

# Mi eventbrite.com

1. Descripción del negocio

2. Problemas y soluciones

- 2.1. ¿Como un usuario gestiona su evento?
- 2.2. ¿Como un usuario adquiere su entrada?
- 2.3. ¿Como eventbrite les notifica a los usuarios acerca de un evento?

3. Patrones de diseño implementados

4. Arquitectura propuesta

5. Seguridad

6. Prototipo del front-end

- 6.1. Inicio
- 6.2. Menú de usuario
- 6.3. Suscribirse a zonas de interés
- 6.4. Crear evento
- 6.5. Mis eventos
- 6.6. Administrar entradas
- 6.7. Adquirir entrada (principal)
  - 6.7.1. Adquirir entrada de pago
  - 6.7.2. Adquirir entrada gratuita
  - 6.7.3. Adquirir entrada de donación

7. Diagramas

7.1. Diagrama de casos de uso

7.2. Diagramas de clases

- 7.2.1. Diagrama de clases de dominio
- 7.2.2. Diagrama de clases de control

7.3. Diagrama entidad relación

7.4 Diagramas de secuencia

- 7.4.1. Suscribirse a zonas de interés
- 7.4.2. Crear evento
- 7.4.3. Administrar entradas
- 7.4.4. Confirmar evento
- 7.4.5. Adquirir entrada
  - 7.4.5.1 Bonus track: Logica de adquisicion de entrada en codigo (Java)

# Descripción del Negocio<sup>1</sup>

Eventbrite permite crear eventos para luego postearlos en su aplicación, así como también permite que los usuarios puedan comprar entradas para dichos eventos.

## Sobre la creación de eventos:

Previamente a que un usuario quiera organizar un evento este debe configurar su perfil especificando su cuenta bancaria para poder hacer los depósitos por cada entrada vendida.

Una vez configurado esto, el organizador puede crear eventos indicando nombre del evento, tópico (opcional), ubicación, fecha y hora de inicio y fin, imagen de portada del evento y una descripción.

Por otra parte, el organizador puede armar grupos de entradas para un evento especificando la cantidad y si son gratuitas, de pago o a donación (el cliente dona el monto que desee). Los grupos de entradas tienen nombre, fecha de inicio de venta, fecha fin de venta, y precio si corresponde. Cada entrada tiene un código QR, el organizador del evento debe subir un archivo pdf con una entrada por página para luego poder guardar cada código QR en distintos archivos.

Una vez que el organizador haya completado todos estos datos está en disposición de confirmar el evento para que Eventbrite pueda publicarlo.

Eventbrite cobra un porcentaje del precio de la entrada y dicho porcentaje actualmente es de 6.99% más IVA.

## Sobre la adquisición de las entradas:

Un usuario puede intentar adquirir una entrada, si la entrada es gratuita se le asigna automáticamente mientras que, si es de paga o a donación el usuario debe ingresar los datos de su tarjeta de crédito, luego el sistema valida estos datos y si todo es correcto se le asigna una entrada a dicho usuario. En caso de que el pago no se pueda validar se le notificará al usuario que no se pudo realizar la transacción correctamente.

Cuando un usuario adquiere una entrada satisfactoriamente esta entrada queda guardada en su perfil ya disponible para que la pueda imprimir o llevarla electrónicamente.

## Funcionalidades para cualquier usuario:

Un usuario puede especificar en su perfil qué tópicos son de su interés y a su vez que zonas son las que le importa que le avisen cuando ocurra un evento. Una vez que la persona configura esto, cada vez que un organizador publique un evento se le enviará una notificación a todos los usuarios interesados en la zona donde ocurre.

---

<sup>1</sup> Para realizar la descripción del negocio se tomaron algunos datos de la página real de Eventbrite y otros datos son hipótesis planteadas por mí para modelar el dominio del negocio.

# Problemas y soluciones

## 1er problema: ¿Como un usuario gestiona sus eventos?

Los encargados de resolver este problema son los siguientes casos de uso:

- Crear un evento: Con este CU especificamos los datos principales del evento tales como nombre del evento, fecha inicio, fecha fin, tópico, descripción, localización e imagen de portada
- Modificar datos del evento: Este CU es para actualizar alguno de los datos anteriores en caso de que el organizador los haya pasado por alto o si se haya confundido en alguno.
- Administrar entradas: El organizador utiliza esta funcionalidad para poder gestionar los grupos de entrada que va a tener su evento.

Por cada grupo de entradas que se cree, el organizador debe proporcionar un archivo pdf con una entrada por página. El sistema tomará cada código QR que se encuentra en el archivo pdf y lo separará en archivos distintos enlazando cada uno con una entrada.

- Confirmar evento/Cancelar evento: Una vez que el usuario haya configurado por completo su evento deberá confirmarlo o bien cancelarlo, una vez que el evento sea confirmado este se publicará en la página de Eventbrite.

## 2do problema: ¿Como un usuario adquiere su entrada?

El comprador debe elegir el grupo de entrada que desea (CU Adquirir entrada), luego según el tipo de entrada que sea se piden distintos datos:

- Gratis: No se pide ningún dato se le asigna una entrada automáticamente al usuario (no se puede pedir más de una entrada gratis por usuario).
- Donación: El comprador debe proporcionar los datos de su tarjeta, el monto que va a donar por entrada y la cantidad de entradas que va a querer. Una vez que presiona “Comprar” el sistema pasa a verificar si realmente hay suficientes entradas disponibles según la cantidad ingresada (en caso de que no haya suficientes el sistema le notificara al usuario), se reserva la entrada antes de verificar el pago y si la transacción se realiza correctamente se le asigna las entradas al usuario, en caso contrario se lanza una excepción avisando de que el pago no se pudo realizar por X motivo y se liberan las entradas previamente reservadas.
- Pago: Se maneja de la misma manera que por donación con la diferencia de que el monto de la entrada no lo elige el comprador, sino que ya está definido por el organizador.

## 3er problema: Notificar a los usuarios cuando ocurra un evento en una zona de su interés

Un usuario para disfrutar de esta funcionalidad debe ir a la configuración de su perfil (CU Configurar perfil) y establecer qué tópicos le gusta. Una vez hecho esto el usuario debe hacer uso del CU “Suscribirse a zonas de interés” para configurar qué zonas son las que le interesa que Eventbrite le notifique cuando ocurra un evento que tenga que ver con los tópicos que previamente configuró. Luego cada vez que ocurra un evento (CU confirmar evento) el sistema le notificará al usuario de dicho suceso.

# Patrones de diseño implementados

1. Para resolver la lógica de las notificaciones a los usuarios utilice el patrón Observer, este patrón me permite agregar distintos tipos de observadores por ejemplo un MailObserver para notificar por email sobre el suceso de un evento.
2. Como el sistema debe interactuar con distintos servicios tales como los de cada tarjeta (VISA, MasterCard, etc.) aplique el patrón Adapter para poder adaptar cada transacción dependiendo del sistema con el que interactúo.
3. Patrón Repository para el acceso a la base de datos y el mapeo objeto-relacional.

## Arquitectura propuesta

Para desarrollar la página eventbrite.com propongo la arquitectura MVC debido a que es una de las más sencillas y la mayoría de los frameworks están basados en esta arquitectura.

Para intercambiar datos entre el back-end y front-end utilizaría objetos JSON, para el front Angular 7(HTML5, CSS, Typescript), para el back Spring (Java), para persistir los datos una base de datos relacional en mi caso MySQL e Hibernate para realizar el mapeo objeto-relacional.



### ¿Por qué estas tecnologías?

**Angular:** Angular permite armar SPA (single page applications) es decir páginas que no necesitan recargarse por completo, sino que solo se cargan los componentes necesarios haciendo que la página web sea más dinámica y rápida.

Este framework tiene un gran soporte en internet y provee ciertos módulos que facilitan el trabajo a la hora de: validar datos, hacer consultas http, realizar el routing de la página, etc.

**Spring:** Este poderoso framework tiene varias ventajas a la hora de realizar el back-end ya que tiene todas las cualidades de Java (uso de paquetes, interfaces, manejo de excepciones, listas, lambdas, etc.), así como también características propias (por ej.: inyección de dependencias, programación orientada a aspectos).

**Hibernate:** El framework ORM Hibernate se complementa perfectamente con Spring y además tiene características como: EAGER y LAZY INITIALIZATION, mapeo de herencia por SINGLE-TABLE, JOINED y TABLE-PER-CLASS, mapeo de columnas y tablas a través de anotaciones o un archivo de configuración XML.

**MySQL:** Esta es más que nada una elección personal ya que cualquier base de datos relacional ofrece características similares, opte por MySQL porque es libre y ofrece el estándar SQL.

Por otra parte, la elección de una base de datos relacional es debido a que estas ofrecen consistencia de los datos, transacciones atómicas, back-ups y restauración.

## Seguridad<sup>2</sup>

Para la parte de seguridad se debe implementar técnicas básicas contra la inyección SQL y XSS (cross site scripting) y CORS (cross-origin resource sharing).

Luego como el negocio maneja datos sensibles tales como cuentas bancarias, tarjetas de crédito, usuarios y contraseñas la página web debe tener un certificado SSL y aplicar el protocolo HTTPS que es el estándar para todas las páginas en las que se manejan datos personales.

Otro aspecto a tener siempre en cuenta es un firewall para evitar las siguientes cuestiones:

- **Mitigación de ataque distribuido de denegación de servicio (DDoS).** Son ataques que aumentan dramáticamente el tráfico de tu sitio web dejándolo inactivo e incluso deshabilitado.
- **Protección y prevención de fuerza bruta.** Por ejemplo, cuando se prueban todas las combinaciones de usuarios y contraseñas para tratar de encontrar la correcta y tener acceso a tu página web.

Por último y por eso no menos importante es realizar un back-up de los datos de forma periódica para tener siempre un respaldo actualizado ante cualquier inconveniente de hardware o software.

