# Índice de contenido

# INTRODUCCIÓN

Desde hace muchos años, se tiene como mercado de cualquier tipo de software, a personas de edad media o adulta. Este es el mercado mas simple de capturar, y a su vez el más numeroso.

Pero el uso de herramientas de software para la educación, ha ampliado estos horizontes, por lo que se debe incluir a los niños y menores de edad dentro del posible mercado, por lo que se debe iniciar a estos en los conceptos y entornos del software.

Debido a la dificultad de comprensión, se deben usar métodos mas sutiles de convencimiento, los cuales no deben atemorizarlos, sino, inculcarles un criterio básico sobre el software, además de fomentar el uso de las tecnologías de la información.

## 1.1 Objetivos

## 1.2 Objetivos Generales

Desarrollar un juego de rompecabezas colaborativo para niños comprendidos entre los 3 y 6 años, para reducir la brecha de aprendizaje del software, utilizando una arquitectura cliente-servidor en un entorno de red.

## 1.3 Objetivos Específicos

- Realizar la captura de requisitos del software, mediante investigaciones sobre tecnología de red y juegos.

- Seguir el desarrollo sistemático de todas las actividades según el PUDS “Proceso Unificado de Desarrollo de Software”, para el software de componente de seguridad.

- Aplicar UML “Lenguaje de Modelado Unificado”, para representar todos los diagramas y estereotipos que utiliza el PUDS en cada flujo de trabajo.

- Refinar los requisitos a partir del conjunto de requerimientos para el software, para esto aplicar el flujo de trabajo análisis según PUDS.

- Utilizar la tecnología y componentes necesarios y adecuados para el desarrollo, en este caso, RMI de Java.

- Investigar sobre tendencias y preferencias de los clientes, mediante datos estadísticos, o simple observación.

- Aplicar una interfaz entendible, tomando en cuenta las limitaciones de los clientes potenciales, debido a su condición actual.
- Manejar distintas partidas de juego, las cuales contaran con equipos de juego.

- Desarrollar la aplicación final con la documentación requerida.

## 1.4 Alcance

### *1.4.1 Requerimientos funcionales*

**Partidas de juego**

- Administrar las sesiones de juego de todos los clientes que se conecten.
- Comunicar a todos los clientes sobre algún nuevo jugador.
- Manejar los posibles errores de conexión y funcionamiento en red.
- Manejar y llevar datos estadísticos de cada jugador.
- Registrar los datos de los clientes, ademas de todos aquellos datos relevantes de su partida.
- Registrar los movimientos generados por los jugadores, manipular las preferencias de una partida.
- Permitir la correcta ejecución del juego, reduciendo errores y cuelgues.
- Generar y manipular los eventos del jugador, como la interacción con la piezas y otros.

**Gestión de Comunicación**

- Permitir la correcta comunicación entre los clientes y el servidor de la aplicación.
- Administrar el envío y recepción de mensajes.
- Administrar la configuración de las direcciones de objetos remotos.

# MARCO TEÓRICO

## *1 Java*

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de Lenguaje de programación C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de Puntero (programación) punteros o memoria.

Las aplicación Java|aplicaciones Java están típicamente compilador compiladas en un bytecode, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el bytecode es normalmente intérprete informático|interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del bytecode por un procesador Java también es posible.

La implementación original y de referencia del compilador, la máquina virtual y las librerías de clases de Java fueron desarrollados por Sun Microsystems en 1995. Desde entonces, Sun ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del Java Community Process, si bien otros han desarrollado también implementaciones alternativas de estas tecnologías de Sun, algunas incluso bajo licencias de software libre.

Entre noviembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus

tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre (aunque la biblioteca (programación)|biblioteca de clases de Sun que se requiere para ejecutar los programas Java todavía no es software libre).

Algunas de sus características:

- Orientado a Objetos
- Independiente de la plataforma
- Recolector de basura
- Soporte para entornos de red y dispositivos portátiles

## *2 RMI*

RMI (Java Remote Method Invocation) es un mecanismo ofrecido en Java para invocar un método remotamente. Al ser RMI parte estándar del entorno de ejecución Java, usarlo provee un mecanismo simple en una aplicación distribuida que solamente necesita comunicar servidores codificados para Java. Si se requiere comunicarse con otras tecnologías debe usarse CORBA o SOAP en lugar de RMI.

Al estar específicamente diseñado para Java, RMI puede darse el lujo de ser muy amigable para los programadores, proveyendo pasaje por referencia de objetos (cosa que no hace SOAP), "recolección de basura" distribuida y pasaje de tipos arbitrarios (funcionalidad no provista por CORBA).

Por medio de RMI, un programa Java puede exportar un objeto. A partir de esa operación este objeto está disponible en la red, esperando conexiones en un puerto TCP. Un cliente puede entonces conectarse e invocar métodos. La invocación consiste en el "marshalling" de los parámetros (utilizando la funcionalidad de "serialización" que provee Java), luego se sigue con la invocación del método (cosa que sucede en el servidor). Mientras esto sucede el llamador se queda esperando por una respuesta. Una vez que termina la ejecución el valor de retorno (si lo

hay) es serializado y enviado al cliente. El código cliente recibe este valor como si la invocación hubiera sido local.

La arquitectura RMI puede verse como un modelo de cuatro capas:

- **Primera capa**

  La primera capa es la de aplicación y se corresponde con la implementación real de las aplicaciones cliente y servidor.

  - **Segunda capa**

  Esta capa es la que interactúa directamente con la capa de aplicación. Todas las llamadas a objetos remotos y acciones junto con sus parámetros y retorno de objetos tienen lugar en esta capa.

  - **Tercera capa**

  La capa 3 es la de referencia remota, y es responsable del manejo de la parte semántica de las invocaciones remotas.

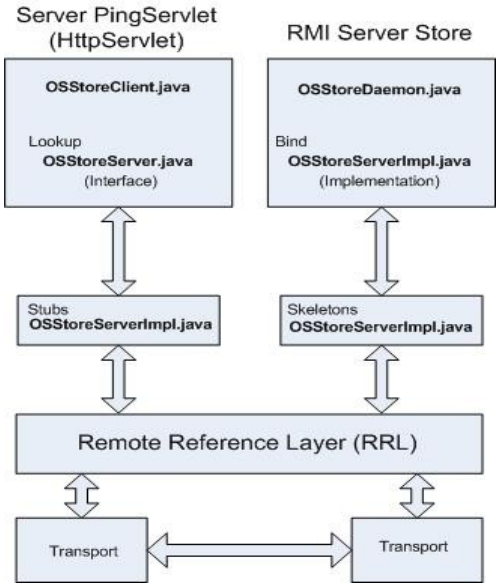
  - **Cuarta Capa**

  La capa 4 es la de transporte. Es la responsable de realizar las conexiones necesarias y manejo del transporte de los datos de una máquina a otra. El protocolo de transporte subyacente para RMI es JRMP (Java Remote Method Protocol).

### 2.1 Elementos

Toda aplicación RMI normalmente se descompone en 2 partes:

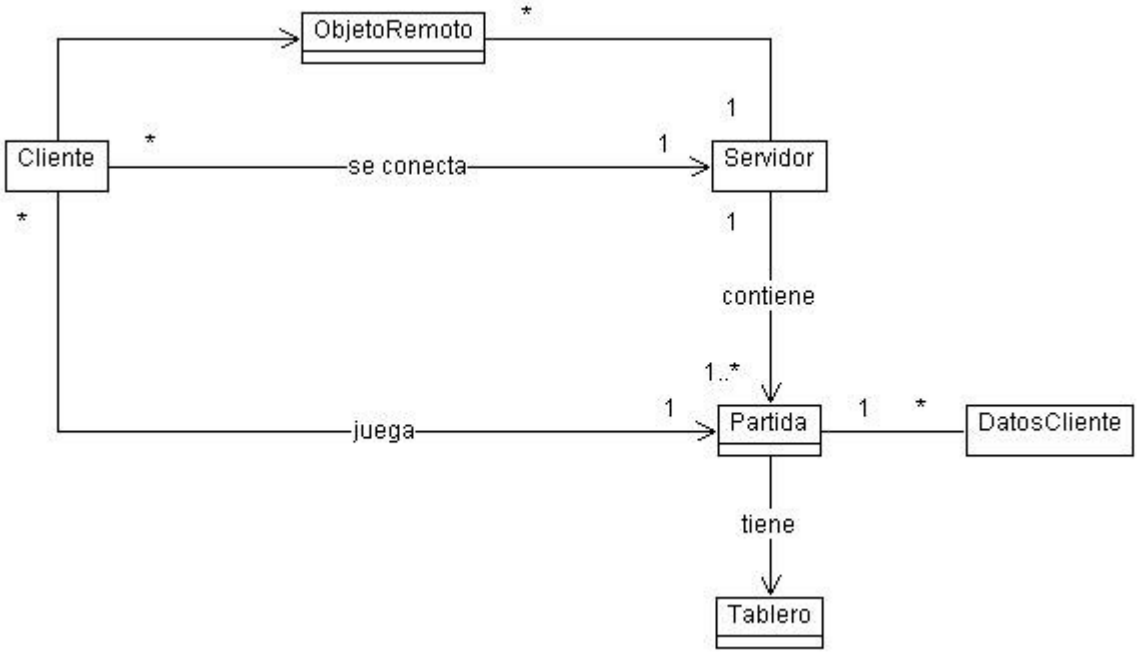
- Un servidor, que crea algunos objetos remotos, crea referencias para hacerlos accesibles, y espera a que el cliente los invoque.
- Un cliente, que obtiene una referencia a objetos remotos en el servidor, y los invoca.



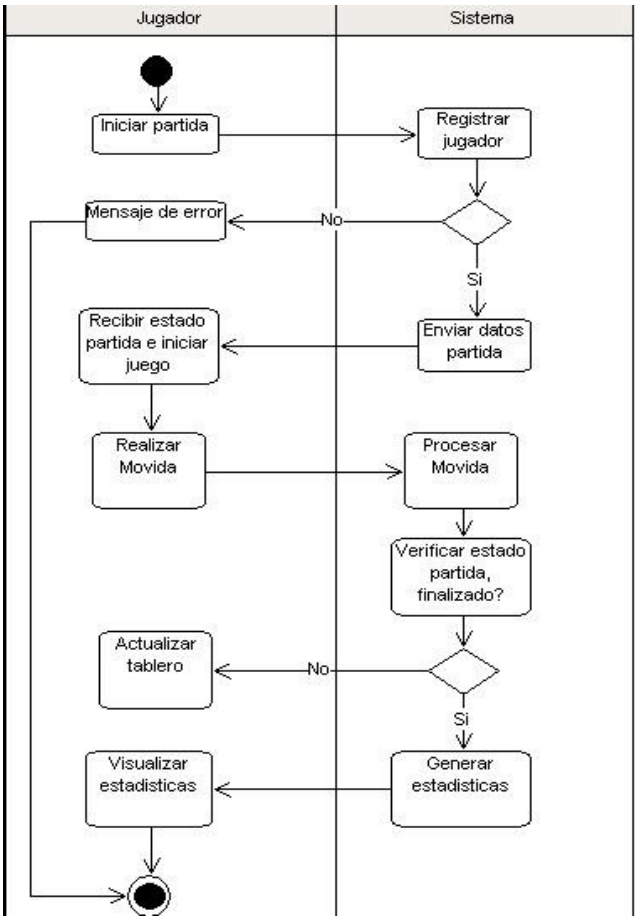

*Esquema RMI*

# REQUERIMIENTOS

## *1 Modelo de dominio*



## *2 Modelo de negocio*




# REQUISITOS

## *1 Actores*

Jugador: Persona que utiliza el juego, en este caso, un niño, con una edad establecida entre los 3 y 6 años.

