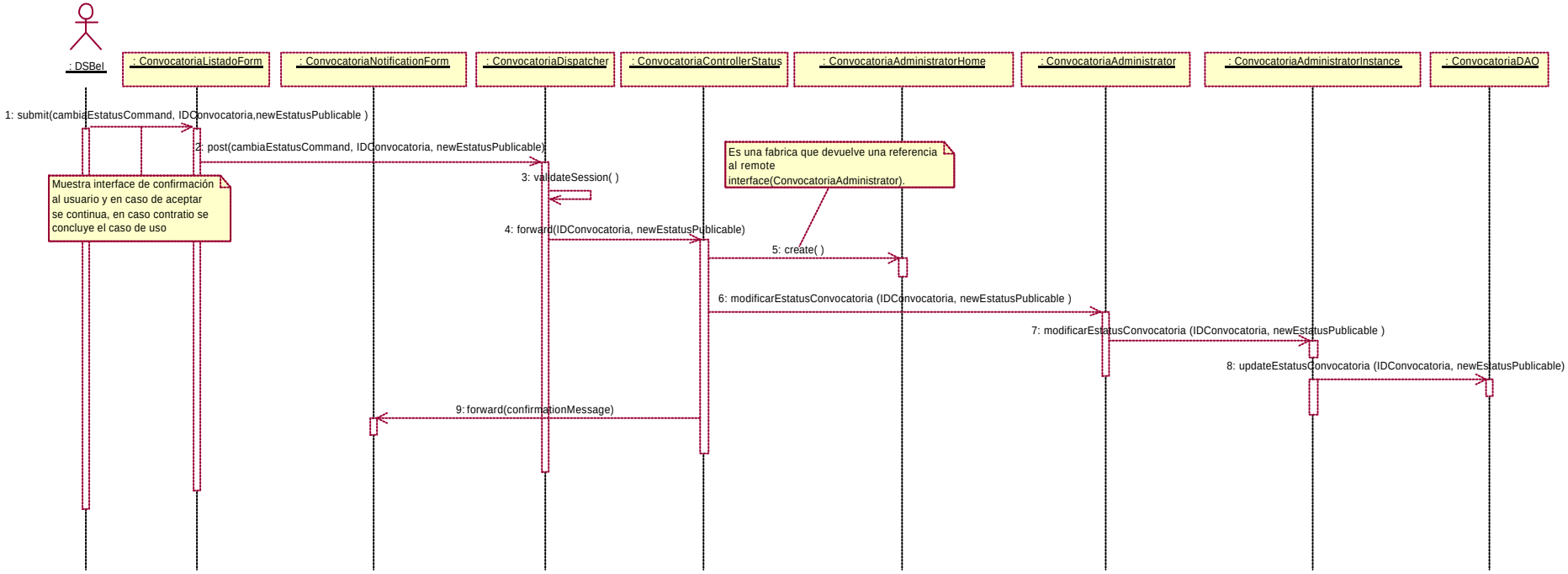
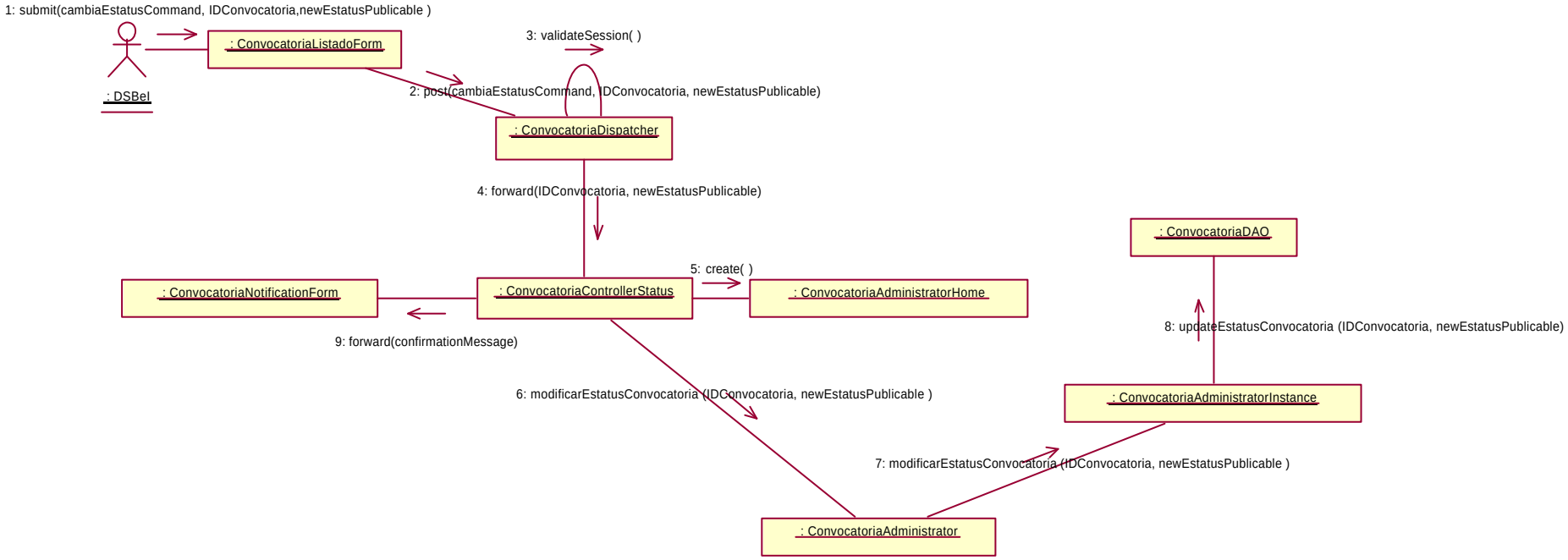


Diagrama de colaboración de cambia estatus a convocatorias publicadas



4.4.7 Diagrama de secuencia y de colaboración de consulta información completa de convocatorias

Empleando la lista de convocatorias, el actor DSBEI envía la solicitud de consulta y el identificador de la convocatoria a consultar mediante el mensaje submit(). La interfaz ConvocatoriaListadoForm reenvía los datos a la clase ConvocatoriaDispatcher. La clase ConvocatoriaDispatcher ejecuta el método validateSession donde al ser valida la sesión reenvía dichos datos a la clase ConvocatoriaControllerConsulta. Ésta última invoca al método create de la interfaz ConvocatoriaAdministratorHome y así obtener una referencia del tipo de la interfaz remota ConvocatoriaAdministrator. Una vez hecho esto, se invoca al método getConvocatoriaData de la clase ConvocatoriaAdministrator y ésta, posteriormente reenviara la solicitud de información de la convocatoria a la clase ConvocatoriaAdministratorInstance quien finalmente empleará el método selectConvocatoriaData de la clase ConvocatoriaDAO para colocar la información de la convocatoria en una instancia de la clase ConvocatoriaData donde viajara desde la clase ConvocatoriaControllerConsulta a la interfaz ConvocatoriaDetallesForm para ser presentada finalmente al actor.

Diagrama de secuencia de consulta información completa de convocatorias

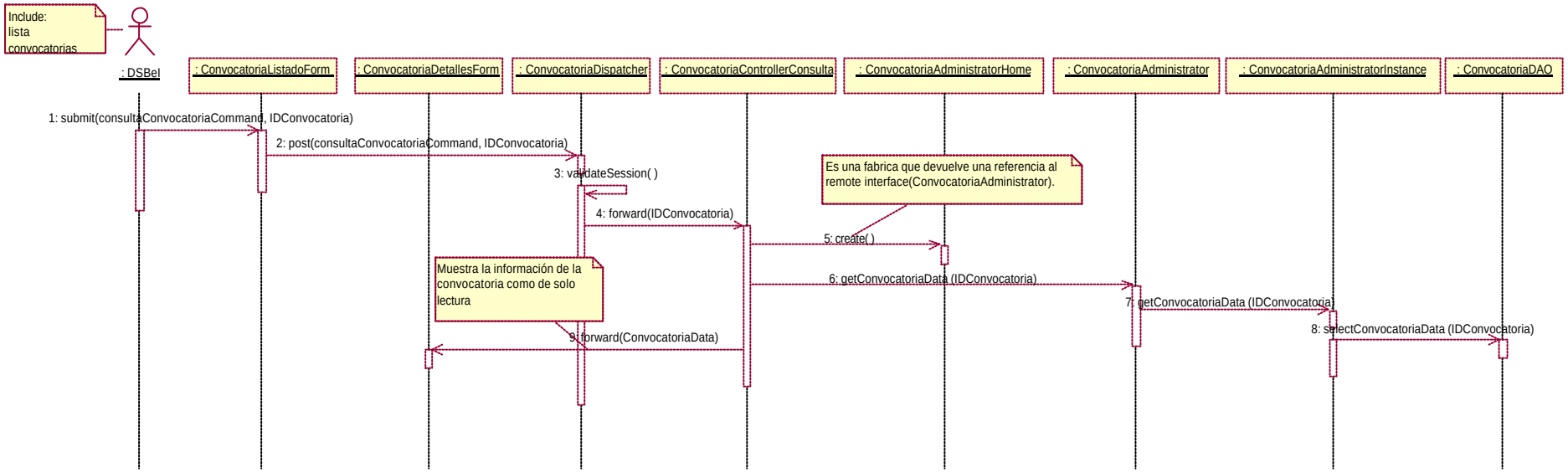
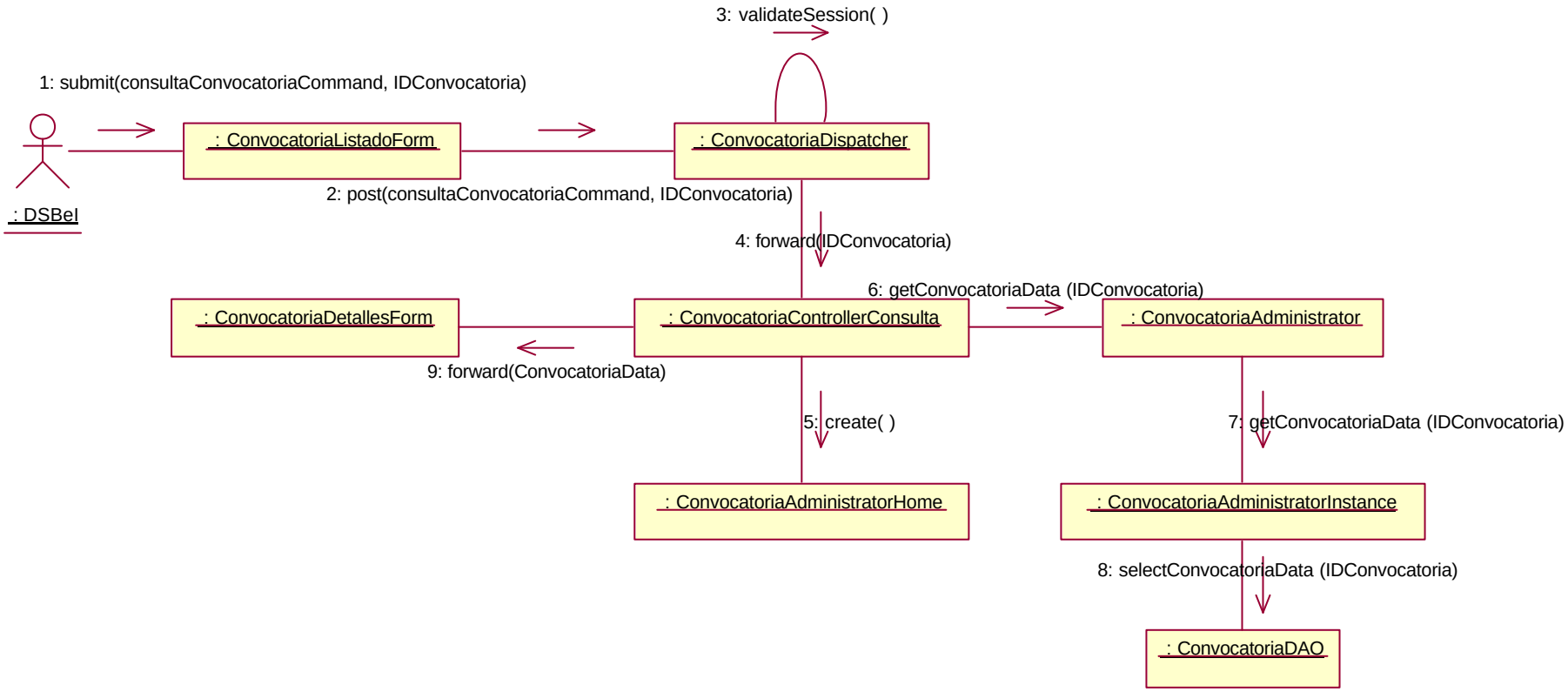


Diagrama de colaboración de consulta información completa de convocatorias



4.4.8 Diagrama de secuencia y de colaboración de lista convocatorias

El actor DSBEI envía la solicitud de listado de las convocatorias mediante el mensaje submit(). La interfaz DSBEIMenuForm reenvía los datos a la clase ConvocatoriaDispatcher. En el mensaje se indica el criterio por el cual se realiza la selección de las convocatorias para el listado. La clase ConvocatoriaDispatcher ejecuta el método validateSession donde al ser valida la sesión reenvía dichos datos a la clase ConvocatoriaControllerListado. Ésta última, invoca al método create de la interfaz ConvocatoriaAdministratorHome y así obtener una referencia del tipo de la interfaz remota ConvocatoriaAdministrator. Una vez hecho esto, invoca al método getListaConvocatorias de la clase ConvocatoriaAdministrator que a su vez empleará el método getListaConvocatorias de la clase ConvocatoriaAdministratorInstance en donde se empleará el método selectConvocatorias de la clase ConvocatoriaDAO quien genera un objeto de la clase ConvocatoriaLista en donde se contendrá el conjunto de objetos de la clase ConvocatoriaData. Finalmente el objeto de la clase ConvocatoriaLista que contiene la información obtenida de la consulta a la base de datos se envía a la interfaz ConvocatoriaListadoForm para presentar la información al actor.

Diagrama de secuencia y de colaboración de lista convocatorias

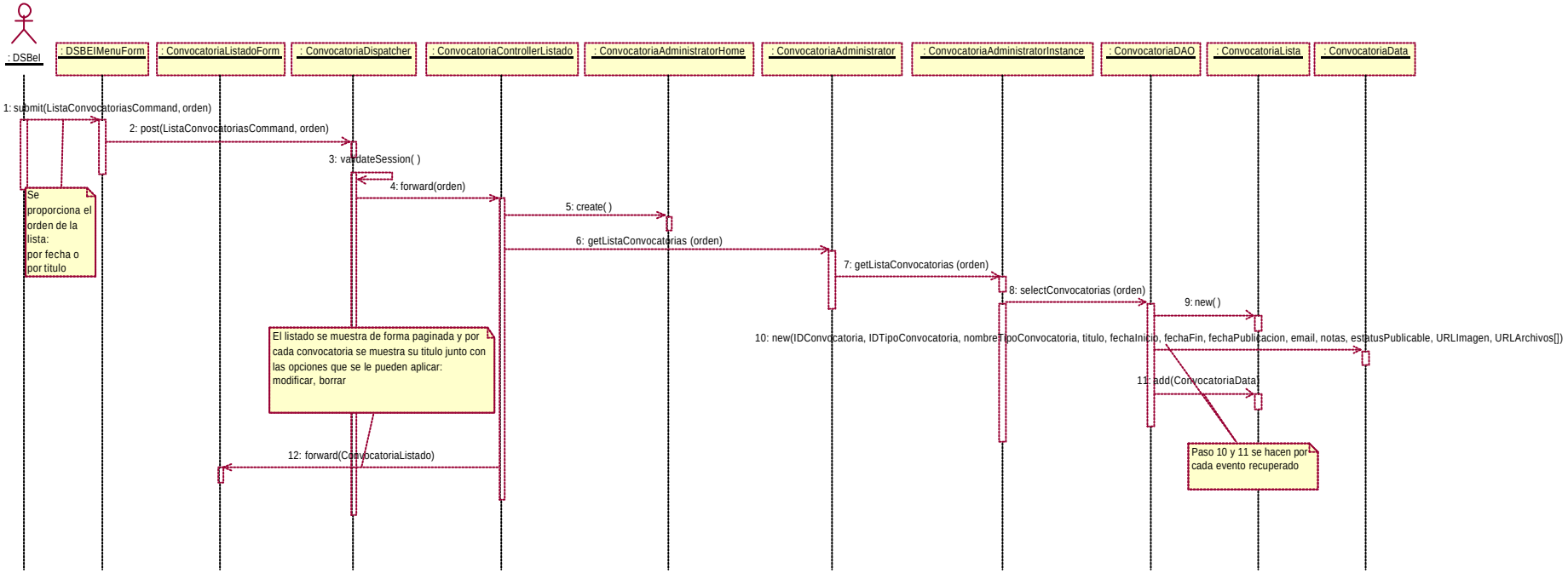
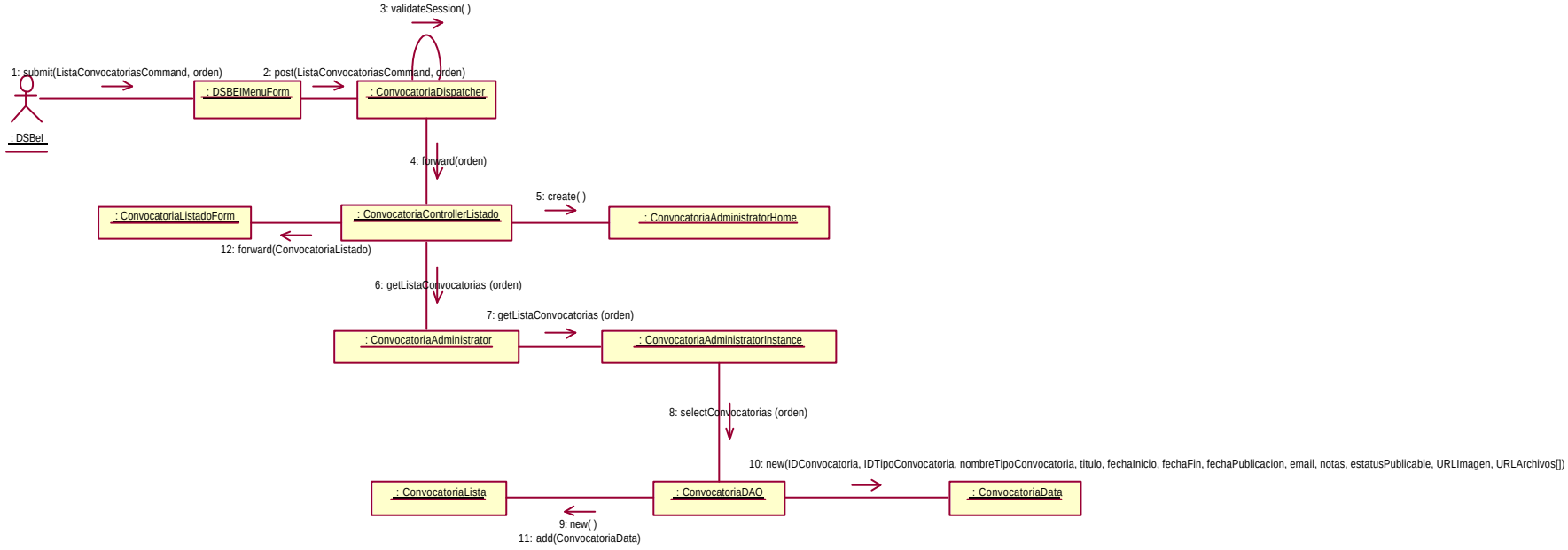


Diagrama de secuencia y de colaboración de lista convocatorias



4.4.9 Diagrama de secuencia y de colaboración de modifica datos de convocatorias publicadas

La secuencia para modificar datos de una convocatoria se divide en dos etapas. La primera muestra los datos de la convocatoria a modificar. En la segunda se envían los datos ya modificados de la convocatoria para que se reflejen en la base de datos.

En la primer etapa, el actor DSBEI emplea la lista de convocatorias y manda la solicitud de modificar la convocatoria mediante el mensaje submit(). La interfaz ConvocatoriaListadoForm reenvía los datos a la clase ConvocatoriaDispatcher. En el mensaje se indica el identificador de la convocatoria a modificar, así como también se especifica que el proceso se encuentra en su primera etapa.

De la clase ConvocatoriaDispatcher se ejecuta el método validateSession donde al ser valida la sesión reenvía dichos datos a la clase ConvocatoriaControllerModificaStep1. Ésta última, invoca al método create de la interfaz ConvocatoriaAdministratorHome y así obtener una referencia del tipo de la interfaz remota ConvocatoriaAdministrator y ésta, posteriormente reenviara la solicitud de información de la convocatoria a la clase ConvocatoriaAdministratorInstance quien finalmente empleará el método selectConvocatoriaData. La instancia con los datos de la convocatoria se pasa desde la clase ConvocatoriaControllerModificaStep1 a la interfaz ConvocatoriaModificaForm, en donde se presentan los datos al actor y poder realizar los cambios correspondientes.

En la segunda etapa, el actor DSBEI envía el mensaje submit() con el identificador de la convocatoria a modificar, el parámetro que indica que el proceso se encuentra en la segunda etapa y los datos ya modificados de la convocatoria. La interfaz ConvocatoriaModificaForm es empleada para realizar los cambios a los datos de la convocatoria presentada y reenvía los datos a la clase ConvocatoriaDispatcher.

De la clase ConvocatoriaDispatcher se ejecuta el método validateSession donde al ser valida la sesión reenvía dichos datos a la clase ConvocatoriaControllerModificaStep2. Ésta clase ejecuta el método saveFiles de la clase ConvocatoriaHelper para guardar en el sistema de archivos, los archivos que conforman a la convocatoria. Posteriormente se invoca al constructor de la clase ConvocatriaData donde se establecerán los atributos de la instancia con los datos modificados de la convocatoria seleccionada. La nueva instancia, se envía como parámetro en el método updateConvocatoriaData, el cual es ejecutado desde la Interfaz remota ConvocatoriaAdministrator y que posteriormente invocara al mismo método de la clase ConvocatoriaAdministratorInstance. Éste último método emplea el método updateConvocatoria de la clase ConvocatoriaDAO para almacenar la información del objeto de la clase

ConvocatoriaData en la Base de Datos. Una vez que se guardan los cambios de la convocatoria, se envía un mensaje de confirmación de la actualización mediante la interfaz ConvocatoriaNotificationForm

Diagrama de secuencia de modifica datos de convocatorias publicadas

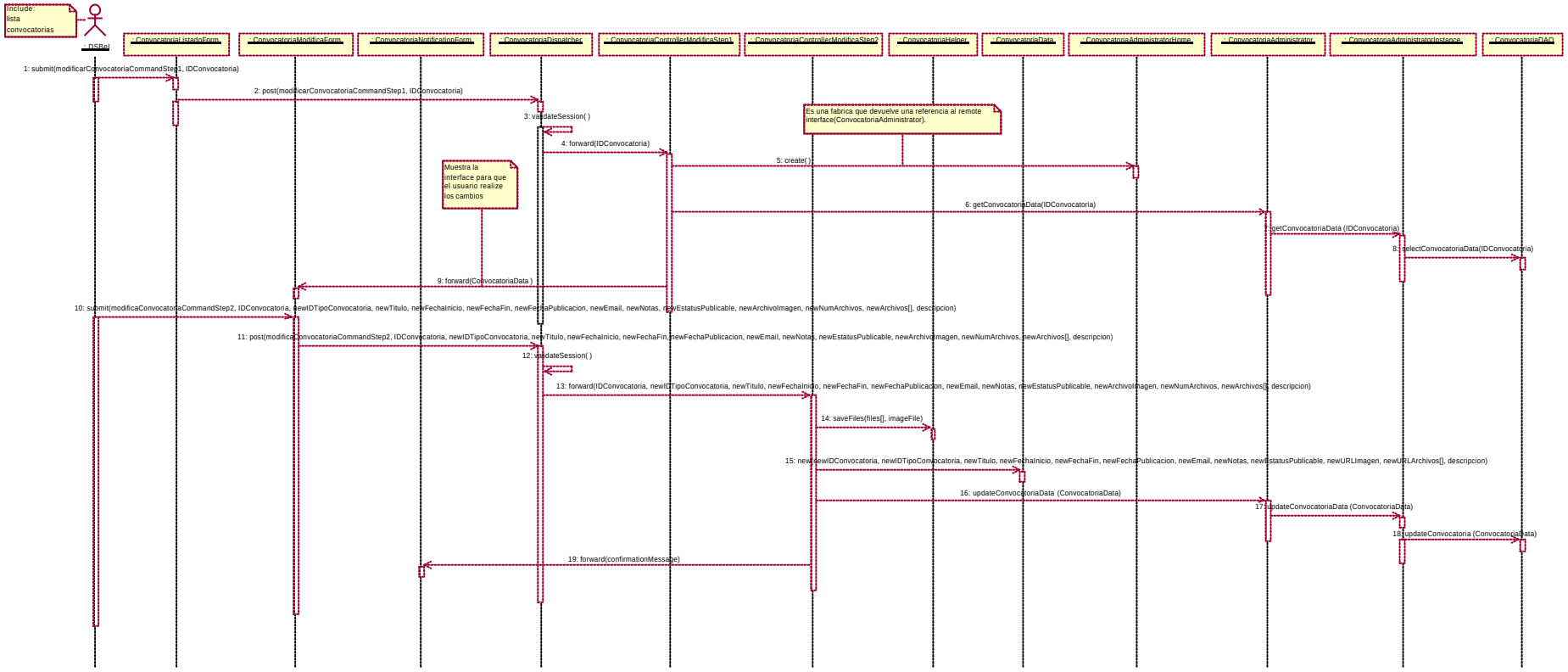
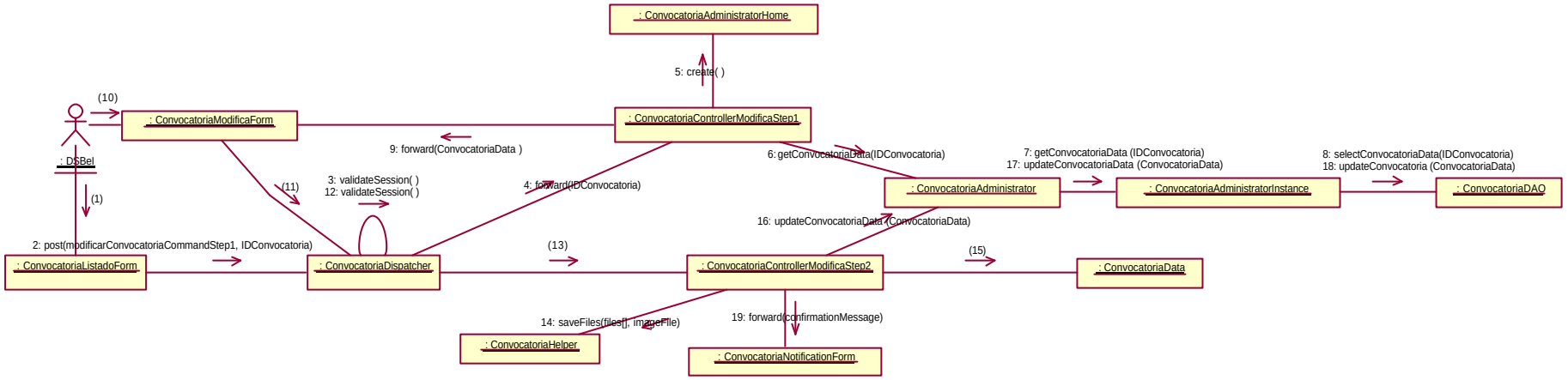


Diagrama de colaboración de modifica datos de convocatorias publicadas



1: submit(modificarConvocatoriaCommandStep1, IDConvocatoria)

10: submit(modificaConvocatoriaCommandStep2, IDConvocatoria, newIDTipoConvocatoria, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[], descripcion)

11: post(modificaConvocatoriaCommandStep2, IDConvocatoria, newIDTipoConvocatoria, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[], descripcion)

13: forward(IDConvocatoria, newIDTipoConvocatoria, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[], descripcion)

15: new(newIDConvocatoria, newIDTipoConvocatoria, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newURLImagen, newURLArchivos[], descripcion)

4.4.10 Diagrama de secuencia de publica convocatorias

EL actor DSBel manda la solicitud de publicar convocatoria mediante el mensaje submit(). La interfaz DSBEIMenuForm reenvía la información a la clase ConvocatoriaDispatcher. De la clase ConvocatoriaDispatcher se ejecuta el método validateSession donde al ser valida la sesión presenta la interfaz ConvocatoriaPublicaForm para que el usuario introduzca los datos de la nueva convocatoria.

El actor DSBei envía los datos de la nueva convocatoria en el mensaje submit desde la interface ConvocatoriaPublicaForm a la clase ConvocatoriaDispatcher. La clase ConvocatoriaDispatcher ejecuta el método validateSession para posteriormente envía dichos datos a la clase ConvocatoriaControllerPublica siempre y cuando la sesión sea válida. La clase ConvocatoriaControllerPublica invoca al método create de la interfaz ConvocatoriaAdministratorHome y así obtener una referencia del tipo de la interfaz remota ConvocatoriaAdministrator.

Posteriormente, la clase ConvocatoriaControllerPublica invoca al constructor de la clase ConvocatriaData donde se establecerán los atributos de la nueva instancia con los datos correspondientes de la nueva convocatoria. A continuación se invoca al método newConvocatoria de la Interfaz remota ConvocatoriaAdministrator desde donde se invocara al mismo método de la clase ConvocatoriaAdministratorInstance. El constructor de la clase ConvocatoriaAdministratorInstance emplea el método insertConvocatoria de la clase ConvocatoriaDAO para almacenar la información del objeto de la clase ConvocatoriaData en la Base de Datos. Una vez que la información es almacenada, se ejecuta el método saveFiles de la clase ConvocatoriaHelper para guardar en el sistema de archivos, los archivos que conforman a la convocatoria.

Finalmente la clase ConvocatoriaControllerPublica envía el mensaje de confirmación a la interfaz ConvocatoriaNotificationForm, donde se presentara la notificación de que la alta de la convocatoria ocurrió de forma correcta.

Diagrama de secuencia de publica convocatorias

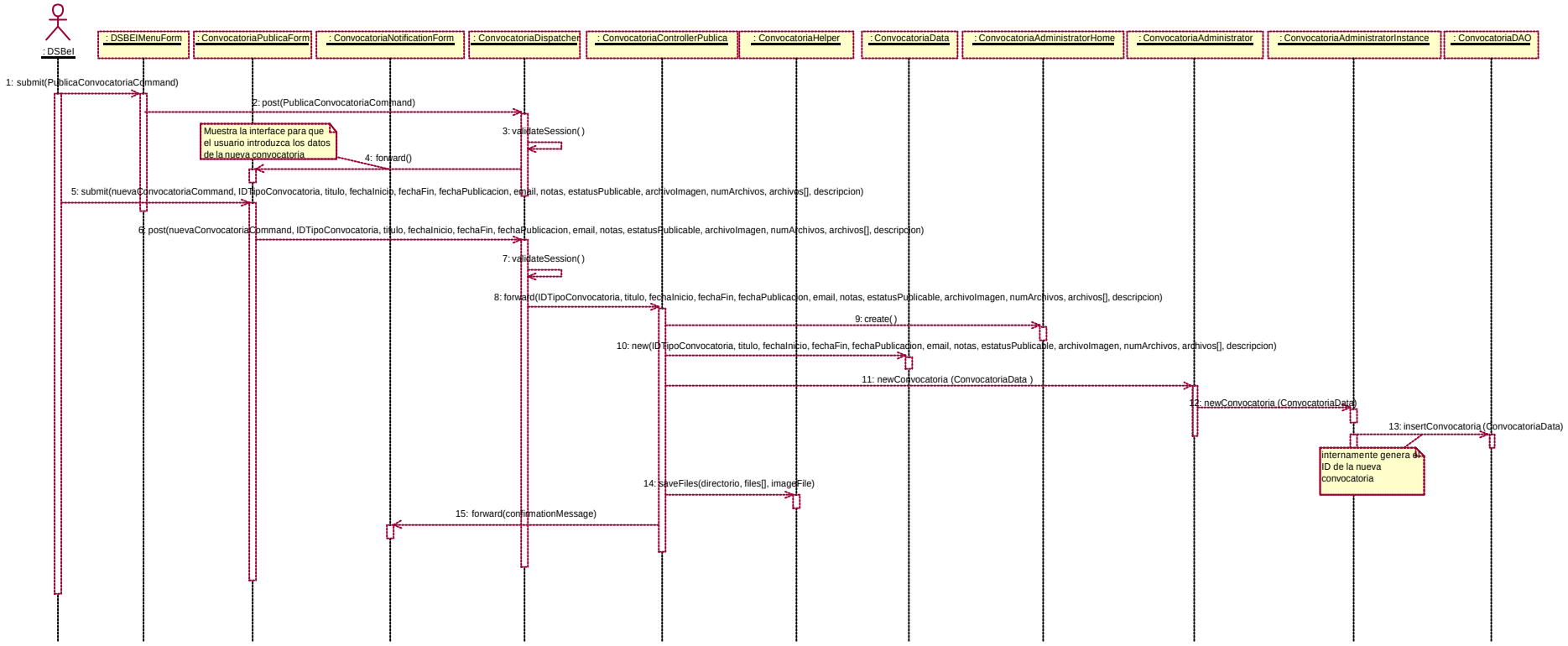
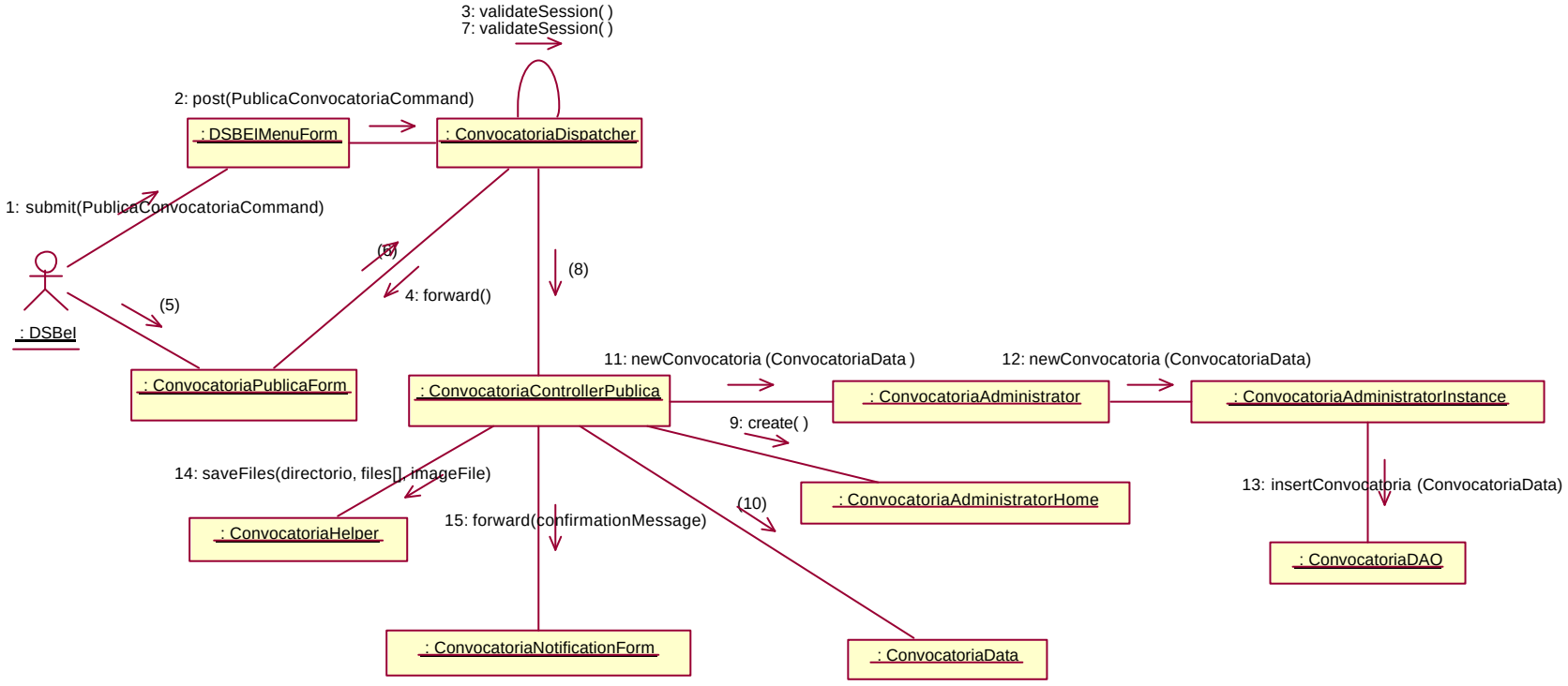


Diagrama de colaboración de publica convocatorias



5: submit(nuevaConvocatoriaCommand, IDTipoConvocatoria, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[], descripcion)

6: post(nuevaConvocatoriaCommand, IDTipoConvocatoria, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[], descripcion)

8: forward(IDTipoConvocatoria, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[], descripcion)

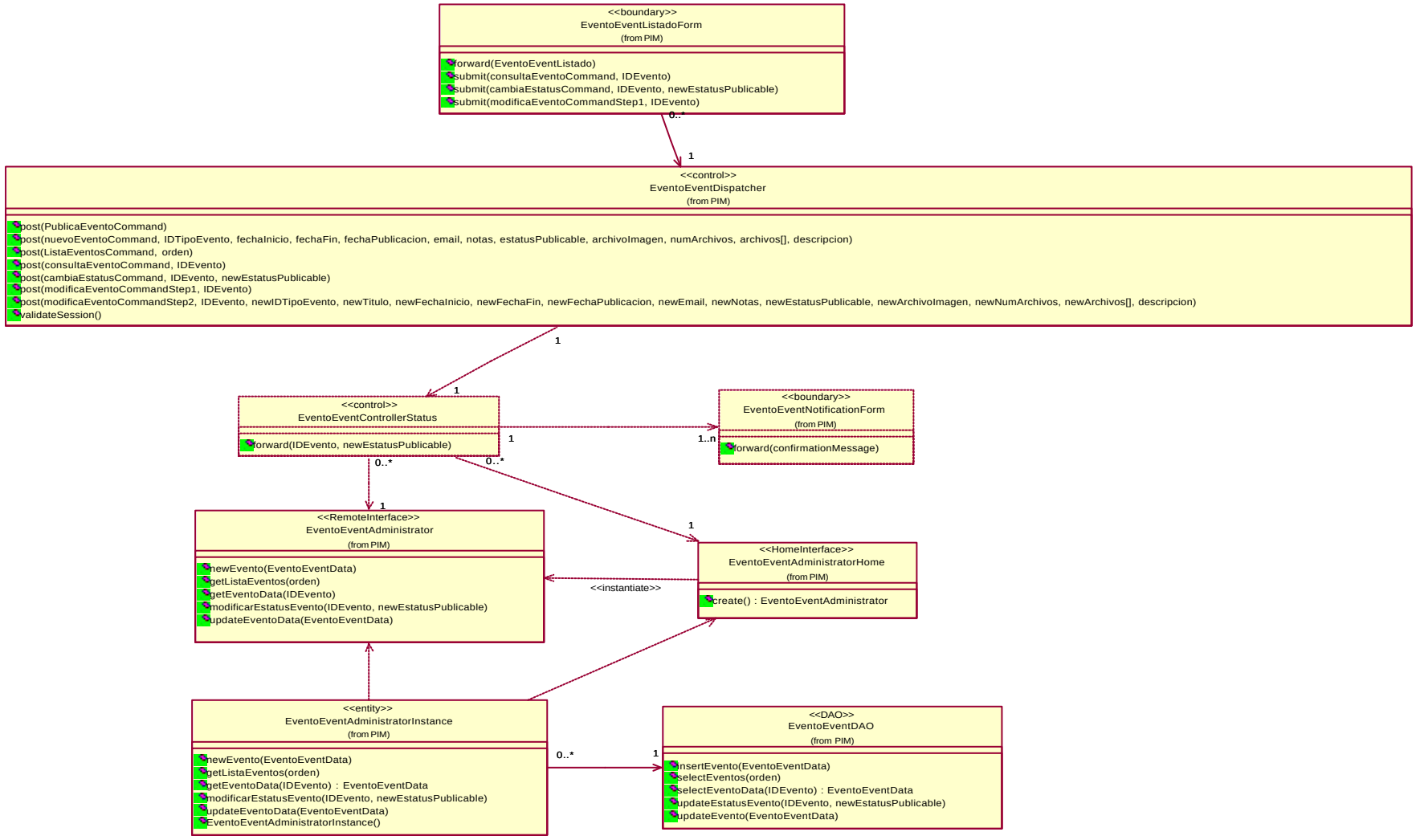
10: new(IDTipoConvocatoria, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[], descripcion)

4.5 Eventos

Describe los procesos que tiene el sistema para controlar y manipular la información de loa eventos.