---

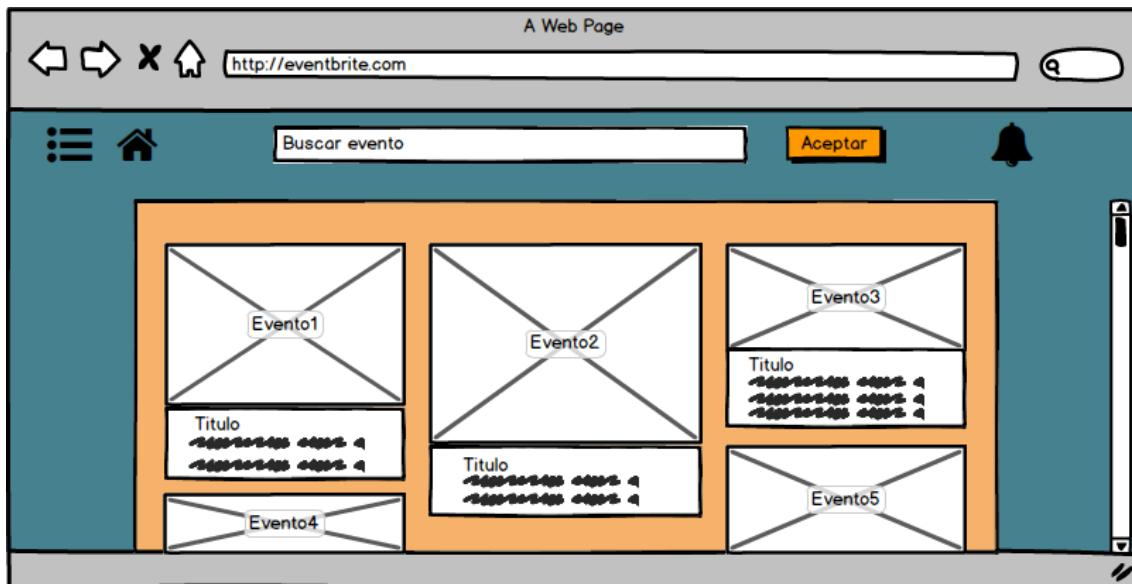
<sup>2</sup> Información recopilada de internet sobre cuestiones de seguridad básica en páginas web.

# Prototipo del diseño front-end

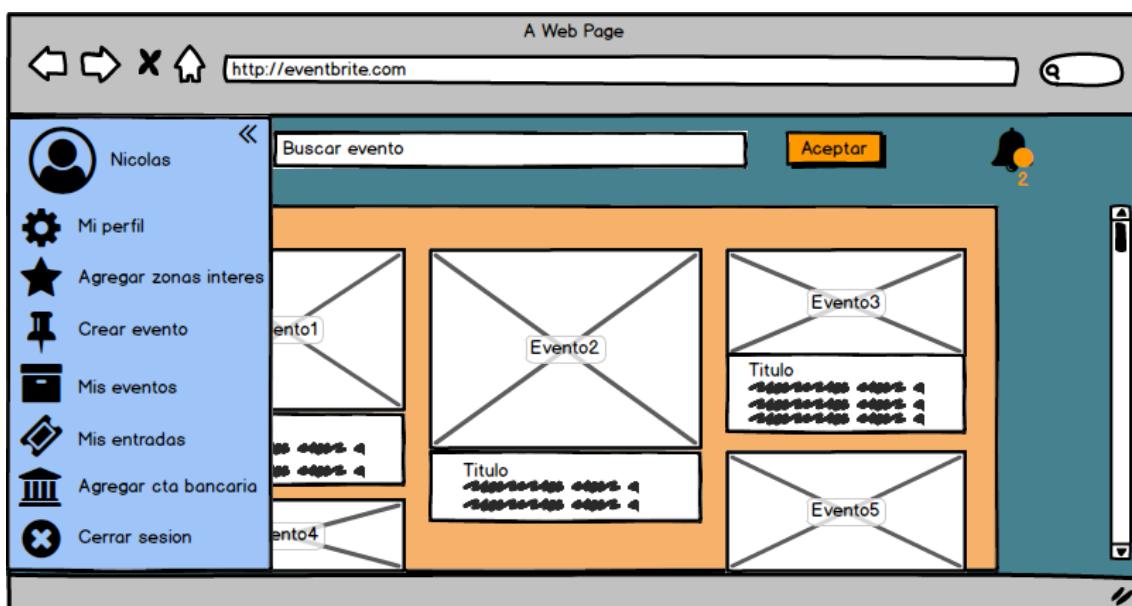
A continuación, se muestran algunos ejemplos de cómo se vería la página eventbrite.com para las funcionalidades más importantes del sistema.

Estas interfaces son a modo de ejemplo y fueron hechas con el programa Balsamiq.

## Inicio



## Menú usuario



## Suscribirse a zonas de interés

A screenshot of a web browser window titled "A Web Page" showing the URL <http://eventbrite.com>. The page has a teal header with icons for back, forward, search, and accept. A search bar says "Buscar evento". Below it is a section titled "Mis zonas de interés" (My Interest Areas). On the left, there's a dropdown menu for "País" (Country) with options Argentina, Brazil, Chile, and EEUU. Underneath is another dropdown for "Buenos Aires" with options like "Buenos Aires". To the right is a list of neighborhoods with checkboxes: 1 Tigre, 2 Malvinas Argentinas (checked), 3 Martinez, 4 Pilar, 5 Palermo (checked), 6 Ramos Mejía, 7 Lugano (checked), 8 Nuñez, and 9 Caballito. A green "Suscribirme!" button is at the bottom right.

## Crear evento

A screenshot of a web browser window titled "A Web Page" showing the URL <http://eventbrite.com>. The page has a teal header with icons for back, forward, search, and accept. A search bar says "Buscar evento". Below it is a section titled "Ingrrese los datos de su evento" (Enter event details). The form includes fields for "Nombre evento" (Event name), "Fecha de inicio" (Start date) set to 01/01/2019, "Fecha de finalización" (End date) set to 01/01/2019, "Hora de inicio" (Start time) set to 00, "Hora de finalización" (End time) set to 00, "Imagen de portada" (Thumbnail image), "Selección tópico" (Topic selection) with a dropdown set to "Tópico", "Ubicación" (Location) with dropdowns for "Argentina", "Bs As", and "Zona", "Domicilio" (Address), and a large "Descripción" (Description) text area. A green "Agregar evento!" button is at the bottom right.

## Mis eventos

A screenshot of a web browser window showing the 'Mis eventos' (My events) section of the Eventbrite website. The URL http://eventbrite.com is visible in the address bar. The page displays a list of four events with columns for ID, Title, Description, State, Edit, Administer, Confirm, and Cancel.

ID	Título	Descripción	Estado	Editar	Administrar	Confirmar entradas	Cancelar
4	Concierto	[REDACTED]	A confirmar				
3	Expo arte	[REDACTED]	En marcha				
2	Degustacion	[REDACTED]	Finalizado				
1	Taller ecológico	[REDACTED]	Cancelado				

Pág. 1 2 3

## Administrar entradas

A screenshot of a web browser window showing the 'Administrar entradas' (Manage tickets) section of the Eventbrite website. The URL http://eventbrite.com is visible in the address bar. The page allows users to manage ticket groups for an event with fields for group name, acquisition type, price, and date ranges.

Id evento: 1

Nombre grupo entrada

Fecha de inicio venta: 01/01/2019

Fecha de finalización de venta: 01/01/2019

PDF con entradas  
(Una entrada por hoja)

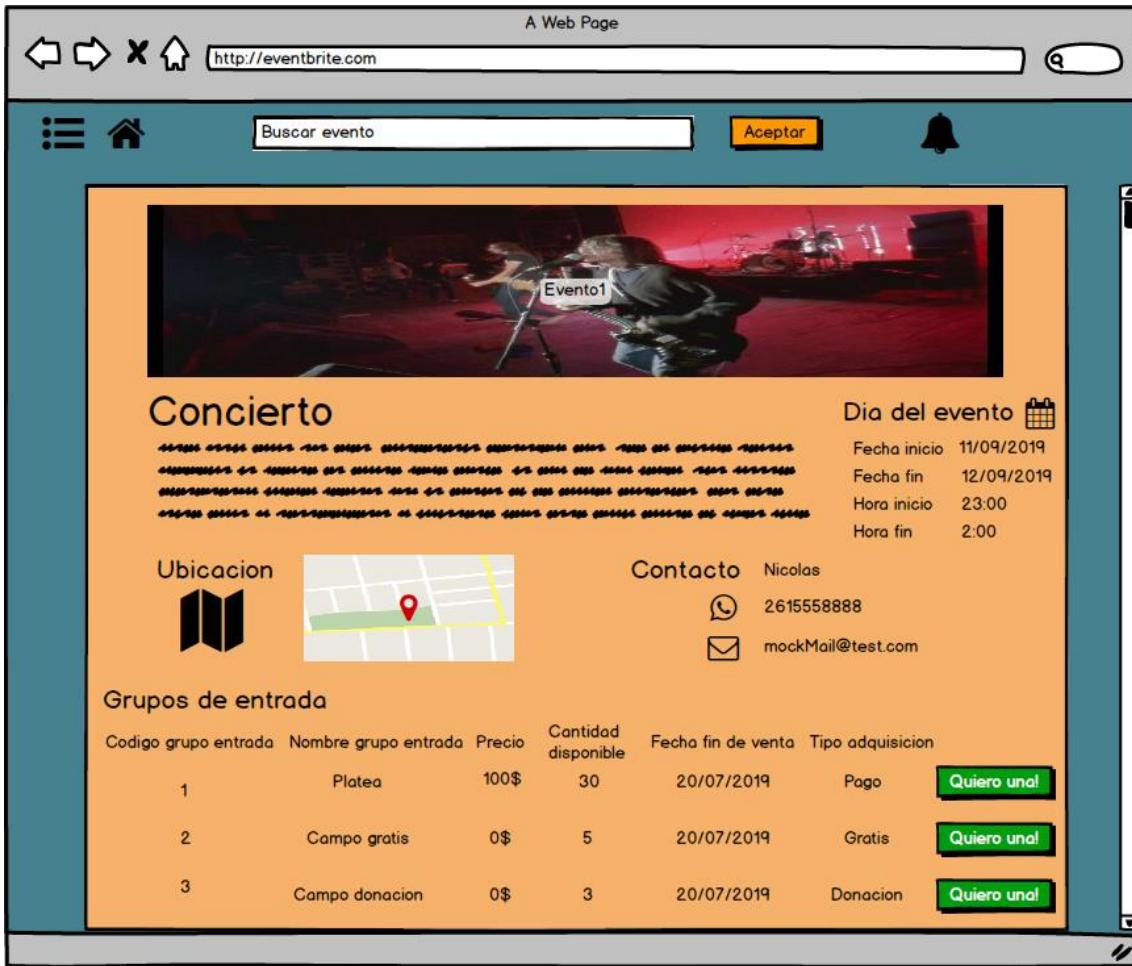
TipoAdquisicion: Gratis  
Precio: 0

Agregar otro grupo de entrada >

Agregar grupos de entradas!

