

Ingeniería del Software 2008/2009

ISParty

Organizador de eventos

GRUPO A - 4ºC

del Campo Montejo, Julián
Escobedo Ruiz, Ramón Eduardo
Feijoo Ugalde, Pablo
Miguel García, Manuel Javier
Martínez Ponte, Carmen
Nicolás Fúnez, Cristina
Rodríguez Acero, Fernando
Rodríguez Cerezo, Daniel
de Santos Sierra, Daniel
Valles Mercado, Héctor

Distribución de Tareas

Requisitos	Daniel S.
Diagramas de casos de uso	Todos
Diagramas de clases	Daniel C, Daniel S.
Precondiciones y postcondiciones	Fernando, Ramón
Invariantes OCL de las clases	Todos
Diagramas de estados	Julián
Diagramas de secuencias	Daniel C., Daniel S., Ramón
Diagramas de actividades	Todos
Diagramas de componentes	Daniel C.
Diagramas de seguridad	Ramón
Base de datos	Carmen, Cristina, Daniel S.
Diagrama de clase de la interfaz	Pablo
Apariencia de la interfaz	Héctor, Pablo
Invariantes de la interfaz	Pablo
Investigación de los componentes utilizados en el sistema	Julián
Pruebas	Carmen, Cristina, Manuel
Implementación	Julián

ÍNDICE

1. ESPECIFICACIÓN
 - 1.1. INTRODUCCIÓN
 - 1.2. REQUISITOS
 - 1.3. DIAGRAMAS DE CASO DE USO
 - 1.3.1. EMPLEADO Y JEFE
 - 1.3.2. CLIENTE, USUARIO Y NO USUARIO
2. DISEÑO
 - 2.1. DIAGRAMAS DE CLASES
 - 2.1.1. DIAGRAMA GENERAL
 - 2.1.2. DIAGRAMA P.A.R.E.
 - 2.2. MÉTODOS Y ATRIBUTOS DE DIAGRAMAS DE CLASES
 - 2.2.1. PRECONDICIONES Y POSTCONDICIONES
 - 2.3. INVARIANTES OCL DE LAS CLASES
 - 2.3.1. MENSAJE
 - 2.3.2. ARCHIVOS MULTIMEDIA
 - 2.3.3. LOCAL
 - 2.3.4. USUARIO
 - 2.3.5. EVENTO
 - 2.3.6. DENUNCIA
 - 2.3.7. RECIBO
 - 2.3.8. ARTISTA
 - 2.3.9. SUBCONTRATA
 - 2.4. DIAGRAMAS DE ESTADOS
 - 2.4.1. DIAGRAMA DE RECIBO
 - 2.4.2. DIAGRAMA DE EVENTO
 - 2.4.3. DIAGRAMA DE ARTISTA
 - 2.5. DIAGRAMAS DE SECUENCIAS
 - 2.5.1. CREACIÓN DE RECURSOS PREVIOS
 - 2.5.2. ELIMINAR EVENTO
 - 2.5.3. SOLICITAR EVENTO
 - 2.5.4. BANEAR USUARIO
 - 2.5.5. SOLICITAR MODIFICACION
 - 2.5.6. ACEPTAR MODIFICACION
 - 2.5.7. CANCELAR MODIFICACION
 - 2.5.8. FINALIZAR EVENTO
 - 2.6. DIAGRAMAS DE ACTIVIDADES
 - 2.6.1. ACTIVIDADES COMUNES USUARIO/CLIENTE
 - 2.6.1.1. REGISTRO USUARIO/CLIENTE
 - 2.6.2. ACTIVIDADES CLIENTE
 - 2.6.2.1. SOLICITAR EVENTO
 - 2.6.2.2. ACEPTAR EVENTO
 - 2.6.2.3. ACEPTAR MODIFICACIÓN
 - 2.6.2.4. MODIFICAR EVENTO
 - 2.6.3. ACTIVIDADES DE USUARIOS
 - 2.6.3.1. APUNTARSE A EVENTO
 - 2.6.3.2. DESAPUNTARSE DE EVENTO

- 2.6.3.3. VOTAR ARTISTA
- 2.6.3.4. SUBIR ARCHIVO MULTIMEDIA
- 2.6.3.5. DENUNCIAR CONTENIDO
- 2.6.4. ACTIVIDADES DEL EMPLEADO
 - 2.6.4.1. LOGUEO
 - 2.6.4.2. BANEAR USUARIO DE LA RED SOCIAL
 - 2.6.4.3. AÑADIR ARTISTA
- 2.6.5. ACTIVIDADES DE JEFE
 - 2.6.5.1. ANULAR EVENTO
- 2.6.6. ACTIVIDADES DE P.A.R.E.
 - 2.6.6.1. ASIGNACION DE RECURSOS
- 2.7. DIAGRAMAS DE COMPONENTES
- 2.8. DIAGRAMAS DE SEGURIDAD
 - 2.8.1. CLIENTE Y EMPLEADO RESPECTO A EVENTO
 - 2.8.2. EMPLEADO Y USUARIO RESPECTO A RED SOCIAL
 - 2.8.3. EMPLEADO RESPECTO A EVENTO
 - 2.8.4. USUARIO RESPECTO A EVENTO
- 2.9. BASE DE DATOS
 - 2.9.1. TABLAS DE LA BASE DE DATOS
 - 2.9.2. FUNCIONES DE LA BASE DE DATOS
- 3. INTERFAZ GRÁFICA
 - 3.1. DIAGRAMA DE CLASES DE INTERFAZ
 - 3.2. APARIENCIA DE LA INTERFAZ
 - 3.2.1. INTERFAZ EMPLEADO
 - 3.2.2. INTERFAZ JEFE
 - 3.2.3. INTERFAZ WEB DE USUARIO Y CLIENTE
 - 3.3. INVARIANTES DE LA INTERFAZ
- 4. INVESTIGACIÓN DE LOS COMPONENTES UTILIZADOS EN EL SISTEMA
 - 4.1. PROPUESTA PARA EL COMPONENTE DE BASES DE DATOS
 - 4.2. PROPUESTA PARA EL COMPONENTE DE PAGOS
 - 4.3. PROPUESTA PARA EL COMPONENTE MULTIMEDIA
- 5. PRUEBAS
 - 5.1. PRUEBAS DE USUARIO/CLIENTE
 - 5.2. PRUEBAS DE EMPLEADO
 - 5.3. PRUEBAS DE JEFE
- 6. IMPLEMENTACIÓN
 - 6.1. LENGUAJE UTILIZADO

1. ESPECIFICACIÓN

1.1. INTRODUCCIÓN

El proyecto es una aplicación informática dedicada a empresas que gestionan y organizan eventos. El jefe y los empleados la utilizarán para ese fin. Nuestro software incluye una base de datos, con sus correspondientes gestores, que manejan toda la información del sistema, como por ejemplo los artistas, los locales, las subcontratas, etc. También incluye una red social a la que podrá apuntarse cualquier persona, dichas personas serán consideradas usuarios de la red social y clientes. Esta red agilizará, economizará y optimizará, entre otras cosas, el proceso de publicitar los eventos. Cualquier persona podrá registrarse como cliente a través de nuestro portal y gestionar desde ahí sus propios eventos, sin necesidad de ir a la oficina de la empresa.

1.2. REQUISITOS

Para ser **usuario de la Red Social**:

- Registrarte como usuario de la Red Social
- Iniciar sesión como usuario

Una vez dentro de la Red Social serás considerado usuario de la misma y cliente y podrás realizar cada una de las siguientes actividades:

- Enviar mensaje privado a otros usuarios: sólo el emisor y el receptor pueden leerlos. Los mensajes tienen un tamaño límite.
- Enviar mensaje público a otros usuarios: los mensajes podrán leerlos cualquier usuario, accediendo a la parte pública del usuario receptor. Los mensajes tienen un tamaño límite.
- Enviar invitación: se generará un correo instantáneo a las direcciones dadas por el usuario.
- Borrar mensajes: sólo podrá borrar los mensajes en los que es el receptor
- Leer mensajes
- Denunciar mensaje: el usuario deberá dar un motivo concreto de la denuncia.
- Subir archivos multimedia: se pueden subir archivos multimedia que serán visibles a todos los usuarios de la Red Social. Los archivos multimedia no podrán superar un tamaño límite preestablecido y no habrán sido retirados por una denuncia. Estos archivos se visualizarán en la parte pública del usuario emisor.
- Visualizar archivo multimedia .
- Borrar archivo multimedia: del que seas emisor.
- Denunciar archivo multimedia: el usuario, al visualizar un archivo multimedia, podrá denunciarlo dando un motivo, que sea lo mas detallado posible.
- Votar artistas: si el usuario ha asistido a un evento, podrá votar una vez a cada artista por evento.
- Modificar el perfil: sólo aquellos datos que no afecten a la gestión, por ejemplo, se podrá cambiar el nick de usuario, pero no el Nombre.
- Apuntarse a eventos: un usuario podrá apuntarse a los eventos públicos que desee, mientras el aforo del evento no esté completo, en tal caso, pasará a una lista de espera, a la espera de que otro usuario que esté en la lista de usuarios que van al evento se desapunte. Tampoco podrá apuntarse si el usuario está apuntado a otro evento en la misma fecha. El usuario sólo podrá asistir al evento si está apuntado al evento y no está en la lista de espera. Posteriormente, habrá un control de los usuarios que han ido al evento y los que no, estos últimos serán denunciados automáticamente.
- Desapuntarse de eventos: dependerá del estado del evento. Si el evento aún puede estar sujeto a cambios, entonces el usuario podrá retirar la solicitud, dejando una solicitud libre para otro usuario, en cambio, si el evento está fuera de fecha de modificación, el usuario no podrá desapuntarse.
- Un usuario de la red social será considerado cliente, realice o no alguna vez un evento. Esto implica que, si el usuario quiere organizar un evento no se le pedirá otra vez los datos.
- Darse de baja: si no está apuntado a eventos o tiene algún evento en proceso como cliente
- Cerrar sesión

Denominamos **cliente** a aquella persona que contrata nuestros servicios.
Para ser cliente de nuestro gestor de eventos:

- Registrarte como cliente o ser usuario de la Red Social
- Iniciar sesión como cliente o tener una sesión de usuario abierta

Una vez dentro del gestor de eventos, un cliente podrá realizar las siguientes actividades:

- Solicitar evento: el cliente desea crear un evento, para ello, exigimos información, zona donde se realizaría, aforo máximo, aforo mínimo (por defecto será 0, pero puede ser que el cliente sólo quiera crear el evento si le aseguramos un número mínimo de personas), fecha de realización, etc. Se guardará un recibo de la solicitud.
 - Nuestro sistema realizará un precálculo según las condiciones del cliente y se las mostrará por pantalla. En este precálculo no se incluye el precio de las subcontratas, este se incluirá en el precio final. El cliente aceptará o no el precio orientativo y se enviará la solicitud al sistema para ser procesado por el empleado.
- La asistencia del cliente al evento es opcional. Si desea ir se le incluirá en la lista de invitados.
- Confirmar evento: mostrar un desglose del evento con el precio final, señal, etc. El cliente podrá aceptarlo o rechazarlo. Si lo rechaza, el evento se elimina o vuelve a un estado anterior, dependiendo de si la confirmación viene de la solicitud de un evento o de la modificación del mismo. Si lo acepta, el evento empezará a ser preparado una vez el cliente haya pagado la señal.
- Cancelar evento: el cliente sólo podrá eliminar un evento si está en el periodo de modificación. Si desea eliminarlo, estando en dicho periodo, entonces se le informará de la anulación del evento y si acepta el evento será eliminado, si no, el evento seguirá en marcha.
- Modificar evento: el cliente sólo podrá modificar algún aspecto del evento si está en el periodo de modificación. Se guardará un recibo de la modificación.
 - Nuestro sistema calculará la posibilidad del cambio y el coste aproximado del mismo. Por ejemplo, si modificamos la fecha, a lo mejor el artista no puede ir al evento en dicha fecha, entonces el cliente debería pagar, además, la señal para el nuevo artista. Si el cliente acepta el cambio, el empleado revisará la modificación y dará un nuevo presupuesto, hasta entonces, no podrá realizar más cambios.
- Visualizar estado de los eventos: el cliente, en cualquier momento podrá visualizar el estado del evento, para ello se le mostrará la cantidad que lleva pagada hasta el momento y la cantidad total a pagar, número de personas apuntadas al evento, etc. Además tendrá las opciones mencionadas en este apartado.
- Pagar señal: para que un evento pase a ser creado, el cliente deberá pagar una señal preestablecida en un plazo máximo.
- Pagar evento: el cliente paga todo el evento para que este se realice. El límite para pagar un evento es el periodo de modificación, si no se ha pagado antes de dicha fecha, el evento será cancelado.
- Un cliente será borrado si durante cierto tiempo no ha creado un evento, por ejemplo, durante un año.
- Darse de baja: si no tiene ningún evento en proceso. Si es usuario de la red social también será dado de baja en ella.
- Cerrar sesión

Para usar el programa de **Empleado**, una persona deberá:

- Iniciar sesión como empleado.

Una vez dentro del programa, el empleado podrá realizar las siguientes actividades:

- Consultar denuncias: El empleado puede consultar las denuncias por actuales (sin procesar), por denunciante, por denunciado,... El empleado podrá aceptar o rechazar denuncias conforme a su criterio o el criterio de la empresa. Con los datos que se le presentan podrá banear a un usuario(denunciado) o generar una denuncia por abuso(denunciante).
- Banear usuario: El empleado podrá banear a cualquier usuario que haya sido denunciado según su criterio o el criterio de la empresa. Si un usuario es baneado, será expulsado tanto de la red social como del organizador de evento. Para poder ser expulsado no debe tener ningún evento pendiente, por lo que si en un primer momento no puede ser eliminado, no se le dejará realizar ninguna operación ni como usuario ni como cliente, y en cuanto se finalicen los eventos será expulsado.
- Es el encargado de añadir, modificar y eliminar(si no están en ningún evento), mediante formulario, a un artista, local o subcontrata en la base de datos, así como hacer consultas.
- Visualizar mensajes del foro y archivos multimedia de la red social, ya sea para comprobar una denuncia u otros motivos.
- Podrá ver, de cualquier evento su estado, incluido lo que se lleva pagado hasta el momento.
- Consultar solicitudes de evento: el empleado consultará un evento solicitado, junto con la lista de posibles artistas y locales, pudiendo eliminar aquellos artistas o locales no disponibles. El empleado sopesará la necesidad de una subcontrata y escogerá las subcontratas de una lista recogida por el sistema.
 - Nuestro sistema calculará las subcontratas a partir de los requisitos establecidos por el empleado.

El empleado tendrá que dar el desglose final del evento.

El empleado podrá buscar manualmente artistas y locales para añadirlos a un evento.

- El empleado podrá anular un evento antes de confirmarlo, si el P.A.R.E. no le da alguna disponibilidad en algunos de los requisitos.
- Consultar eventos a modificar: actúa prácticamente igual que consultar solicitud de evento.
- Tramitar cancelación de evento: el cliente ha cancelado un evento y el empleado tiene que avisar a las partes contratadas, por lo que se mostrará un listado con las mismas.
- El empleado podrá ver datos y denunciar a un usuario.
- Salir del programa

Para usar el programa de **Jefe**, una persona deberá:

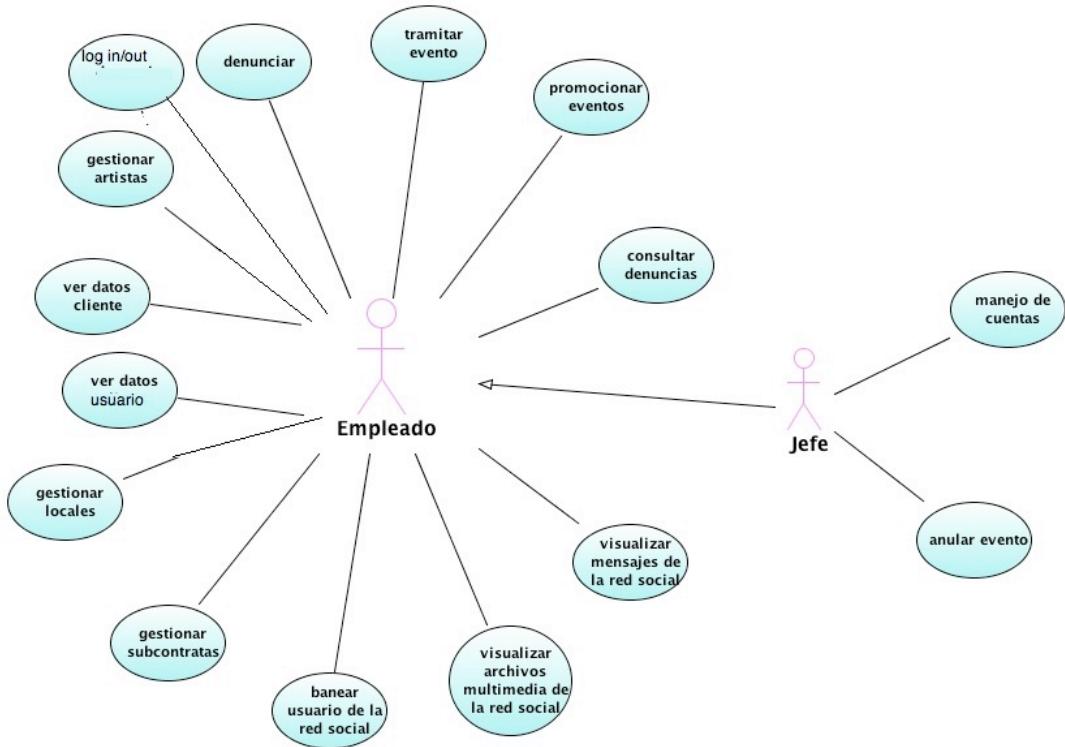
- Iniciar sesión como jefe.

Una vez dentro del programa, el jefe podrá realizar las siguientes actividades:

- El jefe podrá realizar todas las actividades del empleado.
- Tendrá la capacidad de anular un evento, aunque el cliente no haya solicitado su cancelación.
- Tendrá acceso a las cuentas de la empresa y podrá realizar gestiones con ellas.

1.3. DIAGRAMAS DE CASO DE USO

1.3.1. EMPLEADO Y JEFE



EMPLEADO

Se puede decir que este rol es el encargado de supervisar y de validar los principales puntos del sistema.

Sus Funciones son las de:

- **Log in/out:** identificarse
- **Visualizar mensajes de la Red social:** Permite ver los mensajes públicos de la red social.
- **Visualizar archivos multimedia de la red social:** Acceso a los archivos multimedia que han subido los usuarios.
- **Denunciar:** Hace referencia a la capacidad de poder emitir denuncias a determinados usuarios por usos incorrectos de la red social.
- **Consultar denuncias:** Ver las denuncias a los usuarios, a los mensajes o a los archivos.
- **Ver los datos del evento:** tales como la lista de invitados, lugar, fechas, artistas, si está pagado, si está aceptado por el cliente.
- **Gestionar artistas:** es decir modificar añadir y eliminar artistas al sistema, estos artistas podrán ser utilizados en los eventos.
- **Gestionar locales:** capacidad para ver y modificar las características de un local asociado al sistema. Estos locales también pueden ser utilizados en los eventos.
- **Gestionar Subcontratas:** Capacidad de ver, editar y eliminar las subcontratas asociadas al sistema. Estas están disponibles para los eventos que soliciten sus servicios.

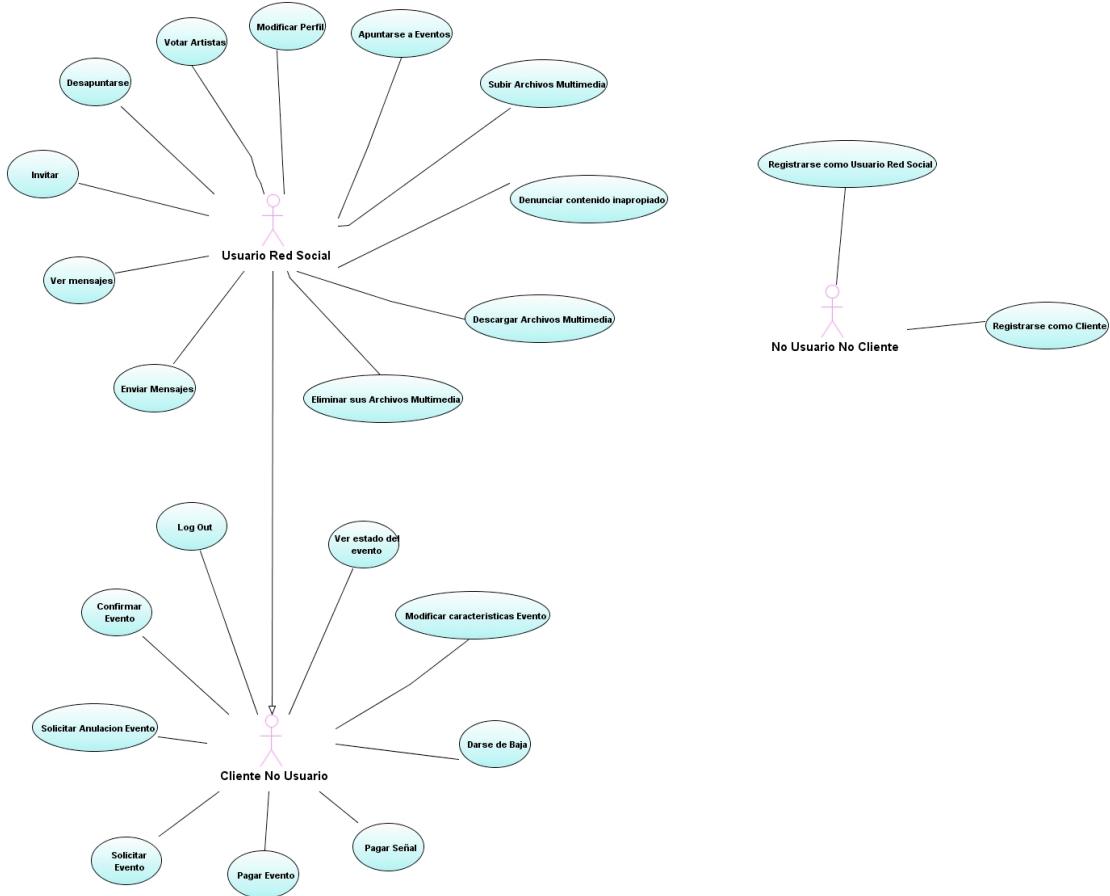
- **Promocionar Eventos:** Hace referencia a la capacidad del empleado de mandar publicidad a determinados usuarios de la red social, informando de eventos que pueden tener determinado interés para los usuarios.
- **Ver datos del cliente**
- **Ver datos del usuario de la red social**
- **Tramitar evento:** El empleado podrá atender solicitudes de evento (constará en reservar artistas, locales y las subcontratas que el empleado vea conveniente, según la solicitud del cliente y la ayuda de PARE), atender solicitudes de modificación de evento (ver en qué cambia el evento y actuar en consecuencia) y solicitudes de anulación de evento.
- **Ejecutar anulación de un evento:** Cuando un cliente pide que se le anule un evento que ha contratado el empleado verifica que esta acción es posible y ejecuta esta orden.
- **Banear usuario de la red social** Expulsar a un usuario de la red social.

Jefe

El jefe puede hacer todo lo que hace el empleado. También tiene capacidades propias que son:

- **Manejo de cuentas:** Capacidad para ver el estado de los movimientos de cuentas, y poder hacer transferencias.
- **Anular Eventos:** El Jefe puede anular un evento aunque el cliente no lo haya solicitado.

1.3.2. CLIENTE, USUARIO Y NO USUARIO



CLIENTE NO USUARIO

El cliente es aquel que solicita los servicios de ISPARTY,

- **Log in/out:** Entrar y salir del sistema de manera segura.
- **Solicitar un evento:** Pedir que se le organice un evento con unas determinadas características.
- **Confirmar evento:** Aceptar que se quiere encargar un evento, de acuerdo con las exigencias establecidas.
- **Solicitar anulación del evento:** Pedir que un evento que se ha solicitado sea anulado.
- **Modificar características del evento:** Mandar una solicitud de que su evento sea modificado.
- **Ver estado del evento:** Consultar el estado de tramitación de su evento y datos del mismo.
- **Pagar la señal del evento:** El cliente deberá pagar una señal para que se empiece a contratar a las distintas partes (artista, local,...).
- **Pagar el evento:** Puede pagar el resto del evento.
- **Darse de baja:**

USUARIO DE LA RED SOCIAL

Todo usuario que este registrado en nuestra red social puede, comportarse como un cliente y además puede hacer lo siguiente.

- **Modificar el perfil:** Cambiar sus datos personales no relevantes, por ejemplo, nick.
- **Enviar mensajes:** Puede enviar mensajes públicos o privados a otros usuarios de la red social.
- **Ver mensajes:** Dependiendo si el mensaje es público o privado lo podrá ver todo el mundo o el receptor, respectivamente, además del emisor.
- **Invitar:** Un usuario puede mandar invitaciones para entrar a formar parte de la red social.
- **Votar artistas:** Puede dar su opinión en forma de votos, acerca de los distintos artistas asociados a un evento al que haya asistido.
- **Apuntarse a un evento:** Si el evento es público el usuario puede apuntarse a este.
- **Desapuntarse de un evento:**
- **Subir, descargar y eliminar archivos multimedia:**
- **Denunciar contenido inapropiado:** Un usuario puede avisar acerca de un mensaje o archivo inapropiado.

NO CLIENTE NO USUARIO

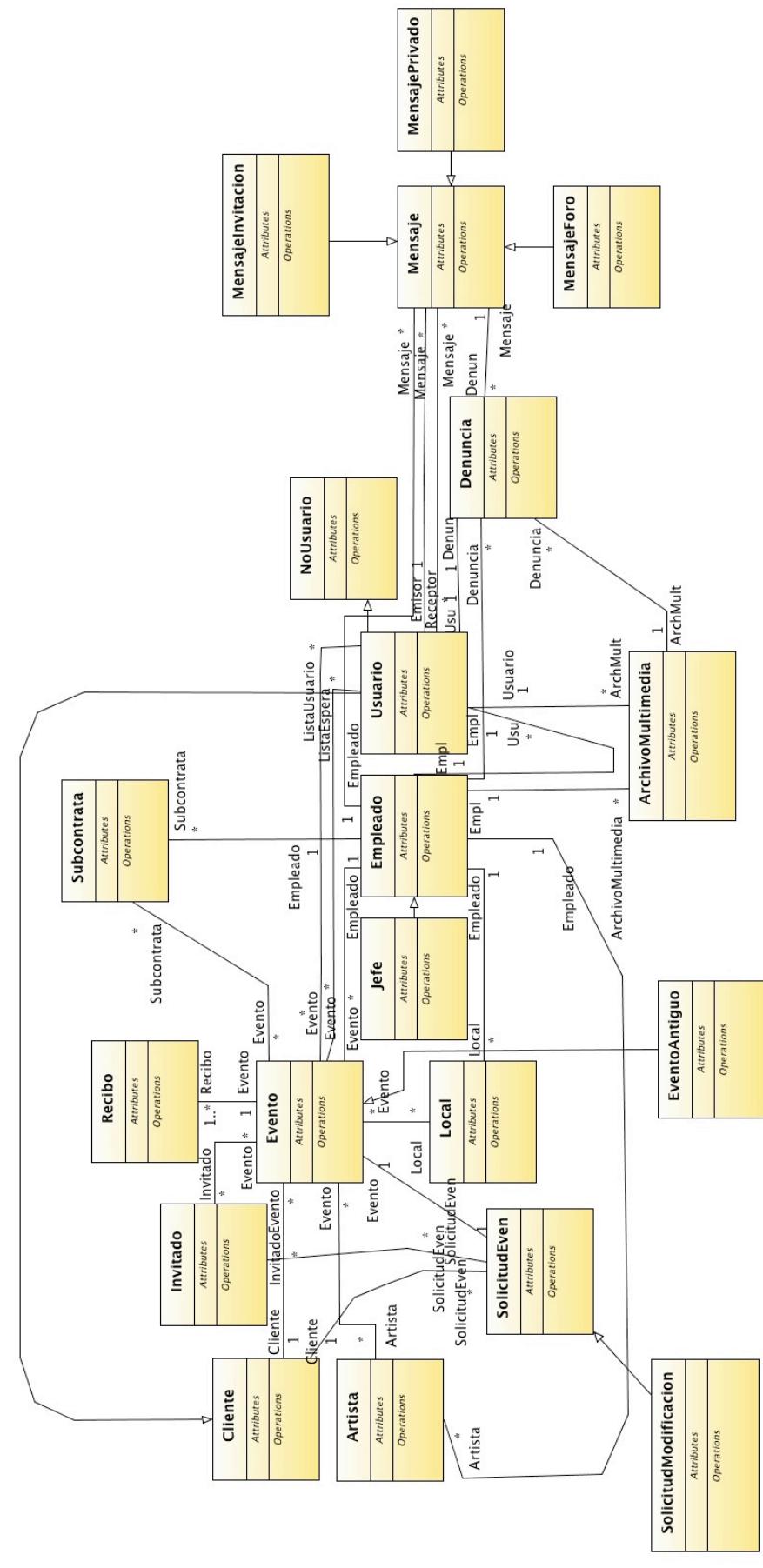
Este rol hace referencia a toda persona ajena a nuestro sistema. Puede hacer:

- **Registrarse como usuario de la red social:** Necesario para formar parte de la red social
- **Registrarse como cliente:** Necesario para poder contratar los servicios de ISPARTY