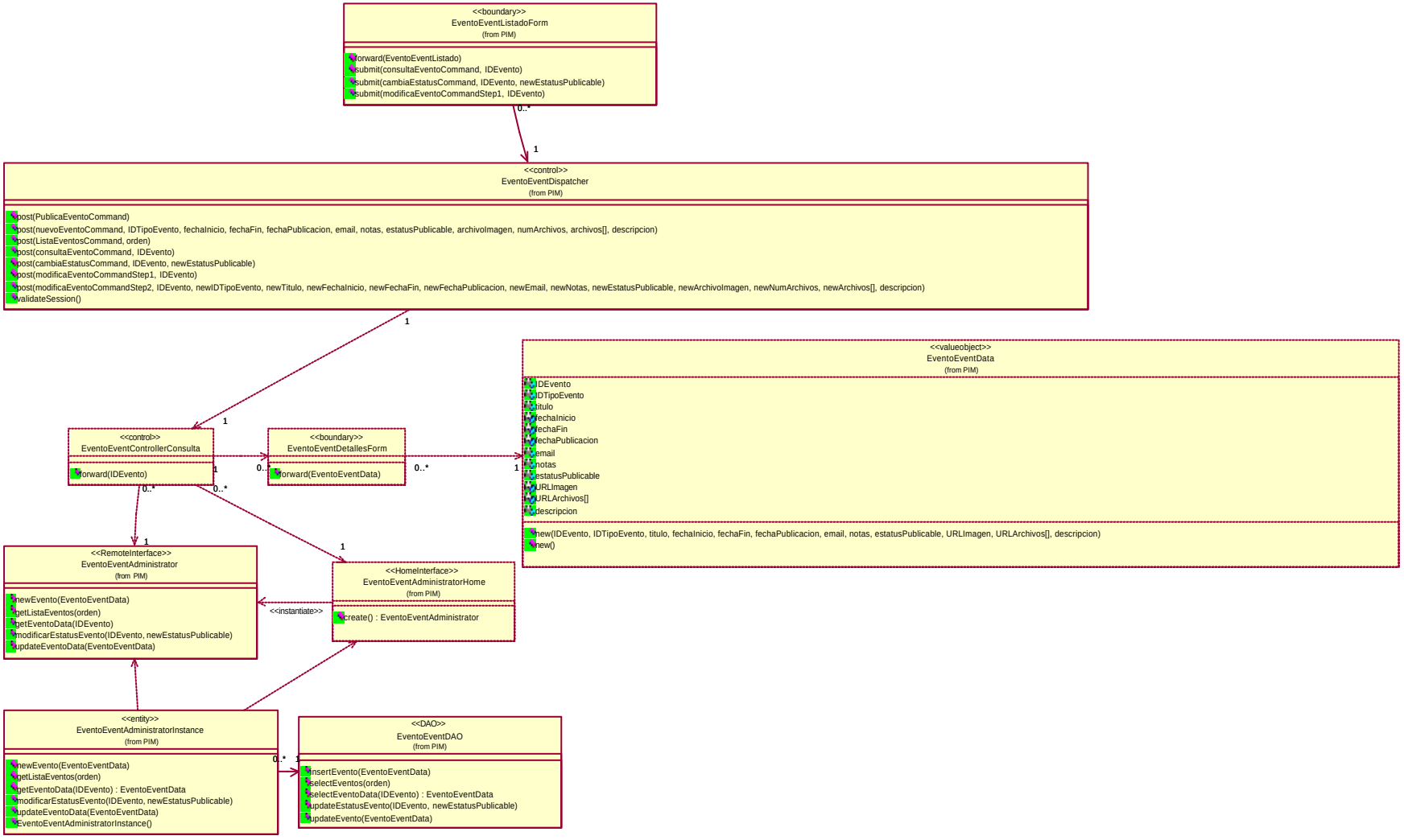
4.5.1 Diagrama de clases de cambia estatus a eventos publicados.

EL JSP-boundary EventoEventListadoForm pasa los datos al Servlet EventoEventDispatcher, éste redirecciona hacia otro Servlet, el EventoEventControllerStatus Éste último, es cliente del EJB EventoEventAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (EventoEventAdministratorHome) y de InterfaceRemote (EventoEventAdministrator), las cuales a través de la clase DAO EventoEventDAO, quien tiene las operaciones SQL, se realiza el cambio del estatus en el evento. Posteriormente el Servlet EventoEventControllerStatus envía una notificación de que se realizó el cambio a través del JSP EventoEventNotificationForm.


<<boundary>>
EventoEventListadoForm
(from PIM)
forward(EventoEventListado)
submit(consultaEventoCommand, IDEvento)
submit(cambiaEstatusCommand, IDEvento, newEstatusPublicable)
submit(modificaEventoCommandStep1, IDEvento)
0..*
1
<<control>>
EventoEventDispatcher
(from PIM)
post(PublicaEventoCommand)
post(nuevoEventoCommand, IDTipoEvento, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[], descripcion)
post(ListaEventosCommand, orden)
post(consultaEventoCommand, IDEvento)
post(cambiaEstatusCommand, IDEvento, newEstatusPublicable)
post(modificaEventoCommandStep1, IDEvento)
post(modificaEventoCommandStep2, IDEvento, newIDTipoEvento, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[], descripcion)
validateSession()
1
1
<<control>>
EventoEventControllerStatus
forward(IDEvento, newEstatusPublicable)
1
<<boundary>>
EventoEventNotificationForm
(from PIM)
forward(confirmationMessage)
1..n
0..*
0..*
1
<<RemoteInterface>>
EventoEventAdministrator
(from PIM)
newEvento(EventoEventData)
getListaEventos(orden)
getEventoData(IDEvento)
modificarEstatusEvento(IDEvento, newEstatusPublicable)
updateEventoData(EventoEventData)
1
<<HomeInterface>>
EventoEventAdministratorHome
(from PIM)
create() : EventoEventAdministrator
<<instantiate>>
<<entity>>
EventoEventAdministratorInstance
(from PIM)
newEvento(EventoEventData)
getListaEventos(orden)
getEventoData(IDEvento) : EventoEventData
modificarEstatusEvento(IDEvento, newEstatusPublicable)
updateEventoData(EventoEventData)
EventoEventAdministratorInstance()
0..*
1
<<DAO>>
EventoEventDAO
(from PIM)
insertEvento(EventoEventData)
selectEventos(orden)
selectEventoData(IDEvento) : EventoEventData
updateEstatusEvento(IDEvento, newEstatusPublicable)
updateEvento(EventoEventData)

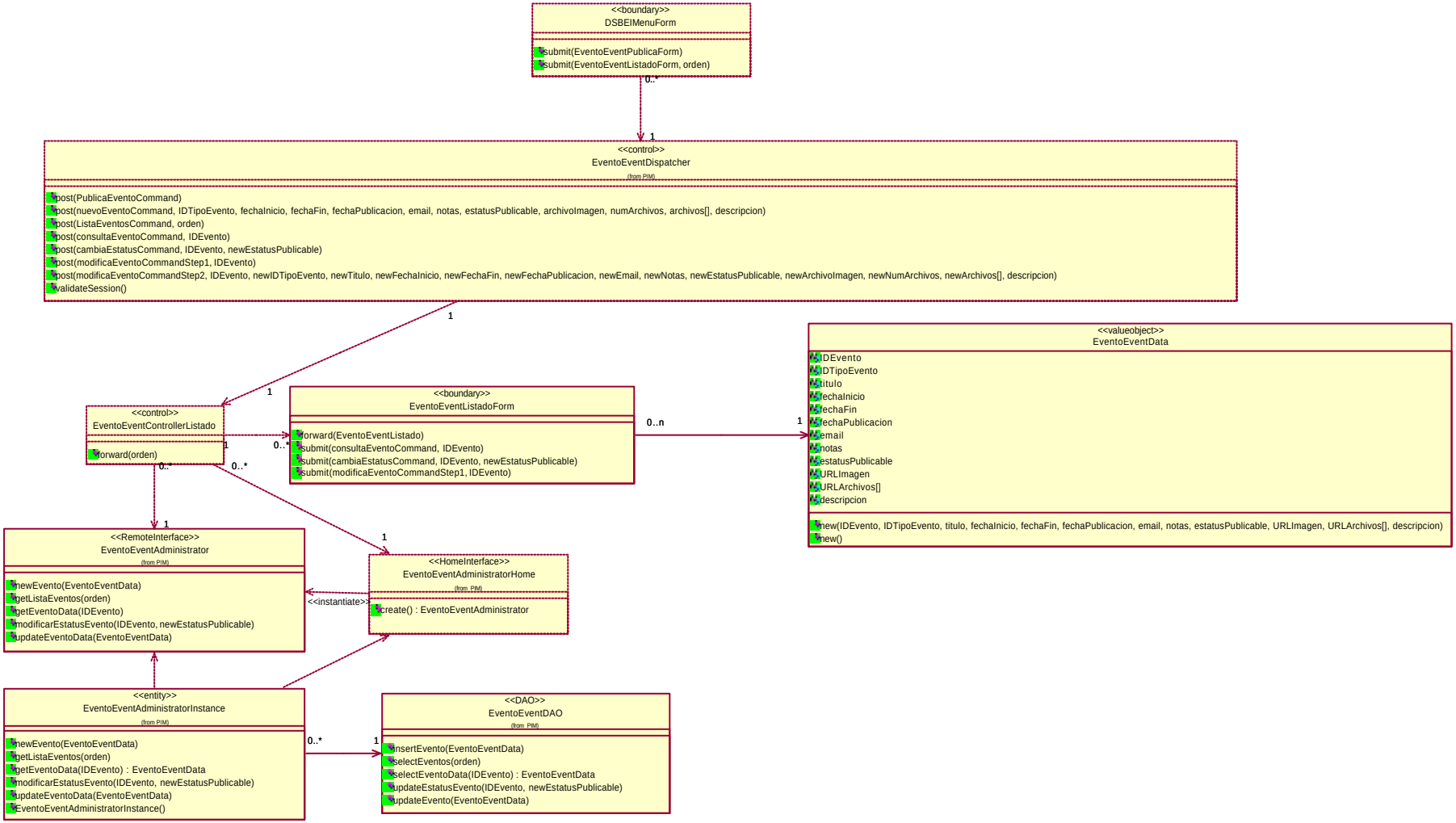
4.5.2 Diagrama de clases de consulta información completa del evento

EL JSP EventoEventListadoForm pasa los datos al Servlet EventoEventDispatcher, quien redirecciona los datos al Servlet EventoEventControllerConsulta. Éste último, es cliente del EJB EventoEventAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (EventoEventAdministratorHome) y de InterfaceRemote (EventoEventAdministrator), las cuales a través de la clase DAO EventoEventDAO, quien tiene las operaciones SQL, obtiene al objeto Entity EventoEventData, con éste, el JSP EventoEventControllerConsulta muestra los datos a través del JSP EventoEventDetalleForm.


<<boundary>>
EventoEventListadoForm
(from PIM)
forward(EventoEventListado)
submit(consultaEventoCommand, IDEvento)
submit(cambiaEstatusCommand, IDEvento, newEstatusPublicable)
submit(modificaEventoCommandStep1, IDEvento)
0..*
1
<<control>>
EventoEventDispatcher
(from PIM)
post(PublicaEventoCommand)
post(nuevoEventoCommand, IDTipoEvento, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[], descripcion)
post(ListaEventosCommand, orden)
post(consultaEventoCommand, IDEvento)
post(cambiaEstatusCommand, IDEvento, newEstatusPublicable)
post(modificaEventoCommandStep1, IDEvento)
post(modificaEventoCommandStep2, IDEvento, newIDTipoEvento, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[], descripcion)
validateSession()
1
1
<<control>>
EventoEventControllerConsulta
forward(IDEvento)
1
0..*
0..*
<<boundary>>
EventoEventDetallesForm
forward(EventoEventData)
0..*
1
<<valueobject>>
EventoEventData
(from PIM)
IDEvento
IDTipoEvento
titulo
fechaInicio
fechaFin
fechaPublicacion
email
notas
estatusPublicable
URLImagen
URLArchivos[]
descripcion
new(IDEvento, IDTipoEvento, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, URLImagen, URLArchivos[], descripcion)
new()
1
1
<<RemoteInterface>>
EventoEventAdministrator
(from PIM)
newEvento(EventoEventData)
getListaEventos(orden)
getEventoData(IDEvento)
modificarEstatusEvento(IDEvento, newEstatusPublicable)
updateEventoData(EventoEventData)
<<instantiate>>
<<HomeInterface>>
EventoEventAdministratorHome
(from PIM)
create() : EventoEventAdministrator
<<entity>>
EventoEventAdministratorInstance
(from PIM)
newEvento(EventoEventData)
getListaEventos(orden)
getEventoData(IDEvento) : EventoEventData
modificarEstatusEvento(IDEvento, newEstatusPublicable)
updateEventoData(EventoEventData)
EventoEventAdministratorInstance()
0..*
1
<<DAO>>
EventoEventDAO
(from PIM)
insertEvento(EventoEventData)
selectEventos(orden)
selectEventoData(IDEvento) : EventoEventData
updateEstatusEvento(IDEvento, newEstatusPublicable)
updateEvento(EventoEventData)

4.5.3 Diagrama de clases de lista eventos

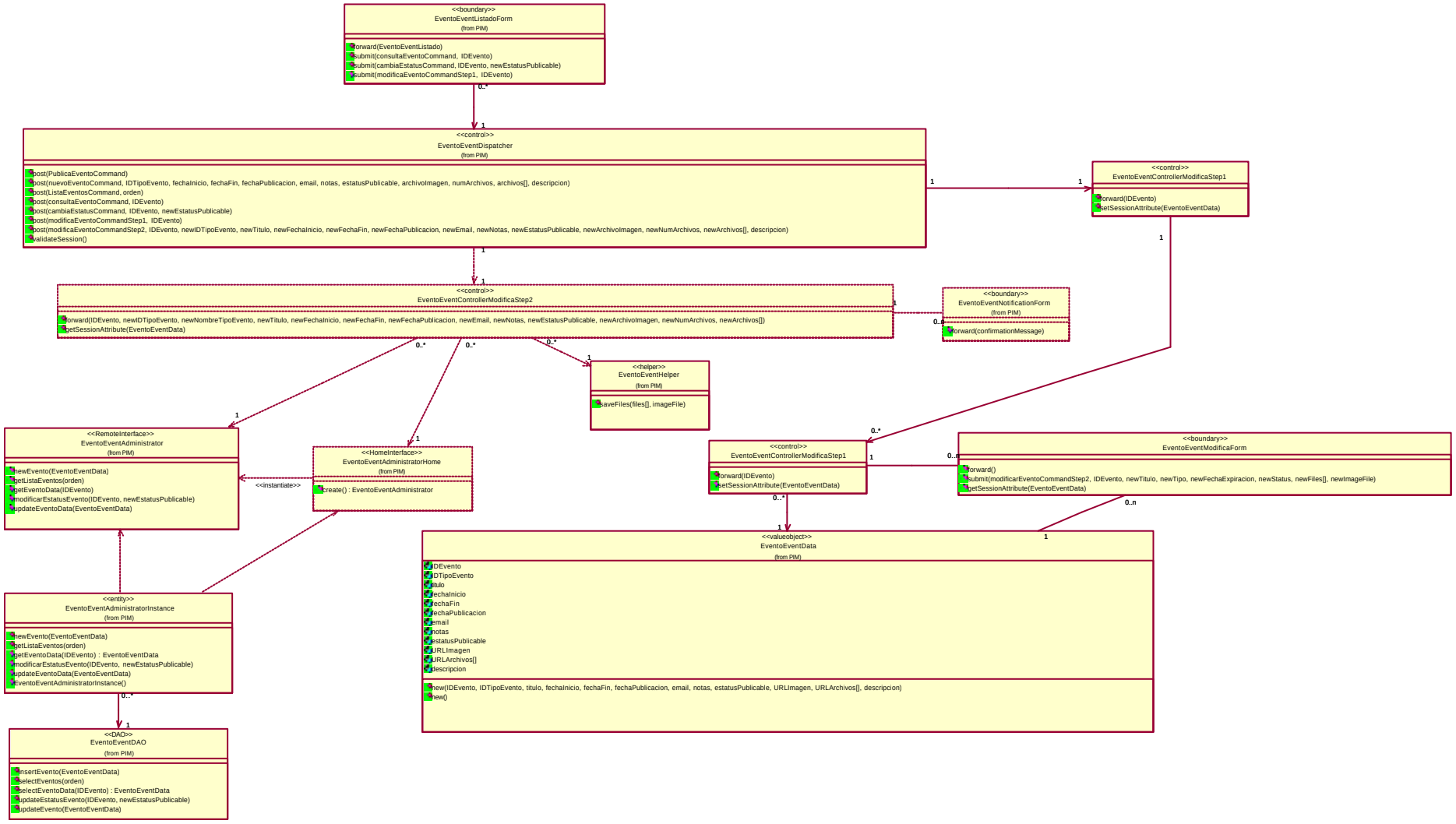
EL JSP-boundary DSBElMenuForm pasa los datos al Servlet EventoEventDispatcher, éste redirecciona hacia otro Servlet, el EventoEventControllerListado. Éste último, es cliente del EJB EventoEventAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (EventoEventAdministratorHome) y de InterfaceRemote (EventoEventAdministrator), las cuales a través de la clase DAO EventoEventDAO, quien tiene las operaciones SQL, obtiene al objeto Collection EventoEventLista que contiene a la clase Entity EventoEventData, así, el Servlet EventoEvenntControllerListado pasa el objeto Collection EventoEventLista al JSP EventoEventListadoForm, quien muestra los eventos como parte del resultado final en la interfaz


<<boundary>>
DSBEIMenuForm
submit(EventoEventPublicaForm)
submit(EventoEventListadoForm, orden)
0..*
1
<<control>>
EventoEventDispatcher
(from PIM)
post(PublicaEventoCommand)
post(nuevoEventoCommand, IDTipoEvento, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[], descripcion)
post(ListaEventosCommand, orden)
post(consultaEventoCommand, IDEvento)
post(cambiaEstatusCommand, IDEvento, newEstatusPublicable)
post(modificaEventoCommandStep1, IDEvento)
post(modificaEventoCommandStep2, IDEvento, newIDTipoEvento, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[], descripcion)
validateSession()
1
1
<<control>>
EventoEventControllerListado
forward(orden)
1
0..*
0..*
<<boundary>>
EventoEventListadoForm
forward(EventoEventListado)
submit(consultaEventoCommand, IDEvento)
submit(cambiaEstatusCommand, IDEvento, newEstatusPublicable)
submit(modificaEventoCommandStep1, IDEvento)
0..n
1
<<valueobject>>
EventoEventData
IDEvento
IDTipoEvento
titulo
fechaInicio
fechaFin
fechaPublicacion
email
notas
estatusPublicable
URLImagen
URLArchivos[]
descripcion
new(IDEvento, IDTipoEvento, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, URLImagen, URLArchivos[], descripcion)
new()
1
<<RemoteInterface>>
EventoEventAdministrator
(from PIM)
newEvento(EventoEventData)
getListaEventos(orden)
getEventoData(IDEvento)
modificarEstatusEvento(IDEvento, newEstatusPublicable)
updateEventoData(EventoEventData)
1
<<HomeInterface>>
EventoEventAdministratorHome
(from PIM)
create() : EventoEventAdministrator
<<instantiate>>
<<entity>>
EventoEventAdministratorInstance
(from PIM)
newEvento(EventoEventData)
getListaEventos(orden)
getEventoData(IDEvento) : EventoEventData
modificarEstatusEvento(IDEvento, newEstatusPublicable)
updateEventoData(EventoEventData)
EventoEventAdministratorInstance()
0..*
1
<<DAO>>
EventoEventDAO
(from PIM)
insertEvento(EventoEventData)
selectEventos(orden)
selectEventoData(IDEvento) : EventoEventData
updateEstatusEvento(IDEvento, newEstatusPublicable)
updateEvento(EventoEventData)

4.5.4 Diagrama de clases de modifica datos a eventos publicados

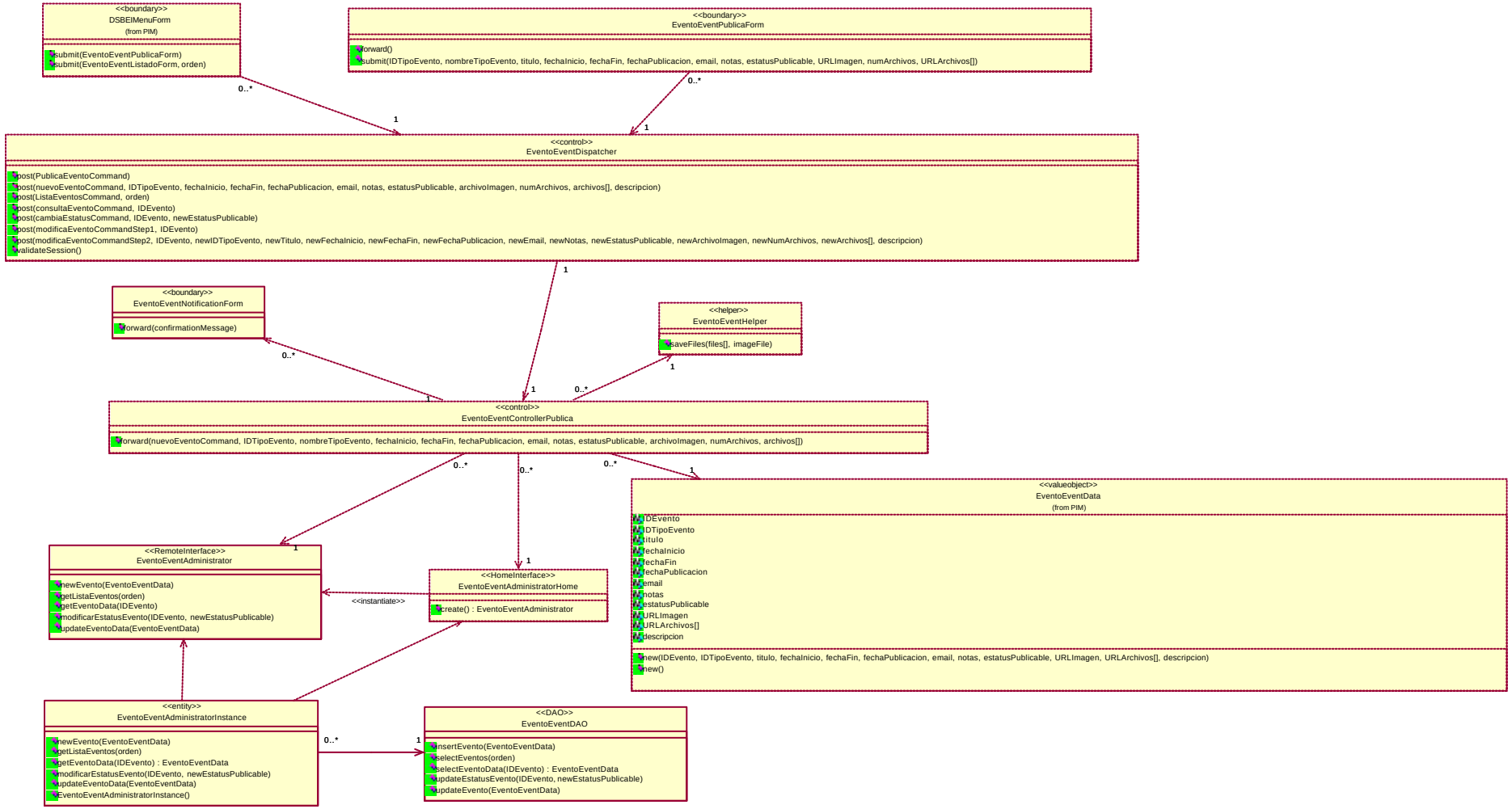
EL JSP-boundary EventoEventListadoForm pasa los datos al Servlet EventoEventDispatcher, el cual redirecciona los datos al Servlet EventoEventControllerModificaStep1, éste último emplea el objeto Entity EventoEventData para presentar los datos del evento en el JSP-boundary EventoEventModificaForm.

Posteriormente el Servlet EventoEventDispatcher envía los datos modificados del evento seleccionado al Servlet EventoEventControllerModificaStep2, quien es cliente del EJB EventoEventAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (EventoEventAdministratorHome) y de InterfaceRemote (EventoEventAdministrator), las cuales a través de la clase DAO EventoEventDAO, quien tiene las operaciones SQL para realizar la actualización de los datos del evento contenidos en el objeto Entity EventoEventData. El Servlet EventoEventControllerModificaStep2 envía los datos adicionales de archivos contenidos en el evento al Helper EventoEventHelper. Posteriormente el Servlet EventoEventControllerModificaStep2 envía una notificación de que se realizó el cambio a través del JSP EventoEventNotificationForm.


<<boundary>>
EventoEventListadoForm
(from PIM)
forward(EventoEventListado)
submit(consultaEventoCommand, IDEvento)
submit(cambiaEstatusCommand, IDEvento, newEstatusPublicable)
submit(modificaEventoCommandStep1, IDEvento)
<<control>>
EventoEventDispatcher
(from PIM)
post(PublicaEventoCommand)
post(nuevoEventoCommand, IDTipoEvento, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[], descripcion)
post(ListaEventosCommand, orden)
post(consultaEventoCommand, IDEvento)
post(cambiaEstatusCommand, IDEvento, newEstatusPublicable)
post(modificaEventoCommandStep1, IDEvento)
post(modificaEventoCommandStep2, IDEvento, newIDTipoEvento, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[], descripcion)
validateSession()
<<control>>
EventoEventControllerModificaStep1
forward(IDEvento)
getSessionAttribute(EventoEventData)
<<control>>
EventoEventControllerModificaStep2
forward(IDEvento, newIDTipoEvento, newNombreTipoEvento, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[])
getSessionAttribute(EventoEventData)
<<boundary>>
EventoEventNotificationForm
(from PIM)
forward(confirmationMessage)
<<helper>>
EventoEventHelper
(from PIM)
saveFiles(files[], imageFile)
<<RemoteInterface>>
EventoEventAdministrator
(from PIM)
newEvento(EventoEventData)
getListaEventos(orden)
getEventoData(IDEvento)
modificarEstatusEvento(IDEvento, newEstatusPublicable)
updateEventoData(EventoEventData)
<<HomeInterface>>
EventoEventAdministratorHome
(from PIM)
create() : EventoEventAdministrator
<<instantiate>>
<<control>>
EventoEventControllerModificaStep1
forward(IDEvento)
getSessionAttribute(EventoEventData)
<<boundary>>
EventoEventModificaForm
forward()
submit(modificarEventoCommandStep2, IDEvento, newTitulo, newTipo, newFechaExpiracion, newStatus, newFiles[], newImageFile)
getSessionAttribute(EventoEventData)
<<valueobject>>
EventoEventData
(from PIM)
IDEvento
IDTipoEvento
titulo
fechaInicio
fechaFin
fechaPublicacion
email
notas
estatusPublicable
URLImagen
URLArchivos[]
descripcion
new(IDEvento, IDTipoEvento, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, URLImagen, URLArchivos[], descripcion)
new()
<<entity>>
EventoEventAdministratorInstance
(from PIM)
newEvento(EventoEventData)
getListaEventos(orden)
getEventoData(IDEvento) : EventoEventData
modificarEstatusEvento(IDEvento, newEstatusPublicable)
updateEventoData(EventoEventData)
EventoEventAdministratorInstance()
<<DAO>>
EventoEventDAO
(from PIM)
insertEvento(EventoEventData)
selectEventos(orden)
selectEventoData(IDEvento) : EventoEventData
updateEstatusEvento(IDEvento, newEstatusPublicable)
updateEvento(EventoEventData)

4.5.5 Diagrama de clases de publica eventos

EL JSP-boundary DSBElMenuForm lanza la solicitud para publicar un evento al Servlet EventoEventDispatcher, el Servlet envía el JSP-boundary EventoEventPublicaForm en donde se recuperan los datos del evento para posteriormente enviarlos al Servlet EventoEventDispatcher, éste redirecciona hacia otro Servlet, el EventoEventControllerPublica, quien es cliente del EJB EventoEventAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (EventoEventAdministratorHome) y de InterfaceRemote (EventoEventAdministrator), las cuales a través de la clase DAO EventoEventDAO, en donde se encuentran las operaciones SQL para realizar la inserción de los datos del evento contenidos en el objeto Entity EventoEventData. El Servlet EventoEventControllerPublica envía los datos adicionales de archivos contenidos en el evento al Helper EventoEventHelper. Posteriormente el Servlet EventoEventControllerPublica envía una notificación de que se realizó la inserción a través del JSP EventoEventNotificationForm


<<boundary>>
DSBElMenuForm
(from PIM)
submit(EventoEventPublicaForm)
submit(EventoEventListadoForm, orden)
<<boundary>>
EventoEventPublicaForm
forward()
submit(IDTipoEvento, nombreTipoEvento, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, URLImagen, numArchivos, URLArchivos[])
<<control>>
EventoEventDispatcher
post(PublicaEventoCommand)
post(nuevoEventoCommand, IDTipoEvento, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[], descripcion)
post(ListaEventosCommand, orden)
post(consultaEventoCommand, IDEvento)
post(cambiaEstatusCommand, IDEvento, newEstatusPublicable)
post(modificaEventoCommandStep1, IDEvento)
post(modificaEventoCommandStep2, IDEvento, newIDTipoEvento, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[], descripcion)
validateSession()
<<boundary>>
EventoEventNotificationForm
forward(confirmationMessage)
<<helper>>
EventoEventHelper
saveFiles(files[], imageFile)
<<control>>
EventoEventControllerPublica
forward(nuevoEventoCommand, IDTipoEvento, nombreTipoEvento, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[])
<<valueobject>>
EventoEventData
(from PIM)
IDEvento
IDTipoEvento
titulo
fechaInicio
fechaFin
fechaPublicacion
email
notas
estatusPublicable
URLImagen
URLArchivos[]
descripcion
new(IDEvento, IDTipoEvento, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, URLImagen, URLArchivos[], descripcion)
new()
<<RemoteInterface>>
EventoEventAdministrator
newEvento(EventoEventData)
getListaEventos(orden)
getEventoData(IDEvento)
modificarEstatusEvento(IDEvento, newEstatusPublicable)
updateEventoData(EventoEventData)
<<HomeInterface>>
EventoEventAdministratorHome
create() : EventoEventAdministrator
<<instantiate>>
<<entity>>
EventoEventAdministratorInstance
newEvento(EventoEventData)
getListaEventos(orden)
getEventoData(IDEvento) : EventoEventData
modificarEstatusEvento(IDEvento, newEstatusPublicable)
updateEventoData(EventoEventData)
EventoEventAdministratorInstance()
<<DAO>>
EventoEventDAO
insertEvento(EventoEventData)
selectEventos(orden)
selectEventoData(IDEvento) : EventoEventData
updateEstatusEvento(IDEvento, newEstatusPublicable)
updateEvento(EventoEventData)
0..*
1

4.5.6 Diagrama de secuencia y de colaboración de cambia estatus a eventos publicados

Empleando la lista de eventos, el actor DSBEI envía la solicitud de cambio de estatus mediante el mensaje submit(). La interfaz EventoEventListadoForm reenvía los datos a la clase EventoEventDispatcher. En el mensaje se envía el identificador del evento y su nuevo estatus. De la clase EventoEventDispatcher se ejecuta el método validateSession donde al ser valida la sesión reenvía dichos datos a la clase EventoEventControllerStatus. Ésta última, invoca al método create de la interfaz EventoEventAdministratorHome y así obtener una referencia del tipo de la interfaz remota EventoEventAdministrator. Una vez hecho esto, invoca al método modificarEstatusEvento de la clase EventoEventAdministrator que a su vez empleará el método modificarEstatusEvento de la clase EventoEventAdministratorInstance en donde se empleará el método updateEstatusEvento de la clase EventoEventDAO quien se ocupara de realizar el cambio del estatus, en la base de datos, al evento seleccionado.

Ya que se realizo la actualización, la clase EventoEventControllerStatus envía el mensaje de confirmación a la interfaz EventoEventNotificationForm, donde se presentara la notificación del cambio realizado al actor.

Diagrama de secuencia de cambia estatus a eventos publicados

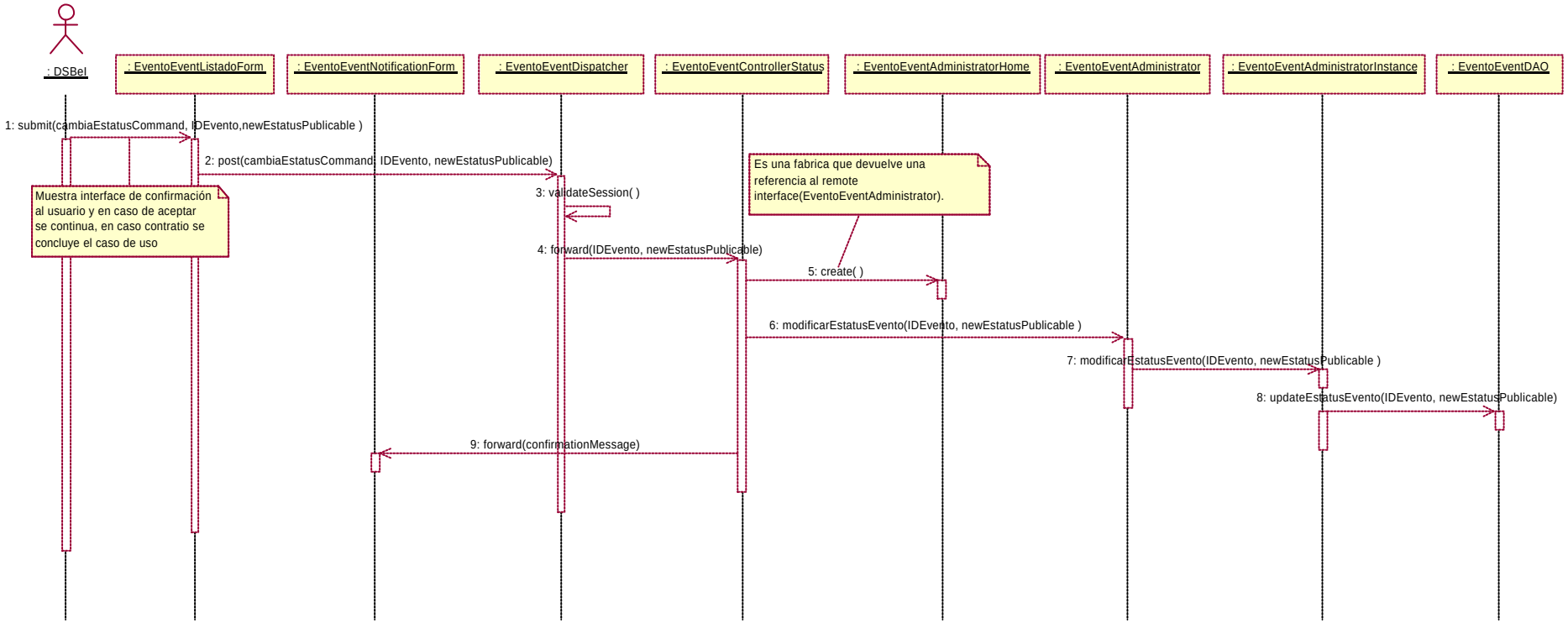
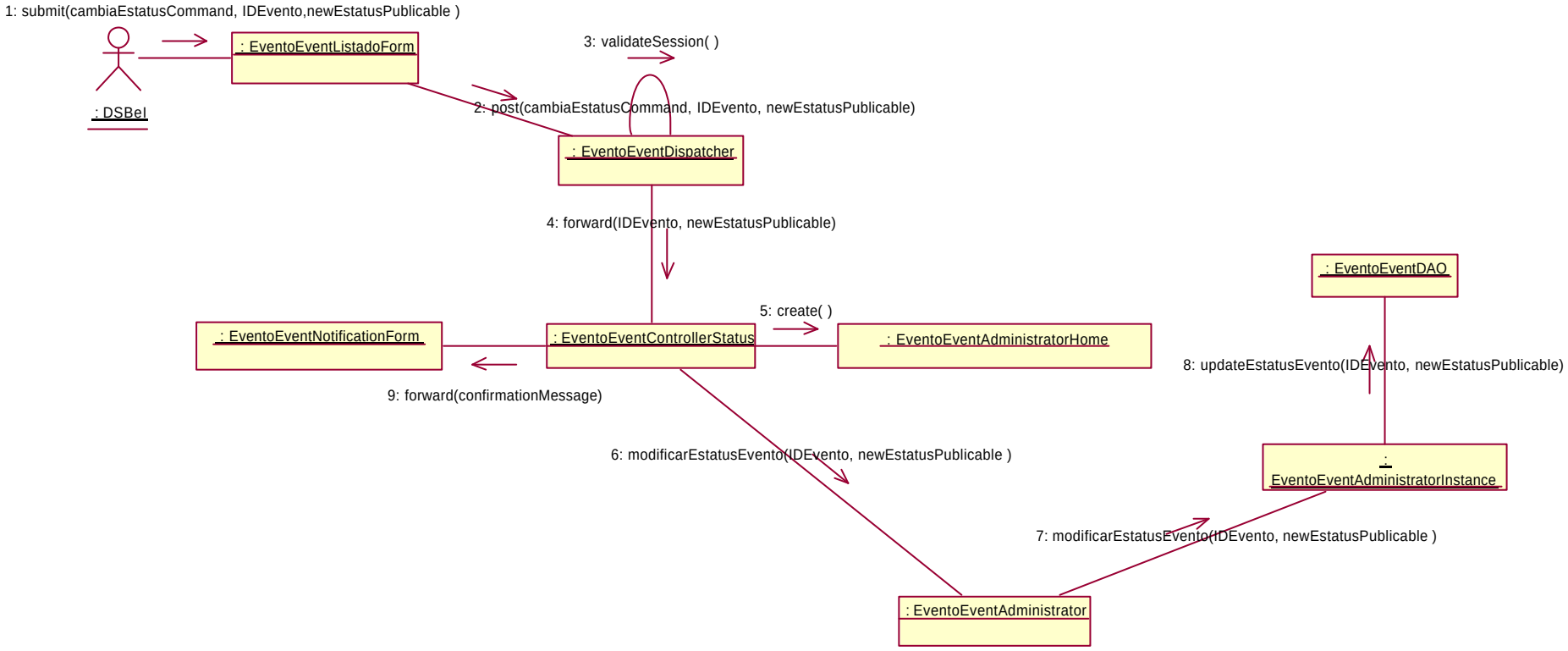


Diagrama de colaboración de cambia estatus a eventos publicados



4.5.7 Diagrama de secuencia y de colaboración de consulta información completa del evento

Empleando la lista de eventos, el actor DSBEI envía la solicitud de consulta y el identificador del evento a consultar mediante el mensaje submit(). La interfaz EventoEventListadoForm reenvía los datos a la clase EventoEventDispatcher. La clase EventoEventDispatcher ejecuta el método validateSession donde al ser valida la sesión reenvía dichos datos a la clase EventoEventControllerConsulta. Ésta última invoca al método create de la interfaz EventoEventAdministratorHome y así obtener una referencia del tipo de la interfaz remota EventoEventAdministrator. Una vez hecho esto, se invoca al método getEventoData de la clase EventoEventAdministrator y ésta, posteriormente reenviara la solicitud de información del evento a la clase EventoEventAdministratorInstance quien finalmente empleará el método selectEventoData de la clase EventoEventDAO para colocar la información del evento en una instancia de la clase EventoEventData donde viajara desde la clase EventoEventControllerConsulta a la interfaz EventoEventDetallesForm para ser presentada finalmente al actor.