Administrador: Persona encargada de la instalación y configuración de la aplicación.

## *2 Casos de Uso*

- **Selección del Servidor** : Permite seleccionar el nombre o dirección del servidor de aplicación.

- **Gestionar Sesiones** : Permite iniciar el juego, si no existe alguna sesión previa, se genera una nueva con las características deseadas. Caso contrario, se conecta con la sesión actual. Además permite salir del juego en cualquier punto del mismo, o al finalizarlo.

- **Gestionar Movida** : Permite mover una pieza en particular para colocarla en algún lugar del tablero, evitando el acceso múltiple a la misma, ademas de controlar todas las excepciones con respecto a la posición correcta.

- **Gestionar mensajes** : Permite el envío de algún tipo de mensaje preestablecido hacia cualquiera de los usuarios en juego.

- **Cambiar imagen:** Permite cambiar la imagen del tablero de juego.

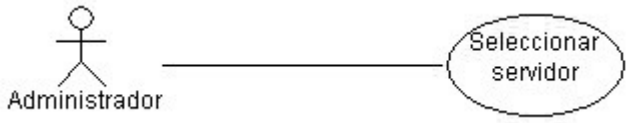
- **Consultar ayuda**: Permite visualizar la ayuda de la aplicación, mediante un navegador o reproductor externo.

## *3 Priorizar casos de uso*

| Caso de Uso | Prioridad |
|---|---|
| Selección servidor | Alta |
| Gestionae sesiones | Alta |
| Gestionar Movida | Alta |
| Gestionar mensajes | Media |
| Cambiar imagen | Baja |
| Consultar ayuda | Baja |

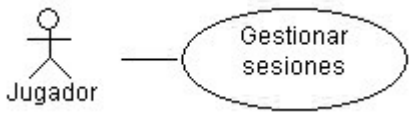
## *4 Detalle de casos de uso*

### Caso de uso : Seleccionar Servidor




| | |
|---|---|
| Caso de Uso | Seleccionar servidor |
| Actores | Administrador |
| Propósito | Permitir la configuración de localización del servidor del juego. |
| Iniciador | Administrador |
| Precondiciones | Ninguna |
| Flujo Principal | 1. El cliente inicia la utilidad de configuración.<br>2. Cargar los datos del archivo de configuración.<br>3. Cliente especifica dirección del servidor y acepta la operación.<br>4. El sistema verifica la disponibilidad del servidor.<br>5. Si el servidor esta disponible en la dirección especificada<br>a) Asignar como servidor a la dirección especificada<br>6. Si no esta disponible<br>a) Mostrar un mensaje de error indicando la falta de disponibilidad del servidor<br>7. Almacenar la dirección especificada en el archivo de configuración, si tal archivo no existe, se crea, en el caso de que no se haya seleccionado correctamente un servidor, se utiliza por defecto la dirección local. |
| Postcondiciones | -- |
| Excepciones | Si no se define una dirección valida, se toma por defecto la dirección del usuario local. |

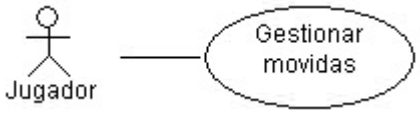
**Caso de uso : Gestionar Sesiones**




| | |
|---|---|
| Caso de Uso | Gestionar sesiones |
| Actores | Jugador |
| Propósito | Permitir el inicio de sesión, el inicio del juego, y la salida del mismo. |
| Iniciador | Jugador |
| Precondiciones | Seleccionar servidor |
| Flujo Principal | 1. Jugador inicia la aplicación.<br>2. Selección de identificador de usuario(nick), se acepta la operación y se conecta al servidor.<br>3. Servidor del juego procesa la petición de conexión.<br>4. Si existe una partida ya definida:<br>a) Si el identificador de usuario no esta en uso<br>a.1 Se asigna un equipo al jugador, y se envían los datos modificados.<br>b) Si el identificador ya esta en uso.<br>b.1 Se envía un mensaje de error y se pide definir otro identificador.<br>5. Si no existe ninguna partida definida:<br>a) Se asigna un equipo al jugador, se le envían los datos modificados, y se pide definir las preferencias de la partida.<br>b) El jugador define las preferencias y acepta la operación.<br>6. Se avisa a todos los jugadores conectados del nuevo participante.<br>7. Se inicia la aplicación y se cargan todas las características.<br>8. Cerrar la aplicación principal, o accionar el botón correspondiente.<br>9. La aplicación se encarga de avisar a todos los usuarios de la desconexión.<br>10. Se recibe el mensaje de confirmación de desconexión, se liberan los recursos y se finaliza la ejecución. |

| | |
|---|---|
| Postcondiciones | -- |
| Excepciones | Si la partida contiene la cantidad máxima de jugadores, se retorna un mensaje de error, y se cierra la aplicación. |

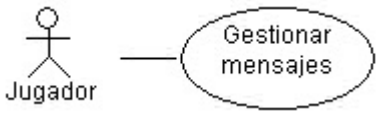
### Caso de Uso : Gestionar Movidas




| | |
|---|---|
| Caso de Uso | Gestionar Movidas |
| Actores | Jugador |
| Propósito | Permitir seleccionar, mover y colocar una pieza del rompecabezas. |
| Iniciador | Jugador |
| Precondiciones | Se debe tener un tablero de juego iniciado, por lo que se requiere que se tenga una sesión iniciada. |
| Flujo Principal | 1. Seleccionar una pieza del tablero, utilizando el dispositivo de entrada ratón.<br>2. Obtener información de la pieza y bloquearla para todos los demás jugadores.<br>3. Mover la pieza por el tablero.<br>4. Actualizar la posición de la pieza y graficarla, esto no requiere ser enviado a los demás jugadores.<br>5. Soltar la pieza en un lugar específico.<br>6. Si la pieza fue soltada en la ubicación correcta:<br>a) Se realiza el movimiento, se avisa a todos los participantes del mismo y se actualizan las estadísticas del jugador.<br>7. Si la pieza fue soltada en una posición incorrecta:<br>a) Se desbloquea la pieza para todos los participantes y se actualizan las estadísticas del jugador.<br>8. Si el tablero esta completado<br>a) Se calculan las estadísticas |

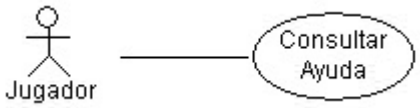
| | |
|---|---|
| | b) Se informa a todos los jugadores sobre el vencedor<br>c) Se envían las estadísticas de cada jugador, el cual las visualiza |
| Postcondiciones | -- |
| Excepciones | |

**Caso de Uso : Gestionar mensajes**



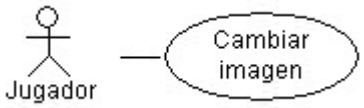

| | |
|---|---|
| Caso de Uso | Gestionar mensajes |
| Actores | Jugador |
| Propósito | Permitir interacción entre usuarios, mediante el intercambio de mensajes |
| Iniciador | Jugador |
| Precondiciones | Se debe tener un tablero de juego iniciado, por lo que se requiere que se tenga una sesión iniciada. |
| Flujo Principal | 1. Seleccionar el jugador a enviar el mensaje.<br>2. Mostrar los diferentes tipos de mensajes a enviar.<br>3. Seleccionar el tipo de mensaje a enviar y aceptar.<br>4. Enviar el mensaje hacia el jugador seleccionado, el cual lo visualiza. |
| Postcondiciones | -- |
| Excepciones | En el caso de que se envie el mensaje a uno mismo, se entiende como envio por difusion a todos los jugadores. |

### Caso de Uso: Consultar Ayuda




| | |
|---|---|
| Caso de Uso | Consultar ayuda |
| Actores | Jugador |
| Propósito | Permitir el despliegue de la guía de ayuda para cualquier requerimiento del usuario. |
| Iniciador | Jugador |
| Precondiciones | Se debe tener la aplicación iniciada, para visualizar la opción de ayuda. |
| Flujo Principal | 1. Seleccionar el botón de ayuda.<br>2. La aplicación cargará los datos necesarios, y los visualizará para los usuarios. |
| Postcondiciones | -- |
| Excepciones | |

### Caso de Uso: Cambiar imagen

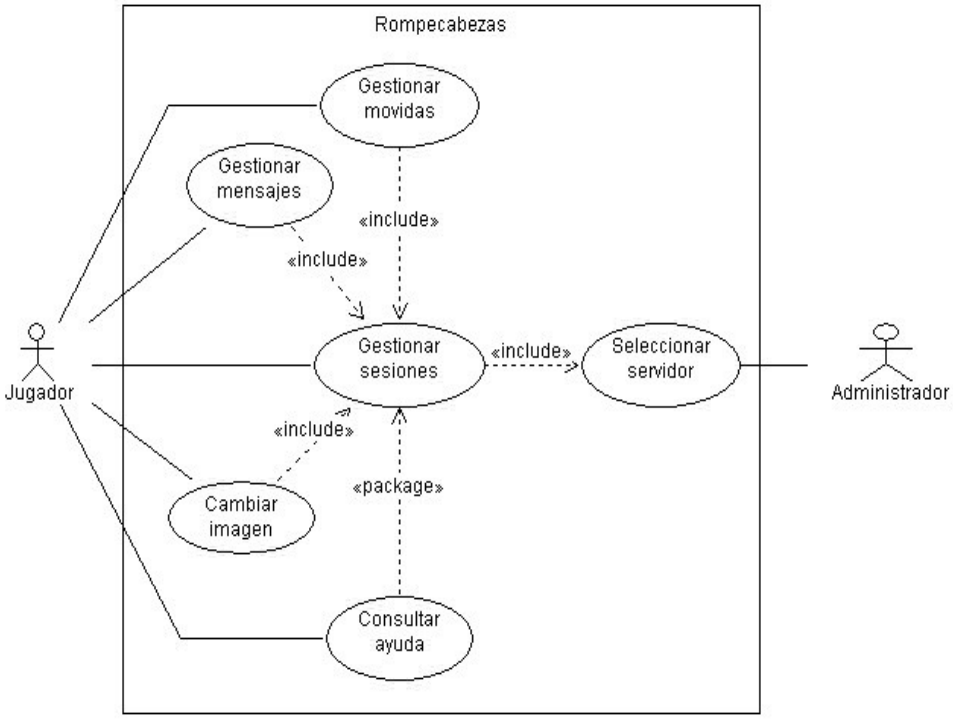



| | |
|---|---|
| Caso de Uso | Cambiar imagen |
| Actores | Jugador |
| Propósito | Permite el cambio de la imagen de juego. |
| Iniciador | Jugador |
| Precondiciones | Se debe tener la aplicación iniciada, para visualizar la opción de cambio de imagen. |
| Flujo Principal | 1. Seleccionar el botón de cambio de imagen.<br>2. La aplicación cargará los datos necesarios, y los visualizará para el usuario. |

