

# ÍNDICE

<u>1.INTRODUCCIÓN.....</u>	3
<u>2.DISEÑO.....</u>	4
<u>2.1.Visión global.....</u>	4
<u>2.2.Subsistema uBet.....</u>	4
<u>2.2.1.Objetivo.....</u>	4
<u>2.2.2.Arquitectura global.....</u>	6
<u>2.2.3.Paquete account.....</u>	8
<u>2.2.3.1.Objetivo.....</u>	8
<u>2.2.3.2.Arquitectura.....</u>	8
<u>2.2.4.Paquete accountoperation.....</u>	10
<u>2.2.4.1.Objetivo.....</u>	10
<u>2.2.4.2.Arquitectura.....</u>	10
<u>2.2.5.Paquete adminfacade.....</u>	12
<u>2.2.5.1.Objetivo.....</u>	12
<u>2.2.5.2. Arquitectura.....</u>	12
<u>2.2.6.Paquete bet.....</u>	15
<u>2.2.6.1.Objetivo.....</u>	15
<u>2.2.6.2.Arquitectura.....</u>	15
<u>2.2.7.Paquete betoption.....</u>	17
<u>2.2.7.1.Objetivo.....</u>	17
<u>2.2.7.2.Arquitectura.....</u>	17
<u>2.2.8.Paquete bettype.....</u>	19
<u>2.2.8.1.Objetivo.....</u>	19
<u>2.2.8.2.Arquitectura.....</u>	19
<u>2.2.9.Paquete category.....</u>	21
<u>2.2.9.1.Objetivo.....</u>	21
<u>2.2.9.2.Arquitectura.....</u>	21
<u>2.2.10.Paquete categoryquestion.....</u>	23
<u>2.2.10.1.Objetivo.....</u>	23
<u>2.2.10.2.Arquitectura.....</u>	23
<u>2.2.11.Paquete country.....</u>	25
<u>2.2.11.1.Objetivo.....</u>	25
<u>2.2.11.2.Arquitectura.....</u>	25
<u>2.2.12.Paquete event.....</u>	27
<u>2.2.12.1.Objetivo.....</u>	27
<u>2.2.12.2.Arquitectura.....</u>	27
<u>2.2.13.Paquete question.....</u>	29
<u>2.2.13.1.Objetivo.....</u>	29
<u>2.2.13.2.Arquitectura.....</u>	29
<u>2.2.14.Paquete searchfacade.....</u>	31
<u>2.2.14.1.Objetivo.....</u>	31
<u>2.2.14.2. Arquitectura.....</u>	31
<u>2.2.15.Paquete userfacade.....</u>	34
<u>2.2.15.1.Objetivo.....</u>	34
<u>2.2.15.2.Arquitectura.....</u>	34
<u>2.2.16.Paquete userprofile.....</u>	37
<u>2.2.16.1.Objetivo.....</u>	37
<u>2.2.16.2.Arquitectura.....</u>	37
<u>2.2.17.Paquete util.....</u>	39
<u>2.2.17.1.Objetivo.....</u>	39

<u>2.2.17.2.Arquitectura.....</u>	39
<u>2.2.18.Paquete controller.actions.....</u>	40
<u>2.2.18.1.Objetivo.....</u>	40
<u>2.2.18.2.Arquitectura.....</u>	40
<u>2.2.19.Paquete controller.caches.....</u>	42
<u>2.2.19.1.Objetivo.....</u>	42
<u>2.2.19.2.Arquitectura.....</u>	42
<u>2.2.20.Paquete controller.frontcontroller.....</u>	43
<u>2.2.20.1.Objetivo.....</u>	43
<u>2.2.20.2.Arquitectura.....</u>	43
<u>2.2.21.Paquete controller.plugin.....</u>	45
<u>2.2.21.1.Objetivo.....</u>	45
<u>2.2.21.2.Arquitectura.....</u>	45
<u>2.2.22.Paquete controller.session.....</u>	45
<u>2.2.22.1.Objetivo.....</u>	45
<u>2.2.22.2.Arquitectura.....</u>	45
<u>2.3.Subsistema WastingMoney.....</u>	46
<u>2.3.1.Objetivo.....</u>	46
<u>2.3.2.Arquitectura global.....</u>	46
<u>2.3.3.Paquete controller.actions.....</u>	47
<u>2.3.3.1.Objetivo.....</u>	47
<u>2.3.3.2.Arquitectura.....</u>	47
<u>2.3.4.Paquete controller.frontcontroller.....</u>	49
<u>2.3.4.1.Objetivo.....</u>	49
<u>2.3.4.2.Arquitectura.....</u>	49
<u>3.IMPLEMENTACIÓN.....</u>	49
<u>3.1.Estructura de la distribución.....</u>	49
<u>3.2.Instrucciones de compilación.....</u>	50
<u>4.INSTALACIÓN.....</u>	51
<u>5.PROBLEMAS CONOCIDOS.....</u>	62
<u>6.REFERENCIAS.....</u>	62

# 1 INTRODUCCIÓN

En este documento se detalla el diseño e implementación de la práctica obligatoria de la asignatura Integración de Sistemas del curso 2005-2006, que consiste en el desarrollo de un sitio web de apuestas deportivas al estilo de betandwin [1], utilizando para ello la tecnología J2EE [2]. En todo momento se ha seguido el estándar de codificación definido por Sun Microsystems [3]. La aplicación web ha sido bautizada como “**uBet**”.

En **uBet** se han desarrollado todos los apartados obligatorios con las siguientes características:

- Se permite la actualización del login de un usuario (sólo la versión plain).
- Adicionalmente a lo pedido en la práctica, un usuario puede disponer de varias cuentas (con la misma o con distinta tarjeta de crédito), entre las que puede cambiar sin tener que volver a autenticarse.
- La aplicación es totalmente plugable, es decir, funciona correctamente tanto con JDBC [4] como con EJB [5]

Además se han implementado **todos** los apartados optativos especificados en el enunciado de la práctica:

- **Casos de prueba** relativos a los casos de uso del modelo con **JUnit** [6]. A mayores de lo requerido, estos casos de prueba también son plugables, es decir, son válidos para JDBC [4] y para EJB [5].
- **Eventos en portada**, con un sistema de rotación de manera que cada vez que se accede a la página principal (después de realizar correctamente una acción, dándole al enlace “Home” en la barra lateral o dándole al enlace “More events” en la página principal), se actualizan los tres eventos en portada.
- **Contenido externo**: se ha implementado una aplicación web externa a **uBet** (llamada “**Wasting Money**”) que permite:
  - Obtener los eventos en portada de **uBet**.
  - Obtener el conjunto de eventos que se han introducido en el día más reciente, correspondiente a un conjunto de categorías hoja.
- **Eventos interesantes**: consiste en la presentación de los eventos introducidos en el día más reciente que posiblemente son los más interesantes para el usuario, según sus últimas interacciones con el sitio web (las apuestas realizadas en el último mes).

# 2 DISEÑO

## 2.1 Visión global

Para esta práctica se ha desarrollado un subsistema para la aplicación correspondiente al sitio web de apuestas deportivas (**uBet**) y otro subsistema para la aplicación externa (**WastingMoney**).

La descripción de los subsistemas es la siguiente:

- **Subsistema uBet:** en este subsistema se han implementado las funcionalidades requeridas por la aplicación. Conceptualmente se divide en cuatro subsistemas, aunque a nivel de implementación se encuentren bajo el mismo subsistema físico. Los subsistemas en los que se divide son los siguientes:
  - *Subsistema uBet-usuario:* en este subsistema se han implementado las funcionalidades relativas a los casos de uso de: registro de usuarios, autenticación, realización de apuestas, consulta del estado de las apuestas, creación de cuentas, modificación de datos de usuarios y cuentas (incluyendo ingreso y retirada de dinero) y consulta de las operaciones realizadas sobre las cuentas.
  - *Subsistema uBet-administrador:* en este subsistema se han implementado las funcionalidades relativas a los casos de uso de: insertar eventos, insertar tipos de apuestas y publicar los resultados de los tipos de apuestas.
  - *Subsistema uBet-búsqueda:* en este subsistema se han implementado las funcionalidades relativas a los casos de uso de: eventos destacados, recorrido por categorías, eventos recientes y visualización de los tipos de apuesta de un evento.
  - *Subsistema uBet-prueba:* en este subsistema se han implementado las pruebas de unidad para todas las fachadas del modelo.
- **Subsistema WastingMoney:** en este subsistema se ha implementado una aplicación web externa totalmente independiente a **uBet**, que permite recuperar cierta información a través de XML [9].

## 2.2 Subsistema **uBet**

### 2.2.1 Objetivo

El subsistema **uBet** se encarga de implementar la gestión de todos los casos de uso relativos al sistema: registro de usuarios, autenticación, realización de apuestas, consulta del estado de las apuestas, creación de cuentas, modificación de datos de usuarios y cuentas (incluyendo ingreso y retirada de dinero), consulta de las operaciones realizadas sobre las cuentas, insertar eventos, insertar tipos de apuestas y publicar los resultados de los tipos de apuestas, eventos destacados, recorrido por categorías, eventos recientes y visualización de los tipos de apuesta de un evento.

- Registro de usuarios: permite el registro de nuevos usuarios. Una vez registrados, podrán acceder a la aplicación introduciendo su login y su password
- Autenticación: permite acceder a la aplicación a los usuarios registrados

introduciendo su login y su password.

- Realización de apuestas: cada vez que se visualiza un tipo de apuesta, se muestran las opciones disponibles y sus cuotas, pudiéndose apostar por cualquiera de ellas mediante un enlace. Sólo los usuarios autenticados podrán realizar apuestas.

- Consulta del estado de las apuestas: un usuario autenticado puede consultar las apuestas realizadas a lo largo del tiempo. Por cada apuesta se muestra la fecha en

la que se hizo, la descripción y fecha del evento, la pregunta del tipo de apuesta, la opción elegida, la cuota de ganancia de esa opción, la opción u opciones ganadoras y una indicación de su estado (pendiente, ganada o perdida).

- Creación de cuentas: un usuario autenticado puede crear cuentas a parte de tener la cuenta que se le crea automáticamente al registrarse. Si tiene varias cuentas puede cambiar entre ellas sin tener que volver a autenticarse.

- Modificación de los datos de usuario y cuentas: un usuario autenticado puede modificar los datos relacionados con su información de registro además de poder cambiar los datos de la tarjeta asociada a cualquiera de sus cuentas. También puede ingresar o retirar dinero de cualquiera de sus cuentas.

- Consulta de las operaciones realizadas sobre las cuentas: un usuario autenticado podrá visualizar todas las operaciones que se han hecho sobre cualquiera de sus cuentas. Por cada una de las operaciones se muestra: la fecha en la que tuvo lugar, el tipo de operación (añadir, retirar, apostar o ganar apuesta) y la cantidad monetaria. Además, en el caso de las operaciones de tipo apostar o ganar apuesta se muestra un enlace que permite visualizar los detalles de la apuesta (fecha en la que se hizo, descripción y fecha de evento, la pregunta del tipo de apuesta, la opción elegida, la cuota de ganancia y la opción u opciones ganadoras).

- Insertar eventos: el administrador, una vez autenticado, puede insertar eventos en cualquier categoría hoja después de navegar por el árbol de categorías. Para insertar el evento también debe especificar el tipo de apuesta destacada de ese evento, con sus opciones de apuesta.

- Insertar tipos de apuesta: el administrador, una vez autenticado, puede insertar nuevos tipos de apuesta para un evento mediante un enlace situado al lado del tipo de apuesta destacado asociado a dicho evento.

- Publicar los resultados de los tipos de apuestas: el administrador, una vez autenticado, puede publicar los resultados de los tipos de apuesta de un evento que ya ha comenzado tras navegar por el árbol de categorías y llegar a la categoría hoja a la que pertenece el evento. Al visualizar los tipos de apuesta de dicho evento, aparecerán enlaces al lado de cada uno de ellos para publicar las opciones ganadoras de cada uno de ellos. Tras publicar los resultados, se les ingresará a los usuarios que hayan apostado por las opciones ganadoras el dinero ganado (lo que apostó \* la cuota).

- Eventos destacados: en la página principal de la aplicación aparecen a lo sumo tres eventos destacados. El administrador puede especificar que un evento sea publicitado en portada o deje de estarlo mediante un enlace. Como posiblemente el administrador desee publicitar más de tres eventos se realiza un sistema de rotación que se ejecuta cada vez que se accede a la página principal (después de realizar correctamente una acción, dándole al enlace "Home" en la barra lateral o dándole al enlace "More events" en la página principal).

El sistema de rotación tiene las siguientes características: los eventos pueden estar en cuatro estados (en portada, listo para entrar en portada, esperando para entrar en portada porque acaban de salir y no estar en portada nunca porque el administrador no desea que sea publicitado o que ya haya empezado), los eventos que están en portada pasan al estado de esperando para entrar en portada y de los que están listos para entrar en portada se cogen los tres primeros y, en caso de no haber tres eventos en estado listo, se ponen todos al estado listo.

- Recorrido por categorías: cualquier usuario (autenticado o no) puede navegar por el árbol de categorías. La selección de una categoría provoca la visualización de todas sus categorías hijas, excepto que sea una categoría hoja, en cuyo caso se muestran todos los eventos de la categoría que todavía no han comenzado. Por cada evento se muestran la fecha de celebración del evento, el tipo de apuesta destacado (con sus opciones de apuesta asociadas) y un enlace para ver todos los tipos de apuesta de ese evento.
- Eventos recientes: un usuario autenticado puede ver una serie de eventos que pueden ser interesantes para él mediante un enlace en la página principal. Para saber que eventos pueden ser interesantes para el usuario se utiliza la información de las apuestas realizadas por dicho usuario en el último mes, y se muestran los eventos introducidos en el día más reciente correspondientes a las categorías a la que pertenecen los eventos por los que apostó el usuario.
- Visualización de los tipos de apuesta de un evento: un usuario autenticado puede ver todos los tipos de apuesta de un evento mediante un enlace situado al lado del tipo de apuesta destacada de ese evento.

### 2.2.2 Arquitectura global

El diagrama que muestra la arquitectura global de paquetes se puede ver a continuación:



## 2.2.3 Paquete account

### 2.2.3.1 *Objetivo*

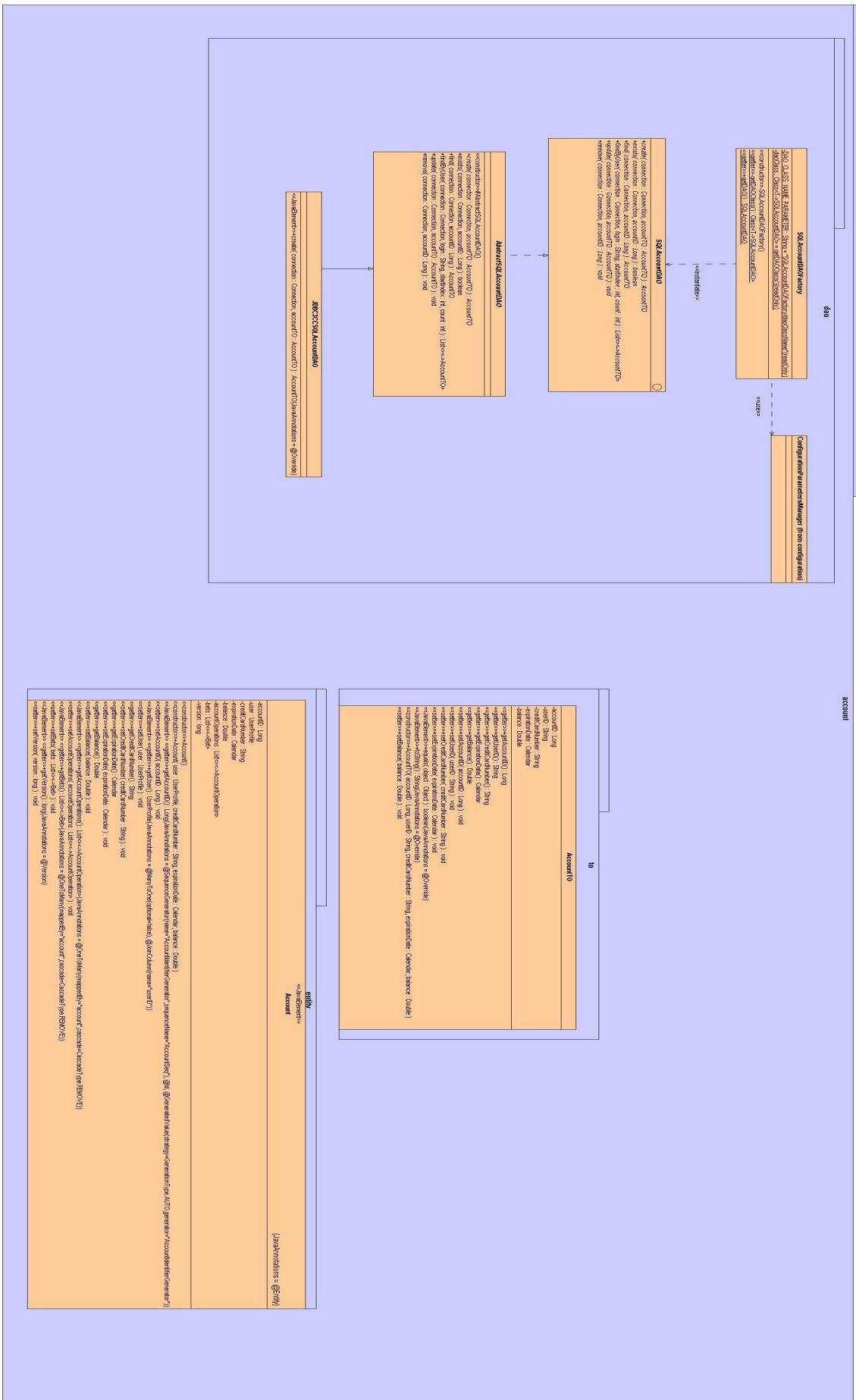
Este paquete modela las cuentas que puede tener un usuario.

