10. Aplicación de patrones

10.1. Patrón de arquitectura Modelo-Vista-Controlador

El objetivo de usar un patrón de arquitectura de software Modelo-Vista-Controlador es el de separar la lógica de aplicación de la lógica de la vista de una aplicación. Aplicando este patrón se consigue mayor reutilización del código ya que el dominio queda totalmente aislado de la interfaz. Esto significa, que aunque se hagan cambios en la interfaz, en ningún momento afectará a la lógica de aplicación ya que los datos estarán separados completamente de la interfaz.

Por tanto, hemos aplicado el patrón definiendo en nuestro diagrama de clases:

- **Modelo**: se encarga de la lógica de aplicación generalmente consultando la base de datos manipulando, gestionando y actualizando los datos. El modelo no tendrá ningún tipo de acoplamiento ni visibilidad directa con la vista.

  En nuestro diagrama de clase tenemos el modelo dividido en varias clases de dominio:

  - Categoría
  - Liga
  - Jornada
  - Partido
  - Acta
  - Jugador_Acta
  - Jugador
  - SancionAutomatica
  - SancionPorFedFutbol
  - Equipo
  - Estadio
  - SancionEquipo
  - Arbitro
  - Ausencia
  - SancionArbitro
  - Arbitro_Liga
  - RepresentanteFedFutbol
  - PresidenteFederacionArbitros
  - Sede
  - Empleado
  - Usuario

- **Vista**: se encarga de mostrarle al usuario el resultado de una solicitud de forma visual (con pantallas, ventanas, páginas...) los datos al usuario.

  En nuestro diagrama de clases tenemos las siguientes vistas:

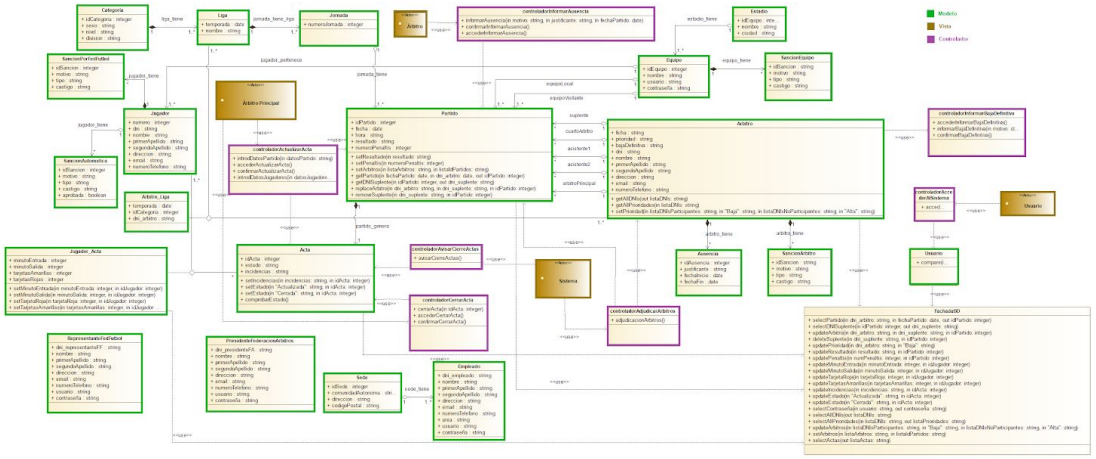
  - Árbitro
  - Árbitro Principal
  - Sistema
  - Usuario: representa todos los posibles usuarios que pueden conectarse al sistema.

- **Controlador:** se encarga de gestionar las instrucciones que se reciben de un usuario, atenderlas y procesarlas. Es el componente intermedio entre el modelo y la vista. El controlador solicita los datos necesarios al modelo manipulándolos para obtener los resultados deseados y entregándolos a la vista para que pueda mostrarlos visualmente.

  En nuestro diagrama de clases tenemos los siguientes controladores:

  - controladorAccederAlSistema
  - controladorAdjudicarArbitros
  - controladorActualizarActa
  - controladorCerrarActa
  - controladorAvisarCierreActas
  - controladorInformarAusencia
  - controladorInformarBajaDefinitiva