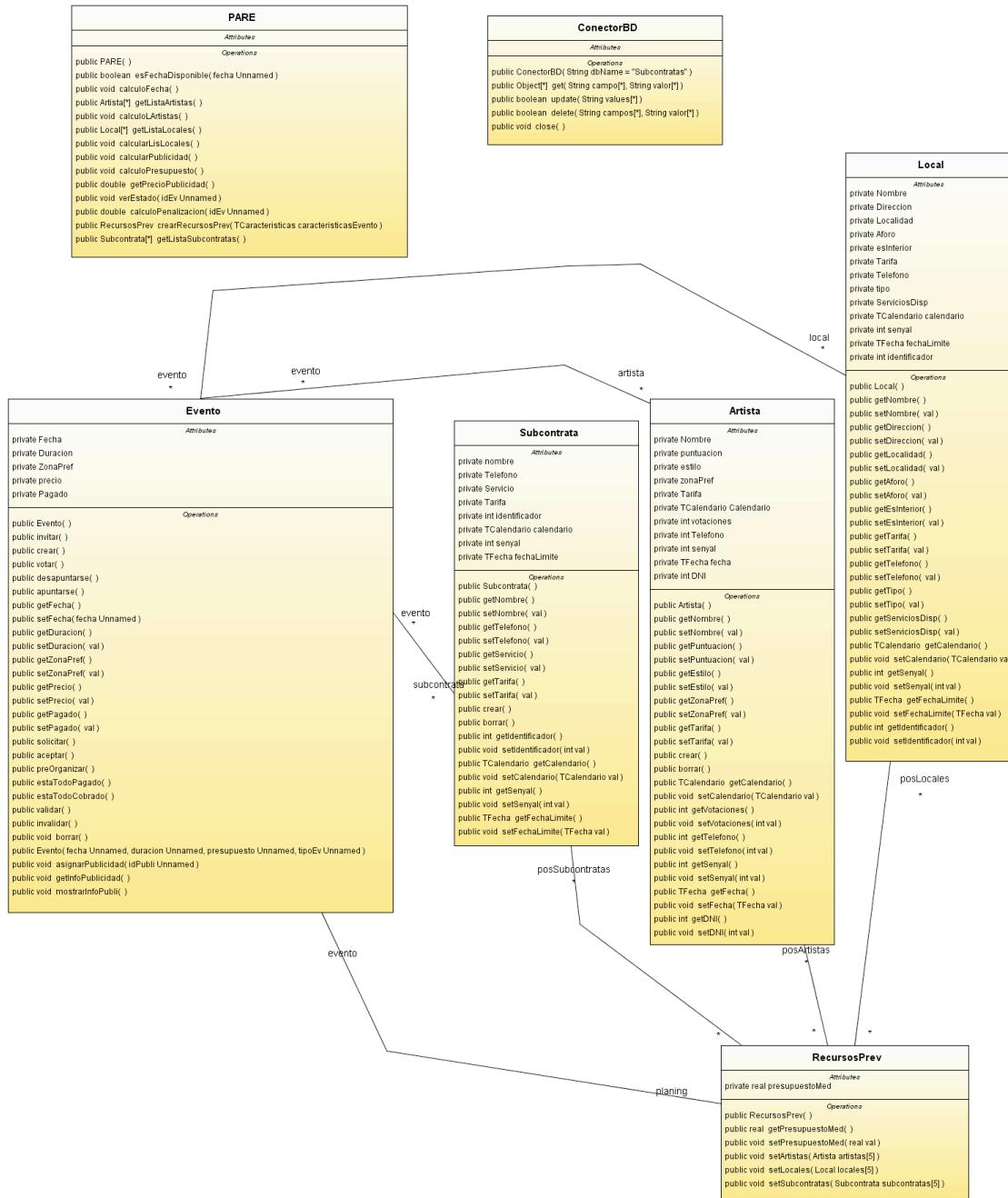
2. DISEÑO

2.1. DIAGRAMAS DE CLASES

2.1.1 DIAGRAMA GENERAL



2.1.2 DIAGRAMA P.A.R.E.



2.2. MÉTODOS Y ATRIBUTOS DE DIAGRAMAS DE CLASES

NOTA: Todas las clases tienen accesoras y mutadoras de cada atributo, a excepción de los password que no tendrá accesos y el mutador será setPassword(String passAntigua, String passNueva)

<<enumerado>> TFECHA

```
private int dia  
private int mes  
private int año  
private int hora  
private int minuto
```

<<enumerado>> TCALENDARIO

```
private boolean[7] dia_semana
```

<<enumerado>> TESTADOEVENTO

```
solicitado  
esperaAceptada  
pendiente  
creado  
bloqueadoCreado  
cancelado  
solicitudCancelacion  
Pagado  
BloqueadoPagado  
NoModificado  
Celebrado
```

<<enumerado>> TUSUARIO

```
normal  
baneado
```

ARTISTA

ATRIBUTOS

- private String Nombre
- private int puntuacion
- private int votaciones
- private String estilo
- private int zonaPref
- private float Tarifa
- private TCalendario Calendario
- private int Telefono
- private float senyal
- private int fechaLimite
- private int DNI

METODOS

- void crearArtista(int DNI, String nombre, String estilo, TCalendario Calendario, int zonaPref, float Tarifa, int Telefono, float senyal, int fechaLimite)
- void eliminarArtista(int DNI)
- void modificarArtista(int DNI, String estilo, TCalendario Calendario, int zonaPref, float Tarifa, int Telefono, float senyal)
- void votarArtista(int puntuacion)
- void reservar (String estilo, int fechaLimite)
- void contratar(float cantidad)

- void cancelaCliente()
- void cancelarReserva()

LOCAL

ATRIBUTOS

- private String Nombre
- private String Direccion
- private int Localidad
- private int aforo
- private boolean esInterior
- private float tarifa
- private int telefono
- private String tipo
- private String ServiciosDisp[]
- private TCalendario calendario
- private float senyal
- private int fechaLimite
- private int identificador

METODOS

- void crearLocal(String Nombre, String Direccion, int localidad, int aforo, boolean esInterior, float tarifa, int telefono, String tipo, String[] ServiciosDisp, TCalendario calendario, float senyal, int fecha)
- void eliminarLocal()
- void modificarLocal(String nombre, float tarifa, int telf, String tipo, TCalendario calendario, float senyal)

CLIENTE

ATRIBUTOS

- private String Nombre
- private String Apellidos
- private int DNI
- private String Direccion
- private int Telefono
- private String password

METODOS

- void crearCliente(int DNI, String nombre, String apellido, String Direccion, int Telefono, String password)
- void modificarCliente(int DNI, String Direccion, int Telefono, String passwordAntigua, String passwordNueva)
- void eliminarCliente(int DNI)
- void verificarContraseñaCliente(int DNI, String password)

RECIBO

ATRIBUTOS

- private String cuentaEmisor
- private String cuentaReceptor
- private TFecha fecha
- private float cantidad

METODOS

- void crearRecibo(float cantidad, String cuentaReceptor)
- void pagarRecibo(float suma)
- void cancelarRecibo()

SOLICITUDEVEN

ATRIBUTOS

- private TFecha fecha
- private float duracion
- private int identificador
- private int Localidad
- private boolean esPublico
- private int aforoMin
- private int aforoMax
- private String TipoEvento
- private int numArtistas
- private int numLocales

METODOS

- crearSolicitud (Tfecha fecha, float duracion, int localidad, boolean esPublico, int aforoMin, int aforoMax, String tipoEvento, int numArtistas, int numLocales)
- setRecibo (SolicitudEven r)
- getRecibo()

SOLICITUDMODIFICACION

ATRIBUTOS

METODOS

- efectuarModificacion()

EVENTO

ATRIBUTOS

- private TFecha fecha
- private float duracion
- private int identificador
- private int Localidad
- private boolean esPublico
- private int aforoMin
- private int aforoMax
- private TEstadoEvento estado
- private String direccion
- private String TipoEvento
- private float precioFinal
- private int fechaLimite
- private int senyal
- private Tfecha fechaSolicitudModificacion

METODOS

- votarArtista(int idenUsuario, idenArtista, idenEvento, voto)
- crearEvento(int idCliente, int[] idArtistas, int[] idLocales, int[] idSubcontratas, boolean esPublico, TFecha fecha, int aforoMin, int aforoMax, float precioFinal, String TipoEvento, float duracion, int zona, String direccion, int fechaLimite, Tfecha fechaSolicitudModificacion)
- solicitarEvento(int idCliente, int[] idArtistas, int[] idLocales, int[] idSubcontratas, boolean esPublico, TFecha fecha, int aforoMin, int aforoMax, float precioFinal, String TipoEvento, float duracion, int zona, String direccion)
- modificarEvento(int idCliente, int[] idArtistas, int[] idLocales, int[] idSubcontratas, boolean esPublico, TFecha fecha, int aforoMin, int aforoMax,

- float precioFinal, String TipoEvento, float duracion, int zona, String direccion)
- pagarCantidad(int idenEvento, float dinero)
- pagarSenyal (int idenEvento, float Senyal)
- cambiarEstadoEvento(int idenEvento, int estado)
- enviarPublicidadEvento()
- apuntarseAEvento(int DNI)
- desapuntarseDeEvento(int DNI)
- eliminarArtistaNoDisponible(int idenArtista)
- eliminarRestoDeArtistas(int idenArtistas)
- añadirArtista(int idenArtista)
- eliminarSubcontrataNoDisponible(int idenSucontrata)
- eliminarRestoDeSubcontratas(int idenSucontrata)
- añadirSubcontrata(int idenSubcontrata)
- eliminarLocalNoDisponible(int idenLocal)
- eliminarRestoLocales(int idenLocal)
- añadirLocal(int idenLocal)
- crearListaInvitados(String[] Nombres, String[] Apellidos, int[] DNIs)
- eliminarListaInvitados()
- invitarAEvento()
- enviarPresupuesto(float cantidad)
- boolean aceptarPresupuesto()
- pagarModificaciones(int cantidad)
- Cancelar()
- asistirAEvento(int Usuario)
- confirmarEvento()
- tramitarCancelacion()

INVITADO

ATRIBUTO

- private String Nombre
- private String Apellidos
- private int DNI

METODOS

- añadirInvitado(String Nombre, String Apellido, int DNI)
- eliminarInvitado()

SUBCONTRATA

ATRIBUTO

- private String nombre
- private int telefono
- private String servicio
- private float tarifa
- private int identificador
- private TCalendario calendario
- private float senyal
- private int fechaLmite

METODOS

- void crearSubcontrata(int ident, String nombre, int telefono, String servicio, float tarifa, TCalendario calendario, float senyal, int fechaLmite)
- void modificarSubcontrata(int ident, String nombre, int telefono, String servicio, float tarifa, TCalendario calendario, float senyal, int fechaLmite)

- void eliminarSubcontrata(int ident)

EMPLEADO

ATRIBUTOS

- private String Nombre
- private String Apellidos
- private int DNI
- private String Direccion
- private int Telefono
- private String password

METODOS

- void crearEmpleado(int DNI, String nombre, String apellido, String Direccion, int Telefono, String password)
- void eliminarEmpleado(int DNI)
- void verificarContraseñaEmpleado(int DNI, String password)
- void banearUsuario(int DNI)

Jefe

ATRIBUTOS

- private String Nombre
- private String Apellidos
- private int DNI
- private String Direccion
- private int Telefono
- private String password

METODOS

- void crearJefe(int DNI, String nombre, String apellido, String Direccion, int Telefono, String password)
- void eliminarJefe(int DNI)
- void verificarContraseñaJefe(int DNI, String password);

USUARIO

ATRIBUTOS

- private String nombre
- private String apellido
- private int dni
- private String nick
- private String password
- private String gustos[]
- private int numDenuncias
- private tUsuario estado

METODOS

- void crearUsuario(String nombre, String apellido, int DNI, String nick, String password, String[] gustos)
- void modificarUsuario(int DNI, String nick, String passwordAntigua, String passwordNueva, String[] gustos)
- boolean verificarContraseñaUsuario(int DNI, String password)
- void eliminarUsuario()
- void denunciar()

NoUSUARIO

ATRIBUTOS

METODOS

ARCHIVOMULTIMEDIA

ATRIBUTOS

- const tamMax
- private int tamanyo
- private String tipo
- private Tfecha fecha
- private int identificador
- private String url

METODOS

- void subirArchivoMultimedia(int tamanyo, String tipo, Tfecha fecha, String url)
- void borrarArchivoMultimedia()
- void consultarArchivoMultimedia()
- void denunciar()

DENUNCIA

ATRIBUTOS

- private int identificador
- private String motivo
- private TFecha fecha
- private int resultado

METODOS

- crearDenuncia(String motivo)
- aceptarDenuncia()
- rechazarDenuncia()

MENSAJE

ATRIBUTOS

- const tamMax
- private String textoMensaje
- private int emisor
- private int[] receptor
- private int tamanyo
- private int identificador
- private boolean leido
- private boolean esPublico
- private TFecha fecha

METODOS

- void enviarMensaje(String textoMensaje, int emisor, int[] receptor, int tamanyo, boolean esPublico)
- String leerMensaje()
- void borrarMensaje()
- denunciar()

MENSAJE PRIVADO

METODOS

- Public MensajePrivado responder (String textoMensaje, int emisor, int[] receptor, int)

MENSAJE FORO

METODOS

- Public mensajePrivado()

MENSAJE INVITACION

ATRIBUTOS

- Private String[] receptores

METODOS

- Public Invitación()
- Public enviarInvitaciones(String[] receptores, String textoMensaje, int emisor)

2.2.1. PRECONDICIONES Y POSTCONDICIONES

ARTISTA

votarArtista(): Se podrá votar a un artista siempre que el evento donde ha participado ya se ha celebrado.

Pre:

Self.Evento->forAll(e:Evento|e.estado=celebrado)

EVENTO

crearEvento(int idCliente, int[] idArtistas, int[] idLocales, int[] idSubcontratas, boolean esPublico, TFecha fecha, int aforoMin, int aforoMax, float precioFinal, String TipoEvento, float duracion, int zona, String direccion, int fechaLimite, Tfecha fechaSolicitudModificacion)

Pre

El aforo minimo siempre es menor o igual que el aforo máximo
context Evento:

Evento.allInstances → forAll(e1: Evento | (0 <= e1.aforoMin) and (e1.aforoMin <= e1.aforoMax))

Post

La direccion del evento y la direccion del local deben ser la misma

Context Evento

Self.allInstances → (e1: Evento | e1.Local.direccion = e1.direccion)

Todos los locales de un evento están en la misma localidad, la del evento:

context Evento
self.allInstances→forAll(e:Evento|e.local→forAll(loc:Local|self.Localidad=loc.Localidad))

El tipo de artista y el tipo de evento iguales.

context Evento:
self.allInstances → forAll(e1: Evento | e1.Artista->forAll(a1: Artista | e1.TipoEvento = a1.estilo))

Si es privado no tiene ni lista de usuarios ni lista de espera

Contex Evento

Self.allInstances → forAll(e:Evento | e.esPublico == false implies ((e.listaUsuario→size() + e.listaEspera-> size())==0))

solicitarEvento(int idCliente, int[] idArtistas, int[] idLocales, int[] idSubcontratas, boolean esPublico, TFecha fecha, int aforoMin, int aforoMax, float precioFinal, String TipoEvento, float duracion, int zona, String direccion)

Pre

El aforo minimo siempre es menor o igual que el aforo máximo
context Evento:

Evento.allInstances → forAll(e1: Evento | (0 <= e1.aforoMin) and (e1.aforoMin <= e1.aforoMax))

Post

La dirección del evento y la dirección del local deben ser la misma

Context Evento
Self.allInstances → (e1: Evento | e1.Local.direccion = e1.direccion)

Todos los locales de un evento están en la misma localidad, la del evento:

context Evento
self.allInstances→forAll(e:Evento|e.local→forAll(loc:Local|self.Localidad=loc.Localidad))

El tipo de artista y el tipo de evento iguales.

context Evento:
self.allInstances → forAll(e1: Evento | e1.Artista->forAll(a1: Artista | e1.TipoEvento = a1.estilo))

Si es privado no tiene ni lista de usuarios ni lista de espera

Contex Evento
Self.allInstances → forAll(e:Evento | e.esPublico == false implies ((e.listaUsuario→size() + e.listaEspera-> size())==0)

modificarEvento(int idCliente, int[] idArtistas, int[] idLocales, int[] idSubcontratas, boolean esPublico, TFecha fecha, int aforoMin, int aforoMax, float precioFinal, String TipoEvento, float duracion, int zona, String direccion)

Pre

El aforo mínimo siempre es menor o igual que el aforo máximo

context Evento:
Evento.allInstances → forAll(e1: Evento | (0 <= e1.aforoMin) and (e1.aforoMin <= e1.aforoMax))
Self.estado=creado and self.solicitudMod->size()>0

Post

La dirección del evento y la dirección del local deben ser la misma

Context Evento
Self.allInstances → (e1: Evento | e1.Local.direccion = e1.direccion)

Todos los locales de un evento están en la misma localidad, la del evento:

context Evento
self.allInstances→forAll(e:Evento|e.local→forAll(loc:Local|self.Localidad=loc.Localidad))

El tipo de artista y el tipo de evento iguales.

context Evento:
self.allInstances → forAll(e1: Evento | e1.Artista->forAll(a1: Artista | e1.TipoEvento = a1.estilo))

Si es privado no tiene ni lista de usuarios ni lista de espera
`modificarEvento()`
`Self.estado=bloqueadoCreado`

`Contex Evento`
`Self.allInstances → forAll(e: Evento | e.esPublico == false implies`
`((e.listaUsuario->size() + e.listaEspera-> size())==0)`

`pagarSenyal` (int idenEvento, float Senyal)

`Pre`

`La señal tiene que ser menor o igual que el precioFinal`
`context Evento:`
`Evento.allInstances → forAll(e1: Evento | e1.senyal <=`
`e1.precioFinal)`

`El evento creado cumpla los requisitos de la solicitud. Es al pagar la señal cuando el evento pasa a estado creado.`

`self.estado=pendiente`

`Post:`

`self.estado= creado and self.fecha=self.SolicitudEven.fecha and`
`self.duracion=self.SolicitudEven.duracion and`
`self.Localidad=self.SolicitudEven.Localidad and`
`self.esPublico=self.SolicitudEven.esPublico and`
`self.aforoMin=self.SolicitudEven.aforoMin and`
`self.aforoMax=self.SolicitudEven.aforoMax and`
`self.TipoEvento=self.SolicitudEven.TipoEvento and`
`self.numArtistas=self.SolicitudEven.numArtistas and`
`self.numLocales=self.SolicitudEven.numLocales`

`enviarPublicidadEvento()`

`Pre:`

`El evento debe ser público`
`Contex Evento inv`
`Self.allInstances → forAll(e: Evento | e.esPublico == true)`

`apuntarseAEvento()`

`Pre:`

`El evento no puede estar cancelado`
`Contex Evento inv`
`Self.allInstances → forAll(e: Evento | e.esPublico == false)`

`añadirArtista`(int idenArtista)

`Pre:`

`El tipo de artista y el tipo de evento iguales.`
`context Evento:`
`self.allInstances → forAll(e1: Evento | e1.Artista->forAll(a1: Artista |`
`e1.TipoEvento = a1.estilo))`

`crearListaInvitados`(String[] Nombres, String[] Apellidos, int[] DNI)

`Pre:`

`El evento no puede estar cancelado`
`Contex Evento inv`
`Self.allInstances → forAll(e: Evento | e.esPublico == false)`

`invitarAEvento()`

`Pre:`

`El evento no puede estar cancelado`

Contex Evento inv
Self.allInstances → forAll(e:Evento | e.esPublico == false)

confirmarEvento()

Pre:

El evento no puede estar cancelado

Contex Evento inv

Self.allInstances → forAll(e:Evento | e.esPublico == false)

enviarPresupuesto() : La asignación de recursos del Evento se ha hecho conforme a las características del Evento

Pre:

Self.estado=solicitado

Post:

Self.estado=esperaAceptado and
Self.Locales->size()>=self.numLocales and
Self.Artistas->size()>=self.numArtistas and
Self.Locales->forAll(l:Local| l.aforo>=self.aforoMax and
l.Localidad=self.Localidad and l.tipo=self.TipoEvento and
l.calendario[self.fecha]=true) and
Self.Artistas->forAll(a:Artista| a.estilo=self.TipoEvento and
a.zonaPref=self.Localidad and a.calendario[self.fecha]=true)
and
Self.Subcontratas→forAll(s:Subcontrata|
s.calendario[self.fecha]=true)

pagarModificaciones(): Pagar las modificaciones implica que el Evento sale del bloqueo y los datos actuales del evento son coherentes con lo pedido en la solicitud de modificación.

Pre:

Self.estado=bloqueadoCreado

Post:

Self.estado=creado and

self.fecha=self.SolicitudMod.fecha and
self.duracion=self.SolicitudMod.duracion and
self.Localidad=self.SolicitudMod.Localidad and
self.esPublico=self.SolicitudMod.esPublico and
self.aforoMin=self.SolicitudMod.aforoMin and
self.aforoMax=self.SolicitudMod.aforoMax and
self.TipoEvento=self.SolicitudMod.TipoEvento and
self.numArtistas=self.SolicitudMod.numArtistas and
self.numLocales=self.SolicitudMod.numLocales and
// y que la asignación de recursos siga siendo también
//coherente
Self.Locales->size()>=self.solicitudMod.numLocales and
Self.Artistas->size()>=self.solicitudMod.numArtistas and
Self.Locales->forAll(l:Local|
l.aforo>=self.solicitudMod.aforoMax and
l.Localidad=self.solicitudMod.Localidad and
l.tipo=self.solicitudMod.TipoEvento and
l.calendario[self.solicitudMod.fecha]=true) and
Self.Artistas->forAll(a:Artista|
a.estilo=self.solicitudMod.TipoEvento and
a.zonaPref=self.solicitudMod.Localidad and
a.calendario[self.solicitudMod.fecha]=true) and
Self.Subcontratas
→forAll(s:Subcontrata| s.calendario[self.solicitudMod.fecha]=true)

USUARIO

crearUsuario(String nombre, String apellido, int DNI, String nick, String password, String[] gustos)

Pre:

El password, el nombre, el apellido, el DNI y el nick de un usuario deben ser diferentes de cadena vacía.

context Usuario:

Self.allInstances → forAll(u1: Usuario |
u1.nombre<> "" and u1.apellido<> "" and u1.nick<> "" and
u1.password<> "" and u1.dni<> "")

Un usuario debe haber introducido al menos un gusto.

context Usuario:

Self.allInstances → forAll(u1: Usuario | u1.gustos.size()>=1)

Post:

Dos usuarios no pueden tener el mismo DNI.

context Usuario:

Self.allInstances → forAll(u1,u2:Usuario | u1<>u2 implies
u1.dni<>u2.dni)

Dos usuarios no pueden tener el mismo Nick.

context Usuario:

Self.allInstances → forAll(u1,u2:Usuario | u1<>u2 implies
u1.nick<>u2.nick)

modificarUsuario(int DNI, String nick, String passwordAntigua, String passwordNueva, String[] gustos)

Postcondiciones:

El password, el nombre de un usuario debe ser diferente de cadena vacía.

context Usuario:

Self.allInstances → forAll(u1: Usuario | u1.password<>)

Un usuario debe haber introducido al menos un gusto.

context Usuario:

Self.allInstances → forAll(u1: Usuario | u1.gustos.size()>=1)

Un usuario no podrá realizar ninguna acción si su estado es baneado

ARCHIVOMULTIMEDIA

subirArchivoMultimedia(int tamanyo, String tipo, Tfecha fecha, String url)

Pre:

Los archivos tienen un tamaño máximo

Context ArchivoMultimedia:

self.allInstances → forAll(am: ArchivoMultimedia | am.tamMax >= am.tamanyo)

Para subir un archivo multimedia, url no debe ser " ".

Context ArchivoMultimedia:

self.allInstances → forAll(am: ArchivoMultimedia | am.url <> "")

DENUNCIA

crearDenuncia(String motivo)

Pre:

El motivo de la denuncia no puede estar vacío

context Denuncia:

Denuncia.allInstances → forAll(d1: Denuncia | d1.motivo <> "")

MENSAJE

enviarMensaje(String textoMensaje, int emisor, int[] receptor, int tamanyo, boolean esPublico)

Pre:

Los mensajes siempre tienen al menos un receptor

Context Mensaje:

self.allInstances → forAll(m: Mensaje | m.Receptor → size() >= 1)

En un mensaje, el emisor y el receptor son distintos.

Context Mensaje:

self.allInstances → forAll(m: Mensaje | m.Emisor.DNI <> m.Receptor.DNI)

Los mensajes tienen un tamaño máximo(el tamaño del mensaje no supera el tamaño máximo)

Context Mensaje:

self.allInstances → forAll(m: Mensaje | m.tamMax >= m.tamanyo)

Post:

Todos los mensajes tienen un único identificador.(para dos mensajes, los identificadores son distintos)

Context Mensaje:

self.allInstances → forAll(m1,m2: Mensaje | m1.identificador <> m2.identificador)

MENSAJE PRIVADO

responder (String textoMensaje, int emisor, int[] receptor, int)

Post:

Todos los mensajes privados tienen el atributo esPublico = false

context MensajePrivado

MensajePrivado.allInstances ->forAll (m: MensajePrivado| m.esPublico = false)

MENSAJE FORO

Post:

Todos los mensajes privados tienen el atributo esPublico = false

context MensajePrivado

MensajePrivado.allInstances ->forAll (m: MensajePrivado| m.esPublico = false)

2.3. INVARIANTES OCL DE LAS CLASES

2.3.1. MENSAJE

Los mensajes sólo tienen un emisor

Context Mensaje:

```
self.allInstances → forAll(m: Mensaje | m.Emisor → size() = 1)
```

Los mensajes siempre tienen al menos un receptor

Context Mensaje:

```
self.allInstances → forAll(m: Mensaje | m.Receptor → size() >= 1)
```

En un mensaje, el emisor y el receptor son distintos.

Context Mensaje:

```
self.allInstances → forAll(m: Mensaje | m.Emisor.DNI <> m.Receptor.DNI)
```

Los mensajes tienen un tamaño máximo(el tamaño del mensaje no supera el tamaño máximo)

Context Mensaje:

```
self.allInstances → forAll(m: Mensaje | m.tamMax >= m.tamanyo)
```

Se pueden mandar mensajes vacíos

Context Mensaje:

```
self.allInstances → forAll(m: Mensaje | m.texto = "" or m.texto <> "")
```

Todos los mensajes tienen un único identificador.(para dos mensajes, los identificadores son distintos)

Context Mensaje:

```
self.allInstances → forAll(m1,m2: Mensaje |  
m1.identificador <> m2.identificador)
```

Todos los mensajes privados tienen el atributo esPublico = false

context MensajePrivado

```
MensajePrivado.allInstances ->forAll (m: MensajePrivado|  
m.esPublico = false)
```

Todos los mensajes del foro han de ser publicos

context MensajeForo

```
MensajeForo.allInstances ->forAll (m: MensajeForo| m.esPublico = true)
```

2.3.2. ARCHIVOS MULTIMEDIA

Los archivos multimedia sólo tienen un emisor

Context ArchivosMultimedia:

```
self.allInstances → forAll(am: ArchivoMultimedia | am.Emisor → size() = 1)
```

Los archivos multimedia tienen un tamaño máximo

Context ArchivoMultimedia:

```
self.allInstances → forAll(am: ArchivoMultimedia |  
am.tamMax >= am.tamanyo)
```

Para subir un archivo multimedia, url no debe ser " ".

Context ArchivoMultimedia:

```
self.allInstances → forAll(am: ArchivoMultimedia | am.url <> "")
```

Todos los archivos multimedia tienen un único identificador.

Context ArchivosMultimedia:

```
self.allInstances → forAll(am1,am2: ArchivoMultimedia |  
am1.identificador <> am2.identificador)
```

2.3.3. LOCAL

Todos los locales tienen un único identificador.

Context Local:

```
self.allInstances → forAll(lo1,lo2: Local |  
lo1.identificador <> lo2.identificador)
```

La señal del local tiene que ser menor o igual que la tarifa.

Context Local:

```
Local.AllInstances → forAll(l1: Local | l1.senyal <= l1.tarifa)
```

No puede haber dos locales con la misma dirección y localidad

Context Local:

```
Local.AllInstances → forAll (l1, l2: Local |  
(l1 <> l2) implies  
((l1.Direccion <> l2.Direccion) OR (l1.Localidad <> l2.Localidad)))
```

El local debe tener aforo mayor que cero

Context Local:

```
Local.AllInstances → forAll( l1: Local | l1.aforo > 0)
```

No pueden tener dos locales el mismo identificador

Context Local:

```
Local.AllInstances → forAll(l1,l2: Local |  
(l1 <> l2) implies (l1.identificador <> l2.identificador))
```

La fecha de un evento en la que actúa una local debe estar en el calendario del local

Context Local

```
self.evento→forAll( x: Evento | self.calendario.disponible(x.fecha) )
```

2.3.4. USUARIO

Un emisor no puede subir archivos multimedia con el mismo url

Context Usuario:

Userio.ArchMult → forAll (AM1,AM2 : ArchivoMultimediao |
AM1.url <> AM2.url)

Un usuario no puede estar apuntado a dos eventos en la misma fecha

context Usuario:

Self.Evento → forAll(e1,e2:Evento| e1<>e2 implies e1.fecha<>e2.fecha)

Dos usuarios no pueden tener el mismo DNI.

context Usuario:

Self.allInstances → forAll(u1,u2:Usuario | u1<>u2 implies u1.dni<>u2.dni)

Dos usuarios no pueden tener el mismo nick.

context Usuario:

Self.allInstances → forAll(u1,u2:Usuario | u1<>u2 implies
u1.nick<>u2.nick)

El password, el nombre, el apellido, el DNI y el nick de un usuario deben ser diferentes de cadena vacía.

context Usuario:

Self.allInstances → forAll(u1: Usuario |
u1.nombre<> "" and u1.apellido<> "" and u1.nick<> "" and
u1.password<> "" and u1.dni<> "")

Un usuario debe haber introducido al menos un gusto.

context Usuario:

Self.allInstances → forAll(u1:Usuario | u1.gustos→size()>=1)

2.3.5. EVENTO

La señal tiene que ser menor o igual que el precioFinal

context Evento:

Evento.allInstances → forAll(e1: Evento | e1.senyal <= e1.precioFinal)

El aforo de un evento debe estar entre aforo min y aforo max cuando el estado del evento es pagado

context Evento:

Evento.allInstances → forAll(e1: Evento |
(e1.aforoMin <= e1.listaUsuario → size() + e1.Invitado → size())
AND
(e1.listaUsuario → size() + e1.Invitado → size())<= e1.aforoMax)
AND e1.estado=pagado)

El aforo mínimo siempre es menor o igual que el aforo máximo

context Evento:

Evento.allInstances → forAll(e1: Evento |
(0 <= e1.aforoMin) and (e1.aforoMin <= e1.aforoMax))

Cuando el evento está como pagado, el precioFinal del evento debe estar pagado (es decir, todos los recibos pagan el evento)

```
context Evento:  
    Evento.allInstances → forAll(e1: Evento |  
        e1.precioFinal <= e1.Recibo.cantidad→sum()  
        AND e1.estado=pagado)
```

Cuando el evento está creado, los recibos deben haber pagado la señal.

```
context Evento:  
    Evento.allInstances → forAll(e1: Evento |  
        e1.senyal = e1.Recibo.cantidad→sum() AND e1.estado=creado)
```

Los usuarios de la lista de usuarios no pueden estar en la lista de espera

```
context Evento:  
    Eventos.listaUsuario → forAll(u1: Usuario |  
        Eventos.listaEspera → forAll(u2: Usuario| u1.DNI <> u2.DNI))
```

El tope para Apuntarse a eventos es durante la fecha de modificación, es decir, fechaLimite días antes de la realización del evento.

```
context Evento:  
    Evento.allInstances → forAll(e1: Evento |  
        e1.fecha - fechaLimite >= sysdate())
```

La dirección del evento y la dirección del local deben ser la misma

```
Context Evento:  
    Self.allInstances → (e1: Evento | e1.Local.direccion = e1.direccion)
```

Todos los locales de un evento están en la misma localidad, la del evento:

```
context Evento  
    self.allInstances→forAll(e:Evento  
        e.local→forAll(loc:Local|self.Localidad=loc.Localidad))
```

El tipo de artista y el tipo de evento iguales.

```
context Evento:  
    self.allInstances → forAll(e1: Evento |  
        e1.Artista->forAll(a1: Artista | e1.TipoEvento = a1.estilo))
```

Si es privado no tiene ni lista de usuarios ni lista de espera

```
Contex Evento inv  
    Self.allInstances → forAll(e:Evento |  
        e.esPublico == false  
        implies ((e.listaUsuario→size() + e.listaEspera-> size())==0)
```

Si el evento esta cancelado no puede tener asociado lista de usuarios, lista de espera, artistas, invitados, locales

```
Contex Evento inv  
    Self.allInstances → forAll (e: Evento |  
        e.estado== cancelado
```

```

implies (( e.listaUsr → size()+
e.listaEspera→ size() +
e.local→ size() +
e.artista →size() +
e.invitado→size() +
e.subcontrata→ size())==0

```

La dirección de un evento debe coincidir con la dirección de alguno de sus locales

Context Evento:

```
self.locales→exists( l: Local | l.direccion = self.direccion)
```

2.3.6. DENUNCIA

El motivo de la denuncia no puede estar vacío

context Denuncia:

```
Denuncia.allInstances → forAll(d1: Denuncia | d1.motivo <> "")
```

2.3.7. RECIBO

En la clase recibo Las cuentas emisor y receptor del recibo han de ser distintas

context Recibo

```
recibo. allInstances ->forAll (R: recibo|
R.cuentaEmisor<>R.cuentaReceptor)
```

En la clase recibo la cantidad ha de ser positiva

context Recibo

```
recibo. allInstances ->forAll (R: recibo| R.cantidad >0 )
```

2.3.8. ARTISTA

La evaluación(puntuacion/votaciones) del artista está entre 0 y 10

Context Artista

```
Artista.allInstances→( a: Artista |
a.puntuacion/a.votaciones >= 0 and
a.puntuacion/a.votaciones<=10)
```

Cada artista tiene únicamente un evento a la vez

Context Artista:

```

self.evento→forAll( x: Evento | self.calendario.disponible(x.fecha) )
self.evento→forAll( x,y : Evento |
x<>y implies
(((x.fecha < y.fecha ) and (x.fecha+x.duracion < y.fecha)) or
((y.fecha < x.fecha ) and (y.fecha+y.duracion < x.fecha)))

```

Cada dni es único para cada artista

Context Artista:

```
Artista.allInstances→forAll(x,y : Artista | x<>y implies x.dni <> y.dni)
```

La puntuación de un artista está entre 0 y 10

Context Artista:

```
Artista.allInstances→forAll( a: Artista |  
    a.puntuacion/a.votos >= 0 and a.puntuacion/a.votos<=10)
```

2.3.9. SUBCONTRATA

Cada subcontrata tiene únicamente un evento a la vez:

Context Subcontrata

```
self.evento→forAll( x,y : Evento |  
    x<>y implies (((x.fecha < y.fecha ) and  
        (x.fecha+x.duracion < y.fecha)) or  
        ((y.fecha < x.fecha ) and  
        (y.fecha+y.duracion < x.fecha)))
```

No se contratarán subcontratas para un evento, si el servicio de la subcontrata está disponible en el local.