### 2.2.3.2 *Arquitectura*

En el siguiente diagrama se muestran las clases del paquete account. Por un lado, se muestran las clases del paquete dao, que implementan el patrón Data Access Object (DAO). Estas clases se usan sólo para la versión Plain de **uBet**. Existe una clase que implementa el patrón Factory y permite obtener instancias de los DAOs. Existe una clase abstracta que implementa todos los métodos que sirven para trabajar con los datos de la BD, excepto el de crear, que se implementa en tres clases distintas (una para BDs con Columnas Countador y drivers JDBC 3.0, otra para BDs con Columnas Countador sin drivers JDBC 3.0 y otra para BDs con Generadores de Identificadores).

También se muestran las clases que modelan los datos usando el patrón Transfer Object y que son los elementos persistentes del sistema.

Por último también se muestra la clase Entity usada en la versión EJB de **uBet**, que contiene los atributos de la tabla correspondiente, así como las relaciones de dicha tabla con las demás tablas usadas por la aplicación.



## 2.2.4 Paquete accountoperation

### 2.2.4.1 *Objetivo*

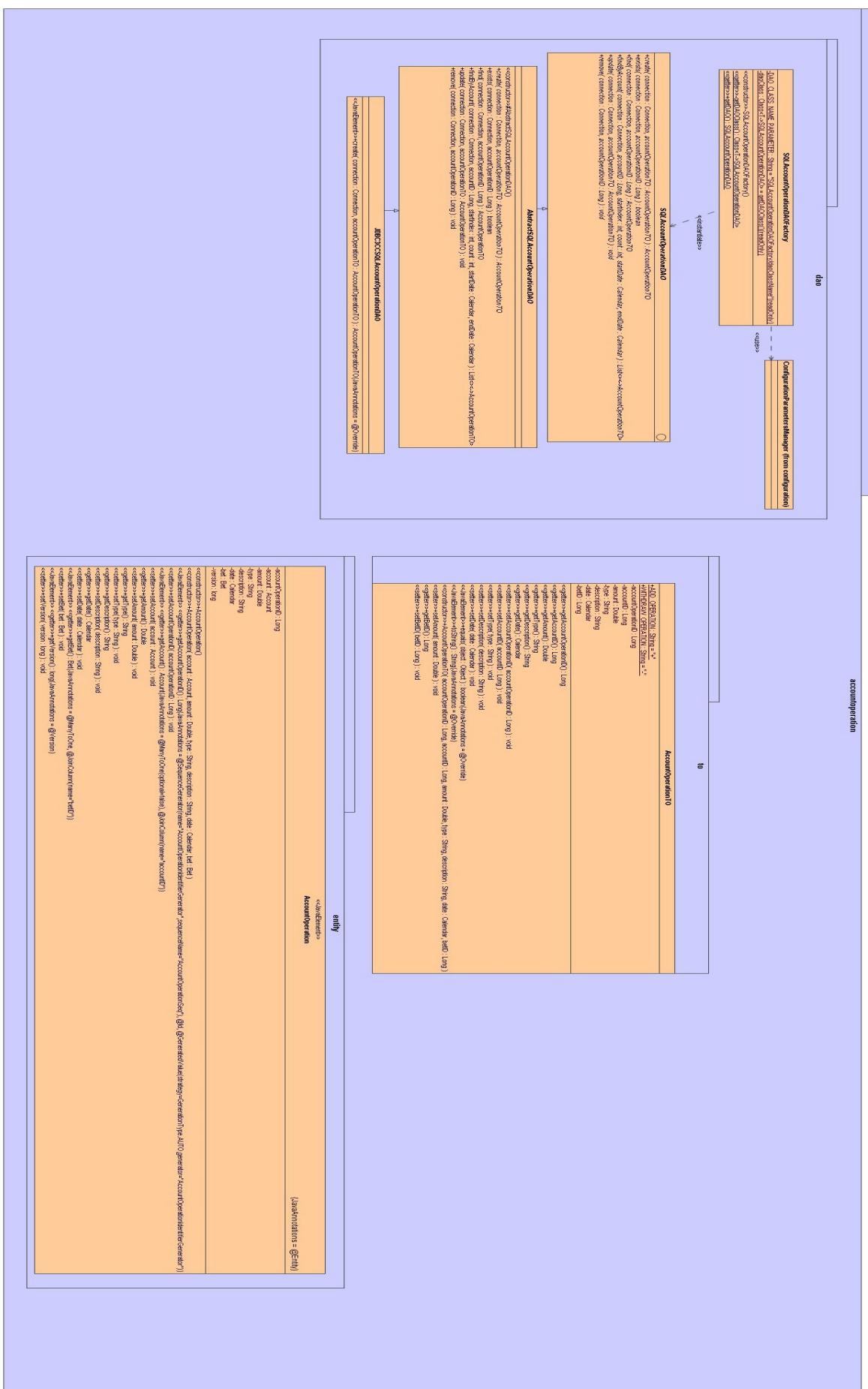
Este paquete modela las operaciones realizadas sobre las cuentas de los usuarios.

### 2.2.4.2 *Arquitectura*

En el siguiente diagrama se muestran las clases del paquete accountoperation. Por un lado, se muestran las clases del paquete dao, que implementan el patrón Data Access Object (DAO). Estas clases se usan sólo para la versión Plain de **uBet**. Existe una clase que implementa el patrón Factory y permite obtener instancias de los DAOs. Existe una clase abstracta que implementa todos los métodos que sirven para trabajar con los datos de la BD, excepto el de crear, que se implementa en tres clases distintas (una para BDs con Columnas Countador y drivers JDBC 3.0, otra para BDs con Columnas Countador sin drivers JDBC 3.0 y otra para BDs con Generadores de Identificadores).

También se muestran las clases que modelan los datos usando el patrón Transfer Object y que son los elementos persistentes del sistema.

Por último también se muestra la clase Entity usada en la versión EJB de **uBet**, que contiene los atributos de la tabla correspondiente, así como las relaciones de dicha tabla con las demás tablas usadas por la aplicación.



## 2.2.5 Paquete adminfacade

### 2.2.5.1 *Objetivo*

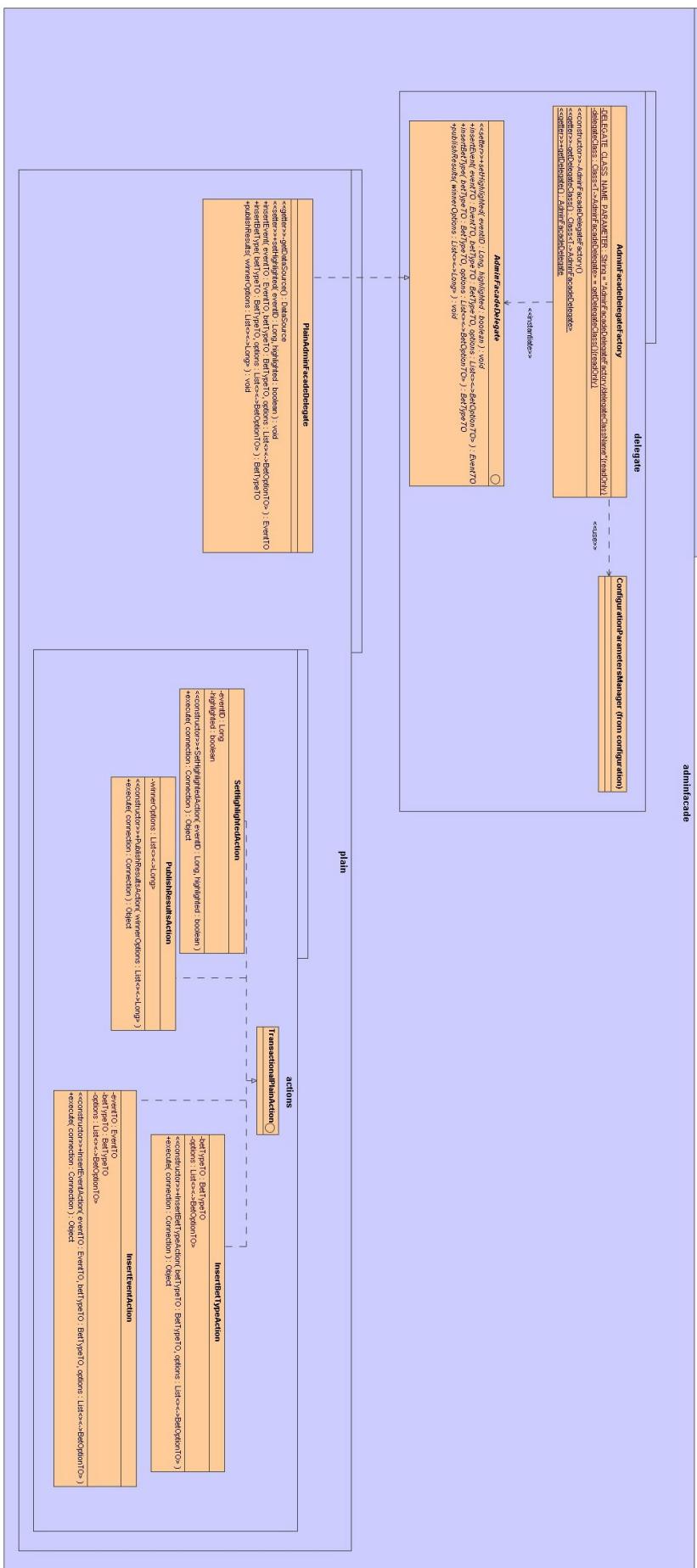
En este paquete se implementan los casos de uso del subsistema uBet-administrador.

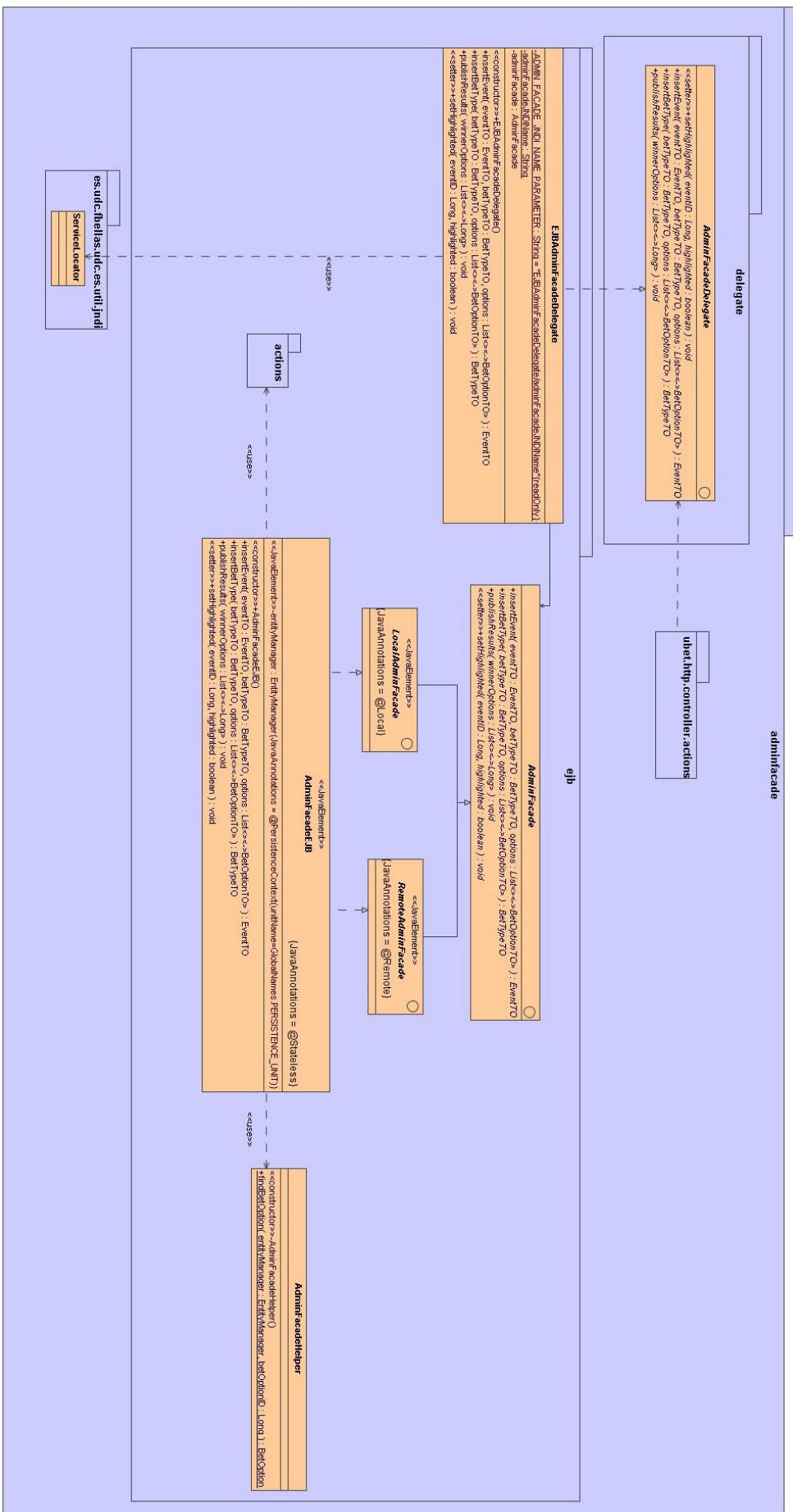
### 2.2.5.2 *Arquitectura*

En los siguientes diagramas se muestran las clases que forman parte del paquete adminfacade. En el primer diagrama se muestra el paquete delegate, en el que están las clases que implementan el patrón Business Delegate, además de haber una clase que implementa el patrón Factory para devolver instancias de la clase Facade correspondiente, la cual implementa el patrón Session Facade.

También se muestran las clases Action que implementan la clase NonTransactionalPlainAction (si no necesitan que exista integridad transaccional para su correcta ejecución) o TransactionalPlainAction (si necesitan que exista integridad transaccional para su correcta ejecución).

En el segundo diagrama se muestran las clases que se usan en la versión EJB de **uBet**. En este caso se usa una fachada sin estado, por lo tanto se trabaja sobre un proxy que interactuará con el pool de instancias de la fachada. Hace falta tener una fachada local y otra remota para que sea usada por el servidor de aplicaciones y por las pruebas de unidad, respectivamente.





## 2.2.6 Paquete bet

### 2.2.6.1 *Objetivo*

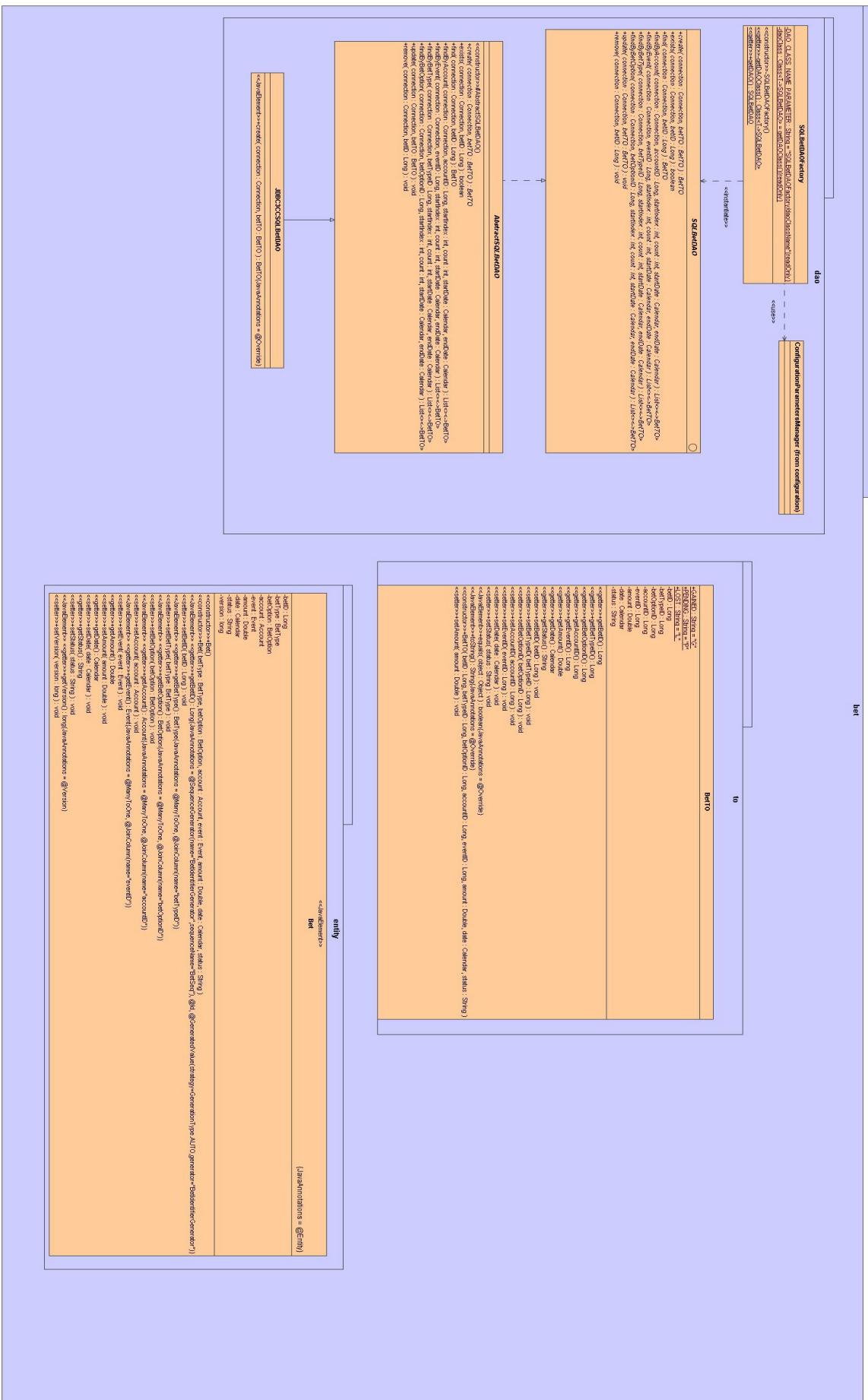
Este paquete modela las apuestas realizadas por los usuarios.