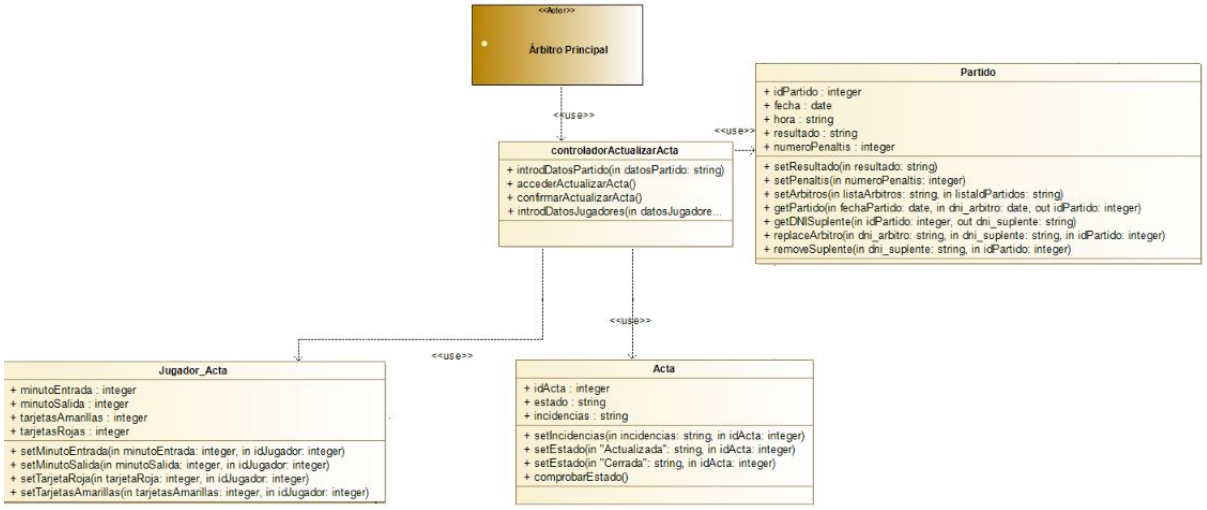
En la siguiente imagen se puede ver cómo se ha aplicado el patrón Modelo-Vista-Controlador.



Un ejemplo más visual de la aplicación del patrón Modelo-Vista-Controlador en nuestro diagrama de clases es el siguiente.

En este ejemplo Árbitro Principal es la vista, controladorActualizarActa es el controlador y la clase Acta, Jugador_Acta y Partido es el modelo.

Como se puede observar, el controlador actúa de intermediario entre el modelo y la vista. El controlador, tendrá en este caso que responder a un evento invocado por un usuario Árbitro Principal por lo que hará una petición al modelo para que le ofrezca la información necesaria para responder a la solicitud recibida. Acta, Partido y Jugador_Acta que conforman el modelo tendrán la información que el controlador necesita proporcionar a la vista. Una vez que el modelo transfiera los datos al controlador, el controlador enviará a la vista la respuesta para que esta finalmente se la muestre gráficamente al usuario.



10.2. Patrones de diseño

10.2.1. Experto

El patrón de diseño Experto sirve para asignar la responsabilidad de una clase a la clase que cuenta con información para cumplir la responsabilidad. Con esto se consigue una gran cohesión y un nivel de acoplamiento bajo.

El patrón Experto se ve reflejado en el diagrama de clases y los diagramas de colaboración. Estos son algunos ejemplos de uso del patrón Experto:

- **INFORMAR DE UNA AUSENCIA**

En el caso de uso **informar de una ausencia**, la clase partido aplica el patrón experto ya que posee toda la información necesaria para poder cumplir la responsabilidad del caso de uso, debido a que necesita saber la fecha del partido, el identificador del partido, recuperar el DNI del árbitro suplente y borrar el árbitro suplente, y esta información está guardada como atributos o son métodos de la clase partido.

Diagrama de colaboración

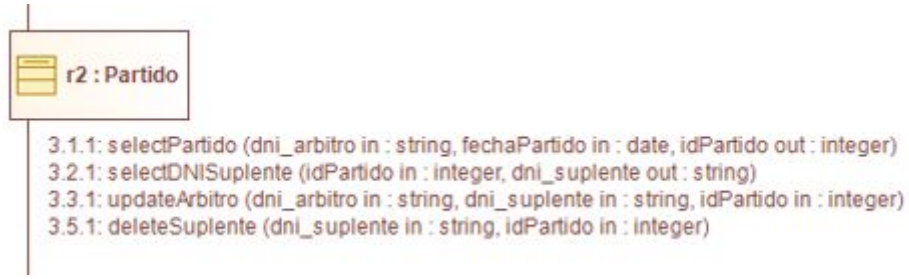
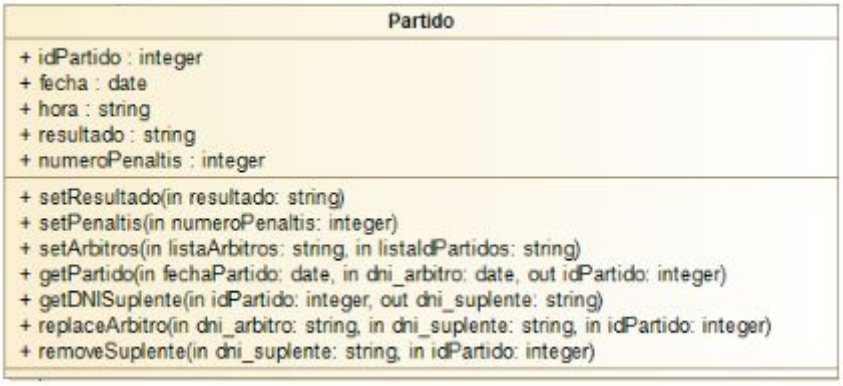