| | |
|---|---|
| Postcondiciones | -- |
| Excepciones | |

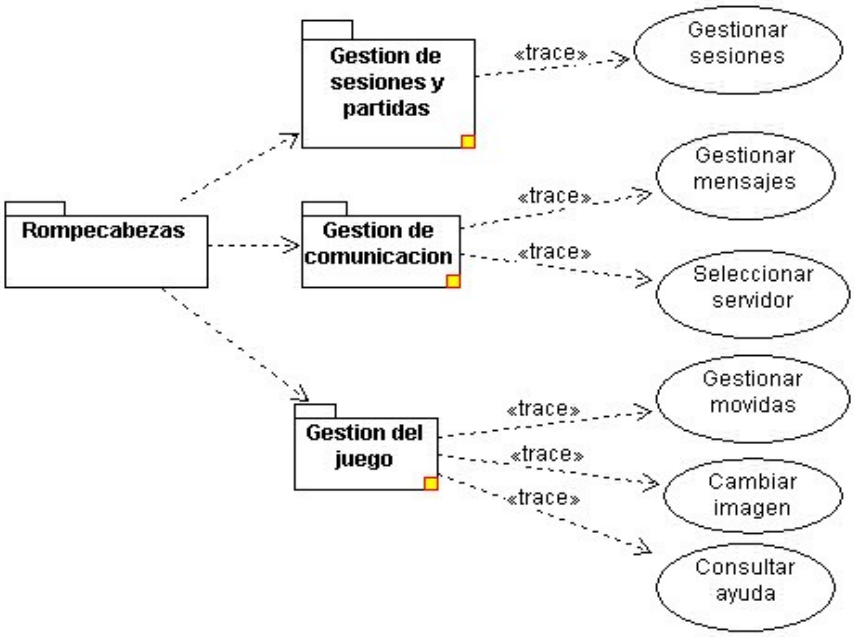
# *5 Estructurar Modelos de Casos de Uso*

## 5.1 Diagrama general de Casos de Uso



# ANÁLISIS

## *1 Análisis de la Arquitectura*




### 1.1 Descripción de Paquetes

***Gestión de sesiones y partidas***

Paquete encargado de administrar las sesiones de juego de todos los clientes conectados, y por conectar, también es el encargado de la notificación de clientes nuevos a todos los usuarios.

Además de manejar las posibles excepciones generadas durante el inicio o salida de alguna sesión.

***Gestión de Comunicación***

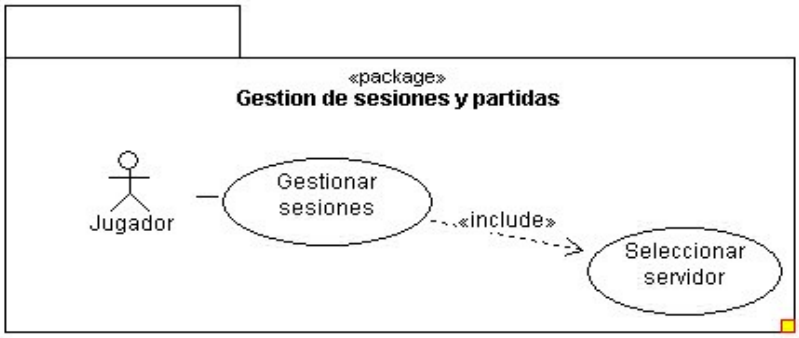
Paquete encargado de gestionar la comunicación entre jugadores, y el servidor, básicamente administra las tareas de envío y recepción de mensajes.

***Gestión del juego***

Paquete encargado de gestionar todas las acciones referentes al juego, como por ejemplo, movida de las piezas, su validación con respecto a la posición. Contiene todos los elementos necesarios para ejecutar un tablero de juego.

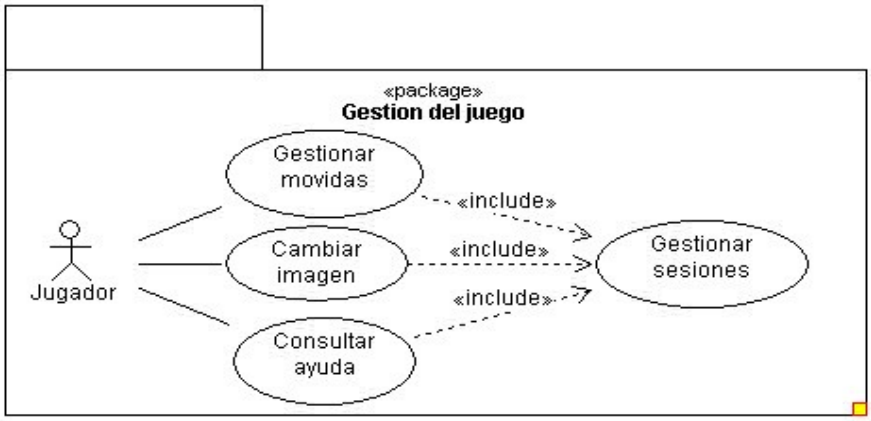
## 1.2 Exploración de paquetes

***Gestión de Sesiones y partidas***



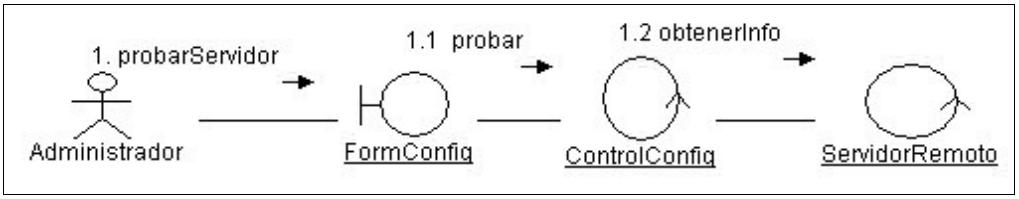

***Gestión de Comunicación***


***Gestión del juego***




## 1.3 Análisis de Casos de Uso

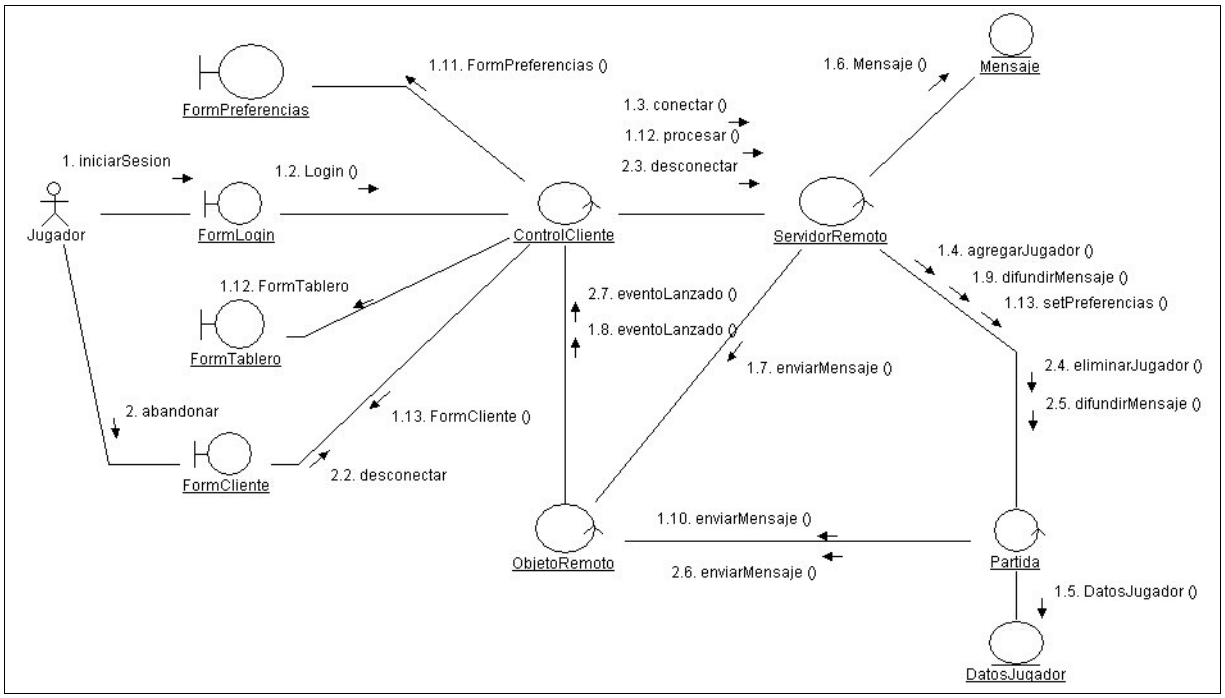
***Caso de Uso : Seleccionar servidor***




**Flujo de Suceso:**

El iniciador de este caso de uso es el Administrador, que es aquel usuario que instala y configura la aplicación, se inicia la utilidad de configuración, una vez iniciada, se especifica la dirección del servidor del juego. Si es que existe un servidor en esa dirección, se guarda la configuración y se finaliza la utilidad, caso contrario, se vuelve a pedir una dirección.