Diagrama de secuencia de consulta información completa del evento

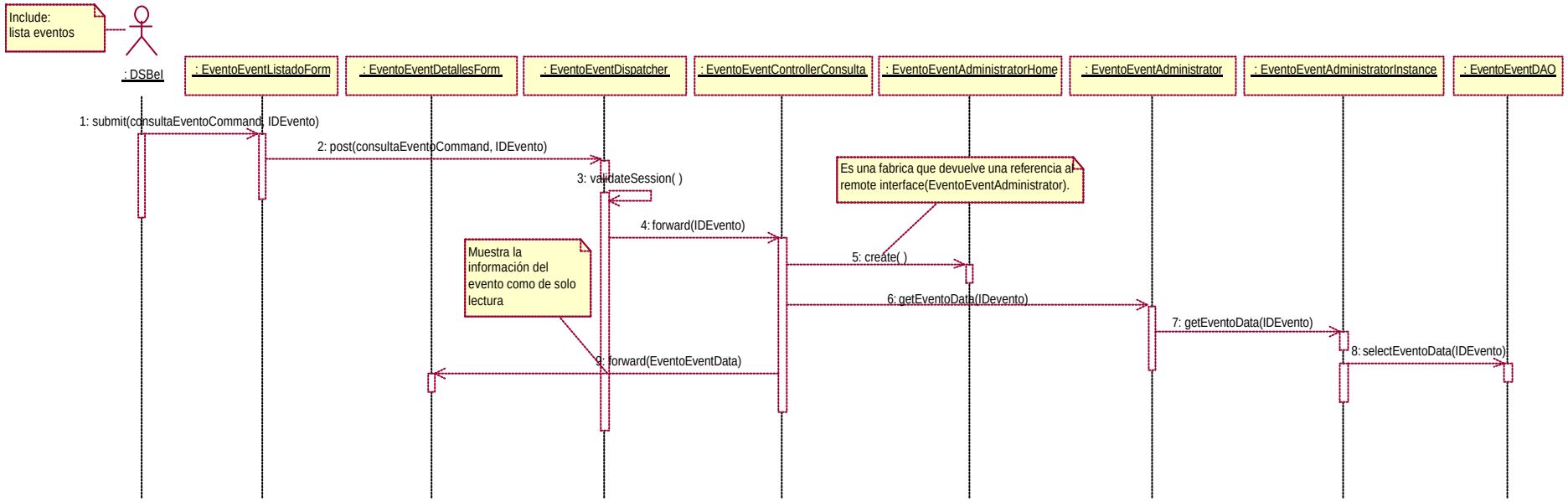
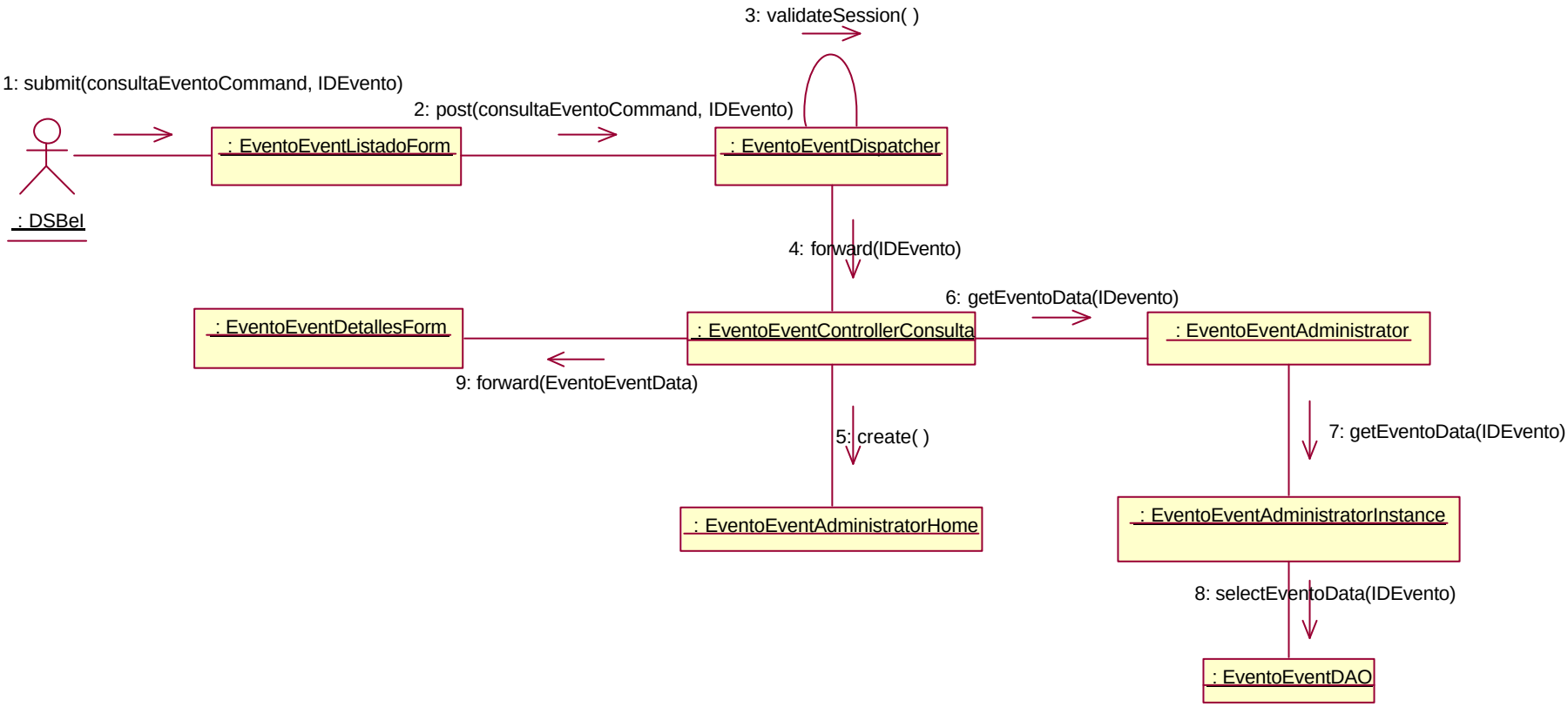


Diagrama de colaboración de consulta información completa del evento



4.5.8 Diagrama de secuencia y de colaboración de lista eventos

El actor DSBEI envía la solicitud de listado de los eventos mediante el mensaje submit(). La interfaz DSBEIMenuForm reenvía los datos a la clase EventoEventDispatcher. En el mensaje se indica el criterio por el cual se realiza la selección de los eventos para el listado. La clase EventoEventDispatcher ejecuta el método validateSession donde al ser valida la sesión reenvía dichos datos a la clase EventoEventControllerListado. Ésta última, invoca al método create de la interfaz EventoEventAdministratorHome y así obtener una referencia del tipo de la interfaz remota EventoEventAdministrator. Una vez hecho esto, invoca al método getListaEventos de la clase EventoEventAdministrator que a su vez empleará el método getListaEventos de la clase EventoEventAdministratorInstance en donde se utilizará el método selectEventos de la clase EventoEventDAO el cual genera un objeto de la clase EventoEventLista en donde se contendrán el conjunto de objetos de la clase eventoEventData. Finalmente el objeto de la clase EventoEventLista que contiene la información obtenida de la consulta a la base de datos se envía a la interfaz EventoEventListadoForm para presentar la información al actor.

Diagrama de secuencia de lista eventos

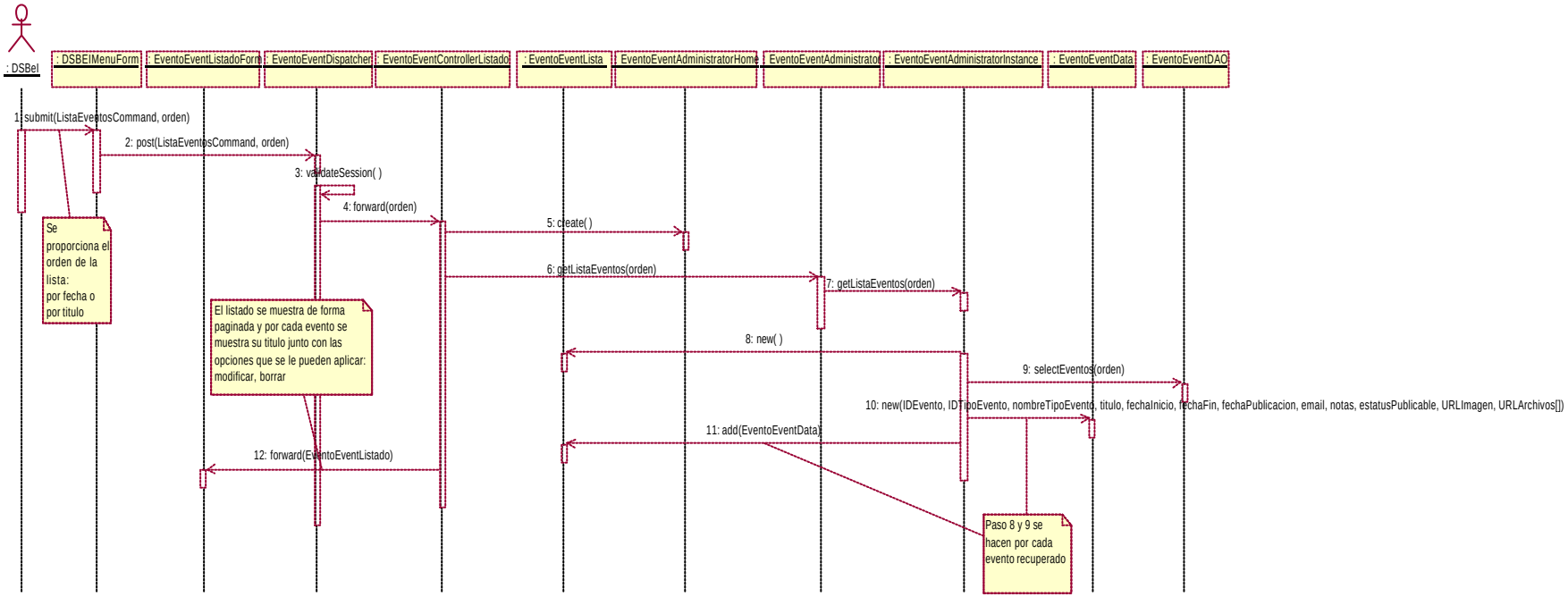
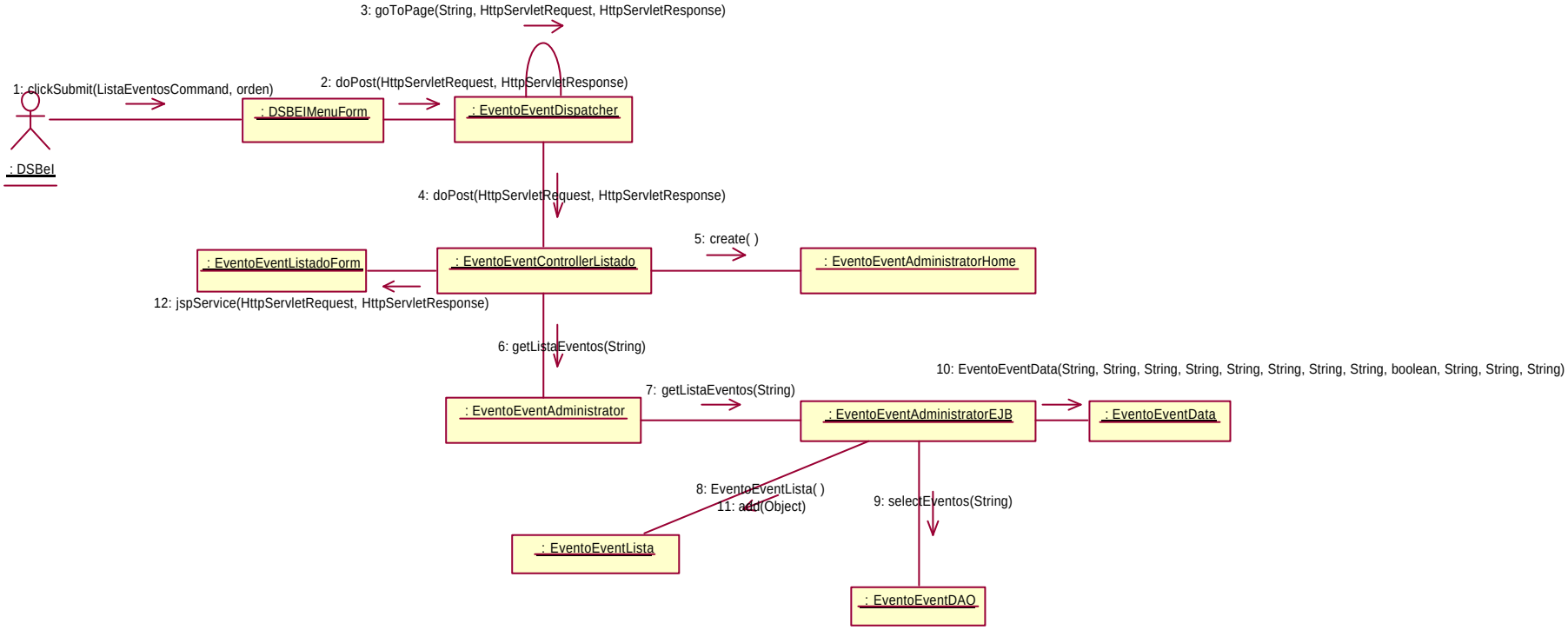


Diagrama de colaboración de lista eventos



4.5.9 Diagrama de secuencia y de colaboración de modifica datos a eventos publicados.

La secuencia para modificar datos de una evento se divide en dos etapas. La primera muestra los datos del evento a modificar. En la segunda se envían los datos ya modificados del evento para que se reflejen en la base de datos.

En la primer etapa, el actor DSBEI emplea la lista de eventos y envía la solicitud de modificar el evento mediante el mensaje submit(). La interfaz EventoEventListadoForm reenvía los datos a la clase EventoEventDispatcher. En el mensaje se indica el identificador del evento a modificar, así como también se especifica que el proceso se encuentra en su primera etapa.

De la clase EventoEventDispatcher se ejecuta el método validateSession donde al ser valida la sesión reenvía dichos datos a la clase EventoEventControllerModificaStep1. Ésta última, invoca al método create de la interfaz EventoEventAdministratorHome y así obtener una referencia del tipo de la interfaz remota EventoEventAdministrator y ésta, posteriormente reenviara la solicitud de información del evento a la clase EventoEventAdministratorInstance quien finalmente empleará el método selectEventoData de la clase EventoDAO para colocar la información del evento en una instancia de la clase EventoEventData. La instancia con los datos del evento se pasa como parámetro al método setSessionAttribute de la clase EventoEventControllerModificaStep1 y posteriormente a la interfaz EventoEventModificaForm, en donde se ejecuta el método getSessionAttribute con el que recuperarán los datos del evento seleccionado, y así presentarlos en la interfaz para poder realizar los cambios correspondientes.

En la segunda etapa, el actor DSBEI envía el mensaje submit() con el identificador del evento a modificar, el parámetro que indica que el proceso se encuentra en la segunda etapa y los datos ya modificados del evento. La interfaz EventoEventModificaForm es empleada para realizar los cambios a los datos del evento presentado y reenvía los datos a la clase EventoEventDispatcher.

De la clase EventoEventDispatcher se ejecuta el método validateSession donde al ser valida la sesión reenvía dichos datos a la clase EventoEventControllerModificaStep2. Ésta clase ejecuta el método saveFiles de la clase EventoEventHelper para guardar en el sistema de archivos, los archivos que conforman los datos del evento. Posteriormente se invoca al constructor de la clase EventoEventData donde se asignarán los datos ya modificados a los atributos del evento seleccionado. La nueva instancia, se envía como parámetro en el método updateEventoData, el cual es ejecutado desde la Interfaz remota EventoEventAdministrator y que posteriormente invocara al mismo método de la clase EventoEventAdministratorInstance.

Éste último método emplea el método updateEvento de la clase EventoEventDAO para almacenar la información del objeto de la clase EventoEventData en la Base de Datos. Una vez que se guardan los cambios del evento, se envía un mensaje de confirmación de la actualización mediante la interfaz EventoEventNotificationForm.

Diagrama de secuencia y de colaboración de modifica datos a eventos publicados

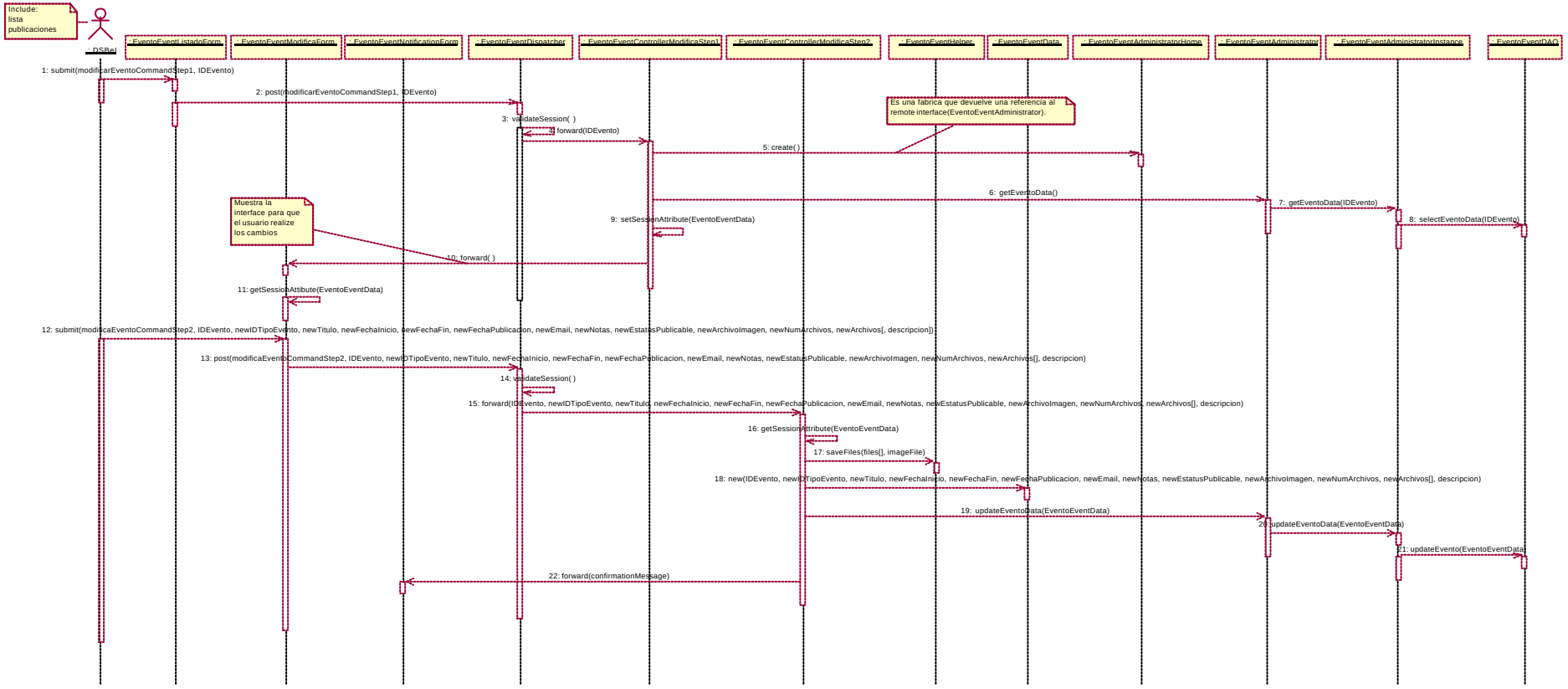
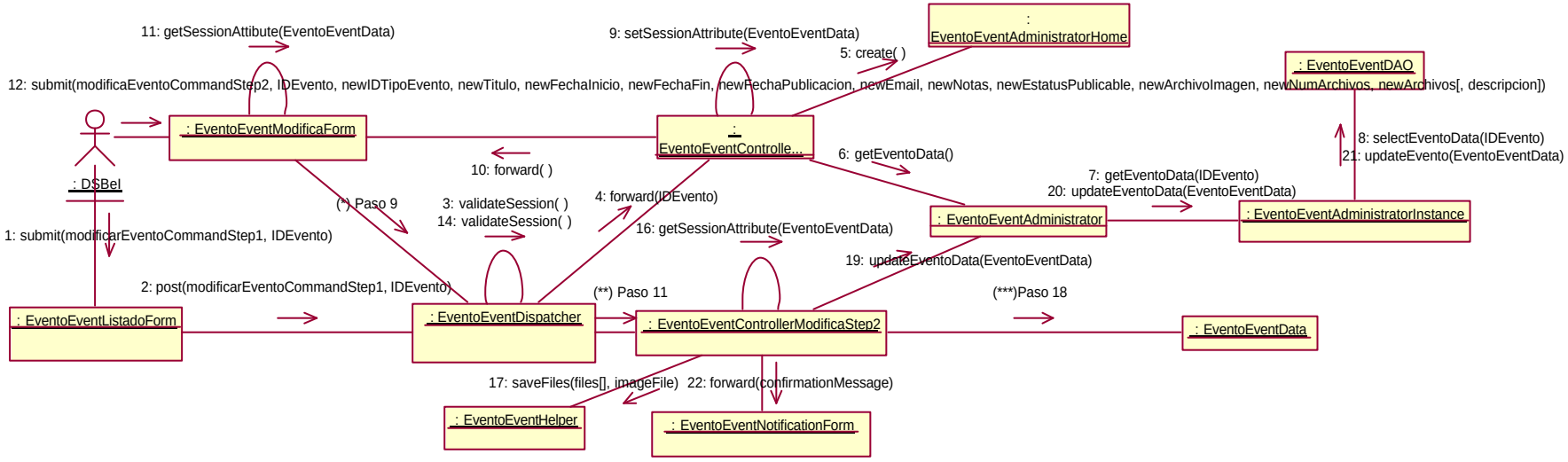


Diagrama de secuencia y de colaboración de modifica datos a eventos publicados



(*) 13: post(modificaEventoCommandStep2, IDEvento, newIDTipoEvento, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[], descripcion)

(**) 15: forward(IDEvento, newIDTipoEvento, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[], descripcion)

(***) 18: new(IDEvento, newIDTipoEvento, newTitulo, newFechaInicio, newFechaFin, newFechaPublicacion, newEmail, newNotas, newEstatusPublicable, newArchivoImagen, newNumArchivos, newArchivos[], descripcion)

4.5.10 Diagrama de secuencia y de colaboración de publica eventos

EL actor DSBel envía la solicitud de publicar eventos mediante el mensaje submit(). La interfaz DSBEIMenuForm reenvía la información a la clase EventosEventDispatcher. De la clase EventosEventDispatcher se ejecuta el método validateSession donde al ser valida la sesión presenta la interfaz EventoEventPublicaForm para que el usuario introduzca los datos del nuevo evento.

El actor DSBei envía los datos del nuevo evento en el mensaje submit desde la interfaz EventoEventPublicaForm a la clase EventoEventDispatcher. La clase EventoEventDispatcher ejecuta el método validateSession para posteriormente reenviar dichos datos a la clase EventoEventControllerPublica siempre y cuando la sesión sea válida. La clase EventoEventControllerPublica invoca al método create de la interfaz EventoEventAdministratorHome y así obtener una referencia del tipo de la interfaz remota EventoEventAdministrator.

Posteriormente, la clase EventoEventControllerPublica invoca al constructor de la clase EventoEventData donde se establecerán los atributos de la nueva instancia con los datos correspondientes al nuevo evento. A continuación, se invoca al método newEvento de la Interfaz remota EventoEventAdministrator desde donde se invocara al mismo método de la clase EventoEventAdministratorInstance. El constructor de la clase EventoEventAdministratorInstance emplea el método insertEvento de la clase EventoEventDAO para almacenar la información del objeto de la clase EventoEventData en la Base de Datos. Una vez que la información es almacenada, se ejecuta el método saveFiles de la clase EventoEventHelper para guardar en el sistema de archivos, los archivos que conforman el evento.

Finalmente la clase EventoEventControllerPublica envía el mensaje de confirmación a la interfaz EventoEventNotificationForm, donde se presentara la notificación de que la alta del evento ocurrió de forma correcta.

Diagrama de secuencia de publica eventos

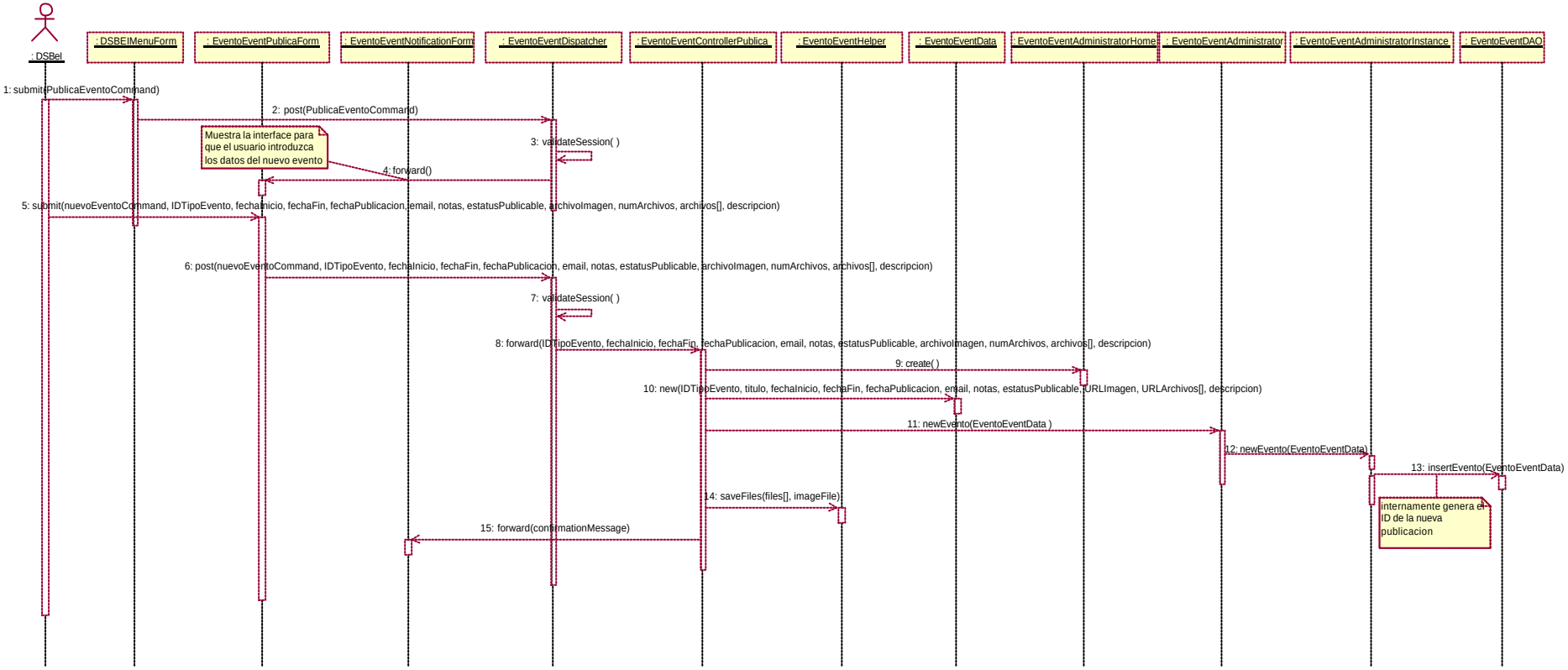
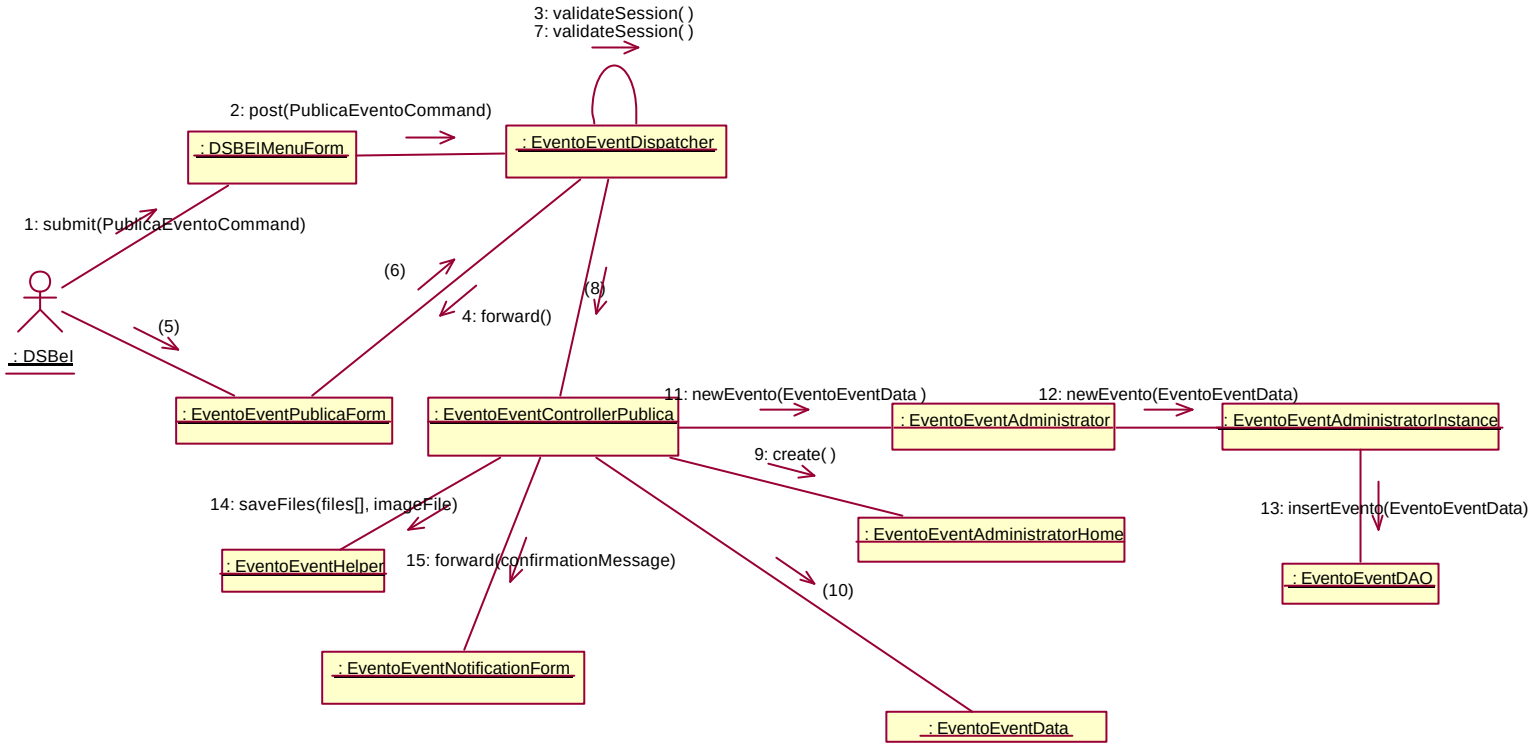


Diagrama de colaboración de publica eventos



5: submit(nuevoEventoCommand, IDTipoEvento, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[], descripcion)

6: post(nuevoEventoCommand, IDTipoEvento, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[], descripcion)

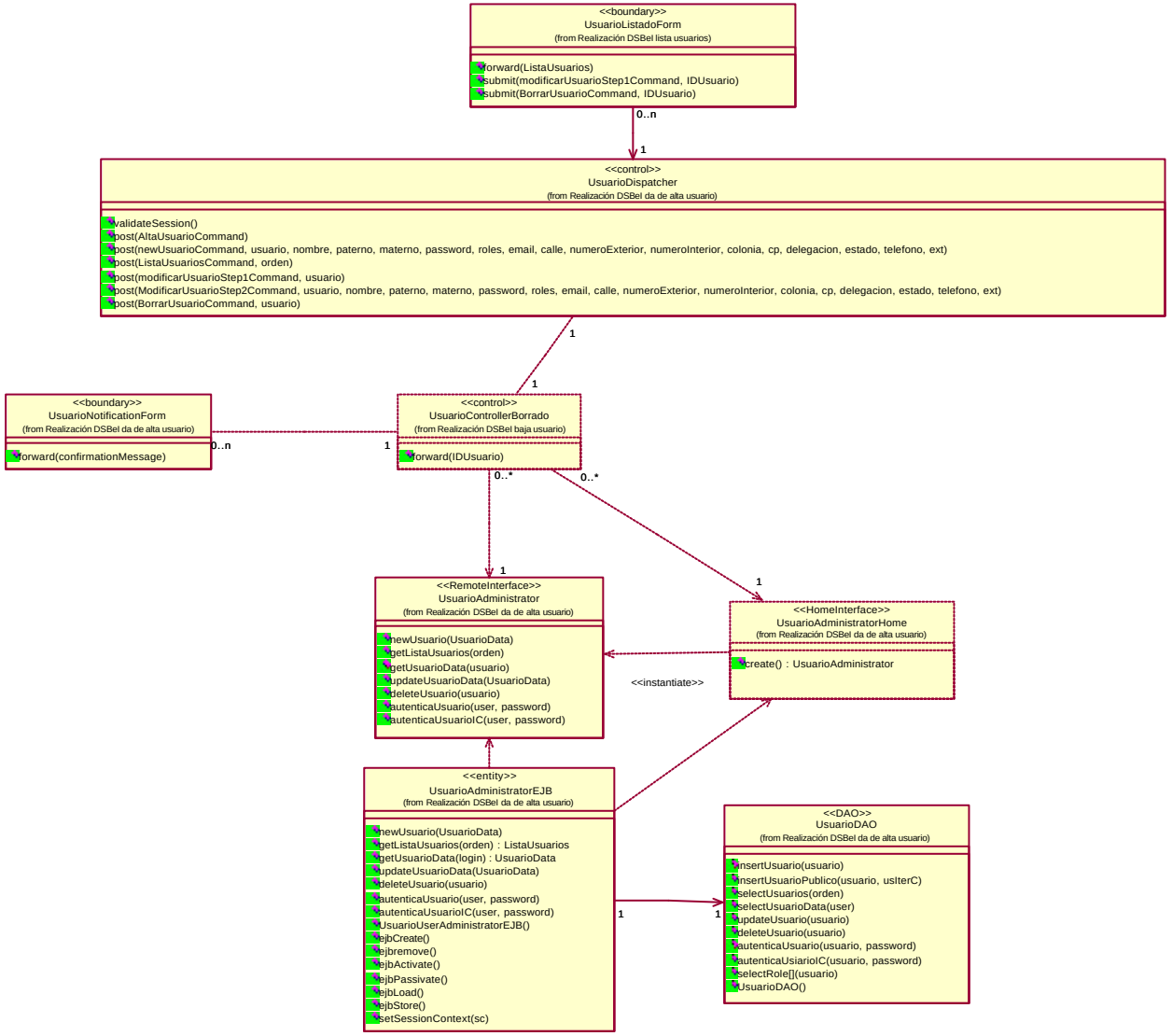
8: forward(IDTipoEvento, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, archivoImagen, numArchivos, archivos[], descripcion)

10: new(IDTipoEvento, titulo, fechaInicio, fechaFin, fechaPublicacion, email, notas, estatusPublicable, URLImagen, URLArchivos[], descripcion)