## Adquirir entrada (menú principal)

A Web Page  
<http://eventbrite.com>



The screenshot shows a concert event titled "Concierto". The event details are as follows:

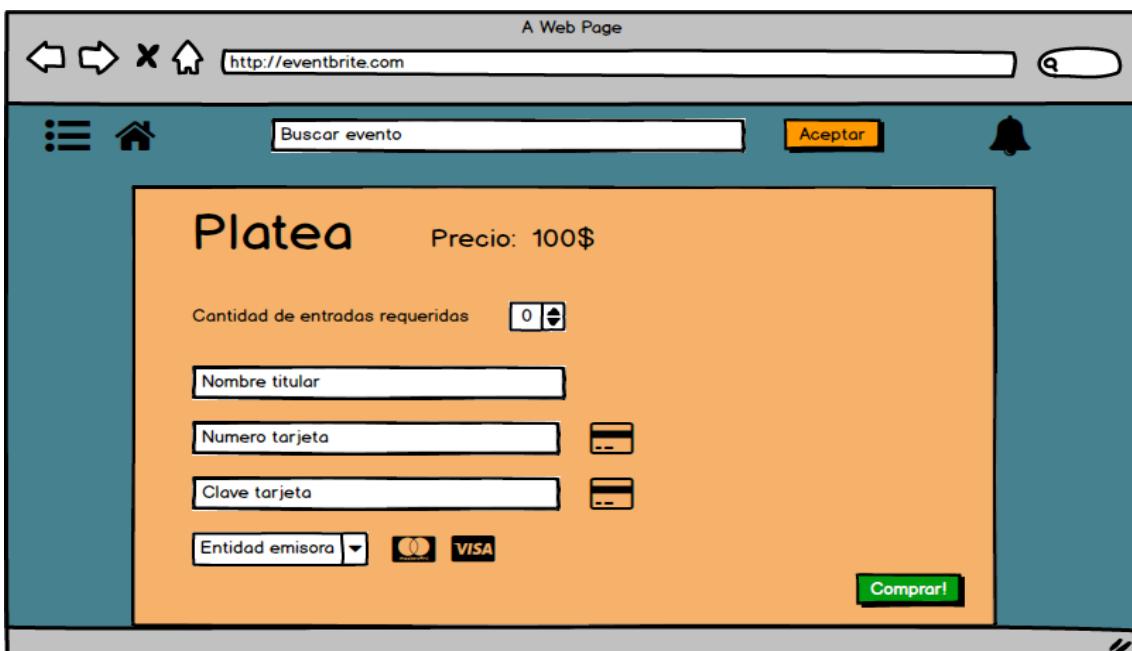
Ubicacion		Contacto		Dia del evento	
1	Platea	Nicolas	2615558888	Fecha inicio	11/09/2019
2	Campo gratis			Fecha fin	12/09/2019
3	Campo donacion			Hora inicio	23:00
				Hora fin	2:00

**Grupos de entrada**

Codigo grupo entrada	Nombre grupo entrada	Precio	Cantidad disponible	Fecha fin de venta	Tipo adquisicion	Acción
1	Platea	100\$	30	20/07/2019	Pago	Quiero una!
2	Campo gratis	0\$	5	20/07/2019	Gratis	Quiero una!
3	Campo donacion	0\$	3	20/07/2019	Donacion	Quiero una!

## Adquirir entrada de pago

A Web Page  
<http://eventbrite.com>



The screenshot shows a payment form for a "Platea" ticket at a price of 100\$. The form includes fields for:

- Cantidad de entradas requeridas: 0
- Nombre titular: [Input field]
- Numero tarjeta: [Input field] [Card icon]
- Clave tarjeta: [Input field] [Card icon]
- Entidad emisora: [Dropdown menu] [Logos for American Express and VISA]

A green "Comprar!" button is located at the bottom right.