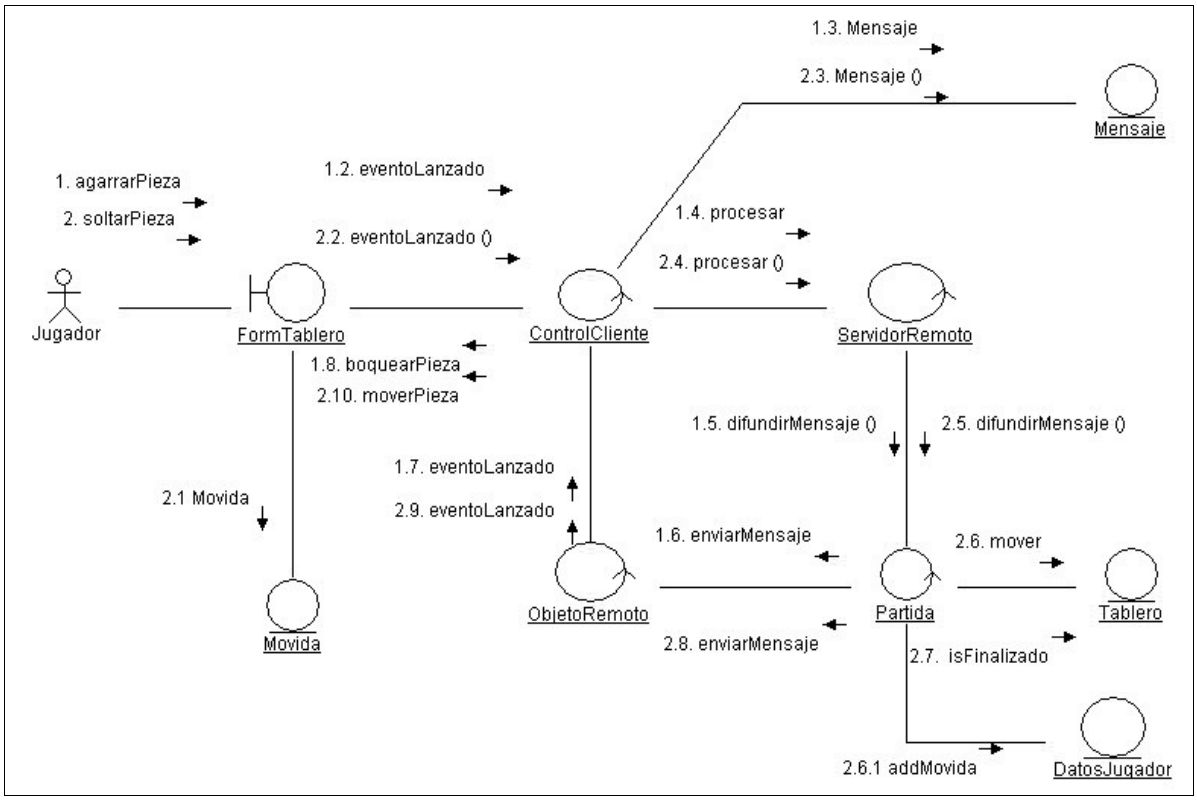
***Caso de Uso : Gestionar Sesiones***




**Flujo de Suceso:**

Este caso de uso es iniciado con la aplicación, se define el identificador del usuario, el cual se envía al servidor del juego, el cual retorna el estado del tablero de juego actual. En su defecto, envía una petición para que se definan las preferencias, esto se realiza mediante la interfaz específica. Una vez definidas las preferencias, se envían al servidor como las preferencias de la sesión de juego.

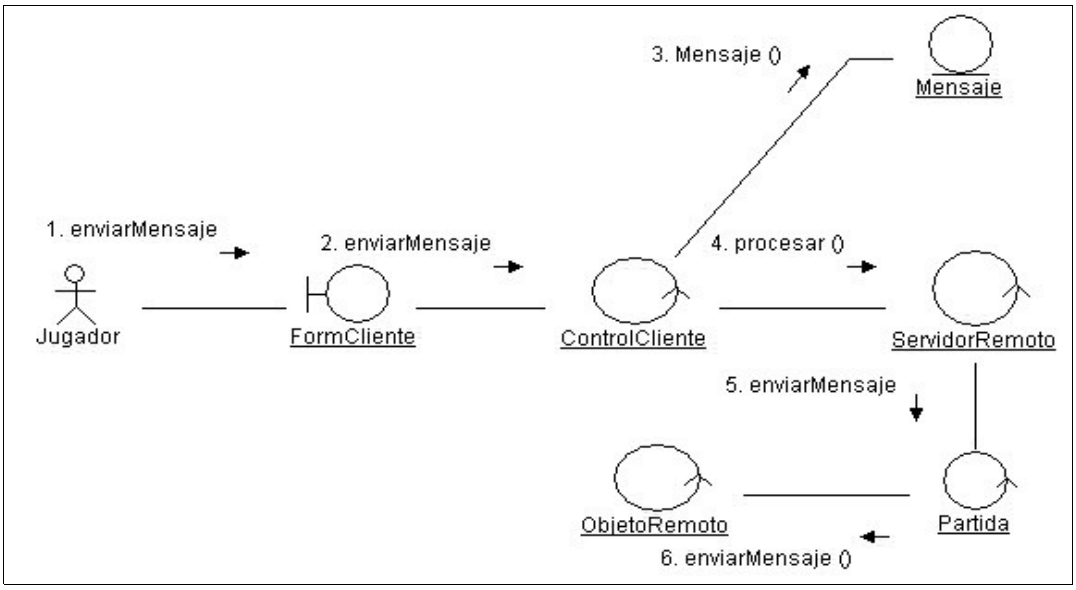
***Caso de Uso : Gestionar Movida***




**Flujo de Suceso:**

Este caso de uso manipula las acciones de agarrar, mover y soltar una pieza del tablero, por lo que cede el control del envío de mensajes al control respectivo.

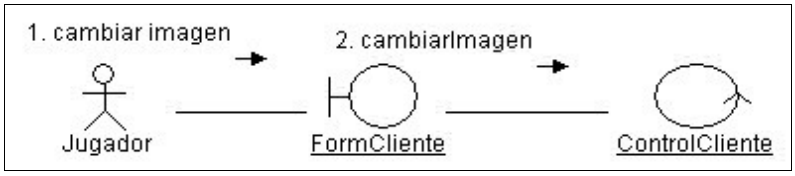
***Caso de Uso : Gestionar mensajes***




**Flujo de Suceso:**

El jugador remitente del mensaje, selecciona de entre la lista de jugadores aquel al que quiera enviar un mensaje, automáticamente se visualizara una lista de mensajes posibles, de entre los cuales se debe seleccionar alguno.

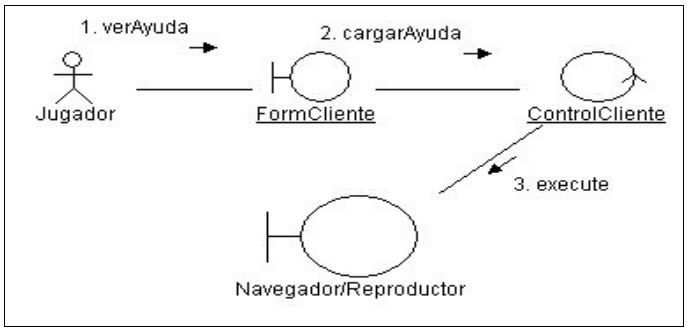
***Caso de Uso : Cambiar imagen***




**Flujo de Suceso:**

El jugador con una partida iniciada, selecciona la acción de abandonar el juego, presionando el botón correspondiente o cerrando la aplicación, a lo que automáticamente la aplicación iniciará un procedimiento de salida de la partida.

***Caso de Uso : Consultar ayuda***




**Flujo de Suceso:**

El jugador, selecciona el botón apropiado, y se visualiza la ayuda en un navegador o reproductor externo.

## 1.4 Análisis de Clases

### *1.4.1 Clases Interfaz*

**FormLogin**

| | |
|---|---|
| Nombre | FormLogin |
| Tipo | Formulario/Diálogo |
| Propósito | Ingresar o definir el identificador de usuario |
| Atributos | Identificador |
| Operaciones | Aceptar, cancelar y salir |
| Observaciones | |

**FormConfig**

| | |
|---|---|
| Nombre | FormConfig |
| Tipo | Formulario/Diálogo |
| Propósito | Ingresar o definir la dirección del servidor |
| Atributos | Campo de texto dirección |
| Operaciones | Aceptar, cancelar y salir |
| Observaciones | |

**FormPreferencias**

| | |
|---|---|
| Nombre | FormPrerencias |
| Tipo | Formulario/Diálogo |
| Propósito | Ingresar o definir las preferencias de una partida |
| Atributos | Selector de dificultad, selector de imagen. |
| Operaciones | Aceptar |
| Observaciones | |

**FormCliente**

| | |
|---|---|
| Nombre | FormCliente |
| Tipo | Formuario |
| Propósito | Permitir la interacción con juego, visualizar a los demás jugadores y permitir la interacción con ellos. |
| Atributos | Panel de tablero, lista de clientes, panel de opciones |
| Operaciones | Enviar mensaje a jugador, salir de la aplicación, consultar ayuda |
| Observaciones | |

**FormTablero**

| | |
|---|---|
| Nombre | FormTablero |
| Tipo | Panel/Diálogo/Componente |
| Propósito | Permitir la interacción con el jugador, visualizando las piezas del rompecabezas, y permitiendo su manipulación |
| Atributos | Tablero de juego, imagen de juego |

| | |
|---|---|
| Operaciones | Agarrar , mover y soltar pieza, lanzar evento. |
| Observaciones | |

### *1.4.2 Clases de Control*

**ControlConfig**

| | |
|---|---|
| Nombre | ControlConfig |
| Propósito | Administrar las configuraciones de la aplicación, como las direcciones y opciones del juego. |
| Entradas | Dirección del servidor |

**ControlCliente**

| | |
|---|---|
| Nombre | ControlCliente |
| Propósito | Manipular la lógica de comunicación entre el cliente y el servidor, manipular el juego y sus estados, manejar la recepción y envío de mensajes al servidor. Esta es la clase principal de la aplicación cliente. |
| Entradas | Identificador de usuario |

**ObjetoRemoto**

| | |
|---|---|
| Nombre | ObjetoRemoto |
| Propósito | Fungir como nexo de comunicación entre el servidor de juego, y el cliente, mediante la recepción de mensajes y la generación de eventos. |
| Entradas | Observador de eventos |

**ObjetoRemotoImp**

| | |
|---|---|
| Nombre | ObjetoRemotoImp |
| Propósito | Implementar a la interfaz objetoRemoto con todos sus metodos |
| Entradas | Observador de eventos |

**ServidorRemoto**

| Nombre | ServidorRemoto |
|---|---|
| Propósito | Objeto remoto persistente en algún servidor, encargado de la conexión y el procesamiento de mensajes enviados por los clientes. Esta es la clase accesible para todos los jugadores potenciales. |
| Entradas | Nombre de usuario y objeto remoto |

**ServidorRemotoImp**

| Nombre | ServidorRemotoImp |
|---|---|
| Propósito | Implementar a la interfaz ServidorRemoto con todos sus metodos |
| Entradas | Nombre de usuario y objeto remoto |

**Partida**

| Nombre | Sesion |
|---|---|
| Propósito | Clase que manipula los datos y la comunicación de mensajes entre jugadores, contiene a los jugadores, las preferencias de la partida y el tablero correspondiente. |
| Entradas | Nombre de usuario y objeto remoto |

**Evento**

| Nombre | Evento |
|---|---|
| Propósito | Clase que se instancia una vez generado un evento particular, lanzado desde un ObjetoEscuchable |
| Entradas | Contenido del evento |