Diagrama de clases

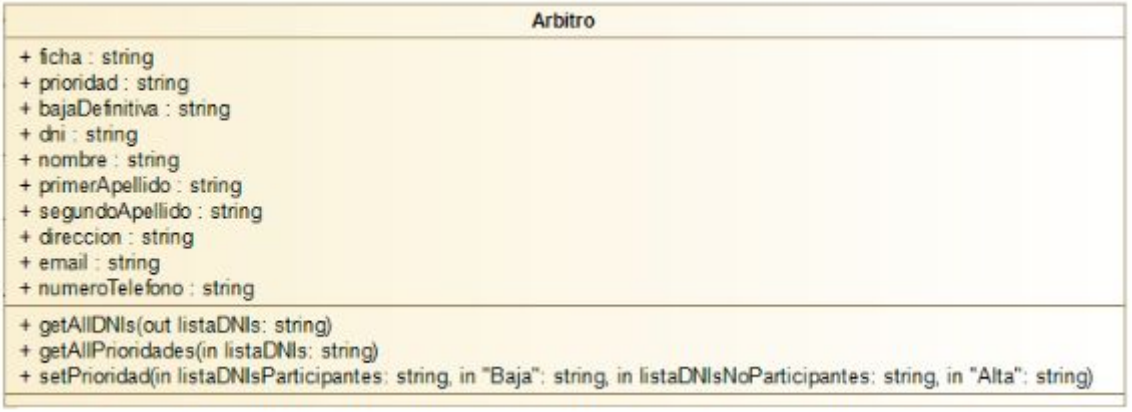


-ADJUDICAR ÁRBITROS A PARTIDOS

En el caso de uso **adjudicar árbitros a partidos**, la clase árbitro aplica el patrón experto ya que contiene toda la información necesaria para poder cumplir la responsabilidad del caso de uso, debido a que necesita saber los DNIs y las prioridades de todos los árbitros, y esta información está guardada como atributos de la clase árbitro.

Diagrama de colaboración

Diagrama de clases

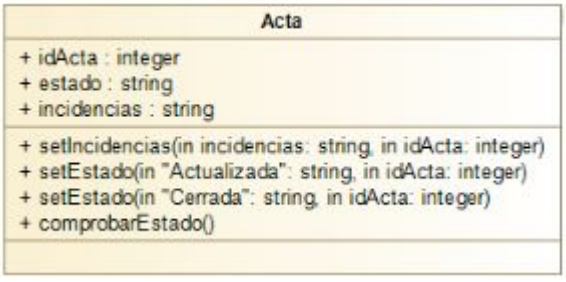


- **AVISAR DEL INMINENTE CIERRE DE LAS ACTAS**

En el caso de uso **avisar del inminente cierre de las actas**, la clase acta aplica el patrón experto ya que contiene toda la información necesaria para poder cumplir la responsabilidad del caso de uso, debido a que necesita saber el estado del acta y esta información está guardada como atributo de la clase acta.

Diagrama de colaboración

Diagrama de clases



10.2.2. Controlador

El patrón Controlador sirve para tener un responsable de manejar eventos del sistema generados por un actor. El patrón Controlador consigue que haya bajo acoplamiento al haber independencia entre las clases de dominio e interfaz y consigue además gran cohesión ya que los controladores solo se encargan de gestionar eventos.

En nuestro diagrama de clases hemos aplicado el patrón Controlador en varios casos. A continuación se mostrarán algunos ejemplos con su explicación e imagen adjunta.

ControladorAdjudicarArbitros

El actor “Sistema” genera un evento en el sistema con la intención de adjudicar árbitros a partidos; para ello, se contará con el controlador “controladorAdjudicarArbitros”. Este controlador decidirá qué clases de dominio deben intervenir para responder al evento producido. En este caso, el controlador decide que las clases “Arbitro” y “Partido” deben intervenir ya que se necesita información acerca de los árbitros registrados en el sistema y los partidos disponibles en la temporada.