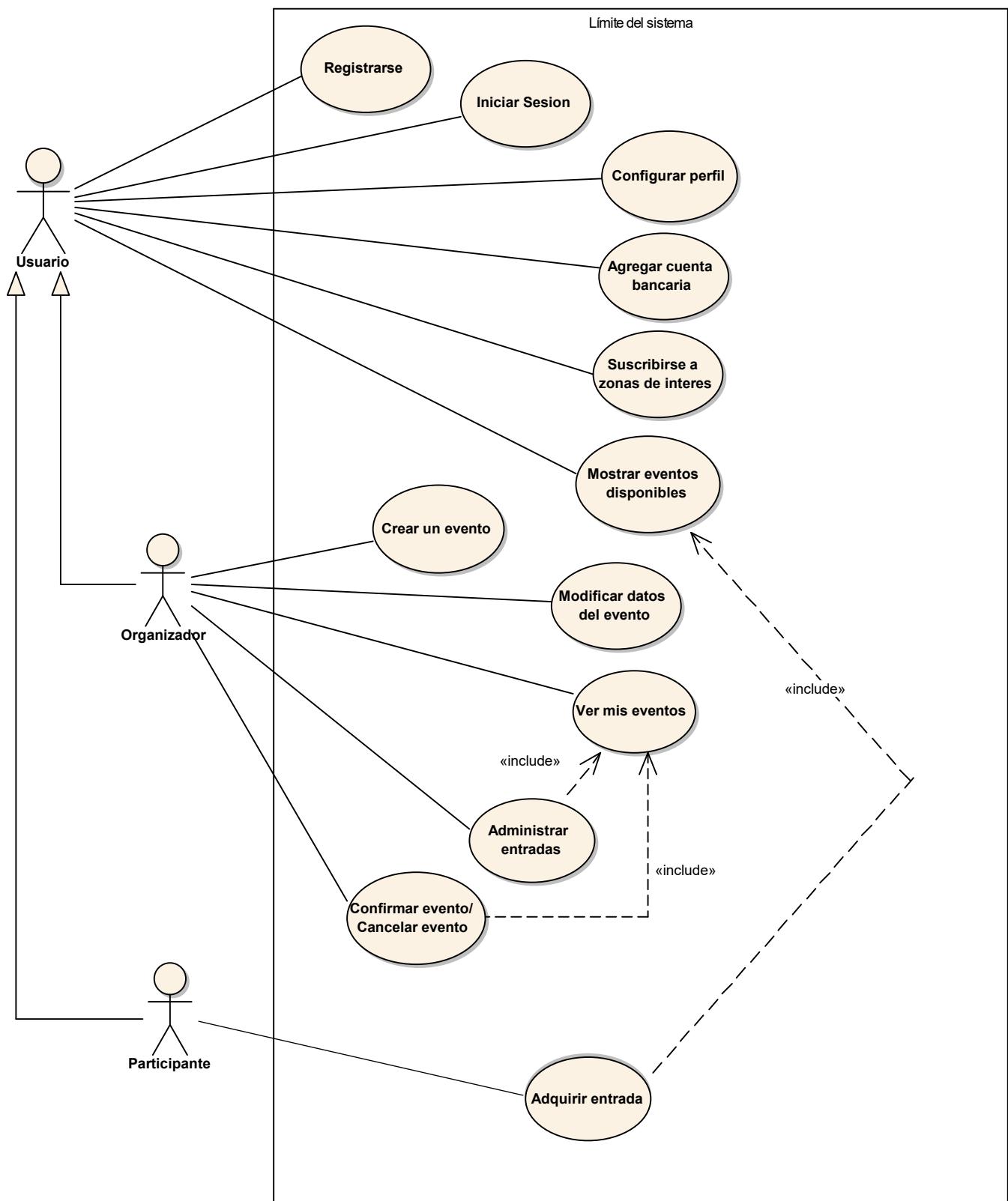
## Adquirir entrada gratuita



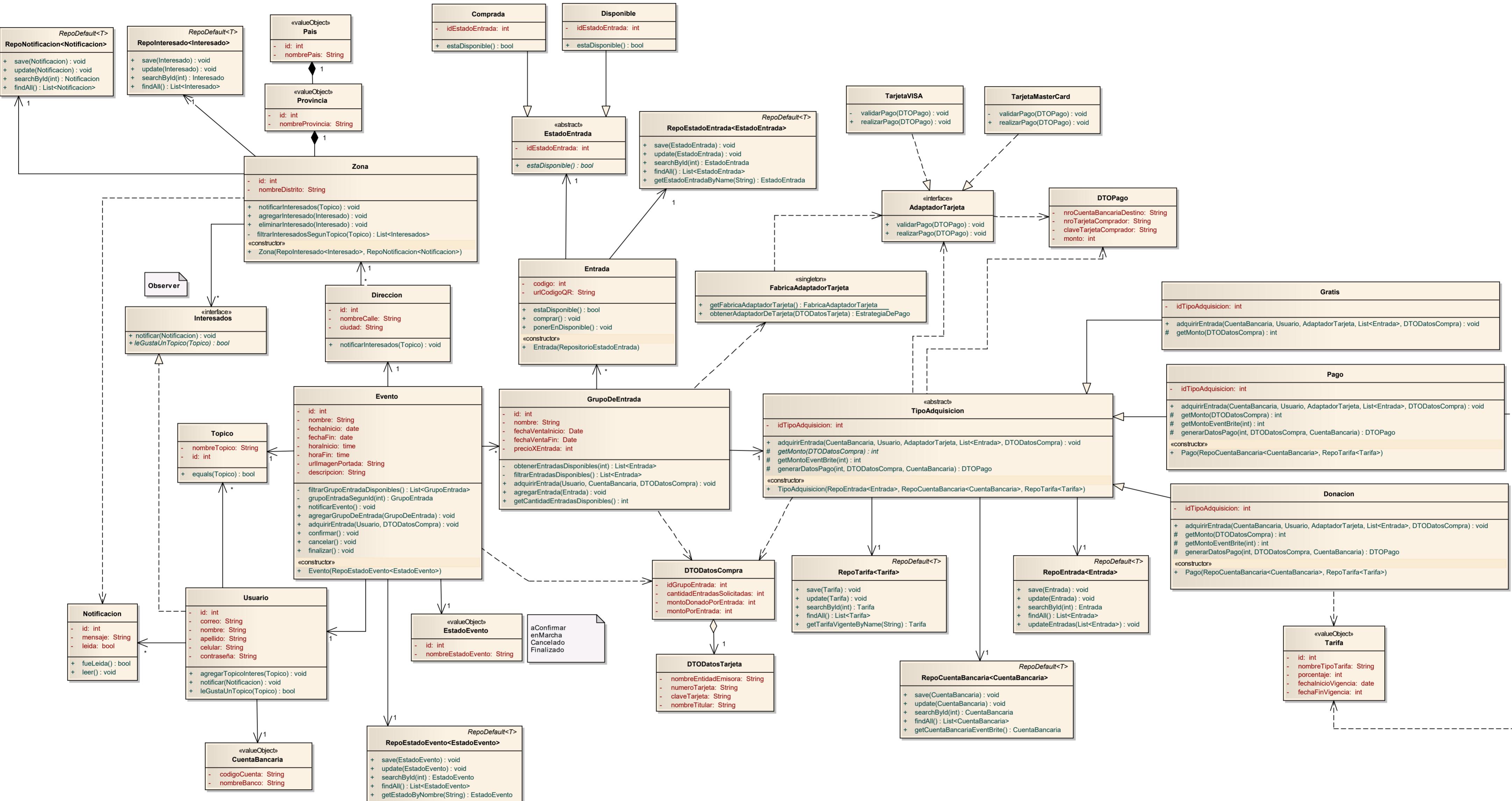
## Adquirir entrada de donación

A screenshot of a web browser window titled "A Web Page" showing the URL "http://eventbrite.com". The page has a teal header with a search bar labeled "Buscar evento" and a yellow "Aceptar" button. Below the header, there's a large orange rectangular box containing the text "Campo donacion" and "Precio: -". Inside this box, there are two input fields: "Cantidad de entradas requeridas" with a value of "0" and "Monto a donar" with a value of "0". Below these fields are three text input fields: "Nombre titular", "Número tarjeta", and "Clave tarjeta", each with a small credit card icon to its right. There is also a dropdown menu labeled "Entidad emisora" with a downward arrow, and icons for American Express and VISA. In the bottom right corner of the orange box is a green "Comprar!" button.

# Diagrama de casos de uso



## Diagrama de clases de dominio



## Diagrama de clases de control

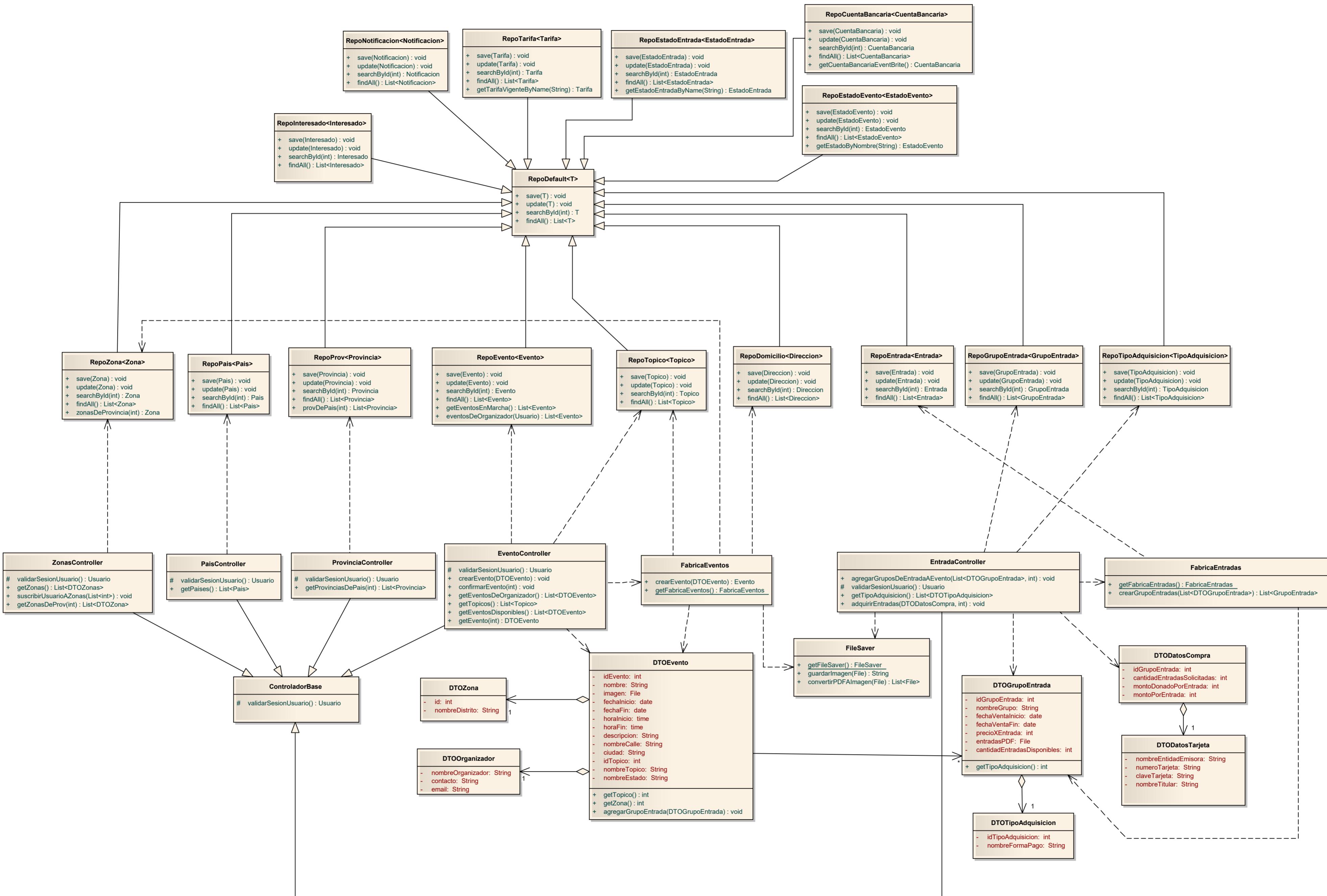
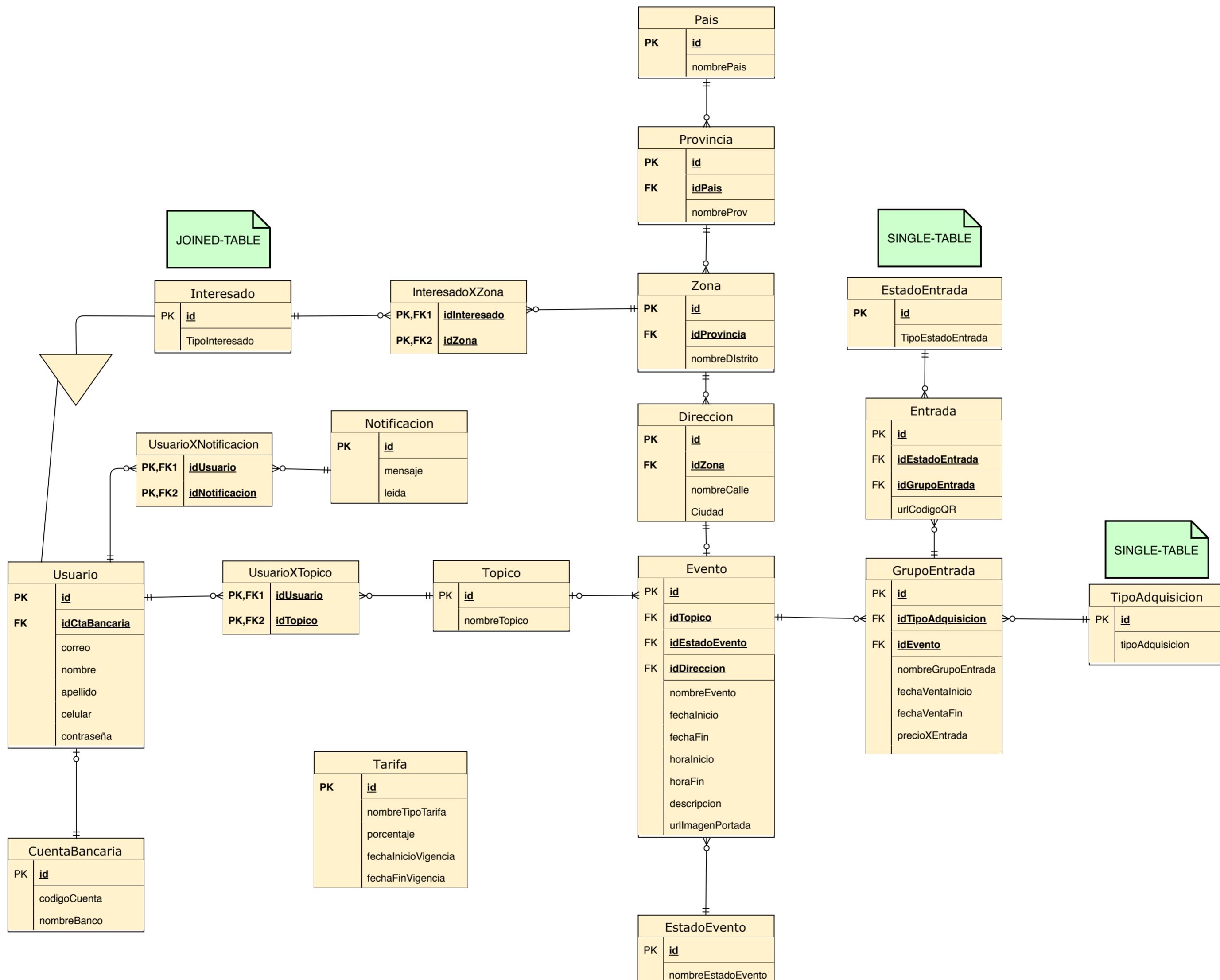