**EventoListener**

| Nombre | EventoListener |
|---|---|
| Propósito | Interfaz que debe implementar toda clase que necesite escuchar los eventos generados en un ObjetoEscuchable |

| | |
|---|---|
| Entradas | |

**ObjetoEscuchable**

| | |
|---|---|
| Nombre | ObjetoEscuchable |
| Propósito | Interfaz que define los métodos a implementar por cualquier clase que quiera generar eventos particulares |
| Entradas | |

### *1.4.3 Clases Entidad*

**DatosJugador**

| | |
|---|---|
| Nombre | DatosJugador |
| Responsabilidad | Almacenar los datos referentes a cada jugador registrado en el servidor del juego, incluido el objeto remoto correspondiente. |
| Atributos | Identificador, objeto remoto, movidas acertadas, movidas erradas. |
| Relación | Sesion |

**Mensaje**

| | |
|---|---|
| Nombre | Mensaje |
| Responsabilidad | Tipo de dato a enviar para toda comunicación, es la unidad básica de transferencia entre los diferentes actores de la aplicación. |
| Atributos | Identificador, dato |
| Relación | Ninguna |

**MatrizDispersa**

| | |
|---|---|
| Nombre | MatrizDispersa |
| Responsabilidad | Almacenar los datos en una estructura de matriz simulada, para aumentar la eficiencia |
| Atributos | Cantidad de filas, columnas, vector de datos |
| Relación | Ninguna |

**Tablero**

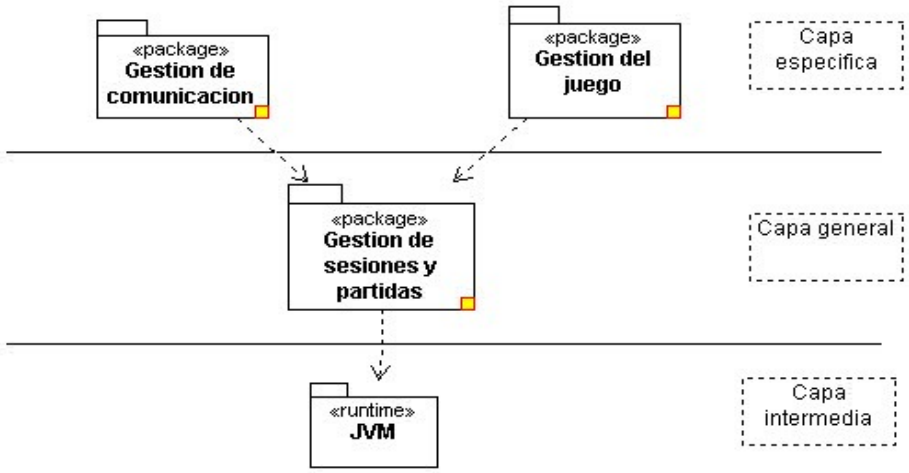
| Nombre | Tablero |
|---|---|
| Responsabilidad | Manipular los movimientos entre dos matrices de datos, una matriz identifica a los elementos no colocados, y la otra con los elementos ya colocados. |
| Atributos | Matriz de elementos no colocados, matriz de elementos colocados |
| Relación | Ninguna |

**Movida**

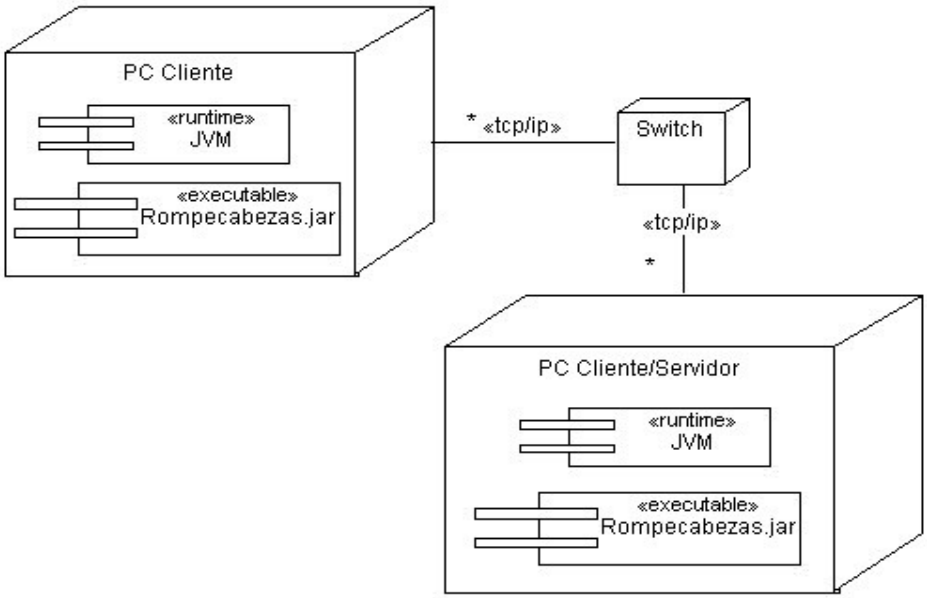
| Nombre | Movida |
|---|---|
| Responsabilidad | Tipo de dato que contiene las posiciones respectivas de una movida realizada por el usuario. |
| Atributos | Posición origen y destino |
| Relación | Ninguna |