4.6 Administración de usuarios

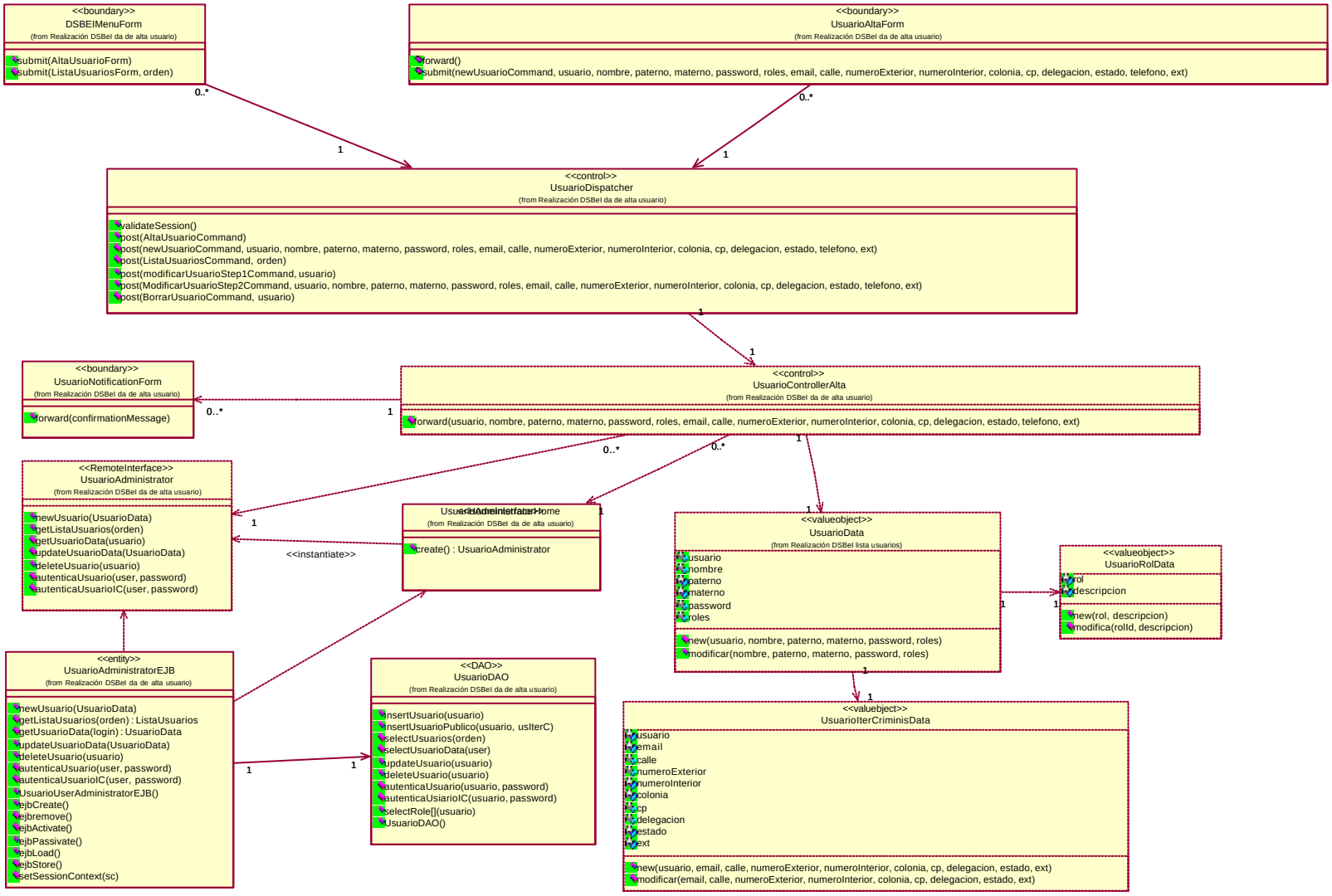
4.6.1 Diagrama de clases de baja de usuarios

JSP-boundary UsuarioListadoForm pasa los datos al Servlet UsuarioDispatcher, éste redirecciona hacia otro Servlet, el UsuarioControllerBorrado Éste último, es cliente del EJB UsuarioAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (UsuarioAdministratorHome) y de InterfaceRemote (UsuarioAdministrator), las cuales a través de la clase DAO UsuarioDAO, quien tiene las operaciones SQL, realiza la baja del usuario. Posteriormente el Servlet UsuarioControllerBorrado envía una notificación de que se realizó el cambio a través del JSP UsuarioNotificationForm)


<<boundary>>
UsuarioListadoForm
(from Realización DSBel lista usuarios)
forward(ListaUsuarios)
submit(modificarUsuarioStep1Command, IDUsuario)
submit(BorrarUsuarioCommand, IDUsuario)
0..n
1
<<control>>
UsuarioDispatcher
(from Realización DSBel da de alta usuario)
validateSession()
post(AltaUsuarioCommand)
post(newUsuarioCommand, usuario, nombre, paterno, materno, password, roles, email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, telefono, ext)
post(ListaUsuariosCommand, orden)
post(modificarUsuarioStep1Command, usuario)
post(ModificarUsuarioStep2Command, usuario, nombre, paterno, materno, password, roles, email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, telefono, ext)
post(BorrarUsuarioCommand, usuario)
1
1
<<boundary>>
UsuarioNotificationForm
(from Realización DSBel da de alta usuario)
forward(confirmationMessage)
0..n
1
<<control>>
UsuarioControllerBorrado
(from Realización DSBel baja usuario)
forward(IDUsuario)
0..*
0..*
1
1
<<RemoteInterface>>
UsuarioAdministrator
(from Realización DSBel da de alta usuario)
newUsuario(UsuarioData)
getListaUsuarios(orden)
getUsuarioData(usuario)
updateUsuarioData(UsuarioData)
deleteUsuario(usuario)
autenticaUsuario(user, password)
autenticaUsuarioIC(user, password)
<<HomeInterface>>
UsuarioAdministratorHome
(from Realización DSBel da de alta usuario)
create() : UsuarioAdministrator
<<instantiate>>
<<entity>>
UsuarioAdministratorEJB
(from Realización DSBel da de alta usuario)
newUsuario(UsuarioData)
getListaUsuarios(orden) : ListaUsuarios
getUsuarioData(login) : UsuarioData
updateUsuarioData(UsuarioData)
deleteUsuario(usuario)
autenticaUsuario(user, password)
autenticaUsuarioIC(user, password)
UsuarioUserAdministratorEJB()
ejbCreate()
ejbremove()
ejbActivate()
ejbPassivate()
ejbLoad()
ejbStore()
setSessionContext(sc)
1
1
<<DAO>>
UsuarioDAO
(from Realización DSBel da de alta usuario)
insertUsuario(usuario)
insertUsuarioPublico(usuario, usIterC)
selectUsuarios(orden)
selectUsuarioData(user)
updateUsuario(usuario)
deleteUsuario(usuario)
autenticaUsuario(usuario, password)
autenticaUsiarioIC(usuario, password)
selectRole[](usuario)
UsuarioDAO()

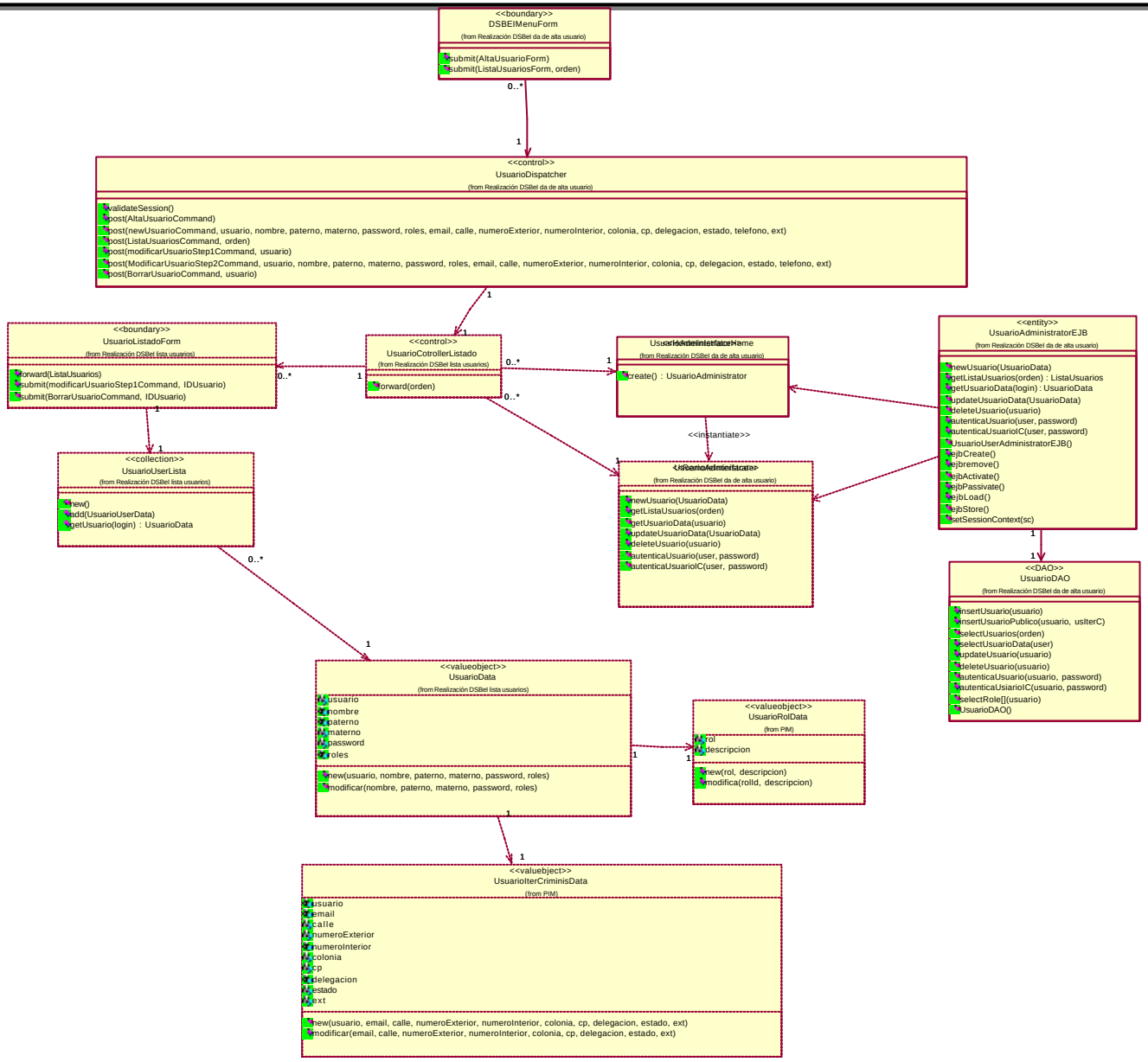
4.6.2 Diagrama de clases de alta de usuarios

EL JSP-boundary DSBElMenuForm lanza la solicitud para dar de alta un usuario al Servlet UsuarioDispatcher, el Servlet envía el JSP-boundary UsuarioAltaForm en donde se recuperan los datos del evento para posteriormente enviarlos al Servlet UsuarioDispatcher, quien redirecciona hacia otro Servlet, el UsuarioControllerAlta, éste último, es cliente del EJB UsuarioAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (UsuarioAdministratorHome) y de InterfaceRemote (UsuarioAdministrator), las cuales a través de la clase DAO UsuarioDAO, en donde se tienen las operaciones SQL para realizar la inserción de los datos del evento contenidos en el objeto Entity UsuarioData. Posteriormente el Servlet UsuarioControllerAlta envía una notificación de que se realizó la inserción a través del JSP UsuarioNotificationForm.


<<boundary>>
DSBElMenuForm
(from Realización DSBel da de alta usuario)
submit(AltaUsuarioForm)
submit(ListaUsuariosForm, orden)
<<boundary>>
UsuarioAltaForm
(from Realización DSBel da de alta usuario)
forward()
submit(newUsuarioCommand, usuario, nombre, paterno, materno, password, roles, email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, telefono, ext)
<<control>>
UsuarioDispatcher
(from Realización DSBel da de alta usuario)
validateSession()
post(AltaUsuarioCommand)
post(newUsuarioCommand, usuario, nombre, paterno, materno, password, roles, email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, telefono, ext)
post(ListaUsuariosCommand, orden)
post(modificarUsuarioStep1Command, usuario)
post(ModificarUsuarioStep2Command, usuario, nombre, paterno, materno, password, roles, email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, telefono, ext)
post(BorrarUsuarioCommand, usuario)
<<boundary>>
UsuarioNotificationForm
(from Realización DSBel da de alta usuario)
forward(confirmationMessage)
<<control>>
UsuarioControllerAlta
(from Realización DSBel da de alta usuario)
forward(usuario, nombre, paterno, materno, password, roles, email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, telefono, ext)
<<RemoteInterface>>
UsuarioAdministrator
(from Realización DSBel da de alta usuario)
newUsuario(UsuarioData)
getListaUsuarios(orden)
getUsuarioData(usuario)
updateUsuarioData(UsuarioData)
deleteUsuario(usuario)
autenticaUsuario(user, password)
autenticaUsuarioIC(user, password)
<<instantiate>>
UsuarioAdministrator-Home
(from Realización DSBel da de alta usuario)
create() : UsuarioAdministrator
<<valueobject>>
UsuarioData
(from Realización DSBel lista usuarios)
usuario
nombre
paterno
materno
password
roles
new(usuario, nombre, paterno, materno, password, roles)
modificar(nombre, paterno, materno, password, roles)
<<valueobject>>
UsuarioRolData
rol
descripcion
new(rol, descripcion)
modifica(rolId, descripcion)
<<entity>>
UsuarioAdministratorEJB
(from Realización DSBel da de alta usuario)
newUsuario(UsuarioData)
getListaUsuarios(orden) : ListaUsuarios
getUsuarioData(login) : UsuarioData
updateUsuarioData(UsuarioData)
deleteUsuario(usuario)
autenticaUsuario(user, password)
autenticaUsuarioIC(user, password)
UsuarioUserAdministratorEJB()
ejbCreate()
ejbremove()
ejbActivate()
ejbPassivate()
ejbLoad()
ejbStore()
setSessionContext(sc)
<<DAO>>
UsuarioDAO
(from Realización DSBel da de alta usuario)
insertUsuario(usuario)
insertUsuarioPublico(usuario, usIterC)
selectUsuarios(orden)
selectUsuarioData(user)
updateUsuario(usuario)
deleteUsuario(usuario)
autenticaUsuario(usuario, password)
autenticaUsiarioIC(usuario, password)
selectRole[](usuario)
UsuarioDAO()
<<valuebject>>
UsuarioIterCriminisData
usuario
email
calle
numeroExterior
numeroInterior
colonia
cp
delegacion
estado
ext
new(usuario, email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, ext)
modificar(email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, ext)

4.6.3 Diagrama de clases de lista usuarios

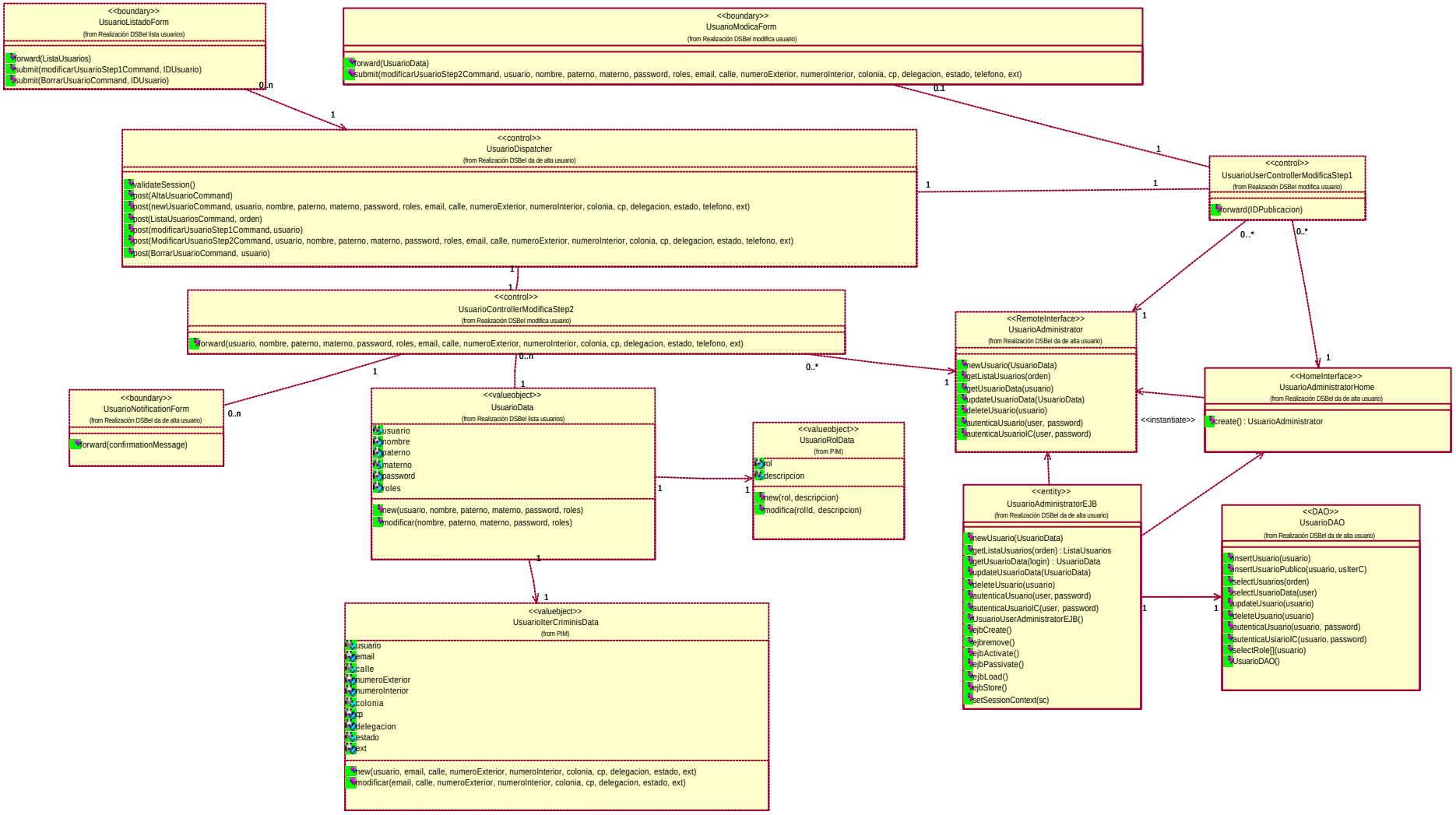
EL JSP-boundary DSBElMenuForm pasa los datos al Servlet UsuarioDispatcher, éste redirecciona hacia otro Servlet, el UsuarioControllerListado. Éste último, es cliente del EJB UsuarioAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (UsuarioAdministratorHome) y de InterfaceRemote (UsuarioAdministrator), las cuales a través de la clase DAO UsuarioDAO, quien tiene las operaciones SQL, obtiene al objeto collection UsuarioUserLista que contiene a la clase Entity UsuarioData y ésta a su vez contiene a la clase UsuarioRolData. El Servlet UsuarioControllerListado pasa el objeto collection UsuarioUserLista al JSP UsuarioListadoForm, donde se mostrarán los usuarios obtenidos de la consulta.


<<boundary>>
DSBElMenuForm
(from Realización DSBel da de alta usuario)
submit(AltaUsuarioForm)
submit(ListaUsuariosForm, orden)
0..*
1
<<control>>
UsuarioDispatcher
(from Realización DSBel da de alta usuario)
validateSession()
post(AltaUsuarioCommand)
post(newUsuarioCommand, usuario, nombre, paterno, materno, password, roles, email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, telefono, ext)
post(ListaUsuariosCommand, orden)
post(modificarUsuarioStep1Command, usuario)
post(ModificarUsuarioStep2Command, usuario, nombre, paterno, materno, password, roles, email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, telefono, ext)
post(BorrarUsuarioCommand, usuario)
1
<<boundary>>
UsuarioListadoForm
(from Realización DSBel lista usuarios)
forward(ListaUsuarios)
submit(modificarUsuarioStep1Command, IDUsuario)
submit(BorrarUsuarioCommand, IDUsuario)
0..*
1
<<control>>
UsuarioCotrollerListado
(from Realización DSBel lista usuarios)
forward(orden)
0..*
1
0..*
<<entity>>
UsuarioAdministratorEJB
(from Realización DSBel da de alta usuario)
newUsuario(UsuarioData)
getListaUsuarios(orden) : ListaUsuarios
getUsuarioData(login) : UsuarioData
updateUsuarioData(UsuarioData)
deleteUsuario(usuario)
autenticaUsuario(user, password)
autenticaUsuarioIC(user, password)
UsuarioUserAdministratorEJB()
ejbCreate()
ejbremove()
ejbActivate()
ejbPassivate()
ejbLoad()
ejbStore()
setSessionContext(sc)
create() : UsuarioAdministrator
<<instantiate>>
(from Realización DSBel da de alta usuario)
newUsuario(UsuarioData)
getListaUsuarios(orden)
getUsuarioData(usuario)
updateUsuarioData(UsuarioData)
deleteUsuario(usuario)
autenticaUsuario(user, password)
autenticaUsuarioIC(user, password)
<<collection>>
UsuarioUserLista
(from Realización DSBel lista usuarios)
new()
add(UsuarioUserData)
getUsuario(login) : UsuarioData
0..*
1
<<DAO>>
UsuarioDAO
(from Realización DSBel da de alta usuario)
insertUsuario(usuario)
insertUsuarioPublico(usuario, usIterC)
selectUsuarios(orden)
selectUsuarioData(user)
updateUsuario(usuario)
deleteUsuario(usuario)
autenticaUsuario(usuario, password)
autenticaUsiarioIC(usuario, password)
selectRole[](usuario)
UsuarioDAO()
<<valueobject>>
UsuarioData
(from Realización DSBel lista usuarios)
usuario
nombre
paterno
materno
password
roles
new(usuario, nombre, paterno, materno, password, roles)
modificar(nombre, paterno, materno, password, roles)
1
<<valueobject>>
UsuarioRolData
(from PIM)
rol
descripcion
new(rol, descripcion)
modifica(rolld, descripcion)
1
1
<<valuebject>>
UsuarioIterCriminisData
(from PIM)
usuario
email
calle
numeroExterior
numeroInterior
colonia
cp
delegacion
estado
ext
new(usuario, email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, ext)
modificar(email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, ext)

4.6.4 Diagrama de clases de modifica usuarios

EL JSP-boundary UsuarioListadoForm pasa los datos al Servlet UsuarioDispatcher, el cual redirecciona los datos al Servlet UsuarioUserControllerModificaStep1, quien es cliente del EJB UsuarioAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (UsuarioAdministratorHome) y de InterfaceRemote (UsuarioAdministrator), las cuales a través de la clase DAO UsuarioDAO realizan la consulta de los datos del usuario seleccionado y emplea el objeto Entity UsuarioData que a su vez emplea al objeto UsuarioRolData para presentar los datos del usuario en el JSP-boundary UsuarioModificaForm.

El Servlet UsuarioDispatcher envía los datos modificados del usuario seleccionado al Servlet UsuarioControllerModificaStep2, quien es cliente del EJB UsuarioAdministratorEJB, por lo tanto, obtiene una referencia de tipo InterfaceHome (UsuarioAdministratorHome) y de InterfaceRemote (UsuarioAdministrator), las cuales a través de la clase DAO UsuarioDAO, quien tiene las operaciones SQL para realizar la actualización de los datos del usuario contenidos en el objeto Entity UsuarioData que a su vez contiene a la clase UsuarioRolData. El Servlet UsuarioControllerModificaStep2 envía una notificación de que se realizó el cambio a través del JSP UsuarioNotificationForm.


<<boundary>>
UsuarioListadoForm
(from Realización DSBel lista usuarios)
forward(ListaUsuarios)
submit(modificarUsuarioStep1Command, IDUsuario)
submit(BorrarUsuarioCommand, IDUsuario)
<<boundary>>
UsuarioModicaForm
(from Realización DSBel modifica usuario)
forward(UsuarioData)
submit(modificarUsuarioStep2Command, usuario, nombre, paterno, materno, password, roles, email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, telefono, ext)
<<control>>
UsuarioDispatcher
(from Realización DSBel da de alta usuario)
validateSession()
post(AltaUsuarioCommand)
post(newUsuarioCommand, usuario, nombre, paterno, materno, password, roles, email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, telefono, ext)
post(ListaUsuariosCommand, orden)
post(modificarUsuarioStep1Command, usuario)
post(ModificarUsuarioStep2Command, usuario, nombre, paterno, materno, password, roles, email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, telefono, ext)
post(BorrarUsuarioCommand, usuario)
<<control>>
UsuarioUserControllerModificaStep1
(from Realización DSBel modifica usuario)
forward(IDPublicacion)
<<control>>
UsuarioControllerModificaStep2
(from Realización DSBel modifica usuario)
forward(usuario, nombre, paterno, materno, password, roles, email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, telefono, ext)
<<RemoteInterface>>
UsuarioAdministrator
(from Realización DSBel da de alta usuario)
newUsuario(UsuarioData)
getListaUsuarios(orden)
getUsuarioData(usuario)
updateUsuarioData(UsuarioData)
deleteUsuario(usuario)
autenticaUsuario(user, password)
autenticaUsuarioIC(user, password)
<<HomeInterface>>
UsuarioAdministratorHome
(from Realización DSBel da de alta usuario)
create() : UsuarioAdministrator
<<instantiate>>
<<boundary>>
UsuarioNotificationForm
(from Realización DSBel da de alta usuario)
forward(confirmationMessage)
<<valueobject>>
UsuarioData
(from Realización DSBel lista usuarios)
usuario
nombre
paterno
materno
password
roles
new(usuario, nombre, paterno, materno, password, roles)
modificar(nombre, paterno, materno, password, roles)
<<valueobject>>
UsuarioRolData
(from PIM)
rol
descripcion
new(rol, descripcion)
modifica(rolId, descripcion)
<<entity>>
UsuarioAdministratorEJB
(from Realización DSBel da de alta usuario)
newUsuario(UsuarioData)
getListaUsuarios(orden) : ListaUsuarios
getUsuarioData(login) : UsuarioData
updateUsuarioData(UsuarioData)
deleteUsuario(usuario)
autenticaUsuario(user, password)
autenticaUsuarioIC(user, password)
UsuarioUserAdministratorEJB()
ejbCreate()
ejbremove()
ejbActivate()
ejbPassivate()
ejbLoad()
ejbStore()
setSessionContext(sc)
<<DAO>>
UsuarioDAO
(from Realización DSBel da de alta usuario)
insertUsuario(usuario)
insertUsuarioPublico(usuario, usIterC)
selectUsuarios(orden)
selectUsuarioData(user)
updateUsuario(usuario)
deleteUsuario(usuario)
autenticaUsuario(usuario, password)
autenticaUsiarioIC(usuario, password)
selectRole[](usuario)
UsuarioDAO()
<<valueobject>>
UsuarioIterCriminisData
(from PIM)
usuario
email
calle
numeroExterior
numeroInterior
colonia
cp
delegacion
estado
ext
new(usuario, email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, ext)
modificar(email, calle, numeroExterior, numeroInterior, colonia, cp, delegacion, estado, ext)

4.6.5 Diagrama de secuencia y de colaboración de baja de usuarios

Empleando la lista de usuarios, el actor DSBEI deberá seleccionar al usuario a borrar. Se muestra una interfaz de confirmación donde en caso de aceptar continuar con la operación, se lanza la solicitud de eliminar al usuario mediante el envió del mensaje submit(). La interfaz UsuarioListadoForm reenvía los datos a la clase UsuarioDispatcher. En el mensaje se envía el identificador del usuario. De la clase UsuarioDispatcher se ejecuta el método validateSession donde al ser valida la sesión reenvía dichos datos a la clase UsuarioControllerBorrado. Ésta última, invoca al método create de la interfaz UsuarioAdministratorHome y así obtener una referencia del tipo de la interfaz remota UsuarioAdministrator. Una vez hecho esto, invoca al método deleteUsuario de la clase UsuarioAdministrator que a su vez empleará el método deleteUsuario de la clase UsuarioAdministratorEJB en donde se empleará el método deleteUsuario de la clase UsuarioDAO, quien se ocupara de realizar la baja del usuario en la base de datos.

Ya que se realizo la baja del usuario, la clase UsuarioControllerBorrado envía el mensaje de confirmación a la interfaz UsuarioNotificationForm, donde se presentara la notificación al actor de que la baja fue realizada.

Diagrama de secuencia de baja de usuarios

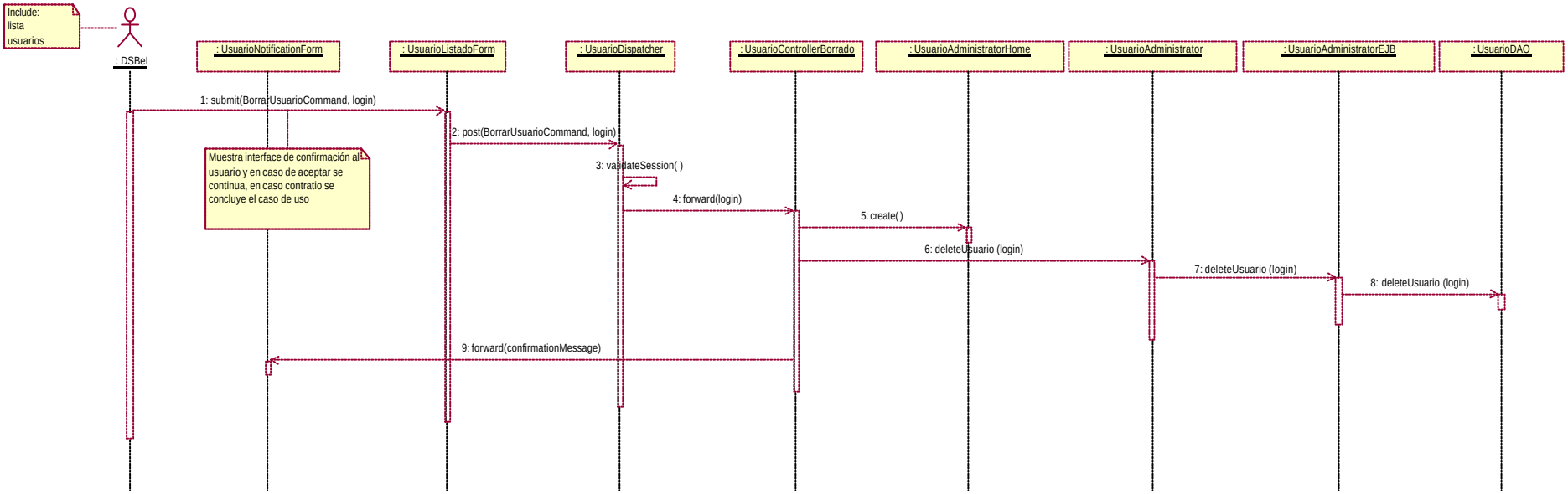
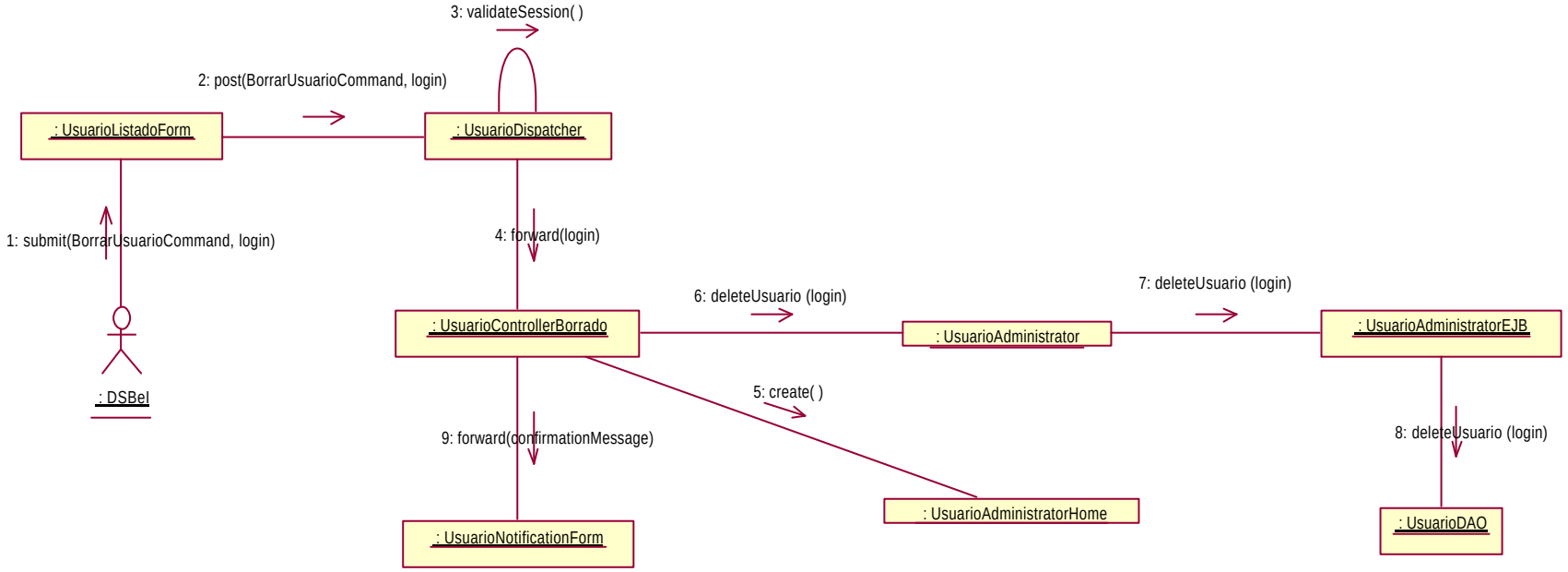


Diagrama de colaboración de baja de usuarios



4.6.6 Diagrama de secuencia y de colaboración de alta de usuarios

EL actor DSBel envía la solicitud de alta de usuario mediante el mensaje submit(). La interfaz DSBEIMenuForm reenvía la información a la clase UsuarioDispatcher. De la clase UsuarioDispatcher se ejecuta el método validateSession donde al ser valida la sesión presenta la interfaz UsuarioAltaForm para que el actor introduzca los datos del nuevo usuario.

El actor DSBei envía los datos del nuevo usuario en el mensaje submit desde la interfaz UsuarioAltaForm a la clase UsuarioDispatcher. La clase UsuarioDispatcher ejecuta el método validateSession para posteriormente reenvía dichos datos a la clase UsuarioControllerAlta siempre y cuando la sesión sea válida.

La clase UsuarioControllerAlta invoca al constructor de la clase UsuarioData donde se establecerán los atributos de la nueva instancia con los datos correspondientes del usuario a crear. Posteriormente, la clase UsuarioControllerAlta invoca al método create de la interfaz UsuarioAdministratorHome y así obtener una referencia del tipo de la interfaz remota UsuarioAdministrator.

A continuación se invoca al método newUsuario de la Interfaz remota UsuarioAdministrator desde donde se invocara al mismo método de la clase UsuarioAdministratorEJB. El constructor de la clase UsuarioAdministratorEJB emplea el método insertUsuario de la clase UsuarioDAO para almacenar la información del objeto de la clase UsuarioData en la base de datos

Finalmente la clase UsuarioControllerAlta envía el mensaje de confirmación a la interfaz UsuarioNotificationForm, donde se presentara la notificación actor de que la alta del usuario ocurrió de forma correcta.