Diagrama entidad relación



## Diagrama de secuencia "Suscribirse a zonas de interes"

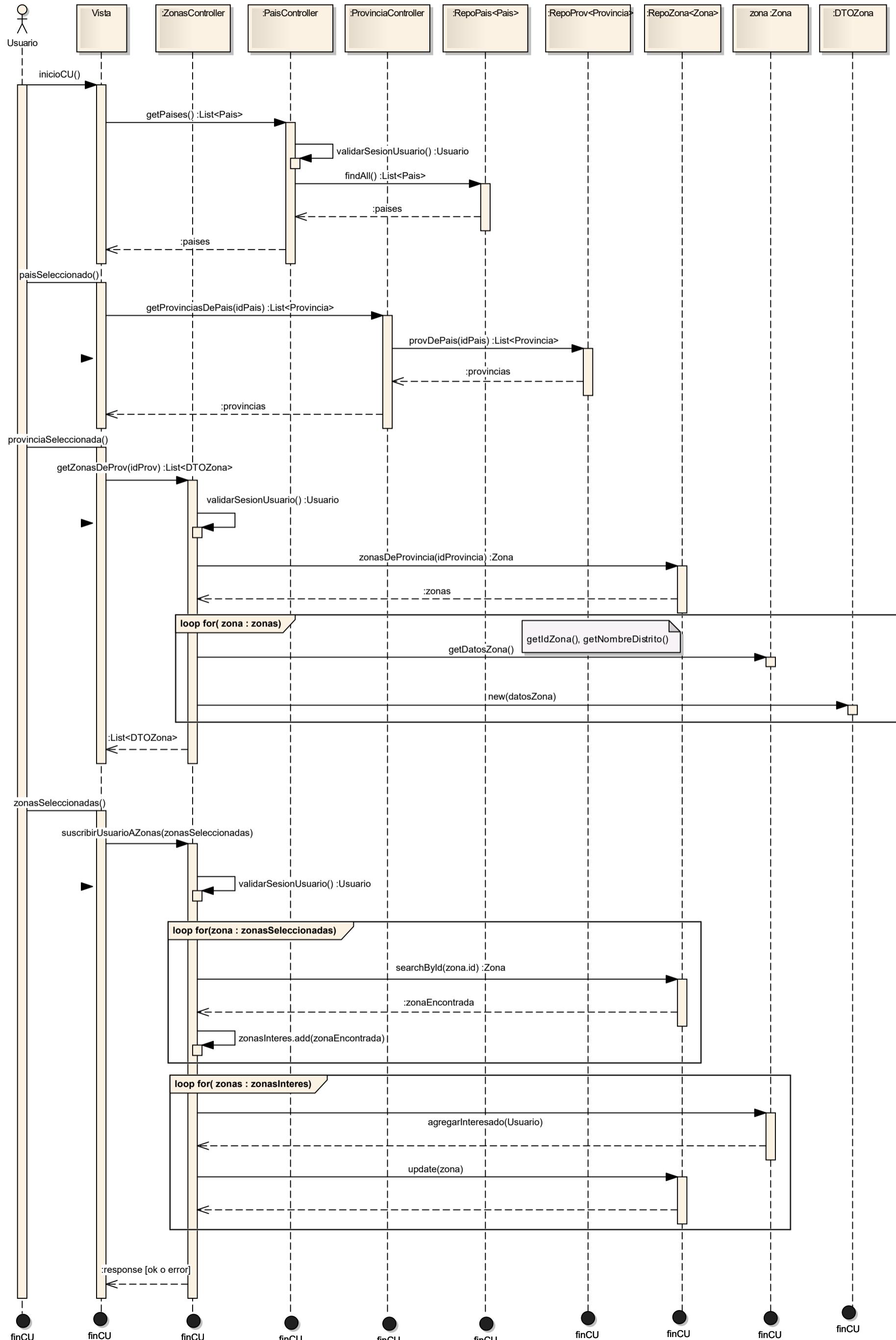
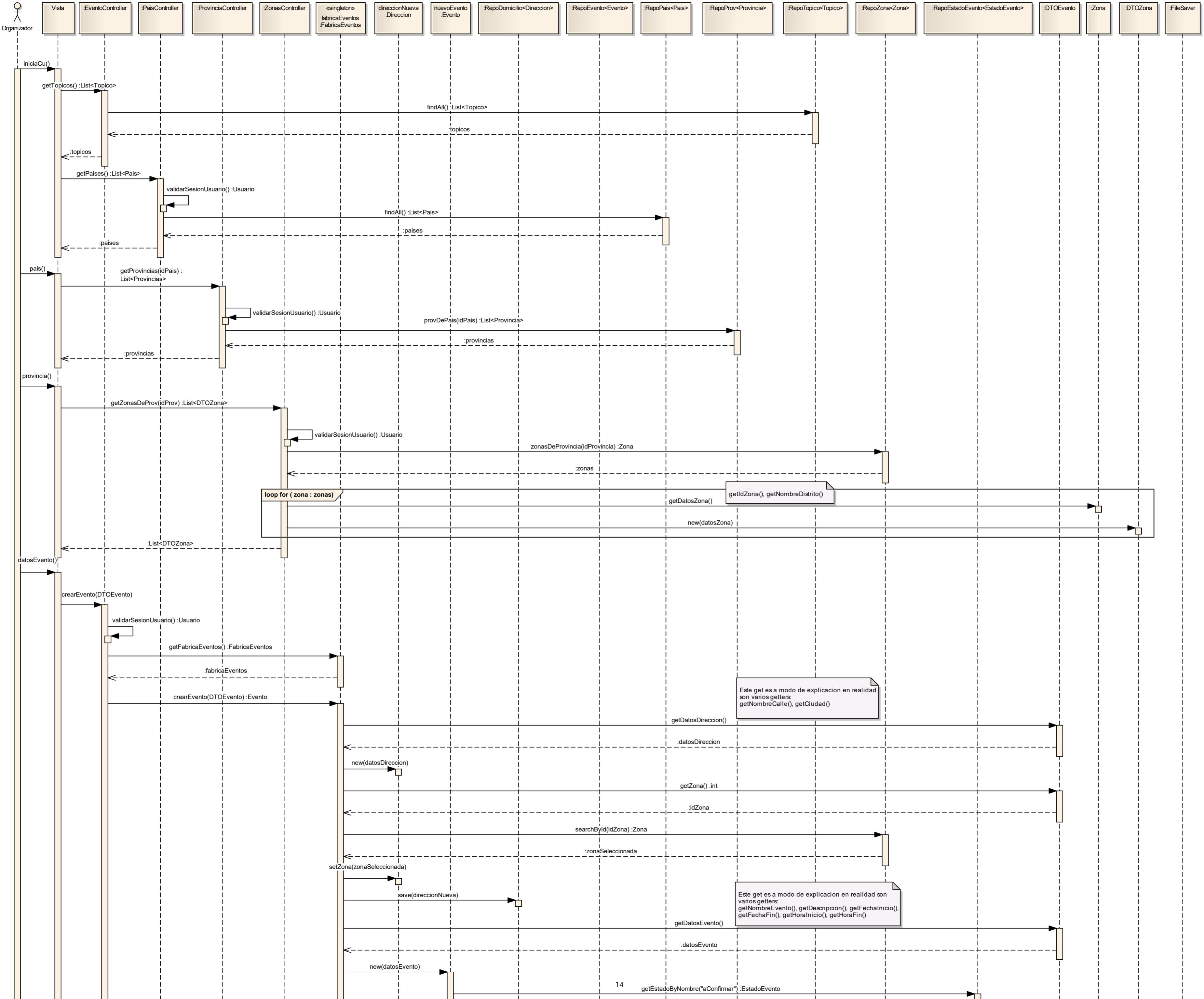


Diagrama de secuencia  
"Crear evento"



