Programación 4
Laboratorio 3

Roberto de Armas
Bruno Fernandez
Agustin Koltukian
Rafael Schol
Agustina Martinez

Impacto de los nuevos requerimientos

Los nuevos requerimientos no conllevan grandes cambios en los modelos anteriores, más que agregar algunos conceptos nuevos y asociarlos con otros conceptos ya existentes.

Cambios realizados

I Por el requerimiento de la búsqueda, se agrega el concepto Búsqueda. Este concepto se relaciona con Jugador (ya que los jugadores son quienes realizar las búsquedas).

II Por el requerimiento de las estadísticas, se agrega el concepto Estadística. Este concepto se asocia con Desarrollador (ya que los desarrolladores seleccionan sus estadísticas de interés).

III Dentro del concepto Videojuego, se tenía anteriormente un atributo categorías, que almacenaba todas las categorías del videojuego. Esto se cambió, siendo ahora Categoría un concepto en lugar de un datatype, y se cambia el atributo multivaluado categorías por una asociación entre Videojuego y Categoría. Esto se hace ya que por el nuevo requerimiento de la búsqueda, tener la asociación de Videojuego con Categoría facilitará al momento de realizar búsquedas por categoría.

Diseño de los nuevos requerimientos

Estadísticas

Para resolver el requerimiento de las estadísticas, se usó en el diseño el patrón Strategy. El propósito de este patrón es definir algoritmos intercambiables que variarán según lo que se necesite. En este caso, los algoritmos serán las funciones que calculen alguna estadística en particular, y habrá un algortimo por cada tipo de estadística que se pueda solicitar. Se cumple el requerimiento no funcional de la flexibilidad, ya que para agregar un nuevo tipo de estadística, simplemente se agregará una nueva clase a la cuál se le definirá el algoritmo de cálculo para esa estadística.

Roles:

I Estrategia: Estadística

II Estrategia concreta: Estadística particular (PuntajePromedio, CantidadSuscipciones, ...)

III Contexto: Desarrollador

Buscador de videojuegos

Para resolver el requerimiento del buscador de videojuegos, en el diseño se utilizó un patrón Composite, dado que composite es útil para crear objetos complejos a partir de otros más simples y similares entre sí, y una Búsqueda avanzada se compone por Búsquedas simples (que son criterios por los cuáles buscar). Esto se puede representar como un árbol a través de la recursión en la estructura, y cuya raíz sería la misma búsqueda avanzada mientras su ramas serían las búsquedas simples que componen la avanzada.
Este patrón resuelve el requerimiento no funcional pedido, ya que para cada nueva Búsqueda simple (criterio de búsqueda) que se quiera agregar, se tiene la flexibilidad de que esto se hará simplemente agregando una nueva clase hija de Búsqueda, y especificando su operación para realizar la búsqueda.

Roles:

I Componente: Búsqueda.

II Hoja: Búsqueda simple (PorNombre, PorCategoría, ...).

III Compuesto: Búsqueda avanzada.

Criterios GRASP y patrones de diseño

Controller

Se ulilizaron 3 controller en el diseño. Estos mismos no corresponden a la definicion de façade controller ni de use-case controller, se trata de un hibrido entre dichos tipos de controller. Especificamente nuestros controladores se encargan de de manejar las operaciones del sistema de los siguientes casos de uso:

Controlador de Videojuego: eliminar videojuego y suscribirse a videojuego.

Controlador de Partida: iniciar partida y finalizar partida.

Controlador de Usuario: alta de usuario, publicar videojuego y consultar estadisticas.
Gracias a estas asignaciones se logra mantener una alta cohesión y un bajo acoplamiento.

Expert

Se usó muy frecuentemente en gran parte de las operaciones, para asignar algunas responsabilidades a las clases expertas que conozcan la información necesaria para relizar alguna operación.

Por ejemplo:
Dentro de la operación listarVideojuegosConSuscripciónActiva(), en un momento la clase Jugador delega la operación getVideojuegoConSuscripción(), a la

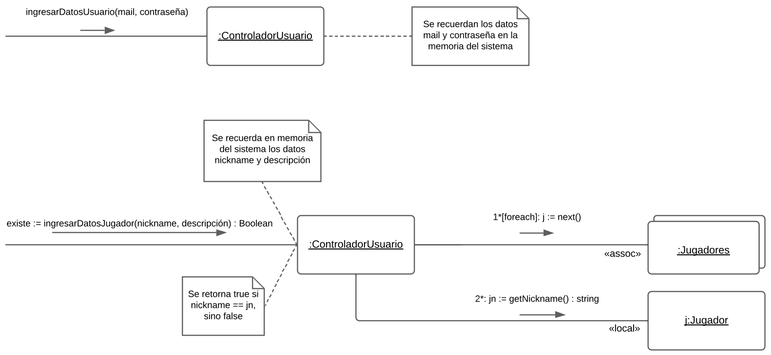
clase JugadorVideojuego, ya que se piden las suscripciones para un videojuego, y la clase JugadorVideojuego es la experta en conocer las suscripciones de un jugador con un videojuego.

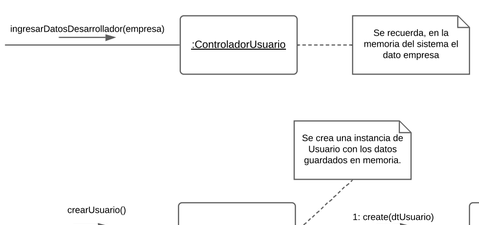
Creator

Se usó para determinar cuál clase es la responsable de la creación de instancias de otra clase.