### 2.2.6.2 *Arquitectura*

En el siguiente diagrama se muestran las clases del paquete bet. Por un lado, se muestran las clases del paquete dao, que implementan el patrón Data Access Object (DAO). Estas clases se usan sólo para la versión Plain de **uBet**. Existe una clase que implementa el patrón Factory y permite obtener instancias de los DAOs. Existe una clase abstracta que implementa todos los métodos que sirven para trabajar con los datos de la BD, excepto el de crear, que se implementa en tres clases distintas (una para BDs con Columnas Countador y drivers JDBC 3.0, otra para BDs con Columnas Countador sin drivers JDBC 3.0 y otra para BDs con Generadores de Identificadores).

También se muestran las clases que modelan los datos usando el patrón Transfer Object y que son los elementos persistentes del sistema.

Por último también se muestra la clase Entity usada en la versión EJB de **uBet**, que contiene los atributos de la tabla correspondiente, así como las relaciones de dicha tabla con las demás tablas usadas por la aplicación.



## 2.2.7 Paquete betoption

### 2.2.7.1 *Objetivo*

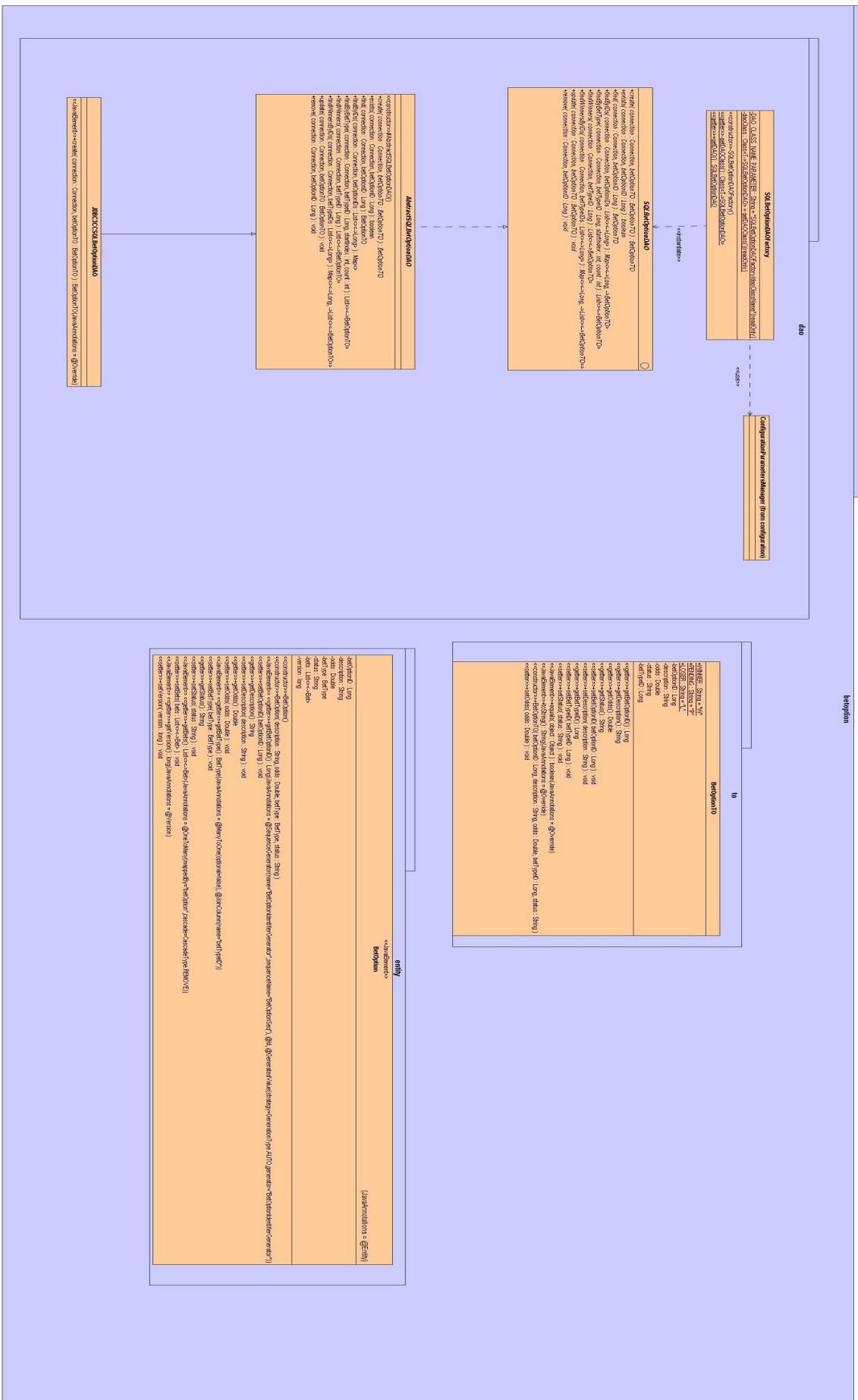
Este paquete modela las opciones de apuesta asociadas a los tipos de apuesta de los eventos.

### 2.2.7.2 *Arquitectura*

En el siguiente diagrama se muestran las clases del paquete betoption. Por un lado, se muestran las clases del paquete dao, que implementan el patrón Data Access Object (DAO). Estas clases se usan sólo para la versión Plain de **uBet**. Existe una clase que implementa el patrón Factory y permite obtener instancias de los DAOs. Existe una clase abstracta que implementa todos los métodos que sirven para trabajar con los datos de la BD, excepto el de crear, que se implementa en tres clases distintas (una para BDs con Columnas Countador y drivers JDBC 3.0, otra para BDs con Columnas Countador sin drivers JDBC 3.0 y otra para BDs con Generadores de Identificadores).

También se muestran las clases que modelan los datos usando el patrón Transfer Object y que son los elementos persistentes del sistema.

Por último también se muestra la clase Entity usada en la versión EJB de **uBet**, que contiene los atributos de la tabla correspondiente, así como las relaciones de dicha tabla con las demás tablas usadas por la aplicación.



## 2.2.8 Paquete bettype

### 2.2.8.1 *Objetivo*

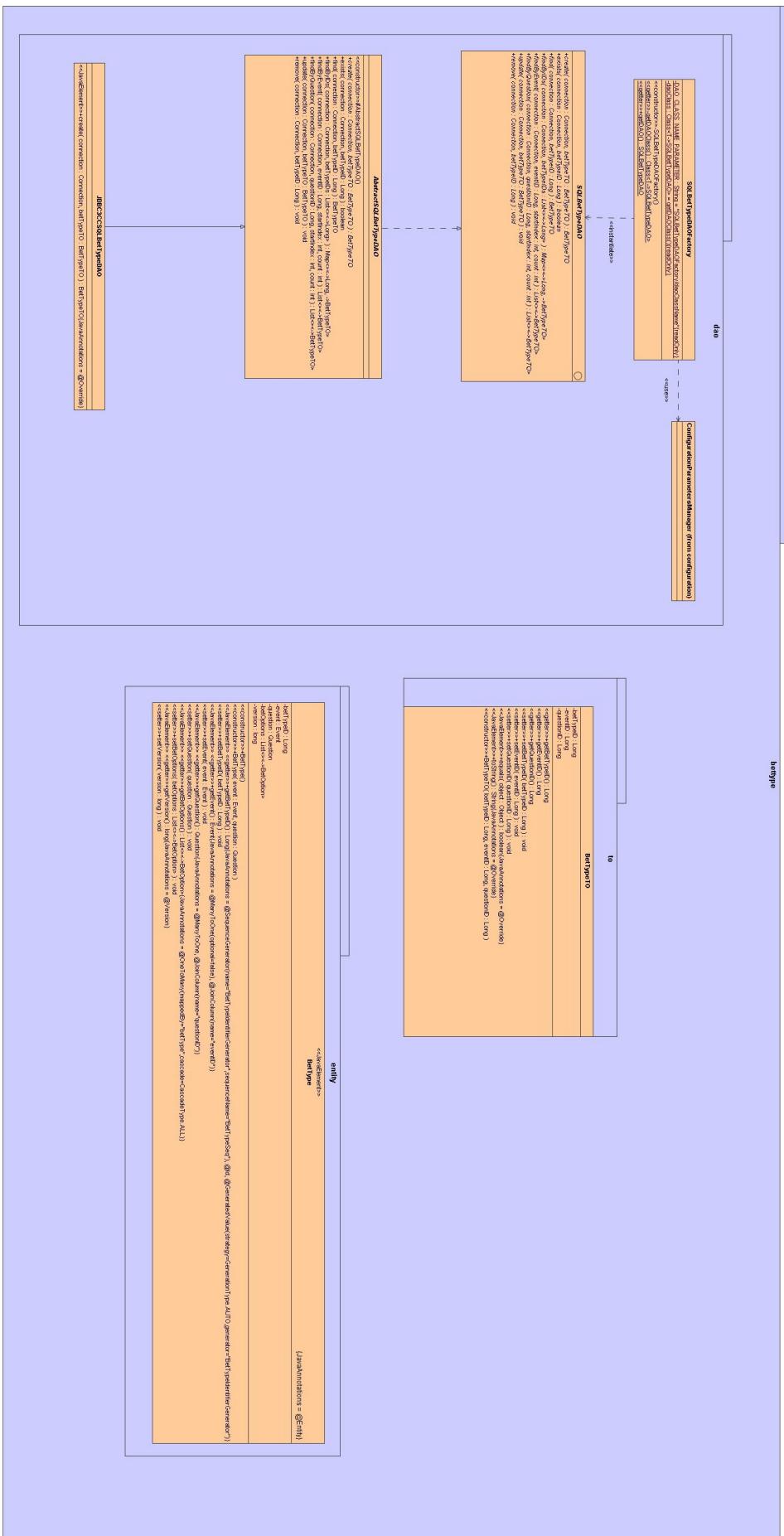
Este paquete modela los tipos de apuesta asociados a los eventos.

### 2.2.8.2 *Arquitectura*

En el siguiente diagrama se muestran las clases del paquete bettype. Por un lado, se muestran las clases del paquete dao, que implementan el patrón Data Access Object (DAO). Estas clases se usan sólo para la versión Plain de **uBet**. Existe una clase que implementa el patrón Factory y permite obtener instancias de los DAOs. Existe una clase abstracta que implementa todos los métodos que sirven para trabajar con los datos de la BD, excepto el de crear, que se implementa en tres clases distintas (una para BDs con Columnas Countador y drivers JDBC 3.0, otra para BDs con Columnas Countador sin drivers JDBC 3.0 y otra para BDs con Generadores de Identificadores).

También se muestran las clases que modelan los datos usando el patrón Transfer Object y que son los elementos persistentes del sistema.

Por último también se muestra la clase Entity usada en la versión EJB de **uBet**, que contiene los atributos de la tabla correspondiente, así como las relaciones de dicha tabla con las demás tablas usadas por la aplicación.



## 2.2.9 Paquete category

### 2.2.9.1 *Objetivo*

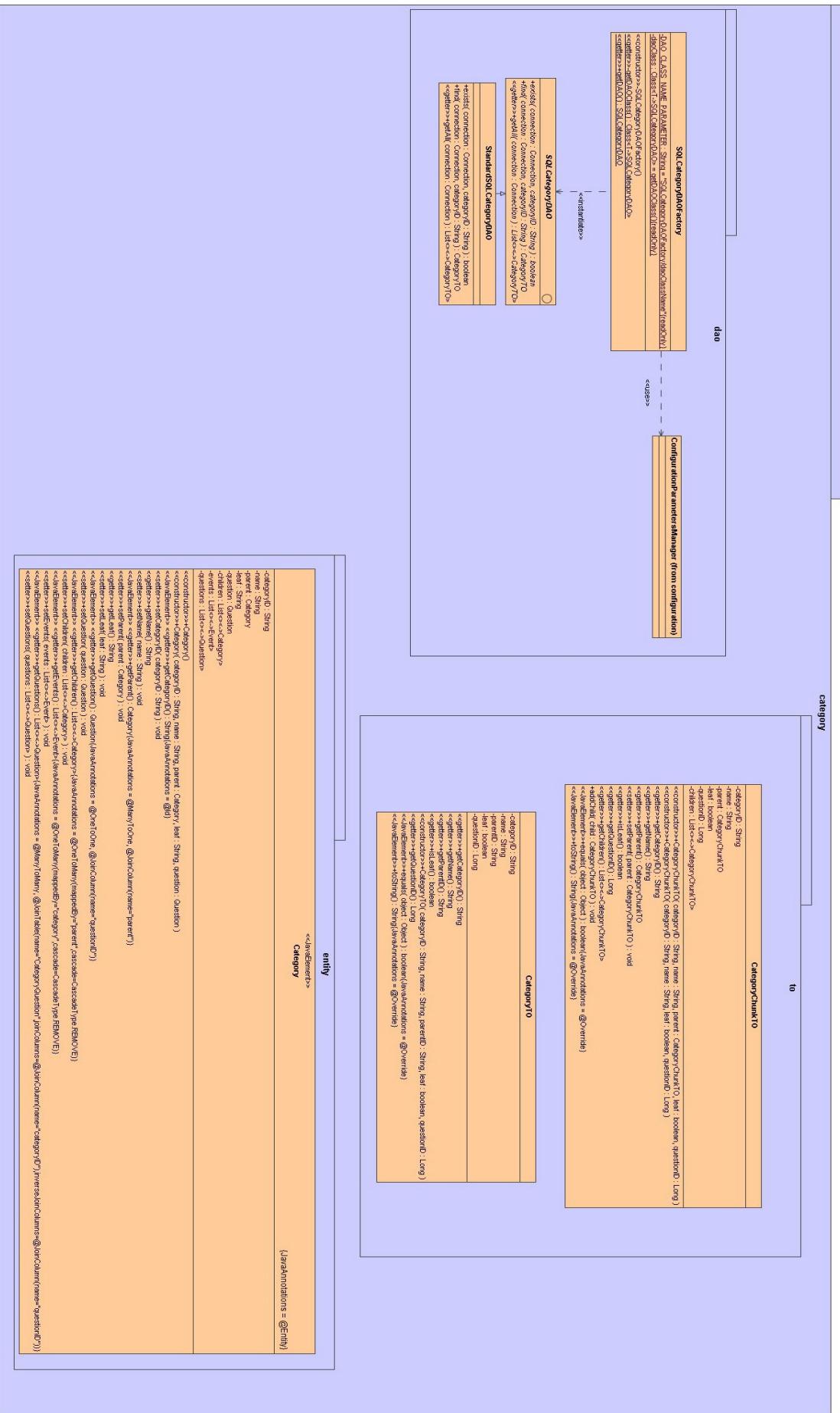
Este paquete modela las categorías a las que pertenecen los eventos.

### 2.2.9.2 *Arquitectura*

En el siguiente diagrama se muestran las clases del paquete category. Por un lado, se muestran las clases del paquete dao, que implementan el patrón Data Access Object (DAO). Estas clases se usan sólo para la versión Plain de **uBet**. Existe una clase que implementa el patrón Factory y permite obtener instancias de los DAOs. Existe una clase abstracta que implementa todos los métodos que sirven para trabajar con los datos de la BD, excepto el de crear, que se implementa en tres clases distintas (una para BDs con Columnas Countador y drivers JDBC 3.0, otra para BDs con Columnas Countador sin drivers JDBC 3.0 y otra para BDs con Generadores de Identificadores).

También se muestran las clases que modelan los datos usando el patrón Transfer Object y que son los elementos persistentes del sistema.

Por último también se muestra la clase Entity usada en la versión EJB de **uBet**, que contiene los atributos de la tabla correspondiente, así como las relaciones de dicha tabla con las demás tablas usadas por la aplicación.



