Reto 3

Actualización de diseño e implementación del Triqui RMI Cliente/Servidor hacia arquitectura P2P

23/10/2012

UNIVERSIDAD EAFIT

Luisa Fernanda Querubín Osorio

Jorman Andrés Bustos Gómez

# Antecedentes del Reto3:

* Se entrega el diseño e implementación de un juego sencillo del triqui, en arquitectura cliente/servidor y con ambos jugadores desde un mismo cliente.
* El código fuente (TriquiRMI.zip), se encuentra en Eafit Interactiva.
* También se tomará como base, el desarrollo realizado en el reto2, en el cual, se distribuyeron los 2 jugadores en máquinas cliente diferente.
* Mirar el estilo de arquitectura utilizado en el EchoServiceP2P, del cual también se entregan los códigos fuentes en eafit interactiva.

# Requerimientos del Reto3:

A continuación se presentan los requerimientos del reto 3:

1. Realizar el diseño del Triqui en la arquitectura P2P hibrida con servidor. El diseño debe considerar que los jugadores están en máquinas diferentes para un mismo juego.
2. Implementar el Triqui bajo la arquitectura P2P híbrida con servidor utilizando JavaRMI o similar.
3. Realizar el diseño del Triqui en la arquitectura P2P pura.
4. Implementar el Triqui bajo la arquitectura P2P pura utilizando un middleware/framework p2p puro como JXTA, P2P2, etc.

# Requisitos funcionales y no funcionales del sistema distribuido

Requisitos Funcionales:

* Elementos en una red de datos.
* La comunicación de estos debe ser por el paso de mensajes.

Requisitos No funcionales:

* Concurrencia: Varias personas  (usuarios) pueden acceder al mismo tiempo a la aplicación del "triqui" sin afectar el rendimiento de este. El servicio que este va a ofrecer un servicio multiusuario puesto que debe atender varios jugadores y es al mismo tiempo concurrente puesto que varias parejas pueden jugar al mismo tiempo.
* Transparencia: Los usuarios no pueden ver la lógica del negocio ya que esta se aísla del cliente y el cliente solo interactúa con la interfaz.
* Tolerancia a fallas: el sistema no es tolerante a fallos ya que si se le cae un proceso a un cliente el otro cliente tiene que iniciar un juego con otro.
* Escalable: El sistema es escalable ya que se le puede ir aumentando clientes, funcionalidades, etc. y esto no afectara su rendimiento.
* Seguro: Al mandar mensajes entre clientes y con el servidor estos no sean inferidos o alterados.

# Triqui con Arquitectura P2P hibrida

# Requisitos Funcionales y no Funcionales de la aplicación

Servicios y Requisitos Funcionales:

* Buscar Pareja
* Crear Tablero
* Iniciar juego
* Asignación de símbolo
* Realizar movimiento (Marcar casilla con “X” o “O”)
* Validar movimiento
* Anunciar ganador
* Reiniciar Partida
* Esperar por pareja

Requisitos no funcionales:

* Usabilidad: El diseño de la aplicación es muy amigable, lo que garantiza la fácil comprensión y manejo del juego.
* Eficiencia: Los tiempos de acceso y actualización son mínimos ya que la aplicación está diseñada para funcionar en tiempo real.
* Portabilidad: La aplicación debe funcionar en cualquier ambiente siempre y cuando se tenga acceso a la red.
* Escalabilidad: La aplicación debe soportar múltiples usuarios jugando de manera simultánea.