# DISEÑO

## *1 Diseño de la Arquitectura*

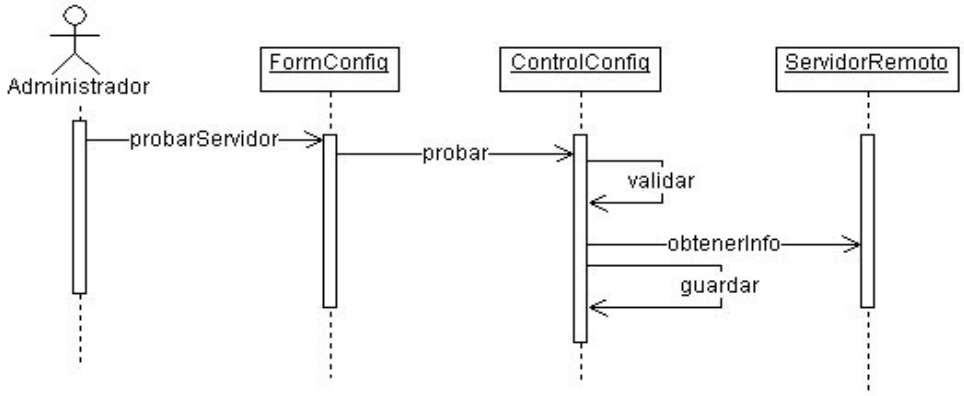


## *2 Diseño de la Arquitectura( Despliegue )*

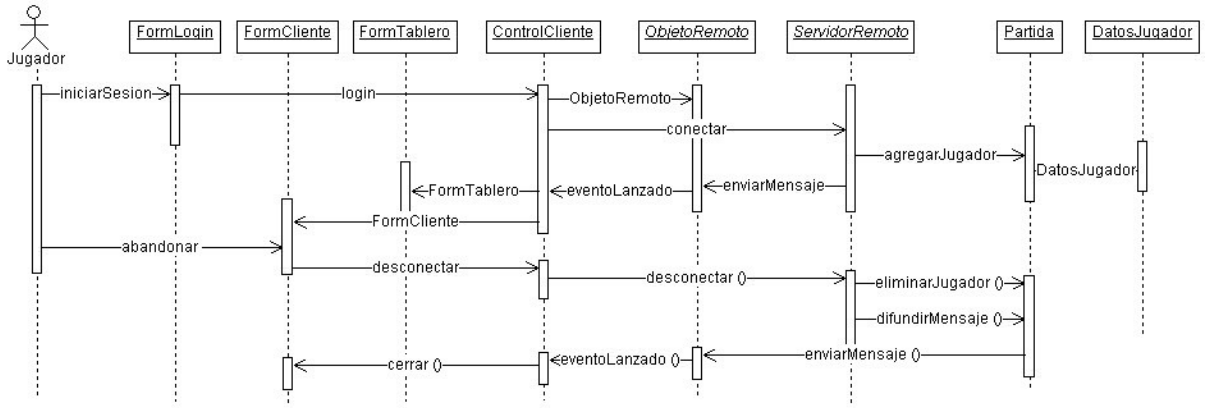


## *3 Diseño de Casos de Uso*

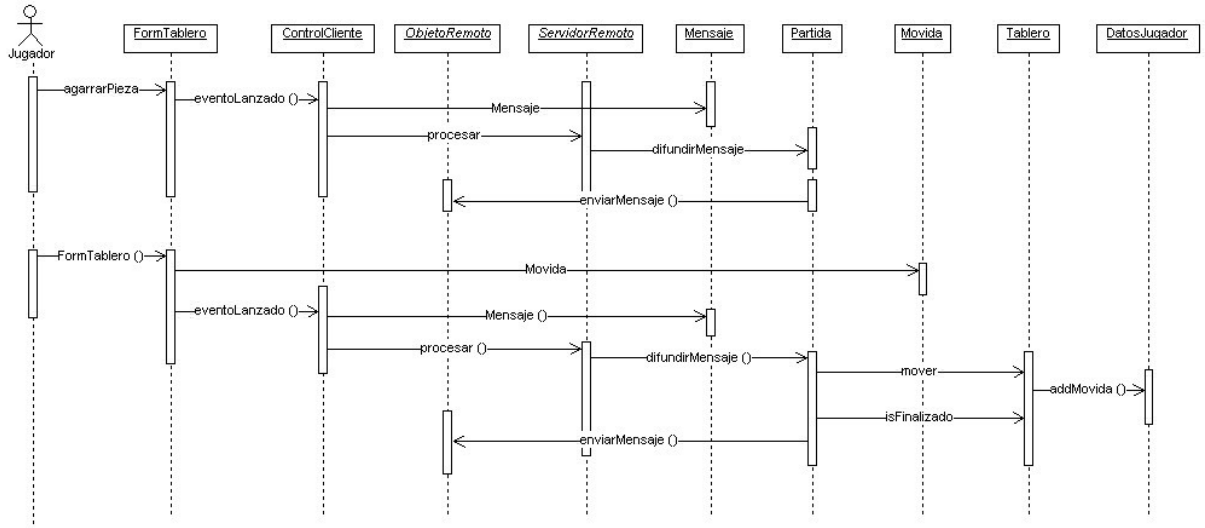
### Caso de Uso: Seleccionar servidor



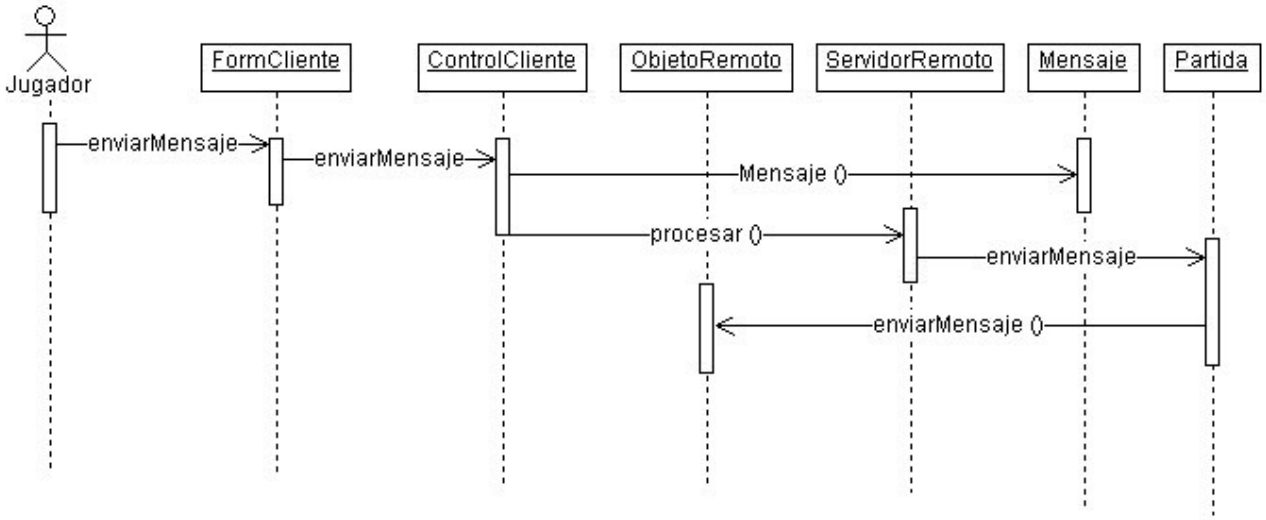
## Caso de Uso: Gestionar sesiones



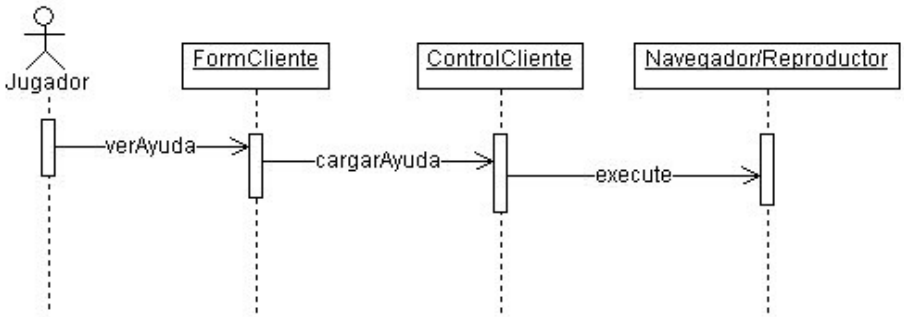

## Caso de Uso : Gestionar movidas



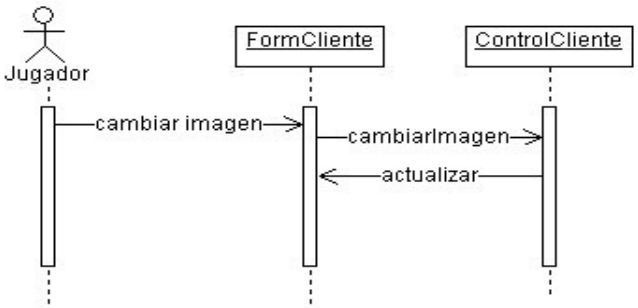
**Caso de Uso : Gestionar mensajes**




**Caso de Uso : Consultar ayuda**

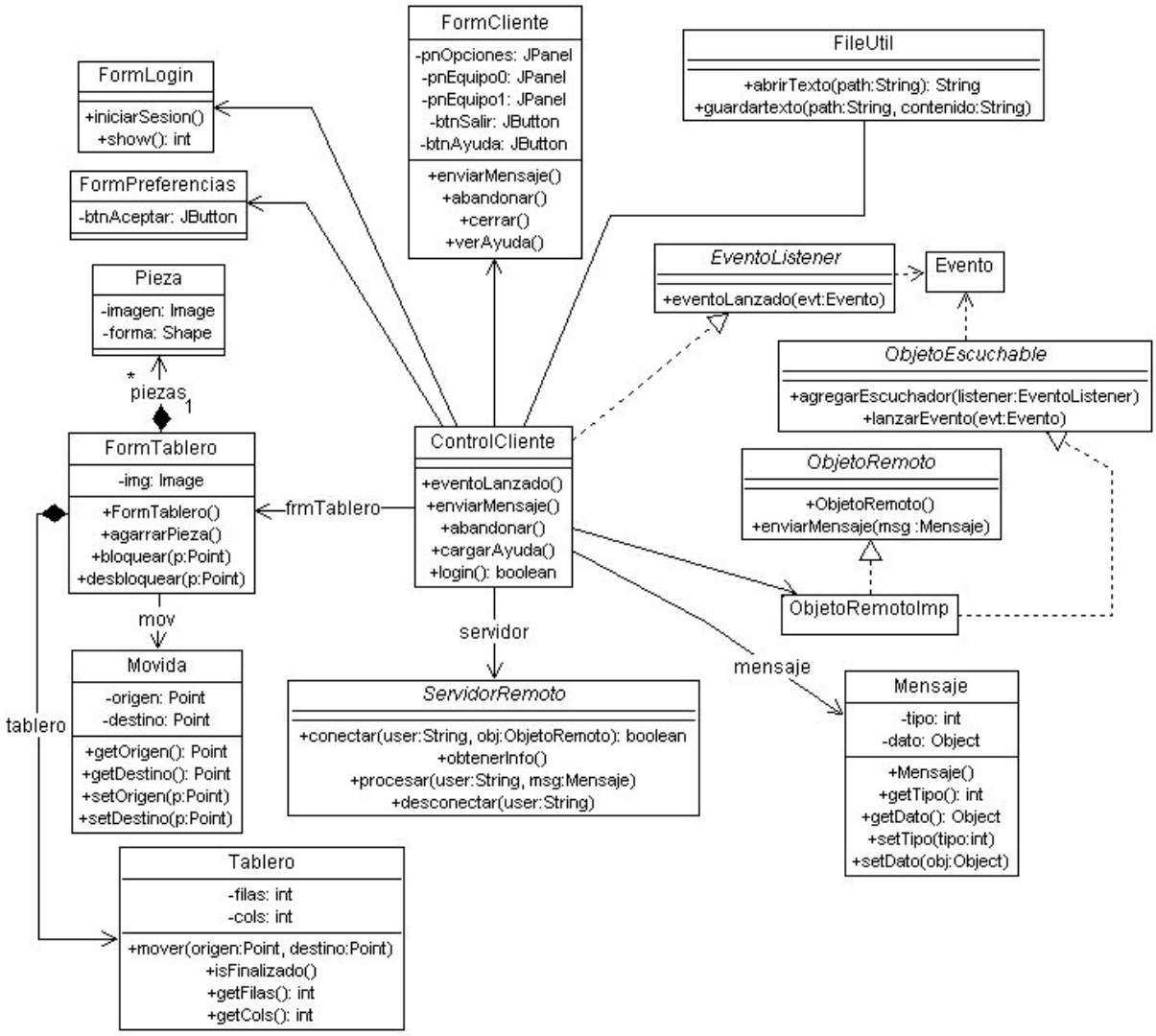



**Caso de Uso : Cambiar imagen**

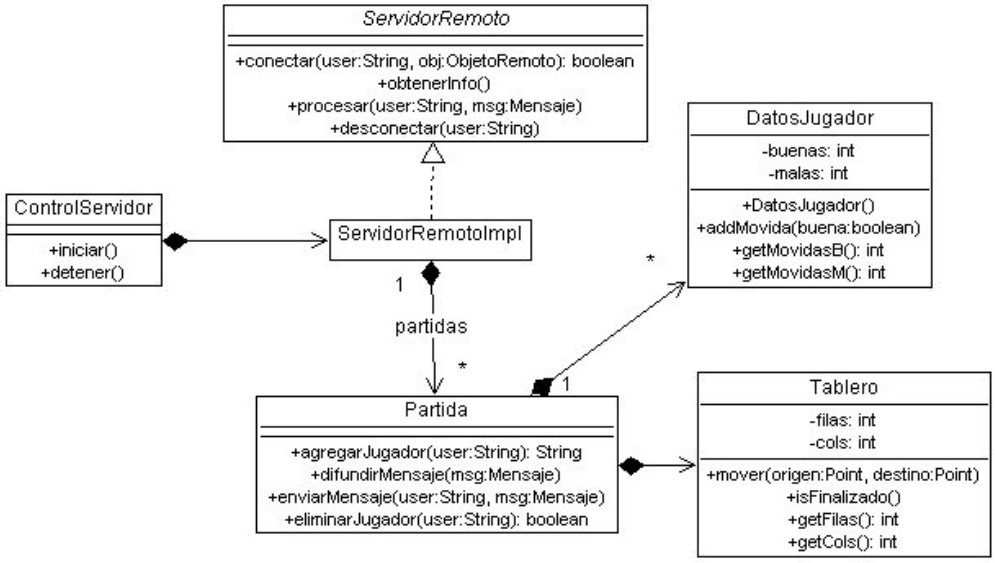