## 2.2.10 Paquete categoryquestion

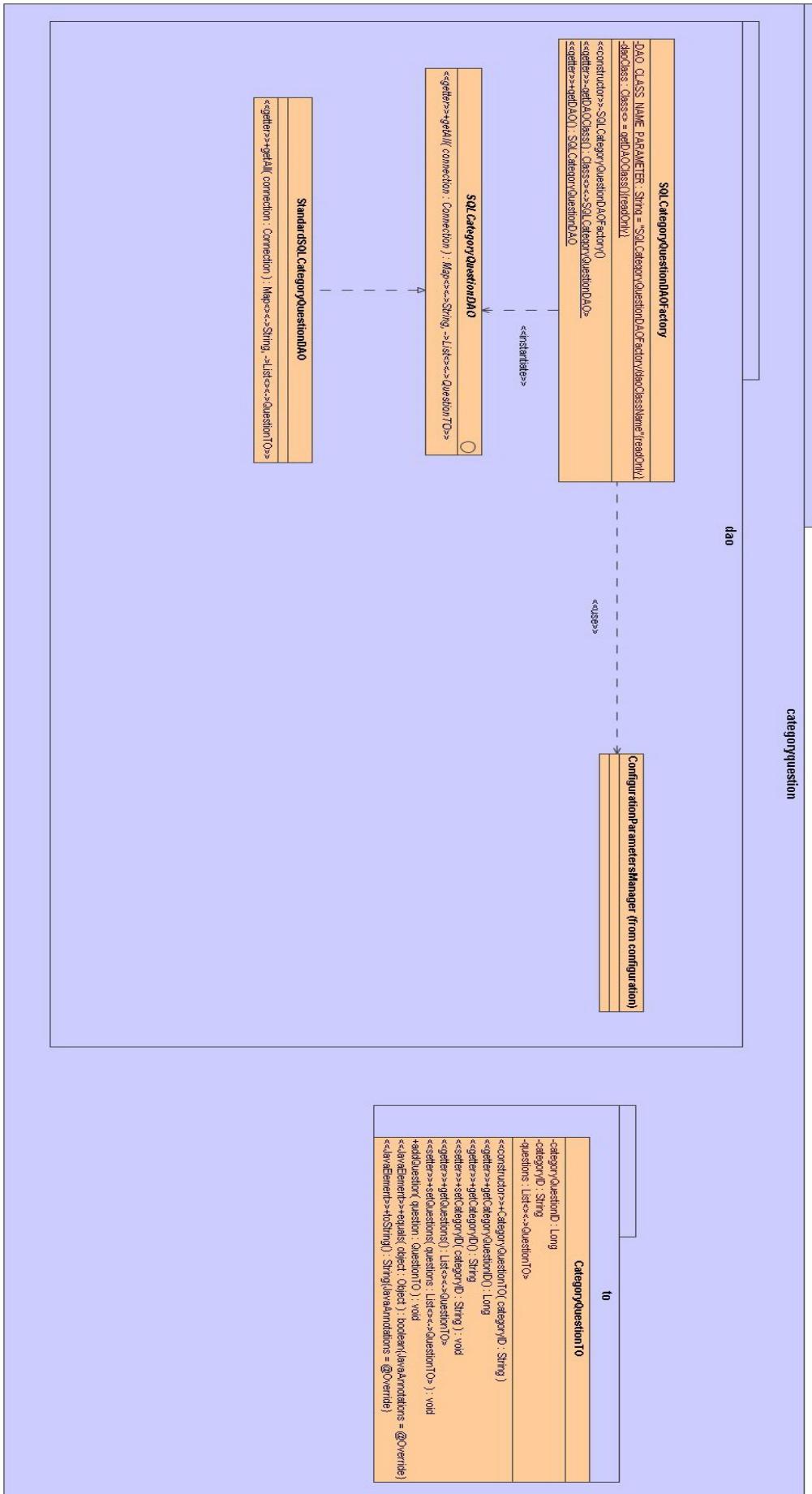
### 2.2.10.1 Objetivo

Este paquete modela las preguntas asociadas a las categorías a las que pertenecen los eventos.

### 2.2.10.2 Arquitectura

En el siguiente diagrama se muestran las clases del paquete categoryquestion. Por un lado, se muestran las clases del paquete dao, que implementan el patrón Data Access Object (DAO). Estas clases se usan sólo para la versión Plain de uBet. Existe una clase que implementa el patrón Factory y permite obtener instancias de los DAOs. Existe una clase abstracta que implementa todos los métodos que sirven para trabajar con los datos de la BD, excepto el de crear, que se implementa en tres clases distintas (una para BDs con Columnas Countador y drivers JDBC 3.0, otra para BDs con Columnas Countador sin drivers JDBC 3.0 y otra para BDs con Generadores de Identificadores).

También se muestran las clases que modelan los datos usando el patrón Transfer Object y que son los elementos persistentes del sistema.



## 2.2.11 Paquete country

### 2.2.11.1 Objetivo

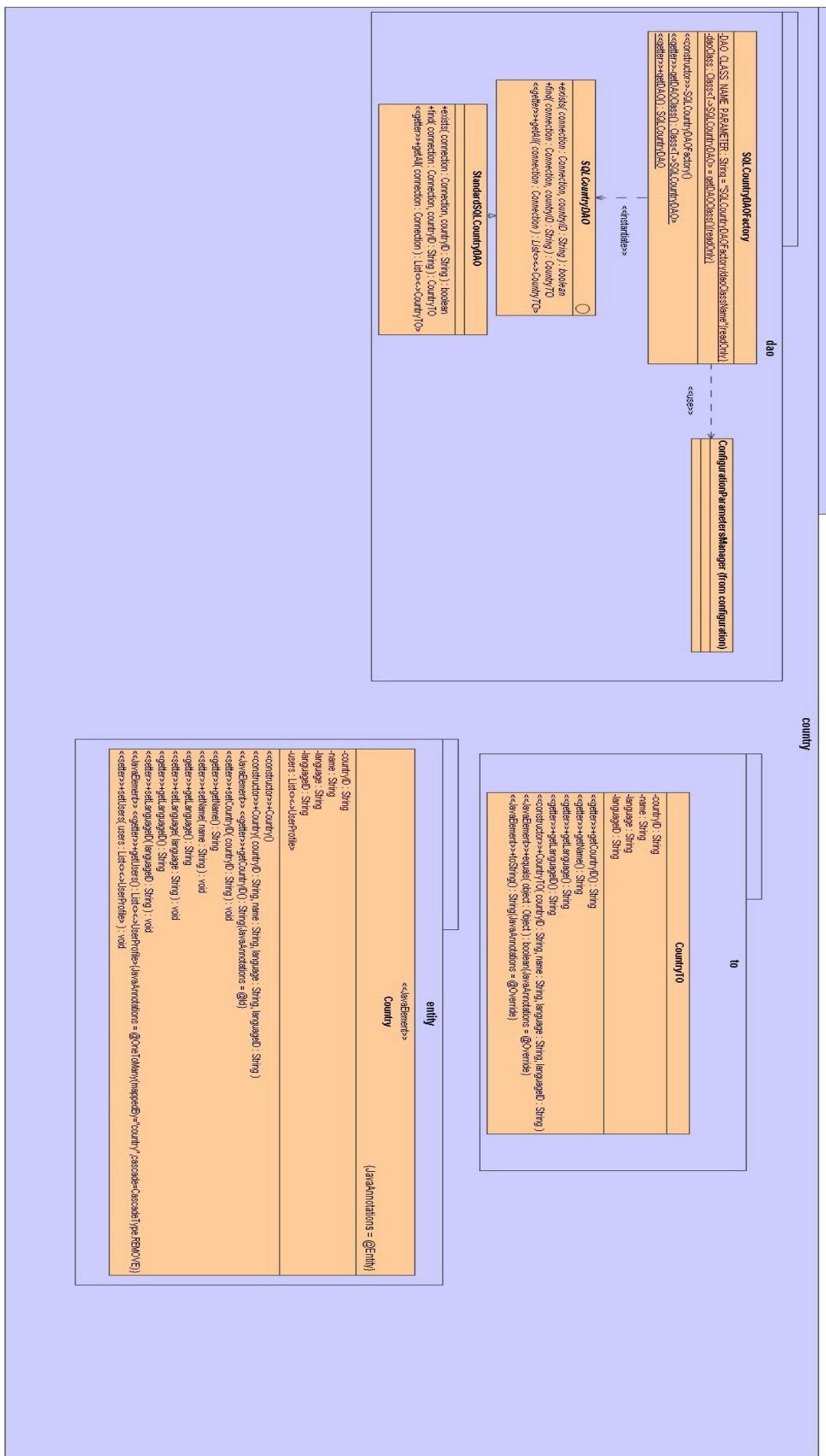
Este paquete modela los países en los que viven los usuarios.

### 2.2.11.2 Arquitectura

En el siguiente diagrama se muestran las clases del paquete country. Por un lado, se muestran las clases del paquete dao, que implementan el patrón Data Access Object (DAO). Estas clases se usan sólo para la versión Plain de **uBet**. Existe una clase que implementa el patrón Factory y permite obtener instancias de los DAOs. Existe una clase abstracta que implementa todos los métodos que sirven para trabajar con los datos de la BD, excepto el de crear, que se implementa en tres clases distintas (una para BDs con Columnas Countador y drivers JDBC 3.0, otra para BDs con Columnas Countador sin drivers JDBC 3.0 y otra para BDs con Generadores de Identificadores).

También se muestran las clases que modelan los datos usando el patrón Transfer Object y que son los elementos persistentes del sistema.

Por último también se muestra la clase Entity usada en la versión EJB de **uBet**, que contiene los atributos de la tabla correspondiente, así como las relaciones de dicha tabla con las demás tablas usadas por la aplicación.



## 2.2.12 Paquete event

### 2.2.12.1 Objetivo

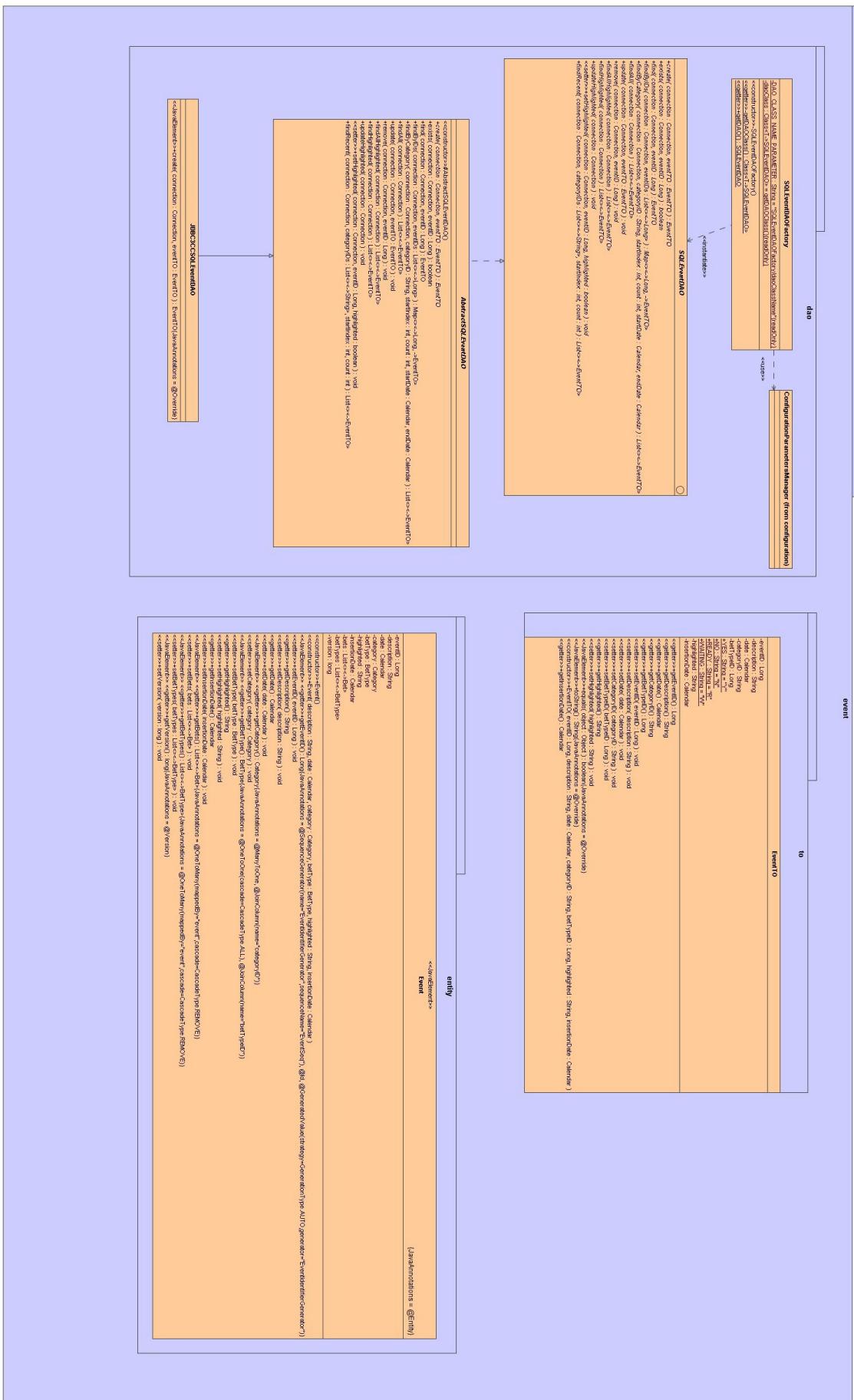
Este paquete modela los eventos deportivos.

### 2.2.12.2 Arquitectura

En el siguiente diagrama se muestran las clases del paquete event. Por un lado, se muestran las clases del paquete dao, que implementan el patrón Data Access Object (DAO). Estas clases se usan sólo para la versión Plain de **uBet**. Existe una clase que implementa el patrón Factory y permite obtener instancias de los DAOs. Existe una clase abstracta que implementa todos los métodos que sirven para trabajar con los datos de la BD, excepto el de crear, que se implementa en tres clases distintas (una para BDs con Columnas Countador y drivers JDBC 3.0, otra para BDs con Columnas Countador sin drivers JDBC 3.0 y otra para BDs con Generadores de Identificadores).

También se muestran las clases que modelan los datos usando el patrón Transfer Object y que son los elementos persistentes del sistema.

Por último también se muestra la clase Entity usada en la versión EJB de **uBet**, que contiene los atributos de la tabla correspondiente, así como las relaciones de dicha tabla con las demás tablas usadas por la aplicación.



## 2.2.13 Paquete question

### 2.2.13.1 Objetivo

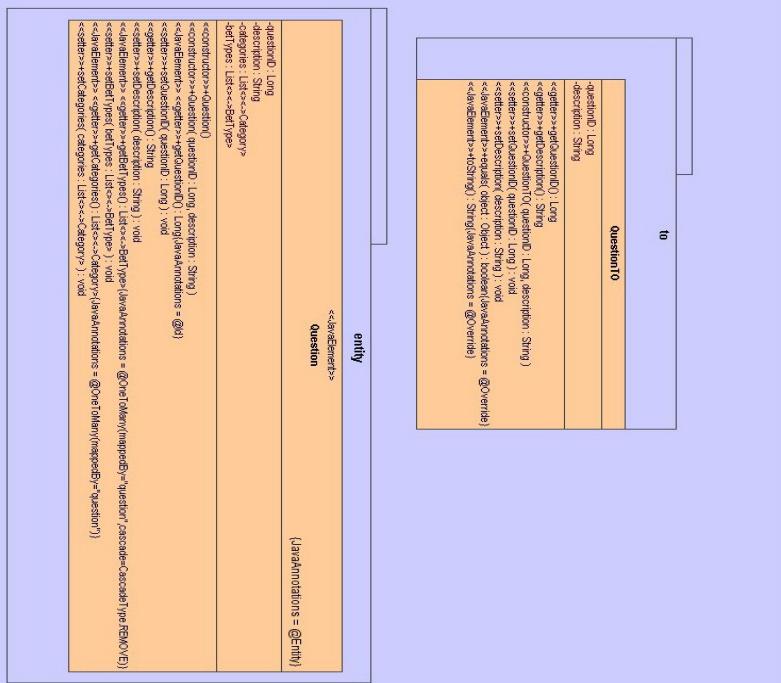
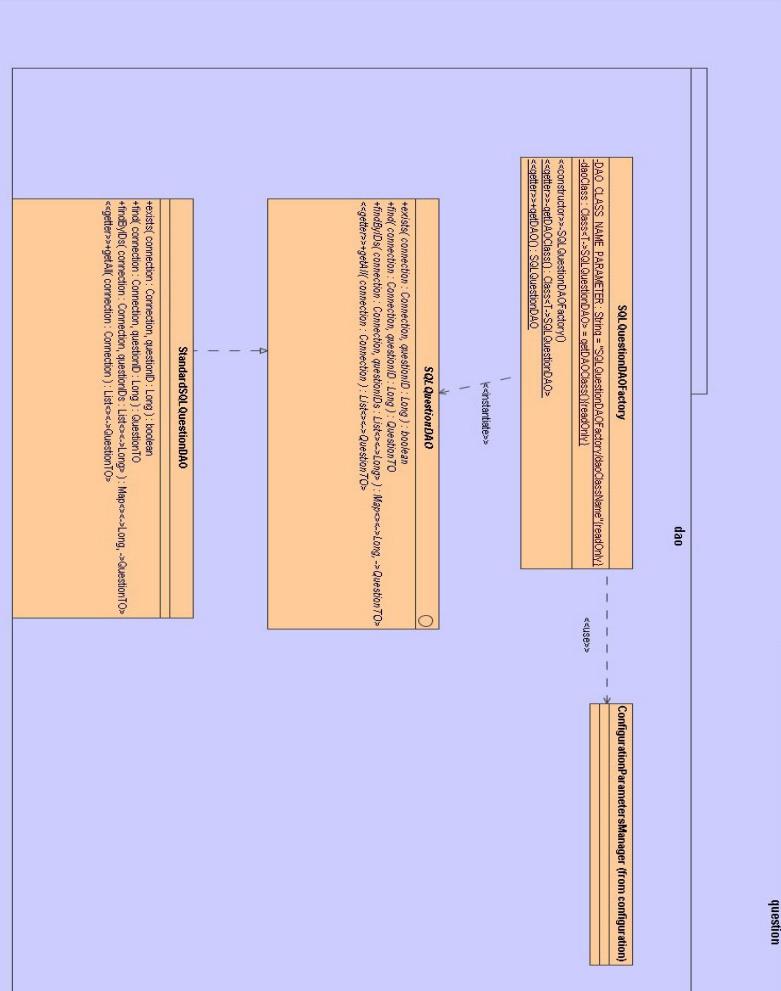
Este paquete modela las preguntas por las que se puede apostar.

### 2.2.13.2 Arquitectura

En el siguiente diagrama se muestran las clases del paquete question. Por un lado, se muestran las clases del paquete dao, que implementan el patrón Data Access Object (DAO). Estas clases se usan sólo para la versión Plain de **uBet**. Existe una clase que implementa el patrón Factory y permite obtener instancias de los DAOs. Existe una clase abstracta que implementa todos los métodos que sirven para trabajar con los datos de la BD, excepto el de crear, que se implementa en tres clases distintas (una para BDs con Columnas Countador y drivers JDBC 3.0, otra para BDs con Columnas Countador sin drivers JDBC 3.0 y otra para BDs con Generadores de Identificadores).