# Entidades del sistema

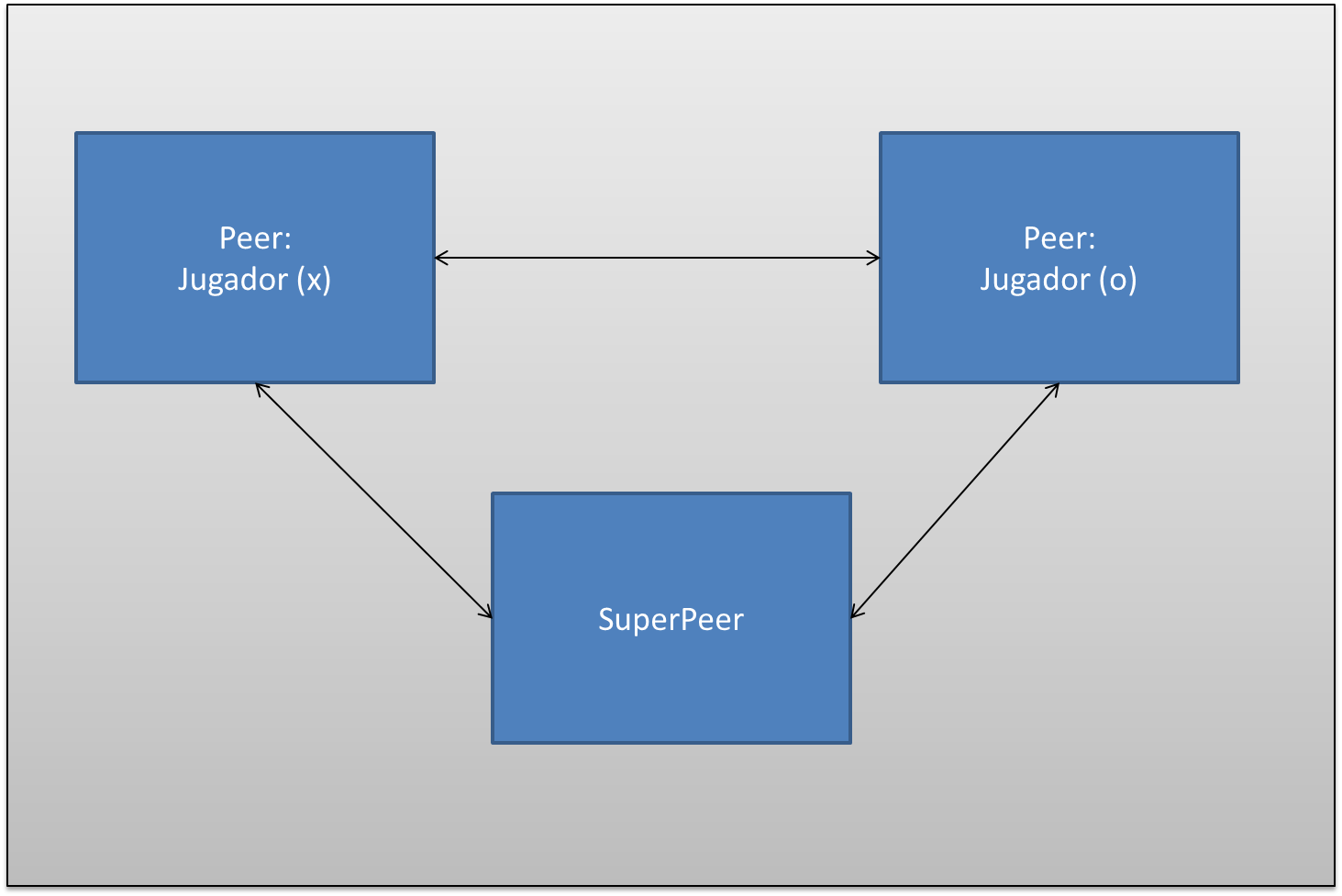


Figura 1: Diagrama de Clases.

* Peer: Jugador 1(X), Jugador 2 (O)
* Superpeer

# Modelo de interacciones

* Modelo P2P
* Variante: Arquitectura P2P hibrida.