La fecha de un evento en la que actúa una subcontrata debe estar en el calendario de la subcontrata

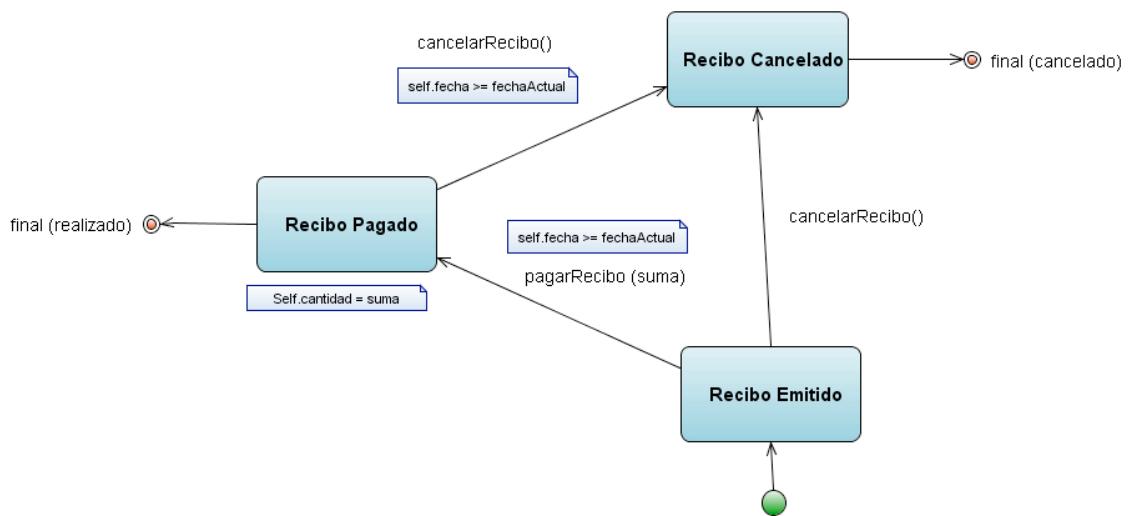
Context Subcontrata:

```
self.evento→forAll( x: Evento | self.calendario.disponible(x.fecha) )
```

2.4. DIAGRAMAS DE ESTADOS

2.4.1. DIAGRAMA DE ESTADOS DE RECIBO

Recibo Emitido es nuestro estado inicial, pasamos a un estado final **Recibo Pagado** si se ha pagado la cantidad del mismo en las fechas correspondientes. En los casos anteriores se puede dar un paso unilateral de cancelación, pasando a un estado final de **Recibo Cancelado**, siempre y cuando se cancele en fecha si está pagado.



2.4.2. DIAGRAMA DE ESTADOS DE EVENTO

Estos son los diferentes estados en los que puede estar un evento.

En el diagrama vemos explicadas las especificaciones de cada caso y los métodos que provocan los cambios de estado.

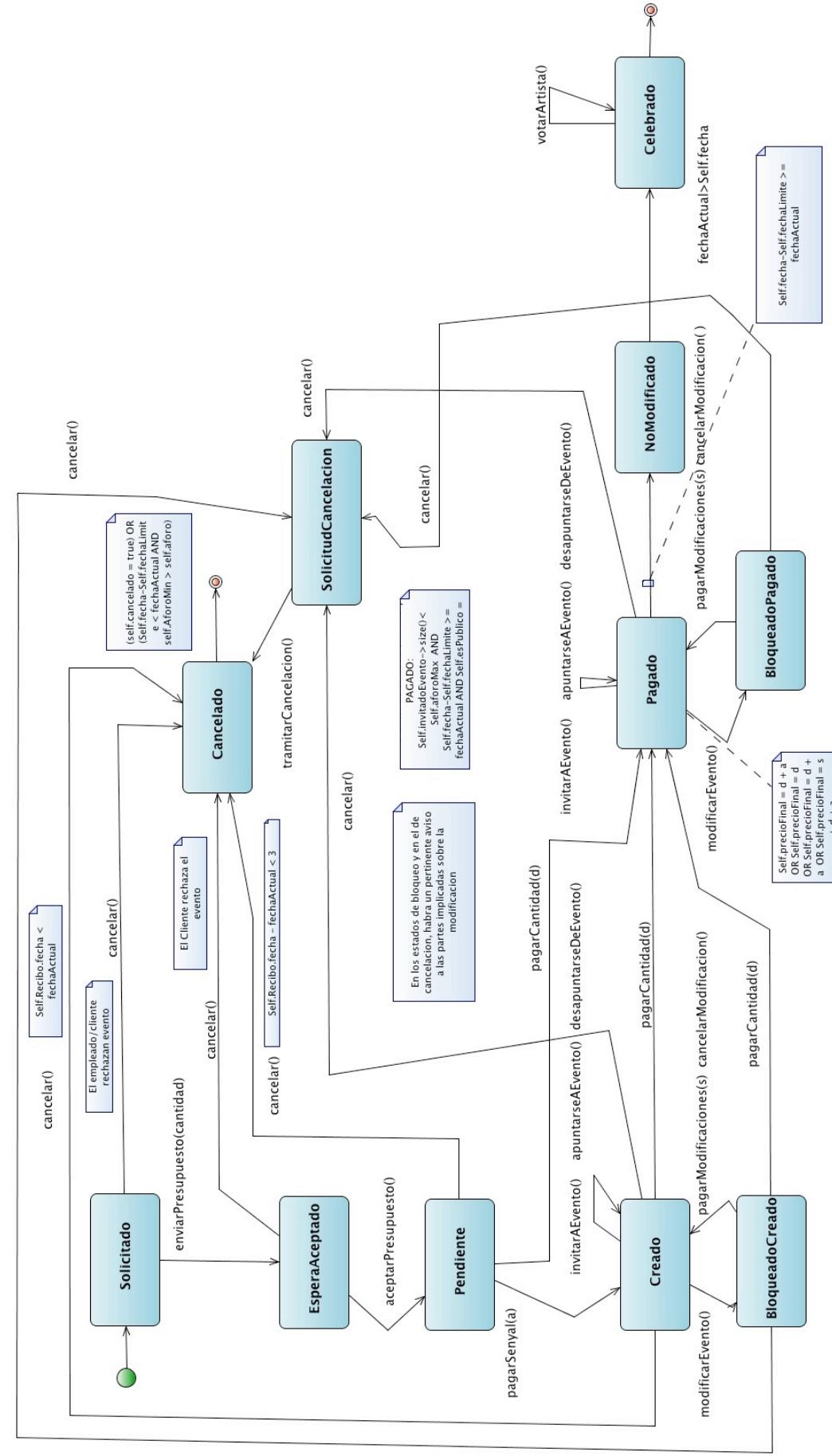
En general, tenemos unos primeros estados en los que se solicita el evento, se gestionan los presupuestos, y se espera a la aceptación.

Posteriormente, están los diferentes estados de pagos y modificaciones.

Finalmente, pasamos a un estado de no modificación, y de ahí se celebra el evento pudiéndose votar el mismo posteriormente.

En cada uno de los estados anteriores, salvo en el de no modificado y celebrado, podemos tramitar la cancelación del evento.

Ésta será solicitada, y habrá que esperar la aceptación de la misma, o bien será directa si se cumplen ciertos requisitos que vemos en los comentarios que proceden.



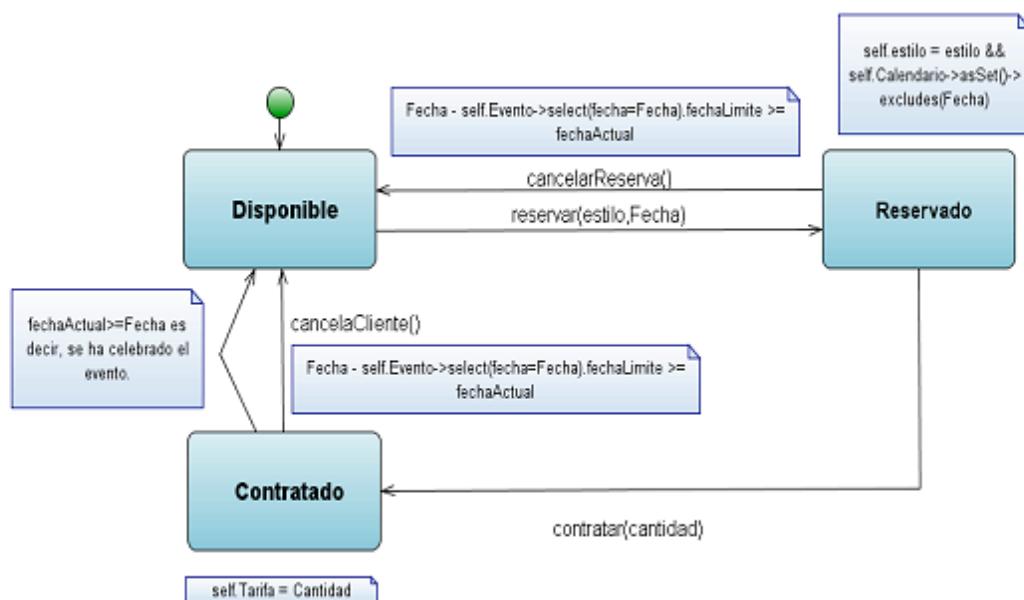
2.4.3. DIAGRAMA DE ESTADOS DE ARTISTA

El artista comienza estando **Disponible**. Recibe una reserva con una fecha y un estilo.

Si está disponible para esa fecha y es de su estilo, queda **reservado**.

Puede ser ahora cancelada la reserva siempre y cuando esté dentro de su plazo de cancelación en cuyo caso vuelve a estar **disponible**, o bien ser **contratado**, para lo que habrá tenido que recibir el pago de su tarifa.

Una vez se celebre el evento o se cancele unilateralmente por el cliente dentro de su plazo, volverá a estar **disponible**.



2.5. DIAGRAMAS DE SECUENCIAS

2.5.1. CREACIÓN DE RECURSOS PREVIOS

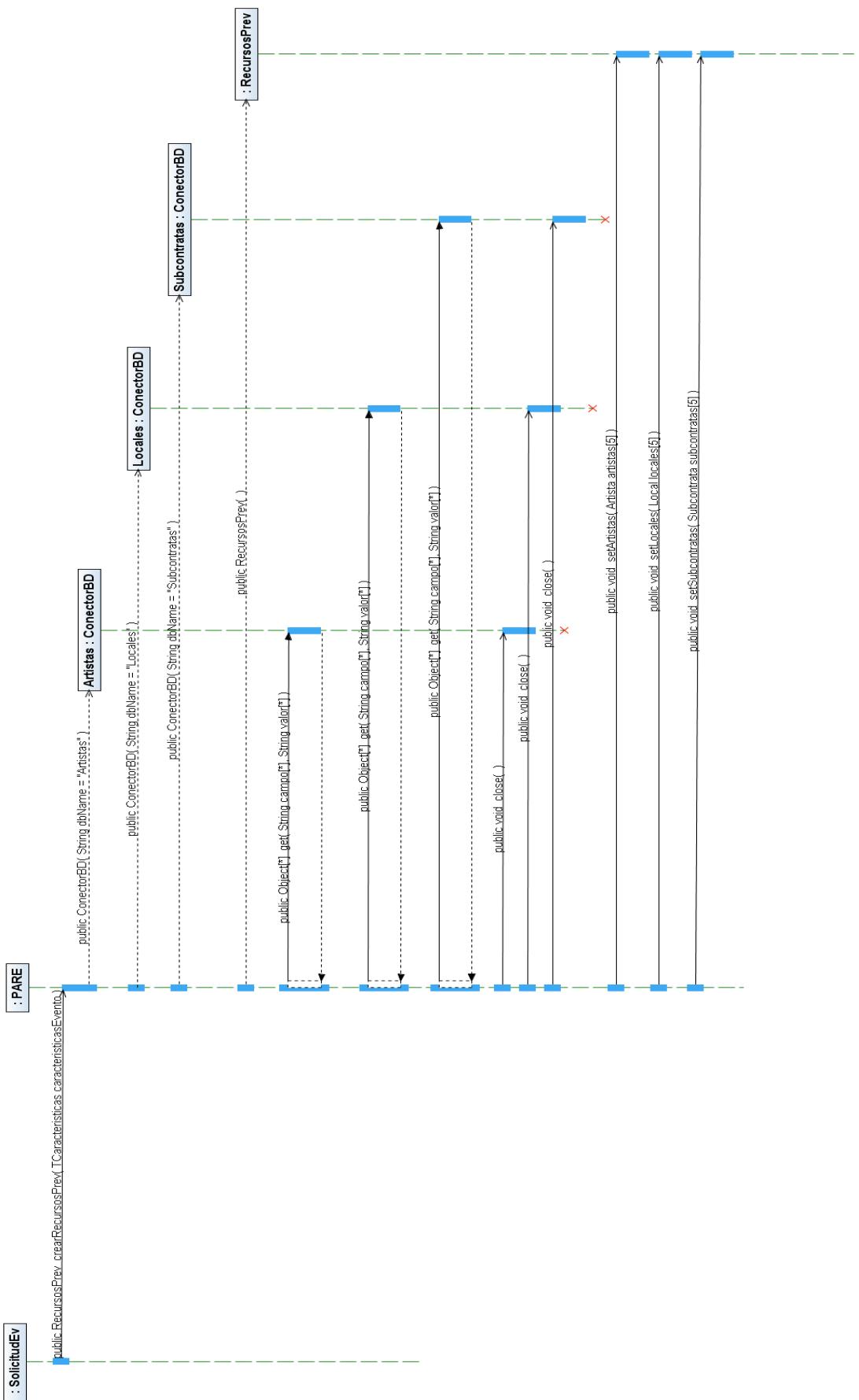
Esta operación la hará el sistema después de que el cliente haya solicitado el evento.

Precondición:

Tanto cliente como la solicitud del evento deben estar registrados/guardados en el sistema. También debe estar almacenado en la base de datos las tablas de Artistas, Locales y Subcontratas.

Postcondición:

Se obtiene un objeto de la clase **RecursosPrev** con los 5 mejores artistas, subcontratas y locales para el evento solicitado.

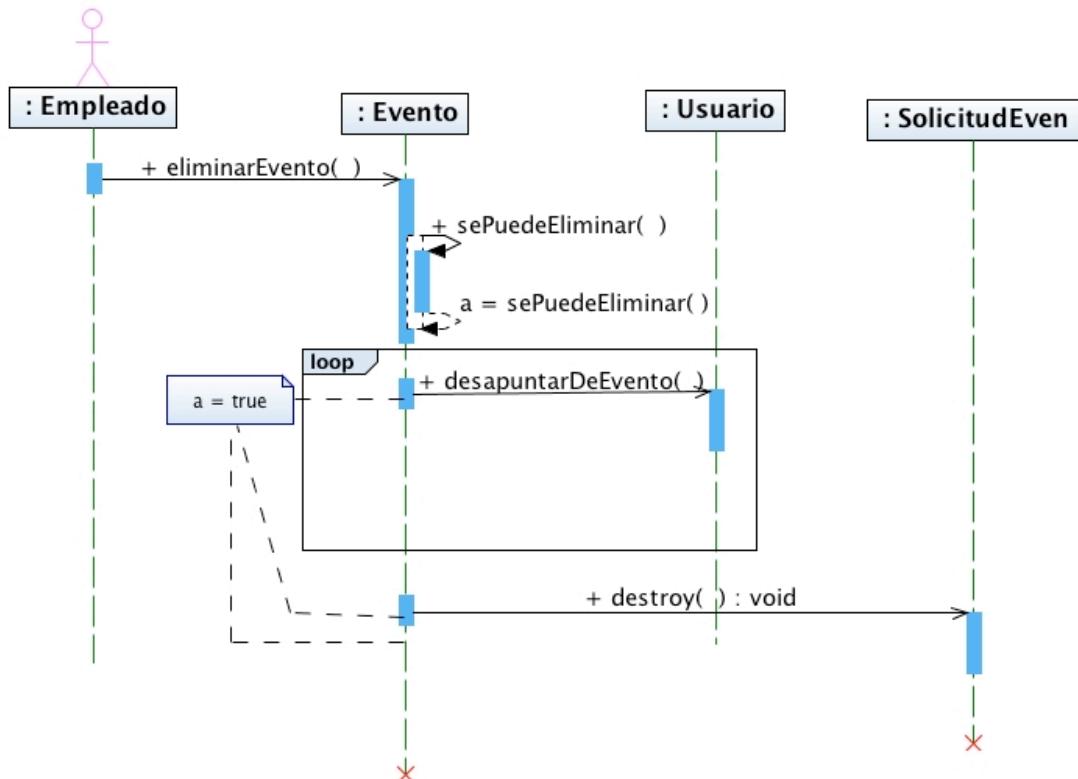


2.5.2. ELIMINAR EVENTO

El empleado elimina un evento. Si el evento se puede eliminar se desapunta del evento a todos los usuarios apuntados al mismo. Al desapuntar del evento se le informa al usuario que ha sido desapuntado.

Precondición

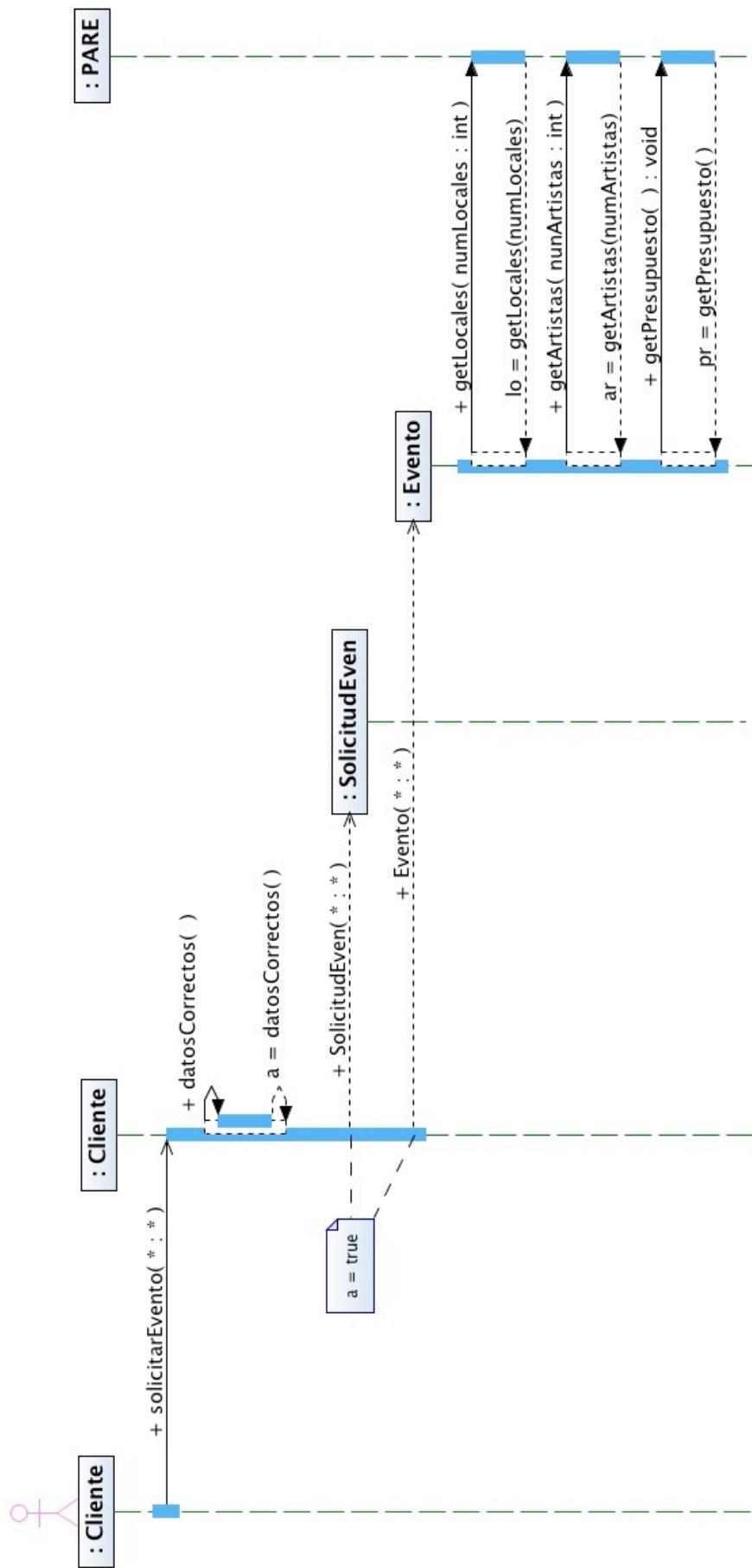
El cliente debe haber enviado una solicitud de anulación de dicho evento.



2.5.3. SOLICITAR EVENTO

* : * ➔ fecha : TFecha, duracion : float, localidad : int, esPublic : boolean, aforoMin : int, aforoMax : int, tipoEvento : String, fechaLimite : int, numArtistas : int, numLocales : int

El cliente solicita un evento, para ello debe llenar los datos correspondientes. Se guarda una solicitud del evento como resguardo del mismo y se crea el evento con datos provisionales ofrecidos por el P.A.R.E.



2.5.3. BANEAR USUARIO

El empleado expulsa de la red social a un usuario.

Para ello, el usuario no debe tener eventos como cliente, de no ser así, se intenta eliminar los eventos (eliminar un evento sólo puede hacerse si el empleado es el jefe) y se avisa a los usuarios apuntados al mismo.

El usuario tampoco debe estar apuntado a eventos, de no ser así, se intenta eliminar de los eventos (si el evento aún está en periodo de modificación podrá ser eliminado) y se actualiza la lista de espera.

Siempre se eliminarán todos los mensajes y archivos multimedia de este usuario.

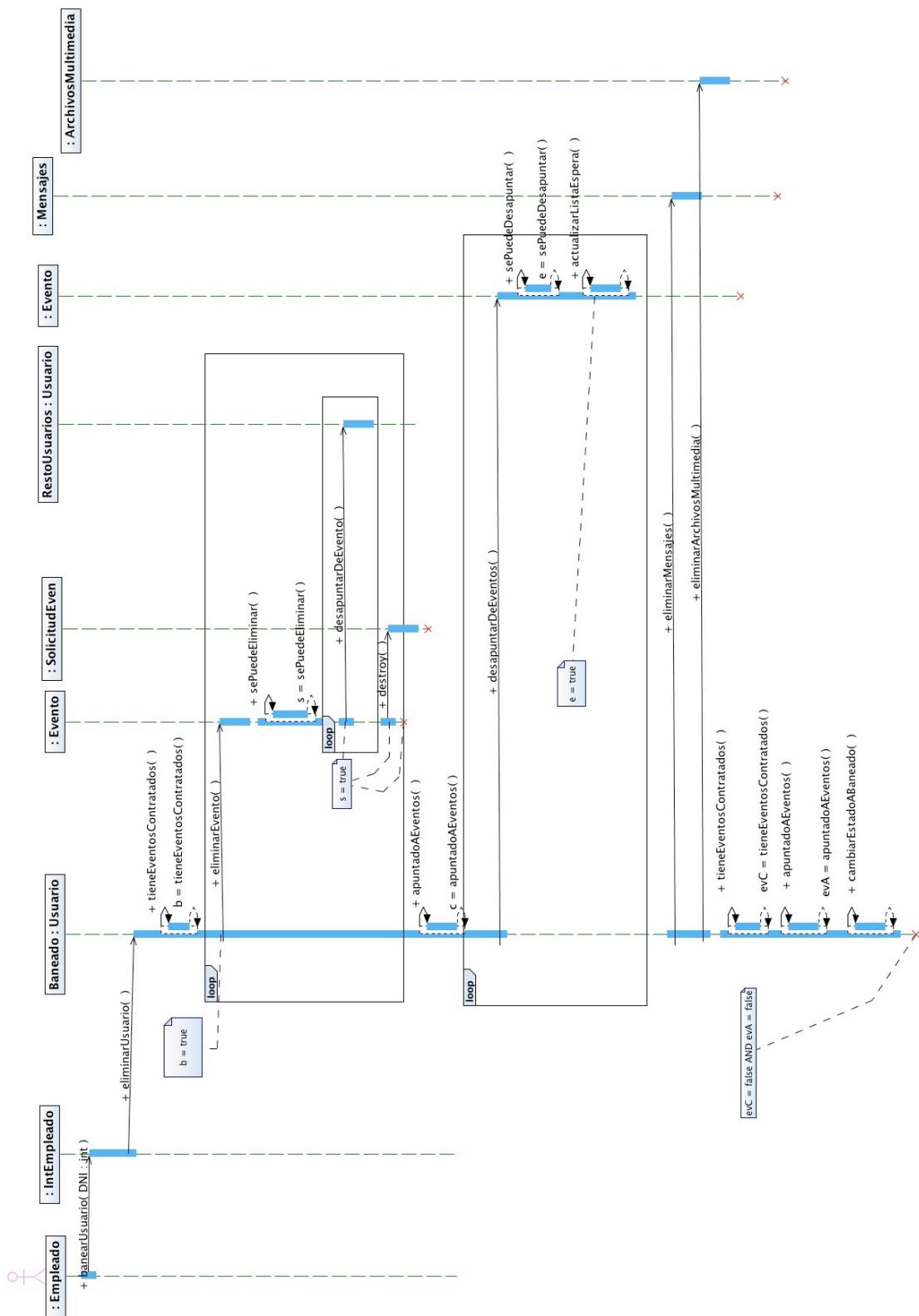
Finalmente, el usuario será eliminado si no tiene eventos ni como usuario ni como cliente.

Precondición:

El usuario a banear debe haber recibido al menos una denuncia.

Postcondición:

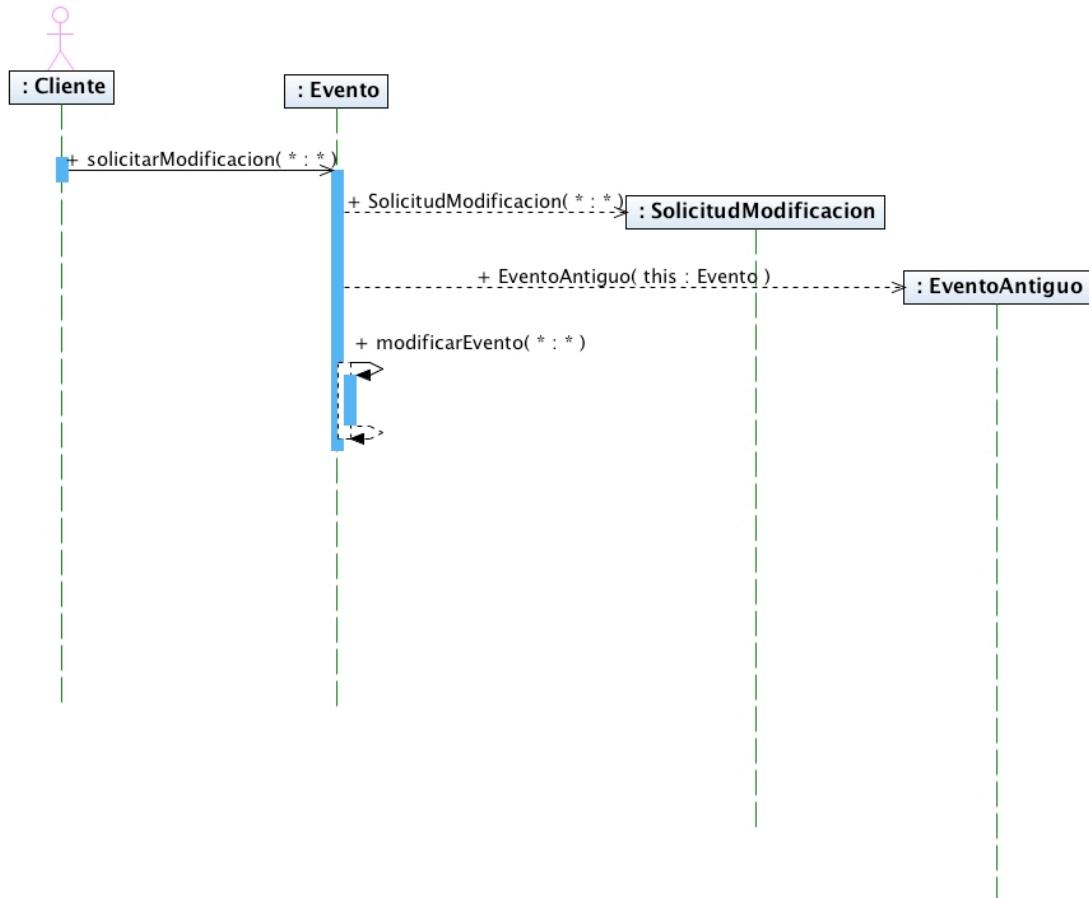
El usuario queda eliminado si todos sus eventos han podido ser eliminados y se le ha desapuntado de todos los eventos. En caso contrario, el estado del usuario cambiará a baneado.