También se muestran las clases que modelan los datos usando el patrón Transfer Object y que son los elementos persistentes del sistema.

Por último también se muestra la clase Entity usada en la versión EJB de **uBet**, que contiene los atributos de la tabla correspondiente, así como las relaciones de dicha tabla con las demás tablas usadas por la aplicación.



## 2.2.14 Paquete searchfacade

### 2.2.14.1 Objetivo

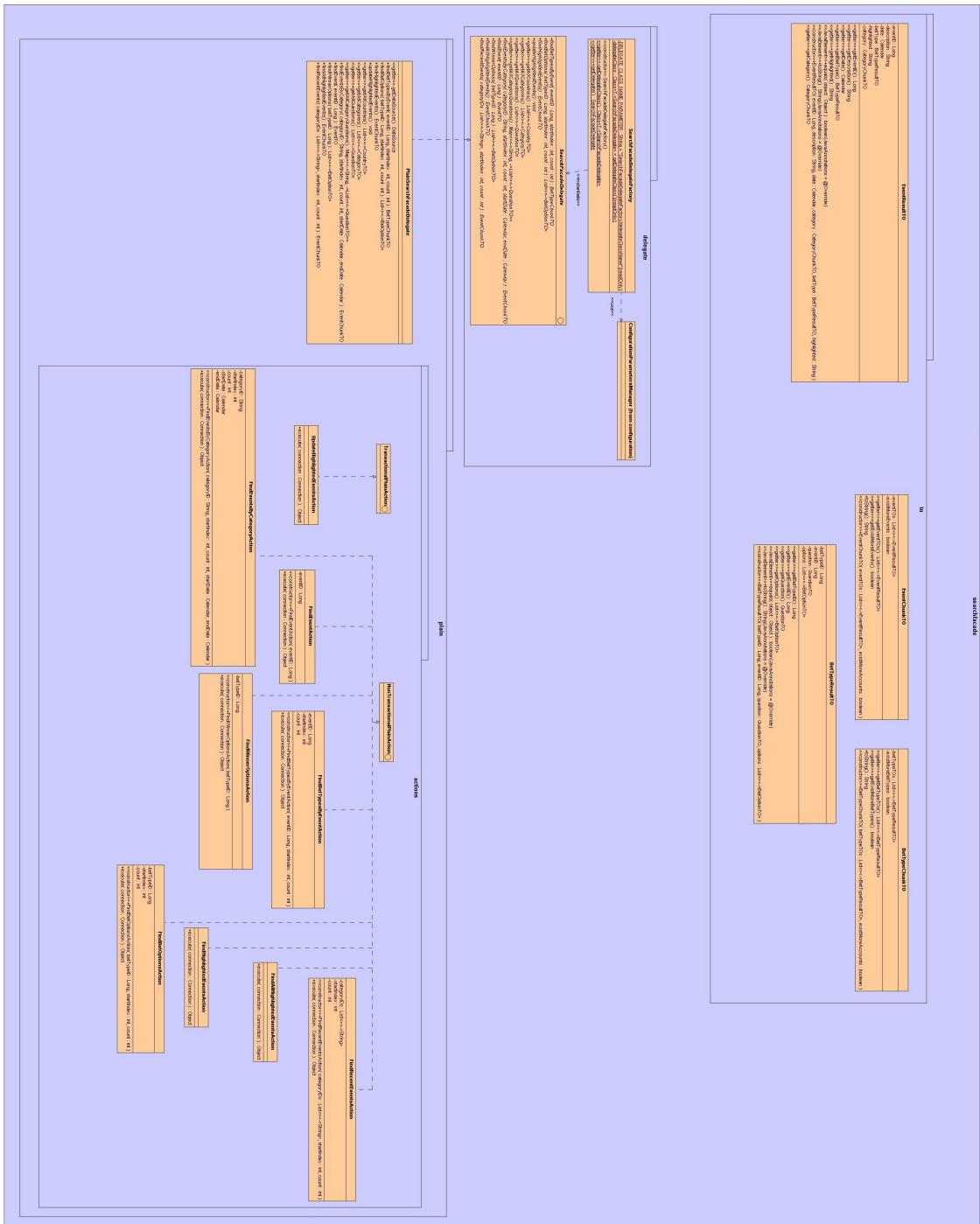
En este paquete se implementan los casos de uso relativos al subsistema uBet-búsqueda.

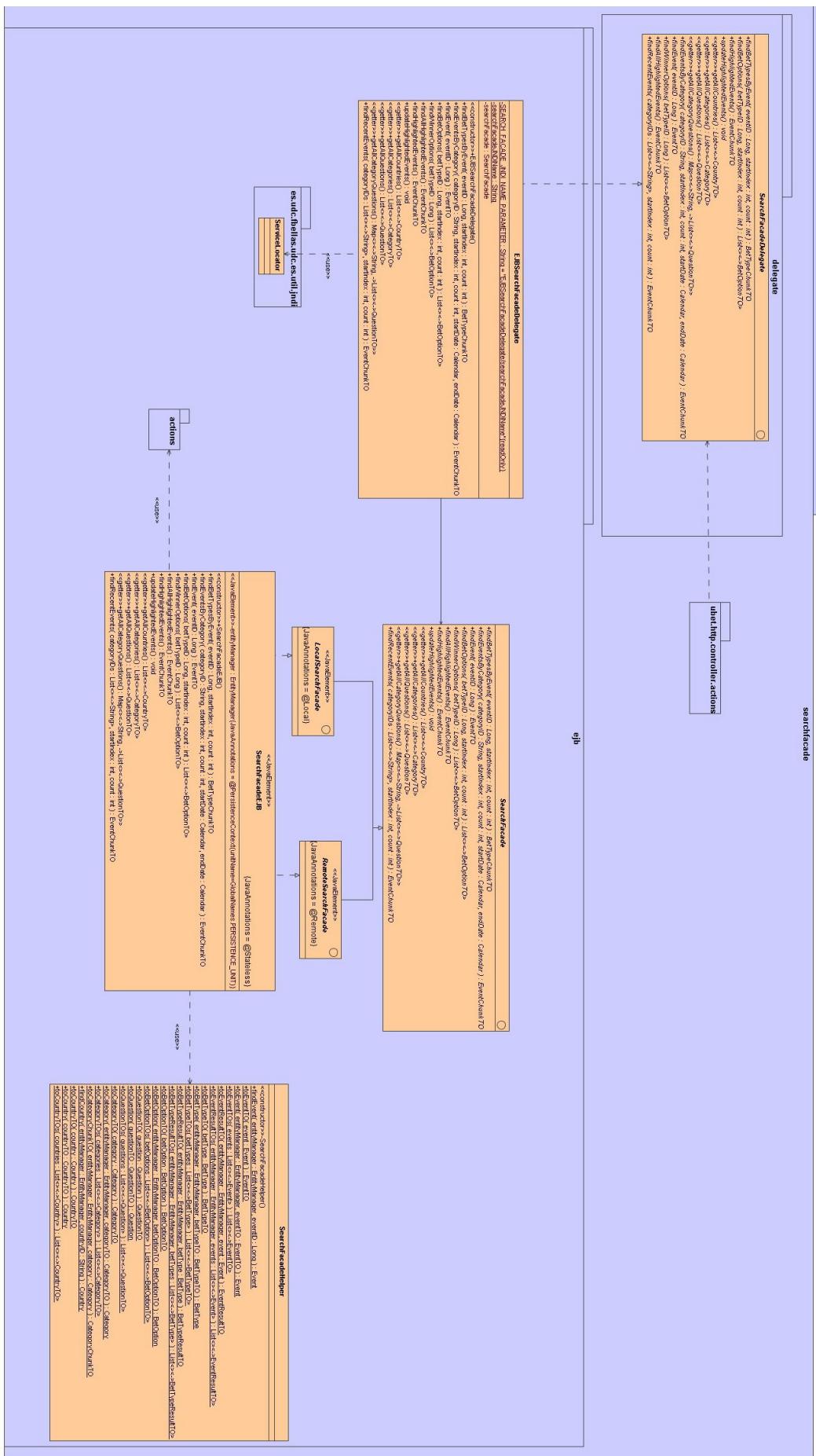
### 2.2.14.2 Arquitectura

En los siguientes diagramas se muestran las clases que forman parte del paquete searchfacade. En el primer diagrama se muestra el paquete delegate, en el que están las clases que implementan el patrón Business Delegate, además de haber una clase que implementa el patrón Factory para devolver instancias de la clase Facade correspondiente, la cual implementa el patrón Session Facade.

También se muestran las clases Action que implementan la clase NonTransactionalPlainAction (si no necesitan que exista integridad transaccional para su correcta ejecución) o TransactionalPlainAction (si necesitan que exista integridad transaccional para su correcta ejecución).

En el segundo diagrama se muestran las clases que se usan en la versión EJB de **uBet**. En este caso se usa una fachada sin estado, por lo tanto se trabaja sobre un proxy que interactuará con el pool de instancias de la fachada. Hace falta tener una fachada local y otra remota para que sea usada por el servidor de aplicaciones y por las pruebas de unidad, respectivamente.





## 2.2.15 Paquete userfacade

### 2.2.15.1 Objetivo

En este paquete se implementan los casos de uso relativos al subsistema uBet-usuario.

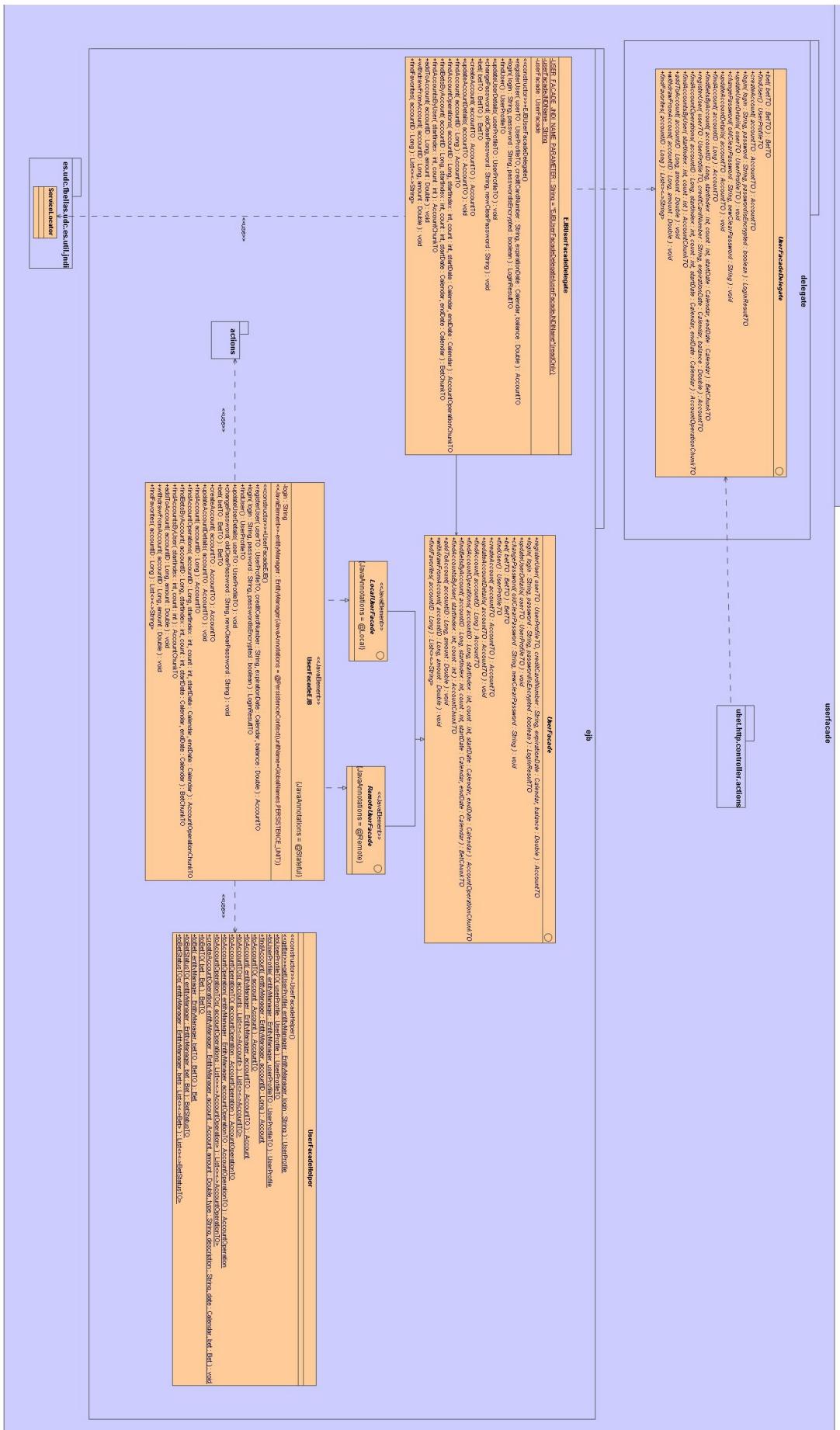
### 2.2.15.2 Arquitectura

En los siguientes diagramas se muestran las clases que forman parte del paquete userfacade. En el primer diagrama se muestra el paquete delegate, en el que están las clases que implementan el patrón Business Delegate, además de haber una clase que implementa el patrón Factory para devolver instancias de la clase Facade correspondiente, la cual implementa el patrón Session Facade.

También se muestran las clases Action que implementan la clase NonTransactionalPlainAction (si no necesitan que exista integridad transaccional para su correcta ejecución) o TransactionalPlainAction (si necesitan que exista integridad transaccional para su correcta ejecución).

En el segundo diagrama se muestran las clases que se usan en la versión EJB de **uBet**. En este caso se usa una fachada con estado, por lo tanto se crea una instancia de la clase de implementación cuando el cliente busca la fachada por JNDI y el Proxy que obtiene la clase cliente está ligado a esa instancia. Hace falta tener una fachada local y otra remota para que sea usada por el servidor de aplicaciones y por las pruebas de unidad, respectivamente.





## 2.2.16 Paquete userprofile

### 2.2.16.1 Objetivo

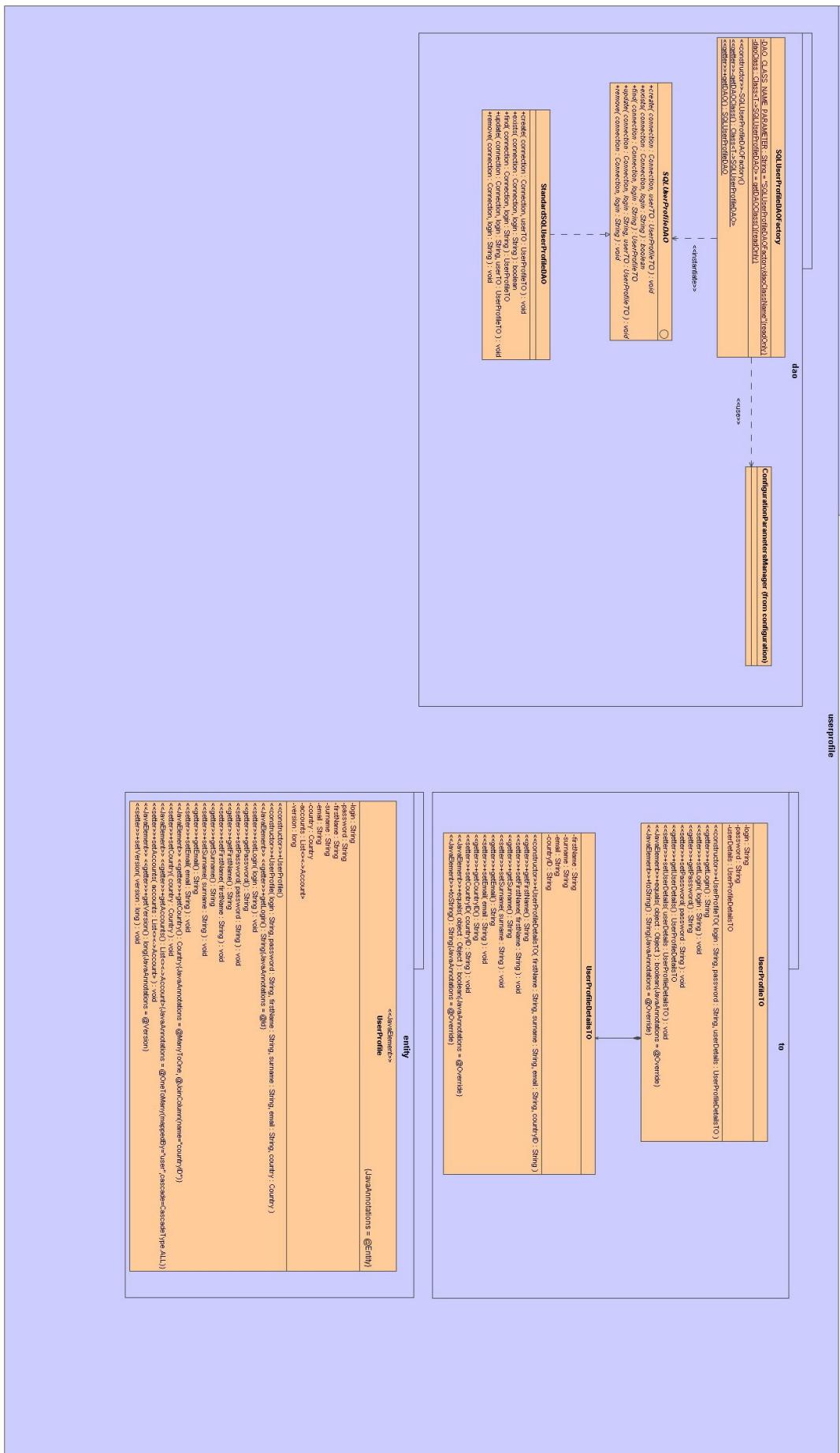
Este paquete modela los datos de los usuarios registrados.