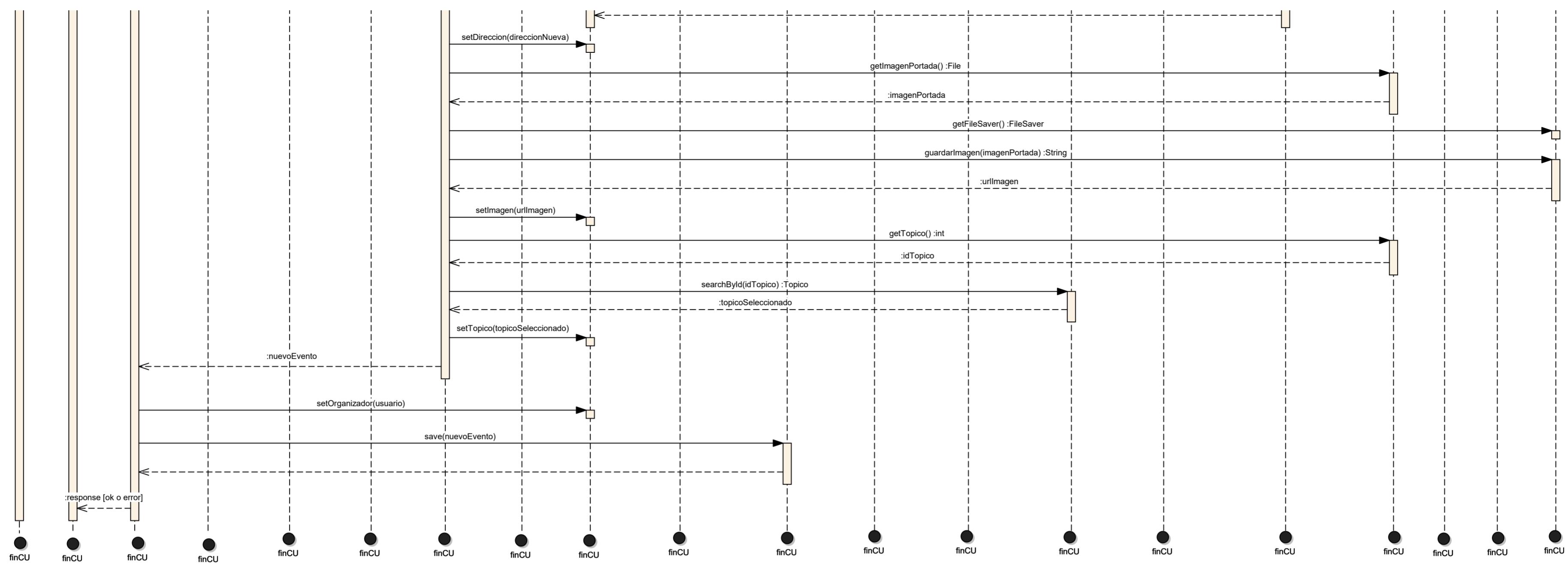
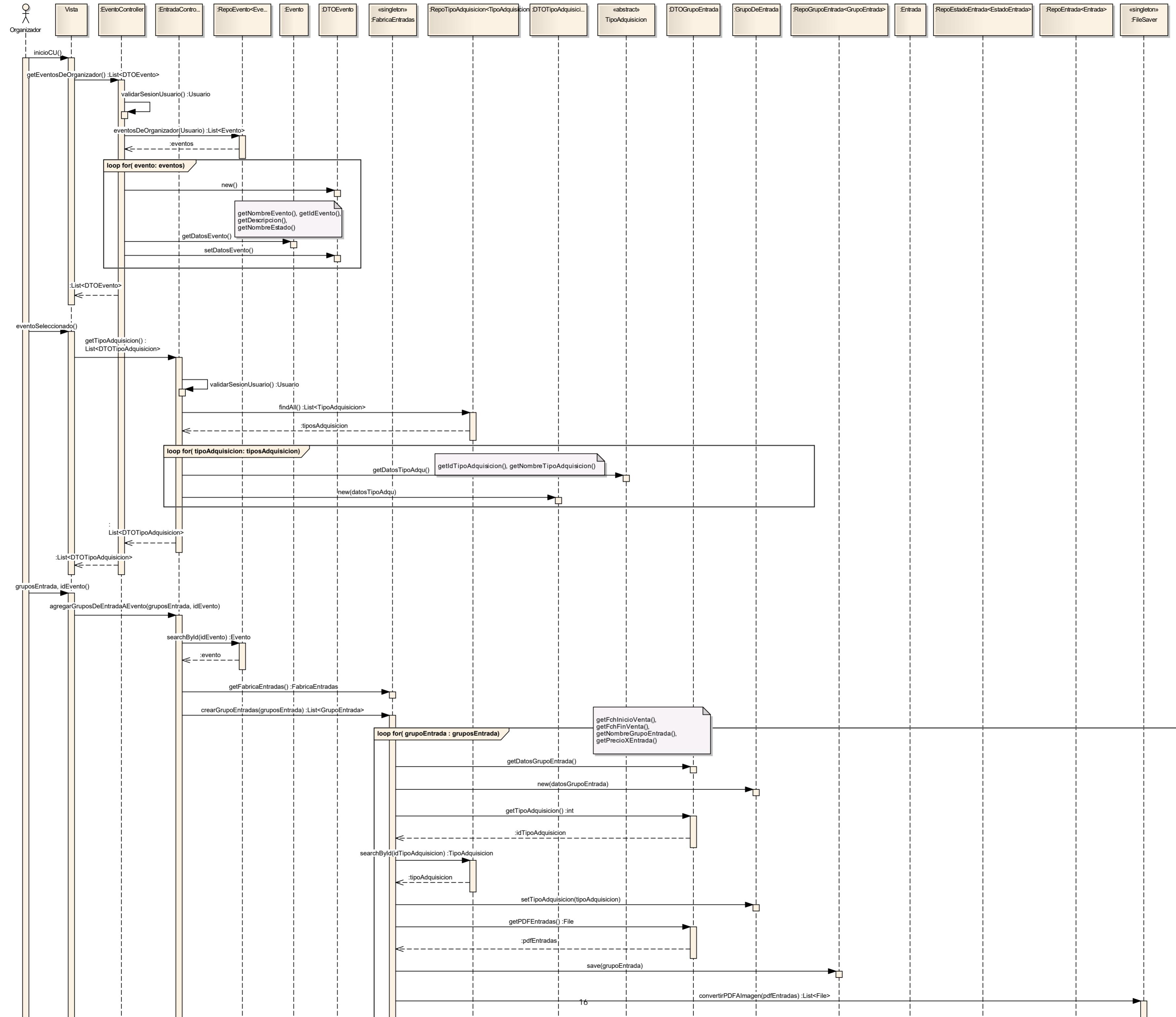
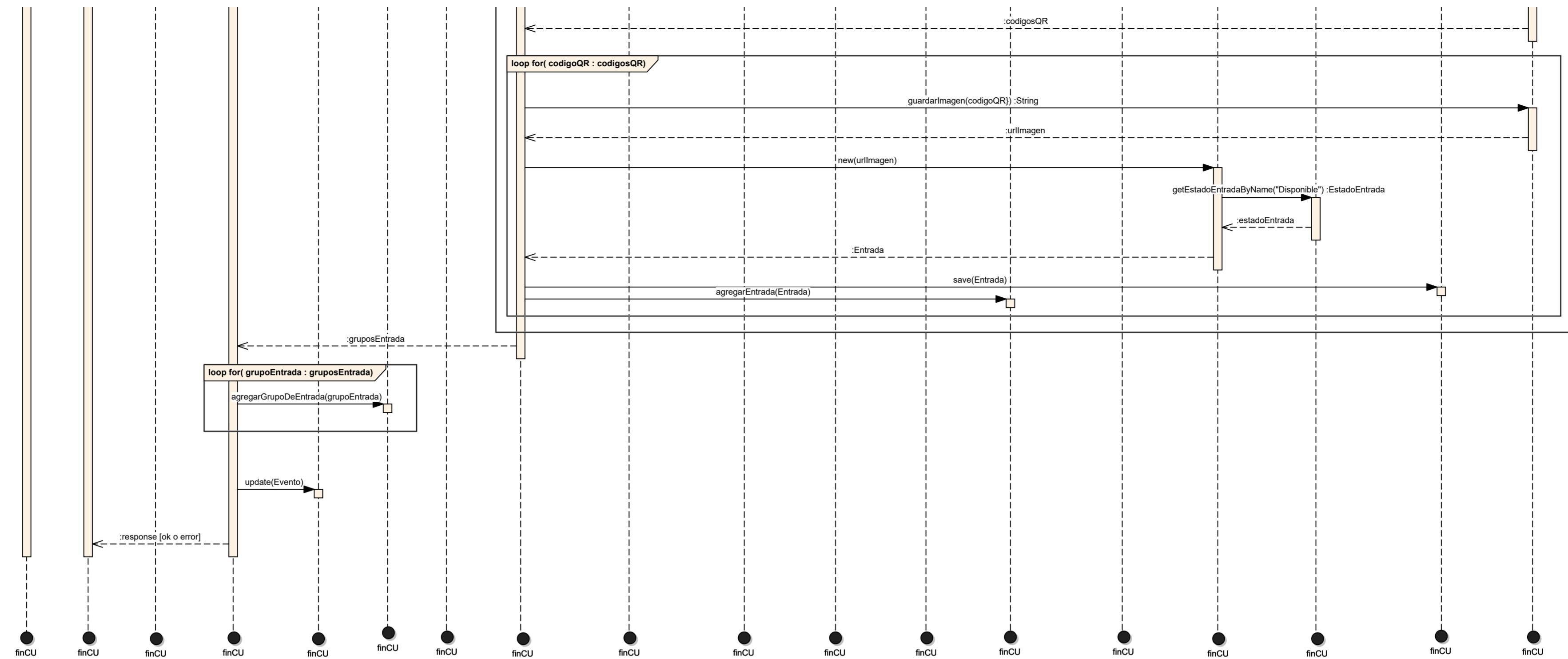


Diagrama de secuencia  
"Administrar entradas"