2.5.5. SOLICITAR MODIFICACIÓN

* : * ➔ fecha : TFecha, duracion : float, localidad : int, esPublic : boolean, aforoMin : int, aforoMax : int, tipoEvento : String, fechaLimite : int, numArtistas : int, numLocales : int

El cliente selecciona un evento y modifica alguno de sus atributos. Generamos entonces una solicitud de modificación como recibo de la misma. Guardamos el estado actual del evento, para asegurarnos poder volver atrás si el cliente se retracta y modificamos el evento con los nuevos datos introducidos.



2.5.6. ACEPTAR MODIFICACION

El cliente tendrá que aceptar el nuevo presupuesto dado por el empleado. El nuevo recibo ya no es el de solicitud del evento sino el de solicitud de modificación, así que, actualizamos el recibo del evento.

Borramos el evento antiguo y la solicitud de modificación.

La modificación del evento puede conllevar cambios en el aforo, por ejemplo, luego tenemos que actualizar la lista de usuarios que van al evento y la lista de espera.

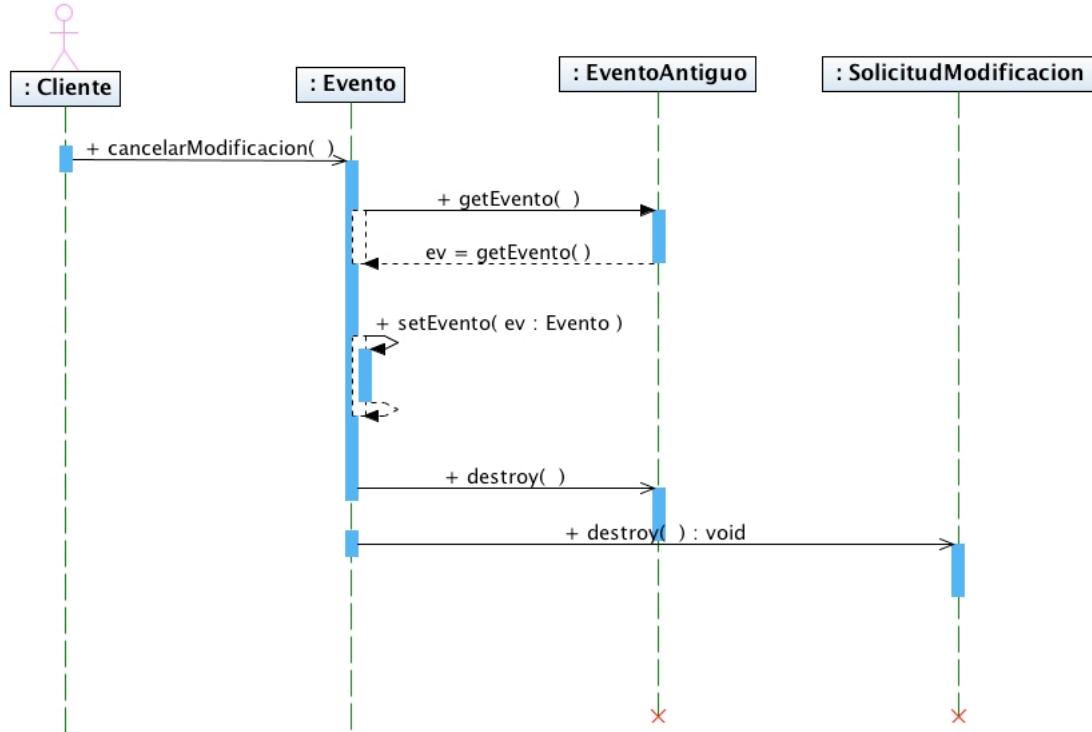
Avisamos de los cambios a los usuarios que se han apuntado al evento.



2.5.7. CANCELAR MODIFICACIÓN

El usuario no acepta el nuevo presupuesto y rechaza la modificación. Tenemos que restaurar el estado del evento. Para ello cogemos los datos necesarios del evento antiguo y actualizamos el evento actual.

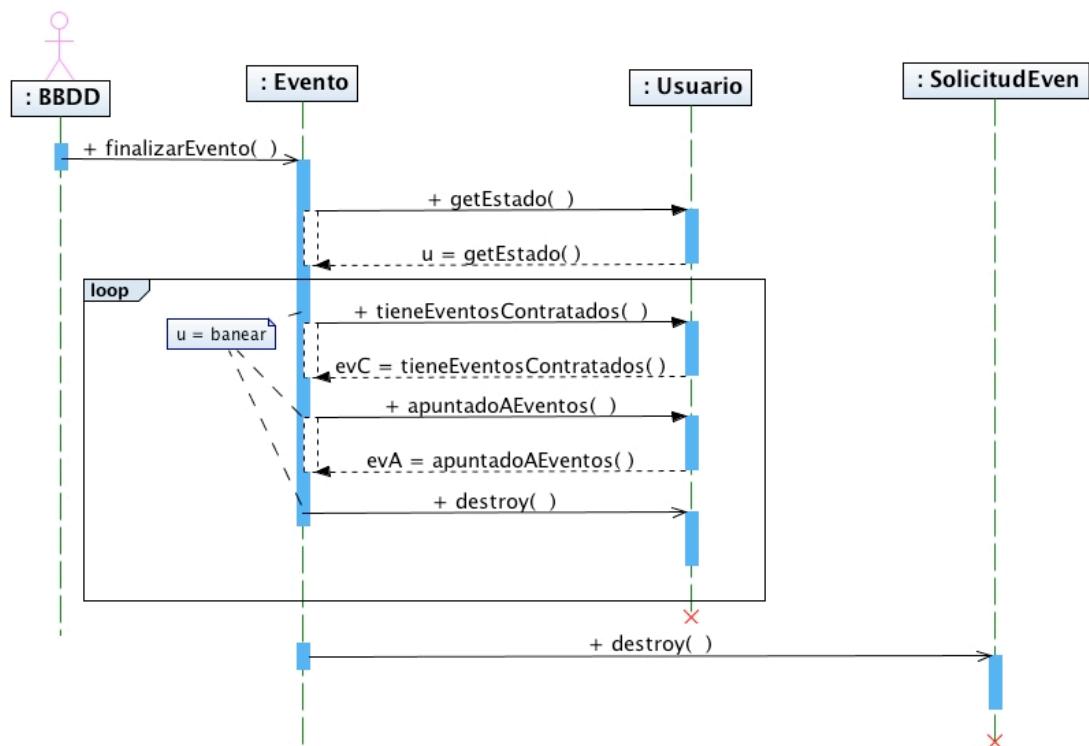
Borramos el evento antiguo y la solicitud de modificación.



2.5.8. FINALIZAR EVENTO

Cuando un evento ha sido celebrado (la fecha de realización del evento ha pasado) el controlador de la base de datos disparará la función finalizarEvento. Comprobamos el estado de los usuarios asociados al evento y si anteriormente se les ha intentado banear, entonces serán eliminados, si no tienen eventos pendientes.

Borramos la solicitud del evento, ya que este ha sido realizado, ya no lo necesitaremos más.

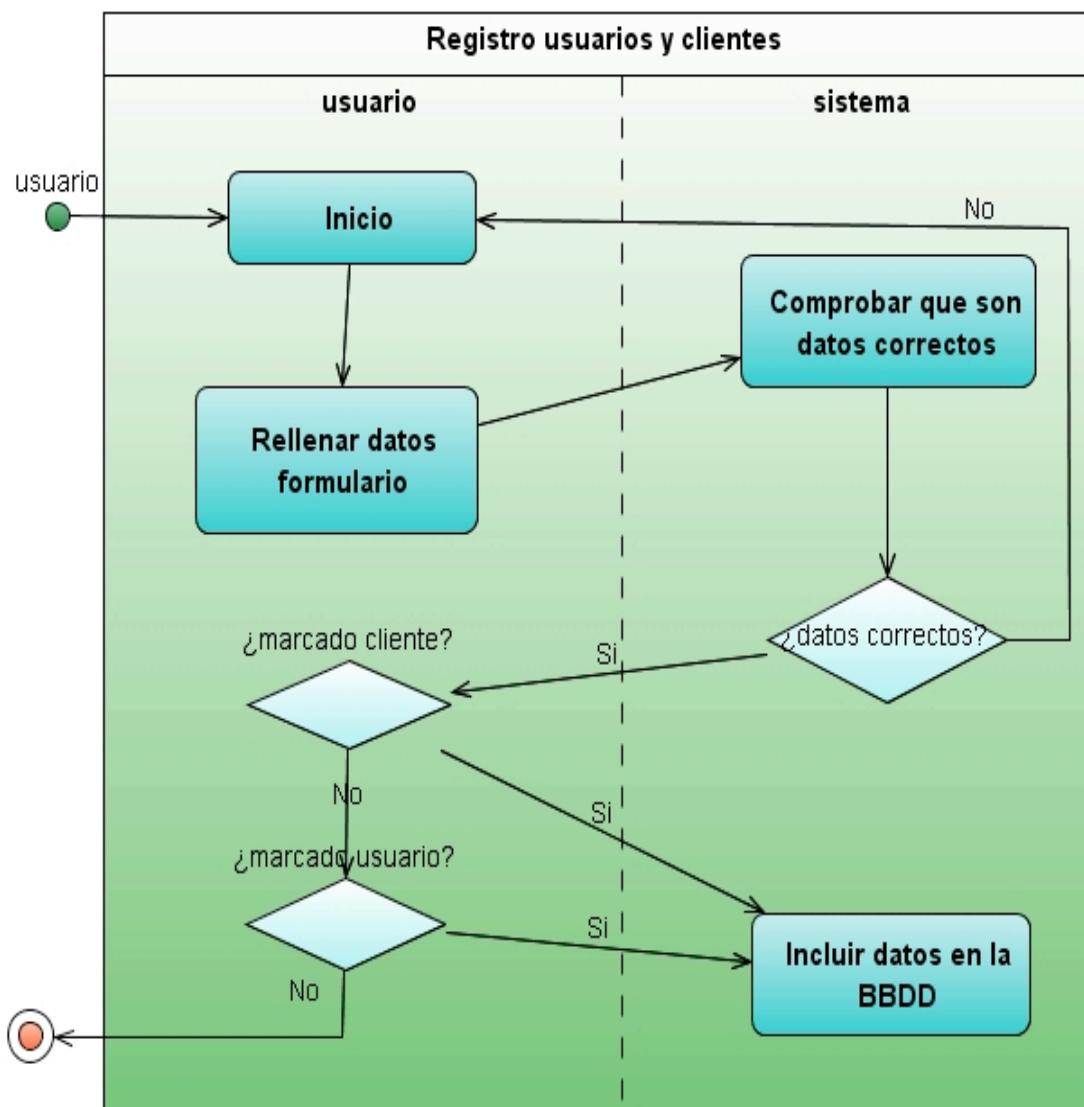


2.6. DIAGRAMAS DE ACTIVIDADES

2.6.1. ACTIVIDAD COMÚN USUARIO / CLIENTE

2.6.1.1. REGISTRO USUARIO/CLIENTE

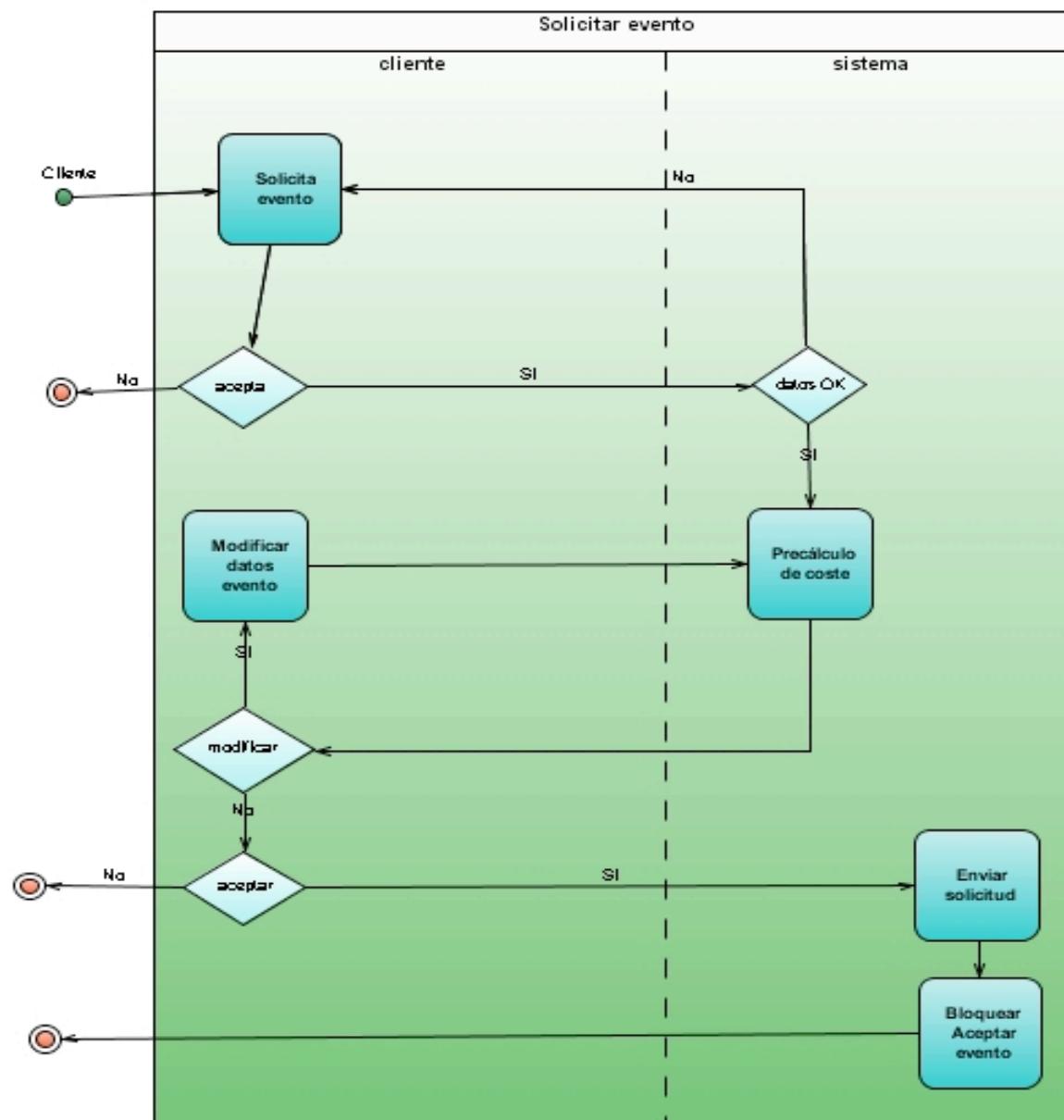
Para que un usuario pueda registrarse, deberá llenar un formulario con sus datos y marcando la casilla que desee según quiera ser cliente exclusivamente o usuario de la red social (implica ser cliente), en ningún caso permitiremos que una persona pueda marcar las dos opciones. El sistema comprobará que estos datos son correctos. En tal caso, dependiendo de si ha seleccionado ser sólo cliente, o usuario de la red social, se incluirán los datos correspondientes en la BBDD.



2.6.2. ACTIVIDADEDES CLIENTE

2.6.2.1. SOLICITAR EVENTO

El cliente solicita un evento rellenando todos los datos. Si estos son correctos, se hace un cálculo del coste del evento. Se da opción al cliente para que pueda modificar datos del evento, calculando de nuevo el precio. Si al final el cliente acepta esa cantidad, se enviará la solicitud.



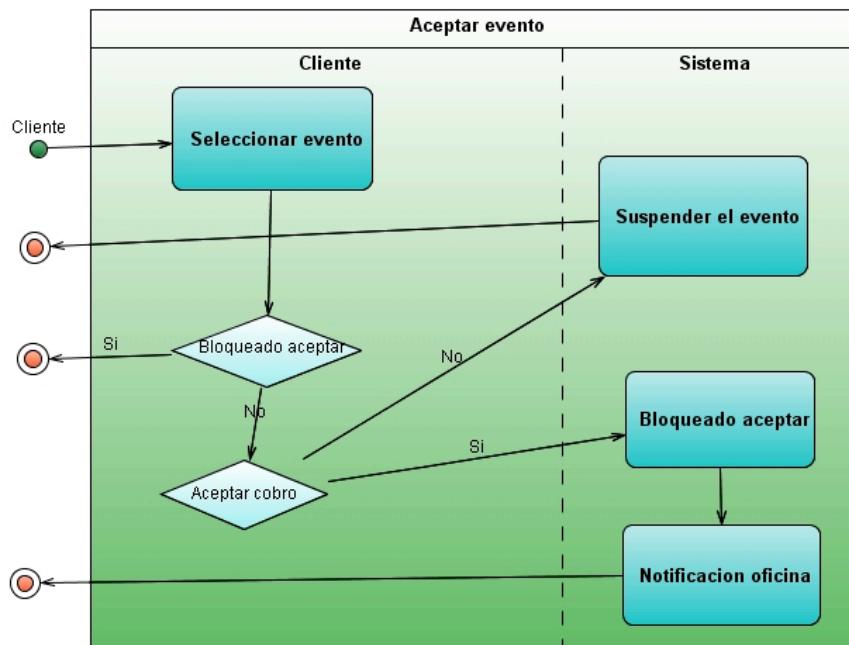
2.6.2.2. ACEPTAR EVENTO

El cliente selecciona un evento, pudiéndose suspender desde el sistema.

Si la aceptación estaba bloqueada de antes, no se le permite al cliente continuar y salimos.

Ahora preguntamos si acepta el cobro. Si no es así, el sistema suspenderá el evento.

Si lo acepta, pasamos a bloquear futuras aceptaciones para evitar desajustes y notificamos el caso a la oficina para que actúe en consecuencia.



2.6.2.3. ACEPTAR MODIFICACIÓN

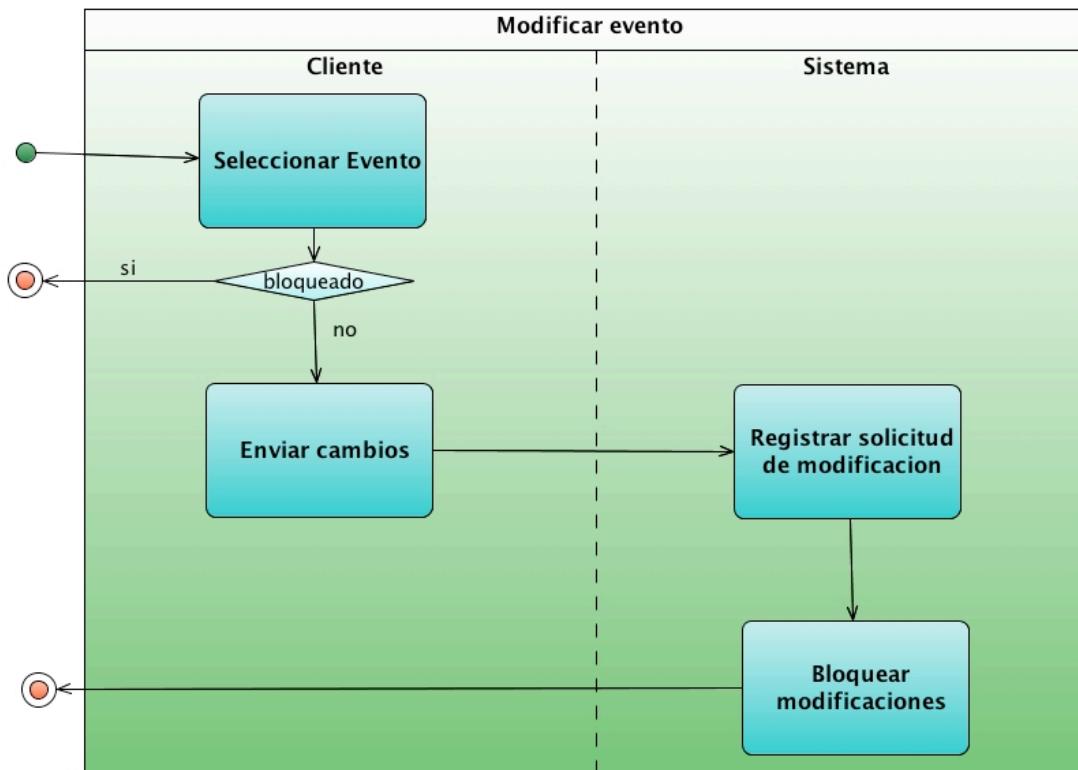
Igual que aceptar evento, pero en vez de suspender el evento al no aceptar el cobro, copiamos el evento antiguo (estado del evento antes de la modificación) en el evento actual. Borramos la solicitud de modificación y el evento antiguo. En este mismo estado, desbloqueamos el evento para posibles modificaciones.

2.6.2.4. MODIFICAR EVENTO

El cliente selecciona un evento. Si estaba bloqueado antes, no se le permite al cliente continuar y salimos.

Si no, solicita los cambios pertinentes, y envía la petición.

El sistema registrará la petición, creando un recibo de modificación y guardando el evento actual por si acaso el usuario no acepta el nuevo presupuesto.

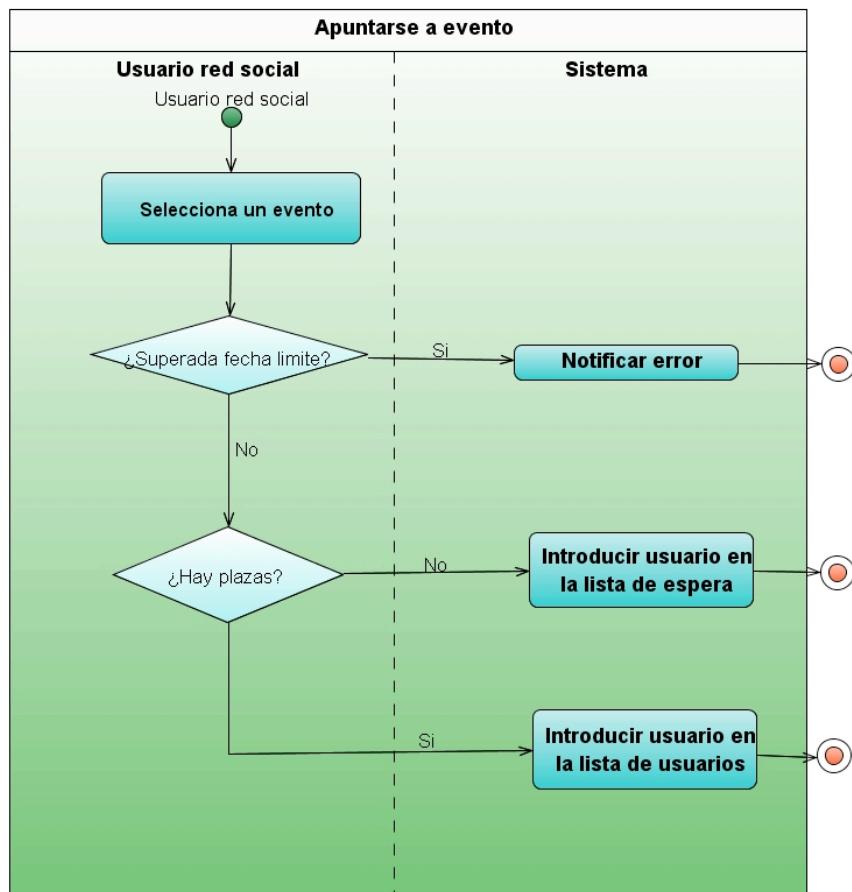


2.6.3. ACTIVIDADES DE USUARIO

2.6.3.1 APUNTARSE A EVENTO

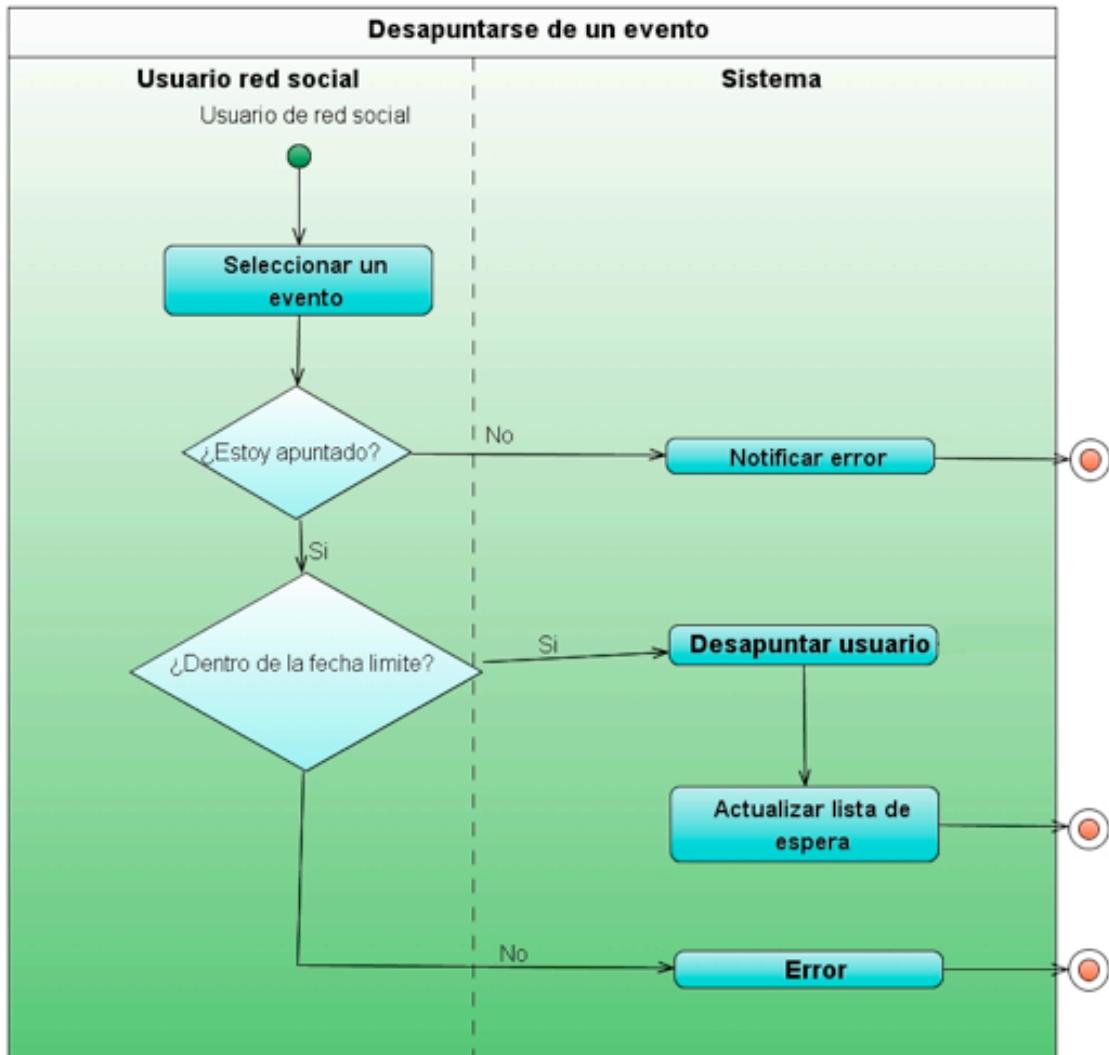
El Usuario selecciona un evento, y se comprueba si se ha superado la fecha límite para apuntarse a ese evento.

Si es que no, se pregunta si quedan plazas disponibles y según sea sí o no, se apunta en la lista de usuarios o en la lista de espera respectivamente.



2.6.3.2. DESAPUNTARSE DE EVENTO

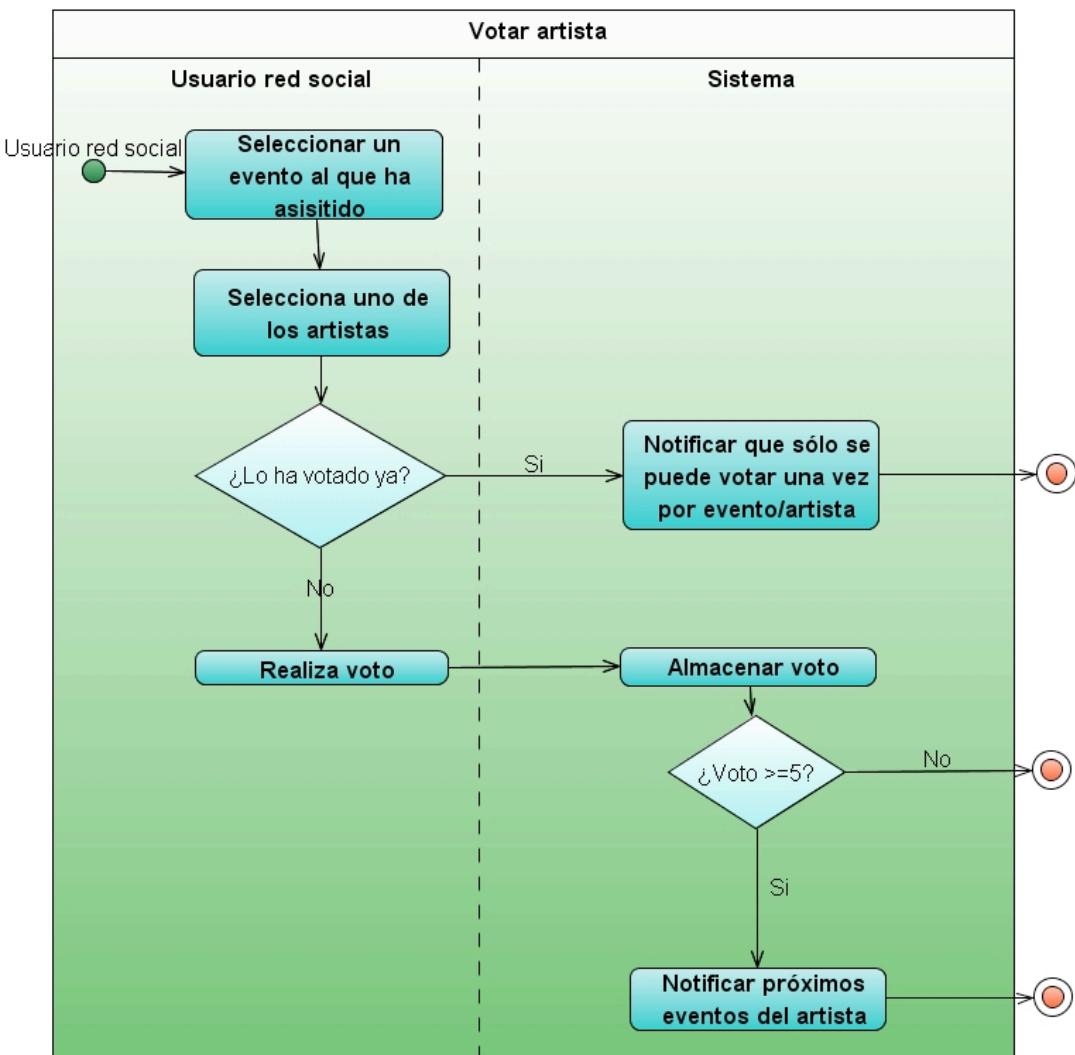
Para que un usuario pueda desapuntarse de un evento, tras seleccionarlo, deberá comprobarse que estaba previamente apuntado al mismo y actualizar la lista de espera, es decir, pasando un usuario de la lista de espera a las lista de usuarios.