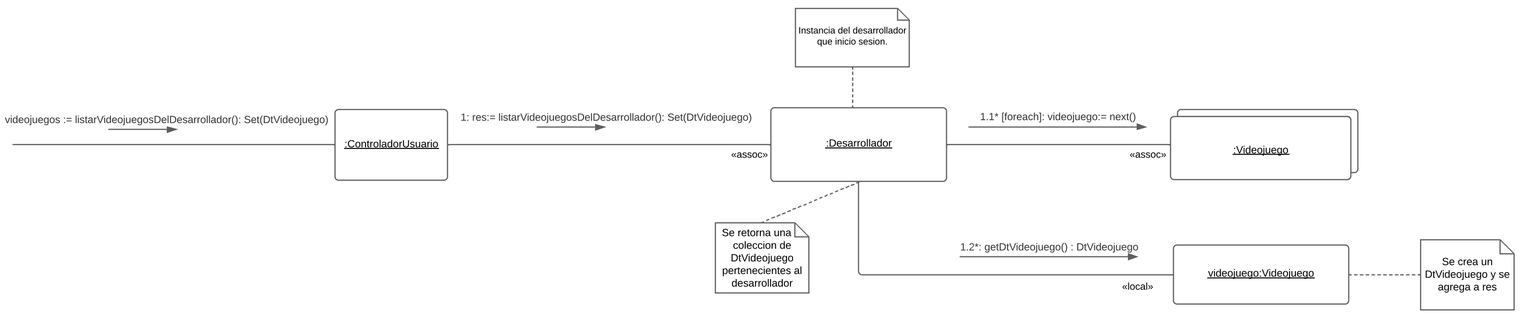
Por ejemplo:
Al crear una instancia de Suscripción, la responsabilidad de crearla se le pasa a JugadorVideojuego, ya que esta clase es quien registra las instancias de Suscripción.
Por último, se intentó seguir en la mayoría de los casos (exceptuando excepciones puntuales) a los criterios de bajo acoplamiento, alta cohesión y no hables con extraños, ya que estos tres criterios nos brindan buenas prácticas para el diseño, que a futuro ayudan a prevenir problemas.

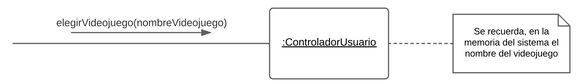
ALTA DE USUARIO:

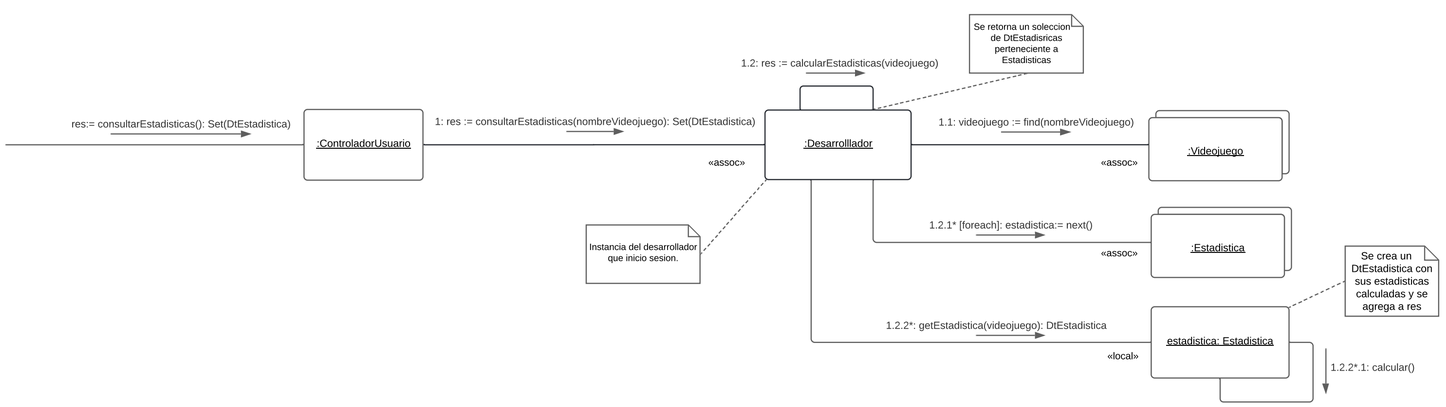




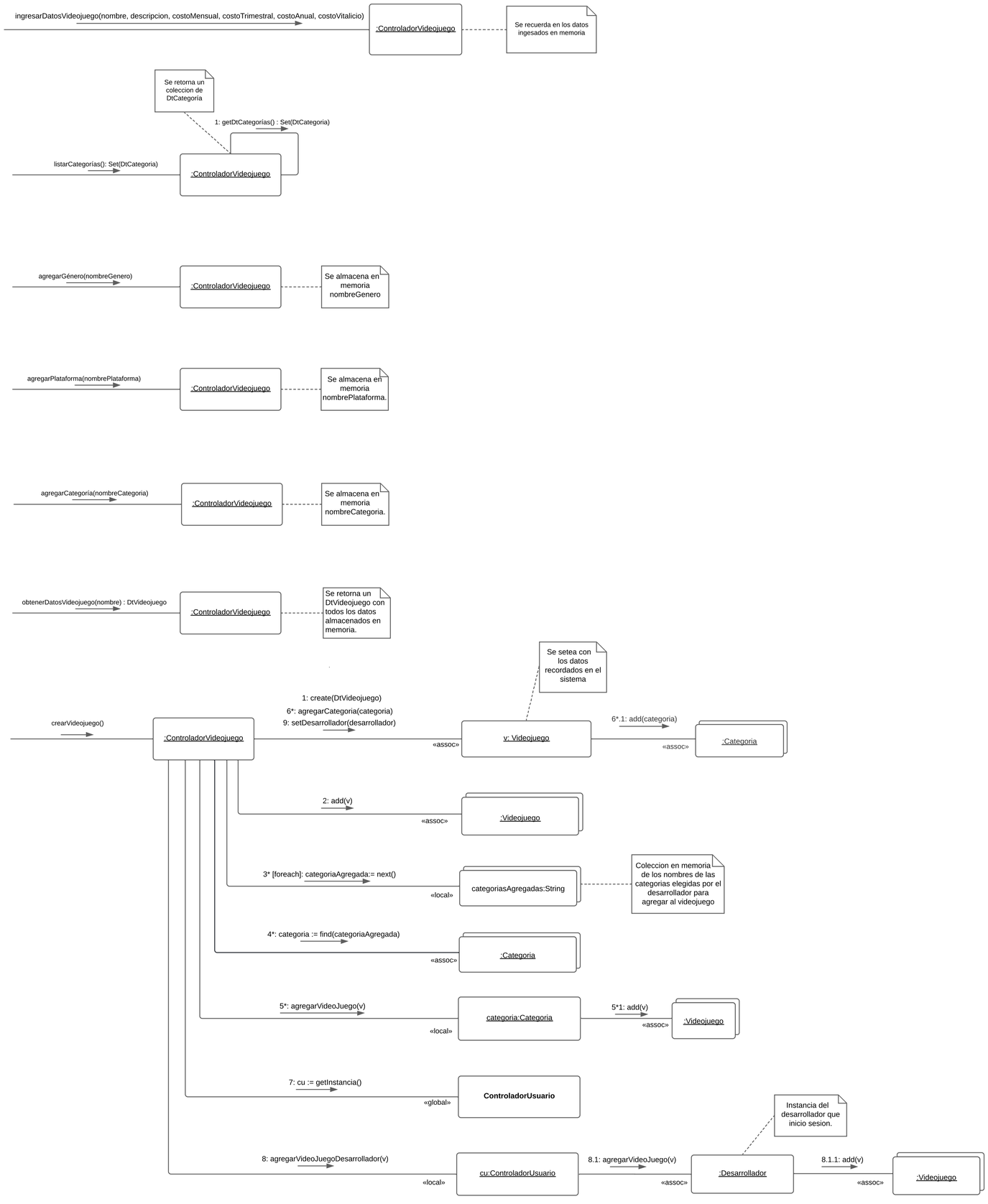
CONSULTAR ESTADISTICAS:



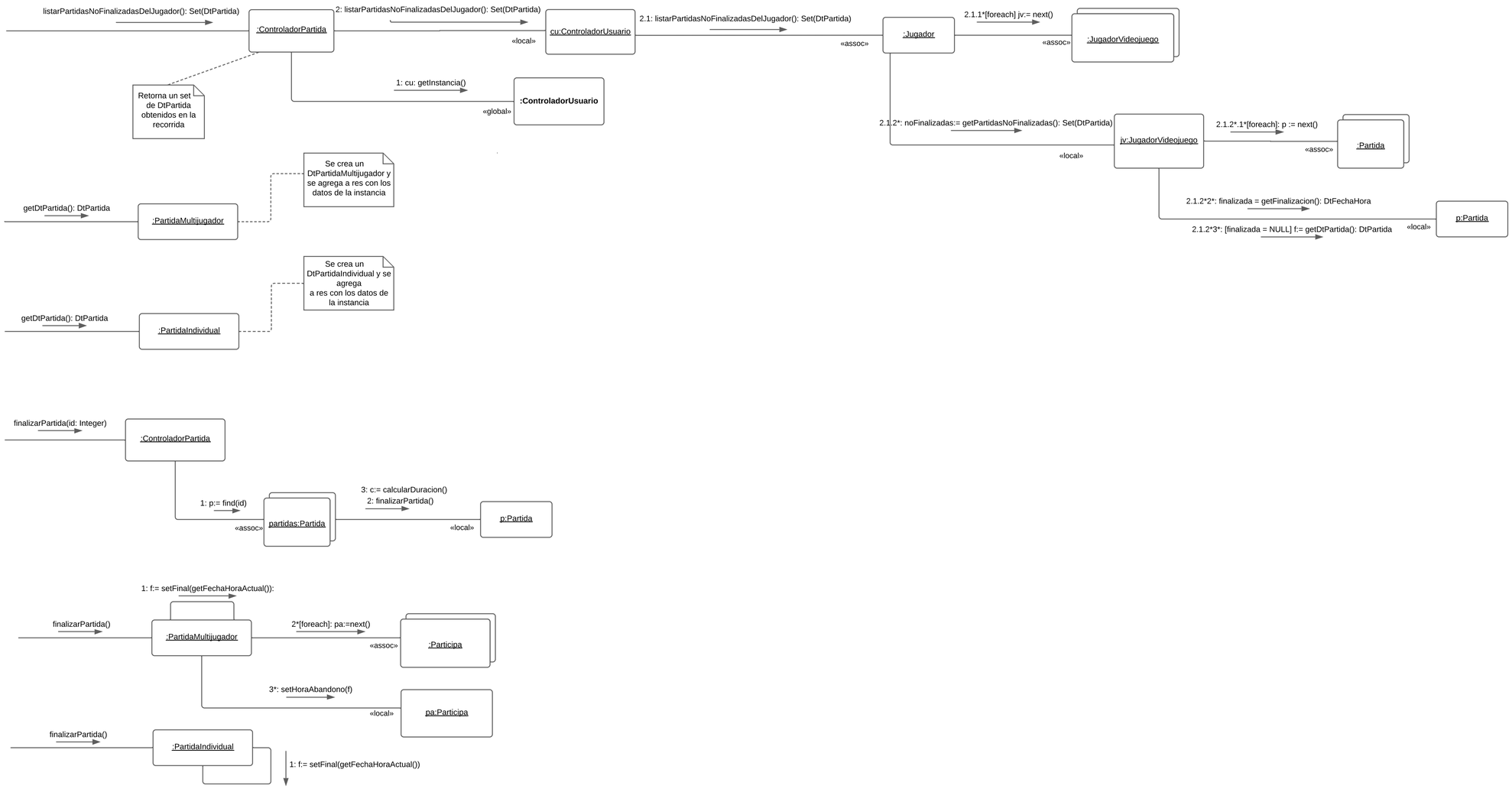




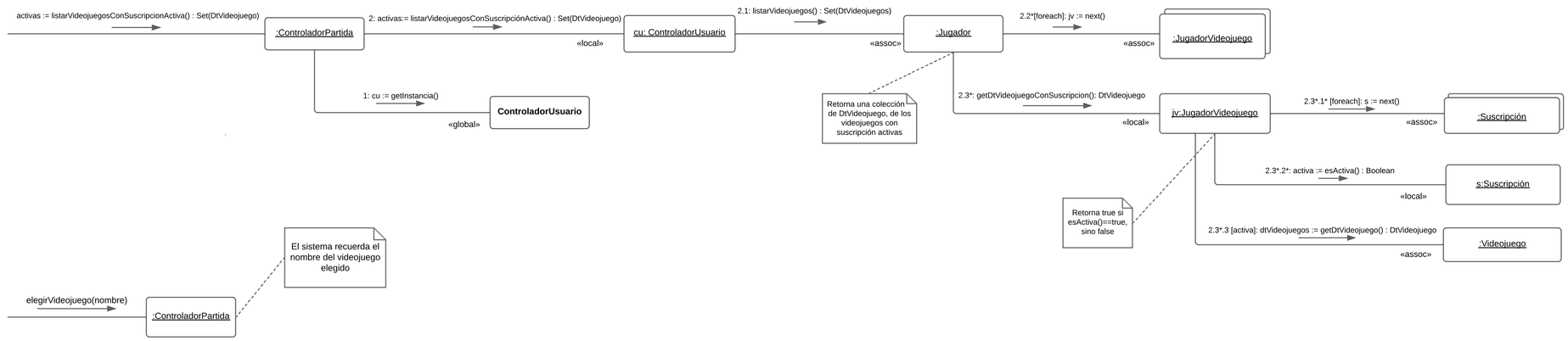
PUBLICAR VIDEOJUEGO:

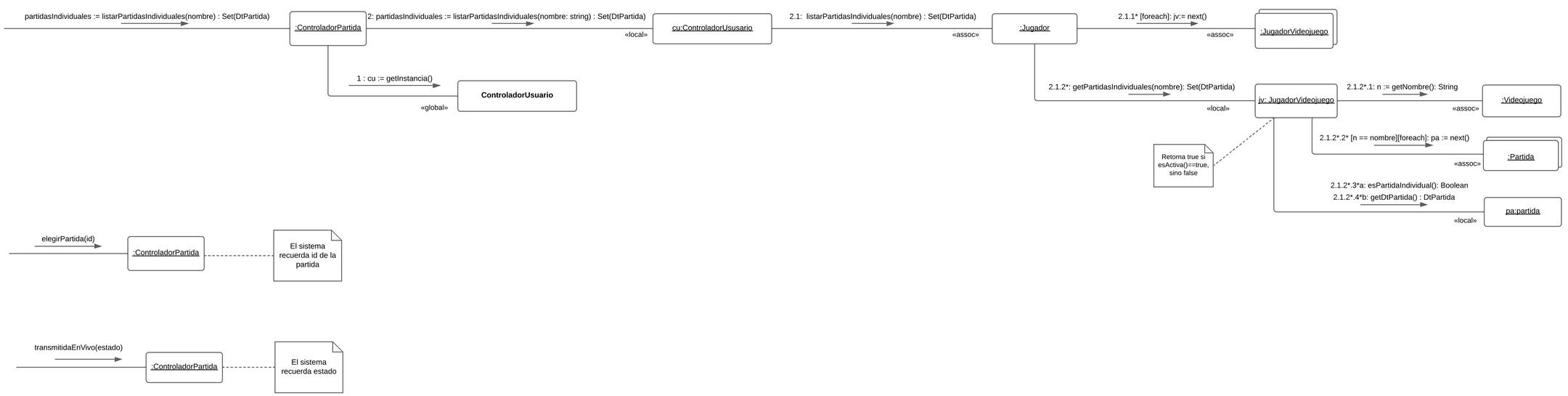


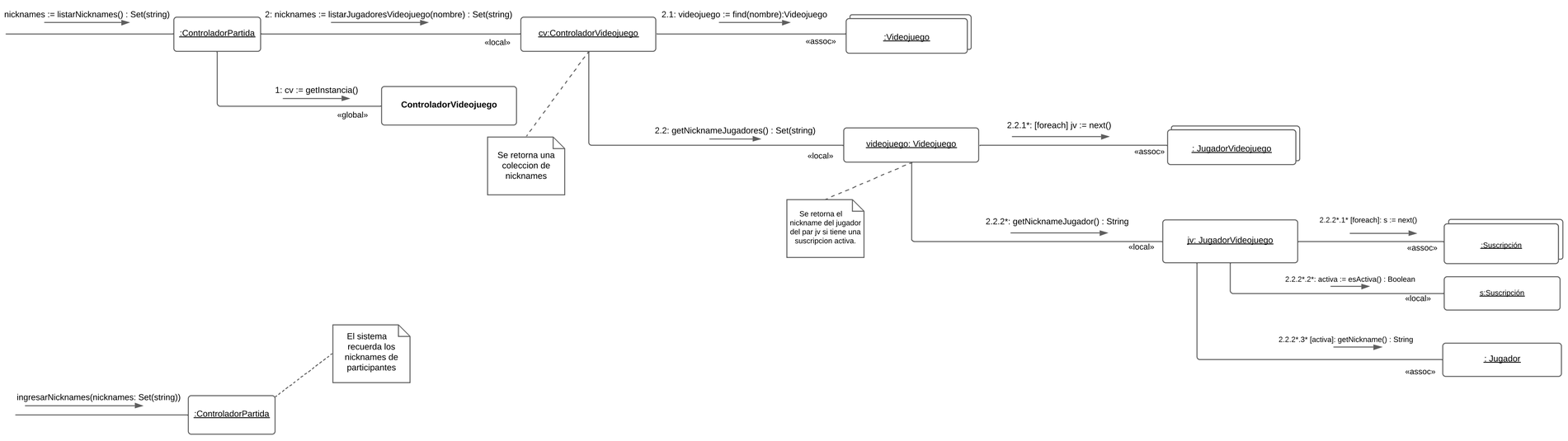
FINALIZAR PARTIDA:



INICIAR PARTIDA:







confirmarInicioPartida()

:ControladorPartida

3: confirmarInicioPartidaIndividual (dtPartida,nombre)

«local»

cu:ControladorUsuario

3.1: confirmarInicioPartidaIndividual(dtPartida, nombre)

3.1.1: parVideojuego := buscarParVideojuego(nombre) : JugadorVideojuego

«assoc»

:Jugador

3.1.2* [foreach]: jv := next()

«assoc»

:JugadorVideojuego

Si la partida es continuacion de otra, se busca la partida a continuar

1: cu := getInstancia()

«global»

ControladorUsuario

2: partidaAContinuar := find(id)

:Partida

Se retorna jv si nombre == nombreVideojuegoPar

3.1.3*: nombreVideojuegoPar := getNombreVideojuego() : String

«local»

jv: JugadorVideojuego

3.1.3*.1: getNombre() : String

«assoc»

:Videojuego

3.1.2.1.1: setContinua(partidaAContinuar)

Si la partida es continuacion de otra se establece el link a ella.

3.1.4: agregarPartida(dtPartida)

«local»

parVideojuego: JugadorVideojuego

3.1.4.1: create(dtPartida)

«assoc»

partida:PartidaIndividual

3.1.4.2: add(partida)

«assoc»

:Partida

confirmarInicioPartida()

:ControladorPartida

2: confirmarInicioPartidaMultijugador(dtPartida,nombre)

«local»

cu:ControladorUsuario

2.1: jugadores := getJugadores(): Set(Jugador)

2.2: confirmarInicioPartidaMultijugador(dtPartida,jugadores,nombre)

2.2.1: parVideojuego := buscarParVideojuego(nombre) : JugadorVideojuego

«assoc»

:Jugador

2.2.2*[foreach]: jv := next()

«assoc»

:JugadorVideojuego

Si la partida es individual y continuacion de otra

1: cu := getInstancia()

«global»

ControladorUsuario

Se retorna jv si nombre == nombreVideojuegoPar

2.2.3*: nombreVideojuegoPar := getNombreVideojuego() : String

«local»

jv: JugadorVideojuego

2.2.3*.1: getNombre() : String

«assoc»

:Videojuego

2.2.4: agregarPartida(dtPartida)

«local»

parVideojuego: JugadorVideojuego

2.2.4.1: create(dtPartida)

«assoc»

partida:PartidaMultijugador

continua debajo

2.2.4.2: add(partida)

«assoc»

:Partida

continuacion

partida: PartidaMultijugador

2.2.4.1.1* [foreach]: nickname := next()

«param»

nicknames: String

Existe dentro del DtPartida que viene por parametro

2.2.4.1.2*: jugadorEncontrado := find(nickname)

«param»

jugadores: Jugador

2.2.4.1.3*: parVideojuego := buscarParVideojuego(nombre) : JugadorVideojuego

«local»

jugadorEncontrado: Jugador

La operacion buscarParVideojuego usa siempre la misma implementacion, que ya fue definida anteriormente

2.2.4.1.4*.1: setJugadorVideojuego(parVideojuego)

2.2.4.1.4*.2: setPartida(partida)

2.2.4.1.4*: create()

«assoc»

participa: Participa

2.2.4.1.5*: agregarParticipacion(participa)

«local»

parVideojuego: JugadorVideojuego

2.2.4.1.5*1.: add(participa)

«assoc»