# *4 Diseño de Clases*

## 4.1 Diseño de clases ( cliente )

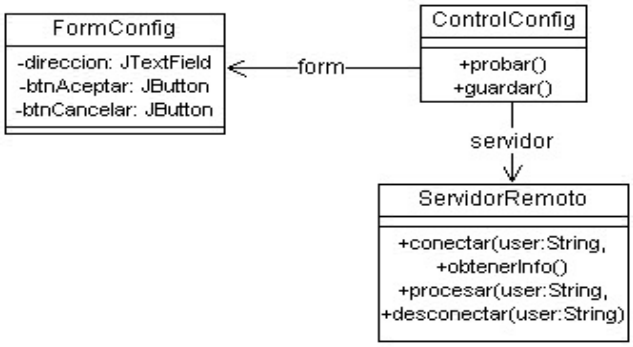


## 4.2 Diseño de clases ( servidor )

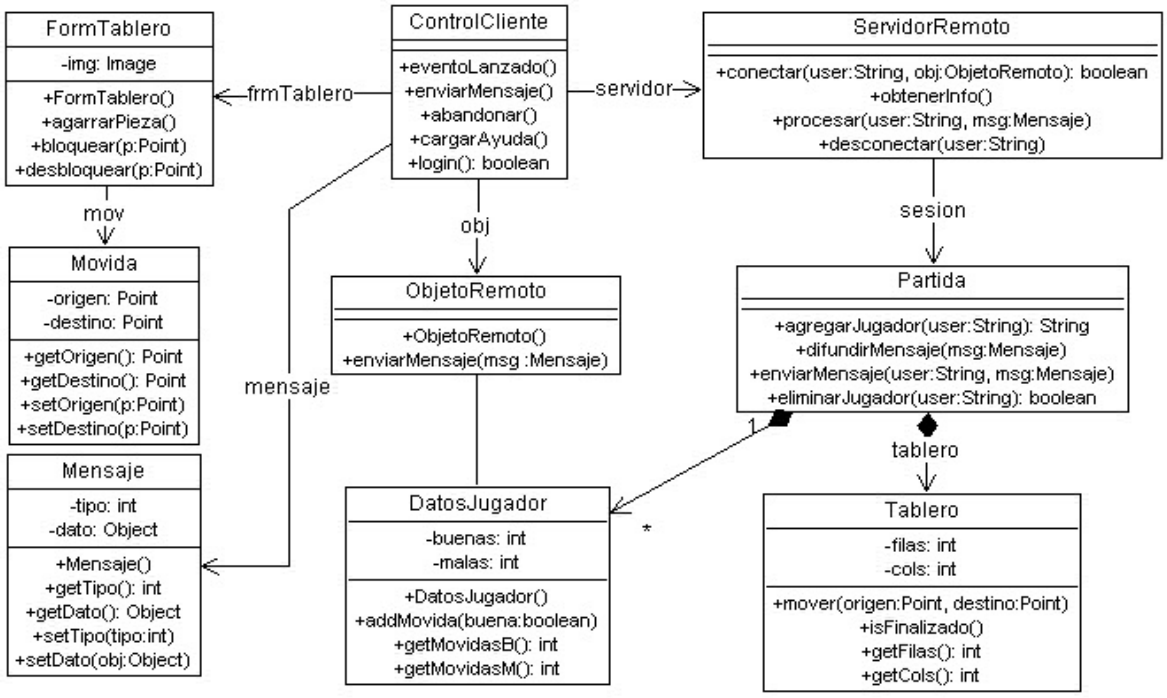



## 4.3 Diseño de clases ( por detalle de caso de uso)

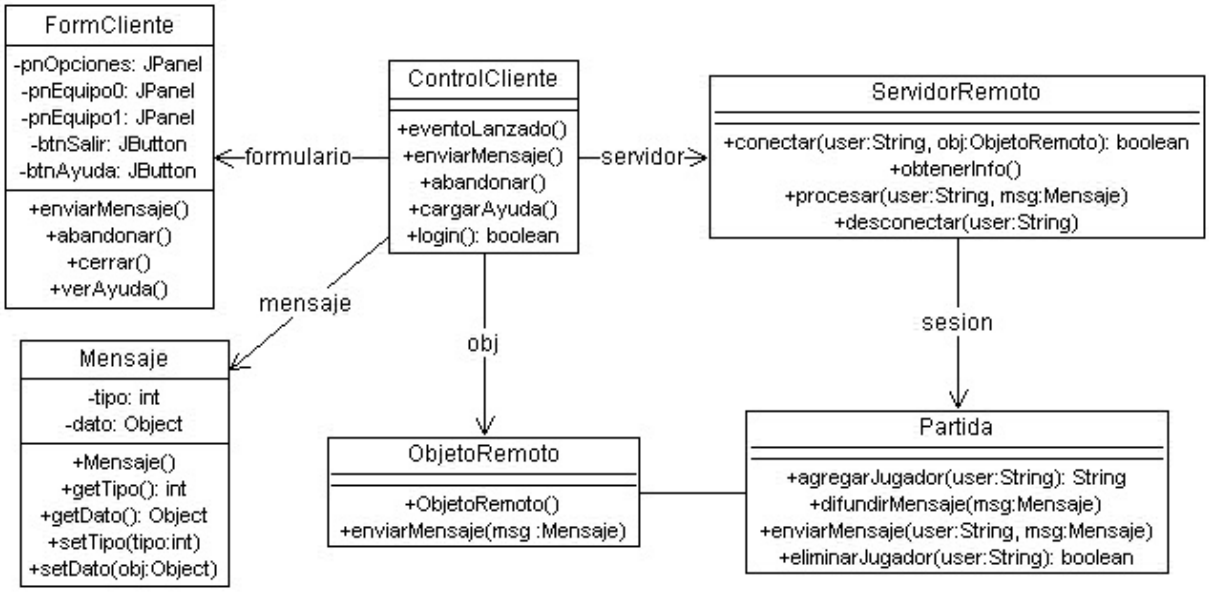
## Caso de Uso: Seleccionar Servidor



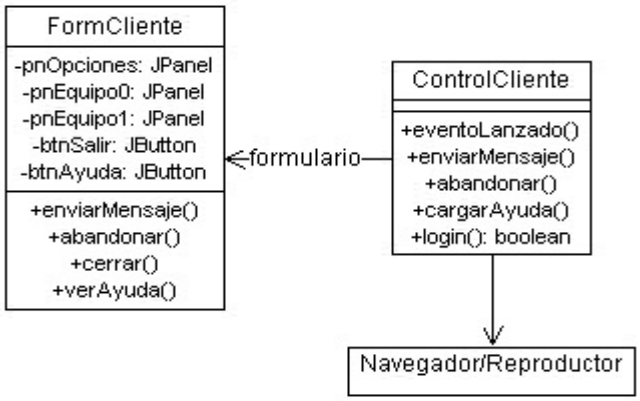
## Caso de Uso: Gestionar Movidas




## Caso de Uso: Gestionar mensajes



## Caso de Uso: Consultar ayuda

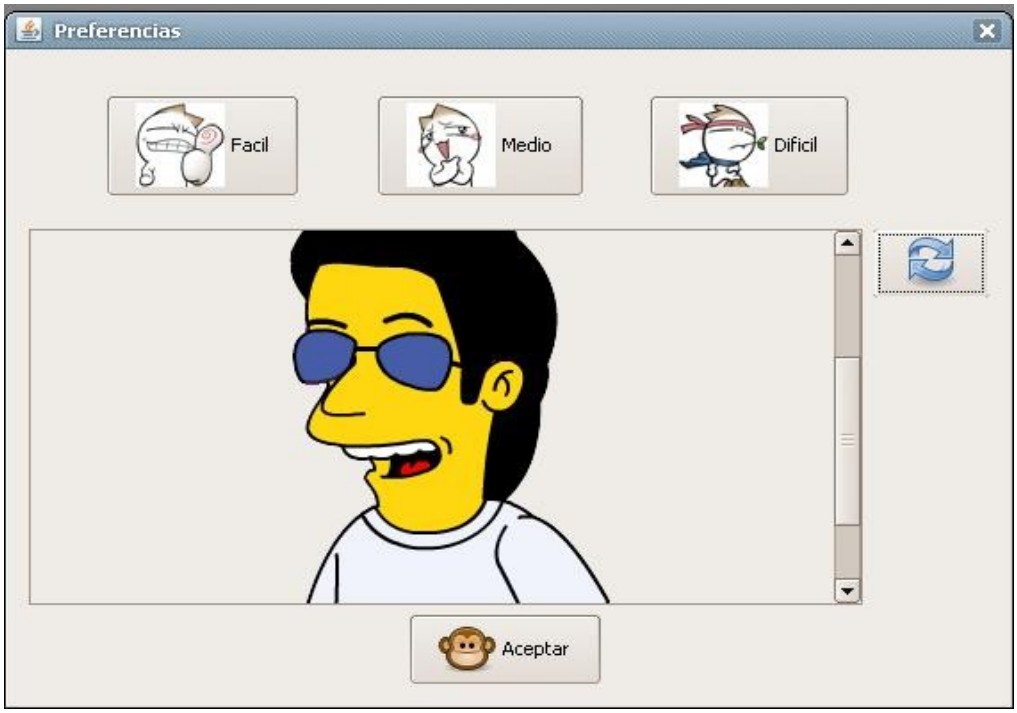


# *5 Diseño de Interfaces*

### Plantilla de configuración del servidor


### Plantillas de inicio de sesión (selección de identificador)

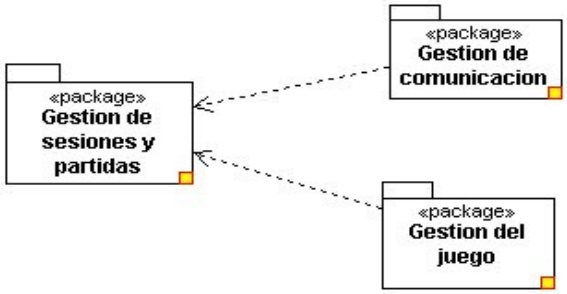
**Plantillas de inicio de sesión(definición de preferencias)**