# Servicios de las entidades del sistema distribuido (interfaces)

ENTIDAD ´SuperPeer´

* RegistrarPareja (id\_peer1, id\_peer2)
* RegistrarPeer(id\_peer)

ENTIDAD ‘Peer’:

* RegistrarJugada (id\_cliente, jugada)
* PublicarJugada (Jugada)
* EstablecerConexion(id\_peer)
* RecibirJugada (jugada, id\_peer)
* VerificarJugada (jugada, id\_peer)

# Características de las interacciones

* Sincrónico: puesto que un peer ‘X’ no puede ejecutar su próxima jugada hasta que el peer ’O’ ejecute la suya.
* Asimétrico: puesto que el peer ‘X’ al realizar un request por parte del peer ‘O’ espera un response que es la ubicación del símbolo en el tablero.
* Manejo de estado: El triqui maneja estado puesto que en el turno del peer ‘X’ el estado de éste es activo, mientras que el estado de espera del peer ‘O’ es inactivo. El estado solo existe para la ejecución de cada instancia de juego nuevo.
* Manejo de sesión: El encolamiento por parte de los peers hace que al llegar un peer al tope de la cola el servidor pueda atender su solicitud e inicie una sesión de nuevo juego.
* Sincronización: Los peers se ponen de acuerdo para jugar en la manera en que se encuentre un ‘X’ con una ‘O’ en el tope del encolamiento. Luego el Superpeer notifica a los dos peers el inicio de la partida.

# Definición de middleware adecuado al diseño

Para el diseño del middleware usaremos invocación remota de procedimientos puesto que las llamadas de los métodos o funciones se realiza de manera sincrónica y el envió del mensaje se hace por medio de un marshalling que nos ayuda a estandarizar todos los datos o parámetros para que puedan ser enviados a la otra máquina.

# Triqui con Arquitectura P2P Pura

# Requisitos Funcionales y no Funcionales de la aplicación

Servicios y Requisitos Funcionales:

* Buscar Pareja
* Crear Tablero
* Iniciar juego
* Asignación de símbolo
* Realizar movimiento (Marcar casilla con “X” o “O”)
* Validar movimiento
* Anunciar ganador
* Reiniciar Partida
* Esperar por pareja

Requisitos no funcionales:

* Usabilidad: El diseño de la aplicación es muy amigable, lo que garantiza la fácil comprensión y manejo del juego.
* Eficiencia: Los tiempos de acceso y actualización son mínimos ya que la aplicación está diseñada para funcionar en tiempo real.
* Portabilidad: La aplicación debe funcionar en cualquier ambiente siempre y cuando se tenga acceso a la red.
* Escalabilidad: La aplicación debe soportar múltiples usuarios jugando de manera simultánea.