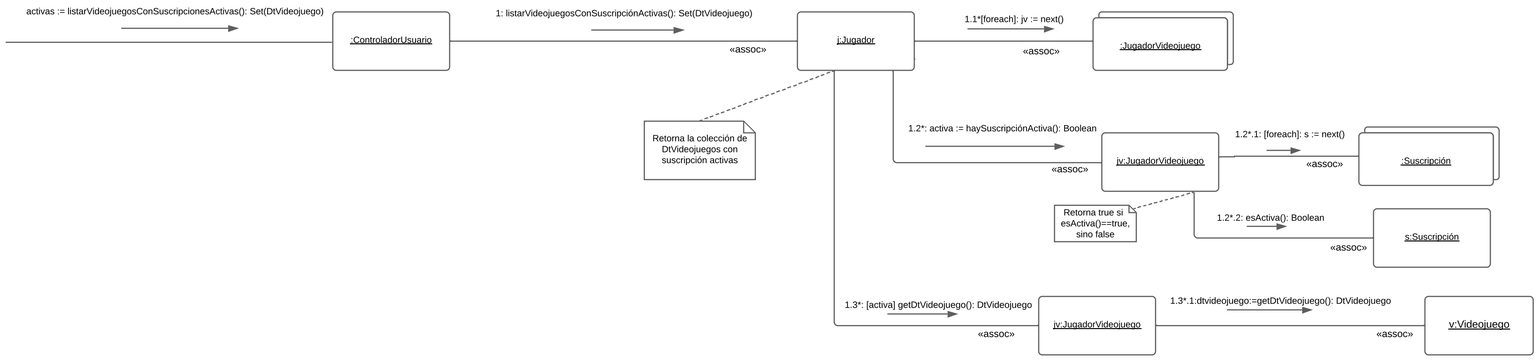
:Participa

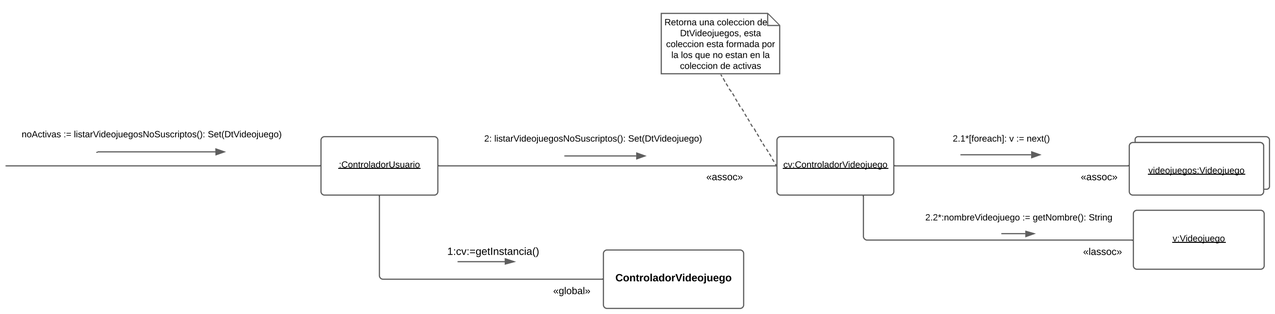
2.2.4.1.6*: add(participa)

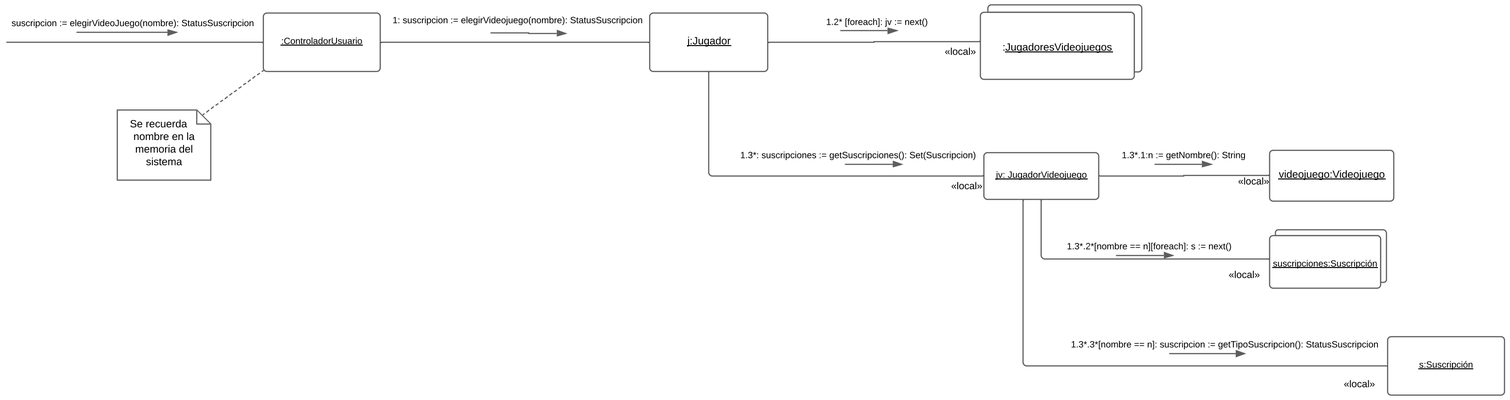
«assoc»