Diagrama de clases

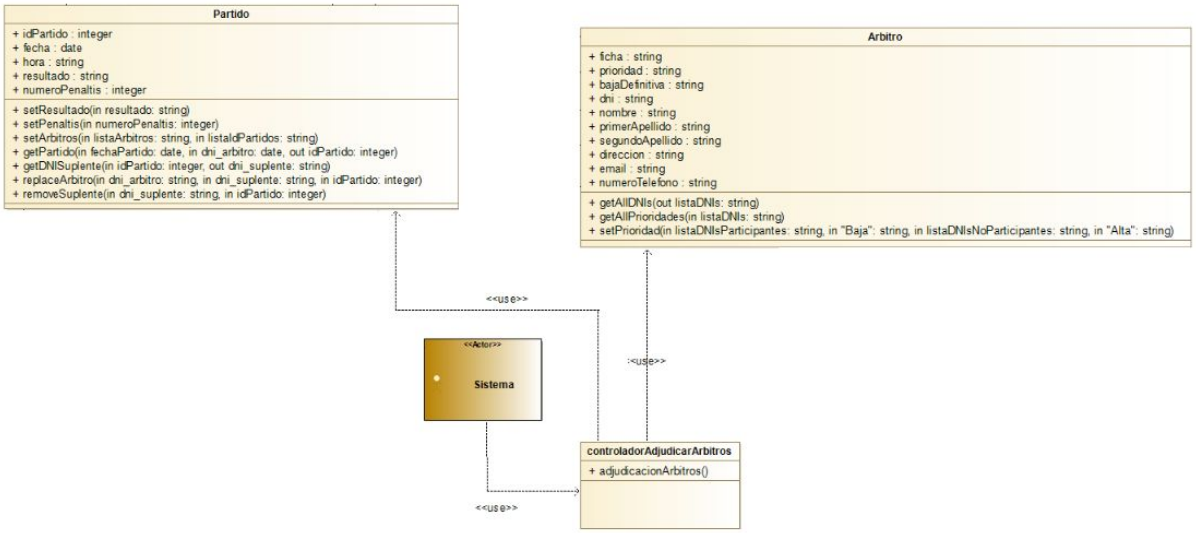
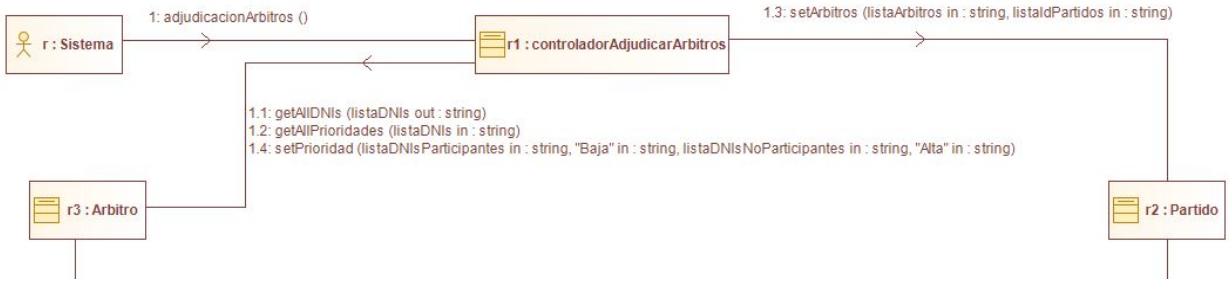


Diagrama de colaboración



ControladorActualizarActa

En este caso, el actor “Árbitro Principal” genera un evento en el sistema ya que quiere actualizar un acta, para ello se ha aplicado el controlador “controladorActualizarActa”. Este controlador manejará todos los eventos para este caso.

El controlador por tanto decide que las clases de dominio que son necesarias que intervengan para la petición del usuario son “Partido” y “Acta”. Esto es debido a que para actualizar un acta es necesario conocer e introducir los datos de un partido y también es necesario conocer la información de ese acta.