2.6.3.3. VOTAR ARTISTA

Para que un usuario pueda votar a un artista, selecciona un evento al que haya asistido y a uno de los artistas.

Se comprueba que es la primera vez que lo realiza para evitar votos fraudulentos, realiza la votación, se almacena el mismo, y por último si esta actuación le ha gustado, lo tenemos en cuenta para avisarle de futuras actuaciones.

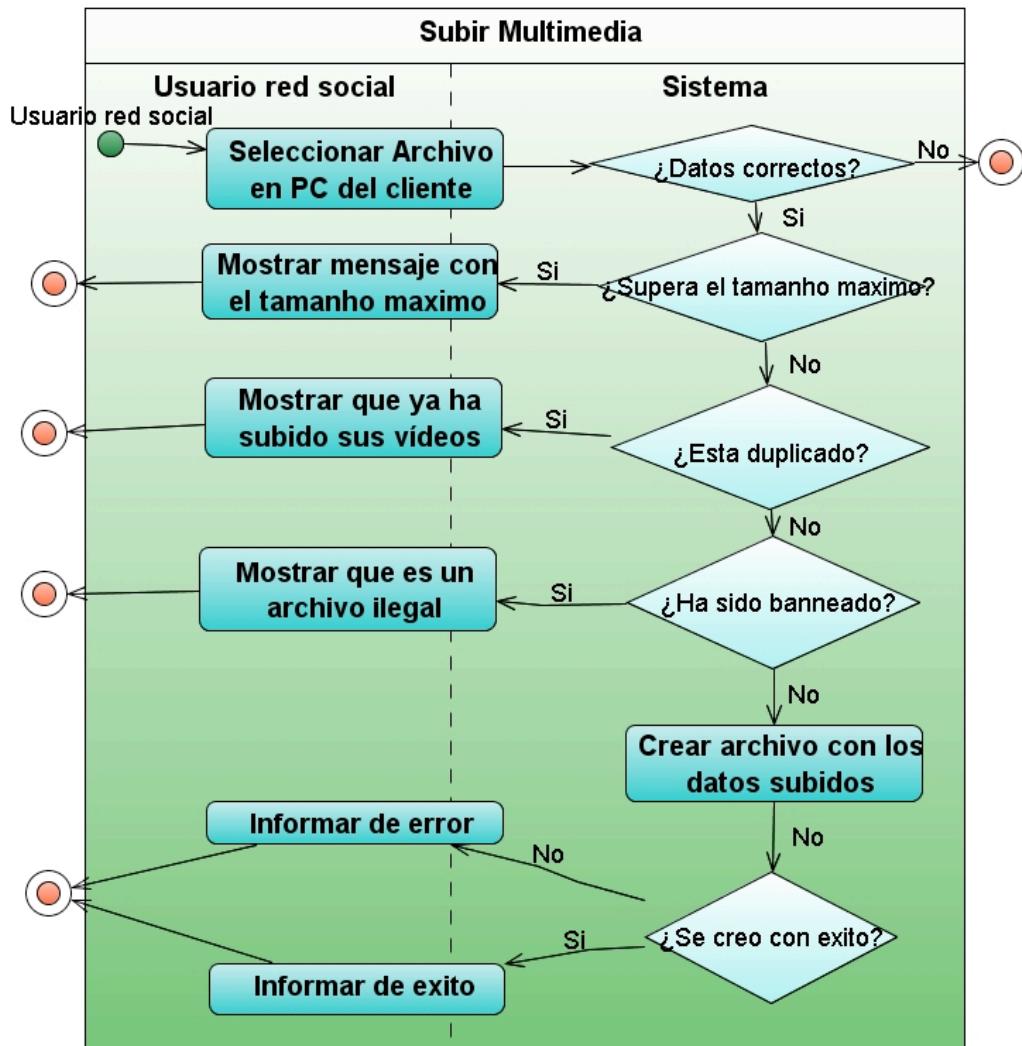


2.6.3.4. SUBIR ARCHIVO MULTIMEDIA

Para que un usuario pueda subir un archivo Multimedia, se realizan las diferentes comprobaciones que observamos en el diagrama para garantizar el correcto funcionamiento de la Red.

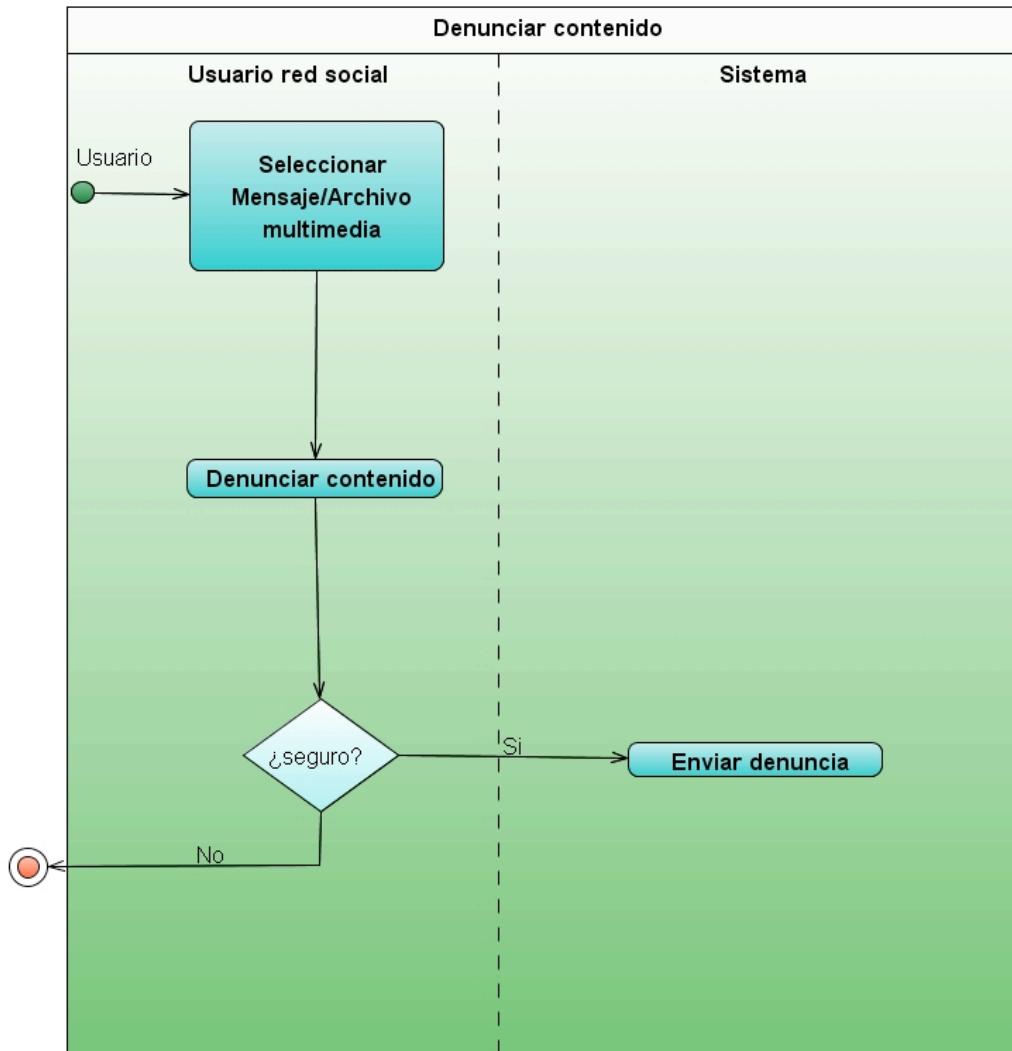
Así mismo, comprobamos que el Usuario no haya sido baneado previamente.

Por último, el sistema creará el archivo y notificará del éxito o del fallo del proceso.



2.6.3.5. DENUNCIAR CONTENIDO

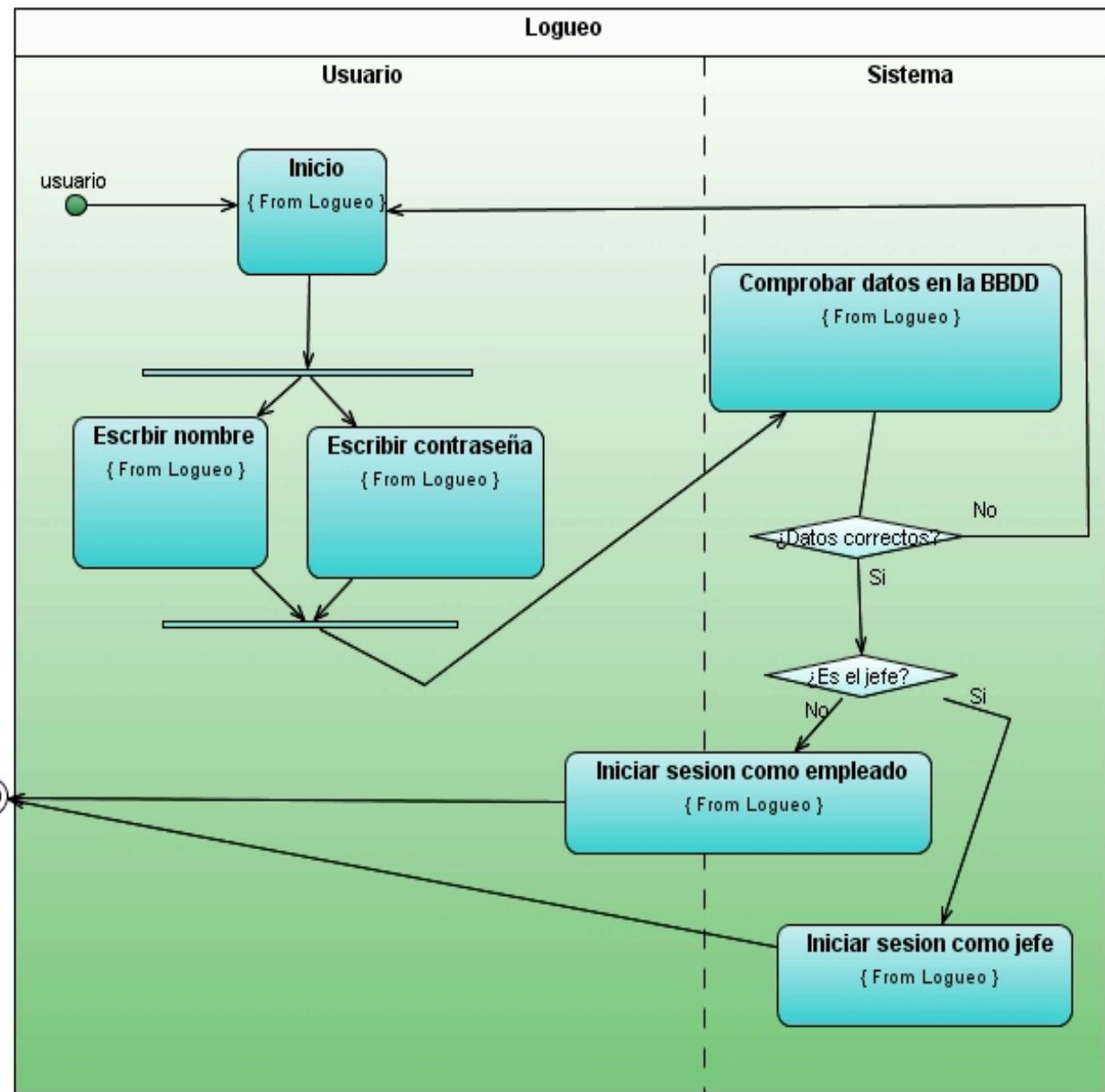
El Usuario envía su mensaje de denuncia y se confirma.
La denuncia tendrá que ser atendida posteriormente por el empleado.



2.6.4. ACTIVIDADEDES DEL EMPLEADO

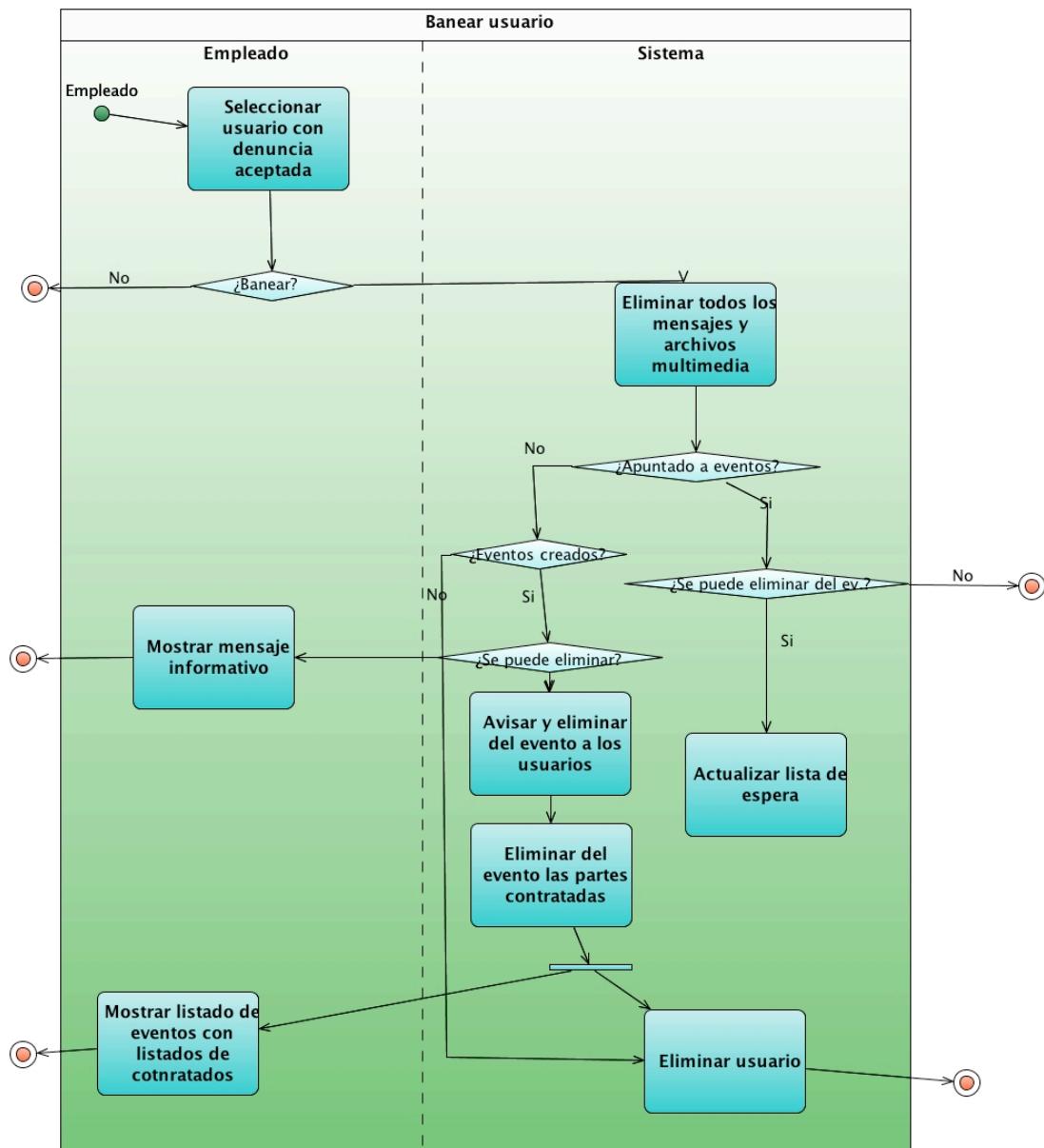
2.6.4.1. LOGUEO

Con usuario nos referiremos, en este diagrama, a empleado y jefe. El usuario escribirá su nombre y contraseña. El sistema comprobará que sus datos se hallan en la BBDD y que son correctos. Si es identificado como jefe, entonces se iniciará la sesión con las opciones de jefe y sino, se iniciará la sesión con las opciones de empleado.



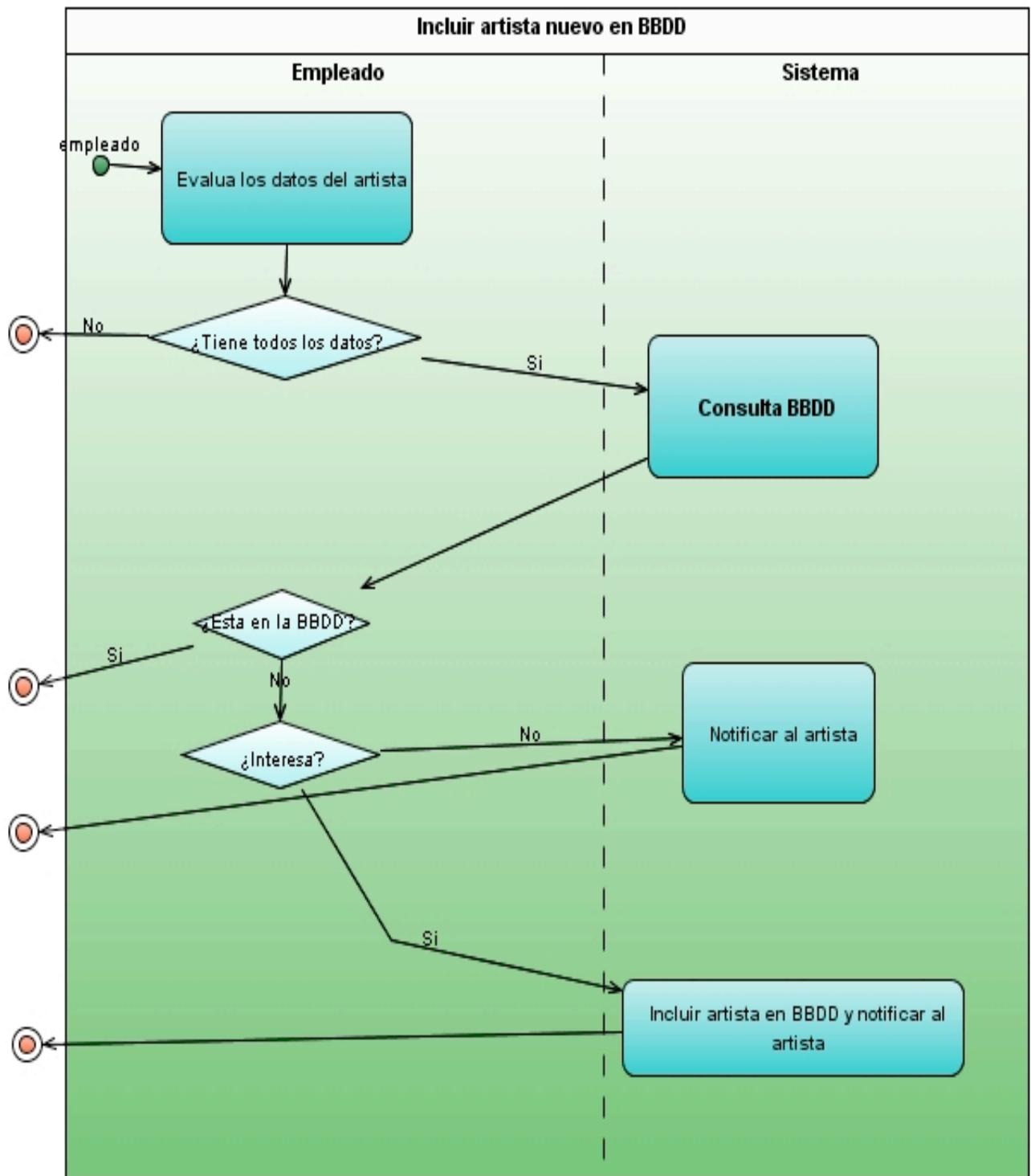
2.6.4.2. BANEAR USUARIO DE LA RED SOCIAL

Este diagrama explica el proceso de Baneo de un Usuario de la Red Social. Se tiene en cuenta las denuncias aceptadas, si el empleado decide banear al usuario, entonces, se eliminan sus mensajes y archivos multimedia y si no tiene eventos pendientes y que no sean eliminables (ya sea como usuario o como cliente) entonces se podrá eliminar al usuario, si no se puede eliminar el usuario será marcado como baneado para ser eliminado en cuanto terminen todos los eventos pendientes.



2.6.4.3. AÑADIR ARTISTA

Para añadir un artista a la BBDD, el empleado debe evaluar sus datos, decidir si interesa a la empresa y comprobar que no está en la BBDD. Si cumple estas condiciones, se añadirá a la BBDD y se le notificará tal acción.

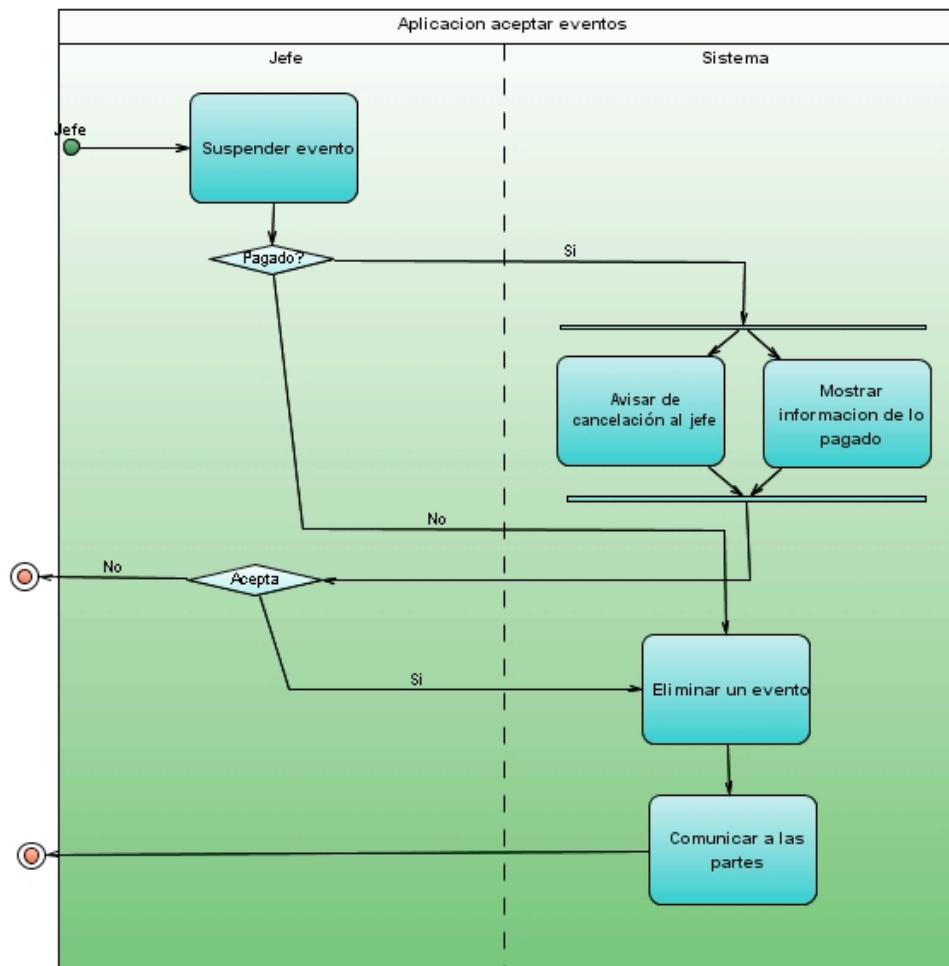


2.6.5. ACTIVIDADES DEL JEFE

2.6.5.1. ANULAR EVENTO

El evento que desea anular el Jefe puede estar o no pagado.

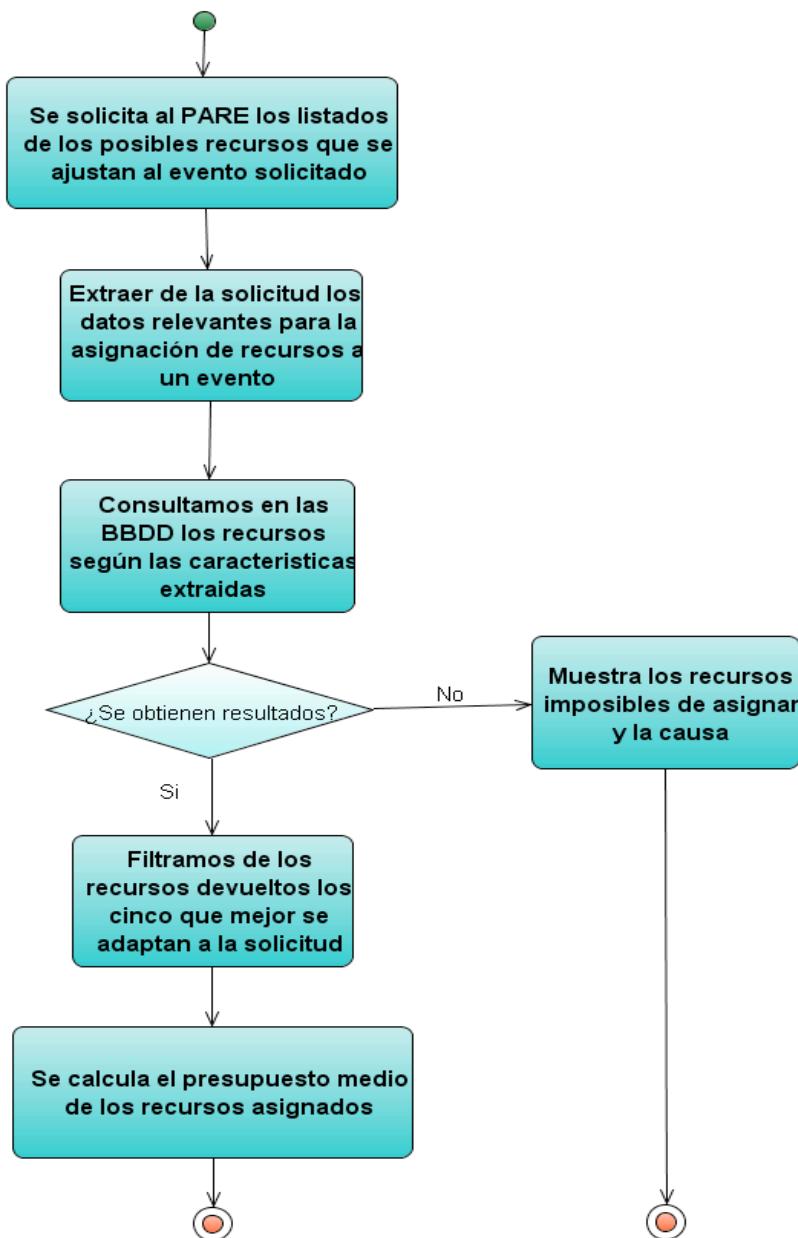
En función de esto, o bien se elimina unilateralmente, o bien se informa al jefe del evento y de lo que hay pagado y toma la decisión.



2.6.6. ACTIVIDADEDES DE P.A.R.E

(PRECÁLCULO DE ASIGNACION DE RECURSOS DE EVENTOS)

2.6.6.1. ASIGNACIÓN DE RECURSOS



2.7. DIAGRAMAS DE COMPONENTES

BBDD

Componente que contendrá las bases de datos donde se almacenará toda la información de la empresa.

SGBD

Componente que gestiona las bases de datos, su espacio físico y nos proporciona interfaces de acceso muy primitivas.

Consta de una interfaz que contiene métodos de acceso a tablas (select), de creación de tablas (create), de borrado de tablas (drop), borrado de información de una tabla (delete), modificación de información contenida en tablas (update) y de las propias tablas (alter), y por ultimo un método para insertar información en las bases de datos (insert).

PARE

Componente que se encarga de buscar recursos en las bases de datos según ciertos criterios y disponibilidad de estos. Se basa en algoritmos de inteligencia artificial, como el algoritmo de Mitchell o el ID3, para encontrar los recursos más adecuados a los criterios introducidos.

Consta de una interfaz llamada *motorAsignación* que constará de métodos para obtener una lista de cinco posibles recursos de un mismo tipo que se ajusten a los criterios pasados al método. Habrá un método por cada uno de los posibles recursos manejados en el sistema, a saber: artistas, locales y subcontratas.

GESTORES

- Gestor de Ficheros

Componente que gestiona ficheros y proporciona interfaces para subir, descargar y borrar dichos archivos.

- Gestor de Usuarios

Componente que proporciona funcionalidad para que los usuarios se registren, hagan login, tengan un perfil y lo actualicen, además permite al programador definir distintos grupos a los que pueden pertenecer los usuarios. En nuestro caso definiremos los siguientes grupos Clientes , Usuarios de la Red Social y Artistas.

- Gestor de Locales

Componente que gestiona la información de los locales y proporciona interfaces para introducirla, modificarla y borrarla.

- Gestor de Subcontratas

Componente que gestiona la información de las subcontratas y proporciona interfaces para introducirla, modificarla y borrarla.

- Gestor de Artistas

Componente que gestiona la información de los artistas y proporciona interfaces para introducirla, modificarla y borrarla. Además proporciona interfaces para modificar la valoración o puntuación de un artista.