# Entidades del sistema

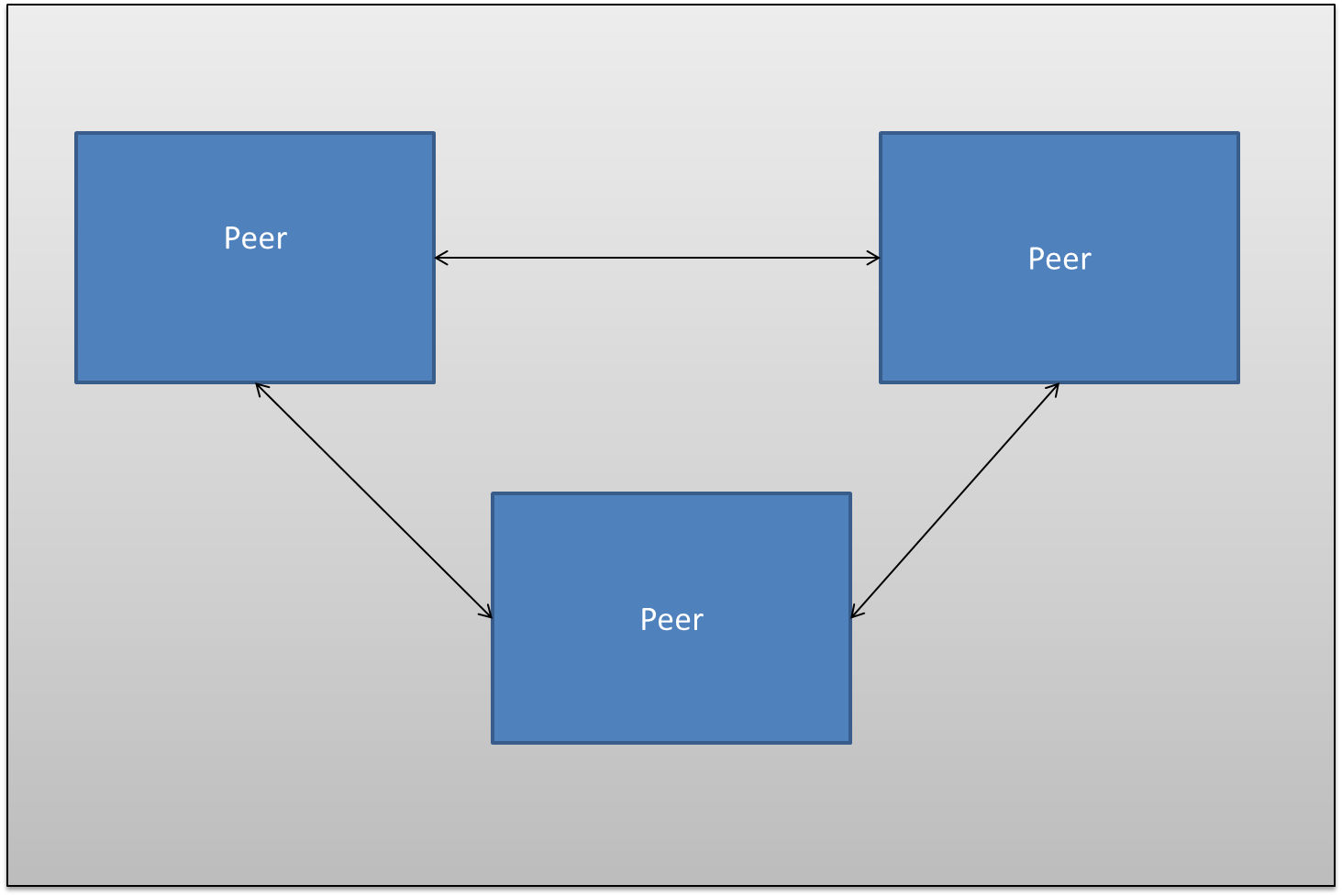


Figura 2: Diagrama de Clases.

* Peer

# Servicios de las entidades del sistema distribuido (interfaces)

ENTIDAD ‘Peer’:

* RegistrarPareja (id\_peer1, id\_peer2)
* RegistrarPeer(id\_peer)
* RegistrarJugada (id\_cliente, jugada)
* PublicarJugada (Jugada)
* EstablecerConexion(id\_peer)
* RecibirJugada (jugada, id\_peer)
* VerificarJugada (jugada, id\_peer)

\*El peer que compita con el símbolo “X” será el encargado de realizar las funciones que en el diseño anterior realizaba el superpeer.

# Características de las interacciones

1. Sincrónico: puesto que un peer ‘X’ no puede ejecutar su próxima jugada hasta que el peer ’O’ ejecute la suya.
2. Asimétrico: puesto que el peer ‘X’ al realizar un request por parte del peer ‘O’ espera un response que es la ubicación del símbolo en el tablero.
3. Manejo de estado: El triqui maneja estado puesto que en el turno del peer ‘X’ el estado de éste es activo, mientras que el estado de espera del peer ‘O’ es inactivo. El estado solo existe para la ejecución de cada instancia de juego nuevo.
4. Manejo de sesión: El encolamiento por parte de los peers hace que al llegar un peer al tope de la cola el servidor pueda atender su solicitud e inicie una sesión de nuevo juego.
5. Sincronización: Los peers se ponen de acuerdo para jugar en la manera en que se encuentre un ‘X’ con una ‘O’ en el tope del encolamiento. Luego uno de los dos peers notifica al otro el inicio de la partida.

# Definición de middleware adecuado al diseño

Para el diseño del middleware usaremos JXTA ya que se adapta a las necesidades de esta implementación y es el middleware más maduro para trabajar con p2p.