### 2.2.16.2 Arquitectura

En el siguiente diagrama se muestran las clases del paquete userprofile. Por un lado, se muestran las clases del paquete dao, que implementan el patrón Data Access Object (DAO). Estas clases se usan sólo para la versión Plain de **uBet**. Existe una clase que implementa el patrón Factory y permite obtener instancias de los DAOs. Existe una clase abstracta que implementa todos los métodos que sirven para trabajar con los datos de la BD, excepto el de crear, que se implementa en tres clases distintas (una para BDs con Columnas Countador y drivers JDBC 3.0, otra para BDs con Columnas Countador sin drivers JDBC 3.0 y otra para BDs con Generadores de Identificadores).

También se muestran las clases que modelan los datos usando el patrón Transfer Object y que son los elementos persistentes del sistema. Los detalles personales del usuario (nombre, apellidos, país, etc.) se guardan en una clase y el login y la password se guardan en otra.

Por último también se muestra la clase Entity usada en la versión EJB de **uBet**, que contiene los atributos de la tabla correspondiente, así como las relaciones de dicha tabla con las demás tablas usadas por la aplicación.



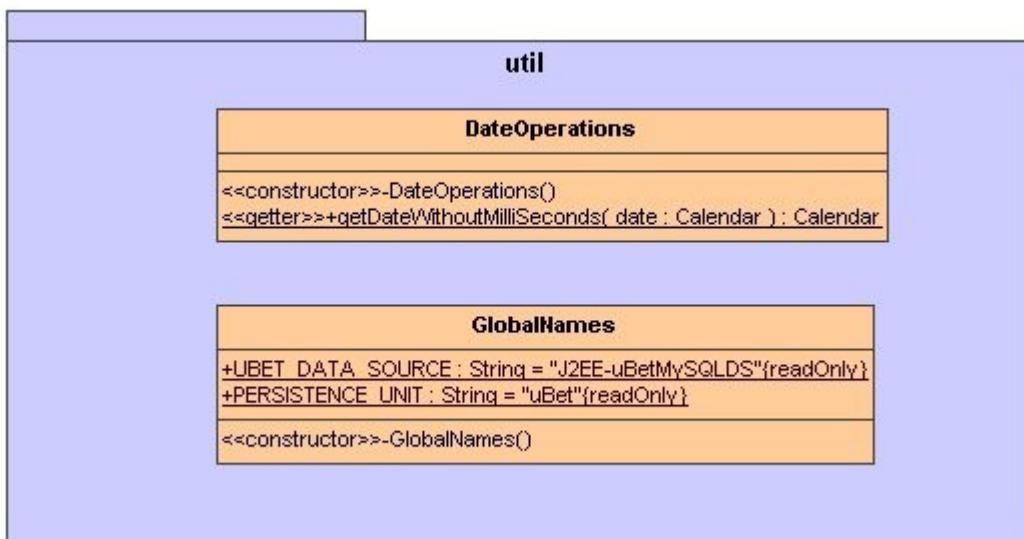
## 2.2.17 Paquete util

### 2.2.17.1 Objetivo

En este paquete se engloban ciertas clases que proporcionan apoyo a la aplicación.

### 2.2.17.2 Arquitectura

En el siguiente diagrama se muestran las clases que forman parte del paquete util. La clase DateOperations permite quitar los milisegundos a las fechas introducidas en el sistema para evitar problemas. Por otra parte, la clase GlobalNames contiene una serie de constantes que se utilizan de manera estática desde otros puntos del programa.



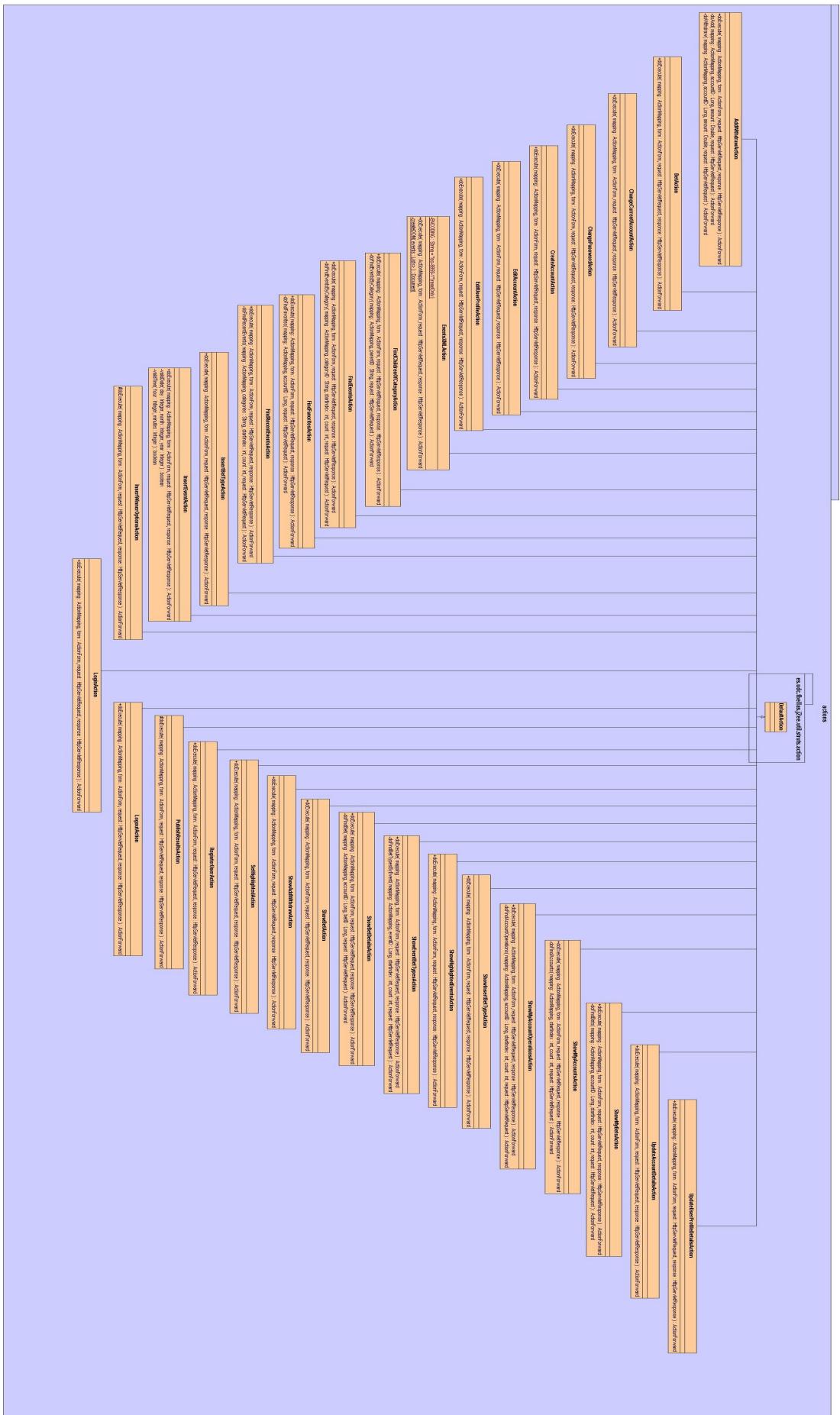
## **2.2.18 Paquete controller.actions**

### **2.2.18.1 *Objetivo***

Este paquete contiene las diferentes clases acción que implementan la funcionalidad de la capa controlador de la aplicación.

### **2.2.18.2 *Arquitectura***

En el siguiente diagrama se muestran todas las clases que forman parte del paquete controller.actions. Todas las clases acción descenden de la clase DefaultAction que proporciona el framework de Struts [13]. En general, cada clase acción está asociada de alguna manera con algún link de la aplicación web, aunque la clase EventsXMLAction, se usa tanto para devolver los eventos destacados como los más recientes en forma de XML para que sean parseados y mostrados por una aplicación externa, que se los pide por medio de una petición a una URL concreta.



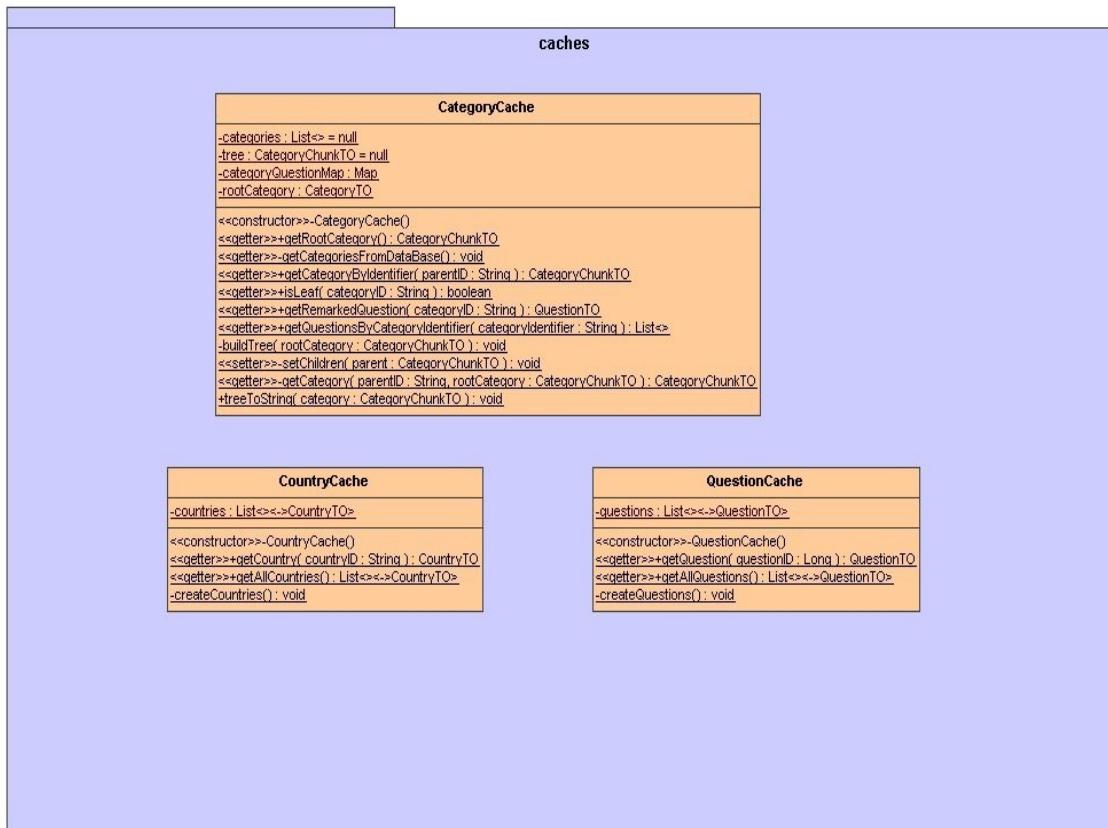
## 2.2.19 Paquete controller.caches

### 2.2.19.1 Objetivo

Este paquete tiene las clases que implementan las cachés de categorías, países y preguntas. Estas cachés almacenan en la memoria del contenedor web la lista de categorías, países y preguntas almacenadas en la base de datos, de manera que agilizan el acceso a estos datos.

### 2.2.19.2 Arquitectura

En el siguiente diagrama se muestran las clases que forman parte del paquete controller.caches. Estas clases sirven para agilizar el acceso a datos que no son actualizados a menudo, por lo que pueden ser cacheados.



## **2.2.20 Paquete controller.frontcontroller**

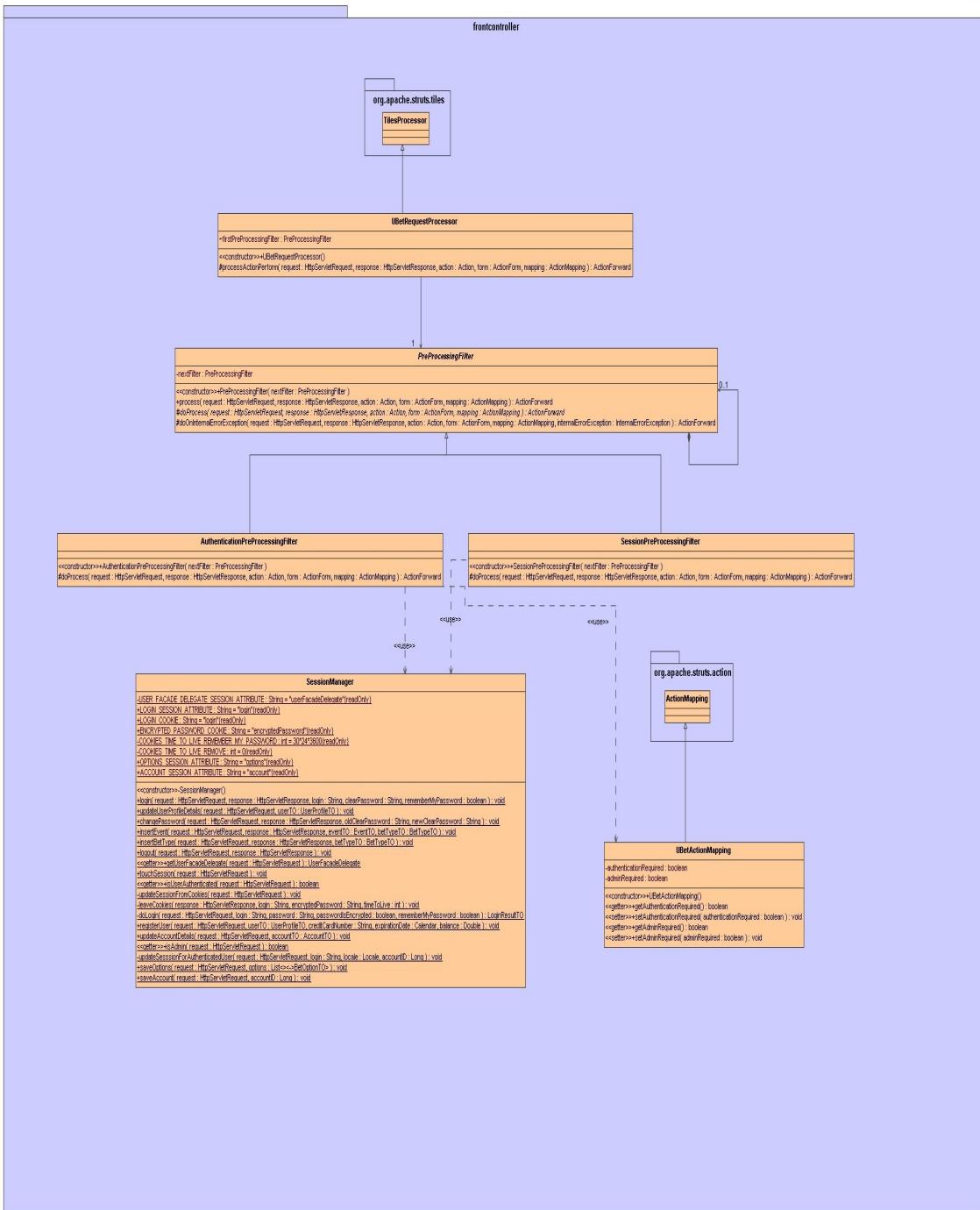
### **2.2.20.1 *Objetivo***

Este paquete contiene las clases que implementan un filtro de preprocesamiento sobre todas las peticiones que se hagan sobre la aplicación web. Además, incorpora una versión propia del ActionMapping de Struts para trabajar con los permisos de los usuarios normales y del administrador.

### **2.2.20.2 *Arquitectura***

Las clases que implementan el filtro mencionado anteriormente siguen el patrón Cadena de Responsabilidad. Cada clase PreProcessingFilter almacena cuál es el siguiente filtro que debe ser invocado para procesar la request. Finalmente, cada filtro devuelve el ActionForward adecuado.

La clase UBetActionMapping es una versión del ActionMapping de Struts que almacena si la acción actual requiere permisos de usuario o permisos de administrador.



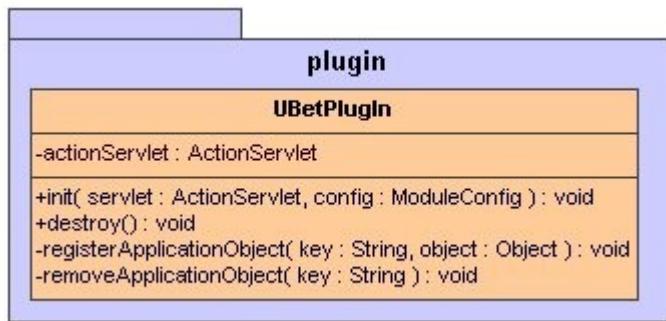
## 2.2.21 Paquete controller.plugin

### 2.2.21.1 *Objetivo*

Este paquete contiene un plugin para validar las fechas que se introducen en la aplicación web.

### 2.2.21.2 *Arquitectura*

UBetPlugIn es una clase que implementa el interfaz PlugIn. Esta clase hace uso del objeto aplicación DataRanges, que define varias listas con los posibles valores para el día, mes y año de las fechas.



## 2.2.22 Paquete controller.session

### 2.2.22.1 *Objetivo*

Este paquete contiene la clase SessionManager que es invocada por las clases acción del controlador que requieren el manejo de los objetos almacenados en la sesión y las cookies.

### 2.2.22.2 *Arquitectura*

A continuación se muestra la clase SessionManager cuyos métodos invocan las operaciones de las fachadas del modelo que modifican datos en la sesión. El resto de los métodos que contiene hace referencia al mantenimiento de la sesión y de la cookies.



## 2.3 Subsistema WastingMoney

### 2.3.1 Objetivo

El objetivo de este subsistema es modelar una aplicación web externa al portal de apuestas que permita acceder a ciertos datos de éste a través de unas URLs proporcionadas por el portal de apuestas que devuelven los datos en XML [9]. Los datos recibidos en XML [9], se formatean utilizando XSLT [7] en el caso de los eventos en portada y los tags de XML [9] que proporciona JSTL [8].