- Gestor de Eventos

Componente que gestiona la información de los eventos y proporciona interfaces orientadas a la administración, visualización, y modificación. Estas interfaces estarán orientadas algunas a los empleados de la empresa y algunas a los clientes.

- Gestor Multimedia

Componente que gestiona los contenidos multimedia de la Red Social permitiendo añadir nuevos archivos y visualizar los ya registrados con un visor adecuado.

- Gestor de Mensajes

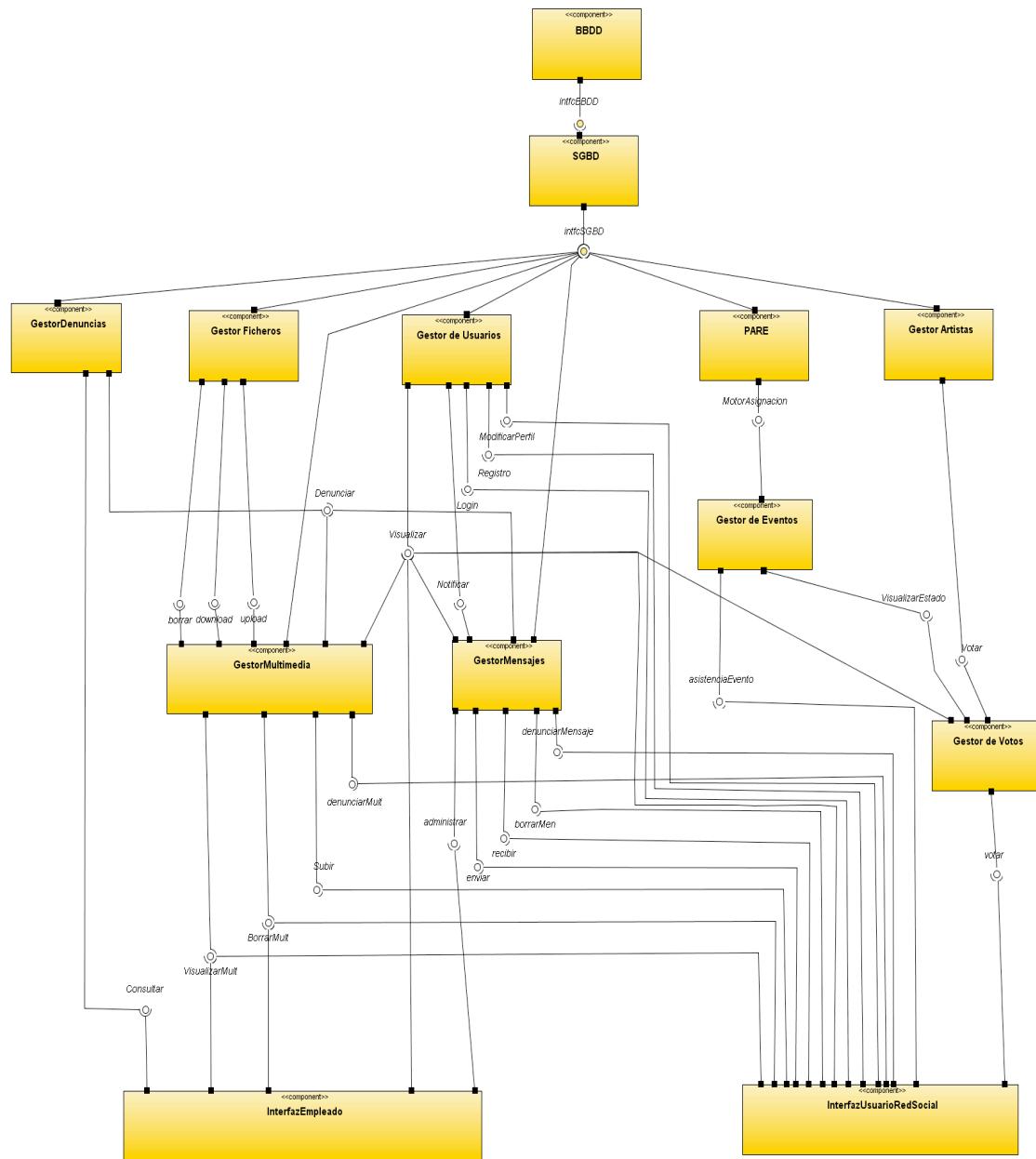
Componente que gestiona los mensajes que se producen en la red social, esto engloba desde los mensajes privados entre usuarios hasta aquellos mensajes públicos. Da funcionalidades de administración y censura.

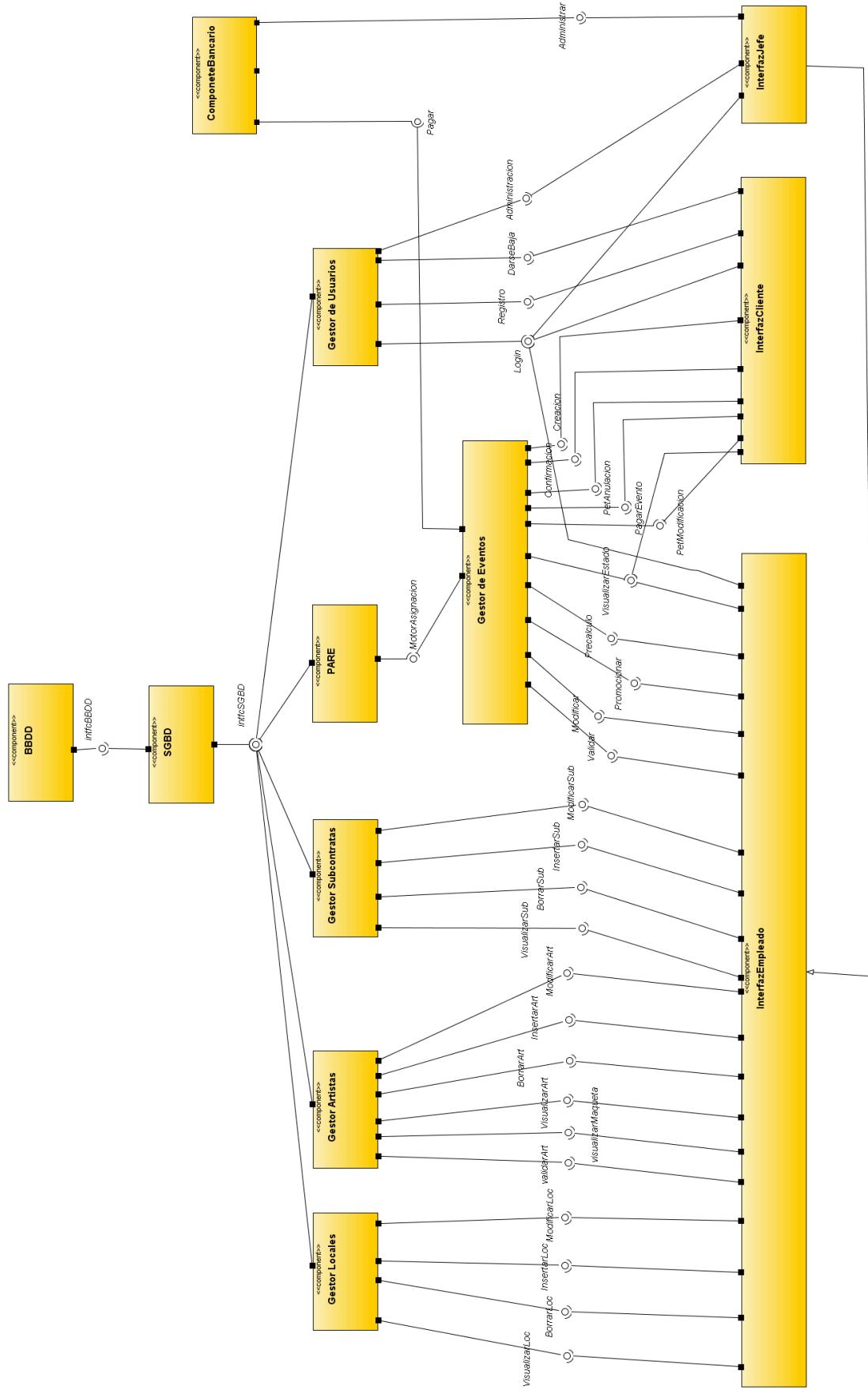
- Gestor de Votos

Componente que gestiona los votos de los usuarios de la red social a los artistas. Se encarga de comprobar que el usuario en cuestión pertenece a la Red Social, que ha asistido al evento en el que el artista al que esta votando ha actuado.

INTERFACES

Todos los componentes llamados Interfaz utilizan uno más de los componentes anteriores para mostrar la información y las opciones en una GUI amigable e intuitiva.





2.8. DIAGRAMAS DE SEGURIDAD

Para una mayor claridad, se ha dividido el diagrama de seguridad según la participación de cada rol y clase. Se ha incluido restricciones que involucren directamente a un llamante o *caller*. Las que son más generales en el sistema, se muestran en las precondiciones y poscondiciones de las clases.

2.8.1. CLIENTE Y EMPLEADO RESPECTO A EVENTO

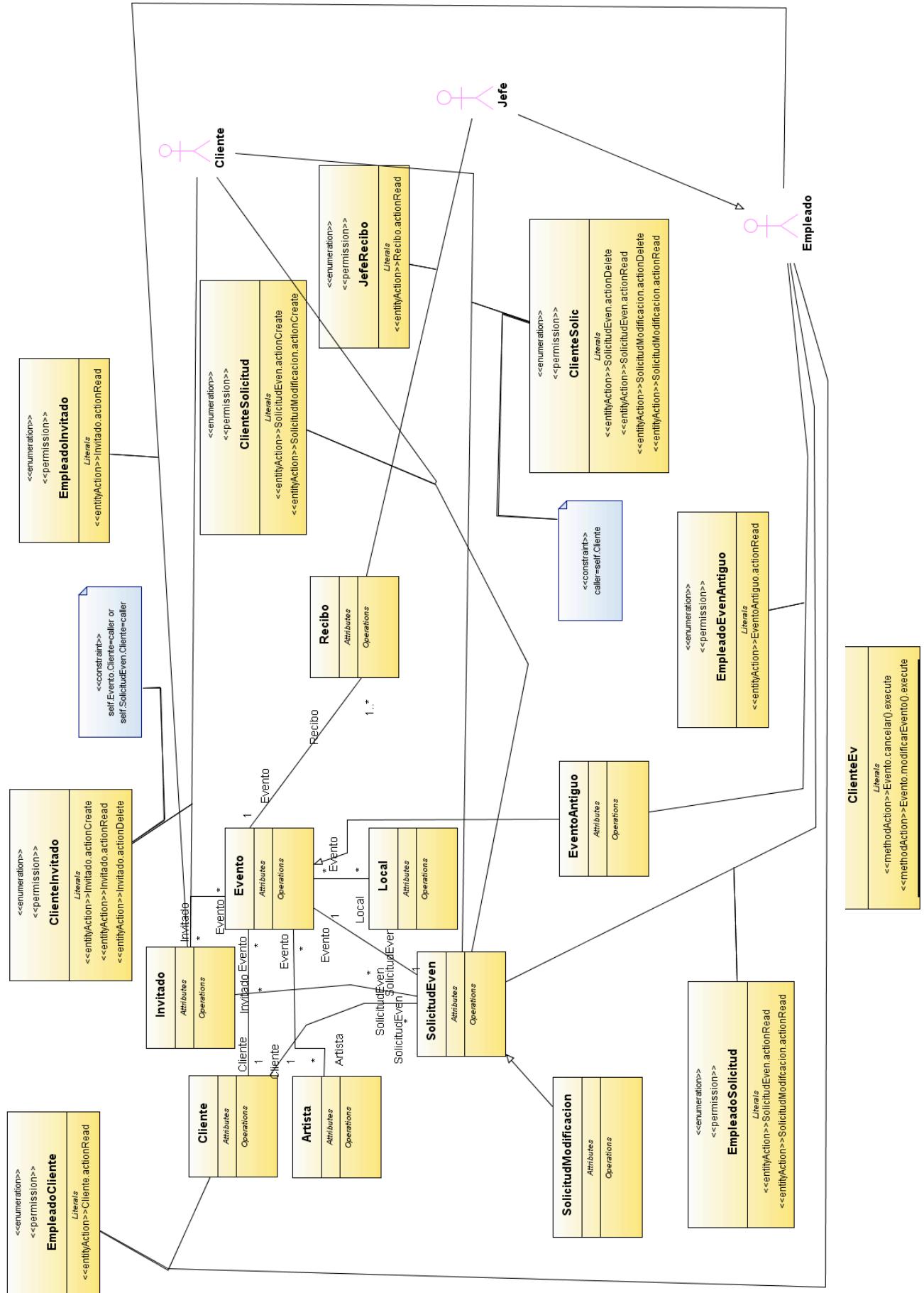
ROLES: Cliente, Empleado y Jefe.

El primer diagrama muestra la relación más directa que hay entre los roles y el Evento ya creado.

El segundo, se centra en los permisos de estos roles con el Evento; y en particular con:

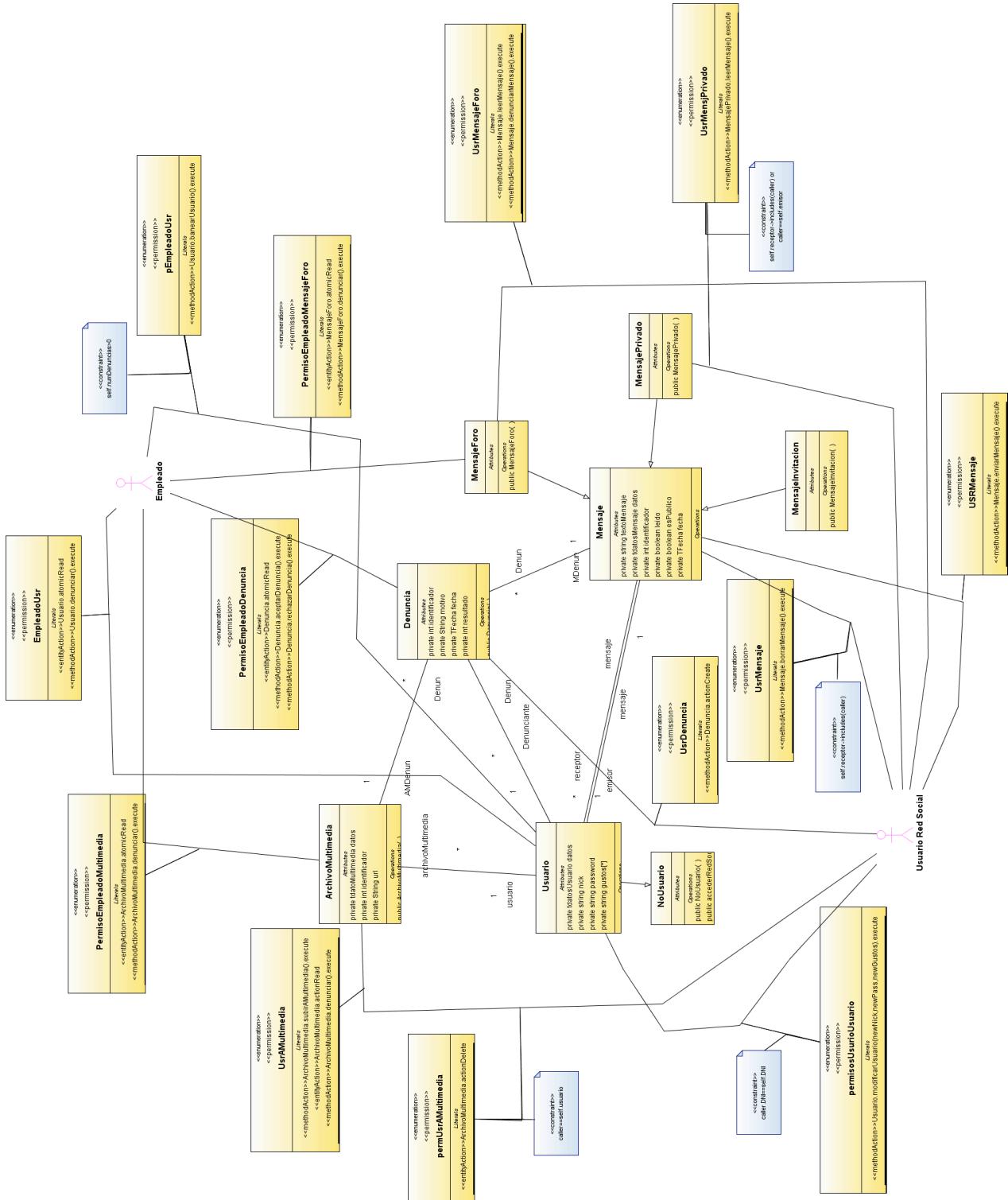
- La solicitud del Evento, que nos permitirá mantener una copia/factura del evento con el que poder respaldarnos, como así mismo el cliente, en cuestiones de contrato.
- Solicitud de modificación, para todo posible cambio por petición del cliente.
- Evento Antiguo, para conservar una copia del estado del Evento, y no sobrescribir la información de éste por cualquier cambio realizado en la asignación de recursos.
- Recibos, solo puede ver el jefe.
- Invitados; el cliente puede leer, crear (añadir), borrar Invitado siempre que sea el dueño/solicitante del evento.

Cliente, el empleado puede ver sus datos (actionRead). En cuanto al borrado del cliente solo se podrá llevar a cabo si este se da de baja o ha pasado un cierto tiempo sin registrar ninguna actividad en eventos.



2.8.2. EMPLEADO Y USUARIO RESPECTO A RED SOCIAL

Se mostrarán los permisos de los roles Empleado y Usuario sobre los recursos de la red Social, Archivos Multimedia, Mensajes, Denuncias y los propios Usuarios



2.8.3. EMPLEADO RESPECTO A EVENTO

Roles Empleado y Jefe

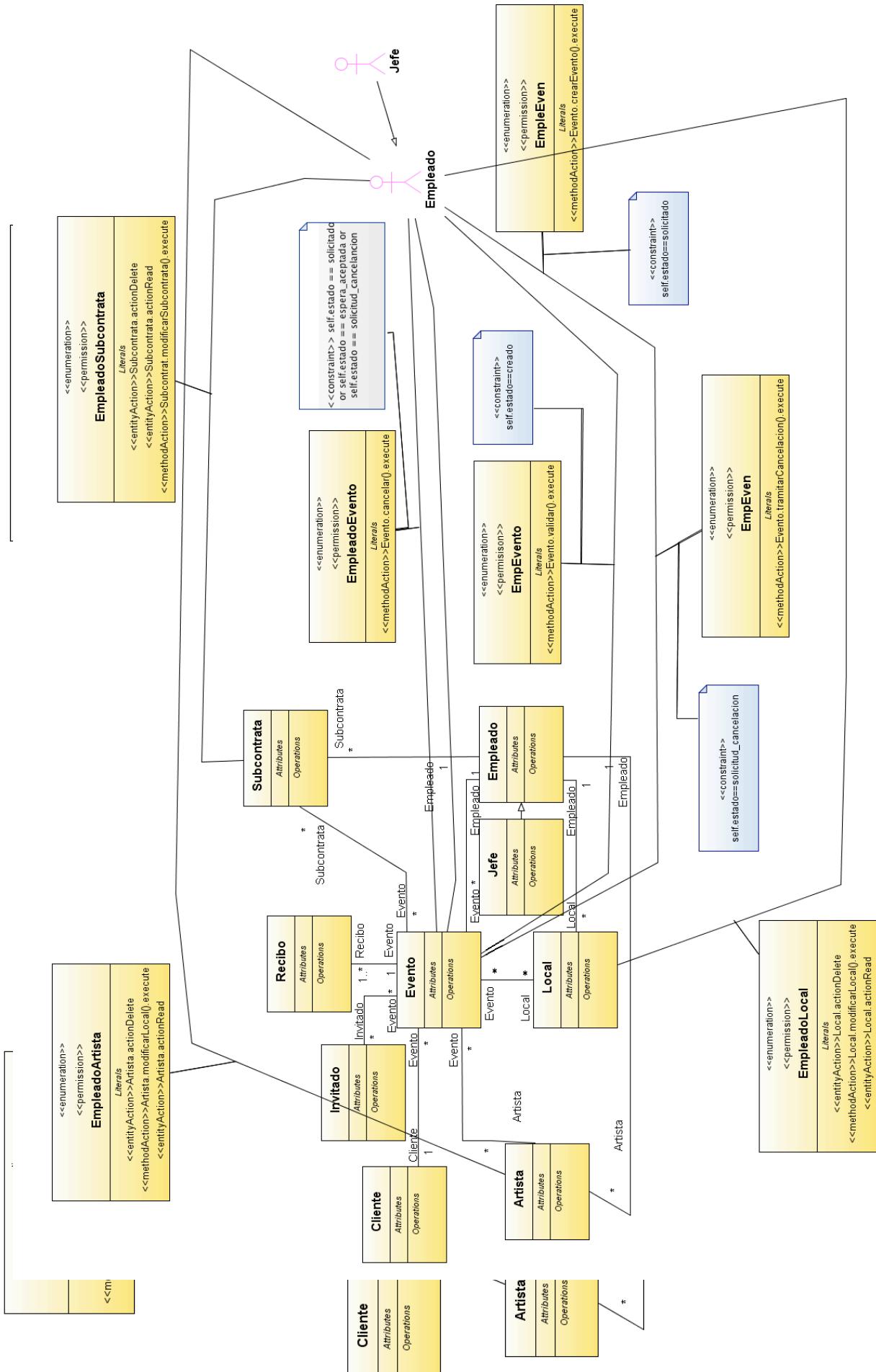
Restricciones para eliminar artistas, locales y subcontratas:

Un empleado podrá eliminarlos siempre que no estén asociados a ningún evento.

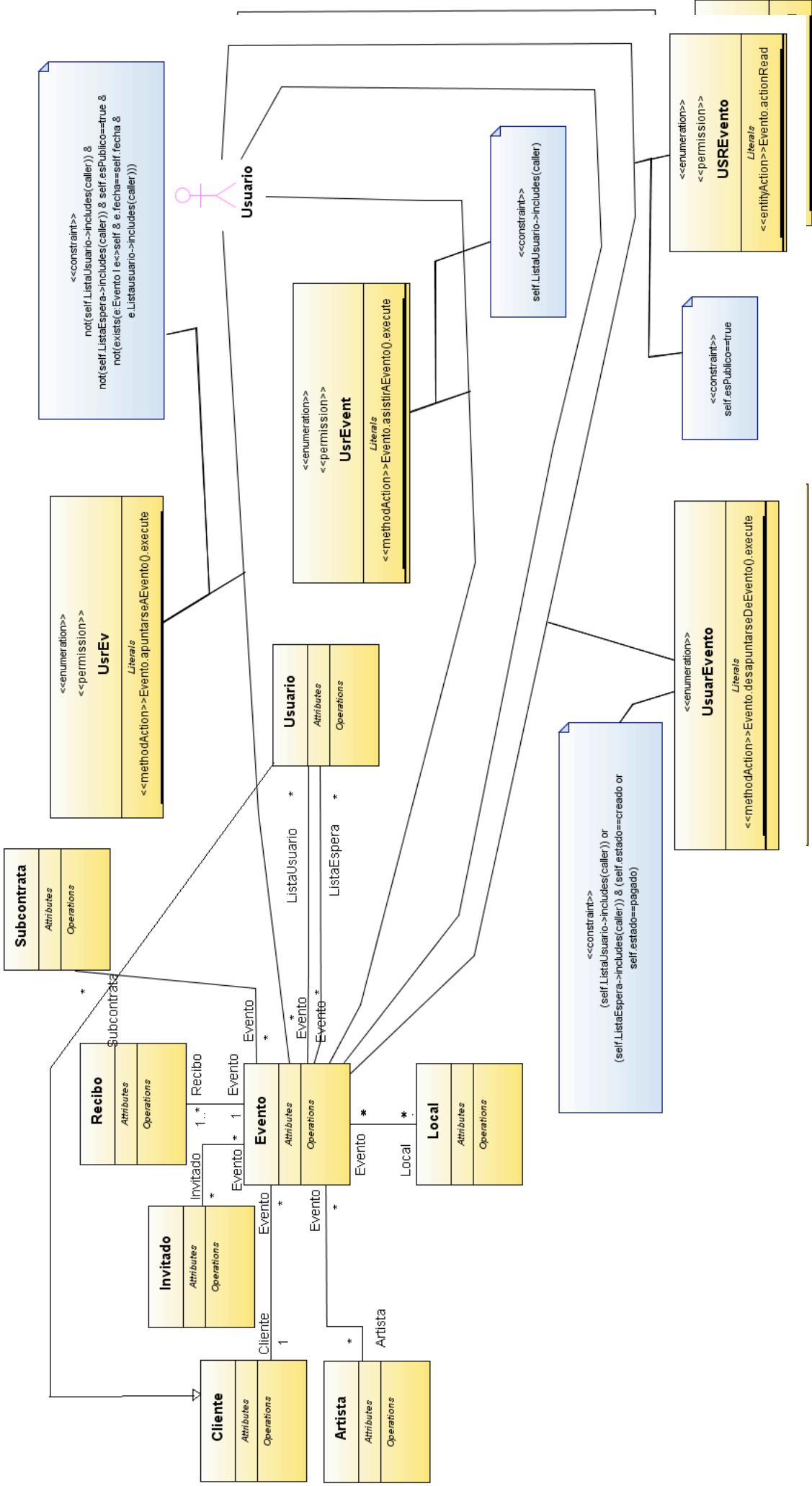
Context Local::eliminarLocal()
Pre: self.Evento->size()=0

Context Artista::eliminarArtista()
pre: self.Evento->size()=0

Context Subcontrata::eliminarSubcontrata()
pre: self.Evento->size()=0



2.8.4. USUARIO RESPECTO A EVENTO



2.9. BASE DE DATOS

2.9.1. TABLAS DE LA BASE DE DATOS

Artista(idArtista, nombre, estilo, Calendario, puntuación, votaciones, zona_preferida, Tarifa, Teléfono, señal, fechaLímite)

En esta tabla se almacenan todos los artistas que pueden participar en los eventos. Para ello guardamos su nombre, el estilo (pop, rock, magia, etc), un calendario con la disponibilidad del artista, la puntuación total de este mismo y el número de votos que ha recibido. El atributo zona_preferida nos servirá para la contratación del artista. A la vez, disponemos de una tarifa base, cuánto dinero quiere por señal, la fecha límite (número de días antes de la realización del evento) para la cancelación del evento y un teléfono de contacto.

Locales(idLocal, Nombre, Dirección, Localidad, Aforo, esInterior, Tarifa, Telefono, tipo, Servicios_Disponibles, Calendario, señal, fechaLímite)

Bajo un identificador de local, tenemos el nombre, dirección y localidad del mismo. Disponemos también de información del local, aforo (cuantas personas entran en el local), tarifa base (cuanto cuesta alquilar el local), tipo (bar, recinto, etc), servicios_disponibles (música, barra, etc) y calendario (disponibilidad del local), esInterior (booleano para saber si el local es interior o exterior), señal (cuánto dinero tenemos que darle como señal) y fechaLímite (número de días antes de la realización del evento para avisar de la cancelación del evento) . Para contactar con el local tenemos un teléfono de contacto.

Clientes(Nombre, Apellidos, DNI, contraseña, Dirección, Teléfono)

Un cliente es aquella persona que contrata nuestros servicios de organización de eventos. Por eso, debemos disponer de los siguientes datos: Nombre, Apellido y DNI, contraseña (para que el cliente valide que es él), Dirección de contacto y Teléfono de contacto.

Usuarios(Nombre, Apellido, DNI, nick, contraseña, gustos,estado)

Un usuario es aquella persona que utiliza la red social, sin tener por ello que utilizar nuestro servicio de organización de eventos, por ello, hasta que un usuario no realice un servicio de cliente, el idCliente permanecerá a null. Del usuario, guardamos, entre otros datos, el nick (como quiere ser llamado en la red social), contraseña (junto con el nick servirán para acceder a la red social), una lista de gustos, esta lista la usaremos para invitarle a eventos, enviar publicidad, etc y el estado del usuario (normal o baneado) lo usaremos para banear automáticamente a usuarios que no han podido ser baneados por tener eventos pendientes.

Subcontratas (idSubcontrata, Nombre, Servicio, Teléfono, Tarifa, Calendario, señal, fechaLímite)

En esta tabla, se almacena el nombre de la subcontrata, el servicio que ofrece (catering, inmobiliario, etc), tarifa base y calendario de disponibilidad. Tenemos también cuánto cobra como señal y con cuánto tiempo límite podemos avisar de la cancelación del evento. Para contactar con la subcontrata disponemos de un teléfono.

Mensajes (idMensaje, textoMensaje, emisor, receptor, tamaño, leido, esPublico, Fecha)

En esta tabla guardamos los mensajes de la red social. Consta de los siguientes atributos, textoMensaje (texto escrito por el emisor), emisor (persona que envía el mensaje), receptor (usuario que recibe el mensaje), tamaño (tamaño del mensaje para limitar la cantidad de almacenaje), leído (atributo booleano para avisar al receptor que tiene mensajes por leer), esPublico (atributo booleano para hacer visible el mensaje a la Red Social en el foro del receptor), fecha (fecha de envío del mensaje, para mostrar los mensajes en orden).

ArchivosMultimedia (idArchMult, url, tamaño, fecha, idUsuario, tipoFormato)

Almacenamos en esta tabla los datos suficientes para tratar con archivos multimedia. Para ello, precisamos de url (dirección física del archivo multimedia), tamaño (tamaño del archivo multimedia), fecha (fecha de subida del archivo multimedia) y tipoFormato (audio, video, etc).

SolicitudEvento(idSolicitud, idEvento, Fecha, duracion, localidad, esPublico, aforoMin, aforoMax, tipoEvento, fechaLimite, numArtistas, numLocales)

Se guarda los datos que el usuario a introducido en la solicitud del evento.

SolicitudModificacion(idModificacion, idEvento, Fecha, duracion, localidad, esPublico, aforoMin, aforoMax, tipoEvento, fechaLimite, numArtistas, numLocales)

Se guarda los datos que el usuario a introducido para modificar un evento.

Eventos (idEvento, idCliente, esPublico, Fecha, AforoMin, AforoMax, estado, precioFinal, tipoEvento, Duracion, Localidad, Direccion, fechaLimite, fechaSolicitudModificacion)

Almacenamos en esta tabla idCliente (identifica al cliente que solicitó el evento), esPublico (atributo booleano, si es verdadero, entonces los usuarios de la red social pueden apuntarse al evento libremente, si es falso, entonces sólo podrán asistir aquellas personas que están en la lista de invitados correspondiente con el evento), Fecha (fecha de realización del evento), Aforo (número de personas que pueden asistir al evento), estado (fase por la que pasa el evento, solicitado, creado, cancelado, realizado, etc), precioFinal (dependiendo del estado, será o una estimación del precio del evento o el precio real del evento, por ejemplo, si el estado es solicitado, el precio es una estimación, en cambio en los demás estados el precio es el precio del evento), tipoEvento (fiesta, cumpleaños, congreso, etc), duración (duración del evento), localidad (zona por donde se debe organizar el evento), dirección (dirección final del evento, dirección del local), fechaLimite (número de días antes de un evento, en los que se puede modificar el evento) y fechaSolicitudModificacion (guardamos la fecha de solicitud del evento o de modificación del mismo, para poder saber cuántos días han pasado desde que el usuario haya solicitado o modificado el evento).