**Plantilla principal de la aplicación**

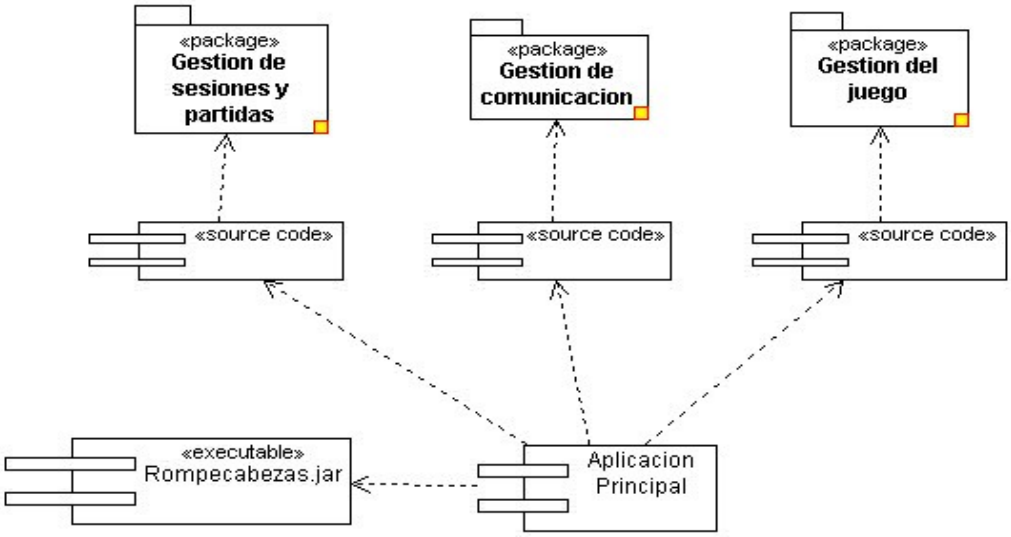
### *6 Diseño de SubSistemas*

**Relaciones entre subsistemas (cohesión y acoplamiento)**



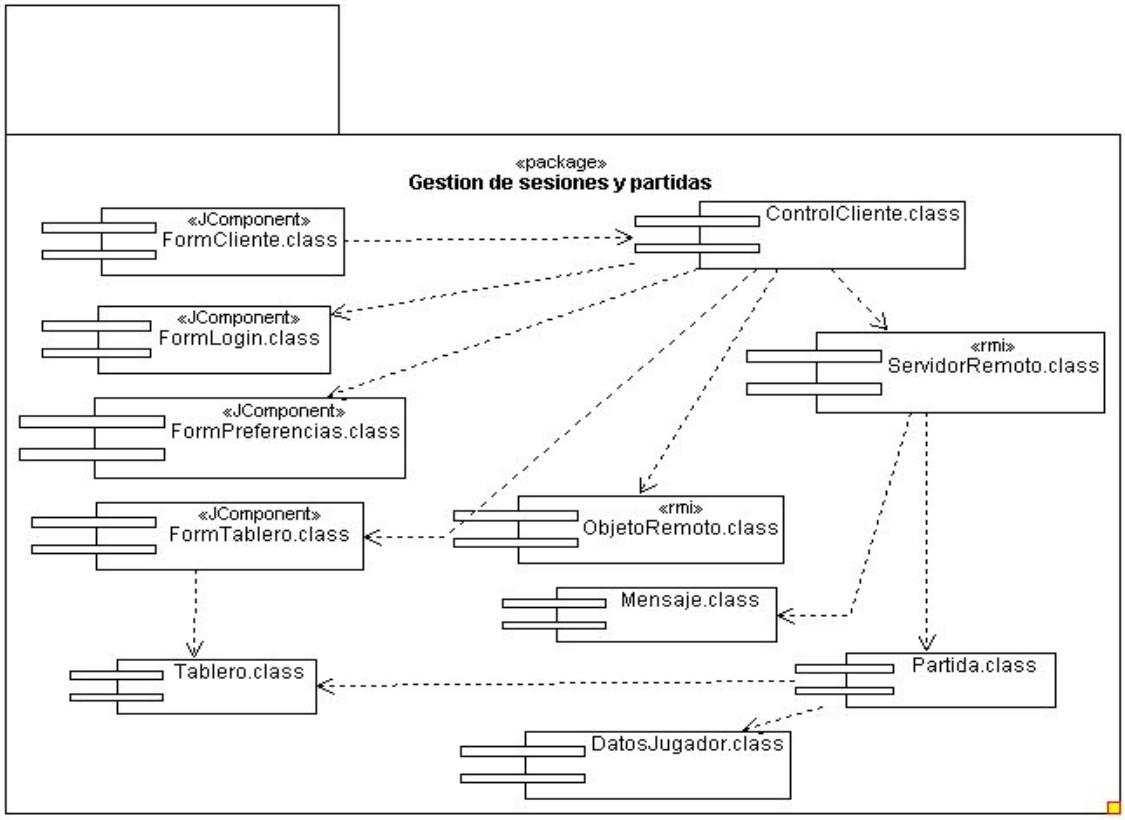

# IMPLEMENTACIÓN

### *1 Implementación de la Arquitectura*

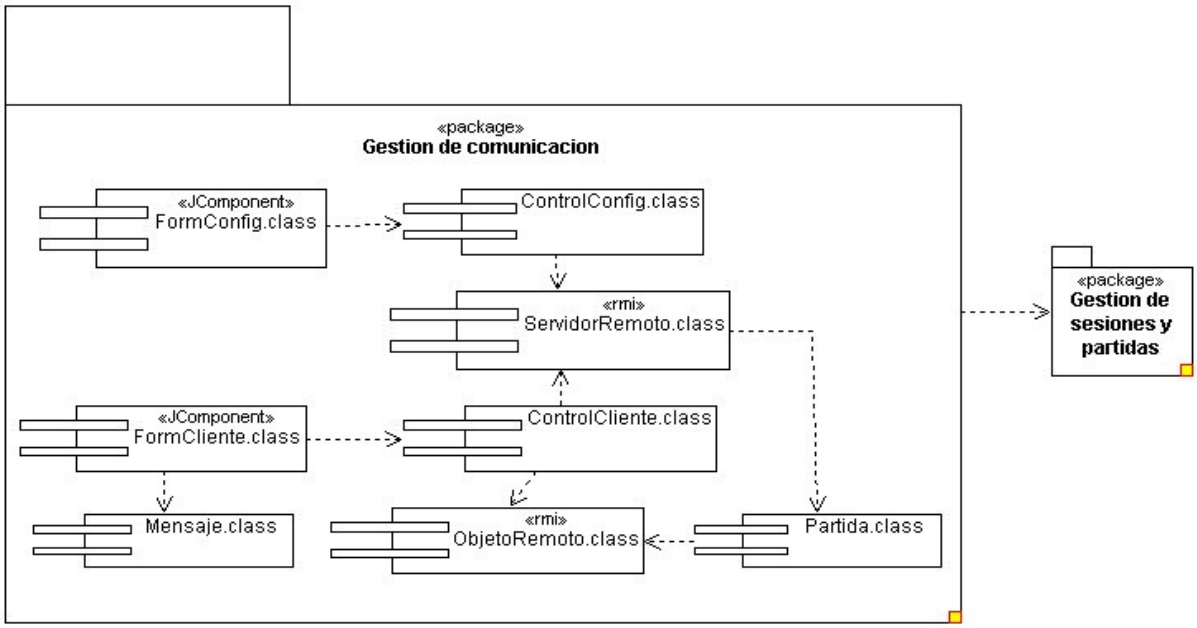


# *2 Implementación de Subsistemas*

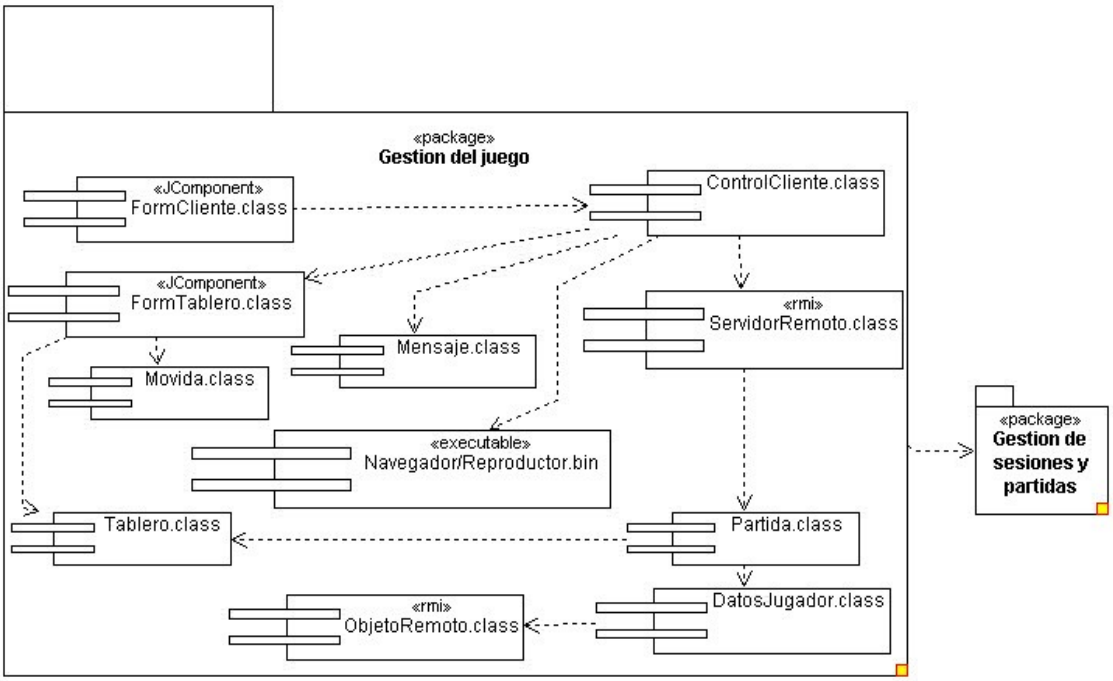
## Gestión de sesiones y partidas




## Gestión de comunicación




## Gestión del juego



# *3 Plataforma de Desarrollo*

## Sistema Operativo

El desarrollo de la aplicación se hizo utilizando los sistemas operativos:

- Microsoft Windows XP Service Pack 2
- Slackware Linux 12.0

## Entorno de Desarrollo

- Java Development Kit 1.6
- NetBeans IDE 5.5
- Javadoc 1.6

## Herramientas Complementarias

- UML Studio 6
- OpenOffice Writer 2.0

## Requerimientos de Hardware

Para la correcta ejecución de la aplicación, se requiere como mínimo:

- Procesador x86 de 1000 Mhz
- 256 MB en memoria primaria(RAM) y 10 MB en secundaria(disco duro)
- Tarjeta NIC(de red) con soporte nativo para el estándar Ethernet

## Escenario de trabajo

La aplicación se ejecutará tanto en un ordenador personal, como en una red de ordenadores de área local, entre todos aquellos que cumplan con los requerimientos mínimos previamente establecidos.

# PRUEBAS

El modelo de pruebas usado en este sistema es el alfa, dado que solo fue probado por los implementadores y pocos usuarios, por lo que el flujo de pruebas se omite.

# CONCLUSIONES

El desarrollo de una aplicación para entornos de red se facilita con la utilizzación de herramientas avanzadas, como RMI, ademas que existen múltiples fuentes de ayuda para la realización de este tipo de proyectos.

Debido a las limitaciones de los clientes potenciales, la aplicación se desarrolló con la mínima necesidad de configuración, y conocimientos.

Por las pruebas realizadas se puede concluir que la dificultad no esta en el uso de la aplicacion, sino en el entendimiento de la misma.

Por lo que se prevee que los cambios en las proximas versiones requerirá una mayor participacion de los clientes.

## Recomendaciones

La instalación y configuración de la aplicación, a pesar de ser muy básica, requiere del apoyo de algun tutor o persona mayor, ya que requiere de ciertos conocimientos.

El apoyo de personas mayores o tutores facilitará el aprendizaje del uso de la aplicacion, de sus bondades y limitaciones.

## Glosario

**RMI**: Invocación de Método Remoto

**PUDS**: Proceso Unificado de Desarrollo de Software

**UML**: Lenguaje Unificado de Modelado

**JRMP**: Protocolo de Método Remoto

**NIC**: Tarjeta de interfaz de red

# Bibliografía


- **Ingeniería del Software**, Roger Pressman 4º edición.
- **JFC Swing Tutorial, The: A Guide to Constructing GUIs, Second Edition**, Kathy Walrath, Mary Campione, Alison Huml, Sharon Zakhour.
- **Java Network Programming**, 3rd Edicion, Elliotte Rusty Harold
- **Thinking in Java** , 2º Edicion, Mattias Ettrich.


# Anexo

## Estandar de Codificación

El estandar de codificación utilizado, es el definido por la empresa Sun Microsystems Inc., de fecha 1997: http://www.sun.com/java/docs/codeconventions.html

## Estandar de Documentación

El estandar de documentación de codificación, es el establecido por la comunidad de desarrolladores de Java, Javadoc, con todas sus definiciones: http://www.sun.com/java/docs/codeconventions-doc.html