Diagrama de clases

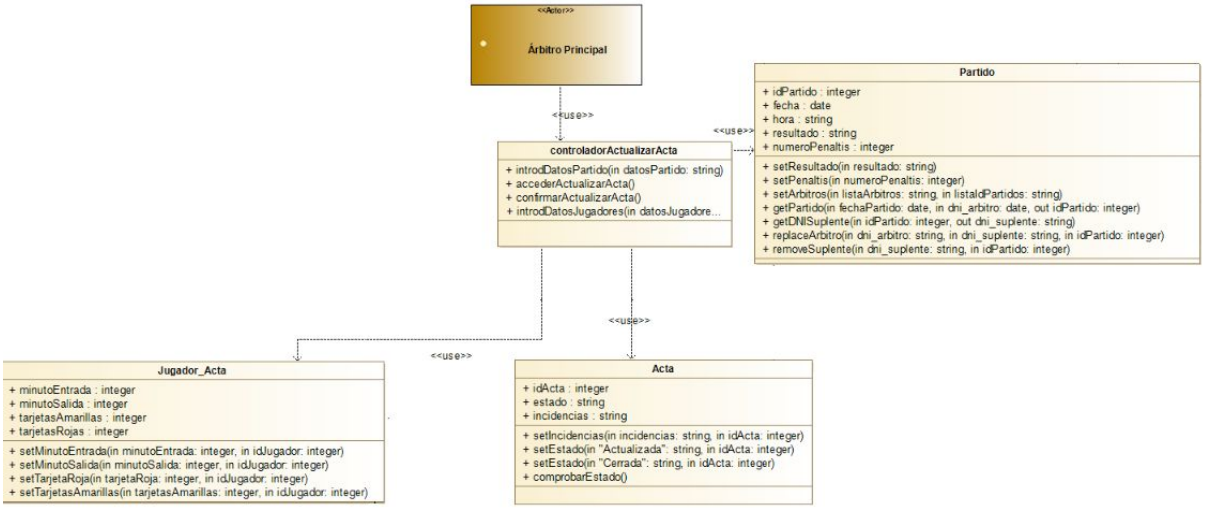
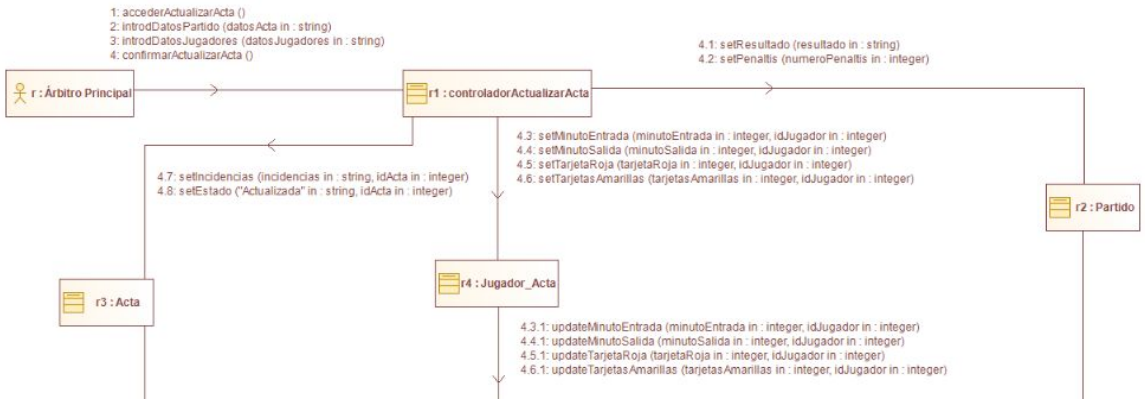


Diagrama de colaboración



ControladorCerrarActa

En este caso, el actor “Árbitro Principal” genera un evento en el sistema con la intención de cerrar un acta y para ello, se contará con el controlador “controladorCerrarActa”. El controlador por tanto, decidirá qué clases de dominio deben intervenir para responder al evento producido. En este caso, el controlador decide que la única clase que debe intervenir es “Acta” ya que se necesita información acerca del acta que va a ser cerrada.