### 2.3.2 Arquitectura global

El diagrama que muestra la arquitectura global de paquetes se puede ver a continuación:



### **2.3.3 Paquete controller.actions**

#### **2.3.3.1 *Objetivo***

Este paquete implementa las acciones del controlador para realizar las peticiones a **uBet**.

#### **2.3.3.2 *Arquitectura***

En el siguiente diagrama se muestran todas las clases que forman parte del paquete controller.actions. Todas las clases acción descenden de la clase DefaultAction que proporciona el framework de Struts [13].

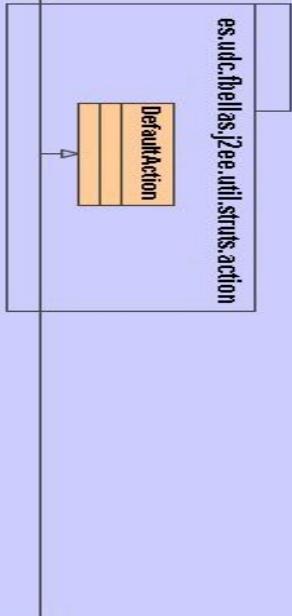
La clase ShowHighlightedEventsAction, primero hace una petición a una URL proporcionada por el sitio web de apuestas deportivas y éste le devuelve un documento XML. Luego parsea el documento devuelto usando XSLT [7] y una vez parseado le pasa el resultado en la request a la página jspx para que muestre los eventos.

La clase ShowRecentEventsAction construye la URL para hacer una petición al sitio web de apuestas deportivas, poniendo como parámetro los identificadores de las categorías a la que pertenecen los eventos que le interesa mostrar, y le pasa la URL y el path donde se encuentra el archivo XSL a la página jspx para que muestre los eventos tras haber parseado el XML que le devuelve **uBet**.

actions

```
es.udc.flwlls;jee.util.struts.action
```

**DefaultAction**



**ShowhighlightedEventsAction**

```
STYLESHEET: String = "#events .list{readOnly:1}
 SOURCE: String = "http://localhost:8080/EJBButEvents/XML/doAction=HIGHLIGHTED/readOnly"
```

**ShowRecentEventsAction**

```
STYLESHEET: String = "#events .list{readOnly:1}
 SOURCE: String = "http://localhost:8080/EJBButEvents/XML/doAction=RECENT/readOnly"
```

```
+doExecute(mapping: ActionMapping, form: ActionForm, request: HttpServletRequest, response: HttpServletResponse): ActionForward
```

## 2.3.4 Paquete controller.frontcontroller

### 2.3.4.1 *Objetivo*

Este paquete contiene las clases que implementan un filtro de preprocesamiento sobre todas las peticiones que se hagan sobre **Wasting Money**. Además, incorpora una versión propia para trabajar con los permisos de los usuarios normales y de los registrados.

### 2.3.4.2 *Arquitectura*

Como este paquete tiene una estructura análoga al paquete controller.frontcontroller del subsistema uBet, no creemos necesario poner el diagrama UML.

## 3 IMPLEMENTACIÓN

### 3.1 *Estructura de la distribución*

Tras descomprimir el archivo J2EE-uBet-src-1.0.(tar.gz ó zip) se crea una carpeta que contiene los siguientes directorios: Diagramas, Jars, magicdraw, PropertiesConfiguration, Scripts y Subsystems. También se incluye un fichero README al estilo del proporcionado en los ejemplos de la asignatura, así como un fichero PDF que contiene esta memoria.

En el directorio Diagramas, se encuentran los distintos diagramas realizados en MagicDraw utilizados para la realización de esta memoria.

En el directorio Jars, se encuentran los archivos .jar correspondientes a las clases de utilidad proporcionadas con los ejemplos de la asignatura y reutilizadas en la realización de esta práctica.

En el directorio PropertiesConfiguration se encuentran los archivos de configuración .properties necesarios para la realización de las pruebas de unidad (**JUnit** [6]).

Dentro del directorio Scripts se encuentran los archivos necesarios para preparar la ejecución del subsistema generando las variables de entorno necesarias.

Dentro del directorio Subsystems se encuentran dos directorios: uBet y WastingMoney (correspondientes a los dos subsistemas). Además, se incluyen tres ficheros XML [9]: build.XML [9] (utilizado por ant), CommonPathReferences.XML [9] y CommonProperties.XML [9] (utilizados por el build.XML [9]).

Dentro del directorio uBet, se encuentran los subdirectorios: Scripts (con los scripts de creación de tablas de la base de datos, tanto para MySQL como para PostgreSQL) y Sources (con los ficheros fuente del subsistema uBet, además de un fichero build.XML [9]).

Dentro del directorio WastingMoney, se encuentra el subdirectorio Sources (con los ficheros fuente del subsistema WastingMoney, además de un fichero build.XML [9]).

### **3.2 Instrucciones de compilación**

Tras haber leído el fichero README proporcionado y, tras haber instalado y configurado correctamente las aplicaciones necesarias, para compilar la aplicación con ant basta con poner la siguiente línea de comandos estando dentro del directorio J2EE-uBet-src-1.0/Subsystems:

*ant all initdb*

Para ejecutar la aplicación, bastaría con arrancar el Tomcat o el JBoss (versión Plain o EJB respectivamente) y abrir un navegador introduciéndole la siguiente dirección:

*<http://localhost:8080/PlainuBet>* (versión Plain)  
ó  
*<http://localhost:8080/EJBuBet>* (versión EJB)

Para ejecutar las pruebas de unidad de **uBet** primero hay que iniciar el servidor de base de datos (ya sea MySQL o PostgresSQL) y el JBoss (en el caso de querer probar los casos de uso de la versión EJB) y luego, tras haber modificado correctamente el fichero ConfigurationParameters.properties que se encuentra en el directorio J2EE-uBet-src-1.0/Properties Configuration, bastaría con poner la siguiente línea de comandos estando dentro del directorio J2EE-uBet-src-1.0/Subsystems/uBet/Sources:

*ant test*

## 4 INSTALACIÓN

Para instalar y ejecutar el software basta con seguir las instrucciones incluidas en el fichero README proporcionado, que se incluye a continuación:

```
***** J2EE-uBet v1.0 *****

Jesús Ángel Pérez-Roca Fernández (djalma_fd@yahoo.es)
José Antonio Pereira Suárez (ja_pereira@eresmas.com)

Contents
-----
1. Software requirements
2. Basic installation and configuration of software packages
3. Environment variables
4. Building and initialization of the database
5. Distribution
6. Ant files
7. Executing uBet
8. Executing Wasting Money
9. Distributed environment
10. Known problems with the development environment
11. Bug reports and suggestions for improvement
12. Documentation

1. Software requirements
-----
This software has been developed and tested with:
* Operating system. It should be possible to compile and execute this
distribution in Unix-like operating systems and any version of
MS-Windows. In particular, we have tested the following combinations:
- Ubuntu 5.10 + {MySQL, PostgreSQL}.
- MS Windows XP + {MySQL, PostgreSQL}.

All scripts are provided both in ".sh" and ".bat" files.

* An implementation of J2SE 5.0.

We have used Sun's JDK 1.5.0 (Update 4) on Linux and MS-Windows.

* ant 1.6.5.

To manage the project (compile, generate JavaDoc, jars, and so on).

* MySQL Community Edition 4.1.14 and PostgreSQL 8.0.3. Both of them are
available for Unix-like and MS-Windows operating systems.

We have used the following drivers:
- mysql-connector-java-3.1.10-bin.jar (JDBC 3 driver) with
MySQL 4.1.14 Community Edition.

- postgresql-8.0-312.jdbc3.jar (JDBC 3 driver) with PostgreSQL 8.0.3.

The use of another relational database, assuming that provides a decent
SQL, should be very easy. Probably it should only require to port SQL
scripts for table creation and modify the configuration. Otherwise,
some interfaces should be implemented.

* An application server conforming to Servlet 2.4, JSP 2.0 and EJB 3.0.

We have used Jakarta Tomcat 5.5.9 (servlet and JSP container) and
JBoss 4.0.4RC1 (J2EE application server).

* An implementation of the JSP Standard Tag Library (JSTL) 1.1.

We have used Jakarta Standard TagLibs 1.1.2.

* Jakarta Struts 1.2.7. A Model-View-Controller framework for JSP.

* To read the sources, you should use a text editor configured with
```

\*tab size\* to \*four\* blanks.

## 2. Basic installation and configuration of software packages

---

Most software packages needed by J2EE-uBet are very easy to install and configure for development. However, some of them must be installed with care. In this section it is commented how to do a basic installation and configuration of such packages. Check section 3 for environment variables when needed.

### 2.1 Basic installation and usage of MySQL Community Edition

---

\* Installation of MySQL 4.1.14 on Unix-like operating systems:

Install MySQL as root user and run the server as normal user.

As "root" user, proceed as follows:

- Unpack mysql-standard-4.1.14-pc-linux-gnu-i686.tar.gz on some directory (e.g. /usr/local).
- chown -R root:root mysql-standard-4.1.14-pc-linux-gnu-i686
- chmod -R 755 mysql-standard-4.1.14-pc-linux-gnu-i686
- Create the symbolic link "/usr/local/mysql" pointing to the directory where MySQL is installed:  
ln -s `pwd`/mysql-standard-4.1.14-pc-linux-gnu-i686 /usr/local/mysql

As normal user, proceed as follows:

- Create a directory where MySQL will store its databases. For example: /home/user/.MySQLData.
- Create \$HOME/.my.cnf file with a similar content to the following:

```
[mysqld]
datadir=/home/user/.MySQLData
```

Change the value of "datadir" to the directory created previously.

- cd /usr/local/mysql
- scripts/mysql\_install\_db

This will create "mysql" and "test" databases in the directory specified by the previous "datadir" option.

\* Installation of MySQL 4.1.14 on MS-Windows operating systems with support for Microsoft Windows Installer (MSI):

- Double-click on mysql-essential-4.1.14-win32.msi. Use default options.
- After the installation of MySQL, the MySQL Server Instance Configuration Wizard will run. Use default options but the following ones:

- + Best Support For Multilingualism (UTF-8).
- + Uncheck "Install As Windows Service" and check "Install Bin Directory In Windows PATH".

\* In order to minimize changes to J2EE-uBet default configuration, create a database with name "j2ee", and a user with name "j2ee" and password "j2ee". In the case of a Unix-like operating system, run the following commands as normal user.

- Start the MySQL server.

```
mysqld
```

- Create a database with name "j2ee".

```
mysqladmin -u root create j2ee
```

- Create a user with name "j2ee" and password "j2ee", and allow him to access from local host.

```
mysql -u root
```

```
GRANT ALL PRIVILEGES ON j2ee.* to j2ee@localhost IDENTIFIED BY 'j2ee',
j2ee@localhost.localdomain IDENTIFIED BY 'j2ee';
```

- Try to access to "j2ee" database as "j2ee" user with password "j2ee".

```
mysql -u j2ee --password=j2ee j2ee
```

- Shutdown the MySQL server.

```
mysqladmin -u root shutdown
```

## 2.2 Basic installation and configuration of PostgreSQL on Unix-like ----- operating systems -----

- Required packages to install PostgreSQL 8.0.3:

```
+ postgresql-libs-8.0.3-1PGDG.i686.rpm  
+ postgresql-8.0.3-1PGDG.i686.rpm  
+ postgresql-server-8.0.3-1PGDG.i686.rpm
```

Initialize the database:

- Define the environment variable PGDATA, which must specify a directory where PostgreSQL will save data (tables, indexes, etc.). This variable is defined in the example `~/.bash_profile` provided in section 3. Do not create the directory ("initdb" will create it).

- initdb

It creates the directory specified by PGDATA and initializes the database.

In order to minimize changes to J2EE-uBet default configuration, create a database with name "j2ee" and a user with name "j2ee" and password "j2ee".

- Start the PostgreSQL server.

```
postmaster
```

Later, you can shutdown the PostgreSQL server by pressing CTRL-C.

- Create a user with name "j2ee" and password "j2ee".

```
createuser -P j2ee
```

- Create a database with name "j2ee".

```
createdb -U j2ee -W j2ee
```

- Try to access to "j2ee" database as "j2ee" user with password "j2ee".

```
psql j2ee j2ee
```

- Shutdown the PostgreSQL server.

You can shutdown the PostgreSQL server by pressing CTRL-C.

## 2.3 Basic installation and configuration of Tomcat -----

After unpacking jakarta-tomcat-5.5.9.tar.gz:

- You must have write permissions on the following directories (and the files/subdirectories they contain): conf, logs, temp, webapps and work.

- In order to avoid a lot of problems with sessions when working with SFSBs, it is better to disable session persistence across Tomcat restarts. In `$CATALINA_HOME/conf/context.xml`, remove the comments surrounding the line reading:

```
<Manager pathname="" />
```

- Modify `$CATALINA_HOME/conf/tomcat-users.xml` in order to create a user in the role of "manager". In our installation, we have modified the definition of the "tomcat" user as follows:

```
<user name="tomcat" password="tomcat" roles="tomcat,manager"/>
```

This allows you to access Tomcat Manager web application as user "tomcat" and password "tomcat".

- Copy the JDBC driver to `$CATALINA_HOME/common/lib`.

- Define a global datasource named "jdbc/J2EE-uBetMySQLDS" (if using MySQL) or

```

"jdbc/J2EE-uBetPostgreSQLDS" (if using PostgreSQL).

* Add the following lines to $CATALINA_HOME/conf/server.xml, inside the
"<GlobalNamingResources>" tag.

<!-- MySQL -->
<Resource name="jdbc/J2EE-uBetMySQLDS"
    auth="Container"
    type="javax.sql.DataSource"
    driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost/j2ee"
    username="j2ee"
    password="j2ee"
    maxActive="4"
    maxIdle="2"
    maxWait="10000"
    removeAbandoned="true"
    removeAbandonedTimeout="60"
    logAbandoned="true"
    validationQuery="SELECT COUNT(*) FROM PingTable"/>

<!-- PostgreSQL -->
<Resource name="jdbc/J2EE-uBetPostgreSQLDS"
    auth="Container"
    type="javax.sql.DataSource"
    driverClassName="org.postgresql.Driver"
    url="jdbc:postgresql://localhost/j2ee"
    username="j2ee"
    password="j2ee"
    maxActive="4"
    maxIdle="2"
    maxWait="10000"
    removeAbandoned="true"
    removeAbandonedTimeout="60"
    logAbandoned="true"
    validationQuery="SELECT COUNT(*) FROM PingTable"/>

* Add the following lines to $CATALINA_HOME/conf/context.xml, inside
the <Context> tag.

<ResourceLink name="jdbc/J2EE-uBetMySQLDS" global="jdbc/J2EE-uBetMySQLDS"
    type="javax.sql.DataSource"/>

<ResourceLink name="jdbc/J2EE-uBetPostgreSQLDS" global="jdbc/J2EE-uBetPostgreSQLDS"
    type="javax.sql.DataSource"/>

2.4 Basic installation and configuration of JBoss
-----
After unpacking jboss-4.0.4RC1.zip:

- java -jar jboss-4.0.4RC1-installer.jar

    In the installation wizard, choose "ejb3" installation type (by default,
    "ejb3-clustered" is installed).

- You must have write permissions on the directory (and the
    files/subdirectories it contain) server/default.

- If using MySQL, configure JBoss for it to be able to connect to MySQL
    server.

    * cp docs/examples/jca/mysql-ds.xml server/default/deploy
    * Modify server/default/deploy/mysql-ds.xml as follows:

<datasources>
<local-tx-datasource>
    <jndi-name>MySqlDS</jndi-name>
    <connection-url>jdbc:mysql://localhost/j2ee</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>j2ee</user-name>
    <password>j2ee</password>
    <exception-sorter-class-
name>org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter</exception-sorter-class-name>
    <!-- should only be used on drivers after 3.22.1 with "ping" support -->
    <valid-connection-checker-class-
name>org.jboss.resource.adapter.jdbc.vendor.MySQLValidConnectionChecker</valid-connection-checker-
class-name>
    <!-- sql to call when connection is created -->
```

```

<new-connection-sql>SELECT COUNT(*) FROM PingTable</new-connection-sql>
<!-- sql to call on an existing pooled connection when it is obtained from pool --
MySQLValidConnectionChecker is preferred for newer drivers
<check-valid-connection-sql>some arbitrary sql</check-valid-connection-sql>
-->

<!-- corresponding type-mapping in the standardjbosscmp-jdbc.xml (optional) -->
<metadata>
  <type-mapping>mySQL</type-mapping>
</metadata>
</local-tx-datasource>
</datasources>

* Copy MySQL JDBC driver to server/default/lib.

- If using PostgreSQL, configure JBoss for it to be able to connect to
PostgreSQL server.

* cp docs/examples/jca/postgres-ds.xml server/default/deploy
* Modify server/default/deploy/postgres-ds.xml as follows:

<datasources>
<local-tx-datasource>
  <jndi-name>PostgresDS</jndi-name>
  <connection-url>jdbc:postgresql://localhost/j2ee</connection-url>
  <driver-class>org.postgresql.Driver</driver-class>
  <user-name>j2ee</user-name>
  <password>j2ee</password>
    <!-- sql to call when connection is created. Can be anything, select 1 is valid for
PostgreSQL -->
    <new-connection-sql>select 1</new-connection-sql>

    <!-- sql to call on an existing pooled connection when it is obtained from pool. Can be
anything, select 1 is valid for PostgreSQL -->
    <check-valid-connection-sql>select 1</check-valid-connection-sql>

    <!-- corresponding type-mapping in the standardjbosscmp-jdbc.xml (optional) -->
    <metadata>
      <type-mapping>PostgreSQL 7.2</type-mapping>
    </metadata>
</local-tx-datasource>
</datasources>

* Copy PostgreSQL JDBC driver to server/default/lib.