EventoAntiguo (idEvento, idCliente, esPublico, Fecha, AforoMin, AforoMax, estado, precioFinal, tipoEvento, Duracion, Localidad, Direccion, fechaLimite, fechaSolicitudModificacion)

Si el usuario manda una solicitud de modificación se guarda el evento creado hasta el momento, por si acaso, rechaza la modificación, poder hacer un back up del mismo.

ListaArtistasEvento (idEvento, idArtista)

En esta tabla guardamos los artistas que participan en un evento.

ListaLocalesEvento (idEvento, idLocal)

En esta tabla guardamos los locales contratados para un evento.

ListaSubcontrataEvento (idEvento, idSubcontrata)

Almacenamos en esta tabla las subcontratas contratadas para un evento.

ListaInvitadosPrivado(idEvento,Nombre)

El cliente crea un evento privado y proporciona una lista con los nombres de los invitados.

ListaInvitadosPublicos (idEvento, idUsuario)

El cliente crea un evento publico al que se puede apuntar cualquier usuario de la red social. El usuario de la red social sólo se puede apuntar una vez a un evento. Esta lista será usada, cuando el evento haya sido realizado, para saber quién puede valorar al artista que ha participado en el evento.

ListaDeEspera (idEvento, idUsuario, numSec)

Almacenamos en esta tabla aquellos usuarios que se han intentado apuntar a un evento público y no han podido por estar el aforo completo.

Denuncias (idDenuncia, idDenunciante, idDenunciado, idMensaje, idArchMult, motivo, fecha, resultado)

Almacenamos en esta tabla las denuncias que usuarios (idDenunciante) hacen a contenidos (idMensaje o idArchMult) de otro usuario (idDenunciado). El denunciante debe dar un motivo justificado. Se guarda la fecha de la denuncia para procesarlas en orden y el resultado de la misma una vez procesada, los valores válidos son prospera o rechazada.

Recibo (idRecibo, idEvento, cantidad)

En esta tabla almacenamos todos los recibos en los que se puede pagar un evento.

Empleado(idEmpleado, Nombre, Apellido, DNI, direccion, telefono, contraseña)
Guardamos los datos correspondientes a los empleados

Jefe(idJefe, , Apellido, DNI, direccion, telefono, contraseña)
Guardamos los datos correspondientes al jefe

2.9.2. FUNCIONES DE LA BASE DE DATOS

- ARTISTA

crearArtista(idA, nombre, estilo, Calendario, z_p, Tarifa, Teléfono, señal, fechaLimite)

pre:

not(Artista.filas -> exists(f1: fila | f1.idArtista=idA))

post:

Artista.filas -> exists(f1:fila | f1.idArtista = idA and
f1.Nombre = nombre and
f1.estilo = estilo and
f1.Calendario = Calendario and
f1.puntuacion = 0 and
f1.votaciones = 0 and
f1.zona_preferida = z_p and
f1.Tarifa = Tarifa and
f1.Teléfono = Teléfono and

f1.señal = señal and
f1.fechaLimite = fechaLimite)

eliminarArtista(idA)

pre:

Artista.filas -> exists(f1:fila | f1.idArtista = idA)

post:

not(Artista.filas -> exists(f1: fila | f1.idArtista=idA))

modificarArtista(idA,e1,c1,z1,t1,tel1,s1)

pre:

Artista.filas -> exists(f1:fila | f1.idArtista = idA)

post:

Artista.filas -> exists(f1:fila | f1.idArtista = idA and
f1.estilo = e1 and
f1.Calendario = c1 and
f1.zona_preferida = z1 and
f1.Tarifa = t1 and
f1.Teléfono = tel1 and

f1.señal = f1.s1 and
)

votarArtista(idU, idA, idE, voto)

pre: se puede votar al artista si el artista ha ido al evento, el invitado también y el estado del evento es realizado

Eventos.filas -> exists(f1:fila | f1.idEvento = idE and f1.estado = realizado)

ListaArtistasEventos.filas -> exists(f1:fila | f1.idEvento = idE and f1.idArtista = idA)

ListaInvitadosPublicos.filas -> exists(f2:fila|f2.idEvento = idE and
f2.idUsuario = idU)

post: al terminar el método, el artista debe haber incrementado su puntuacion y votaciones y el usuario debe haber sido eliminado de la lista de invitados para que no pueda volver a votar.

Artista.filas -> exists(f1:fila | f1.idArtista = idA and
f1.puntuacion = f1.puntuacion@pre + voto and

```

f1.votaciones = f1.votaciones@pre + 1 and
)
not(ListaInvitadosPublicos.filas -> (f2.filas|f2.idEvento = idE and
f2.idUsuario = idU)

```

- LOCALES

crearLocal(idL,n,d,l,a,e,t,tel,tip,s,c,señ,fech)

pre:

not(Local.filas -> exists(f1: fila | f1.idLocal=idL))

post:

Local.filas -> exists(f1: fila | f1.idLocal=idL and
f1.Nombre = n and
f1.Direccion = d and
f1.Localidad = l and
f1.Aforo = a and
f1.esInterior = e and
f1.Tarifa = t and
f1.Telefono = tel and
f1.Tipo = tipo and
f1.Servicios_Disponibles = s and
f1.Calendario = c and

f1.señal = señ and

f1.fechaLimite = fech)

eliminarLocal(idL) : similar a eliminarArtista

modificarLocal(idL,n,t,tel,tip,s,c,señ)

pre:

Local.filas -> exists(f1: fila | f1.idLocal=idL)

post:

Local.filas -> exists(f1: fila | f1.idLocal=idL and
f1.Nombre = n and
f1.Tarifa = t and
f1.Telefono = tel and
f1.Tipo = tipo and
f1.Servicios_Disponibles = s and
f1.Calendario = c and

f1.señal = señ and

)

- CLIENTES

crearCliente : similar a crearArtista, pero con los datos del cliente.

eliminarCliente (idC) : similar a eliminarArtista

modificarCliente : similar a modificarArtista pero sólo modificando los atributos de entrada

verificarContraseñaCliente(idC, pass)

body:

b: boolean

b:= (Clientes.filas->exists(f1:fila | f1.idCliente = idC and

```
f1.contraseña = pass))
```

- USUARIOS

crearUsuario : similar a crearArtista, pero con los datos del cliente. El estado del usuario será inicializado a normal.

eliminarUsuario

modificarUsuario

verificarContraseñaUsuario

- SUBCONTRATAS

crearSubcontrata

eliminarSubcontrata

modificarSubcontrata

- MENSAJES

enviarMensaje(idM,text,em,re,tam,pub)

Si el mensaje es publico se marca como leído directamente, si es privado no.

pre:

```
not(Mensajes.filas->exists(f1:fila | f1.idMensaje = idM))
```

post:

```
Mensajes.filas -> exists (f1:fila | f1.idMensaje = idM and  
f1.textoMensaje = text and  
f1.emisor = em and  
f1.receptor = re and  
f1.tamaño = tam and  
f1.leido = pub and  
f1.esPublico = pub and  
f1.fecha = sysdate())
```

leerMensaje(idM,idU)

pre:

```
Mensajes.filas -> exists (f1:fila | f1.idMensaje = idM and  
f1.receptor = idU)
```

post:

```
Mensaje.filas -> exists (f1:fila | f1.idMensaje = idM and  
f1.leido = true and  
)
```

borrarMensaje(idM, idU)

pre:

```
Mensajes.filas -> exists(f1:fila | f1.idMensaje = idM and  
f1.receptor = idU)
```

post:

```
not(Mensajes.filas ->exists(f1:fila | f1.idMensaje = idM))
```

- ARCHIVOS MULTIMEDIA

subirArchivoMultimedia(idAM,url,idC,tam,es,idU, tipo)

pre:

```
not(ArchivosMultimedia.filas -> exists (f1:fila | f1.idArchMult = idAM))
```

post:

```

ArchivosMultimedia.filas -> exists(f1:fila | f1.idArchMult = idAM and
f1.url = url and
f1.tamaño = tam and
f1.estado = subido and
f1.fecha = sysdate() and
f1.idUsuario = idU and
f1.tipoFormato = tipo)

```

borrarArchivoMultimedia(idAM, idU)

pre:

```

ArchivosMultimedia.filas -> exists(f1:fila | f1.idArchMult = idAM and
f1.receptor = idU)

```

post:

```

not(ArchivosMultimedia.filas -> exists(f1:fila | f1.idMensaje = idAM))

```

consultarArchivoMultimedia

- EVENTOS

crearEvento(idE,idC,[idA],[idL],[idS],esP,fecha,af_m,af_M,pF,tipoe,Dur,Zon,Dir)

pre:

- not(Eventos.filas -> exists(f1:fila | f1.idEvento = idE))
- los artistas están disponibles
- las subcontratas están disponibles
- los locales están disponibles

post:

```

Eventos.filas -> exists(f1:fila | f1.idEvento = idE and
f1.idCliente = idC and
f1.esPublico = esP and
f1.fecha = fecha and
f1.aforoMin = af_m and
f1.aforoMax = af_M and
f1.estado = solicitado and
f1.precioFinal = pF and
f1.pagadoHastaElMomento = 0 and
f1.tipoEvento = tipoe and
f1.Duracion = Dur and
f1.Zona = Zon and
f1.Direccion = Dir)

```

- meter en la tabla de Artistas los identificadores de la lista
- meter en la tabla de Subcontratas los identificadores de la lista
- meter en la tabla de Locales los identificadores de la lista

modificarEvento(idE,idC,[idA],[idL],[idS],esP,fecha,af_m,af_M,pF,tipoe,Dur,Zon,Dir)

pre: (Eventos.filas->exists(f1:fila | f1.idEvento = idE and
f1.estado = creado)) fianza pagada

post: Eventos.filas -> exists(f1:fila | f1.idEvento = idE and
f1.idCliente = idC and
f1.esPublico = esP and
f1.fecha = fecha and
f1.aforoMin = af_m and
f1.aforoMax = af_M and
f1.estado = solicitado and

$\text{pagadoHastaElMomento} @ \text{pre}$ and
 $f1.\text{precioFinal} = pF$ and
 $f1.\text{pagadoHastaElMomento} =$
 $f1.\text{tipoEvento} = \text{tipoE}$ and
 $f1.\text{Duracion} = \text{Dur}$ and
 $f1.\text{Zona} = \text{Zon}$ and
 $f1.\text{Direccion} = \text{Dir}$)

pagarCantidad

cambiarEstadoEvento(idE, estado)

$\text{pre: } (\text{Eventos.filas} \rightarrow \exists(f1:\text{fila} \mid f1.\text{idEvento} = idE))$

$\text{post: } \text{Eventos.filas} \rightarrow \exists(f1:\text{fila} \mid f1.\text{idEvento} = idE \text{ and } f1.\text{estado} = \text{estado})$

enviarPublicidadEvento(idE, texto)

body: seleccionamos el tipo de evento y enviamos un mensaje a todos los usuarios que tienen gustos similares. El mensaje se envia como público para que la gente pueda leerlo.

$\text{pre: } (\text{Eventos.filas} \rightarrow \exists(f1:\text{fila} \mid f1.\text{idEvento} = idE \text{ and } f1.\text{esPublico} = \text{true}))$

body:
 tipo: String
 $\text{tipo:}= \text{Eventos.filas} \rightarrow \text{select}(f1:\text{fila} \mid f1.\text{idEvento} = idE) \rightarrow \text{first()}. \text{tipoEvento}$
 $\text{Usuarios.filas} \rightarrow \text{select}(f1:\text{fila} \mid f1.\text{gustos} = \text{tipo})$
 $\rightarrow \text{forAll}(f2:\text{fila} \mid \text{enviarMensaje(null, texto, null, f2.DNI, null, true)))$

apuntarseAEVENTO (idU, idE)

pre:

$\text{Eventos.filas} \rightarrow \exists(f1:\text{fila} \mid f1.\text{idEvento} = idE \text{ and } f1.\text{Fecha} > \text{sysdate}())$
 $\neg(\text{ListaInvitadosPublicos.filas} \rightarrow \exists(f2:\text{fila} \mid f2.\text{idEvento} = idE \text{ and } f2.\text{idUsuario} = idU))$
 $\neg(\text{ListaDeEspera.filas} \rightarrow \exists(f3:\text{fila} \mid f3.\text{idEvento} = idE \text{ and } f3.\text{idUsuario} = idU))$

post:

$(\text{ListaInvitadosPublicos.filas} \rightarrow \exists(f2:\text{fila} \mid f2.\text{idEvento} = idE \text{ and } f2.\text{idUsuario} = idU)) \text{ or }$
 $(\text{ListaDeEspera.filas} \rightarrow \exists(f3:\text{fila} \mid f3.\text{idEvento} = idE \text{ and } f3.\text{idUsuario} = idU))$

desapuntarseAEVENTO (idU, idE)

pre:

Te puedes desapuntar de un evento si te desapuntas antes de los días límites a la fecha de realización del evento

$\text{Eventos.filas} \rightarrow \exists(f1:\text{fila} \mid f1.\text{idEvento} = idE \text{ and } (f1.\text{Fecha} - f1.\text{fechaLimite}) > \text{sysdate}()))$

post:

$\neg(\text{ListaInvitadosPublicos.filas} \rightarrow \exists(f2:\text{fila} \mid f2.\text{idEvento} = idE \text{ and } f2.\text{idUsuario} = idU))$

not (ListaDeEspera.filas -> exists (f3:fila | f3.idEvento = idE and f3.idUsuario = idU))

añadirFechaLímite(idE,fecha) : añadir fecha límite a un evento antes de que el usuario haya pagado la señal.

- LISTA ARTISTA EVENTO

crearListaDePosiblesArtistas (idE)

pre:

not(ListaArtistaEvento .filas -> existis(f1:fila|f1.idEvento = idE))

post:

Seleccionamos los artistas que tengan el calendario disponible para la fecha del evento, la zona en la que se realiza el evento sea la misma y con el mismo estilo que el evento.

Artistas.filas -> exists(f1:fila | ListaArtistaEventos.filas ->
 exists(f2:fila| f2.idEvento = idE and
 f2.zona = f1.zona_preferida and
 f2.idArtista = f1.idArtista))

eliminar artista no disponible

eliminar resto de artistas (menos los artistas seleccionados)

- LISTA LOCALES EVENTO

crear lista de posibles locales

eliminar local no disponible

eliminar resto de artistas (menos los locales seleccionados)

- LISTA SUBCONTRATA EVENTO

crear lista de posibles locales segun lo pedido

eliminar subcontrata no disponible

eliminar resto de subcontratas (menos las subcontratas seleccionadas)

- LISTA INVITADOS PRIVADO

crear lista de invitados

eliminar lista

eliminar invitado

añadir invitado

- DENUNCIAS

Denunciar (idD, idDenunciante, idDenunciado, idMensaje, idArchMult, motivo)

pre:

```
not(Denuncias.filas->exists(f1:fila|f1.idDenuncia = idD)) and  
Usuarios.filas ->exists(f1:fila|f1.idCliente = idDenunciante)) and  
Usuarios.filas ->exists(f1:fila|f1.idCliente = idDenunciado)) and  
[Mensajes.filas ->exists(f1:fila| f1.idMensaje = idMensaje) or  
ArchivosMultimedia ->exists(f1:fila | f1.idArchMult = idArchMult)]
```

post:

```
Denuncias.filas->exists(f1:fila|f1.idDenuncia = idD) and  
f1.idDenunciante = idDenunciante and  
f1.idDenunciado = idDenunciado and  
f1.idMensaje = idMensaje and  
f1.idArchMult = idArchMult and  
f1.motivo = motivo and  
f1.fecha = sysdate() and  
f1.resultado = en_proceso)
```

aceptarDenuncia(idD)

pre:

```
Denuncias.filas -> (f1:fila|f1.idDenuncia = idD and  
f1.resultado = en_proceso)
```

post:

```
Denuncias.filas -> (f1:fila|f1.idDenuncia = idD and  
f1.resultado = aceptada)
```

rechazarDenuncia(idD)

pre:

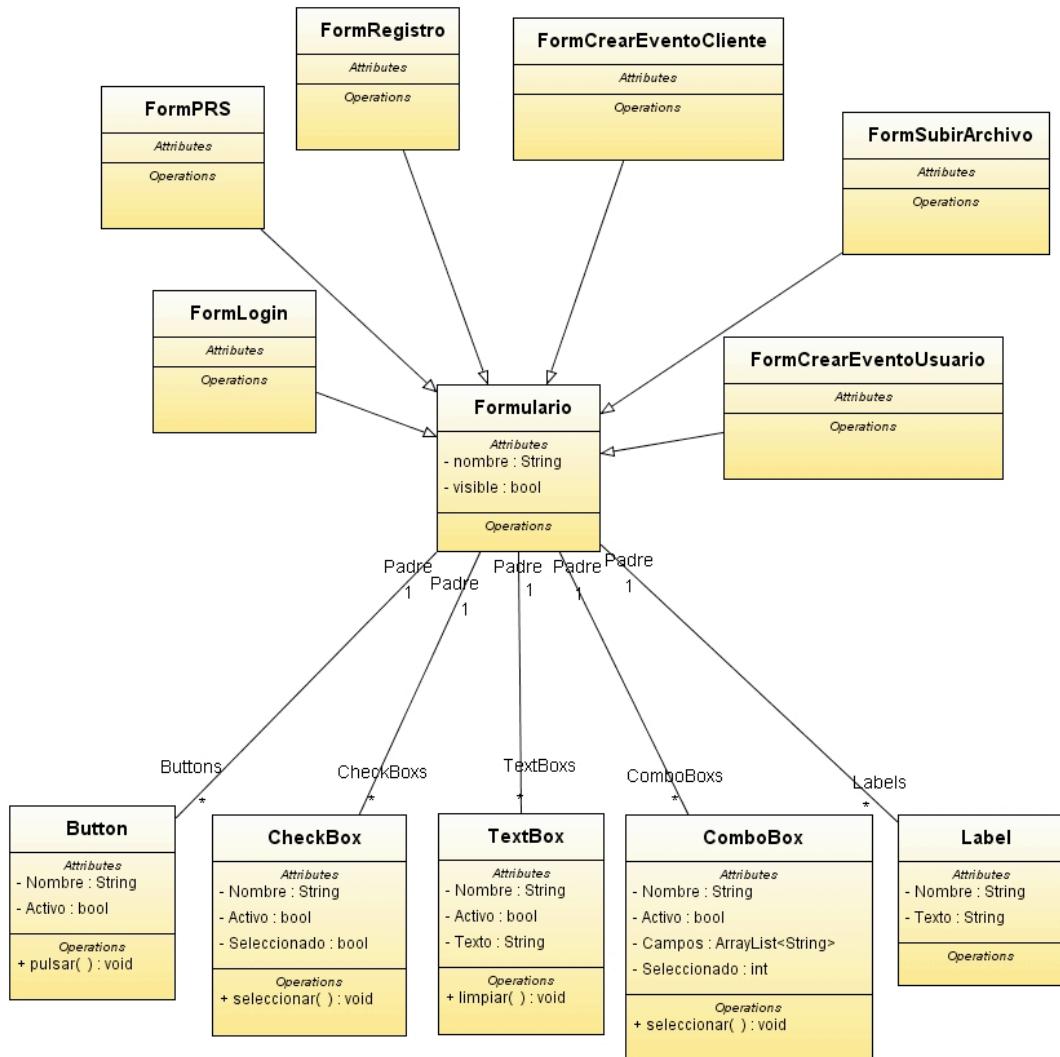
```
Denuncias.filas -> (f1:fila|f1.idDenuncia = idD and  
f1.resultado = en_proceso)
```

post:

```
Denuncias.filas -> (f1:fila|f1.idDenuncia = idD and  
f1.resultado = rechazada)
```

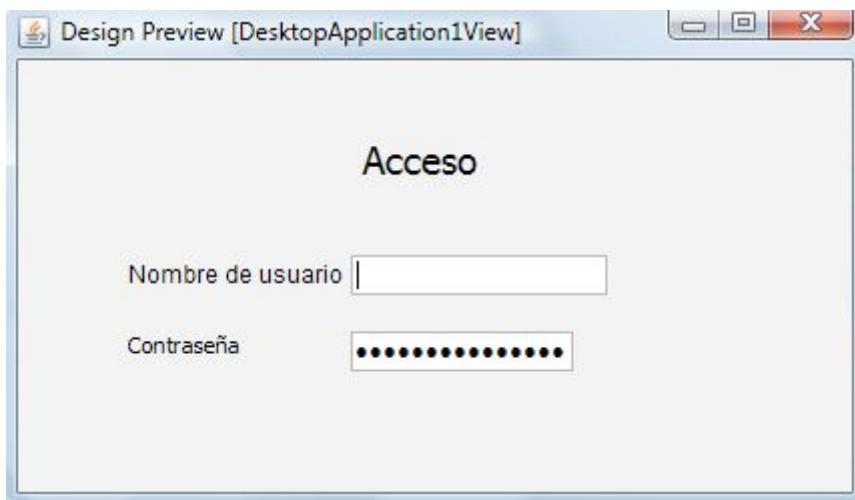
3. INTERFAZ GRÁFICA

3.1. DRIAGAMA DE CLASES DE INTERFAZ



3.2. APARIENCIA DE LA INTERFAZ

Ventana de Login del sistema:



3.2.1. INTERFAZ EMPLEADO

Aquí puede ver el menú principal al que se accede como empleado:



Permite acceder a las secciones de:

-Crear evento:

Se rellenan los datos y al hacer clic en crear se añaden a la BBDD y se cierra la ventana.

The screenshot shows a Windows-style application window titled "Design Preview [CrearEvento]". The window has a yellow header bar with standard minimize, maximize, and close buttons. The main area is titled "Rellene los datos del evento:" and contains the following fields:

Público	<input type="checkbox"/>
Fecha	<input type="text"/>
Aforo Mín	<input type="text"/>
Aforo Máx	<input type="text"/>
Tipo evento	<input type="text" value="Tipo 1"/> <input type="button" value="▼"/>
Duración (min...)	<input type="text"/>
Zona	<input type="text"/>
Dirección	<input type="text"/>

At the bottom of the window are three buttons:

-
-
-

-Gestionar locales:

Permite modificar el valor los atributos de los locales

Primero se seleccionan uno se llevan a cabo los cambios pertinentes y ese cambio se refleja en la BBDD cuando se selecciona guardar cambios.

Para crear uno nuevo se rellenan los campos y al seleccionar crearla como nueva, se inserta la nueva entrada en la BBDD

Seleccione un Local para editarlo o rellene y cree una nueva

Local 1
Local 2
Local 3
Local 4
Local 5

Nombre

Dirección

Localidad

Aforo

Teléfono

Interior

Tipo

Tarifa

Servicios

Ok regresa al menú principal.

-La gestión de clientes, artistas y subcontratas es similar:

Seleccione una subcontrata para editarla o rellene y cree una nueva

Subcontrata 1	Nombre <input type="text"/>
Subcontrata 2	Teléfono <input type="text"/>
Subcontrata 3	Servicio <input type="text"/>
Subcontrata 4	Tarifa <input type="text"/>
Subcontrata 5	

Seleccione un artista para editarlo o rellene y cree una nueva

Artista 1	Nombre <input type="text"/>
Artista 2	Puntuación <input type="text"/>
Artista 3	Estilo <input type="text"/>
Artista 4	Zona <input type="text"/>
Artista 5	Tarifa <input type="text"/>
	DNI <input type="text"/>

Design Preview [GestionarClientes]

Seleccione un Cliente para editarlo o rellene y cree una nueva

Cliente 1	Nombre	<input type="text"/>
Cliente 2	Apellidos	<input type="text"/>
Cliente 3	Teléfono	<input type="text"/>
Cliente 4	Dirección	<input type="text"/>
Cliente 5	DNI	<input type="text"/>
		<input type="text"/>

Crearla como nueva **Guardar cambios del cliente seleccionado**

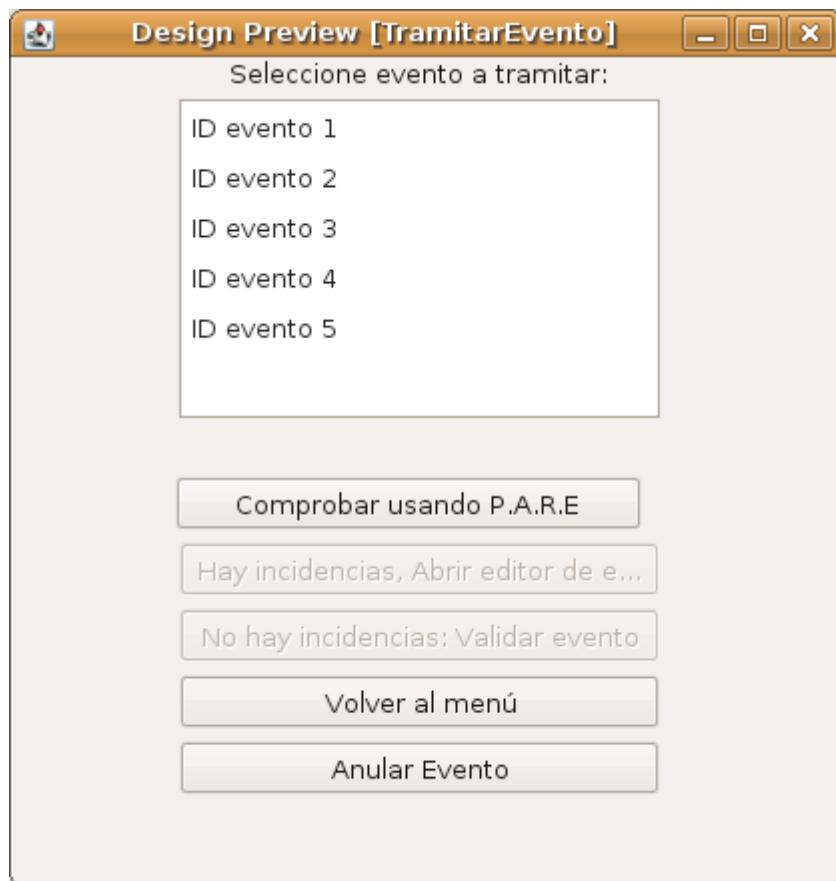
Ok

-Tramitar eventos:

Permite visualizar los eventos pendientes de tramitar, se selecciona uno de la lista y para que el sistema lo procese se hace clic en comprobar usando P.A.R.E.

Permite anular un evento si la petición no puede ser satisfecha.

Si el evento no ha podido ser calculado completamente, se habilita el botón de "hay incidencias", sinó el de "no hay incidencias" al hacer clic en cualquiera de ellas muestra los datos del evento.



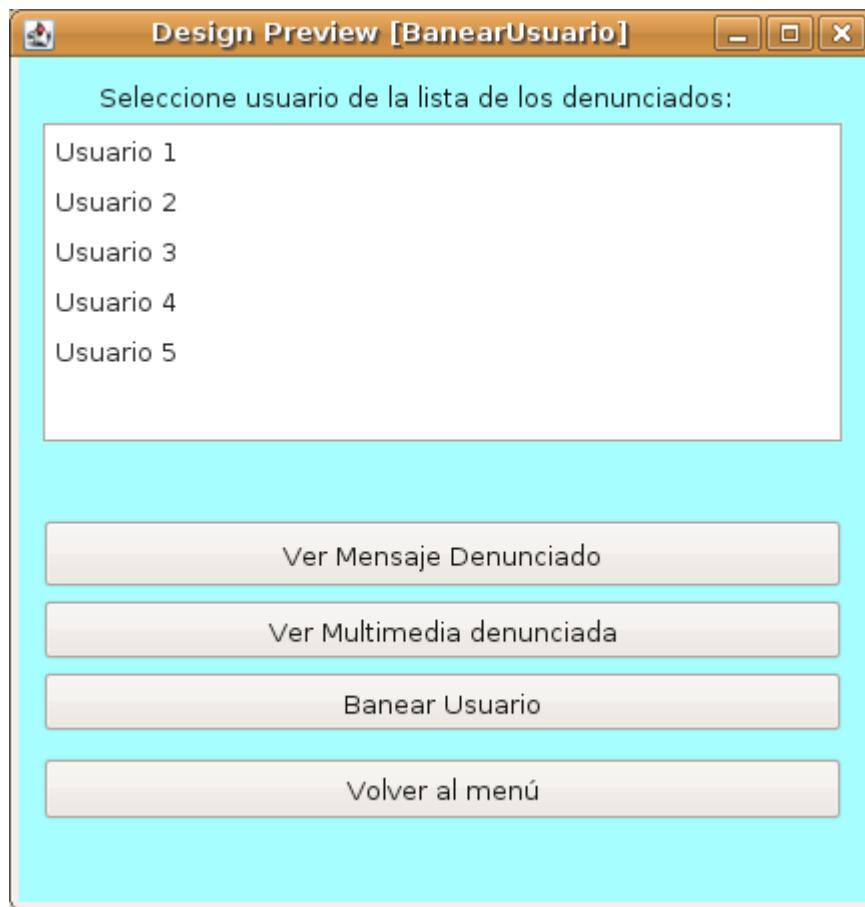
En la nueva ventana puede ver los datos marcados en rojo si no han podido ser satisfechos y permite seleccionar subcontrata, artista o local y muestra los datos de los mismos para poder contactar con ellos directamente.

Modifique los datos del evento, los campos en rojo no se han podido calcular con el P.A.R.E.:

Pase el ratón sobre los caninos para ver lo que demanda el cliente en cuanto a relleno.

-Banear usuario:

Muestra los usuarios que han sido denunciados y permite visualizar el mensaje o la multimedia.



-Ver eventos:

Sirve para visualizar eventos previamente creados.

Te permite acceder a la ventana de modificación de eventos por fuerza mayor.