Diagrama de clases

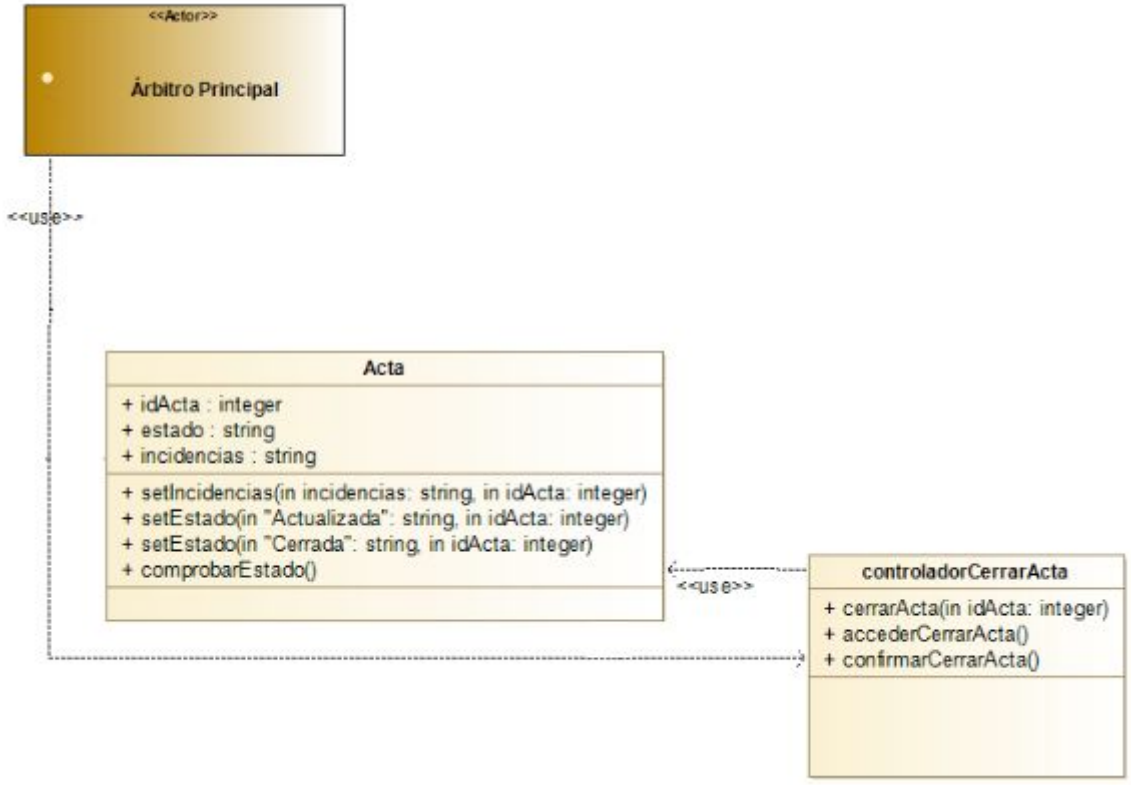
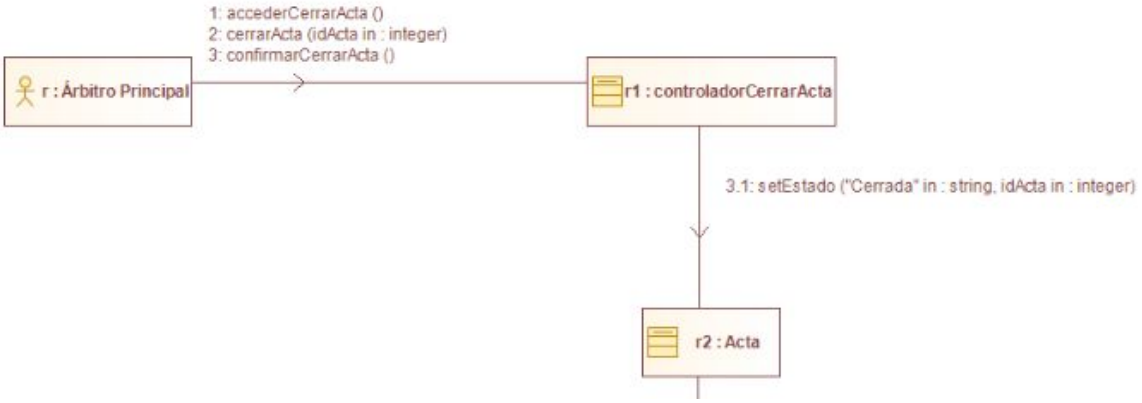


Diagrama de colaboración



ControladorInformarAusencia

En este caso, el actor “Árbitro” genera un evento en el sistema con el objetivo de informar de una ausencia, para ello se ha aplicado el patrón controlador “controladorInformarAusencia”.

El controlador por tanto decide que la clase de dominio que es necesaria que intervenga para resolver la petición del usuario es la clase de dominio “Partido”. Esto es debido a que para informar de una ausencia es necesario que el árbitro informe de qué partido en particular va a ausentarse.