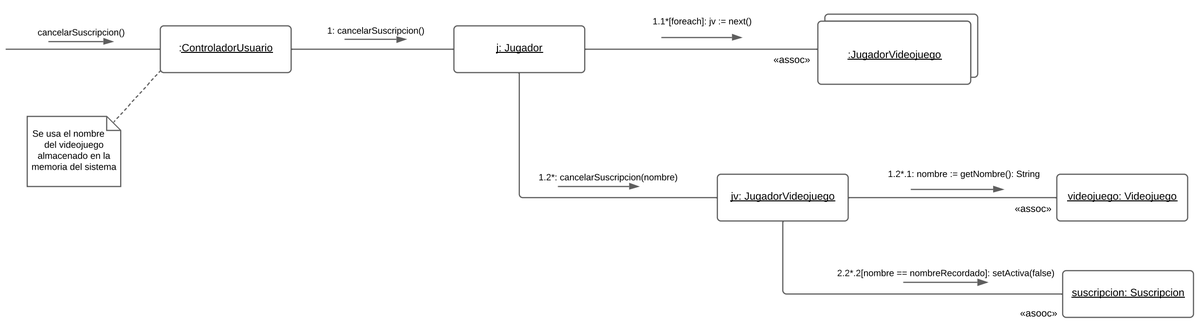
:Participa

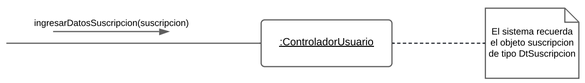
SUSCRIBIRSE A VIDEOJUEGO:

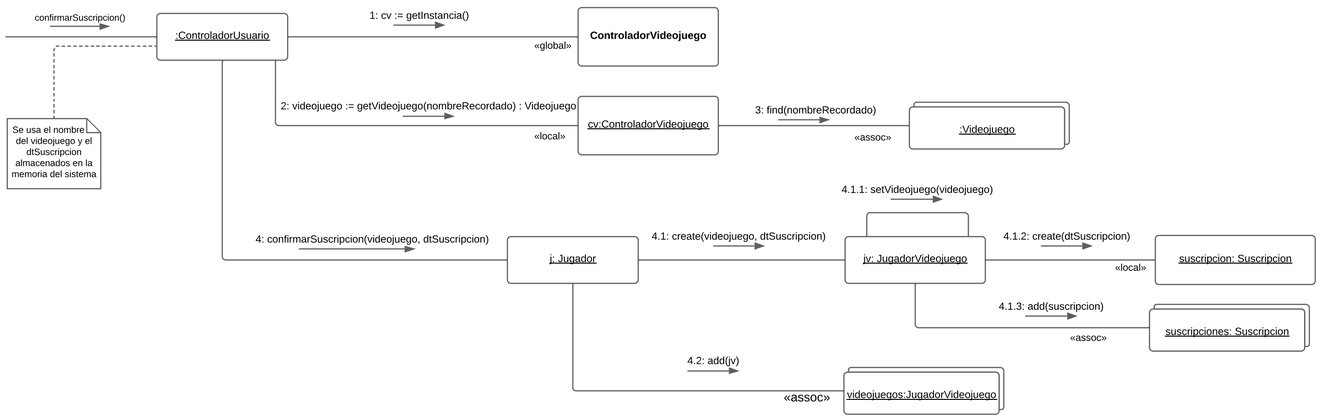




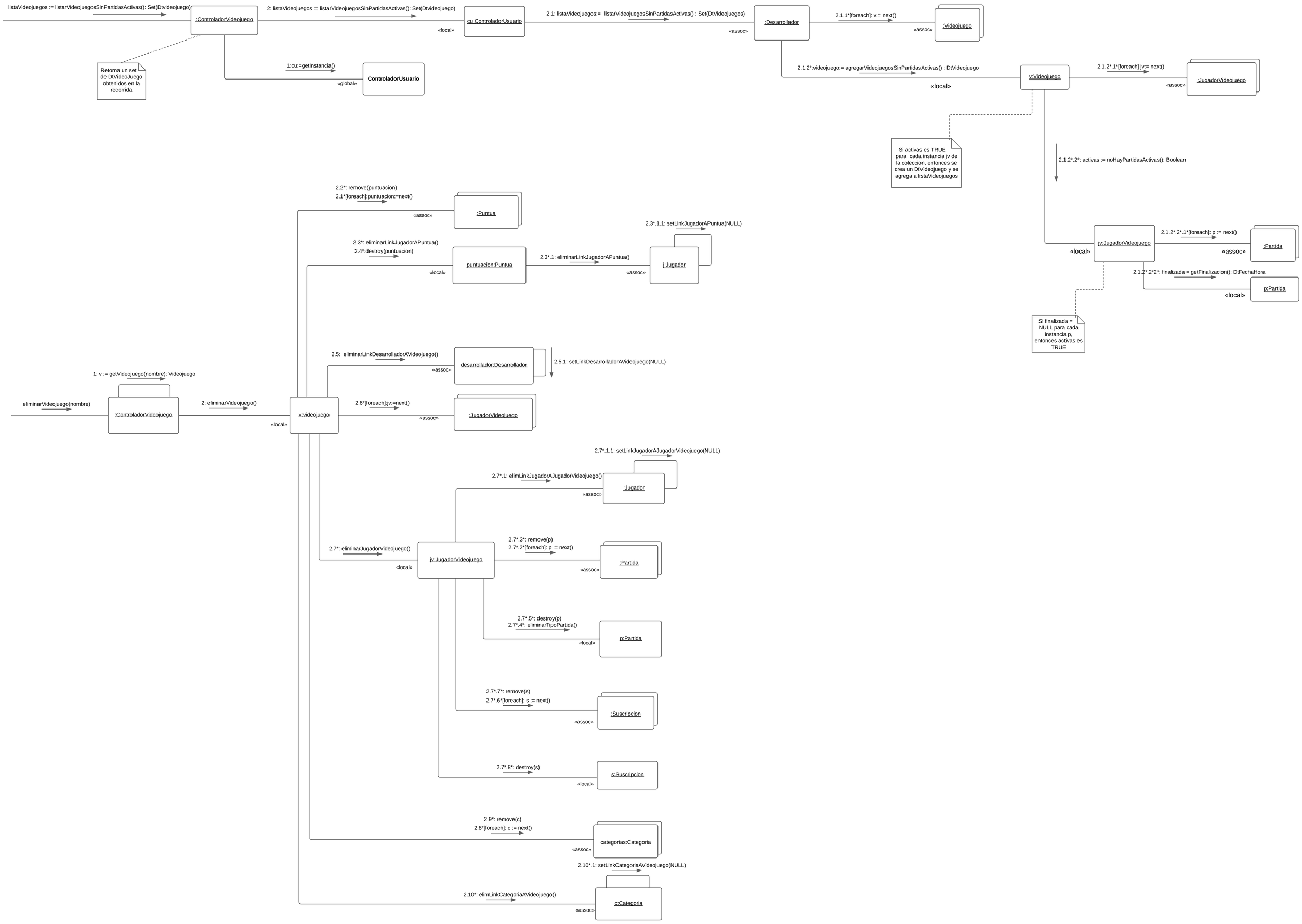







confirmarSuscripcion()
:ControladorUsuario
1: cv := getInstancia()
«global»
ControladorVideojuego
2: videojuego := getVideojuego(nombreRecordado) : Videojuego
«local»
cv:ControladorVideojuego
3: find(nombreRecordado)
«assoc»
:Videojuego
Se usa el nombre del videojuego y el dtSuscripcion almacenados en la memoria del sistema
4.1.1: setVideojuego(videojuego)
4: confirmarSuscripcion(videojuego, dtSuscripcion)
j: Jugador
4.1: create(videojuego, dtSuscripcion)
jv: JugadorVideojuego
4.1.2: create(dtSuscripcion)
«local»
suscripcion: Suscripcion
4.1.3: add(suscripcion)
«assoc»
suscripciones: Suscripcion
4.2: add(jv)
«assoc»
videojuegos:JugadorVideojuego

ELIMINAR VIDEOJUEGO:



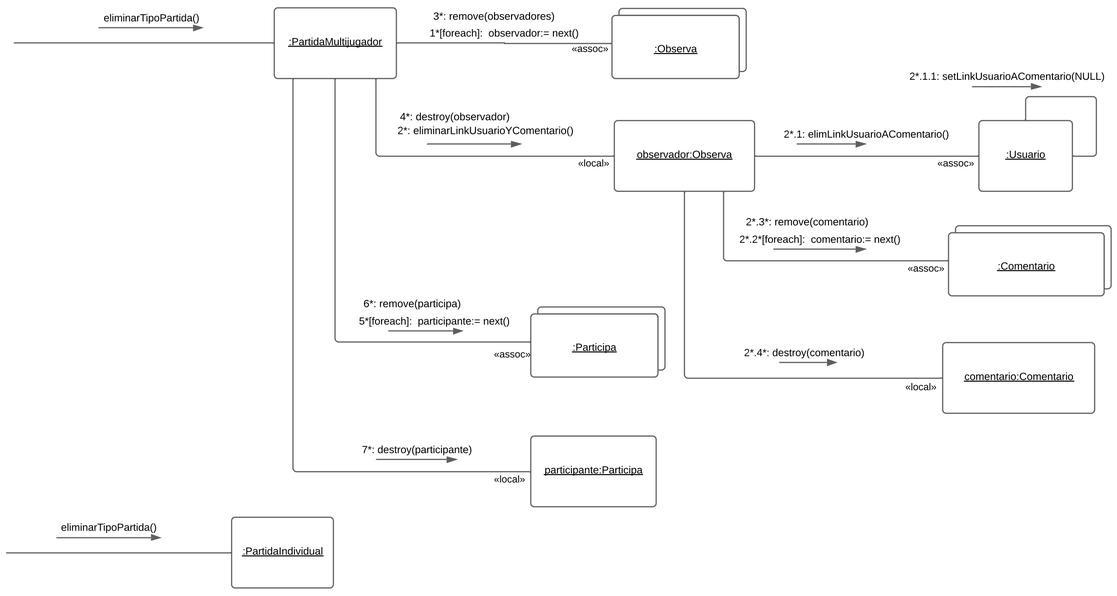

eliminarTipoPartida()
:PartidaMultijugador
3*: remove(observadores)
1*[foreach]: observador:= next()
«assoc»
:Observa
2*.1.1: setLinkUsuarioAComentario(NULL)
4*: destroy(observador)
2*: eliminarLinkUsuarioYComentario()
«local»
observador:Observa
2*.1: elimLinkUsuarioAComentario()
«assoc»
:Usuario
2*.3*: remove(comentario)
2*.2*[foreach]: comentario:= next()
«assoc»
:Comentario
6*: remove(participa)
5*[foreach]: participante:= next()
«assoc»
:Participa
2*.4*: destroy(comentario)
«local»
comentario:Comentario
7*: destroy(participante)
«local»
participante:Participa
eliminarTipoPartida()
:PartidaIndividual

DIAGRAMA DE CLASES