## Diagrama de secuencia "Confirmar evento"

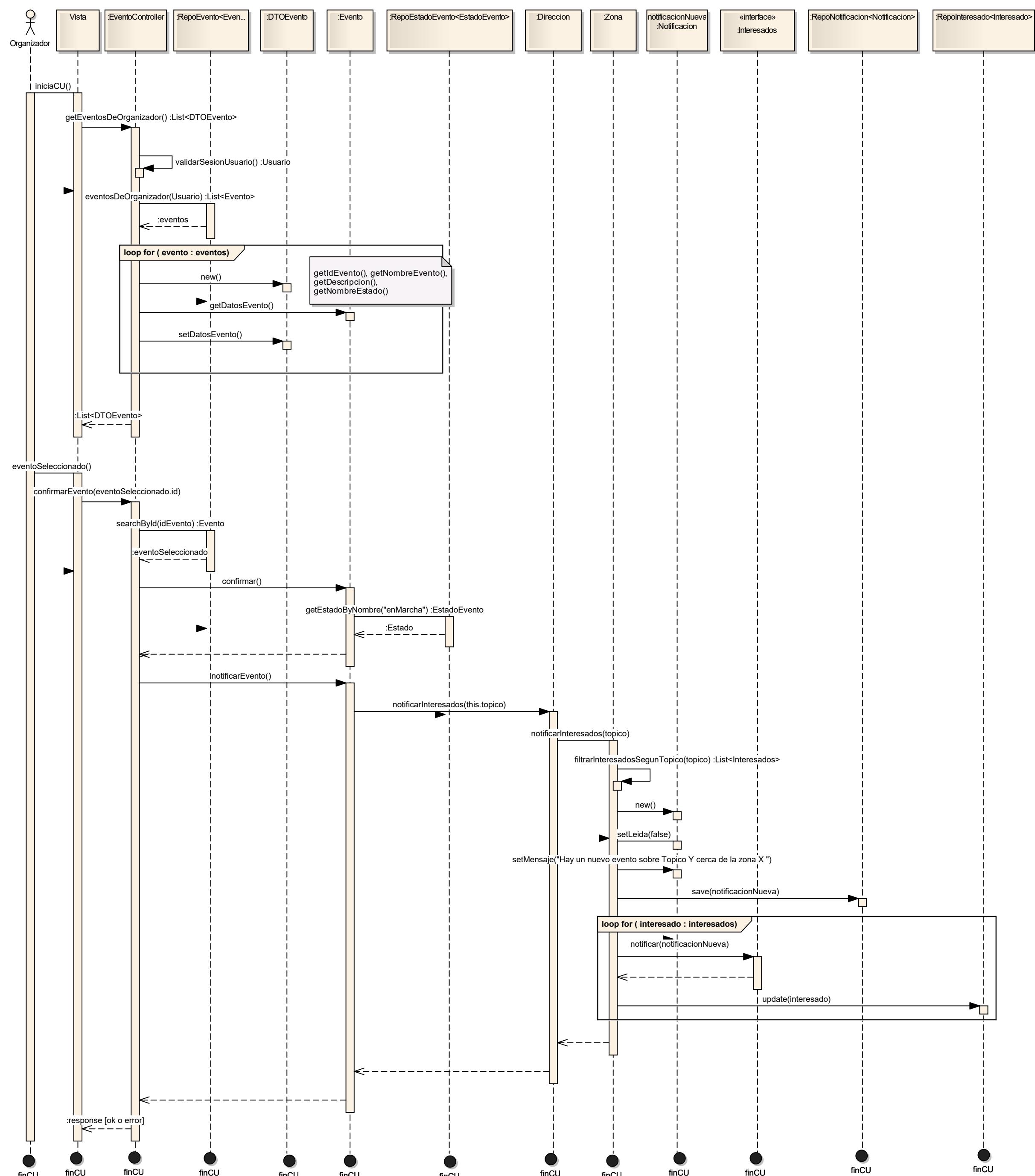
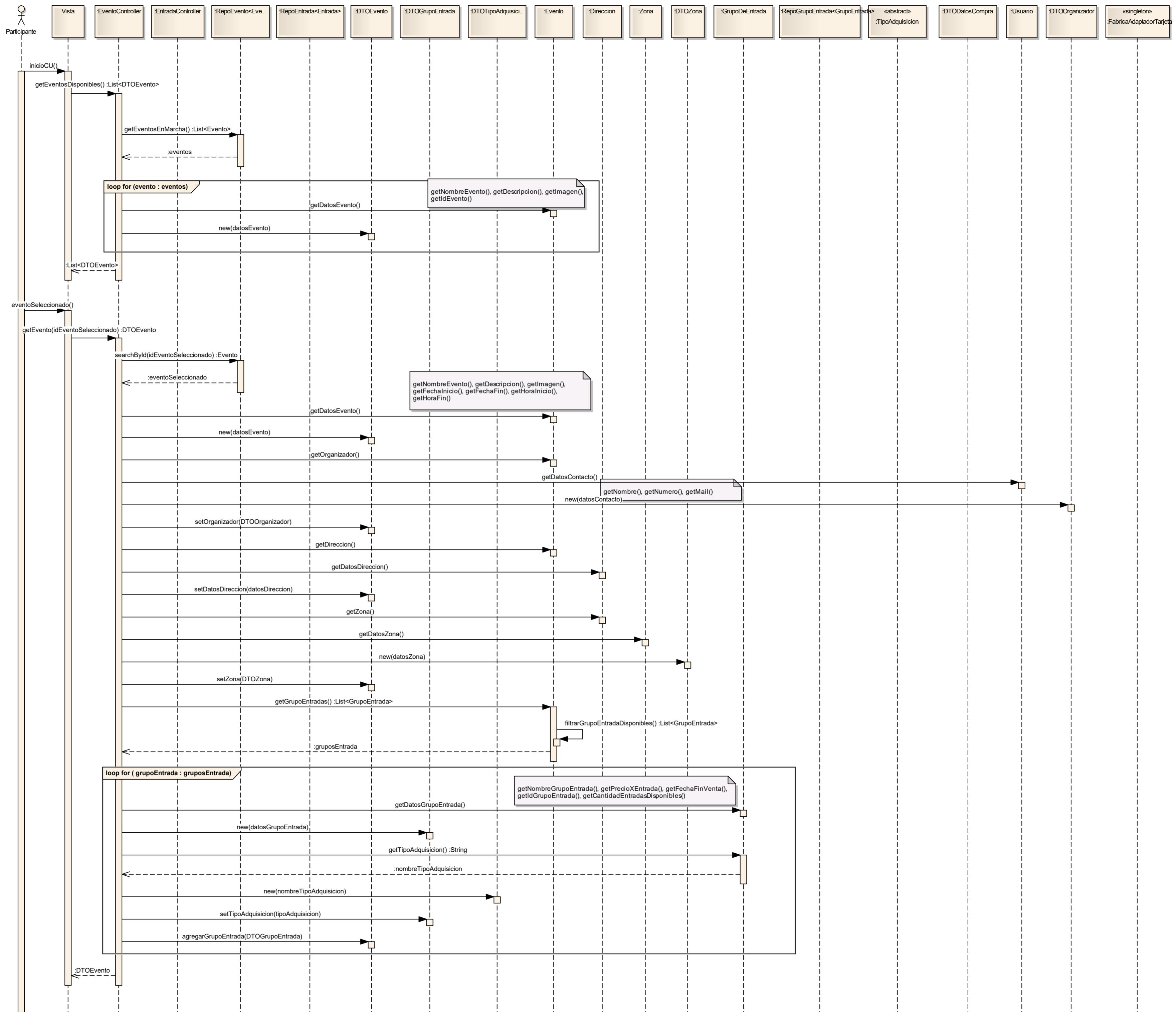
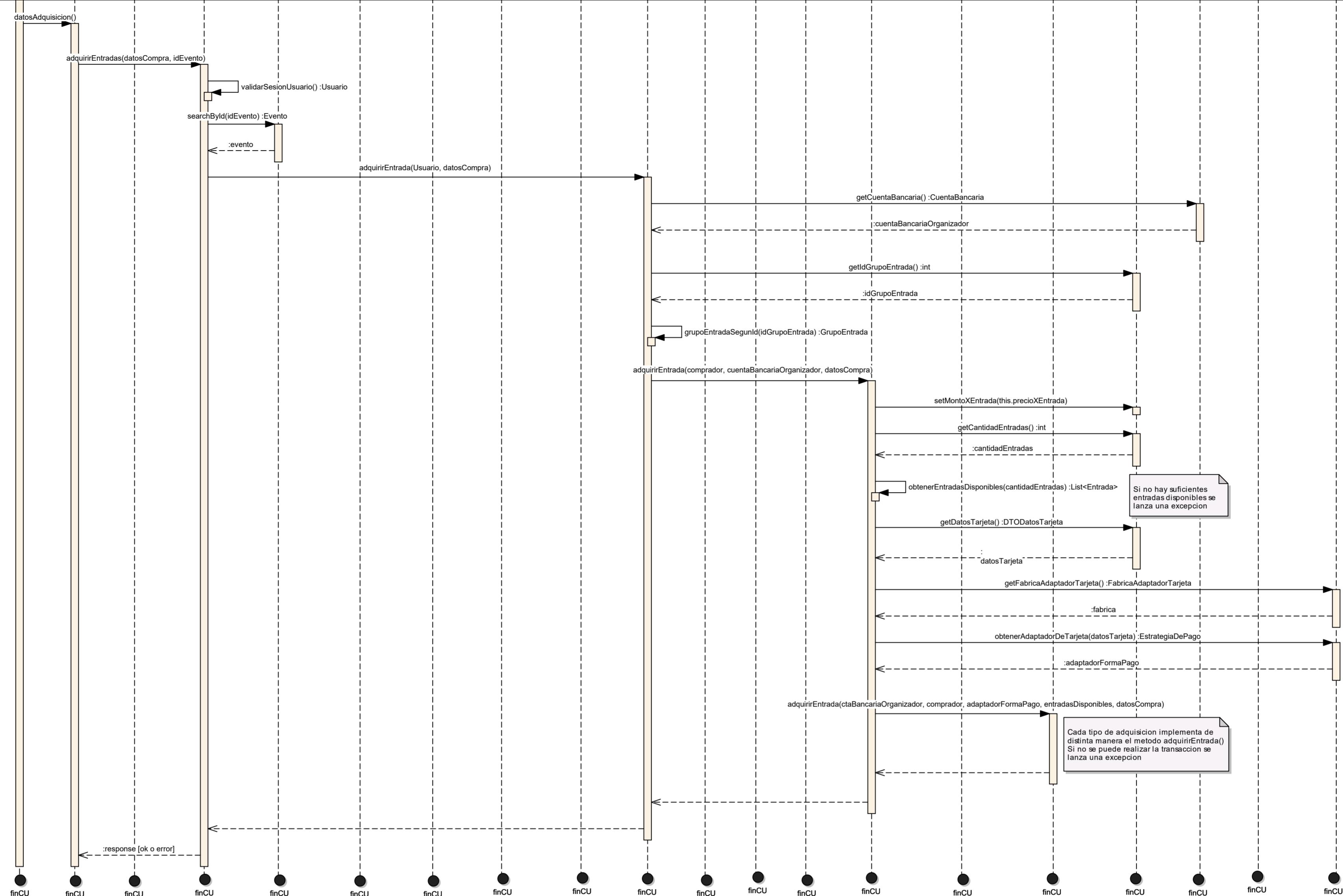