Diagrama de clases

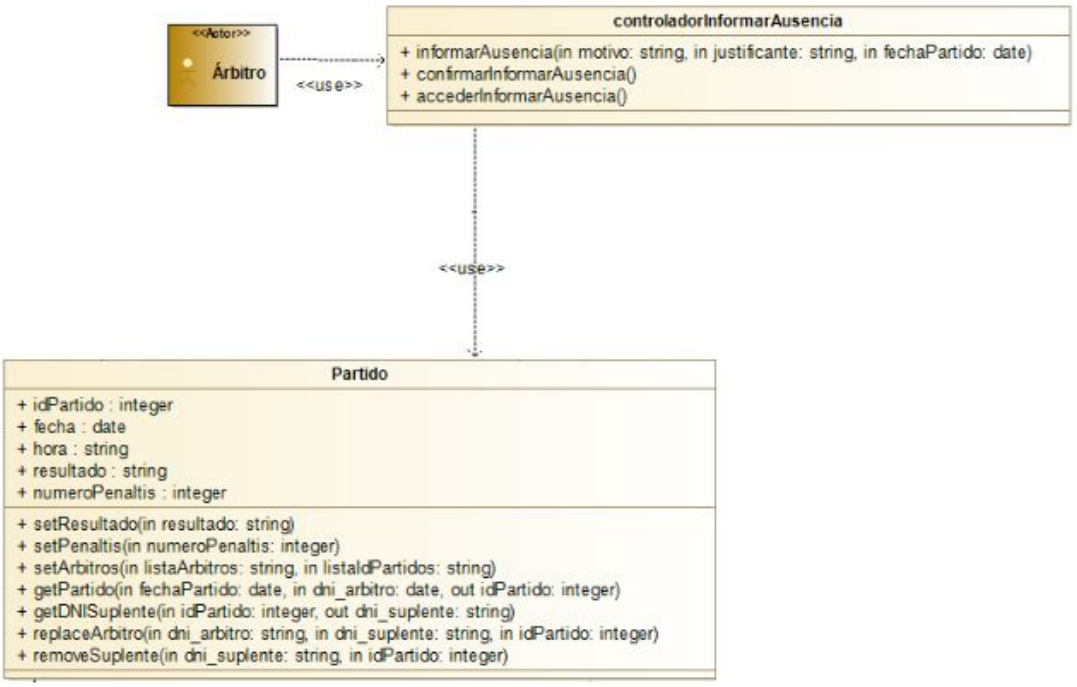
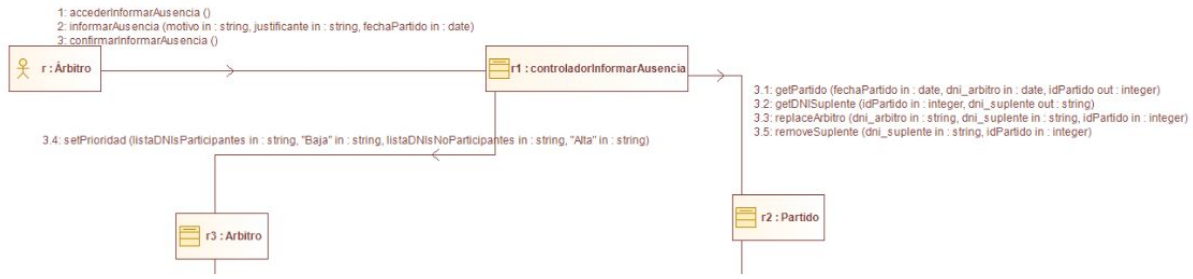


Diagrama de colaboración



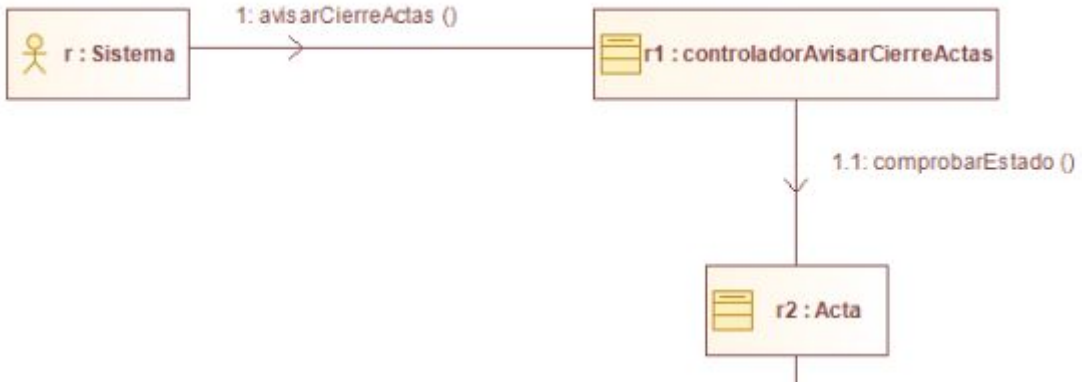
ControladorAvisarCierreActas

En este caso, el actor “Sistema” genera un evento con la intención de avisar de que un acta lleva 23 horas en estado “actualizada” y que aún no se ha cerrado. Para manejar este evento se ha aplicado el controlador “controladorAvisarCierreActas”.