Diagrama de secuencia de alta de usuarios

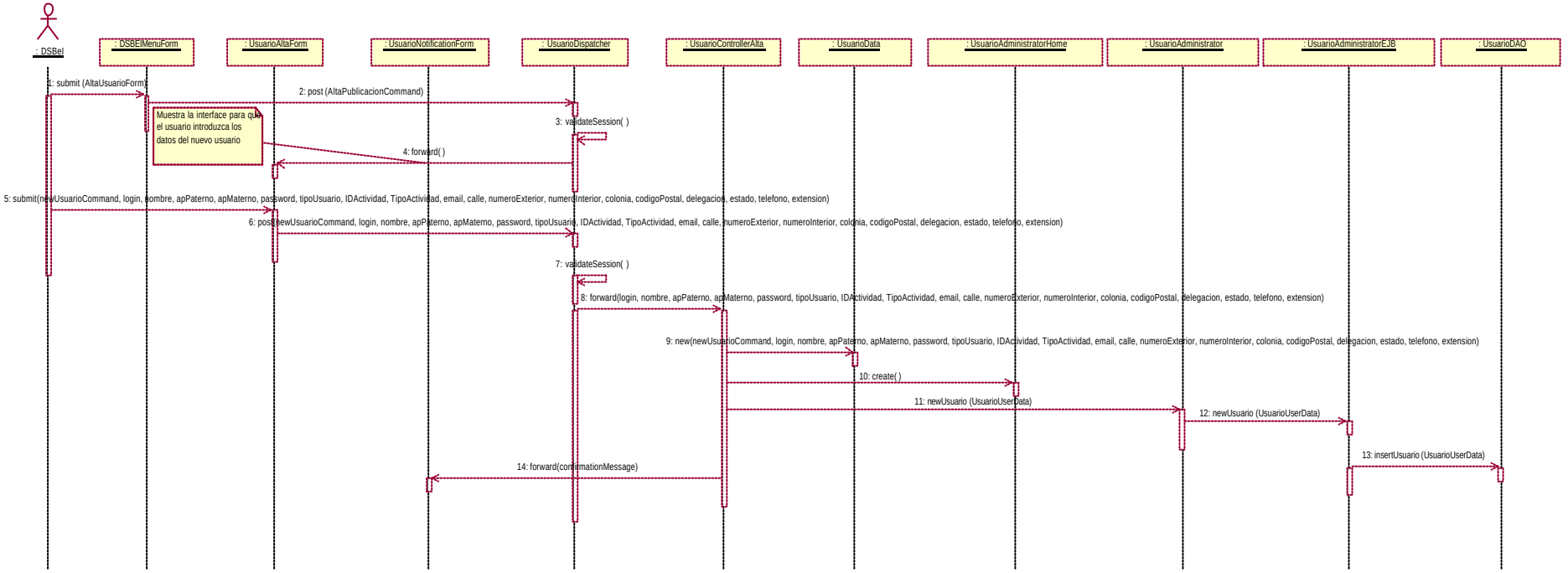
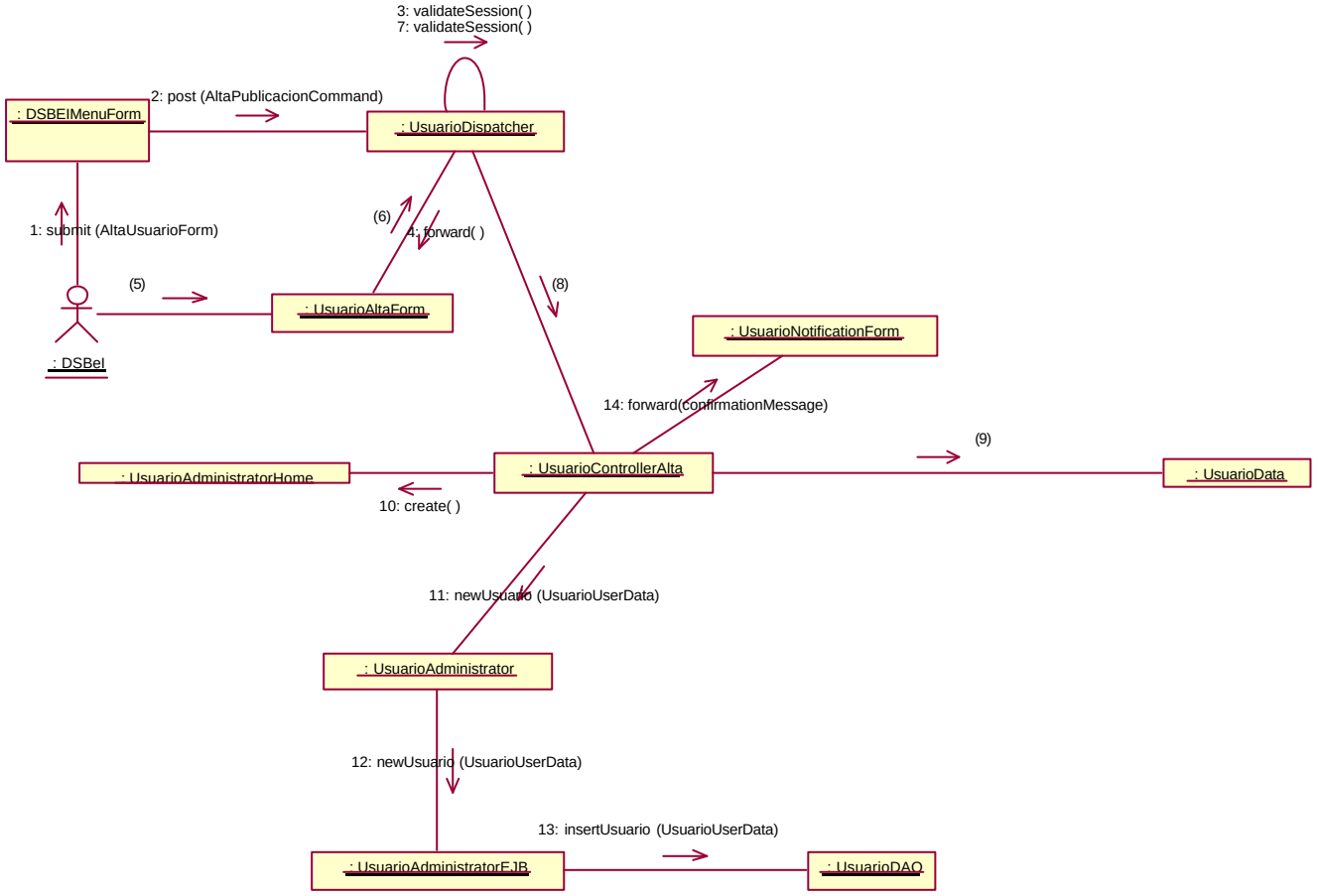


Diagrama de colaboración de alta de usuarios



5: submit(newUsuarioCommand, login, nombre, apPaterno, apMaterno, password, tipoUsuario, IDActividad, TipoActividad, email, calle, numeroExterior, numeroInterior, colonia, codigoPostal, delegacion, estado, telefono, extension)
6: post(newUsuarioCommand, login, nombre, apPaterno, apMaterno, password, tipoUsuario, IDActividad, TipoActividad, email, calle, numeroExterior, numeroInterior, colonia, codigoPostal, delegacion, estado, telefono, extension)
8: forward(login, nombre, apPaterno, apMaterno, password, tipoUsuario, IDActividad, TipoActividad, email, calle, numeroExterior, numeroInterior, colonia, codigoPostal, delegacion, estado, telefono, extension)
9: new(newUsuarioCommand, login, nombre, apPaterno, apMaterno, password, tipoUsuario, IDActividad, TipoActividad, email, calle, numeroExterior, numeroInterior, colonia, codigoPostal, delegacion, estado, telefono, extension)

4.6.7 Diagrama de secuencia y de colaboración de lista de usuarios

El actor DSBEI envía la solicitud de listado de usuarios mediante el mensaje submit(). La interfaz DSBEIMenuForm reenvía los datos a la clase UsuarioDispatcher. En el mensaje se indica el criterio por el cual se realiza la selección de los usuarios para el listado. La clase UsuarioDispatcher ejecuta el método validateSession donde al ser valida la sesión reenvía dichos datos a la clase UsuarioControllerListado. Ésta última, invoca al método create de la interfaz UsuarioAdministratorHome y así obtener una referencia del tipo de la interfaz remota UsuarioAdministrator. Una vez hecho esto, invoca al método getListaUsuarios de la clase UsuarioAdministratorEJB que a su vez empleará el método selectUsuarios de la clase UsuariosDAO. Por cada usuario recuperado se genera un objeto de la clase UsuarioData y cada objeto será agregado una instancia de la clase UsuarioUserLista mediante el método add.

Finalmente el objeto de la clase UsuarioLista que contiene la información obtenida de la consulta a la base de datos se envía a la interfaz UsuarioListadoForm para presentar la información al actor.

Diagrama de secuencia de lista de usuarios

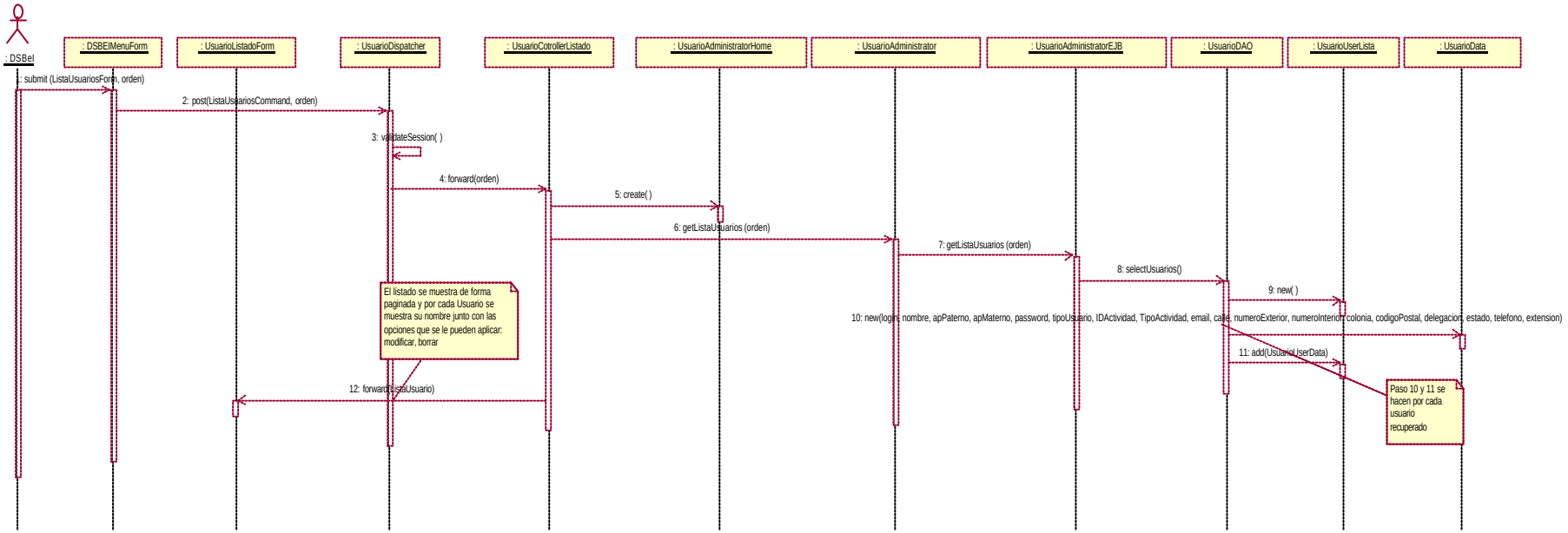
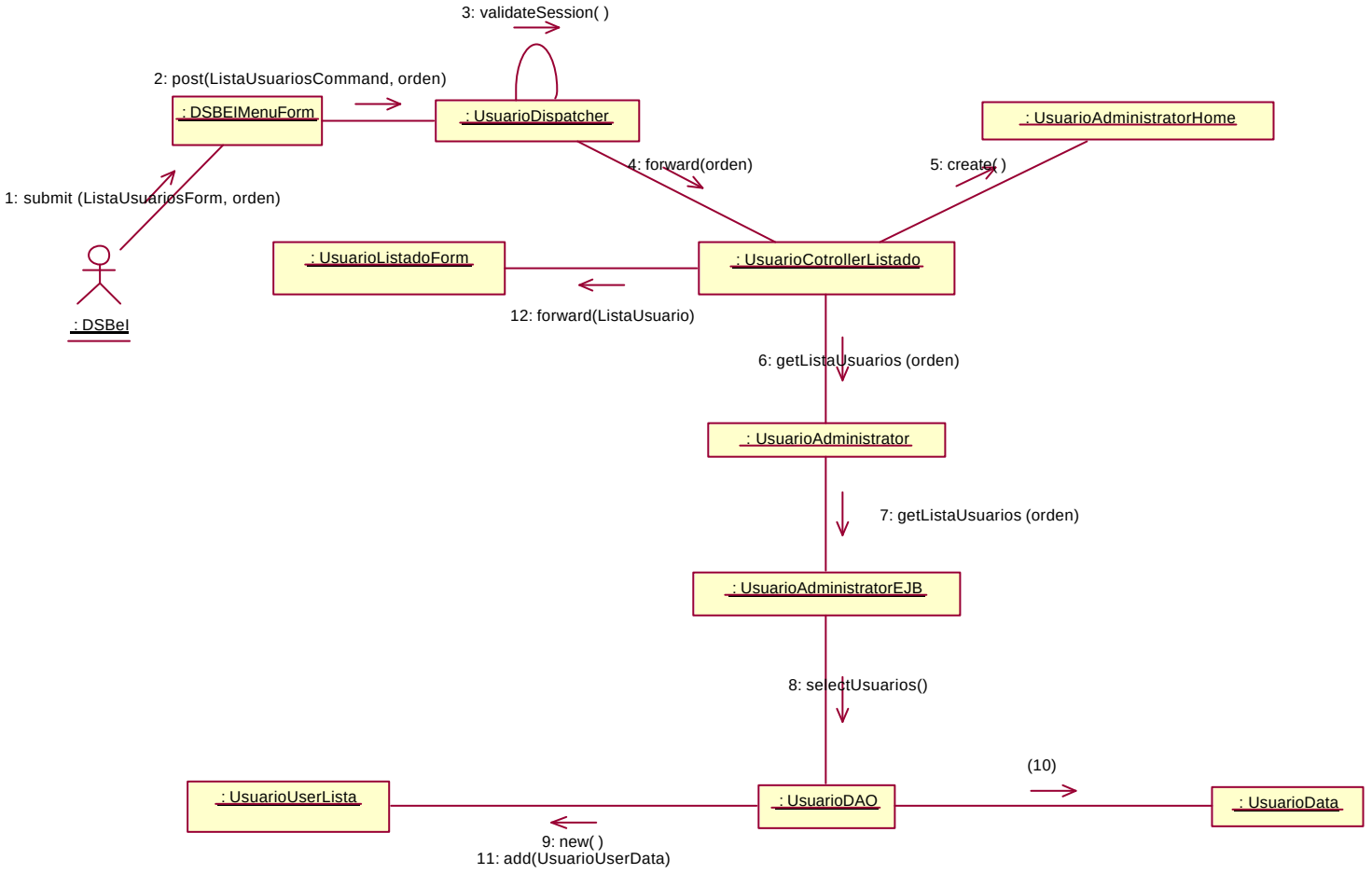


Diagrama de colaboración de lista de usuarios



10: new(login, nombre, apPaterno, apMaterno, password, tipoUsuario, IDActividad, TipoActividad, email, calle, numeroExterior, numeroInterior, colonia, codigoPostal, delegacion, estado, telefono, extension)

4.6.8 Diagrama de secuencia y de colaboración de modifica usuarios

La secuencia para modificar datos de un usuario se divide en dos etapas. La primera muestra los datos del usuario a modificar. En la segunda se envían los datos ya modificados del usuario para que se reflejen en la base de datos.

En la primer etapa, el actor DSBEI emplea la lista de usuarios y manda la solicitud de modificar al usuario mediante el mensaje submit(). La interfaz UsuarioListadoForm reenvía los datos a la clase UsuarioDispatcher. En el mensaje se indica el identificador del usuario a modificar, así como también se especifica que el proceso se encuentra en su primera etapa.

De la clase UsuarioDispatcher se ejecuta el método validateSession donde al ser valida la sesión reenvía dichos datos a la clase UsuarioUserControllerModificaStep1. Ésta última, invoca al método create de la interfaz UsuarioAdministratorHome y así obtener una referencia del tipo de la interfaz remota UsuarioAdministrator y ésta, posteriormente reenviara la solicitud de información del usuario a la clase UsuarioAdministratorEJB quien finalmente empleará el método getUsuarioData. La instancia con los datos del usuario se pasa desde la clase UsuarioUserControllerModificaStep1 a la interfaz UsuarioModificaForm, en donde se presentan los datos al actor y así poder realizar los cambios correspondientes.

En la segunda etapa, el actor DSBEI envía el mensaje submit() con el identificador del usuario a modificar, el parámetro que indica que el proceso se encuentra en la segunda etapa y los datos ya modificados del usuario. La interfaz UsuarioModificaForm es empleada para realizar los cambios a los datos del usuario presentado y reenvía los datos a la clase UsuarioDispatcher.

De la clase UsuarioDispatcher se ejecuta el método validateSession donde al ser valida la sesión reenvía dichos datos a la clase UsuarioControllerModificaStep2. Posteriormente se invoca al constructor de la clase UsuarioData donde se establecerán los atributos de la instancia con los datos modificados del usuario seleccionado. La nueva instancia, se envía como parámetro en el método updateUsuarioData, el cual es ejecutado desde la Interfaz remota UsuarioAdministrator y que posteriormente invocara al mismo método de la clase UsuarioAdministratorEJB. Éste último emplea el método updateUsuario de la clase UsuacioDAO para almacenar la información del objeto de la clase UsuarioUserData en la base de datos. Una vez que se guardan los cambios de la convocatoria, se envía un mensaje de confirmación de la actualización mediante la interfaz UsuarioNotificationForm

Diagrama de secuencia de modifica usuarios

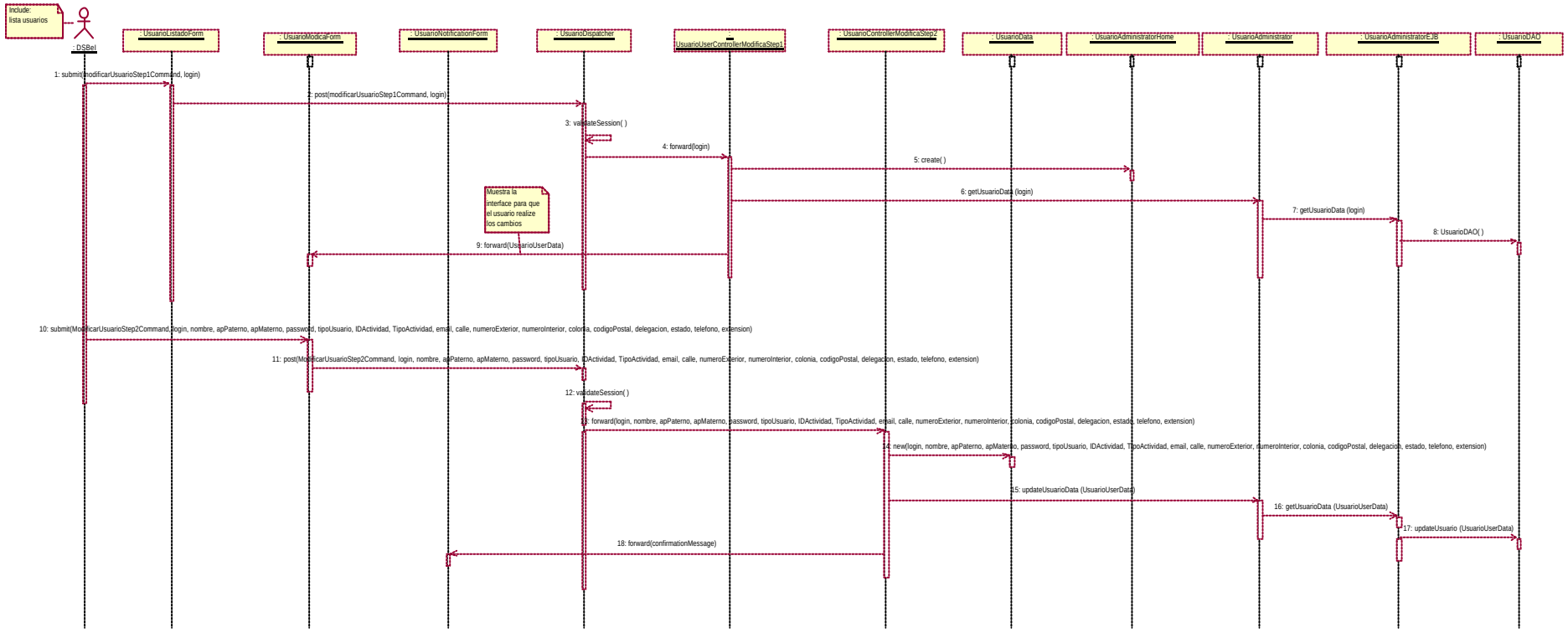
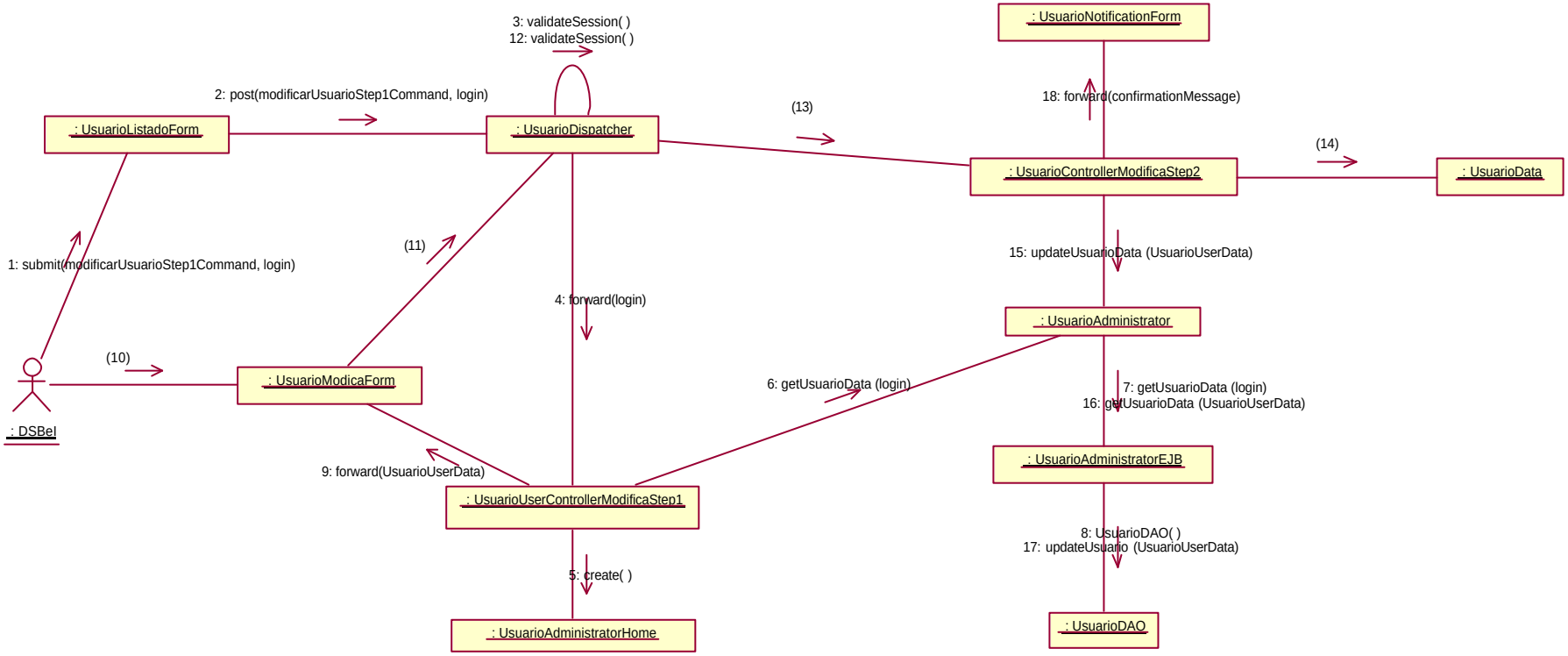


Diagrama de colaboración de modifica usuarios



10: submit(ModificarUsuarioStep2Command, login, nombre, apPaterno, apMaterno, password, tipoUsuario, IDActividad, TipoActividad, email, calle, numeroExterior, numeroInterior, colonia, codigoPostal, delegacion, estado, telefono, extension)
11: post(ModificarUsuarioStep2Command, login, nombre, apPaterno, apMaterno, password, tipoUsuario, IDActividad, TipoActividad, email, calle, numeroExterior, numeroInterior, colonia, codigoPostal, delegacion, estado, telefono, extension)
13: forward(login, nombre, apPaterno, apMaterno, password, tipoUsuario, IDActividad, TipoActividad, email, calle, numeroExterior, numeroInterior, colonia, codigoPostal, delegacion, estado, telefono, extension)
14: new(login, nombre, apPaterno, apMaterno, password, tipoUsuario, IDActividad, TipoActividad, email, calle, numeroExterior, numeroInterior, colonia, codigoPostal, delegacion, estado, telefono, extension)

CAPÍTULO 5. Arquitectura tecnológica

5.1 Sistemas operativos

5.1.1 Unix

Es un sistema operativo multiusuario que incorpora multitarea. Fue desarrollado originalmente por Ken Thompson y Dennis Ritchie en los laboratorios de AT&T Bell en 1969 para su uso en mini computadoras. El sistema operativo Unix tiene diversas variantes y se considera potente, más transportable e independiente de equipos concretos que otros sistemas operativos porque esta escrito en lenguaje C. El Unix esta disponible en varias formas, entre las que se cuenta AIX, una versión de Unix adaptada por IBM (para su uso en estaciones de trabajo basadas en RISC), A/ux (versión gráfica para equipos Apple Macintosh) y Mach (un sistema operativo reescrito, pero esencialmente compatible con Unix, para las computadoras NeXT).

El Unix y sus clones permiten múltiples tareas y múltiples usuarios. Su sistema de archivos proporciona un método sencillo de organizar archivos y permite la protección de archivos. Sin embargo, las instrucciones del Unix no son intuitivas.

Este sistema ofrece una serie de utilidades muy interesantes, como las siguientes:

- Inclusión de compiladores e intérpretes de lenguaje.
- Existencia de programas de interfaz con el usuario, como ventanas, menús, etc.
- Muchas facilidades a la hora de organización de ficheros.
- Facilidades gráficas.
- Programas de edición de textos.

5.1.2 Solaris

Características: Entre las características de Solaris tenemos:

Portabilidad: El software conformado por una Aplicación de Interfaces Binaria (Application Binary Interface, ABI) ejecuta con un Shrink-wrapped (Contracción envuelta) el software en todos los sistemas vendidos con la misma arquitectura del microprocesador. Esto obliga a los desarrolladores de aplicaciones a reducir el costo del desarrollo del software y traer productos al mercado rápidamente y obliga a los usuarios a actualizar el hardware mientras retienen sus aplicaciones de software y minimizan sus costos de conversión.

Escalabilidad: Las aplicaciones se usan con más frecuencia en el sobre tiempo y requiere sistemas más poderosos para soportarlos. Para operar en un ambiente creciente, el software debe ser capaz de ejecutar en un rango de ancho poderoso y debe ser capaz de tomar ventajas del poder adicional que se está procesando.

Interoperatibidad: La computación del ambiente heterogéneo es una realidad hoy. Los usuarios compran de muchos vendedores para implementar la solución que necesitan. La estandarización y una clara interfaz son criterios para un ambiente heterogéneo, permitiendo a los usuarios desarrollar estrategias para comunicarse por medio de su red. El sistema operativo de Solaris puede interoperar con unos sistemas muy populares hoy en el mercado, y aplicaciones que se ejecutan en Unix se pueden comunicar fácilmente.

Compatibilidad: La tecnología de la computación continua avanzando rápidamente, pero necesita permanecer en el ámbito competitivo para minimizar sus costos y maximizar sus ingresos.

5.2 Patrones

A estas alturas todos estamos acostumbrados al concepto de “framework”, no obstante el tamaño y la complejidad de estas estructuras es alimentada por un elemento más pequeño pero de no menos importancia denominado “pattern (patrón)”.

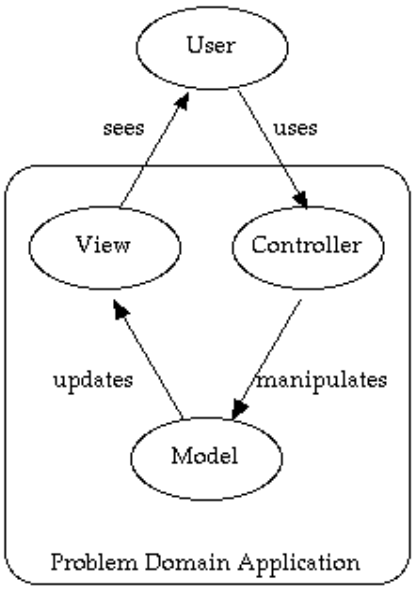
El patrón es la unidad de reutilización en el diseño de objetos, en cierta medida un patrón es a los objetos el equivalente del subprograma a las instrucciones. Permite manipular conceptos de arquitectura expresados en un idioma simple y en consecuencia mejora la comunicación y comprensión entre diseñadores.

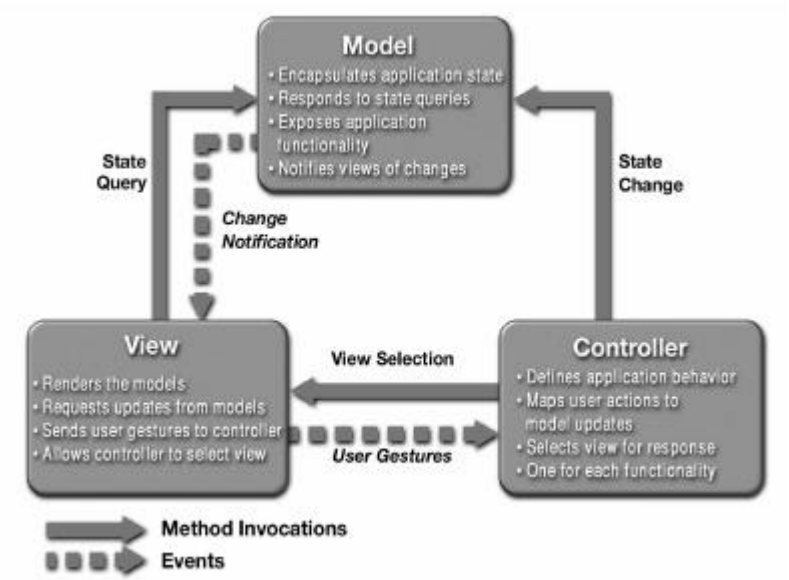
5.2.1 Modelo vista controlador (Model View Controller MVC)

Es el patrón de diseño recomendado para aplicaciones interactivas. El MVC organiza una aplicación interactiva en tres módulos separados: uno para el modelo de aplicación con su representación de datos y la lógica del negocio (Model), el segundo para las vistas que proporcionan la presentación de los datos y la entrada del usuario (View) y el tercero para un controlador que despache peticiones y controle el flujo (Controller).

El MVC separa las tareas de diseño (persistencia de datos y comportamiento, presentación y control), disminuyendo la duplicación de código, centralizando el control y haciendo a la aplicación más fácil de modificar. Así se vuelve más fácil agregar presentaciones de datos para los mismos datos y facilita nuevos tipos de presentación de datos tan pronto la tecnología se avanza.

Los componentes de Modelo y de Vista pueden variar independientemente (excepto en su interfaz), aumentando las capacidades de mantenimiento, de extensión y de prueba. El separar el Controlador de la Vista (comportamiento de la aplicación de la presentación) permite la selección en tiempo de ejecución de la Vista apropiada, basada en el flujo del trabajo, las preferencias del usuario o el estado del Modelo. El separar al Controlador del Modelo (comportamiento de la aplicación de la representación de los datos) permite un mapeo configurable de las acciones de los usuarios sobre el Controlador a funciones de la aplicación en el Modelo.





5.3 Lenguaje de programación

5.3.1 Java

Java 2 (antes llamado Java 1.2 o JDK 1.2) es la tercera versión importante del lenguaje de programación Java.

Existen distintos programas comerciales que permiten desarrollar código Java. La compañía Sun, creadora de Java, distribuye gratuitamente el Java(tm) Development Kit (JDK). Se trata de un conjunto de programas y librerías que permiten desarrollar, compilar y ejecutar programas en Java. Existe también una versión reducida del JDK, denominada JRE (Java Runtime Environment) destinada únicamente a ejecutar código Java (no permite compilar).

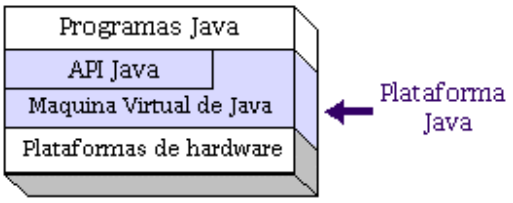
Los IDEs (Integrated Development Environment), tal y como su nombre indica, son entornos de desarrollo integrados. En un mismo programa es posible escribir el código Java, compilarlo y ejecutarlo sin tener que cambiar de aplicación. Estos entornos integrados permiten desarrollar las aplicaciones de forma mucho más rápida, incorporando en muchos casos librerías con componentes ya desarrollados, los cuales se incorporan al proyecto o programa.

5.3.2 La Máquina Virtual de Java (Java Virtual Machine)

La existencia de distintos tipos de procesadores y ordenadores llevó a los ingenieros de Sun a la conclusión de que era muy importante conseguir un software que no dependiera del tipo de procesador utilizado. Se planteó la necesidad de conseguir un código capaz de ejecutarse en cualquier tipo de máquina. Una vez compilado no debería ser necesaria ninguna modificación por el hecho de cambiar de procesador o de ejecutarlo en otra máquina. La clave consistió en desarrollar un código "neutro" el cual estuviera preparado para ser ejecutado sobre una "máquina hipotética o virtual", denominada Java Virtual Machine (JVM). Es esta JVM quien interpreta este código neutro convirtiéndolo a código particular de la CPU utilizada. Se evita tener que realizar un programa diferente para cada CPU o plataforma.

La plataforma Java consta de dos componentes:

- La Máquina Virtual de Java (JVM)
- La Interfaz de Programación de Aplicaciones de Java (API Java)



5.3.3 Tipos de programas en Java

- Applets

Los applets son pequeños programas que se incorporan en una página Web y que por lo tanto, necesitan de un navegador web compatible con Java para poder ejecutarse. A menudo los applets se descargan junto con una página HTML desde un servidor web y se ejecutan en la máquina cliente.

- Aplicaciones

Las aplicaciones son programas standalone de propósito general que normalmente se ejecutan desde la línea de comandos del sistema operativo. Con Java se puede realizar cualquier programa que normalmente se crearía con algún otro lenguaje de programación.

- Servlets

Los servlets son programas que se ejecutan en un servidor web, que fungen como capa intermedia entre una petición proveniente de un navegador Web u otro cliente HTTP, y las bases de datos o aplicaciones del servidor HTTP. Su finalidad es la de:

a) Leer los datos enviados por el usuario. (Los datos pueden ser introducidos en un formulario o en una página Web, pero también podrían provenir de un subprograma como un applet o cualquier otro.)

b) Obtener los datos esperados. (Los servlets pueden comunicarse con una base de datos, ejecutar una llamada RMI o CORBA, invocar a una aplicación existente o calcular la respuesta directamente.

c) Dar formato a los resultados dentro de un documento. (Generalmente se inserta la información en una página HTML).

d) Devolver el documento al cliente. (Los documentos pueden ser enviados en un formato de texto "HTML", binario "imágenes GIF"), o incluso en un formato comprimido como gzip que forme parte de algún otro formato subyacente.

e) Ejecución multihilo. Cada una de las peticiones sobre el Servlet creará una instancia que se ejecutará de manera independiente. A no ser de que le indiquemos lo contrario. El Servlet permanece cargado en memoria por lo que atiende rápidamente las peticiones.

f) Los servlets son más eficientes que los CGI debido a que con los CGI tradicionales, se arranca un nuevo proceso por cada petición HTTP. Con los servlets, la maquina

virtual de Java permanece en ejecución y administra cada petición mediante un ligero subproceso de Java.

En un CGI, si hay n peticiones simultaneas al mismo programa CGI, el código del programa se carga la misma cantidad de veces en memoria, Sin embargo, con los servlets podría haber n subprocesos y sólo una copia del Servlet en memoria.

Un Servlet puede llamar a otro Servlet, incluso a métodos de otros servlets. Esto nos permite que un Servlet realice balanceado de carga entre diferentes servlets. Además, desde un Servlet, podemos redirigir una petición sobre otro Servlet (en la misma máquina o en una máquina remota).

El Servlet puede obtener información acerca de la maquina que ha realizado la petición (IP, puerto, tipo de método de envío: get o post,...).

Uno de los problemas del protocolo HTTP es que es un protocolo sin estado. No existe una relación entre las diferentes peticiones HTTP realizadas por un usuario sobre un servidor, sino que tiene que ser el propio servidor el que mantenga esta sesión. Por ejemplo, por si queremos mantener algún tipo de información del usuario (su identificación, los productos comprados en las diferentes pantallas,...). En los servlets podemos utilizar las sesiones y cookies para poder llevar a acabo esto. La única diferencia es que en las sesiones la información del usuario se almacena en el servidor, mientras que con las cookies la información del usuario se almacena en su propia máquina.

Conexión a Bases de Datos. A través de los servlets podemos establecer conexiones a diferentes tipos de bases de datos. Esta característica acopla perfecta a los servlets dentro de una arquitectura cliente/servidor en 3 capas (cliente - servidor - datos).

Muy por encima podríamos decir que la secuencia de acciones que se producen en un Servlet son las siguientes. La primera vez que realicemos una petición sobre el Servlet se ejecutará un método de inicio, denominado init, en el cual inicializaremos las variables generales. Una vez que nos hemos inicializado nos pondremos a la escucha en espera de peticiones. Cada una de las peticiones que recibamos serán atendidas en hilos de ejecución diferentes, a no ser que indiquemos lo contrario. Dependiendo de como lleguen los datos (mediante post o get) al Servlet se ejecutará un método u otro doPost o doGet. Por último el Servlet tendrá un estado de finalización en el cual eliminará las variables creadas en su inicialización, conexiones a bases de datos (método destroy).

A la hora de codificar lo primero que debemos de saber es que nuestro Servlet deberá de heredar de la clase HttpServlet la cual contendrá todos los métodos necesarios para generar un Servlet. Dicha clase la podemos encontrar en el paquete javax.Servlet.

- Java Server Pages

Los JavaServer Pages permiten la combinación de un lenguaje de marcas, como puede ser HTML, DHTML o XML con trozos de código en Java produciendo contenidos dinámicamente generado a partir de los servlets. Cuando un cliente pide una página al servidor web este la envía al Servlet responsable de su traducción y envía el resultado al cliente. La hojas JSP se ejecutan en el servidor, no en el cliente, por lo que no es posible realizar validaciones de cliente con código JSP.

Los JSP son transportables a otros sistemas operativos y servidores web, no están restringidos a Windows NT/2000 e IIS. Los documentos JSP son automáticamente traducidos a servlets. Sin embargo, es más simple incrustar código de Java en una página HTML que una gran cantidad de instrucciones println que lo generen. Además, al separar la presentación del contenido, se puede hacer que las personas realicen distintas tareas: Los diseñadores web pueden generar HTML mediante diversas herramientas, y dejar espacios para que los desarrolladores de servlets inserten el contenido dinámico.

- Java Beans

El API de JavaBeans hace posible escribir "componentes de software" en el lenguaje Java. Los componentes son elementos reutilizables que pueden incorporarse gráficamente a otros componentes como applets y aplicaciones utilizando herramientas gráficas de desarrollo.

Cada componente ofrece sus características concretas (por ejemplo sus métodos públicos y sus eventos) a los entornos gráficos de desarrollo permitiendo su manipulación visual. Son análogos a otros componentes de algunos entornos visuales, como por ejemplo los controles de Visual Basic.

- JavaScript

JavaScript es distinto del lenguaje de programación Java y se utiliza para generar HTML de forma dinámica o para realizar validaciones del lado del cliente.

Con JavaScript se puede dar respuesta a eventos iniciados por el usuario, tales como la entrada de una forma o algún enlace. Esto sucede sin ningún tipo de transmisión, de tal forma que cuando un usuario escribe algo en una forma, no es necesario que sea transmitido hacia el servidor, verificado y devuelto. Las entradas son verificadas por la aplicación cliente y pueden ser transmitidas después de esto.

5.3.4 Java 2 Enterprise Edition (J2EE)

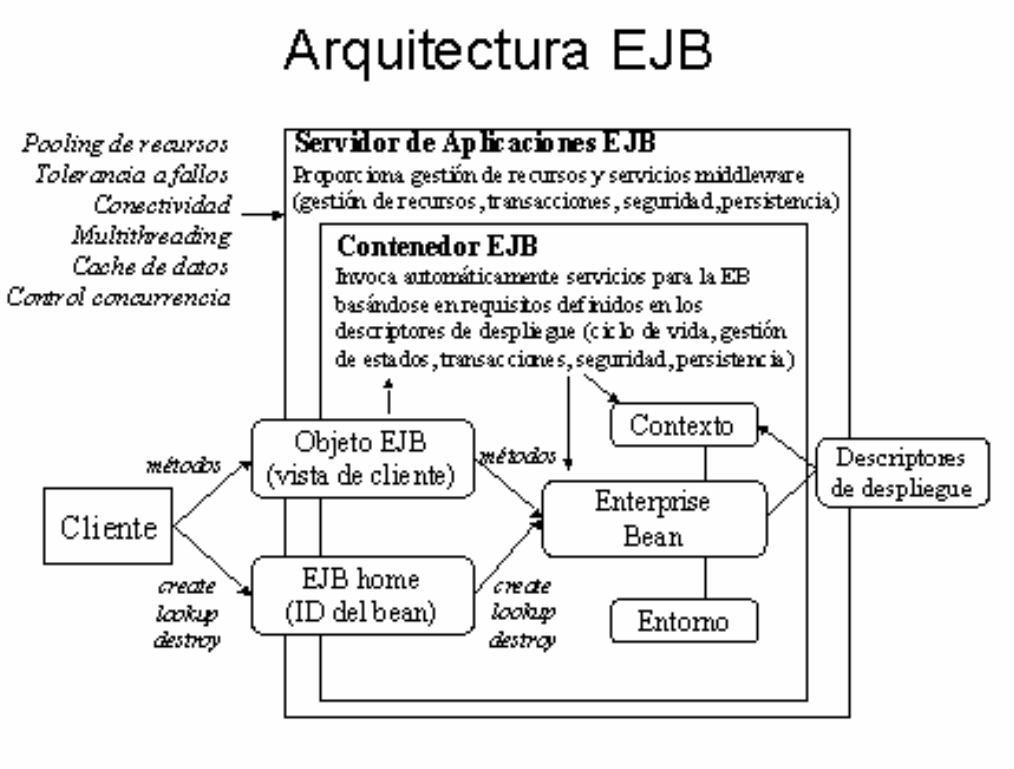
Sun define J2EE como "La plataforma para Soluciones Empresariales". J2EE simplifica las aplicaciones basándose en componentes estandarizados, modulares, pero proporcionando un juego completo de servicios y gestionando automáticamente detalles de comportamiento de las aplicaciones.