```

### 3. Environment variables

---

This is an excerpt of our " ~/.bash\_profile ". Adapt to your environment.

```

# -----
# Programming Tools.
# -----

# J2SE.
JAVA_HOME=/usr/java/jdk1.5.0_04
export JAVA_HOME
# For convenience.
PATH=$JAVA_HOME/bin:$PATH

# Ant.
ANT_HOME=/usr/apache-ant-1.6.5
export ANT_HOME
PATH=$PATH:$ANT_HOME/bin

# PostgreSQL.
PGDATA=$HOME/.PostgreSQLData
export PGDATA

# MySQL.
MYSQL_HOME=/usr/local/mysql
export MYSQL_HOME
PATH=$PATH:$MYSQL_HOME/bin

# Tomcat.
CATALINA_HOME=/usr/jakarta-tomcat-5.5.9
export CATALINA_HOME

```

```

# -----
# J2EE-uBet.
# -----


# The home directory of J2EE-uBet.
J2EE_UBET_HOME=$HOME/J2EE-uBet-src-1.0
export J2EE_UBET_HOME

```

In MS Windows operating system, you have to create the environment variables this way: 1. Right-click on My Computer desktop icon. 2. Select Properties from the contextual menu. 3. Select the Advanced Options tab. 4. Click on Environment variables button. 5. Click on New button to create the a new environment variable. You have to create all of these:

- JAVA\_HOME
- ANT\_HOME
- CATALINA\_HOME
- J2EE\_UBET\_HOME

Then you have to modify the environment variable PATH to append this:

```
;%JAVA_HOME%\bin;%ANT_HOME%\bin;%CATALINA_HOME%\bin
```

#### 4. Building and initialization of the database

---

Set up environment variables (section 3). Execute ". ~/.bash\_profile" if you are in an Unix-like operating system.

In an Unix-like operating system do this:

```
Unpack J2EE-uBet-src-1.0.tar.gz
cd $J2EE_UBET_HOME/Subsystems
```

In MS Windows operating system do this:

```
Unpack J2EE-uBet-src-1.0.zip
cd %J2EE_UBET_HOME%\Subsystems
```

You may need to adapt some of the below files. Both if you use MySQL and PostgreSQL, you need to adapt CommonProperties.xml and ..../Scripts/CommonEnvironmentVariables.{sh, bat}. If you use MySQL with the database "j2ee" and the user "j2ee" with password "j2ee", you don't need to modify more files. If you use PostgreSQL, you need to adapt data sources and notice that PostgreSQL is a DBMS providing identifier generators (sequences) while MySQL is a DBMS providing counter columns. To facilitate the use of both DBMS, we have included commented default sections for PostgreSQL in all files. So, you only need to uncomment them.

- CommonProperties.xml. Go to "Development environment" and "Development environment options" sections and follow instructions.
- ..../Scripts/CommonEnvironmentVariables.{sh, bat}. Go to "Run-time system" section and follow instructions.
- ..../PropertiesConfiguration/ConfigurationParameters.properties
   
\*\*\*REQUIRES MODIFICATION FOR PostgreSQL\*\*\*
- ..../PropertiesConfiguration/ServiceLocatorJNDIInitialContext.properties
- uBet/Sources/HTMLView/WEB-INF/plainweb.xml
   
\*\*\*REQUIRES MODIFICATION FOR PostgreSQL\*\*\*
- uBet/Sources/HTMLView/WEB-INF/ejbweb.xml
- uBet/Sources/EJBConfiguration/persistence.xml
   
\*\*\*REQUIRES MODIFICATION FOR PostgreSQL\*\*\*

Finally, start the database and execute:

```
ant all initdb
```

During execution of target "initdb" you will see a number of SQL errors reported by ant whenever a DROP sentence is executed. This is normal since J2EE-uBet SQL scripts drop tables and sequences before creating them, which causes errors when such tables or sequences don't exist yet.

## 5. Distribution

-----

```
PropertiesConfiguration
  |
  Scripts
  |
  Subsystems
  |
  |-- uBet
      |-- Sources
      |-- Scripts
  |-- WastingMoney
      |-- Sources
```

Under "PropertiesConfiguration" there are configuration files that maybe you need to adapt to your environment.  
"ConfigurationParameters.properties" is intended to be only used to run unit tests or standalone applications, and not for web applications (which use env-entry tags in web.xml for configuration information) or EJB components (which use env-entry tags in ejb-jar.xml for configuration information).

## 6. Ant files

-----

Each subsystem provides a "build.xml" file in its "Sources" directory, providing the following main targets (some targets may not be in WastingMoney build.xml file):

- \* compile (default target): compiles only source files newer than the corresponding ".class" files, which are generated under "<subsystem>/Build/Classes".
- \* rebuild: unconditionally recompile all source files. This target is specially important when the value of a constant is modified. If such a constant is used by other classes, you need to recompile all of them, even though the corresponding source files have been not modified (because Java compilers copy the value of the constant).
- \* jars : generates ".jar" files in "<subsystem>/Build/Jars".
- \* wars : generates ".war" files under "<subsystem>/Build/Wars".
- \* ears : generates ".ear" files under "<subsystem>/Build/Ears".
- \* javadoc: generates JavaDoc documentation under "<subsystem>/Build/JavaDoc".
- \* deployYYY : deploys ".war" or ".ear" files.

Inside each subsystem, targets are safe, which means that a target depends on other targets. For example, "javadoc" depends on "compile", which means that executing "ant javadoc" will recompile modified classes (if any) before generating JavaDoc documentation.

Each "build.xml" file includes "CommonProperties.xml" and "CommonPathReferences.xml" from "\$J2EE\_UBET\_HOME/Subsystems" directory.

Under "Subsystems" directory there is a "build.xml" file which links all "build.xml" files. Main targets:

- \* compile (default target): calls compile target on all "build.xml" files.
- \* rebuild: calls rebuild target on all "build.xml" files.
- \* jars: calls jars target on all "build.xml" files.
- \* wars: calls wars target on all "build.xml" files.
- \* ears: calls ears target on all "build.xml" files providing ears target.
- \* javadoc: generates \*all\* documentation in "Build/JavaDoc". It does \*not\* call "build.xml" files in the subsystems.
- \* sourcedist: generates a ".tar.gz" file and a ".zip" file with a clean source distribution (with no ".class" files, jars, JavaDoc, and so on). The file is generated in "\$J2EE\_EXAMPLES\_HOME/Build". This is the target we use when we build a source release ready to distribution. It can also be used to make a fast backup of the project.
- \* all: executes rebuild, jars, wars and javadoc targets on all "build.xml" files.
- \* initdb: runs SQL scripts (table creation) required by J2EE-uBet.

## 7. Executing uBet

-----

### 7.1 Unit tests

-----

Unit tests are provided as test case classes and are grouped in a test suite class (MainTest.java).

The classes providing unit tests are under the "test" directory under uBet subsystem.

In general, to execute a unit test do the following:

- Start database.
- Configure PropertiesConfiguration/ConfigurationParameters.properties.
- Initialize the environment variable "J2EE\_UBET\_CLASSPATH" as commented in section 2.
- Execute the class. Example:

In Unix-like operating systems type this:  
java -classpath \$J2EE\_UBET\_CLASSPATH ubet.test.userfacade.TestBet

In MS Windows operating system type this:  
java -classpath %J2EE\_UBET\_CLASSPATH% ubet.test.userfacade.TestBet

You can also execute the unit tests by typing "ant test" inside the uBet subsystem Sources directory or with the Eclipse ant plugin.

## 7.2 Web applications

---

uBet is distributed in two versions:

- PlainuBet: [JDBC Model + View + Controller] --> [DB]
- EJBuBet: [View + Controller] --> [EJB Model] --> [DB]

+ Deploying and running PlainuBet:

```
cd $J2EE_UBET_HOME/Subsystems/uBet/Sources  
ant deployplainwar  
$CATALINA_HOME/bin/startup.sh  
Point your browser to http://localhost:8080/PlainuBet
```

When finished, stop Tomcat: \$CATALINA\_HOME/bin/shutdown.sh.

+ Deploying and running EJBuBet:

```
cd $J2EE_UBET_HOME/Subsystems/uBet/Sources  
ant deployejbear  
$JBoss_HOME/bin/run.sh  
Point your browser to http://localhost:8080/EJBuBet
```

When finished, you can stop JBoss by pressing CTRL-C.

## 7.3 Changes to a war file with Tomcat

---

What if you need to make a change to a war file when Tomcat is started? One option is to stop it, deploy the war file again, and restart Tomcat. However this is time consuming, since Tomcat needs \*some\* time to start up. Below general guidelines are given to speed up this process.

- If you change a JSP or HTML file, you only need to copy it to \$CATALINA\_HOME/webapps/uBet. Example:  
cp Index.jsp \$CATALINA\_HOME/webapps/uBet
- If you change any other file (Java source file, web.xml, etc.), you need to generate the war file and deploy it again. In order to do this, you only need to type:

```
cd $J2EE_UBET_HOME/Subsystems/uBet/Sources  
ant deployplainwar
```

"deployplainwar" target depends on "plainwar" target, which depends on "compile" target. So, the above line compiles necessary source files, creates the war file and deploy it on Tomcat.

Now, you need to tell Tomcat that restarts this web application. In order to do this, access the Tomcat Manager web application by typing the following URL in your browser:

<http://localhost:8080/manager/html>

Authenticate as user "tomcat" and password "tomcat". Finally, reload the appropriate web application (PlainuBet in this case).

## 7.4 Debugging applications with Tomcat

-----  
Tomcat provides log files which help debugging web applications. These files reside under \$CATALINA\_HOME/logs:

- localhost.<DATE>.log: messages written by using javax.servlet.ServletContext.log are sent to this file. When Tomcat has a problem compiling a JSP page, can not load a class, etc., writes a stack trace of the exception in this file. Struts uses javax.servlet.ServletContext.log for logging actions.
- catalina.out: all messages written by using System.out or System.err.

You should remove log files from time to time (they waste disk space). In order to view these files in a convenient way, it is useful to start two terminals, and run the following two commands (assuming your current working directory be \$CATALINA\_HOME/logs):

```
tail -f localhost.<DATE>.log  
tail -f catalina.<DATE>.log
```

"tail" with the option "-f" allows to see the new lines appended to a file as it grows.

## 7.5 More on Tomcat

-----

Its documentation, of course :-)

Assuming Tomcat is started, type http://localhost:8080 in your browser, and be \*patient\* to find what you need.

## 7.6 Changes to an ear file with JBoss

-----

If you make a change to an ear file, you can deploy it (ant deployejbjar) on JBoss while it is running. JBoss will automatically detect that the ear file has changed.

## 8. Executing Wasting Money

-----

### 8.1 Web application

-----

uBet is distributed in one version:

- WastingMoney: [JDBC Model + View + Controller] --> [DB]

+ Deploying and running WastingMoney (with Tomcat):

```
cd $J2EE_UBET_HOME/Subsystems/WastingMoney/Sources  
ant deploywar  
$CATALINA_HOME/bin/startup.sh  
Point your browser to http://localhost:8080/WastingMoney
```

When finished, stop Tomcat: \$CATALINA\_HOME/bin/shutdown.sh.

+ Deploying and running WastingMoney (with JBoss):

```
cd $J2EE_UBET_HOME/Subsystems/WastingMoney/Sources  
ant deploywar  
$JBoss_HOME/bin/run.sh  
Point your browser to http://localhost:8080/WastingMoney
```

When finished, you can stop JBoss by pressing CTRL-C.

### 8.2 Changes to a war file with Tomcat

-----

What if you need to make a change to a war file when Tomcat is started? One option is to stop it, deploy the war file again, and restart Tomcat. However this is time consuming, since Tomcat needs \*some\* time to start up. Below general guidelines are given to speed up this process.

- If you change a JSP or HTML file, you only need to copy it to \$CATALINA\_HOME/webapps/WastingMoney. Example:  
cp Index.jsp \$CATALINA\_HOME/webapps/WastingMoney
- If you change any other file (Java source file, web.xml, etc.), you

need to generate the war file and deploy it again. In order to do this, you only need to type:

```
cd $J2EE_HOME/Subsystems/WastingMoney/Sources  
ant deploywar  
  
"deploywar" target depends on "war" target, which depends on  
"compile" target. So, the above line compiles necessary source files,  
creates the war file and deploy it on Tomcat.
```

Now, you need to tell Tomcat that restarts this web application. In order to do this, access the Tomcat Manager web application by typing the following URL in your browser:

```
http://localhost:8080/manager/html
```

Authenticate as user "tomcat" and password "tomcat". Finally, reload the appropriate web application (WastingMoney in this case).

#### 8.3 Debugging applications with Tomcat

---

Tomcat provides log files which help debugging web applications. These files reside under \$CATALINA\_HOME/logs:

- localhost.<DATE>.log: messages written by using javax.servlet.ServletContext.log are sent to this file. When Tomcat has a problem compiling a JSP page, can not load a class, etc., writes a stack trace of the exception in this file. Struts uses javax.servlet.ServletContext.log for logging actions.
- catalina.out: all messages written by using System.out or System.err.

You should remove log files from time to time (they waste disk space). In order to view these files in a convenient way, it is useful to start two terminals, and run the following two commands (assuming your current working directory be \$CATALINA\_HOME/logs):

```
tail -f localhost.<DATE>.log  
tail -f catalina.<DATE>.log
```

"tail" with the option "-f" allows to see the new lines appended to a file as it grows.

#### 8.4 More on Tomcat

---

Its documentation, of course :-)

Assuming Tomcat is started, type http://localhost:8080 in your browser, and be \*patient\* to find what you need.

### 9. Distributed environment

---

Below, we include some instructions on how to run a distributed 2-tier web application in the development environment we have used, where each component runs in a different machine.

Example:

```
Tomcat/JBoss    --> MySQL/PostgreSQL  
[193.144.50.156]      [193.144.50.158]
```

\* MySQL:

```
- mysql -u root mysql  
  
grant all privileges on j2ee.* to 'j2ee'@'193.144.50.156'  
identified by 'j2ee';
```

This allows the j2ee user to connect to the j2ee database from the machine with IP 193.144.50.156, where JBoss runs.

\* PostgreSQL:

In \$PGDATA/pg\_hba.conf, add the following line:

```
host      j2ee      j2ee      193.144.50.156      255.255.255.255      trust
```

The above line allows the j2ee user to connect to the j2ee database from the machine with IP 193.144.50.156, where JBoss runs.

If using PostgreSQL 8.0.3:

- In \$PGDATA/postgresql.conf, specify:

```
listen_addresses = '*'
```

This allows to accept remote connections.

- Alternatively, the above host entries in \$PGDATA/pg\_hba.conf can also be specified by using CIDR notation. Example:

```
host      j2ee      j2ee      193.144.50.156/32      trust
```

is equivalent to

```
host      j2ee      j2ee      193.144.50.156          255.255.255.255      trust
```

\* JBoss

In "<JBoss>/server/default/deploy/{mysql-ds,postgres-ds}.xml", change connection-url (jdbc:mysql://193.144.50.158/j2ee or jdbc:postgresql://193.144.50.158/j2ee).

\* Tomcat

In "\$CATALINA\_HOME/conf/server.xml", change the value of the url parameter (jdbc:mysql://193.144.50.158/j2ee or jdbc:postgresql://193.144.50.158/j2ee) in the definition of the J2EE-uBetMySQLDS and J2EE-uBetPostgreSQLDS data sources.

## 10. Known problems with the development environment

---

In this section, we comment the problems (bugs) that we have detected in the development environment we have used to develop J2EE-uBet.

### 10.1 Known problems with JBoss

---

If the database is shutdown when JBoss is running, EJBuBet needs to be redeployed (or alternatively, stopping and restarting JBoss). This is due to a bug in JBoss with respect to SFSBs.

### 10.2 The "focus" attribute in Struts' <html:form> tag

---

When the <html:form> tag in JSP documents is not the root tag, the cursor is not placed automatically in the field specified by the "focus" attribute. This may be caused by a bug in Struts or Tomcat.

## 11. Bug reports and suggestions for improvement

---

Send bug reports and suggestions for improvement to "djalma\_fd@yahoo.es" or "ja\_pereira@eresmas.com", including in the subject "J2EE-uBet 1.0".

## 5 PROBLEMAS CONOCIDOS

El problema detectado hasta la fecha es que la versión EJB de **uBet** no deja cambiar el login de usuario, aunque actualiza el login en la sesión. El motivo del problema es que no se puede hacer commit de la transacción, porque en medio de ella se cambia el identificador del entity.

## 6 REFERENCIAS

- [1] betandwin – <http://www.betandwin.com>
- [2] Sun Microsystems, J2EE – <http://java.sun.com/j2ee>
- [3] Sun Microsystems, Java Code Conventions – <http://java.sun.com/docs/codeconv>

- [4] Java DataBase Connectivity, JDBC – <http://java.sun.com/products/jdbc>
- [5] Enterprise JavaBean, EJB – <http://java.sun.com/products/ejb>
- [6] Junit – <http://www.junit.org>
- [7] XSL Transformations, XSLT – <http://www.w3.org/TR/xslt>
- [8] JavaServer Pages Standard Tag Library, JSTL – <http://java.sun.com/products/jsp/jstl>
- [9] w3c, XML – <http://www.w3.org/XML>
- [10] Fernando Bellas Permuy, J2EE-Examples –  
<http://www.tic.udc.es/fbellas/teaching/is/index.html>
- [11] JBoss website – <http://www.jboss.org>
- [12] Tomcat website – <http://jakarta.apache.org/tomcat>
- [13] Apache Software Foundation, Struts framework – <http://jakarta.apache.org/struts>
- [14] Apache Software Foundation, Ant – <http://ant.apache.org>
- [15] MySQL – <http://www.mysql.org>
- [16] Google – <http://www.google.com>