Diagrama de secuencia  
"Adquirir entrada"





## Código de las clases: TipoAdquisicion, Pago, Gratis y Donacion

Para mostrar la lógica de cómo el sistema maneja la adquisición de entradas lo hago a través de Java (se puede aplicar a cualquier lenguaje de programación orientada a objetos) porque considero que se entiende mucho mejor que con un diagrama de secuencia.  
En estas dos primeras imágenes muestro la clase abstracta TipoAdquisicion que define la lógica del método "adquirirEntrada" para que luego la implementen las clases "Pago", "Donacion" y para que la redefina "Gratis". Luego se define el método abstracto "getMonto" para que "Donacion" y "Pago" puedan definirlo.  
Excepciones: Si se desea "comprar" una entrada que no está disponible (porque otro usuario ya la adquirió) la clase Entrada lanzará una excepción que luego la manejo desde el método "adquirirEntrada". Por otra parte si no se puede validar o realizar el pago, las clases que implementan la interfaz "AdaptadorTarjeta", lanzan una excepción la cual manejo desde la clase "adquirirEntrada".  
El método "updateEntradas" me asegura una transacción atómica en la cual se actualizan todas las entradas y si ocurre algún error no se actualiza ninguna.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package adquirirentrada.TipoAdquisicion;

import adquirirentrada.Adaptador.AdaptadorTarjeta;
import adquirirentrada.DTO.DTODatosCompra;
import adquirirentrada.DTO.DTOPago;
import adquirirentrada.Entrada.CuentaBancaria;
import adquirirentrada.Entrada.Entrada;
import adquirirentrada.Entrada.Usuario;
import adquirirentrada.Excepciones.NoSePuedeAsignarLaEntradaException;
import adquirirentrada.Excepciones.NoSePuedeValidarPagoException;
import adquirirentrada.Repositorios.RepoCuentaBancaria;
import adquirirentrada.Repositorios.RepoEntrada;
import adquirirentrada.Repositorios.RepoTarifa;
import java.util.List;

/**
 *
 * @author nicolas
 */
public abstract class TipoAdquisicion {

    private String nombreTipoAdquisicion;
    private RepoTarifa repoTarifa;
    private RepoCuentaBancaria repoCuentaBancaria;
    protected RepoEntrada repoEntrada;

    public TipoAdquisicion(){
        this.repoTarifa= new RepoTarifa();
        this.repoCuentaBancaria= new RepoCuentaBancaria();
        this.repoEntrada=new RepoEntrada();
    }

    protected abstract int getMonto(DTODatosCompra datosCompra);

    protected DTOPago generarDatosPago(int monto,DTODatosCompra datosCompra,CuentaBancaria cuentaBancariaDestino){

        String cuentaBancaria=cuentaBancariaDestino.getCodigoCuenta();
        String nombreBanco=cuentaBancariaDestino.getNombreBanco();
        String nroTarjetaComprador=datosCompra.getDatosTarjeta().getNumeroTarjeta();
        String claveTarjetaComprador=datosCompra.getDatosTarjeta().getClaveTarjeta();
        return new DTOPago(cuentaBancaria,nombreBanco,nroTarjetaComprador,claveTarjetaComprador,monto);
    }

    protected int getMontoEventBrite(int montoTotal){
        int porcentajeIVA=this.repoTarifa.getTarifaByName("IVA").getPorcentaje();
        int porcentajeEB= this.repoTarifa.getTarifaByName("EventBrite").getPorcentaje();

        int montoEventBrite= montoTotal * porcentajeEB;
        montoEventBrite=montoEventBrite + montoEventBrite * porcentajeIVA;
        return montoEventBrite;
    }
}
```

```
|  
|  
public void adquirirEntrada(CuentaBancaria cuentaBancariaDestino, Usuario comprador, AdaptadorTarjeta adaptadorTarjeta, List<Entrada> entradas, DTODatosCompra datosCompra){  
  
    try{  
        entradas.forEach(entrada->entrada.comprar());  
        this.repoEntrada.updateEntradas(entradas);  
    }  
    catch(Exception e){  
        throw new NoSePuedeAsignarLaEntradaException("No se pudieron reservar todas las entradas requeridas por favor intentelo mas tarde");  
    }  
  
    int precioXEntrada=this.getMonto(datosCompra);  
    int montoTotal= precioXEntrada * entradas.size();  
  
    int montoEventBrite=this.getMontoEventBrite(montoTotal);  
    int montoOrganizador= montoTotal - montoEventBrite;  
  
    CuentaBancaria cuentaBancariaEventBrite= this.repoCuentaBancaria.getCuentaBancariaEventBrite();  
  
    DTOPago pagoEventBrite= this.generarDatosPago(montoEventBrite,datosCompra,cuentaBancariaEventBrite);  
    DTOPago pagoOrganizador= this.generarDatosPago(montoOrganizador,datosCompra,cuentaBancariaDestino);  
  
    try{  
        adaptadorTarjeta.validarPago(pagoEventBrite);  
        adaptadorTarjeta.validarPago(pagoOrganizador);  
        adaptadorTarjeta.realizarPago(pagoEventBrite);  
        adaptadorTarjeta.realizarPago(pagoOrganizador);  
  
        entradas.forEach(entrada->entrada.setDuegno(comprador));  
        this.repoEntrada.updateEntradas(entradas);  
  
    }catch(Exception e){  
        entradas.forEach(entrada->entrada.ponerEnDisponible());  
        this.repoEntrada.updateEntradas(entradas);  
        throw new NoSePuedeValidarPagoException("Hubo un error en la validacion del pago");  
    }  
}  
}
```

## Clase "Pago":

Luego la clase "Pago" lo único que tiene que hacer es heredar de "TipoAdquisicion" y definir el método "getMonto" que establece que el precio de cada entrada es el monto fijado por el grupo de entrada.

```
/*
 * To change this license header, choose License Headers in Project
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package adquirirentrada.TipoAdquisicion;

import adquirirentrada.DTO.DTODatosCompra;

/**
 *
 * @author nicolas
 */
public class Pago extends TipoAdquisicion{

    @Override
    protected int getMonto(DTODatosCompra datosCompra){
        return datosCompra.getMontoXEntrada();
    }
}
```

## Clase "Donacion":

La clase "Donacion" hace exactamente lo mismo que "Pago" con la diferencia que el método "getPago" retorna el monto donado que ingresó el usuario.

```
/*
 * To change this license header, choose License Headers in Project
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package adquirirentrada.TipoAdquisicion;

import adquirirentrada.DTO.DTODatosCompra;

/**
 *
 * @author nicolas
 */
public class Donacion extends TipoAdquisicion {

    @Override
    protected int getMonto(DTODatosCompra datosCompra){
        return datosCompra.getMontoDonadoXEntrada();
    }
}
```

Clase "Gratis":

Por último la clase "Gratis" redefine totalmente el método "adquirirEntrada" ya que lo único que debe hacer es asignarle directamente la entrada al usuario sin realizar ninguna transacción.  
Excepciones: Si se desea "comprar" una entrada que no está disponible (porque otro usuario ya la adquirió) la clase Entrada lanzará una excepción que luego la manejo desde el método "adquirirEntrada".

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package adquirirentrada.TipoAdquisicion;

import adquirirentrada.Adaptador.AdaptadorTarjeta;
import adquirirentrada.DTO.DTODatosCompra;
import adquirirentrada.Entrada.CuentaBancaria;
import adquirirentrada.Entrada.Entrada;
import adquirirentrada.Entrada.Usuario;
import adquirirentrada.Excepciones.NoSePuedeAsignarLaEntradaException;
import java.util.List;

/**
 *
 * @author nicolas
 */
public class Gratis extends TipoAdquisicion{

    @Override
    protected int getMonto(DTODatosCompra datosCompra){
        return 0;
    }

    @Override
    public void adquirirEntrada(CuentaBancaria cuentaBancariaDestino, Usuario comprador, AdaptadorTarjeta adaptadorTarjeta, List<Entrada> entradas, DTODatosCompra datosCompra){

        try{
            entradas.forEach(entrada->{
                entrada.comprar();
                entrada.setDueño(comprador);
            });
            this.repoEntrada.updateEntradas(entradas);
        }catch(Exception e){
            throw new NoSePuedeAsignarLaEntradaException("Hubo un error en la asignacion de la entrada vuelva a intentar mas tarde");
        }
    }
}
```