J2EE potencia ideas de la plataforma estándar J2SE, como por ejemplo "Write One, Run Anyware", la conectividad a base de datos a través del API JDBC, interacción con CORBA y el modelo de seguridad JAAS y añade, entre otros a los EJB, los Java Servlet y las Java Server Pages (JSP) y como no la tecnología XML, aunque en la ultima versión de la plataforma estándar viene integrada con el JAXP, Java API for XML Processing. Tampoco hay que olvidar otras tecnologías como RMI y RMI-IIOP, JNDI, JTA, JMS, JCA, JAAS, JWSDK, etc.

5.3.5 Enterprise Java Beans (EJB)

Enterprise Java Beans es una arquitectura que define la forma de construir componentes distribuidos del lado del servidor. Esta tecnología garantiza que los componentes programados sean escalables, eficaces y seguros a pesar de lo sencillo de su desarrollo.

Un EJB corre dentro de un container o contenedor en un servidor de aplicaciones. El contenedor lo facilita el fabricante del servidor de aplicaciones y es el interfaz entre el EJB desarrollado y el servidor escogido. Los contenedores proporcionan independencia al EJB del servidor en el que corre.



Estas son las principales características de los EJB:

- Los EJB son componentes distribuidos, esto quiere decir que podemos hablar de componentes que se ejecutan en diferentes servidores, contra diferentes bases de datos, o no, es una decisión del analista-desarrollador.
- Los EJB se ejecutan dentro de un servidor de aplicaciones, estos servidores deben cumplir el estándar fijado por SUN Microsystem INC., y se encargan de gestionar recursos como red, los pool de conexiones, o la gestión de la seguridad. El desarrollador construye los EJB y los servidores de aplicaciones los gestionan.

- Los EJB favorecen la programación modular, la idea de la programación en base a componentes es el sueño de muchos y los EJB lo hacen posible de una manera sencilla. Puedes programar tu componente o puedes comprarlo a un tercero, pero siempre existirá división e independencia entre los diferentes componentes de un sistema. Dentro de un sistema se hace posible añadir o quitar un componente sin que el resto note la diferencia.
- La especificación que define los EJB es pública y en ella trabajan varias empresas. Cualquiera puede bajarse de Internet ese estándar y programar su propio servidor de aplicaciones. Si ese servidor implementa correctamente las interfaces públicas, y cumple la especificación en el se podrá ejecutar correctamente un EJB.
- EJB son Java. Para programar componentes se necesita una clara separación entre interfaces e implementación y Java lo soporta. Java es estable, seguro y multi-hilo, rara vez nos encontramos ante un bloqueo y tiene una de las APIs más ricas en contenido. Java es multiplataforma, esto permite que nuestros componentes se reutilicen en diferentes proyectos independientemente de la plataforma en la que vayan a ejecutarse.
- Los EJB solucionan los problemas de la vida diaria, pueden comunicarse entre ellos y resolver la lógica de negocio, pueden acceder a cualquier base de datos que tenga implementado un driver JDBC, o acceder a terceros sistemas a través de una interfaz SOAP o un servicio web.

5.3.6 La arquitectura EJB

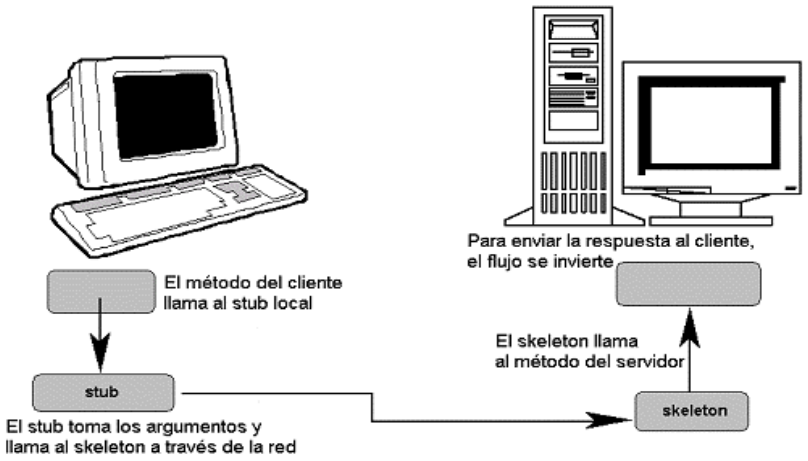
- Componentes distribuidos

Los EJB son componentes de software que se ejecutan en la parte servidora de una aplicación y pueden ser ejecutados en un entorno multicapa distribuido. Si bien podríamos definir componente como pieza de software accesible a través de un interfaz público, en el caso de los EJB, tanto el software como esta interfaz (en realidad dos) deben cumplir el estándar e implementar obligatoriamente ciertos métodos que dicha especificación define, de esta forma los contenedores de EJB pueden gestionar el ciclo de vida de forma uniforme siendo independiente el contenedor que lleve a cabo dicha tarea.

Los EJB son componentes que pueden estar físicamente alejados del cliente por ello se dice que su interfaz pública es remota, Remote Interface. La especificación EJB hace uso de RMI/IIOP para ofrecer esta característica. En RMI/IIOP todo objeto remoto dispone de un interfaz accesible a los clientes, el cliente puede hacer uso de esta interfaz a través del "stub" que es un proxy enviado al cliente.

Cuando el cliente invoca al "stub" este se comunica con el "skeleton", que es le Proxy situado en el lado del servidor. Una vez recibido el mensaje por el "skeleton" este es el encargado de comunicárselo al objeto distribuido, el cual resolverá la operación y dará la respuesta al cliente a través del "skeleton" primero y el "stub" después.

Tanto el "stub" como el "skeleton" son transparentes para el cliente y este siempre tiene la sensación de estar invocando al objeto distribuido directamente.



El problema esta en que este tipo de operaciones es bastante pesada, requiere abrir sockets, transferir objetos a través de la red y transformar estos objetos en datos binarios según el protocolo (en este caso IIOP)

- Middleware implícito o declarativo

La verdadera clave de los componentes distribuidos modernos es que su middleware es implícito, es decir, el código de nuestro software no llama directamente a este software integrado en la parte servidora, sino que define en el descriptor de cada componente, como queremos que sea el contexto en el que se va a ejecutar.

Por ejemplo, un EJB no tiene la necesidad de llamar de forma explícita, en el código, al API de transacciones para comenzar una transacción, sino que podemos definir en un fichero distinto, como queremos que se comporte el EJB.

Para ello aparte de dicho fichero donde se define el comportamiento del EJB, se necesita uno que capture la llamada a los componentes distribuidos y configure la forma en la que estos van a trabajar. En el caso de los EJB el EJB Object es generado por el propio contenedor y es el encargado de interactuar con el contenedor para ofrecer sus servicios. Entre otros ofrece la gestión de transacciones distribuidas, seguridad, gestión del ciclo de vida, persistencia, etc.

Las ventajas, componentes más sencillos, el código se centra en resolver la lógica no en la forma de hacerlo. Además se puede cambiar el comportamiento de los EJB sin tocar su código, sólo modificando el fichero que especifica como debe funcionar, el descriptor.

- Transparencia de situación física

La especificación dice que puede variar el lugar donde se encuentran los EJB, pero de alguna forma se necesita saber como podemos acceder a ellos, esta es la labor del Home Object. El Home Object es el encargado de crear y destruir los EJB Object. Es un objeto generado por el contenedor, de tipo "Factory", que implementa el segundo de los interfaces públicos de un EJB, el Home Interface. De esta forma el contenedor conoce cual es el tipo de EJB, que parámetros debe pasar al EJB Object para crear un nuevo componente o que tipo de búsquedas ha definido un usuario para obtener la referencia a varios componentes.

Es importante señalar que existe un y sólo un EJBHome implementando su correspondiente Home Interface para cada tipo de EJB en cada contenedor. Esto hace posible cachear la referencia en algunos casos.

- Interfaces locales

En la especificación EJB 1.1 no existía este concepto pero muchos servidores de aplicaciones empezaron a implementar mejoras para llamadas dentro de una misma maquina virtual, de esta forma se eliminaba la necesidad de traducir los datos a RMI para hacer una llamada no remota. Las personas que llevan a cabo la especificación se dieron cuenta de esta "necesidad" y crearon los interfaces locales en la versión 2.0, que sólo permiten recibir llamadas de clientes de la misma maquina virtual. Estas llamadas son mucho más rápidas y además los parámetros son pasados por referencia no por valor, con lo que desminuye el consumo de memoria.

- Tipos de componentes

Actualmente el estándar EJB define tres tipos de EJB:

- Session Beans (SB)

Los EJB de sesión se encargan de resolver la lógica de negocio de la aplicación. Se puede decir que cada método contenido en un SB resuelve un caso de uso de la aplicación, ejemplos pueden ser control de usuarios, gestor de precios o control de procesos.
Dentro de la especificación podemos encontrar dos tipos de Sessions Beans, State Less (SLSB) y State Full (SFSB).

- State Less Session Beans (SLSB)

Los SLSB son componentes sin estado, es decir, no guardan ninguna relación entre diferentes llamadas de un mismo cliente. Es mas, entre dos llamadas a un mismo tipo de SLSB es posible que el contenedor direccione al cliente a diferentes instancias del componente. Otro dato esclarecedor de cómo se comportan este tipo de componentes es su relación con el número de usuarios. Para N usuarios hay M instancias siendo M < N en la mayoría de los casos. El valor máximo y mínimo de M es configurable en la mayoría de los servidores. Con estos dos valores se define el tamaño del "pool".

Los pools de objetos han sido utilizados desde los inicios de Java. Se trata de definir un espacio en memoria para varios objetos, de tal forma que en vez de tener que instanciar un objeto cada vez, podemos reutilizar las instancias existentes en el pool. Esto conlleva un consumo de memoria pero ofrece una mayor velocidad. Además hace trabajar menos al recolector de basura, con lo que la mejora de resultados es bastante notable.

Ese es el objetivo de los pools de EJB que utilizan los contenedores, reutilizar las instancias de los EJB en memoria.

El contenedor garantiza que a una instancia de un EJB sólo puede acceder un hilo, con lo cual nunca dentro de un EJB tendremos un problema de concurrencia, el contenedor lo soluciona por nosotros.

- State Full Session Beans (SFSB)

Los SFSB son lo contrario a los SLSB, mantienen el estado, es decir, tienen una conversación "uno a uno" con cada cliente que los invoca. Para mantener esta conversación es necesario hacer uso del objeto Handle.

El objeto Handle es una referencia serializable a un objeto EJB. Guardando esta referencia se puede acceder posteriormente al EJB con el que el cliente había establecido una relación.

Con los SFSB cada cliente tiene asociado un objeto EJB durante una conversación. En caso de que esta referencia se pierda, se puede recuperar gracias al Objeto Handle. Por eso es el cliente quien se tiene que encargar de preservar la instancia de estos objetos si quiere volver a llamar a "su" SFSB.

Para muchos, este tipo de EJB es tachado como verdadera "maquina pesada" y se desaconseja su uso en la mayoría de los casos. Otros mantienen su utilidad por la posibilidad de mantener una sesión distribuida entre varios servidores. En cualquiera de los casos se debe evitar la pasivación de los mismos.

La pasivación es el proceso de serialización a disco de aquellas instancias menos recientemente usadas (LRU, aunque es posible especificar otro política de liberación dependiendo del Servidor de Aplicaciones) cuando el número de instancias requeridas por los clientes es mayor de las que puede soportar el pool de SFSB. Este proceso se puede dar, bien porque se acaba la memoria o porque se ha llegado al límite de instancias máximas configurado en el descriptor.

Al proceso contrario se le llama activación y se produce cuando un cliente llama a un método de su SFSB asociado y este se encuentra pasivado.

Si se prevé que con asiduidad van a ocurrir estos procesos es conveniente intentar buscar otra alternativa, como por ejemplo manteniendo la sesión en el servidor de servlets-JSP.

- Entity Beans (EJB)

Los EJB de entidad están directamente relacionados con los datos de la aplicación, son objetos que mantienen en memoria los datos que maneja la aplicación, ejemplos, Noticia, Foro, Usuario.

Los Entity Beans normalmente mapean las tablas de una base de datos relacional, aunque también es posible que mantengan la persistencia de los datos en ficheros, por ejemplo un XML, o en LDAP. En cualquiera de los casos el objetivo de un Entity Bean es almacenar los datos en memoria desde una fuente persistente y mantener una sincronización total entre el estado de los datos en memoria y la fuente de datos.

Por esta razón se dice que los Entity Bean son los EJB que sobreviven a caídas del sistema, ya que en caso de un fallo del sistema, los datos que hay en memoria están guardados en el dispositivo persistente, con lo cual, cuando se reinicie el servidor se recuperaran sin ningún problema.

Este tipo de EJB abstrae totalmente la capa de persistencia del sistema y funciona como una herramienta de traducción de base de datos relacional a objeto, por ejemplo, se podría cambiar el nombre de una columna de una tabla y el resto de capas no se daría cuenta, ya que a esa columna se accedería a través de un método get/set del Entity Bean.

Quizás las criticas de este tipo de componentes vienen por la lentitud en las funciones de búsqueda, los llamados "finders". Estos métodos devuelven una colección de interfaces remotos que cumplen una serie de condiciones, similares a las instrucciones SQL. Si bien es cierto que es más lento que una simple llamada SQL, no suele ser mucho mayor que estas si se hace un correcto uso de las transacciones y se minimiza el número de llamadas a través de la red. En cualquiera de los casos, no suele ser recomendable hacer este tipo de llamas si sólo se va ha hacer una lectura

Como se ha explicado antes, los EJB se mantienen dentro de un pool en memoria. En caso de los Entity Beans esto significa que tenemos varias filas de la base de datos en memoria, por lo que podemos conseguir accesos muy rápidos a esas filas, por lo tanto es interesante mantener aquellos registros en memoria que creamos que se van a volver a utilizar.

Dentro de los Entity Beans hay dos formas de manejar la persistencia, se puede dejar en manos del programador o en manos del contenedor.

- Bean Managed Persistente (BMP)

Los BMP son los Entity Bean que soportan la persistencia gracias a las instrucciones explicitas del programador, por lo que este deberá introducir las instrucciones necesarias en los métodos definidos por la interfaz EntityBean, ejbLoad, ejbRemove, ejbCreate, etc.

Cada vez se usan menos este tipo de Entity Beans, pero todavía hay algunas operaciones que sólo se pueden realizar gracias a las habilidades de los programadores. En cualquiera de los casos siguen siendo imprescindibles cuando la persistencia no se consigue a través de una base de datos relacional.

- Container Managed Persistente (CMP)

Los CMP soportan la persistencia de forma declarativa o implícita gracias al contenedor, es decir, no es necesario implementar los métodos de la interfaz EntityBean, sólo hay que definir de forma correcta el descriptor para que así el contenedor tenga la información necesaria para gestionar la persistencia contra una base de datos relacional.

A pesar de que en los principios de los EJB, este tipo de Entity Bean no era muy usado, ya que los programadores no se fiaban mucho de los contenedores, actualmente son las más usados e incluso están recomendados por causas de eficiencia con respecto a los BMP.

Crear un CMP es realmente sencillo y la cantidad de funciones que lleva a cabo es realmente impresionante, muchos programadores, la primera vez que ven lo fácil y lo completo de un Entity Bean se quedan boquiabiertos.

- Message-Driven Beans

Son muy parecidos a los beans de sesión pero estos reciben mensajes y no ofrecen respuesta al cliente, son asíncronos. Autorización de compra, Chequeo de tarjeta de crédito son ejemplos de ellos. Nuevos en EJB 2.0

Su único método, onMessage, recibe el tipo de mensaje y con él realiza una serie de operaciones de las cuales el usuario no obtendrá respuesta directa. Cuando este tipo de operaciones se van a realizar en un tiempo indeterminado y no es necesaria la respuesta inmediata al cliente, es donde toman fuerza este tipo de EJB.

- Otras posibilidades

Dentro del mercado de componentes distribuidos los EJB no se encuentran solos, sus más fieles competidores son los objetos COM + de Microsoft y CORBA.

CORBA ha perdido mucho tirón en los últimos años ya que este tipo de componentes son bastante complejos de programar y no acababan de ser portables entre las diferentes versiones comerciales que soportaban el estándar.

Las discusiones entre los tipos de objetos EJB y COM + ha proliferado desde la aparición de plataforma .NET, y dependiendo de las fuentes que se consulten se obtienen respuestas totalmente diferentes. A modo de esquema la siguiente tabla define bastante bien cuáles son las diferencias entre los dos tipos de objetos.

Característica	EJB	COM +
Lenguaje	Sólo Java	Todos
Plataformas (SO)	Todos	Windows 2000
Servidores (Productos Middleware)	30 +	Microsoft
Integración con otros sistemas	RMI/JNI, CORBA, JCA COM TI,	MSMQ, OLE DB
Protocolo	Cualquiera (IIOP)	DCOM
Componentes sin Sesión	Sí	Sí
Componentes con Sesión Componentes persistentes	Sí Sí	No No
Transacciones por método	Sí	No
Equilibrio de carga (Load Balancing)	La mayor parte de vendedores	Sí
Cache de datos (Pools)	Algunos vendedores	No
Componentes de mensajes	Sí	Sí
Precio bajo por transacción	No (Open Source Si)	Sí
Middleware viene con SO	No	Sí
Procesadores por máquina	256 +	16 (32 vía OEMS)
Numero de servidores en un cluster	Teóricamente ilimitado	Teóricamente ilimitado
Herramientas de desarrollo	Varias Opciones	Microsoft .NET Estudio

En cualquiera de los casos, la elección de un tipo de componente u otro al final se basa en la siguiente pregunta ¿SUN o Microsoft?

Mientras SUN intenta defender su plataforma buscando la colaboración de personas de otras empresas (HP, IBM, SAP entre otras 400) para conseguir así que las especificaciones se conviertan en verdaderos estándares de facto, Microsoft apuesta por el marketing como principal defensora de su estrategia junto a la integración de su plataforma dentro de su conocido sistema operativo. Pero es esta razón la que hace ganar la batalla a SUN en lo que a soluciones empresariales se refiere.

No hay que olvidar que hoy en día el dinero en Internet los mueven las grandes empresas y estás cada vez apuesta menos por las soluciones Microsoft. Un dato bastante claro es la dificultad de encontrar un banco on-line implementado con algún producto Microsoft. Seguramente la imposibilidad de utilizar la plataforma .NET y el resto de productos Microsoft en estaciones de trabajo Unix hace que los grandes proyectos sean solamente implementados con J2EE u otras soluciones multiplataforma.

Otro punto a favor de Java y J2EE es la estabilidad contrastada a lo largo de los últimos años de las APIS implementadas por SUN e IBM. Estás han evolucionado progresivamente en la ultima década mientras la plataforma .NET intenta conseguir una arquitectura similar en un espacio mucho más corto de tiempo.

5.3.7 RMI

Tanto RMI (Remote Method Invocation) como Java IDL (Java Interface Definition Language) son herramientas para desarrollar aplicaciones distribuidas. Estas aplicaciones presentan la característica de que una aplicación puede ejecutar funciones y métodos en varios ordenadores distintos. Utilizando una referencia a un objeto que se encuentra en un ordenador remoto, es posible ejecutar métodos de ese objeto desde una aplicación que se ejecuta en un ordenador distinto. RMI y Java IDL proporcionan los mecanismos mediante los cuales los distintos objetos distribuidos se comunican y se transmiten la información. Son por lo tanto tecnologías que permiten la creación y uso de objetos distribuidos, esto es, objetos o programas que interactúan en diferentes plataformas y ordenadores a través de una red. RMI es una solución basada íntegramente en Java. Incorpora métodos para localizar los objetos remotos, comunicarse con ellos e incluso enviar un objeto de Java, "por valor", de un objeto distribuido a otro.

Las aplicaciones RMI normalmente comprenden dos programas por separado. Un servidor y un cliente. Por parte del servidor, la aplicación crea varios objetos remotos y genera una referencia a dichos objetos. La aplicación cliente obtiene una referencia remota de uno o más objetos remotos en el servidor y llama a sus métodos. Es decir, que utilizando una referencia a un objeto que se encuentra en una computadora remota, es posible ejecutar métodos de ese objeto desde una aplicación que se ejecuta en una computadora distinta

RMI-IIOP es para programadores en Java quienes quieren usar RMI. RMI-IIOP provee interacción con otros objetos CORBA implementados en distintos lenguajes, siempre que las interfaces remotas estén originalmente definidas como interfaces JavaRMI. Al igual que CORBA, RMI-IIOP esta basado en estándares abiertos definidos por cientos de usuarios y proveedores miembros de la Object Management Group, además, RMI-IIOP utiliza al igual que CORBA el protocolo Internet Inter-ORB Protocol (IIOP) como capa de transporte.

- Interfaces, objetos y métodos remotos

Una aplicación distribuida construida utilizando RMI de Java, al igual que otras aplicaciones Java, está compuesta por interfaces y clases. Los interfaces definen métodos, mientras que las clases implementan los métodos definidos en los interfaces y, quizás, también definen algunos métodos adicionales. En una aplicación distribuida, se asume que algunas implementaciones residen en diferentes máquinas virtuales. Los objetos que tienen métodos que pueden llamarse por distintas máquinas virtuales son los objetos remotos.

Un objeto se convierte en remoto implementando una interfaz remota, que tenga estas características.

- Una interfaz remota desciende de la interfaz java.rmi.Remote.
- Cada método del interfaz declara que lanza una java.rmi.RemoteException además de cualquier excepción específica de la aplicación.

➢ Carga dinámica de código

Una de las principales y únicas características de RMI es la habilidad de descargar los bytecodes (o simplemente, código) de una clase de un objeto si la clase no está definida en la máquina virtual del recibidor. Los tipos y comportamientos de un objeto, anteriormente sólo disponibles en una sola máquina virtual, ahora pueden ser transmitidos a otra máquina virtual, posiblemente remota. RMI pasa los objetos por su tipo verdadero, por eso el comportamiento de dichos objetos no cambia cuando son enviados a otra máquina virtual. Esto permite que los nuevos tipos sean introducidos en máquinas virtuales remotas y así extender el comportamiento de una aplicación dinámicamente

Java IDL permite la conectividad entre objetos distribuidos utilizando CORBA (Common Object Request Broker Architecture). CORBA es un estándar para la interconexión entre objetos distribuidos. Existen implementaciones de CORBA en varios lenguajes, lo que posibilita comunicar objetos realizados en distintos lenguajes como Java, C/C++, COBOL, etc. Tanto RMI como Java IDL están incluidos en el JDK 1.2 de SUN. En el caso de Java IDL se precisa de una utilidad adicional (llamada idltojava) que genera el código necesario para comunicarse con cualquier implementación CORBA.

5.4 Manejador de base de datos

5.4.1 Acceso a Bases de Datos en Java (Java Data Base Connectivity JDBC)

JDBC (Java DataBase Connectivity) es el estándar de Java para conectarse con bases de datos. Se estima que aproximadamente la mitad del software que se crea en la actualidad incorpora operaciones de lectura y escritura con bases de datos. JDBC está diseñado para ser independiente de la plataforma e incluso de la base de datos sobra la que se desee actuar.

Para conseguir esta independencia, JDBC ofrece un sistema estándar de interconexión con las bases de datos, muy similar al SQL (Structured Query Language). Los distintos vendedores de bases de datos crean los elementos necesarios que actúan como puente entre JDBC y la propia base de datos. La versión JDBC 1.0 forma parte del JDK 1.1. Después de distintas revisiones, actualmente ha aparecido la versión JDBC 2.0, incluida en el JDK 1.2.

A partir de 1996 y ante la repetida insistencia en Internet sobre la necesidad de una base de datos de fuente abierta, un grupo de desarrolladores formó el proyecto PostgreSQL y tomó el control de los fuentes de Postgres95

5.4.2 Postgres

PostgreSQL es una base de datos relacional libre conforme a la norma SQL 92. PostgreSQL es software libre y al igual que muchos proyectos de código abierto, PostgreSQL es desarrollado por un gran número de programadores que utilizan la Intenet como el medio para discutir, acordar y enviar las mejoras que lo han convertido en el líder en su área.

Concretamente está liberado bajo la licencia GLP (General License Public), lo que significa que cualquiera puede disponer de su código fuente, modificarlo a voluntad y redistribuirlo libremente

A diferencia del software comercial, cada 3 ó 5 meses de libera una nueva versión, con nuevas características, con menos errores (bugs) o más acorde a las normas de SQL92.

El código fuente, fundamentalmente en lenguaje C, cuenta con mas de 250,000 líneas; y no cuesta un solo centavo, pues además de ser libre es gratuito y se puede descargar libremente de su página web para multitud de plataformas.

- Ventajas de Postgres

  = Extensible

El código fuente está disponible para todos sin costo. Si su equipo necesita extender o personalizar PostgreSQL de alguna manera, pueden hacerlo con un mínimo esfuerzo, sin costos adicionales. Esto es complementado por la comunidad de profesionales y entusiastas de PostgreSQL alrededor del mundo que también extienden PostgreSQL todos los días.

  = Instalación ilimitada

En postgres no hay costo asociado a la licencia del software, con lo cual se pueden tener modelos de negocios más rentables con instalaciones a gran escala.

Flexibilidad para hacer investigación y desarrollo sin necesidad de incurrir en costos adicionales de licenciamiento.

  = Multiplataforma

PostgreSQL está disponible para Windows y en casi cualquier Unix.

Herramientas gráficas de diseño y administración de bases de datos

Existen varias herramientas gráficas de alta calidad para administrar las bases de datos (pgAdmin, pgAccess) y para hacer diseño de bases de datos (Tora, Data Architect).

Características operacionales :

- Transacciones (Transacctions).
- Disparadores (Triggers).
- Restricciones (Constrains).
- Replication (Replicación).
- Backup & Recovery (Backup y Recuperacion).
- Rules (Reglas).
- Stored Procedures/Functions(Procedimientos Almacenados/Funciones).
- Integridad Referencial.
- Outer Joins.
- Sintaxis ANSI SQL 89, 92 y 98.
- Logging.
- Extensivo y programable.

- Orientado a Objetos
- Características sofisticadas de integridad de datos.
- Tipos de datos y funciones definidos por el usuario.
- Cliente/servidor, entre otros.

= Límites de una base de datos en PostgreSQL:

- Máximo tamaño de una base de datos: ilimitado, solo limitado por la capacidad de almacenamiento del hardware.
- Máximo tamaño de una tabla: hasta 64f Tb (terabytes).
- Máximo tamaño de un campo: 1Gb.
- Máxima cantidad de tuplas o registros: ilimitado.
- Máxima cantidad de columnas en un tabla: hasta 1600.
- Máxima cantidad de índices por tabla: ilimitado.

= Requisitos mínimos para su instalación:

- Memoria principal: 8 MB.
- Espacio libre en disco: 100 MB..
- Sistema Operativo Windows (95/98/NT/ME/2000), Linux
- Protocolo TCP/IP.

5.5 Servidores de aplicaciones

Los servidores de aplicaciones se basan en dar soporte a unos estándares de la industria que marcan como deben ser sus componentes, y que facilitan un desarrollo parecido en diferentes servidores de aplicaciones. Aunque no todos ellos soportan los mismos estándares y alguno de ellos crean especificaciones propias. La base común que debe cumplir es: JSP, JTA, JavaIDL, JavaMail, JNDI, EJB, JMS, servlets, JDBC, SSL, HTTP. Muchos de estos estándares se engloban dentro del J2EE, que esta basado en el Java 2 Standard Edition.

Los servicios que nos ofrecen los servidores de aplicaciones. Entre los mas importantes se encuentran.

- Servicios de seguridad. Los clientes se deben autentificar al servidor, y éste es el responsable de darles acceso a sus diferentes componentes, como puede ser una base de datos. La mayoría de servidores (todos los de la lista anterior) disponen de un mecanismo para incorporar nuevos usuarios y grupos. El control de a que partes del servidor puede acceder un usuario se realiza mediante LDAP.
- Mantenimiento de sesiones. En servidor provee de persistencia a los datos del usuario mediante objetos session que se almacenan en el server pero son propiedad exclusiva de un usuario. Elimina la necesidad de que en nuestro código debamos diferenciar las peticiones de los diferentes usuarios, ya que es el server el responsable de devolverle los datos correctos a cada uno de ellos.
- Balance de cargas. Permite a un grupo de servidores de aplicaciones trabajar como un cluster. Las cargas se balancean entre todos los servidores para no cargar en exceso a ninguno de ellos y permitir escalar aplicaciones fácilmente. Si una de las maquinas falla las peticiones son redirigidas a las otras maquinas en funcionamiento. Algunos de los servidores también replican la información de usuario entre los diferentes servers disponibles para asegurar siempre la persistencia de la sesión.
- Lógica de negocio. La lógica de negocio propia de cada aplicación esta realizada en componentes, que se deben realizar, pero que se les puede asignar mecanismos de seguridad, persistencia, transacción, y comportamientos multithread. Los componentes desarrollados se benefician del mantenimiento de sesiones y de los balanceos de carga de los servidores.
- Generación de HTML. El server puede decodificar una dirección URL que le llegue, y mediante componentes de negocio y consultas a la base de datos generar el HTML o XML a enviar al browser cliente.

- Acceso a datos. Los servidores proveen de objetos de acceso a las bases de datos que se encargan de realizar las conexiones, mantenerlas vivas, gestionar un pool, reconectarlas en caso de error, ahorrando mucho trabajo especifico de base de datos.
- Manejo de transacciones. Se pueden crear transacciones que engloben a varios componentes o a uno de ellos y es el server el responsable de realizar los rollback o commit necesarios.
- Pool de threads. El server es el responsable de tener un número de thread y de instancias de objetos creadas previamente.

Las partes principales de entorno de explotación de un servidor de aplicaciones compatible J2EE son:

- Componentes. Existen cuatro clases de componentes que un servidor compatible J2EE debe soportar.
    - = Aplicaciones clientes. Aplicaciones corrientes, escritas en Java que acceden a los servicios del servidor vía RMI.
    - = Applets.
    - = Servlets y JSP.
    - = Enterprise Jaba Beans (EJB). Estos componentes se ejecutan en el servidor, dentro de un contenedor.
- Containers (contenedor). Son los encargados de dar servicios a los componentes que corren en el servidor de aplicaciones. Existen diferentes contenedores que agrupan a los componentes pos funcionalidades. Los mas importantes son: el web container que contiene servlets y Java Server Pages y el EJB container.
- Resource Manager Driver. Se encarga de proveer a los componentes de conectividad con recursos externos al servidor de aplicaciones. Estas conexiones externas pueden realizarse por JavaMail, Java Database Connectivity (JDBC) y JavaMail.
- Base de datos. El acceso a las bases de datos se debe realizar con JDBC, aunque la mayoría de servidores de aplicaciones dan accesos alternativos a sus bases de datos y deben ser accesibles a todos los componentes, excepto los applets.

La especificación J2EE incluye unos servicios que deben ser implementados obligatoriamente por los servidores de aplicaciones J2EE compatibles. Estos son:

- HTTP. Los componentes, como servlets y Java Server Pages, del servidor deben ser accesibles vía HTTP.
- HTTPS. También debe soportar acceso mediante Secure Socket Layer.
- Java Transaction API (JTA). Da soporte a transacciones.
- JavaIDL. Para la conexión con objetos CORBA mediante JavaIDL ORB.
- JDBC
- Java Naming Directory Interface (JNDI).
- Java Mail. Provee de un API independiente para realizar aplicaciones con acceso a mail.

Estos son los servicios obligatorios, pero hay otros que son recomendables, y que no todos los servidores de aplicaciones compatibles con J2EE los cumplen.

- Remote Method Invocation over Internet Inter-Orb Protocol (RMI-IIOP). Permite el paso de objetos, tanto de referencia como de valor, remotamente.
- Java Message Services (JMS). Provee de un API estándar para el manejo de colas y de comunicación publish and suscribe.

5.5.1 Orion

Orion es un servidor web y de aplicaciones que brinda un ambiente de desarrollo amigable, robusto y escalable. El manejo de transacciones se realiza de una forma sencilla y brinda un gran desempeño en el manejo de la lógica de negocios.

Orion puede interactuar con Java 2 siempre y cuando haya una maquina virtual de java.

Orion puede correr sobre las siguientes plataformas:

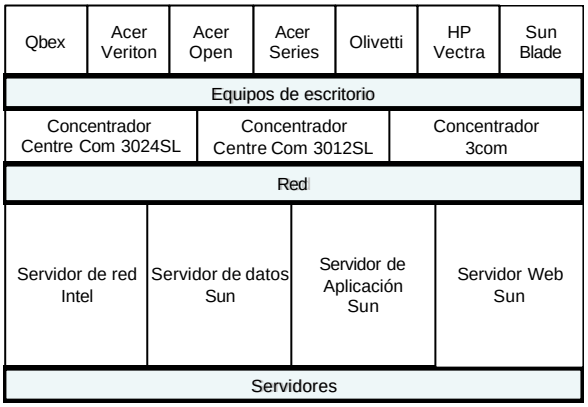
- Windows 95/98/NT/2000/XP
- Sparc Solaris
- Linux
- FreeBSD 4.x
- AS/400
- HP-UX

Muchas características hacen que Orion destaque entre los servidores de aplicaciones. Algunas de esas características son:

- Mayor desempeño. Orion es el Servidor de Aplicaciones basado en J2EE más rápido al realizar las transacciones.
- Orion fue el primer servidor de aplicaciones que se comercializó con soporte completo para J2EE.
- Más económico.