Seleccione evento a visualizar:

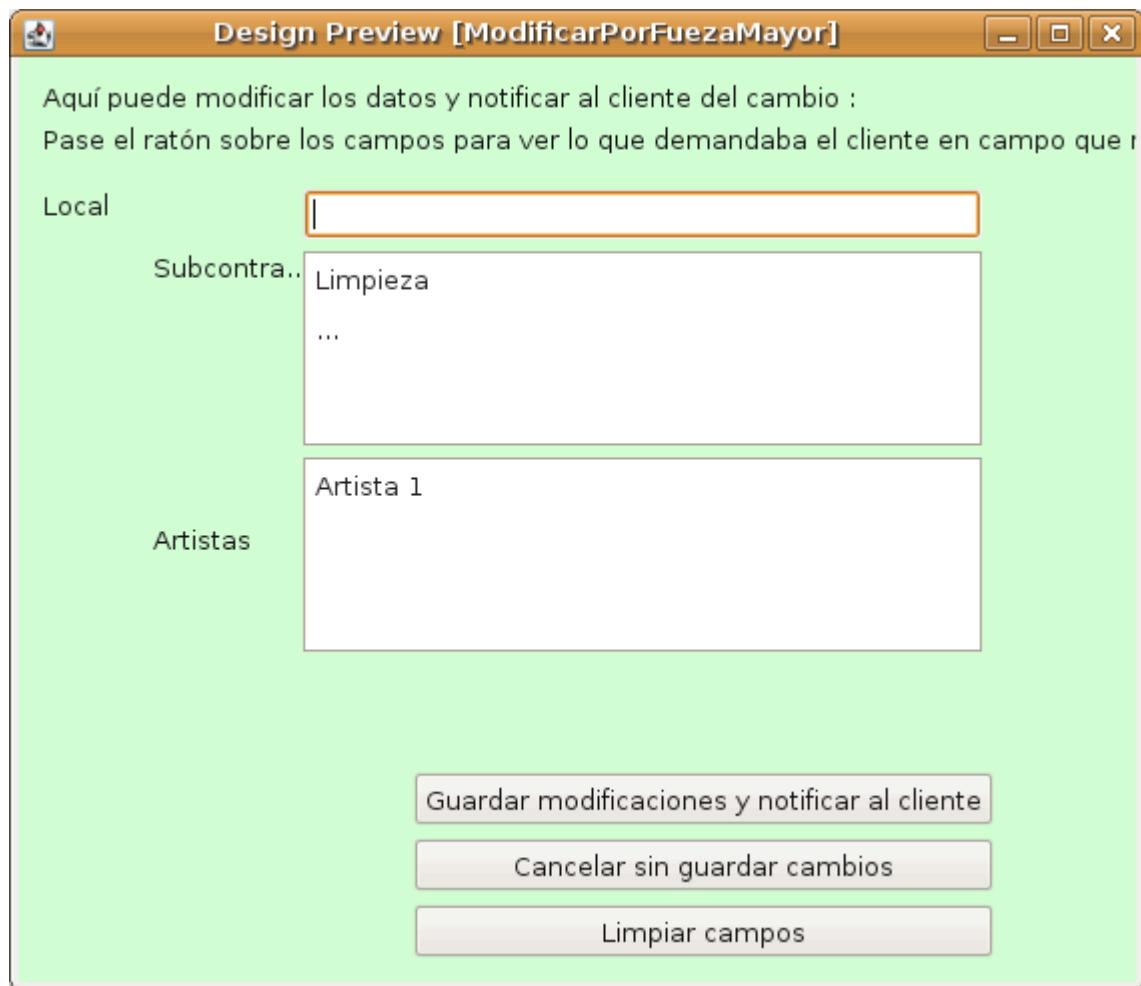
ID evento 1
ID evento 2
ID evento 3
ID evento 4
ID evento 5

visualice los datos del evento:

Público	<input type="checkbox"/>
Fecha	<input type="text"/>
Aforo Mín	<input type="text"/>
Aforo Máx	<input type="text"/>
Tipo evento	<input type="button" value="Tipo 1"/>
Duración (minutos)	<input type="text"/>
Zona	<input type="text"/>
Dirección	<input type="text"/>

Modificar eventos por fuerza mayor:

Permite editar sólo los cambios de local, subcontrata y artistas si se produjera alguna contingencia una vez pagado un evento.



La interfaz para ver mensajes y contenidos multimedia sería:

Design Preview [VerMensajesRedSocial]

En esta ventana podrá ver los mensajes que han sido denunciados o son públicos:

Seleccione Usuario remitente:

- Usuario 1
- Usuario 2
- Usuario 3
- Usuario 4
- Usuario 5

Fecha 1

Seleccione Usuario destinatario:

- Público(Todos)
- Usuario 2
- Usuario 3**
- Usuario 4
- Usuario 5

Cuerpo del mensaje

[Anterior mensaje del usuario](#) [Volver al menú](#) [Siguiente mensaje del usuario](#)

3.2.2. INTERFAZ JEFE:



El jefe puede anular un evento incondicionalmente y ver los movimientos de las cuentas.

Seleccione evento de la lista:

- Evento 1
- Evento 2
- Evento 3
- Evento 4
- Evento 5

El estado del evento es:

Design Preview [ManejarCuentas]

Cuenta: Cuenta 1

Movimientos:

Emisor

Receptor

Cantidad

Concepto

Fecha

3.2.3. INTERFAZ WEB DE USUARIO Y CLIENTE

Cuando una persona ajena a la empresa decide darse de alta en "ISParty" lo primero que deberá hacer es visitar nuestra página "www.ISParty.com".

Interfaz FormLogin

The screenshot shows a blue-themed login interface. At the top center is the 'ISParty' logo. Below it, there are two main sections: 'LOGIN' on the left and 'REGISTRO' on the right. The 'LOGIN' section contains input fields for 'USUARIO' and 'CONTRASEÑA', along with 'Limpiar' and 'Aceptar' buttons. The 'REGISTRO' section contains a single 'Regístrarme' button.

Una vez en ésta página, el internauta podrá introducir su nombre de usuario y su contraseña, si ya está dado de alta con anterioridad, o registrarse como nuevo usuario, en cuyo caso pasaría a una nueva pantalla llamada Registro ISParty.

Interfaz FormRegistro

The screenshot shows a registration form titled 'Registro ISParty'. On the left, there is a section for 'DATOS PERSONALES' with fields for 'USUARIO', 'CONTRASEÑA', 'NOMBRE', 'APELLIDO', 'DNI', 'DIRECCION', 'CIUDAD', and 'PAIS'. On the right, there is a section for 'AFICIONES' divided into categories: 'MUSICA' (with sub-options ROCK, CLASICA, HOUSE), 'DEPORTES' (with sub-options FUTBOL, BALONCESTO, TENIS), 'OCIO' (with sub-options CINE, TEATRO), and 'TEATRO' (with sub-options MUSEOS). At the bottom, there are several buttons: 'Limpiar', 'Registrarse solo como cliente de ISParty', 'Registro Cliente', 'Registrarse como usuario de la red social y como cliente', and 'Registro Usuario'.

Una vez en la pantalla Registro ISParty el usuario deberá introducir una serie de datos personales que la empresa guardará de forma confidencial. Lo más importante de esta página es que el usuario se puede registrar como **Cliente** o como **Usuario de la RS**.

Esta distinción está hecha ya que puede haber clientes que quieran organizar un evento y no les interese pertenecer a la red social, en tal caso, se darán de alta

como Cliente y contarán con una interfaz personal para crear eventos, como muestra la siguiente imagen:

Interfaz FormCrearEventoCliente

En esta página el cliente hace llegar a la empresa un formulario con las diferentes características que quiere para organizar su evento. En esta interfaz, también podrá ver los eventos que tiene pendientes y los que ha organizado. Para saber en qué estado está un evento sólo tiene que pinchar en el evento, en el cuadro de la derecha.

Por el contrario los usuarios que quieran tener todos los privilegios como crear eventos y pertenecer a la red social deberán registrarse como Usuarios. La interfaz de la cual disfrutarán esta clase de usuarios es la siguiente:

Interfaz FormPRS

El uso de esta página está pensado para que sea muy sencilla para el usuario. En el apartado de Mis Eventos salen los diferentes eventos a los que el usuario se ha apuntado, éstos, están ordenados por fecha, así mismo tendrán diferentes colores. El color verde quiere decir que el usuario es el creador del evento y por tanto tiene los derechos necesarios para hacer los cambios que desee, mientras tenga en consentimiento de la empresa y siga el protocolo de la misma. Los eventos que están en rojo son eventos a los que el usuario puede apuntarse pero que no puede modificar.

En el centro de la página disponemos de un lugar en el que se van promocionando los eventos nuevos que la gente va creando, así, con un simple clic el usuario podrá informarse sobre el nuevo evento y apuntarse.

Otra de las novedades es que disponemos de un espacio limitado para albergar los archivos multimedia que el usuario desee. La única restricción que tienen los usuarios en este campo, es que no pueden subir archivos que ocupen más de 10MB. La interfaz para esta acción es:

Interfaz FormSubirArchivo

Subir Archivos

Examinar

Aceptar

Cancelar

Recuerda que los archivos no pueden exceder del tamaño maximo que es de 10MB

Para la gestión de eventos tiene dos opciones que son Buscar Evento que le permite buscar eventos que estén confirmados por fecha, lugar, tipo de música....y Crear evento que le permite al usuario de la red social crear un evento nuevo a su gusto. Para la creación de un nuevo evento el usuario deberá pinchar en el enlace de crear evento y le aparecerá la siguiente interfaz que le permitirá crearlo de forma sencilla.

Interfaz FormCrearEventoUsuario

Crear Evento

Local	<input type="button" value="Sin Local"/>
Estilo	<input type="button" value="Sin Estilo"/>
Artista	<input type="button" value="Sin Artista"/>
Servicios Extra	<input type="button" value="Sin Servicios"/>
Tipo	<input type="button" value="Publico"/>
Aforo Max/Min	<input type="text"/> <input type="text"/>
Fecha Celebracion	<input type="text"/> (dd/mm/aaaa)
Duracion	<input type="text"/>
Voy a asistir al evento	<input type="checkbox"/>
Lista invitados	<input type="text"/> (Nombre,Apellido,DNI)
Datos Adicionales	<input type="text"/>
<input type="button" value="Cancelar"/> <input type="button" value="Enviar Solicitud"/>	

3.3. INVARIANTES DE LA INTERFAZ

FormLogin

Context FormLogin inv:

```
Self.Buttons->exist(b: button | b.Nombre="Aceptar");
Self.Buttons->exist(b: button | b.Nombre="Limpiar");
Self.Buttons->exist(b: button | b.Nombre="Registrarse");
Self.TextBox.Text=" ";
```

(self.Buttons ->select (n | n.Nombre="Aceptar").Activo == true) implies
 self.TextBoxs -> forAll(c:TextBox | c.Texto <> "")

Para que el botón Aceptar este activo implica que todos los campos de texto han de estar escritos

FormRegistro

Context FormRegistro inv:

(self.Buttons ->select (n | n.Nombre="RegistroCliente").Activo == true) implies
 self.TextBoxs -> forAll(c:TextBox | c.Texto <> "" and c.Nombre <> "MusicaOtros"
 and c.Nombre <> "DeportesOtros" and c.Nombre <> "OcioOtros")

Para que el botón RegistroCliente esté activo implica que todos los campos de texto han de estar escritos excepto los campos de textos "otros" de deportes, ocio, música que pueden estar vacíos.

(self.Buttons ->select (n | n.Nombre="RegistroUsuario").Activo == true) implies
 (self.CheckBoxs -> forAll(a | a.Activo = true)->size() > 0) and self.TextBoxs ->
 forAll(c:TextBox | c.Texto <> "" or (c.Nombre == "MusicaOtros" or c.Nombre ==
 "DeportesOtros" or c.Nombre == "OcioOtros"))-> size() > 0

Para que el botón RegistroUsuario esté activo implica que de todos los checkBoxes que hay por lo menos esté uno seleccionado o un campo de texto "Otros" y que todos los campos de texto estén llenos.

FormCrearEventoCliente

Context FormCrearEventoCliente inv:

(self.Buttons ->select (n | n.Nombre="EnviarSolicitud").Activo == true) implies
 self.ComboBoxs -> forAll(c:ComboBox | c.Seleccionado==1) and self.TextBoxes->
 forAll(a: TextBox | a.Texto <> " ")

Para que el botón "EnviarSolicitud" esté activo implica que todos los comboBoxes han de estar seleccionados y todos los textBoxes han de estar llenos.

Interfaz FormSubirArchivo

Context FormSubirArchivos inv:

(self.Buttons ->select (n | n.Nombre="Aceptar").Activo == true) implies
 self.TextBoxs -> forAll(c:TextBox | c.Texto <> "")

Para que el botón Aceptar esté activo todos los textBoxes han de estar llenos.

Interfaz FormCrearEventoUsuario

Context FormCrearEventoUsuario inv:

```
(self.Buttons ->select (n | n.Nombre="EnviarSolicitud").Activo == true) implies  
self.ComboBoxs -> forAll(c:ComboBox | c.Seleccionado == 1) and self.text:Boxes-  
>forAll(t: TextBox | t.Texto < > " ")
```

Para que el botón EnviarSolicitud esté activo requiere que todos los comboBoxes de la página estén seleccionados y que todos los textBoxes estén escritos.

4. INVESTIGACIÓN DE LOS COMPONENTES UTILIZADOS EN EL SISTEMA

4.1. PROPUESTA PARA EL COMPONENTE DE BASES DE DATOS



Consideramos para éste componente distanciarnos un poco de lo establecido con MySQL y quedarnos con la última versión estable de Oracle, la **Database 11g Enterprise Edition**, que es la más completa y data del año 2007.

Oracle tiene el récord mundial en rendimiento y en la relación precio/rendimiento, lo que viene bien para implementar nuestro sistema. Lo podemos ver en ésta página:

http://www.oracle.com/solutions/performance_scalability/tpc-c-11g-windows.html

La partición y la compresión de los datos para ejecutar consultas más rápidas con menos discos puede ser útil para rentabilizar recursos, y es uno de los aspectos más destacables de ésta última versión.

Por otro lado, algunas de las *características más importantes* que encontramos son:

- Fácil de instalar en sistemas operativos Windows y Linux, fácil de manejar gracias a la capacidad de gestión automatizada y fácilmente escalable.
- Proporciona compatibilidad ascendente completa con otras ediciones de la base de datos está optimizada para el despliegue rápido en empresas de rápido crecimiento. Está disponible para servidores de hasta máximo 2 sockets y puede ser instalado en todos los sistemas operativos soportados por Oracle incluidos Windows, Linux y Unix.
- Ofrece una rápida instalación en entornos de un sólo servidor. Es una solución out-of-the-box, la base de datos viene pre-configurada para ambientes de producción, con gestión automatizada de espacio, memoria, respaldos y recuperación y optimizador automático de gestión de estadísticas.
- Adicionalmente la consola de administración Oracle Enterprise Manager Database Control proporciona una interfaz web que muestra el estado actual de la base de datos y del ambiente de cluster, lo que permite realizar acciones de administración de bases de datos desde cualquier navegador.
- Soporta todos los estándares en cuanto al manejo de tipos de datos, así como también en cuanto a almacenamiento nativo de XML, texto, documentos, imágenes, audio, vídeo y datos de ubicación.

- También se incluyen características analíticas, estadísticas y capacidades de modelamiento que pueden ser utilizadas por cualquier ambiente de inteligencia de negocios basado en SQL.
- Utiliza las mismas técnicas probadas de concurrencia que las otras ediciones de la base de datos Oracle utilizan, asegurando el soporte de la base de datos a cargas de trabajo.
- La base de datos provee de mirroring y características de respaldo y recuperación que permiten la protección de los datos de las causas más comunes de pérdida, sin la necesidad de soluciones costosas de almacenamiento.
- Provee de características de Flashback Query que permiten ver y recuperar las versiones antiguas de datos sin tener que realizar actividades complejas y que consumen mucho tiempo en operaciones de recuperación.
- Soporte robusto para el manejo de roles de base de datos y funciones de auditoría, lo que constituye un mecanismo fuerte de control de acceso que garantiza su seguridad y privacidad.

Podemos ver más documentación aquí:

<http://www.oracle.com/pls/db111/homepage>

Por último, citar que las versiones de Oracle son de descarga gratuita para desarrollo y se paga licencia por su uso comercial.

Para éste caso particular, la Enterprise Edition, tenemos los siguientes precios en USD:

Named User Plus: 950
 Software Update, License & Support: 209.00

 Processor License: 47,500
 Software Update, License & Support: 10,450.00

Podemos consultar más precios en éste manual oficial:

<http://www.oracle.com/corporate/pricing/technology-price-list.pdf>

La página de descarga oficial es la siguiente:

<http://www.oracle.com/technology/software/products/database/index.html>

4.2. PROPUESTA PARA EL COMPONENTE DE PAGOS



PayPal es un sistema que permite a cualquier persona que tenga una dirección de e-mail enviar o recibir dinero online utilizando su tarjeta de crédito de manera totalmente segura.

Hemos pensado que sería una forma amena y sencilla de incorporar pagos online para nuestra aplicación, ya que dispone de herramientas guiadas y sencillas para la integración del paquete en la aplicación web. Además, es gratuita.

Aquí podemos ver una guía de integración:

https://www.paypalobjects.com/WEBSCR-570-20090507-1/es_ES/pdf/PP_WebsitePaymentsStandard_IntegrationGuide.pdf

En función de las necesidades, de si dispones o no de desarrollador, o cómo vayas a enfocar los pagos, tienes diferentes métodos de integración.

Podemos ver información para Empresas en esta URL:

<https://www.paypal-promo.es/empresas/>

Y por último, aquí podemos ver las tarifas nacionales e internacionales para transacciones, divisas, etc.:

https://www.paypal.com/es/cgi-bin/webscr?cmd=_display-pop-fees-outside

4.3. PROPUESTA PARA EL COMPONENTE MULTIMEDIA



A la hora de que se realicen las gestiones de los diferentes archivos multimedia, hemos querido diferenciar dos aspectos que hoy en día son de los más empleados. Y muy funcionales desde el punto de vista del que sube y del que visualiza o descarga.

A la hora de visualización o carga de vídeos, es muy útil aprovechar la interfaz de *Youtube*. Es posible incorporar a la aplicación web el banner de *Youtube*, o incluso añadiendo el código *Embed* correspondiente, tener ese mismo vídeo en tu web.

Así, de manera gratuita, y sin ocuparnos almacenamiento, podemos gestionar estos archivos.

http://www.youtube.com/t/yt_handbook_home

Por otro lado, para el resto de archivos, hemos dudado entre dos aplicaciones. Una ha sido *Dropbox*:

Dropbox es una aplicación que sincroniza tus ordenadores, de forma que todos los archivos que guardes en una carpeta concreta de tu ordenador A se subirán a Internet y se descargarán en la misma carpeta de tu ordenador B, C, D...

Y por otro lado, *Rapidshare*, un sistema sencillo y accesible de almacenaje y distribución de archivos a través de Internet.

Ambos tienen dos tipos de cuentas según sean de pago o no, pero ante la duda hemos preferido quedarnos con *Rapidshare* debido a que no implica la instalación de software adicional, para el usuario medio es más intuitivo de usar, y creemos que se decantaría por esta opción.

Las características y precios son:

Features	Free	Premium
Max. file size for Uploads	200 MB	2.000 MB ¹
Personal webspace	-	500 GB
Inclusive download traffic	-	150 GB per month ²
Deletion of files	After 90 days without Download	Never ³
Instant start of Download	No	Yes
Download speed	Limited ⁴	Unlimited
Max. parallel Downloads	1	Unlimited
Support of Download-Accelerator	No	Yes
Resume of broken Downloads	No	Yes

Y según los días de los que deseemos disponer de la cuenta *Premium*, tendremos los precios que podemos ver aquí:

<http://www.rapidshare.com/premium.html>

Por último, nos da la opción de extender la cuenta *Premium* aquí:

<https://ssl.rapidshare.com/premiumzone.html>

Así, para usuarios no habituales de nuestra Red, les valdría una cuenta gratuita, y para casos excepcionales, se habilitaría la cuenta *Premium* para que cada tipo de cliente tuviera una solución específica.

5. PRUEBAS

5.1. PRUEBAS USUARIO/CLIENTE:

1. Registrarse

- 1.1. intentar registrarse como usuario o como cliente dejando los campos relevantes sin llenar
- 1.2. intentar poner menor o mayor número de dígitos que el establecido(para dni y teléfono)
- 1.3. intentar poner letras en lugar de números y viceversa
- 1.4. dejar en blanco las aficiones
- 1.5. intentar registrarse como usuario haciendo los pasos 1.2, ó 1.3, ó 1.4.
- 1.6. intentar registrarse como cliente haciendo los pasos 1.2, ó 1.3, ó 1.4.
- 1.7. comprobar que no se ha registrado
- 1.8. introducir correctamente los datos
- 1.9. comprobar que está registrado

2. Loguearse

- 2.1. introducir un usuario y una contraseña inventados
- 2.2. introducir el usuario correcto y la contraseña incorrecta o viceversa
- 2.3. comprobar que no puedes acceder a la red social con los pasos 2.1, ó 2.2
- 2.4. introducir datos correctos
- 2.5. comprobar que estamos dentro de la red social

3. Eventos

- Solicitar evento:

- 3.1. intentar no llenar ningún campo.
- 3.2. Intentar no llenar los campos relevantes (fecha del evento, duración, localidad y tipo)
- 3.3. intentar llenar de forma incorrecta, al menos, uno de los campos.
- 3.4. comprobar que no se puede enviar la solicitud de evento haciendo los pasos 3.1, ó 3.2, ó 3.3.

- Modificar evento:

- 3.5. seleccionar un evento no creado por el usuario
- 3.6. intentar anular ese evento
- 3.7. comprobar que no deja modificar ese evento haciendo los pasos 3.5 y 3.6

- Anular evento: análogo a modificar evento

- Apuntarse a evento:

- 3.8. seleccionar un evento
- 3.9. apuntarse a ese evento en el cual la fecha límite haya vencido
- 3.10. comprobar que el usuario no está apuntado a ese evento
- 3.11. apuntarse a un evento dentro de la fecha límite
- 3.12. volverse a apuntar a ese evento
- 3.13. comprobar que no deja apuntarse a ese evento haciendo los pasos 3.8, 3.9 y 3.10

- Desapuntarse de evento:

- 3.14. seleccionar un evento
- 3.15. desapuntarse de ese evento en el cual la fecha límite haya vencido
- 3.16. comprobar que no se ha desapuntado el usuario
- 3.17. seleccionar un evento
- 3.18. desapuntarse de ese evento en el cual la fecha límite no haya vencido
- 3.19. desapuntarse de ese evento de nuevo
- 3.20. comprobar que no deja desapuntarse por segunda vez

3.21. desapuntarse de un evento al que no está apuntado es igual que el
3.19

4. Archivos multimedia

- Subir archivo multimedia:

- 4.1. intentar subir un archivo dejando en blanco el url
- 4.2. comprobar que no sube ningún archivo
- 4.3. intentar introducir un url incorrecto
- 4.4. comprobar que no sube ningún archivo
- 4.5. intentar subir una archivo que ocupe más que el tamaño máximo
- 4.6. comprobar que no sube ese archivo
- 4.7. introducir un url correcto
- 4.8. volver a introducir el mismo url
- 4.9. comprobar que no deja

intentar subir un archivo multimedia al tablón de otro usuario

- 4.10. comprobar que no se ha subido el archivo
- 4.11. introducir el url correcta
- 4.12. comprobar que se sube el archivo

- Descargar archivo multimedia:

- 4.13. descargar cualquier archivo multimedia
- 4.14. comprobar que se descarga el archivo multimedia

- Eliminar archivo multimedia:

- 4.15. intentar eliminar un archivo del tablón de otro usuario
- 4.16. comprobar que no se elimina el archivo
- 4.17. intentar eliminar un archivo del tablón del propio usuario
- 4.18. comprobar que se elimina el archivo

5. Mensajes

Los mensajes se mandan desde la página del usuario que va a ser el destinatario, allí se elige si va a ser mensaje de foro o privado, por tanto, no se permiten varios destinatarios

- Escribir mensajes:

- 5.1. intentar enviar un mensaje privado sin destinatario
- 5.2. comprobar que no deja
- 5.3. intentar enviar un mensaje de foro sin destinatario
- 5.4. comprobar que no deja
- 5.5. intentar enviar un mensaje desde mi cuenta a otro usuario
- 5.6. comprobar que no está permitido
- 5.7. intentar enviarme un mensaje privado a mí mismo
- 5.8. comprobar que no deja
- 5.9. intentar enviarme un mensaje de foro a mí mismo
- 5.10. comprobar que no deja
- 5.11. intentar enviar un mensaje desde la página de otro usuario a otro usuario diferente
- 5.12. comprobar que no se permite
- 5.13. intentar enviar un mensaje desde la página de otro usuario a ese usuario sin seleccionar si es mensaje de foro o privado
- 5.14. comprobar que no se envía el mensaje
- 5.15. intentar enviar un mensaje desde la página de otro usuario a ese usuario seleccionando o que es de foro, o que es privado y que supere el tamaño máximo
- 5.16. comprobar que no se envía el mensaje
- 5.17. intentar enviar un mensaje desde la página de otro usuario a ese usuario seleccionando o que es de foro, o que es privado
- 5.18. comprobar que se ha enviado el mensaje
- 5.19. intentar enviar un mensaje desde la página de otro usuario a ese usuario seleccionando o que es de foro, o que es privado y que tenga el texto en blanco
- 5.20. comprobar que se ha enviado el mensaje

- Leer mensajes:

- 5.21. intentar leer un mensaje privado de otro usuario
- 5.22. comprobar que no se puede leer
- 5.23. intentar leer los mensajes privados que he escrito a otros usuarios
- 5.24. comprobar que se pueden leer
- 5.25. intentar leer los mensajes privados que me han mandado otros usuarios
- 5.26. comprobar que se pueden leer
- 5.27. intentar leer un mensaje de foro de otro usuario
- 5.28. comprobar que se puede leer

6. Invitar a la red social

- 6.1. intentar meter una dirección de correo electrónico sin @, o sin punto, o sin los dos.
- 6.2. comprobar que no se envía la invitación
- 6.3. intentar meter una dirección de correo electrónico correcta
- 6.4. comprobar que se envía la invitación

7. Modificar perfil

- 7.1. intentar modificar los datos relevantes
- 7.2. comprobar que no se permite modificar esos datos
- 7.3. intentar modificar el perfil de otro usuario
- 7.4. comprobar que no se puede modificar
- 7.5. intentar modificar datos no relevantes
- 7.6. comprobar que se ha modificado el perfil del usuario

8. Denunciar contenido

- 8.1. intentar denunciar un archivo o mensaje dejando en blanco el motivo
- 8.2. comprobar que no se envía la denuncia
- 8.3. intentar denunciar un archivo o mensaje explicando el motivo
- 8.4. comprobar que se ha enviado la denuncia

9. Votar artista

- 9.1. intentar votar a un artista antes de que se haya acabado el evento
- 9.2. comprobar que no está permitido
- 9.3. intentar votar a un artista sin estar apuntado a ese evento
- 9.4. comprobar que no se puede
- 9.5. intentar votar a un artista una vez pasado el evento
- 9.6. volver a votar a ese artista
- 9.7. comprobar que no permite votar a un artista dos veces
- 9.8. intentar votar a un artista una vez pasado el evento
- 9.9. comprobar que se ha votado al artista

5.2. PRUEBAS DE EMPLEADO

1. Loguearse

- 1.1. introducir un usuario y una contraseña inventados
- 1.2. introducir el usuario correcto y la contraseña incorrecta o viceversa
- 1.3. comprobar que no puedes acceder a la aplicación con los pasos 2.1, ó 2.2
- 1.4. introducir datos correctos
- 1.5. comprobar que estamos dentro de la aplicación

2. Gestionar

-Artista, Local y Subcontrata: (recursos)

- 2.1. intentar llenar de forma incorrecta el formulario de añadir un recurso
- 2.2. comprobar que no deja
- 2.3. rellenarlo de forma correcta
- 2.4. comprobar que deja
- 2.5. intentar modificar los datos relevantes del recurso, por ejemplo dni/nif
- 2.6. comprobar que no se puede

- 2.7. modificar los datos permitidos de forma correcta
- 2.8. comprobar que deja
- 2.9. eliminar un recurso que está contratado para un evento
- 2.10. comprobar que no deja
- 2.11. eliminar recurso en el resto de los casos
- 2.12. comprobar que deja

3. Evento

-Respuesta a solicitud:

- 3.1. intentar poner en el desglose artista/artistas sin respetar la solicitud del cliente. Por ejemplo, poner un artista con distinto tipo al solicitado
- 3.2. intentar poner en el desglose local/locales sin respetar la solicitud del cliente. Por ejemplo, poner un local con distinta zona a la solicitada
- 3.3. comprobar que no se puede haciendo los pasos 4.1 o/y 4.2
- 3.4. poner correctamente los artistas, locales
- 3.5. poner las subcontratas solicitadas y las que se crean oportunas
- 3.6. comprobar que se puede con el paso 4.4 y 4.5

-Respuesta a modificación:

Análogo a Respuesta a solicitud, pero para poder modificar, el cliente tiene que haber pagado la señal

-Respuesta a cancelación de evento:

- 3.7. si no ha recibido petición de anulación intentar anularlo
- 3.8. comprobar que no deja
- 3.9. si la ha recibido y el evento no tiene la señal pagada
- 3.10. comprobar que deja

-Promocionar evento:

- 3.11. intentar promocionar un evento privado
- 3.12. comprobar que no deja
- 3.13. promocionar un evento público con o sin lista de invitados
- 3.14. comprobar que deja
- 3.15. volver a promocionar ese evento
- 3.16. comprobar que no deja

4. Mensajes

-Leer mensajes:

- 4.1. ver un mensaje en el tablón de un usuario
- 4.2. comprobar que se puede

5. Denunciar contenido

- 5.1. denunciar un mensaje privado de un usuario
- 5.2. comprobar que no se puede
- 5.3. denunciar un mensaje de foro o archivo multimedia de un usuario
- 5.4. comprobar que se puede
- 5.5. consultar las denuncias
- 5.6. comprobar que se puede

6. Banear un usuario

- 6.1. intentar banear un usuario sin ninguna denuncia
- 6.2. comprobar que no deja
- 6.3. banear un usuario con al menos una denuncia
- 6.4. comprobar que deja

5.3. PRUEBAS DE JEFE:

Todo análogo al empleado salvo:

-Loguearse: Con nick y contraseña de jefe

-Evento:

-Petición de cancelación de evento:

si no ha recibido petición de anulación intentar anularlo
comprobar que permite anular el evento

6. IMPLEMENTACIÓN

6.1. LENGUAJE UTILIZADO



Para realizar la implementación de nuestra aplicación hemos pensado en el lenguaje *Java*.

Más concretamente, y al ser un proyecto de código abierto, nos hemos decidido por la plataforma de desarrollo *Netbeans*, de manera que su uso es gratuito.

Además de poder realizar la *GUI* de manera amena, dispone de numerosos Add-ons como el de **PHP** lo cual nos facilitará la realización de nuestra aplicación Web.

Todo ello combinado con el gestor de Bases de Datos anteriormente mencionado en el apartado 4.1. , creemos que nos puede dar la cobertura software que necesitamos para un proyecto de esta envergadura.