El controlador por tanto decide que la clase de dominio que es necesaria que intervenga para resolver la petición del usuario es la clase de dominio “Acta” ya que es necesario recuperar la información del acta que tras 23 horas finalizado su partido sigue en estado “actualizada”.

Diagrama de clases

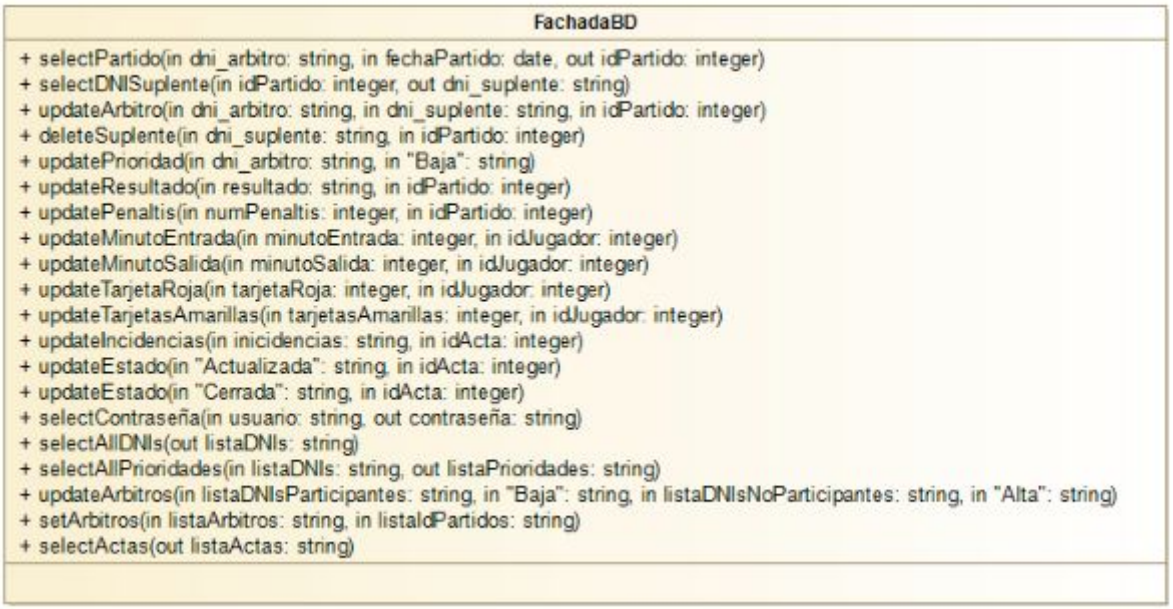
Diagrama de colaboración



10.2.3. Fachada

El patrón de diseño Fachada sirve para tener una interfaz unificada y común para un conjunto de interfaces. La Fachada mantiene un control sobre cada uno de los datos y acciones que proporciona el sistema. Además, permite a cualquiera de los usuarios autorizados que accedan al sistema poder insertar, modificar, recuperar, eliminar y gestionar datos de una manera mucho más rápida y sencilla mediante la creación de una interfaz específica para esa modificación; véase, por ejemplo, que un empleado desea poder editar información sobre “Árbitros”. Se crea entonces una interfaz que permita acceder exclusivamente a esa funcionalidad. Esto es una gran ventaja, ya que en el caso de aquellos usuarios que únicamente deseen consultar datos, permanecerán ajenos a todas las clases que haya dentro de esa interfaz y será ésta la que se encargue de colaborar con el resto de clases y paquetes para llevar a cabo la acción deseada.

En nuestro diagrama de clases hemos aplicado el patrón Fachada de tal forma que recoja todas las acciones que se pueden llevar a cabo en el sistema y guarde también todo tipo de datos, que serán posteriormente solicitados por cualquiera de las clases con las cuales está relacionada. A continuación adjuntamos una imagen y un ejemplo de uso.



- **Ejemplo: updateEstado(in “Cerrada”: string, in idActa: integer)**

Cuando el usuario por ejemplo, cuando el árbitro principal de un partido cierre el acta correspondiente, se lanzará un evento que será recogido por el controlador correspondiente en este caso por el controlador “controladorCerrarActas”. Éste, indicará qué clases de dominio se verán involucradas en este evento. En este caso, una de las clases de dominio de la cual necesitamos extraer información es “Acta”. La clase de dominio “Acta” hará uso de la Fachada para modificar el atributo “estado” de un acta con identificador “idActa”.