5.6 Hardware

Características de hardware donde la página de Internet del Instituto se ejecutará:



5.6.1 Características Técnicas del Hardware

- Equipos de escritorio

FICHA TECNICA DE PC QBEX			
DESCRIPCION	**MARCA**	**MODELO**	**CARACTERISTICAS**
CPU	QBEX	ATLAS	Pentium IV 1.7 Ghz Memoria RAM 128 MB Disco Duro: 40 GB Tarjeta de Red 1 Unidad Felexible de 3.5" HD 1 Unidad de DVD Puertos USB:4 Puertos Seriales: 2 Puertos Paralelos: 1 Puerto de Juegos: 1 Puertos Minidin: 2 Ranuras de Expansión: 7 Fax Modem
Monitor	QBEX	CX-598S	Monitor 15"
Teclado	QBEX	N/A	Teclado para Windows con soporte ergonómico
Mouse	QBEX	N/A	Mouse con 4 botones

Qbex

FICHA TECNICA DE PC ACER VERITON 5200			
DESCRIPCION	**MARCA**	**MODELO**	**CARACTERISTICAS**
CPU	ACER	Veriton 5200	Pentium IV 1.7 Ghz Memoria RAM 256 MB Disco Duro: 40 GB Tarjeta de Red: Fast Ethernet 10/100 1 Unidad Felexible de 3.5" HD 1 Unidad de DVD Puertos USB:4 Puertos Seriales: 2 Puertos Paralelos: 1 Puerto de Juegos: 1 Puertos Minidin: 2 Ranuras de Expansión: 4
Monitor	ACER	Veriton 5200	Monitor 17"
Teclado	ACER	N/A	Teclado para Windows con soporte ergonómico
Mouse	ACER	N/A	Mouse con 2 botones

Hacer Veriton

FICHA TECNICA DE PC ACER VERITON 5200			
DESCRIPCION	**MARCA**	**MODELO**	**CARACTERISTICAS**
CPU	ACER	Veriton 5200	Pentium IV 1.7 Ghz Memoria RAM 256 MB Disco Duro: 40 GB Tarjeta de Red: Fast Ethernet 10/100 1 Unidad Felexible de 3.5" HD 1 Unidad de DVD Puertos USB:4 Puertos Seriales: 2 Puertos Paralelos: 1 Puerto de Juegos: 1 Puertos Minidin: 2 Ranuras de Expansión: 4
Monitor	ACER	Veriton 5200	Monitor 17"
Teclado	ACER	N/A	Teclado para Windows con soporte ergonómico
Mouse	ACER	N/A	Mouse con 2 botones

Acer Open

FICHA TECNICA DE ACER OPEN			
DESCRIPCION	**MARCA**	**MODELO**	**CARACTERISTICAS**
CPU	ACER	A OPEN	Pentium III 866 Ghz Memoria RAM 128 MB Disco Duro: 2 GB Tarjeta de Red: Fast Ethernet 10/100 1 Unidad Felexible de 3.5" HD Puertos Seriales: 1 Puertos Paralelos: 1 Tarjeta de Sonido de 16 bytes Ranuras de Expansión: 8
Monitor	ACER	ACER VIEW 34T	Monitor 15"
Teclado	ACER	N/A	Teclado para Windows
Mouse	ACER	N/A	Mouse con 2 botones

Acer Series

FICHA TECNICA DE PC ACER Series			
DESCRIPCION	**MARCA**	**MODELO**	**CARACTERISTICAS**
CPU	ACERPOWER	S SERIES	Pentium III 450 Mhz Memoria RAM 64 MB Disco Duro: 8 GB Tarjeta de Red: Fast Ethernet 10/100 Puerto de Juegos: 1 1 Unidad Felexible de 3.5" HD 1 Unidad de DVD Puertos USB:2 Puertos Seriales: 2 Puertos Paralelos: 1 Puertos Minidin: 2 Ranuras de Expansión: 2
Monitor	ACER	1455-1	Monitor 14"
Teclado	ACER	N/A	Teclado para Windows
Mouse	ACER	N/A	Mouse con 2 botones

Olivetti

FICHA TECNICA DE PC OLIVETTI			
DESCRIPCION	**MARCA**	**MODELO**	**CARACTERISTICAS**
CPU	OLIVETTI	M4000 DT -520	Pentium II 266 Mhz Memoria RAM 16 MB Disco Duro: 2.5 GB 1 Unidad Felexible de 3.5" HD 1 Unidad de CD 32X Puertos USB:2 Puertos Seriales: 2 Puertos Paralelos: 1 Puertos Minidin: 2 Ranuras de Expansión: 3
Monitor	OLIVETTI	DSM 14-028/S	Monitor 14"
Teclado	OLIVETTI	N/A	Teclado para Windows
Mouse	OLIVETTI	N/A	Mouse con 2 botones

HP Vectra

FICHA TECNICA DE HP VECTRA			
DESCRIPCION	**MARCA**	**MODELO**	**CARACTERISTICAS**
CPU	HP	Vectra HP 500	Pentium 100 Ghz Memoria RAM 64 MB Disco Duro: 4 GB Tarjeta de Red: Fast Ethernet 10/100 Unidad de CD 4X 1 Unidad Felexible de 3.5" HD Puertos Seriales: 1 Puertos Paralelos: 1 Tarjeta de Sonido de 16 bytes Ranuras de Expansión: 8
Monitor	HP	DS2813	Monitor 15"
Teclado	HP	N/A	Teclado para Windows
Mouse	HP	N/A	Mouse con 2 botones

SUN Blade

FICHA TECNICA DE SUN BLADE 100			
DESCRIPCION	**MARCA**	**MODELO**	**CARACTERISTICAS**
CPU	SUN	SUN BLADE 100	UltrasparcIII 500 Mhz Ultra - 128 cache Memoria RAM 128 MB Disco Duro: 20 GB Tarjeta de Red: Fast Ethernet 10/100 1 Unidad de CD 32X 1 Unidad Flexible de 3.5" HD Puertos USB:4 Puertos Seriales: 1 Puertos Paralelos: 1 Ranuras de Expansión: 3 Tarjeta de Video 24 bit
Monitor		N/A	Monitor 17"
Teclado		N/A	Teclado

- Red

Concentrador CentreCom 3024SL

FICHA TECNICA DE CONCENTRADOR CENTRECOM			
DESCRIPCION	**MARCA**	**MODELO**	**CARACTERISTICAS**
CONCENTRADOR	CENTRECOM	3024SL	IEEE 802.3 puerto AUI puerto BNC 24 puertos 10 base _T(RJ45) indicador luminosos recepcion indicadores luminosos en diagnostico y estado MDI/MDI-X switch

Concentrador CentreCom 3012SL

FICHA TECNICA DE CONCENTRADOR CENTRECOM			
DESCRIPCION	**MARCA**	**MODELO**	**CARACTERISTICAS**
CONCENTRADOR	CENTRECOM	3012SL	IEEE 802.3 puerto AUI puerto BNC 12 puertos 10 base _T(RJ45) indicador luminosos recepcion indicadores luminosos en diagnostico y estado MDI/MDI-X switch

Concentrador 3Com

FICHA TECNICA DE CONCENTRADOR 3com			
DESCRIPCION	**MARCA**	**MODELO**	**CARACTERISTICAS**
CONCENTRADOR	3COM	ETHERNET HUB 8/TPO	8 puertos ethernet 10 Mbps 1 puerto de descarga BNC 8 conectores RJ-45, 10 Base- T Indicadores de luz: alerta, iluminacion, Colision indicador de luz de puertos. Estado de la Red

- Seridores

Servidor de red

FICHA TECNICA DE SERVIDOR ACER ALTOS 1200			
DESCRIPCION	**MARCA**	**MODELO**	**CARACTERISTICAS**
SERVIDOR	ACER	ALTOS 1200	Pentium III 1.1 Ghz Memoria RAM 128 MB Disco Duro: 36 GB SCSI 2 Tarjeta de Red: Fast Ethernet 10/100 1 Unidad Felexible de 3.5" HD 1 Unidad de CD 48X 1 Unidad de Respaldo DAT Disco SCI Puertos USB:3 Puertos Seriales: 2 Puertos Paralelos: 1 Ranuras de Expansión: 6
Teclado	ACER	N/A	Teclado
Mouse	ACER	N/A	Mouse con 2 botones
SOFTWARE	MICROSOFT	WINDOWS 2000 SERVER	

Servidor de datos SUN

FICHA TECNICA DE SERVIDOR DE DATOS			
DESCRIPCION	**MARCA**	**MODELO**	**CARACTERISTICAS**
Procesadores Velocidad de procesador	SPARC	UltraSPARCIII	2 Procesadores 900- Mhz RED 1 Gb ethernet y 10/100 Base T Ethernet Serial 2 RS-232C/RS-423 i/o 40 Mb/sec Ultra SCSI Expansion 9 ranuras PCI. Disco Duro 73 GB 1 unidad de respaldo 20 GB DDS-4 interno SCSI. SOPORTE PARA dvd INTERNO
Monitor			color 17"
Adaptador de Video			pgx64

Servidor de aplicación SUN

FICHA TECNICA DE SERVIDOR DE APLICACIÓN			
DESCRIPCION	**MARCA**	**MODELO**	**CARACTERISTICAS**
Procesadores Velocidad de procesador	SPARC	UltraSPARCIII	2 Procesadores 900- Mhz RedEthernet/Fast Ethernet (10-base T y 100 bas Serial 2RS-232C/RS-423 1 paraleloCentronics DB25 40 Mb/sec Ultra SCSI Ranuras de expansion: 4 PCI 64 Bits Disco Duro 73 GB interno SCSI. SOPORTE PARA dvd INTERNO
Monitor			color 17"
Adaptador de Video			pgx64

Servidor web SUN

FICHA TECNICA DE SERVIDOR DE WEB			
DESCRIPCION	**MARCA**	**MODELO**	**CARACTERISTICAS**
Procesadores Velocidad de procesador	**SPARC**	**ULTRASPARC 2I**	1 Procesador 650- Mhz Ethernet/Fast Ethernet (10-base T y 100 base T) 2 Seriales RS-232C/RS-423 1 paraleloCentronics DB25 80 Mb/sec Ultra SCSI-2 Expansion 1 PCI 32 Bits Disco Duro 36 Gb 1 Unidad de DVDInterno estandar 8x

CAPÍTULO 6. Desarrollo del sitio web del INACIPE

6.1 Implantación de la base de datos

En esta sección se hablará acerca de la construcción de la base de datos a partir del diagrama de entidad relación, siguiendo 3FN (Tercera Forma Normal), generando la estructura sobre la cual se construirá el sistema del sitio web del Instituto. En la construcción de la base se realizaron scripts para creación de tablas, generación de constraints y funciones propias del DBMS (PosgresSQL). Ejemplo:

```
Script creafunciones.txt
CREATE FUNCTION WebObtenClaveArticuloIterCrimin () RETURNS varchar
AS '
   DECLARE
      intUltimaClave integer;
      strUltimaClave varchar;
           strClaveArticulo varchar;
   BEGIN

      --Busca la nueva clave de la Publicacion
      SELECT max(int8(IDARTICULO)) into intUltimaClave FROM WEBARTICULODEITERCRIMINIS;
      IF intUltimaClave IS NULL
      THEN
         intUltimaClave := 1;
      ELSE
         intUltimaClave := intUltimaClave + 1;
      END IF;

      strUltimaClave := text(intUltimaClave);
      strClaveArticulo := substring(''00000000'', length(strUltimaClave) + 1) || strUltimaClave;
      RETURN strClaveArticulo;

   END;
' LANGUAGE 'plpgsql';

CREATE FUNCTION WebObtenClaveAutorIterCriminis () RETURNS varchar
AS '
   DECLARE

      intUltimaClave integer;
```

```
    strUltimaClave varchar;
        strClaveAutor varchar;
  BEGIN

    --Busca la nueva clave de la Publicacion
    SELECT max(int8(IDAUTOR)) into intUltimaClave FROM WEBAUTORITERCRIMINIS;
    IF intUltimaClave IS NULL
    THEN
      intUltimaClave := 1;
    ELSE
      intUltimaClave := intUltimaClave + 1;
    END IF;

    strUltimaClave := text(intUltimaClave);
    strClaveAutor := substring(''00000000'', length(strUltimaClave) + 1) || strUltimaClave;
    RETURN strClaveAutor;

  END;
' LANGUAGE 'plpgsql';

CREATE FUNCTION WebObtenClaveAutorPublicacion () RETURNS varchar
AS '
  DECLARE

    intUltimaClave integer;
    strUltimaClave varchar;
        strClaveAutorP varchar;
  BEGIN

    --Busca la nueva clave de la Publicacion
    SELECT max(int8(IDAUTOR)) into intUltimaClave FROM WEBAUTORPUBLICACION;
    IF intUltimaClave IS NULL
    THEN
      intUltimaClave := 1;
    ELSE
      intUltimaClave := intUltimaClave + 1;
    END IF;

    strUltimaClave := text(intUltimaClave);
    strClaveAutorP := substring(''00000000'', length(strUltimaClave) + 1) || strUltimaClave;
    RETURN strClaveAutorP;

  END;
```

```
' LANGUAGE 'plpgsql';

CREATE FUNCTION WebObtenClaveConvocatoria () RETURNS varchar
AS '
   DECLARE
      intUltimaClave integer;
      strUltimaClave varchar;
           strClaveConvocatoria varchar;
   BEGIN

      --Busca la nueva clave de la convocatoria
      SELECT max(int8(IDCONVOCATORIA)) into intUltimaClave FROM WEBCONVOCATORIA;
      IF intUltimaClave IS NULL
      THEN
         intUltimaClave := 1;
      ELSE
         intUltimaClave := intUltimaClave + 1;
      END IF;

      strUltimaClave := text(intUltimaClave);
      strClaveConvocatoria := substring(''00000000'', length(strUltimaClave) + 1) || strUltimaClave;

      RETURN strClaveConvocatoria;
   END; ' LANGUAGE 'plpgsql';

CREATE FUNCTION WebObtenClaveEvento () RETURNS varchar
AS '
   DECLARE
      intUltimaClave integer;
      strUltimaClave varchar;
           strClaveEvento varchar;
   BEGIN

      --Busca la nueva clave del evento
      SELECT max(int8(IDEVENTO)) into intUltimaClave FROM WEBEVENTO;
      IF intUltimaClave IS NULL
      THEN
         intUltimaClave := 1;
      ELSE
         intUltimaClave := intUltimaClave + 1;
      END IF;

      strUltimaClave := text(intUltimaClave);
```

```
    strClaveEvento := substring(''00000000'', length(strUltimaClave) + 1) || strUltimaClave;

    RETURN strClaveEvento;
  END;
' LANGUAGE 'plpgsql';

CREATE FUNCTION WebObtenClavePublicacion () RETURNS varchar
AS '
  DECLARE
    intUltimaClave integer;
    strUltimaClave varchar;
        strClavePublicacion varchar;
  BEGIN

    --Busca la nueva clave de la Publicacion
    SELECT max(int8(IDPUBLICACION)) into intUltimaClave FROM WEBPUBLICACION;
    IF intUltimaClave IS NULL
    THEN
      intUltimaClave := 1;
    ELSE
      intUltimaClave := intUltimaClave + 1;
    END IF;

    strUltimaClave := text(intUltimaClave);
    strClavePublicacion := substring(''00000000'', length(strUltimaClave) + 1) || strUltimaClave;

    RETURN strClavePublicacion;
  END; ' LANGUAGE 'plpgsql';

CREATE FUNCTION WebObtenClaveQueja () RETURNS varchar
AS '
  DECLARE
    intUltimaClave integer;
    strUltimaClave varchar;
        strClaveQueja varchar;
  BEGIN

    --Busca la nueva clave de la queja
    SELECT max(int8(IDQUEJA)) into intUltimaClave FROM WEBQUEJA;
    IF intUltimaClave IS NULL
    THEN
      intUltimaClave := 1;
    ELSE
```

```
        intUltimaClave := intUltimaClave + 1;
      END IF;
      strUltimaClave := text(intUltimaClave);
      strClaveQueja := substring(''00000000'', length(strUltimaClave) + 1) || strUltimaClave;

      RETURN strClaveQueja;
   END;
' LANGUAGE 'plpgsql';

CREATE FUNCTION WebObtenClaveTesis () RETURNS varchar
AS '
   DECLARE
      intUltimaClave integer;
      strUltimaClave varchar;
          strClaveTesis varchar;
   BEGIN

      --Busca la nueva clave de la TESIS
      SELECT max(int8(IDTESIS)) into intUltimaClave FROM WEBTESIS;
      IF intUltimaClave IS NULL
      THEN
         intUltimaClave := 1;
      ELSE
         intUltimaClave := intUltimaClave + 1;
      END IF;

      strUltimaClave := text(intUltimaClave);
      strClaveTesis := substring(''00000000'', length(strUltimaClave) + 1) || strUltimaClave;

      RETURN strClaveTesis;
   END;
' LANGUAGE 'plpgsql';

CREATE FUNCTION importaAcervo (varchar) RETURNS INTEGER AS '
   DECLARE
          acervo RECORD;
      intConta integer;
      strConta integer;
      strClave varchar;
      strSQL varchar;
      pipe varchar;

BEGIN
```

```
            CREATE TEMP TABLE WEBACERVOTEMP (
               tipo integer,
               ficha varchar(10),
               NumClas varchar(50),
               Autor varchar(255),
               Titulo varchar(400),
               Editorial varchar(255),
               Nota varchar(700),
               Tema varchar(500)
            );

        pipe := "|";
            DELETE from WEBACERVO;
              EXECUTE "copy WEBACERVOTEMP from " || quote_literal($1) ||  " using delimiters " ||
quote_literal(pipe);
            intConta :=0;
             FOR acervo IN Select * from WEBACERVOTEMP WHERE titulo <> ""and tipo <= 4 and tipo >=
1 LOOP
               intConta := intConta + 1;
               strConta := text(intConta);
               strClave := substring("0000000000", length(strConta) + 1) || strConta;

               INSERT INTO WEBACERVO VALUES (strClave, acervo.tipo, acervo.titulo, acervo.autor,
acervo.numClas, acervo.editorial, acervo.nota, acervo.tema);
            END LOOP;

            DROP TABLE WEBACERVOTEMP;
            RETURN intConta;

END; 'LANGUAGE 'plpgsql';
```

6.2 Desarrollo de interfaces

Se crearon las pantallas web de la página de Internet del Instituto utilizando JSP's y páginas HTML para la capa del cliente con los elementos definidos en el diseño de interfaces. Este debe corresponder con el diseño del sistema, por ejemplo:

```
<%@ page errorPage="/error.jsp?msg=4" %>
<%@ page import="java.util.*" %>
<%@ page import="java.net.URLEncoder" %>
<%
String ROL = "CNV";
session = request.getSession();
String user = (String)session.getValue("user");
String roles = (String)session.getValue("roles");
if(user == null) {
  response.sendRedirect("/error.jsp?msg=2");
}
if(roles.indexOf(ROL) == -1) {
  response.sendRedirect("/error.jsp?msg=1");
}
if(roles.indexOf(ROL) != -1 && user != null) {
String xtitulo = request.getParameter("titulo");
if(xtitulo == null)
  xtitulo = "";
String xfecha1 = request.getParameter("fecha1");
if(xfecha1 == null)
  xfecha1 = "";
String xfecha2 = request.getParameter("fecha2");
if(xfecha2 == null)
  xfecha2 = "";
String xemail = request.getParameter("email");
if(xemail == null)
  xemail = "";
String tipoevento = request.getParameter("tipo");
if(tipoevento == null)
  tipoevento = "1";
String xnota = request.getParameter("nota");
if(xnota == null)
  xnota = "";
String eliminar = request.getParameter("adjuntos");
String[] adjuntos = null;
HttpSession sesion= request.getSession();
```

```
String[] newfiles = request.getParameterValues("userfile");
String[] temp = (String[])sesion.getAttribute("archivos");
adjuntos = temp;
if(temp != null && newfiles != null) {
  adjuntos = new String[temp.length + newfiles.length];
  int j = 0;
  for(int i=0; i<temp.length; i++)
    adjuntos[j++] = temp[i];
  for(int i=0; i<newfiles.length; i++)
    adjuntos[j++] = newfiles[i];
  sesion.setAttribute("archivos",adjuntos);
}
if(temp == null && newfiles != null) {
  adjuntos = newfiles;
  sesion.setAttribute("archivos",adjuntos);
}
if(eliminar != null) {
  temp = new String[adjuntos.length - 1];
  int j=0;

  for(int i=0; i<adjuntos.length; i++) {
    if(!adjuntos[i].equals(eliminar))
      temp[j++] = adjuntos[i];
  }
  adjuntos = temp;
  sesion.setAttribute("archivos",adjuntos);
}
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Instituto Nacional de Ciencias Penales</title>
<meta HTTP-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<SCRIPT language=JavaScript src="../js/AnchorPosition.js"></SCRIPT>

<SCRIPT language=JavaScript src="../js/PopupWindow.js"></SCRIPT>

<SCRIPT language=JavaScript src="../js/date.js"></SCRIPT>

<SCRIPT language=JavaScript src="../js/CalendarPopup.js"></SCRIPT>

<SCRIPT language=JavaScript>document.write(CalendarPopup_getStyles());</SCRIPT>
```

```
<style type="text/css">
<!--
.justy {
	font-family: Verdana, Arial, Helvetica, sans-serif;
	font-size: 11px;
	color: #FFFFFF;
	text-align: justify;
}
.justback {
	font-family: Verdana, Arial, Helvetica, sans-serif;
	font-size: 10px;
	color: #333333;
	text-align: justify;
}
.mininota {
	font-family: Verdana, Arial, Helvetica, sans-serif;
	font-size: 10px;
	color: #333333;
	text-align: left;
}
.tituloscontent {
	font-family: Verdana, Arial, Helvetica, sans-serif;
	font-size: 10px;
	color: #FFFFFF;
	background-color: #006699;
}
.links {
	font-family: Verdana, Arial, Helvetica, sans-serif;
	font-size: 10px;
	color: #333333;
	text-align: center;
}
.centrado {
	font-family: Verdana, Arial, Helvetica, sans-serif;
	font-size: 9px;
	text-align: center;
	color: #622229;
}
.titlesdialog {
	font-family: Verdana, Arial, Helvetica, sans-serif;
	font-size: 12px;
	color: #FFFFFF;
	text-align: left;
```

```
            font-weight: bold;
}
-->
</style>
<script language="JavaScript" type="text/JavaScript">
<!--

function tipoid(idoption) {
  var i;
  var n = document.form1.tconvocatoria.options.length;
  for(i=0; i<n; i++) {
    if(document.form1.tconvocatoria.options[i].value == idoption) {
      document.form1.tconvocatoria.options[i].selected = true;
    }
  }
}

function agregar(){
  document.adjuntos.titulo.value = document.form1.titulo.value;
  document.adjuntos.fecha1.value = document.form1.date4.value;
  document.adjuntos.fecha2.value = document.form1.date5.value;
  document.adjuntos.email.value = document.form1.email.value;
  document.adjuntos.nota.value = document.form1.nota.value;
  document.adjuntos.tipo.value = document.form1.tconvocatoria.value;
  document.adjuntos.submit();
}

function MM_preloadImages() { //v3.0
 var d=document; if(d.images){ if(!d.MM_p) d.MM_p=new Array();
   var i,j=d.MM_p.length,a=MM_preloadImages.arguments; for(i=0; i<a.length; i++)
   if (a[i].indexOf("#")!=0){ d.MM_p[j]=new Image; d.MM_p[j++].src=a[i];}}
}


function isEmailAddr(email)
{
  var result = false
  var theStr = new String(email)
  var index = theStr.indexOf("@");
  if (index > 0)
  {
    var pindex = theStr .indexOf(".",index);
    if ((pindex > index+1) && (theStr.length > pindex+1))
```

```
            result = true;
  }
   return result;
}

function valida(){

var examinar =true;
var nota = true;
var email = true;
var tconvocatoria = true;
var fec = true;
var tit = true;

tituloal =fechaal = tconvocatoriaal = emailal = notaal = examinaral="";

 a=document.forms[0].date4.value;
 b=document.forms[0].date5.value;

 dia=a.split("/")[0];
 mes=a.split("/")[1];
 anyo=a.split("/")[2];

 dia2=b.split("/")[0];
 mes2=b.split("/")[1];
 anyo2=b.split("/")[2];

fecha1 = new Date(anyo,mes,dia);
fecha2 = new Date(anyo2,mes2,dia2);

examinar = document.form1.examinar.value;
if(examinar.length <= 0 )
 examinar = false;
//validacion notas

if(!examinar){

examinaral= "No seleccionó  la imagen\n";
document.form1.examinar.focus();
}

//validacion notas
```

```
nota = document.form1.nota.value;
if(nota.length <= 0 )
 nota = false;
//validacion notas

if(!nota){

notaal= "No se llenó la sección de notas\n";
document.form1.nota.focus();
}

//Validacion de mail

var str=document.form1.email.value
email = true;
if (str == "")
 {
  email=false;
 }

 if (!isEmailAddr(str))
 {
  email = false;
 }

 if (str.length < 3)
 {
  email = false;
 }

if(!email){
        emailal = "El email es incorrecto\n";
        document.form1.email.focus();
        }

//Validacion de mail

if (document.form1.tconvocatoria.options[0].selected ){
        tconvocatoria=false;
        }
if(!tconvocatoria){
        tconvocatoriaal = "Debes seleccionar una opción en categoría\n";
        document.form1.tconvocatoria.focus();
```

```
            }


if( eval(fecha1 > fecha2) || a.length == 0 || b.length == 0){
            fechaal = "Las fechas no son válidas (Fecha inicial < Fecha final)\n";
            document.form1.date4.focus();
            fec = false;
}

if(document.forms[0].titulo.value.length == 0){
            tituloal = "El campo de título debe ser llenado\n";
            document.form1.titulo.focus();
            tit = false;
 }

 if(tituloal != "" || fechaal != "" || tconvocatoriaal != "" || emailal != "" || notaal != "" || examinaral != "" )
  alert(tituloal+ fechaal  + tconvocatoriaal  + emailal  + notaal + examinaral);

if( nota &&email &&tconvocatoria && fec &&tit && examinar)
 document.form1.submit();
}

function examinar () {
document.forma.buscar.click();
document.form1.examinar.value = document.forma.buscar.value;
}

function examinaradj () {
document.forma.buscar.click();
document.form1.examinaradj.value = document.forma.buscar.value;
}

function eliminar() {
document.form1.examinar.value = "";
}

function eliminaradj() {
  document.form2.submit();
}

//-->
</script>
</head>
```

```
<body background="../images/backs/backmain.gif" text="#666666" link="#666666" vlink="#666666" alink="#666666" leftmargin="0" topmargin="0" marginwidth="0" marginheight="0" onLoad="javascript:tipoid('<%=tipoevento%>');">
<table width="100%" border="0" cellpadding="0" cellspacing="0">
  <tr>
   <td background="../images/backs/backtitles.jpg" bgcolor="622229"> <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" codebase="HTTP://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=6,0,29,0" width="200" height="35">
      <param name="movie" value="../flash/titulosflash/titulosinf-CONVOCATORIAS.swf">
      <param name="quality" value="high">
      <embed src="../flash/titulosflash/titulosinf-CONVOCATORIAS.swf" quality="high" pluginspage="HTTP://www.macromedia.com/go/getflashplayer" type="application/x-shockwave-flash" width="200" height="35"></embed></object></td>
    <td align="right" valign="top" background="../images/backs/backtitles.jpg" bgcolor="622229"><img src="../images/backs/spacer.gif" width="15" height="8"></td>
  </tr>
</table>
<table width="100%" border="0" cellpadding="5" cellspacing="2">
  <tr>
   <td align="center" valign="top">
    <table width="100%" border="0" cellpadding="0" cellspacing="0">
     <tr>
      <td height="2" class="justy">
       <div align="right"></div></td>
     </tr>
     <tr valign="top">
      <td bgcolor="#FFFFFF" class="justback"> <table width="100%" border="0" cellspacing="0" cellpadding="0">
         <tr>
          <td height="481" align="center" valign="middle"> <p class="justback"><span class="justback">
            </span></p>
           <table width="500" border="0" cellspacing="0" cellpadding="0">
            <tr>
             <td colspan="2" align="center" valign="middle"><img src="../images/backs/spacer.gif" width="10" height="20"></td>
            </tr>
            <tr>
             <td height="16" valign="middle" background="../images/backs/backtopdialog.jpg" bgcolor="622229" class="titlesdialog"><img src="../images/backs/spacer.gif" width="15" height="20" align="absmiddle">Publicar
              Convocatorias </td>
             <td align="right" background="../images/backs/backtopdialog.jpg" bgcolor="622229"><img src="../images/backs/backcornerdialog.jpg" width="76" height="20"></td>
            </tr>
```

```
<tr bgcolor="eeeeee">
<td height="361" colspan="2"> <form name="form1" method="post" action="/ConvocatoriaControllerPublica" enctype="multipart/form-data">
<input type=hidden name="seleccion" value="publica">
<table width="100%" border="0" cellspacing="0" cellpadding="3">
<tr>
<td width="20%" align="right"><font color="#000000"><img src="../images/backs/spacer.gif" width="15" height="15" align="absmiddle"><span class="mininota"><strong>T&iacute;tulo:</strong></span></font></td>
<td width="85%" height="25"> <input name="titulo" id="titulo" type="text" class="mininota" style="height=18" value="<%=xtitulo%>" size="30">
</td>
</tr>
<tr>
<td align="right" class="centrado"><div align="right"><font color="#000000"><strong>Fecha
inicio:</strong></font></div></td>
<td>
<!--<input name="textfield3" type="text" class= "mininota" style="height=18" size="30">-->
<SCRIPT language=JavaScript>
var cal4 = new CalendarPopup();
</SCRIPT>
<input name="date4" readonly="yes" type="text" class= "mininota" style="height=18" size="12" value=<%=xfecha1%>>
<A id=anchor3
onclick="cal4.select(document.forms[0].date4,'anchor3','dd/MM/yyyy'); return false;"
href="C:\tmp\INACIPE\js\#"
name=anchor4><font size="1" face="Verdana, Arial, Helvetica, sans-serif"><strong>Selecciona</strong></font></A>
</td>
</tr>
<tr>
<td height="4" align="right" class="centrado"> <div align="right" ><font color="#000000"><strong>Fecha
final:</strong></font></div></td>
<td>
<!--<input name="textfield4" type="text" size="30" style="height=18" class= "mininota">-->
<SCRIPT language=JavaScript>
var cal4 = new CalendarPopup();
</SCRIPT>
<input name="date5" readonly="yes" type="text" class= "mininota" style="height=18" size="12" value="<%=xfecha2%>">
<A id=anchor5
onclick="cal4.select(document.forms[0].date5,'anchor5','dd/MM/yyyy'); return false;"
```

```
href="C:\tmp\INACIPE\js\#"
name=anchor4><font size="1" face="Verdana, Arial, Helvetica, sans-serif"><strong>Selecciona</strong></font></A>
                  </td>
                 </tr>
                 <tr>
                  <td height="4" align="right" class="centrado"><div align="right"><font color="#000000"><strong>Tipo
                     de convocatoria:</strong></font></div></td>
                  <td><select name="tconvocatoria" id="tconvocatoria" size="1" class= "mininota" style="height=18">
                     <option value="1">--Selecione--</option>
                     <option value="PER">Peritos</option>
                     <option value="MAE">Maestr&iacute;a</option>
                     <option value="AMP">AMPFs</option>
                     <option value="ESP">Especialidad</option>
                     <option value="DOC">Doctorado</option>
                     <option value="OTR">Otros</option>
                    </select> </td>
                 </tr>
                 <tr>
                  <td height="8" align="right" class="centrado"><div align="right"><font color="#000000"><strong>email:</strong></font></div></td>
                     <td><input name="email" id="email" type="text" size="30" style="height=18" class= "mininota" value="<%=xemail%>"></td>
                 </tr>
                 <tr>
                  <td height="16" align="right" class="centrado"><div class="centrado">
                     <div class="centrado">
                      <div align="right"><font color="#000000"><strong>Nota:</strong></font></div>
                     </div>
                    </div></td>
                  <td><textarea name="nota" id="nota" class="mininota"cols="48" style="height=60">
<%=xnota%>
</textarea></td>
                 </tr>
                 <tr align="center" valign="middle">
                  <td height="20" colspan="2" class="centrado"> <div align="center">
                     <table width="100%" border="0" cellspacing="0">
                      <tr bgcolor="dddddd">
                        <td height="20" colspan="2" align="center" class="centrado"><strong><font color="#000000" size="2">Agregar
                           Imagen</font></strong></td>
                      </tr>
                      <tr>
```

```
                        <td align="center" colspan=2> <input type="file" name="examinar"
id="examinar">
                        </td>
                       </tr>
                       <tr><!--
                        <td align="center">

                         <a href="javascript:eliminar();"><img src="../../images/eliminar.jpg" width="85"
height="24" border="0" align="right"></a>

</td>-->
                        <td align="center"> </td>
                       </tr>
                       <tr> </tr>
                       <tr bgcolor="dddddd">
                        <td height="20" colspan="2" align="center"><span
class="centrado"><strong><font color="#000000" size="2">Agregar
                          Archivos Adjuntos</font></strong></span></td>
                       </tr>
                       <tr>
                        <td align="center">

                         <a href="javascript:agregar();"><img src="../../images/agregar.jpg" width="85"
height="24" border="0" align="right"></a>

</td>
                        <td align="left"><img src="../images/backs/spacer.gif" width="1" height="1">
                        </td>
                       </tr>
<%
    if(adjuntos != null) {
        for(int i=0; i<adjuntos.length; i++) {
%>
        <input type=hidden name="adjunto<%=i%>" value="<%=adjuntos[i]%>">
<%
            }
        }
%>
                                 </form>
                       <tr>
        <form name=form2 action=ConvocatoriaPublicaForm.jsp method=post>
                        <td width="20" rowspan="2" align="center">
                         <div align="center"> <strong>
                          <select name="adjuntos" id="adjuntos" size= "6" >
```

```
<%
      if(adjuntos != null) {
           for(int i=0; i<adjuntos.length; i++) {
%>
                                  <option value="<%=adjuntos[i]%>"><%=adjuntos[i]%></option>
<%
                     }
               }
%>
                                  </select>
                                  </strong></div></td>
                                 <td height="25" align="center">

                                  <a        href="javascript:eliminaradj();"><img        src="../../images/eliminar.jpg"
width="85" height="24" border="0" align="right"></a>
</td>
                                </tr>
                                 <tr>
                                  <td align="center">

                                  <a    href="javascript:valida();"><img    src="../../images/enviar.jpg"    width="85"
height="24" border="0" align="right"></a>

</td>
                                  </tr>
                                 </table>
                                </div></td>
                               </tr>
                               <tr align="center" valign="top">
                                 <td colspan="2" class="centrado"> <div align="center"></div>
                                   <table width="100%" border="0">
                                     <tr>
                                       <td height="34" align="center" valign="top"><img src="../images/backs/spacer.gif"
width="15" height="35" align="absbottom">

                                   <a href="../../DSBEIMenuForm.jsp" target="_self">
                                    <img src="../../htm/images/menu.jpg" width="85" height="24" border="0"></a>

                                     </td>
                                    </tr>
                                   </table></td>
                                 </tr>
                               </table>
                             </form></td>
```

```
            </tr>
           </table></td>
          </tr>
          <tr align="center" valign="middle">
           <td> </td>
          </tr>
         </table>
         <hr align="center" width="96%"> </td>
      </tr>
    </table>

   </td>
   <td width="2%" align="left" valign="top">  </td>
  </tr>
  <tr>
   <td colspan="2" align="center" valign="top"> </td>
  </tr></table>
<form name="adjuntos" action="/UploadServlet" method=post enctype="multipart/form-data">
<input type=hidden name="forma" value="/htm/convocatorias/ConvocatoriaPublicaForm.jsp">
<input type=hidden name="titulo" value="">
<input type=hidden name="fecha1" value="">
<input type=hidden name="fecha2" value="">
<input type=hidden name="email" value="">
<input type=hidden name="nota" value="">
<input type=hidden name="tipo" value="">
</body>
</html>
<%
}
%>
```

6.3 Desarrollo de clases

Se programó la funcionalidad de los servlets, JavaBeans, custom tags, y templates de la capa de servicios web y la invocación de transacciones que contienen los EJB's. Éste debe corresponder con el diseño de las clases.

Además, e programó la funcionalidad y comportamiento de los EJB's de la capa transaccional y la interacción con la base de datos, el cual corresponde con el diseño del sistema.

A continuación se presenta un ejemplo de un EJB programado (ConvocatoriaAdministrador EJB.java).

```
import java.rmi.RemoteException;
import javax.ejb.*;
import java.util.Vector;

public class ConvocatoriaAdministratorEJB implements SessionBean {

  public String newConvocatoria(ConvocatoriaData convocatoria)
     throws RemoteException {
     ConvocatoriaDAO data = new ConvocatoriaDAO();
     return data.insertConvocatoria(convocatoria);
  }

  public ConvocatoriaLista getListaConvocatorias(String orden)
     throws RemoteException {
     ConvocatoriaDAO data = new ConvocatoriaDAO();
     ConvocatoriaLista lista = data.selectConvocatorias(orden);
     return lista;
  }

  public ConvocatoriaData getConvocatoriaData(String IDConvocatoria)
     throws  RemoteException {
     ConvocatoriaDAO data = new ConvocatoriaDAO();
     ConvocatoriaData convocatoria = data
        .selectConvocatoriaData(IDConvocatoria);
     return convocatoria;
  }

  public void modificarEstatusConvocatoria(String IDConvocatoria,
                                boolean newStatusPublicable)
     throws RemoteException {
     ConvocatoriaDAO data = new ConvocatoriaDAO();
```

```
        data.updateEstatusConvocatoria(IDConvocatoria,newStatusPublicable);
    }

    public void updateConvocatoriaData(ConvocatoriaData convocatoria)
        throws RemoteException {
        ConvocatoriaDAO data = new ConvocatoriaDAO();
        data.updateConvocatoria(convocatoria);
    }

    public ConvocatoriaAdministratorEJB() {
    }

    public void ejbCreate() {}

    public void ejbRemove() {}

    public void ejbActivate() {}

    public void ejbPassivate() {}

    public void ejbLoad() {}

    public void ejbStore() {}

    public void setSessionContext(SessionContext sc) {}

}
```

También se presenta un ejemplo de un Servlet programado (ConvocatoriaServlet Dispacher.java).

```
import javax.servlet.*;
import javax.servlet.HTTP.*;
import java.io.*;

public class ConvocatoriaServletDispatcher extends HttpServlet
                                implements java.io.Serializable{

    public void init(ServletConfig config) throws ServletException {
        super.init(config);
    }

    public void doPost(HttpServletRequest request,
```

```
                    HttpServletResponse response)
        throws ServletException, IOException{
        String seleccion = request.getParameter("seleccion");

        if(seleccion.equals("publica")){
            goToPage("/ConvocatoriaControllerPublica",request,response);
        }
        if(seleccion.equals("lista")){
            goToPage("/ConvocatoriaControllerListado",request,response);
        }
        if(seleccion.equals("consulta")){
            goToPage("/ConvocatoriaControllerConsulta",request,response);
        }
        if(seleccion.equals("cambiast")){
            goToPage("/ConvocatoriaControllerStatus",request,response);
        }
        if(seleccion.equals("modifica1")){
            goToPage("/ConvocatoriaControllerModificaStep1",request,response);
        }
        if(seleccion.equals("modifica2")){
            goToPage("/ConvocatoriaControllerModificaStep2",request,response);
        }
    }

    private void goToPage(String address, HttpServletRequest request,
                          HttpServletResponse response)
                    throws ServletException, IOException{
        getServletConfig().getServletContext().getRequestDispatcher(address)
        .forward(request,response);
    }

}
```

6.4 Diseño gráfico de interfaces

6.4.1 Características y especificaciones de diseño de interfaz para la página web del Instituto

Página Principal

Frames y menús

- Frameset en diseño para pantalla con resolución de 800 x 600 redimensionable a resolución de 1024 x 768 píxeles con la siguiente distribución:
- Frame superior con una altura fija de 79 píxeles. Contiene la siguiente información:
- Menú principal generado en Flash MX

Una imagen de fondo en formato .jpg integrando los siguientes elementos gráficos:

- Logotipo de la PGR
- Nombre de la Institución
- Logotipo de la Institución
- Tamaño total: 31 Kb

Frame lateral izquierdo con una columna fija de 120 píxeles de ancho con el siguiente contenido:

- Menú de contenidos desplegable generado en Flash MX
- Una imagen de fondo estática en formato .jpg
- Tamaño total: 80 Kb
- MainFrame (página de contenidos).

Del lado derecho con ancho redimensionable acorde al tamaño de la ventana, con contenido variable dependiente a la navegación en curso. En la pantalla principal se despliega el siguiente contenido:

En el área izquierda del MainFrame con un ancho variable de acuerdo a la resolución y tamaño de la ventana se despliegan los siguientes contenidos.

- Noticias
- Eventos
- Publicaciones

En el área derecha de la pantalla a modo de columna, con un ancho fijo de 150 pixeles se despliegan:

- Convocatorias
- Revista ITER CRIMINIS
- Agenda de Capacitación

Nota:

A cada contenido se le asigna una imagen en formato .jpg o .gif según convenga, sin exceder los 12 Kb.

Tamaño total: 80 Kb

6.4.2 Herramientas de diseño y construcción utilizadas para la página web del Instituto

Herramientas de diseño

- Adobe Photoshop 6.0 y Macromedia Flash MX
- Herramientas de construcción HTML
- Macromedia Dreamweaver MX.

Tamaño de las páginas en kb

Las páginas de contenido subsecuentes a la página principal no deberán exceder los 40 Kb, contando dos imágenes por página (no más de 10 Kb cada una) y el contenido en texto.

Número de imágenes por página

Las páginas de contenido (exceptuando la página principal) deberán desplegar un máximo de dos imágenes en formato .jpg o .gif (según convenga), no excediendo cada una de estas los 8 Kb.

En caso de que el despliegue de los contenidos exceda dichas características se recurrirá al despliegue de la información extendida por medio de paginación.

Controles permitidos

Las pantallas de acceso en la parte pública serán generadas dinámicamente en HTML con elementos estáticos generados en Flash.

Los controles permitidos para el área de usuario, serán:

- Botones con vínculos directos en menú principal (frame superior).
- Menú desplegable de acceso a contenidos (frame lateral izquierdo).
- Barras de desplazamiento: únicamente se permiten en el MainFrame ya sea en pantallas generadas dinámicamente o en HTML estático.
- Cajas de texto validadas para las ventanas de “logeo” para acceder a consultas en línea del acervo bibliográfico o la revista ITER CRIMINIS.

- Listas desplegables para las opciones de consulta del Acervo Bibliográfico y Tesis Publicadas (selección de criterios de búsqueda).
- Botones “enviar”, “aceptar” o “limpiar”.

Controles permitidos para el área de administración de contenidos

- Menús de acceso al área a modificar/consultar generado en flash.
- Cajas de texto con validación, para “logeo” y para ingreso de caracteres numéricos.
- Listas desplegables para selección de opciones de consulta o modificación.
- Botones abrir/adjuntar para vinculación de archivos gráficos, la cual llamará a la ventana de diálogo del sistema de archivos.
- Botones “buscar”, “enviar”, “aceptar” o “limpiar”.

Colores institucionales y tipografía

Los colores utilizados en el diseño de la página son:

- RGB 98-34-41 Pantone 490C Paleta Web: 622229
- RGB 235-219-196 Pantone 726C Paleta Web: EBDBC4

Fuentes

Se utilizará el siguiente estilo como formato estándar para el texto de títulos:

- CSS Style
- Tipo de fuente: Verdana, Arial, Helvetica, sans-serif
- Tamaño de fuente: 10px, Bold
- Color: #333333
- Alineación de Texto: Justificado, excepto en títulos

Se utilizará el siguiente estilo como formato estándar para el texto de contenidos.

- CSS Style
- Tipo de fuente: Verdana, Arial, Helvetica, sans-serif
- Tamaño de fuente: 10px, Bold

- Color: #333333
- Alineación de Texto: Justificado, excepto en títulos

Pantallas de diálogo

Diseño y estructura

Los colores utilizados en las pantallas de diálogo

-  RGB 126-145-160 Paleta Web: 7E91A0
-  RGB 102-102-102 Paleta Web: 666666
-  RGB 19-82-136 Paleta Web: 135288

Las pantallas de diálogo y de contenido del área pública serán generadas en HTML e imágenes en formato .gif y .jpg, centradas con respecto al "MainFrame" con una barra de título de altura variable; los colores a utilizar en las pantallas tendrán una correspondencia a cada una de las áreas de acuerdo a los colores indicados al inicio de esta sección, y corresponderán a los colores utilizados en las barras de título, las cuales serán de dimensiones variables.

Tipografía para barra de título de pantallas de diálogo

- Tipo de fuente: Verdana
- Tamaño de fuente: 12 px.
- Color: #FFFFFF
- Alineación de texto: Centrado o Izquierdo

Tipografía para contenidos

- Tipo de fuente: Verdana
- Tamaño de fuente: 12 px
- Color: #FFFFFF
- Alineación de texto: Izquierdo Justificado

Los menús emergentes serán generados por medio de javascript, entre éstos se encuentran los mensajes simples y mensajes de confirmación.

Presentación de información

El despliegue de información en pantalla es en formato HTML, imágenes estáticas en formato .jpg o .gif y elementos de diseño generados en Macromedia Flash.

Formatos para descarga de información

El único formato de descarga de información permitido al usuario es: Portable Document Format (PDF).

Estándares de diseño para paginación

En el caso de contenidos extendidos que requieran paginación, se utilizará un Template con la siguiente estructura:

- Pleca superior fija (como elemento de diseño) con una altura de 20 pixeles y un ancho variable con respecto a la ventana del browser en color "1" (especificados en "Colores Institucionales y Tipografía").
- El contenido se desplegará con la tipografía especificada para contenidos en el apartado de "Colores Institucionales y Tipografía".
- Las imágenes (en caso de ser requeridas) se desplegarán en el extremo superior izquierdo del "MainFrame" en caso de existir texto de contenido, o centradas en ausencia de éste.
- La navegación de las páginas generadas será por medio de un control de avance ubicado al fin de la página en curso, el cual indica el número de páginas encontradas (numérico), la página en uso (resaltada), un botón de "siguiente" y uno de "retroceso", siendo los dos últimos representados de modo gráfico.

CAPÍTULO 7. Pruebas e implantación del sistema

Antes de realizar la instalación en los servidores de producción correspondientes, se elaboraron una serie de pruebas para verificar el correcto funcionamiento del sistema.

7.1 Esquema de pruebas

Las pruebas se realizarán bajo el esquema booleano, es decir, se emplean solo dos posibilidades “Cumple” o “No cumple”.

Para la realización de las pruebas se deberá tener una impresión de las pantallas según el diseño presentado y un documento con las especificaciones del diagrama de clases o de secuencia que permita dirigir la secuencia de las pruebas.

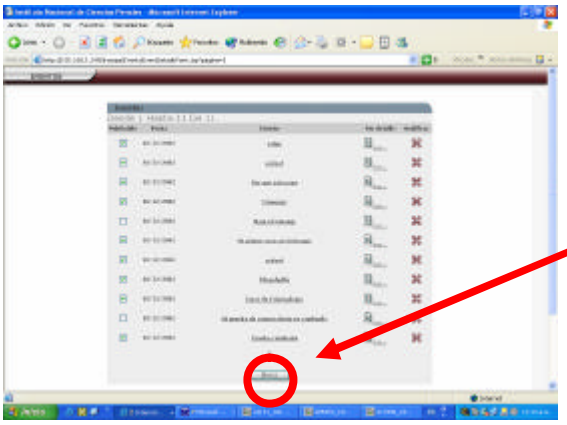
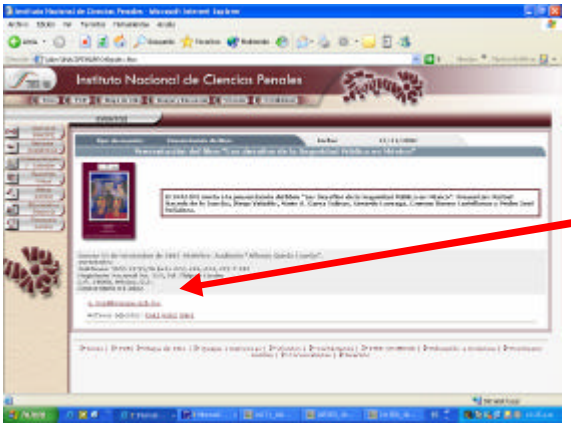
Una vez iniciadas las pruebas se deberá revisar cada pantalla en los siguientes rubros:

- Apariencia (Gráfico)
- Integridad
- Funcionalidad
- Transacción

Donde se deberá revisar que cumpla con las especificaciones marcadas en los documentos presentados.

7.1.1 Reporte de errores

En caso que en la revisión del sistema se encuentre algún error u omisión, el encargado de pruebas deberá reportar la incidencia mediante el formato de "Reporte de Error", por ejemplo:

#	Plataforma			Observaciones	Imagen	Nivel
#	IE 5.0	IE 6.0	NS 4.79			
1		☑		Realizar las observaciones pertinentes acerca de la incidencia. Comentar todo aquello que no pueda observarse a simple vista.		2
2			☑	Realizar las observaciones pertinentes acerca de la incidencia. Comentar todo aquello que no pueda observarse a simple vista.		2

Este formato contiene la información de la pantalla en la que se encontró el error, la versión de sistema operativo y herramientas utilizadas (ej. Internet Explorer 5.0), una copia de la pantalla (capturada con "print screen") colocada dentro del documento, la señalización del error ú omisión (un circulo rojo que indique en que parte de la pantalla está la incidencia) y finalmente la indicación del tipo de incidencia encontrada, de acuerdo a la siguiente tabla:

- Apariencia (gráfico): Se refiere a las diferencias que se encuentren entre el diseño y la pantalla de sistema presentada. Estas pueden ser en el color, tipo de letra, tamaño, tipo de gráficos, posición en la pantalla, etc.
- Integridad: Se refiere a las diferencias que se encuentren entre el diseño presentado y la pantalla de sistema en referencia al número total de elementos de pantalla.
- Funcionalidad: Se refiere a las diferencias que se encuentren entre la definición de lo que cada elemento debe de hacer y lo que realmente hace. El encargado de pruebas probará cada uno de los elementos de la pantalla y según la descripción del documento de pruebas, cada elemento deberá de responder de una forma específica, si esto no ocurre como lo marca la especificación se deberá de reportar como un error de funcionalidad. Esta incidencia no contempla la veracidad de los datos.
- Transacción: Se refiere a la precisión, completez o veracidad de los datos que deberá de arrojar un elemento determinado dentro de una pantalla. El encargado de pruebas deberá de validar los resultados que genere una acción particular.
- Grave: Se refiere a cualquier falla que no se contemple dentro de las cuatro anteriores. El personal de pruebas deberá de anotar lo ocurrido en el cuadro de observaciones.

7.2 Pruebas realizadas al sistema

En seguida se muestran algunas de las pruebas que se realizaron al sistema utilizando el esquema de pruebas antes mencionado.

7.2.1 Acceso al sistema

- Ventana de acceso

Descripción

Para ingresar a la parte de administrador es necesario que ingrese el usuario y la contraseña en los campos correspondientes y presione el botón “Entrar”, de ese modo se puede verificar que la persona que ingresa tiene permiso o no en ciertos roles o pantallas. El botón “Limpiar” borra los caracteres que se ingresen en los campos de usuario y contraseña.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

Ventana de acceso erroneo

Descripción

Si el usuario o contraseña se ingresaron de forma errónea se presenta una pantalla en donde se indica el error por medio de un mensaje y un botón de "Regresar" que lo regresa a la pantalla de Acceso para volver a firmarse.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

7.2.2 Acervo bibliográfico

Liga para acceder al listado del acervo bibliográfico

Descripción

Para el acceso al “Acervo Bibliográfico” sólo hay que hacer clic sobre el botón.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

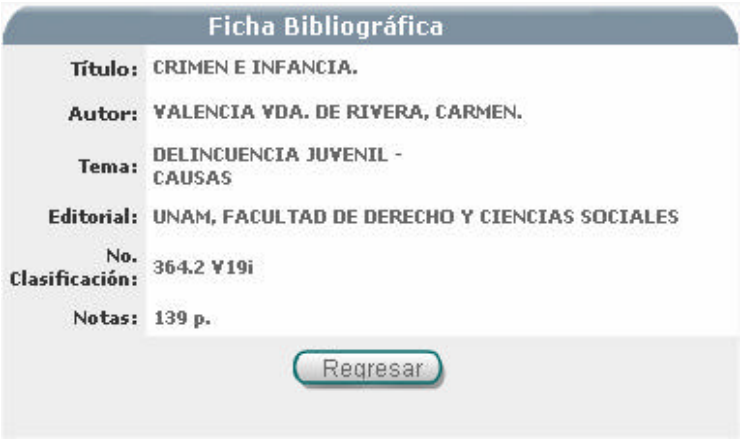
- Ventana del listado del acervo bibliográfico

Descripción

Una vez que se presiona la liga de Acervo Bibliográfico se visualiza una pantalla en donde se realiza la búsqueda llenando los campos de texto a buscar, buscar por y en qué documentos.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

- Ventana de la ficha bibliográfica



Descripción

Al ingresar al titulo se visualiza el titulo, autor, tema, editorial, No. de clasificación y notas. Con el botón de "Regresar" regresamos al listado de resultados de Acervo Bibliográfico.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

7.2.3 Eventos

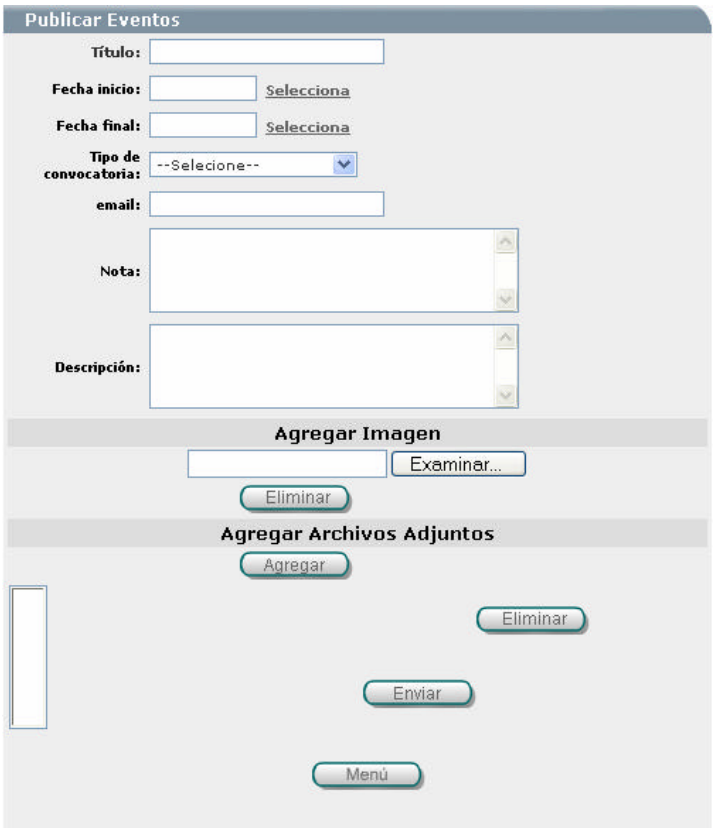
- Ventana de administrador de contenidos (menú de eventos)

Descripción

Cuando hacemos clic sobre el botón de "Eventos" nos muestra el menú que aparece en la imagen.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

- Ventana de publicación de eventos

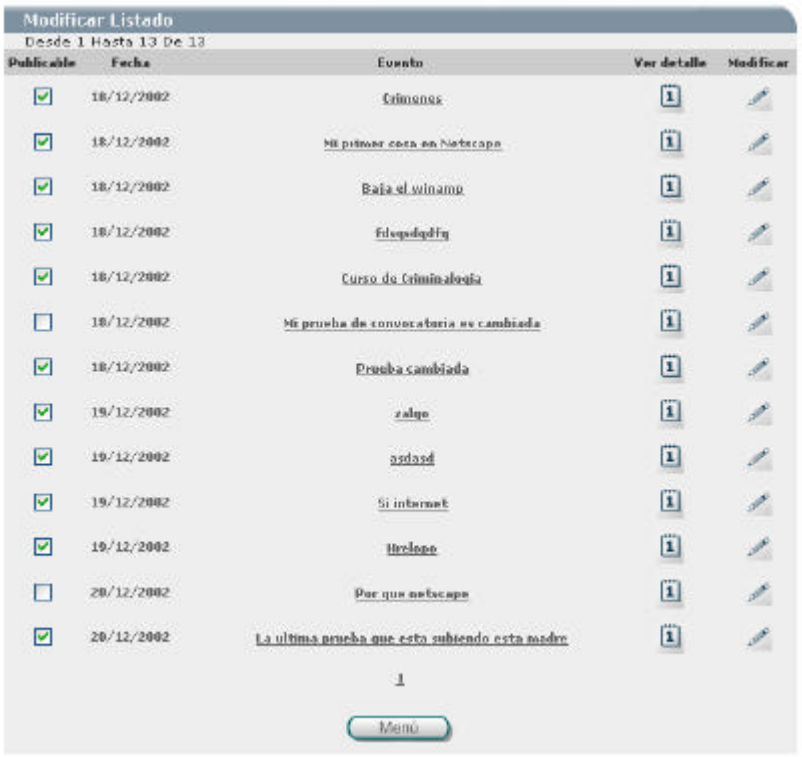


Descripción

En la parte administrativa de Eventos se pueden publicar eventos, llenando el formulario con los datos de titulo, las fechas de inicio y fin en que se llevara acabo, tipo del evento, email, nota (esta descripción es la que aparece en la página web), descripción del evento (esta descripción es la que aparece en la página web) así como agregar una imagen y adjuntar archivos.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

- Ventana del listado para la modificación de eventos

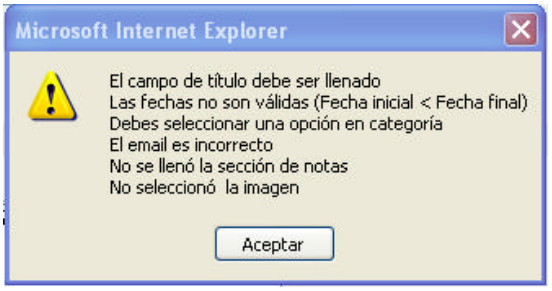


Descripción

En esta pantalla se muestra un formulario como el que se presenta al publicar los eventos. El llenado es el mismo la única diferencia es que puede modificar o todos o solo uno de los registros que se presentan en el formulario.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

- Ventanas de mensajes y notificaciones



Descripción

Una vez terminado con el llenado del formulario y si la información que ingresamos fue correcta, se mostrará un mensaje afirmando que el registro se agrego de forma correcta. De lo contrario el mensaje notificará en que parte del formulario se encuentra el error.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

7.2.4 Convocatorias

- Ventana de administrador de contenidos (menú de convocatorias)

Descripción

Cuando hacemos clic sobre el botón de "Convocatorias" nos muestra el menú que aparece en la imagen.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

- Venta para publicar convocatorias

Descripción

En la parte administrativa de Publicación de Convocatorias se pueden publicar llenando el formulario con los datos de titulo, las fechas de inicio y fin en que se llevará acabo, tipo de convocatoria, email, nota (esta descripción es la que aparece en la página web).

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

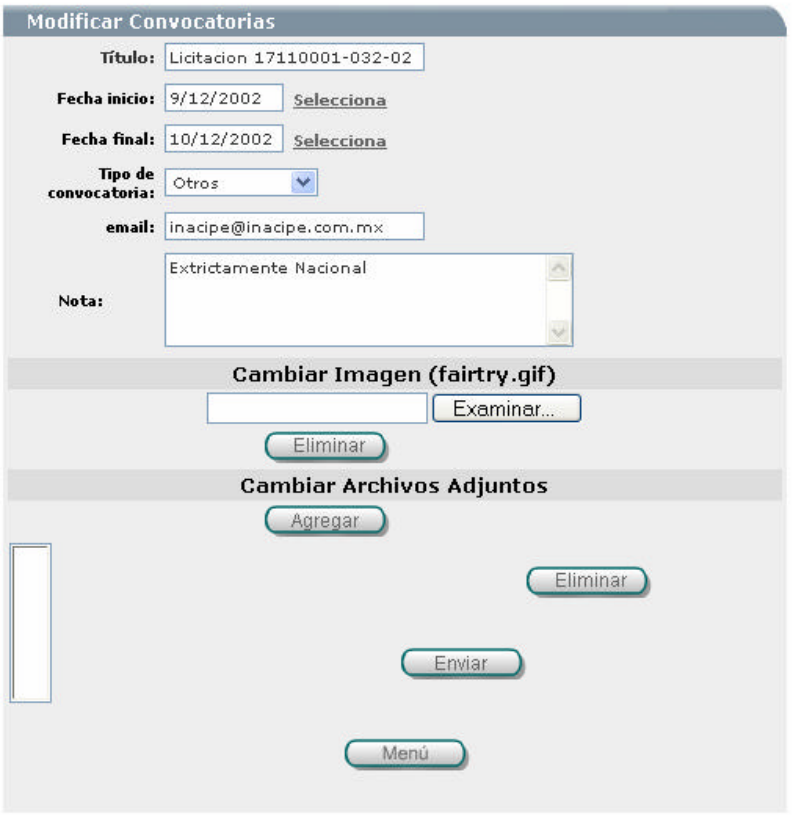
- Ventana del listado para la modificación de convocatorias

Descripción

Al ingresar en la parte de modificar lista se muestra todas las convocatorias que se han ingresado. Presentando la opción de publicable de la convocatoria, fecha en que se iniciará el curso, el título del mismo, su detalle y una opción de modificar.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

- Ventana para la modificación de convocatorias



Descripción

En esta parte, al ingresar se muestra una pantalla con un formulario como el que se presenta en la sección de publicar las convocatorias. El llenado es el mismo la única diferencia es que puede modificar todos o solo uno de los registros que se presentan en el formulario.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

- Ventana para confirmación de cambio de estatus de convocatoria

Descripción

Al realizar el cambio de estatus de la convocatoria aparece un primer mensaje que pregunta si quiere continuar con la acción que pretende realizar. Si cancelamos la acción nos mantenemos en la pantalla de "Modificar Listados". Si la respuesta es aceptar se realiza la acción seleccionando y se presenta una segunda pantalla con la afirmación de que se realizo correctamente el cambio. Y se regresa a la pantalla de "Modificar Listados".

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

7.2.5 Administración de usuarios

- Ventana de administrador de contenidos (menú de administración de usuarios)

Descripción

Cuando se presiona el botón "Usuarios" se muestra el menú que aparece en la imagen.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

- Ventana con el listado de usuarios del sistema

Descripción

Cuando se presiona el botón "Listar" se muestra un listado de los usuarios que se tienen registrados hasta el momento. La lista cuenta con cuatro columnas la primera es el login, el usuario, y las opciones borrar y modificar usuario.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

- Ventana para la modificación de los usuarios del sistema

Descripción

Cuando se presiona con el botón "Modificar" se presenta un formulario como el de la imagen con los datos del usuario. La modificación puede ser sobre un campo o varios, al término de ingresar los nuevos datos presionamos el botón registrar para que sean guardados los nuevos valores.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

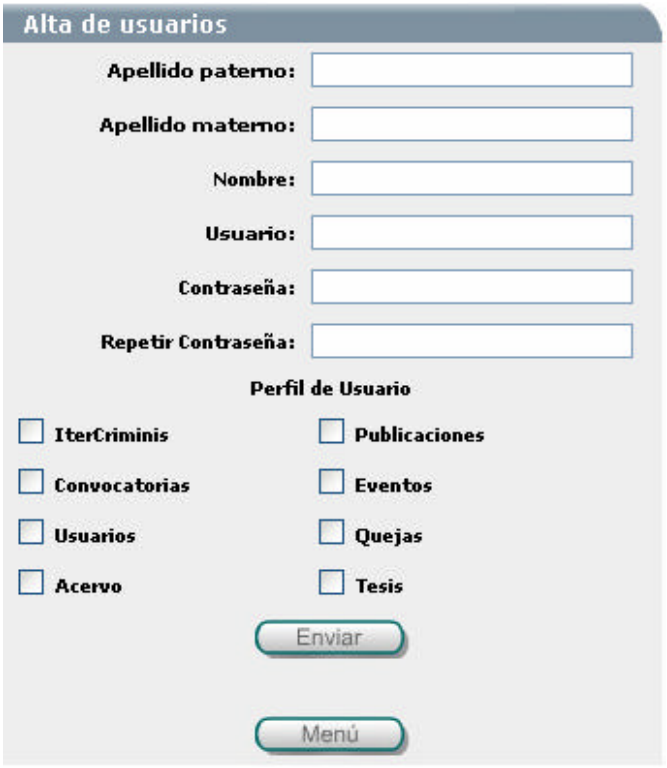
- Ventana para confirmar el borrado del usuario

Descripción

Al presionar el botón "Borrar Usuario" se visualiza un cuadro de texto como aparece en la imagen. Si aceptamos borrar al usuario presionamos el botón "Aceptar" de lo contrario elegimos "Cancelar".

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

- Ventana para alta de los usuarios del sistema

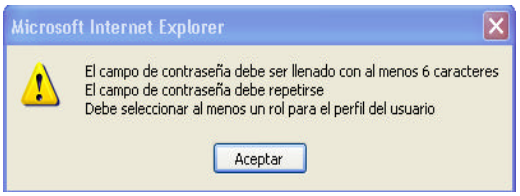


Descripción

Si seleccionamos el botón "Alta de usuario" aparece el formulario de la imagen en donde es necesario que se llenen todos los datos como se están solicitando. Una vez ingresados presionamos el botón "Enviar" y el usuario quedará registrado.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

➢ Ventanas de mensajes y notificaciones



Descripción

Una vez terminado con el llenado del formulario y si la información que ingresamos fue correcta, se mostrará un mensaje afirmando que el registro se agregó de forma correcta. De lo contrario el mensaje notificará en que parte del formulario se encuentra el error.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

- Ventana de mensaje de usuario inválido

Descripción

Si algún usuario intenta ingresar a roles de los cuales no tiene privilegios el mensaje que le enviara es el que se muestra en la imagen.

Esquema de pruebas			
	Revisado	Cumple	No cumple
Apariencia	☐	☐	☐
Integridad	☐	☐	☐
Funcionalidad	☐	☐	☐
Transacción	☐	☐	☐
Grave	☐	☐	☐

7.3 Instalación en servidores de producción

Una vez terminada la etapa de pruebas y después de haber realizado las correcciones correspondientes, se realizó la instalación de la aplicación en las diversas plataformas de hardware en donde el sistema debe ejecutarse: clientes, servidor web, servidor de aplicación y servidor de base de datos.

7.3.1 Instalación de la base de datos

- Creación de la base de datos

Para crear la base de datos deberá iniciar sesión en el servidor que tiene instalado PostgresSQL e ingresar como el usuario postgres

El comando para crear la base es el siguiente:

```
$bin/createdb -E MULE_INTERNAL DBINACIPE
```

Para borrar la base de datos utilizamos:

```
$bin/dropdb DBINACIPE
```

- Interprete de comandos de postgresql

Una vez que se creo la base de datos es necesario entrar el interprete de comandos de PostgreSQL para crear las tablas, funciones o cualquier instrucción SQL que se requiera.

Para entrar al interprete de comandos necesitamos crear una variable de ambiente, lo cual se realiza de la siguiente manera:

```
$export LD_LIBRARY_PATH=/usr/local/lib
```

Ya que se tiene la variable, se debe ejecutar el siguiente comando:

```
$bin/psql DBINACIPE
```

Si la base de datos fue creada correctamente el sistema responderá de la siguiente forma:

```
Welcome to psql 7.3, the PostgreSQL interactive terminal.

    Type:  \copyright for distribution terms

        \h for help with SQL commands

        \? for help on internal slash commands
```

```
\g or terminate with semicolon to execute query
\q to quit
DBINACIPE=#
```

= Instalación del lenguaje procedural

El lenguaje procedural que se maneja en las funciones que utiliza la pagina WEB es el PLSQL. Para instalar el leguaje procedural es necesario saber la ubicación del archivo plpgsql.so . Para encontrar el archivo utilice el siguiente comando desde el sistema operativo .

```
$bash-2.03$ find / -name plpgsql.so
```

Una vez que se sabe la ubicación del archivo se deberán ejecutar las siguientes instrucciones desde la línea de comandos sustituyendo la ruta que se muestra por la que encontró en el sistema donde se esté trabajando.

```
DROP FUNCTION plpgsql_call_handler ();
CREATE FUNCTION plpgsql_call_handler () RETURNS OPAQUE
    AS '/usr/lib/pgsql/plpgsql.so' LANGUAGE 'C';
DELETE FROM pg_language WHERE lanname='plpgsql';
CREATE TRUSTED PROCEDURAL LANGUAGE 'plpgsql' HANDLER
    plpgsql_call_handler LANCOMPILER 'PL/pgSQL';
```

= Creación de tablas

Para crear las tablas se debe ejecutar el script del archivo inacipe_db.txt de la siguiente manera:

```
DBINACIPE=#\i inacipe_db.txt
```

El archivo a ejecutar deberá estar situado en el home directory del usuario postgres o de lo contrario deberá indicar la ruta de donde se encuentra dicho archivo. Por ejemplo:

```
DBINACIPE=#\i /directori/inacipe_db.txt
```

Nota:

En este archivo no se crean las tablas de ADMUSUARIO, ADMROL y ADMPERFIL puesto que son tablas compartidas con los demás sistemas. *Para crear todas las tablas utilice el archivo inacipe_completo_db.txt*

= Eliminar tablas

Para eliminar las tablas que se crearon en el script anterior se puede utilizar el script del archivo borrar_web_db.txt de la siguiente manera:

DBINACIPE=#\i borrar_web_db.txt

Nota:

En este archivo no borra las tablas de ADMUSUARIO, ADMROL y ADMPERFIL puesto que son tablas compartidas con los demás sistemas. Para borrar todas las tablas utilice el archivo borrar_completo_db.txt

= Creación de funciones

Una vez que ya se crearon las tablas y se instaló el lenguaje procedural, se deben crear las funciones que se utilizarán en el sistema. Para crear las funciones ejecute el script del archivo creafunciones.txt de la siguiente manera:

DBINACIPE=#\i creafunciones.txt

= Eliminar funciones

Para eliminar las funciones que se crearon en el script anterior se puede utilizar el script del archivo Borrar_funcion_web_db.txt de la siguiente manera:

DBINACIPE=#\i borrar_funcion_web_db.txt

= Salida del interprete de comandos de postgresql

El comando para salir del interprete de comandos y regresar al shell del sistema operativo es \q, ejemplo:

DBINACIPE=#\q

Importar acervo bibliográfico

= Formato del archivo

El archivo fuente deberá ser un archivo de tipo texto que contendrá los siguientes campos:

- Tipo de acervo
- Ficha bibliográfica
- Numero de Clasificación
- Autor
- Titulo
- Editorial
- Nota
- Tema

Los campos se deberán presentar en el archivo, en una sola línea por registro, la cual contendrá todos los campos separados por "|" (pipe). Ejemplo:

> Tipo de acervo | Ficha bibliografica | Numero de Clasificación | Autor | Titulo | Editorial | Nota | Tema.

Por lo anterior, en cada línea deberán siempre haber 7 pipes (separadores de campos) aunque no existan datos en dichos campos.

Nota:

Es importante que en ninguna de las líneas del archivo se tenga el carácter "\" debido a que dicho carácter no puede ser interpretado por la base de datos y causará error al tratar de realizarse la importación.

= Obtención del archivo fuente

Para obtener el archivo fuente desde Microsoft Access, siga los siguientes pasos:

- Abra el archivo con extensión mdb desde Access.
- Seleccione la tabla a exportar, como por ejemplo la tabla Catalogo
- En el menú de archivo seleccione la opción exporta
- En guardar como tipo seleccione Archivos de Texto (*.txt)
- Presione el botón de Guardar, donde a continuación aparecerá el asistente para exportación de texto.
- En el formato de exportación seleccione la opción Delimitado y oprima siguiente.

- En el delimitador que separa los campos seleccione Otros y escriba el carácter | (pipe).
- En el campo de cualificador de texto seleccione la opción {ninguno}
- Presione el botón de siguiente
- Verifique la ruta donde se guardará el archivo y presione finalizar.
- Una vez que se generó el archivo, verifique que el archivo no contenga el carácter "\". En caso de que el archivo contenga dicho carácter en alguno de los registros, simplemente elimine el carácter y guarde el archivo.

= Importación de los datos

Para importar el archivo, es necesario colocarlo en el servidor que tiene instalada la base de datos.

Para subir el archivo al servidor mediante ftp, conéctese con el usuario postgres y coloque el archivo en el home directory del usuario. En la transferencia asegúrese que el archivo se transmita como tipo ascii.

Una vez que el archivo se encuentra en el sistema de archivos del servidor, inicie sesión como usuarios postgres y entre al intérprete de comandos de PostgreSQL.

La función para importar el acervo se llama importaAcervo y recibe como parámetro una cadena que indica la ruta y nombre del archivo de texto que contiene los datos del acervo y regresa el número de registros que insertó. La forma de ejecutar la función es la siguiente:

```
DBINACIPE=# select importaAcervo('/rutarchivo/nombrearchivo.txt');
```

Cuando se terminan de importar los datos se muestra el número de registros que se lograron insertar en la tabla de acervo.

= Manejo de errores

```
ERROR: Function ´importaAcervo (unknown)´ does not exist
        Unable to identify a function that satisfies the given argument types
        You may need to add explicit typecasts
```

La función no fue creada de forma correcta, asegúrese de que la función existe con el siguiente comando:

```
DBINACIPE=# \df importaAcervo
```

De existir la función el sistema responderá de la siguiente manera:

```
 Result  |  Function      |  Arguments
---------+----------------+-------------------
  integer | importaacervo | character varying
(1 row)
```

De lo contrario indicara (0 row).

Si la función existe pero el error continua, asegúrese de que el argumento que esta pasando es una cadena delimitada por comas simples.

> ERROR: COPY command, running in backend with effective uid 26, could not open file './ruta/archivo.txt' for reading. Errno = No such file or directory (2).

La ruta o el nombre del archivo no es correcto, utilice rutas absolutas para referirse a la ubicación del archivo.

> ERROR: copy: line 1, pg_atoi: error in "Datos de alguna línea del archivo "': can't parse " Datos de alguna línea del archivo"'

El archivo no pudo ser procesado por que no cumple con el formato requerido o por que contiene caracteres no válidos. En este caso, los datos del acervo no serán actualizados y continuarán los anteriores.

7.3.2 Instalación en el servidor de aplicaciones

Ejecutar el comando installapp.sh de la siguiente manera:

bash installapp.sh

el programa realizará las siguientes preguntas:

Path donde se desea instalar la aplicación

Ip de la Base de Datos

Puerto de la Base de Datos

Nombre de la Base de Datos

Usuario de la Base de Datos

Password de la Base de Datos

Cuando termine de realizar la instalación ejecutará el siguiente comando:

bash configapp.sh

el programa realizará las siguientes preguntas:

Path de Orion

aquí solo le indicaremos la ruta en donde se encuentra orion.

7.3.3 Instalación en el servidor web

Ejecutar el comando installweb.sh de la siguiente manera:

bash installweb.sh

el programa realizará las siguientes preguntas:

Path donde se desea instalar la aplicación

Ip del servidor de correos(Para enviar los correos a través de SMTP)

Correo para enviar quejas(Dirección de correo a donde se enviaran las quejas emitidas desde la página web)

Cuando termine de realizar la instalación ejecutara el siguiente comando:

bash configweb.sh

la configuración realizara las siguientes preguntas:

Path de Orion, donde se encuentra ubicado

Ip de Orion remoto (Servidor de aplicaciones)

Admin. password (Servidor de aplicaciones)

Cuando finalice la instalación del sistema se debe ejecutar el orion para que realice el "deploy" de la aplicación de la siguiente manera:

java -jar orion.jar

Orion tardará varios segundos en realizar la operación, cuando finalice se debe detener la ejecución de orion con <CTRL>+C.

Posteriormente se debe editar el archivo orion-application.xml dentro de orion en el subdirectorio application-deployments/Inacipe. Con esto indicaremos al servidor orion que el modulo de los EJBs lo tome del servidor de aplicaciones.

El archivo contiene lo siguiente:

```
<?xml version="1.0"?>
<!DOCTYPE orion-application PUBLIC "-//Evermind//DTD J2EE Application runtime
1.2//EN" "HTTP://www.orionserver.com/dtds/orion-application.dtd">

<orion-application deployment-version="1.5.2">
      <ejb-module remote="false" path="ejb-jar-ic.jar" />
      <web-module id="war-ic" path="war-ic.war" />
      <persistence path="persistence" />
      <principals path="principals.xml" />
      <log>
            <file path="application.log" />
      </log>
      <namespace-access>
          .
          .
          .
```

Hay que cambiar la palabra "false" por la palabra "true". Una vez realizado el cambio la configuración de la aplicación ha finalizado.

Después de verificar que la aplicación funciona correctamente se debe borrar el directorio war-ic en el servidor de aplicaciones contenido dentro del path en donde se instalo la aplicación.

Conclusiones

En el desarrollo del sistema, se empleó la metodología orientada a objetos inicialmente elegida debido a que se requerían avances en forma modular así como un desarrollo iterativo. Por esta razón el lenguaje de modelado que se utilizó fue UML, el cual, es el lenguaje estándar aceptado por la OMG, diseñado por investigadores de reconocido prestigio, además de que muchas herramientas CASE lo han adoptado como lenguaje de modelado.

Una de las ventajas en este sistema fue utilizar el patrón MVC en este desarrollo, debido a que al momento de realizar alguna modificación al sistema basta con hacerla en la sección de código que representaba el problema sin preocuparse de las otras secciones.

Durante la creación del sistema encontramos que UML nos fue muy útil para representar y modelar la información no sólo en la fase de análisis, sino también en el diseño ya que mediante la explotación de los casos de uso fue posible obtener una mejor panorámica de las características que se debían cumplir con el sistema. De igual forma durante la creación de los diagramas de clases aparecieron nuevos detalles que complementaron el análisis previamente realizado con los casos de uso.

Con los diagramas de secuencia y de colaboración fue posible crear una imagen muy cercana a lo que finalmente se programaría gracias a que en este punto conocemos las clases que conforman al sistema y como se deben comportar los métodos para permitir la interacción entre las distintas clases.

Uno de los inconvenientes fue el tiempo requerido para la creación de cada uno de los diagramas ya que al tratarse de un diagrama su elaboración y mantenimiento al no realizarse con alguna herramienta especializada, consume tiempos valiosos del desarrollo del sistema.

Otro inconveniente en el uso de los diagramas es que conforme la complejidad del caso de uso crece, el tamaño de dicho diagrama también, por lo cual resulta difícil poderlo distribuir por algún medio impreso de forma que sea portable y fácil de consultar por los desarrolladores, debido a que una hoja carta resulta insuficiente en su espacio para el área que ocupan los diagramas.

Una desventaja en la utilización de UML sería la curva de aprendizaje al interpretar los diagramas, en caso de que los desarrolladores no conocieran al 100% este lenguaje. Por otro lado si el equipo de desarrollo es experimentado, el tiempo de programación disminuye considerablemente.

Al término de la implementación de la página web podemos percatarnos que el tiempo de publicación disminuye considerablemente así como el personal que se requería para llevar

acabo esta tarea, pudiendo emplear al personal en otras actividades dentro del Instituto. Además, la persona que realice estas actualizaciones no requiere ser experto en HTML o que haya la necesidad de mantenerla ya que basta con colocar el texto, imágenes o archivos y el sistema hará lo conveniente para colocarlo en el formato adecuado.

Sin embargo, este sistema constituye una parte de lo que quiere llegar a tener el Instituto en la proyección del área de sistemas, ya que éste va a ir adaptándose a las necesidades de la organización. Es por ello que el mantenimiento, adecuación y crecimiento del sistema será realizado por personal del área de sistemas.

Bibliografía

Fowler, Martín
UML gota a gota
Addison Wesley Logman de México, S.A. de C.V.
México, 1999

P. Stevens, R. Pooley
Utilización de UML en Ingeniería del Software con Objetos y Componentes
Pearson Educación, S.A.
Madrid, 2002

I. Jacobson, G. Booch, J. Rumbaugh
El proceso Unificado de Desarrollo de Software
Pearson Educación, S.A.
Madrid, 2000

Hall, Marty
Servlets y JavaServer Pages. Guía Práctica
Pearson Educación, S.A.
México, 2001

Martín James, Odell James J.
Análisis y Diseño Orientado a Objetos
Prentice may Hispanoamaricana, S.A. de C.V.
México, 1992

Carlos Alberto Roman Zamitiz
Sistema con arquitectura cliente, servidor distribuida para el laboratorio de computo-die
México, 1999

Pastrana Vicente, Israel
Java2 Versión 1.4
Anaya
México 2000

Eckel, Bruce
Piensa en Java
Pearson
México 1998

Date, C.J.
Introducción a los sistemas de bases de datos
Quinta Edición, 1993
Addison Wesley

Korth Henry F.
Fundamentos de bases de datos
Ed. Mc Graw Hill

Thomas Lockhart
PostgreSQL
HTTP://www.postgresql.org/docs/7/static/postgres.htm

Momjian, Bruce
Frequently Asked Questions (FAQ) for PostgreSQL
HTTP://www.PostgreSQL.org/docs/faqs/FAQ.html.

J2EE Application Overview
HTTP://www.orionserver.com/
Copyright © 2003 IronFlare AB