

## **Reto 2 Tópicos Especiales en Telemática**

Sebastián García Valencia  
Estudiante Ingeniería de Sistemas  
Universidad EAFIT  
Medellín, Colombia, Suramérica

### **Resumen**

El presente documento contiene el análisis y diseño de un triqui distribuido bajo arquitectura P2P.

### **Palabras clave.**

Sistema distribuido, P2P, triqui, java, RMI, sockets.

### **1. Introducción**

En el presente trabajo se pretende evolucionar un sistema preexistente con el objetivo de darle nuevas funcionalidades. Partiendo de un triqui que funciona por sockets en un modelo cliente servidor, se pretende llegar a unas arquitecturas p2p en la cual los jugadores son concurrentes y pueden jugar desde diferentes maquinas.

### **2. Requerimientos del Reto2:**

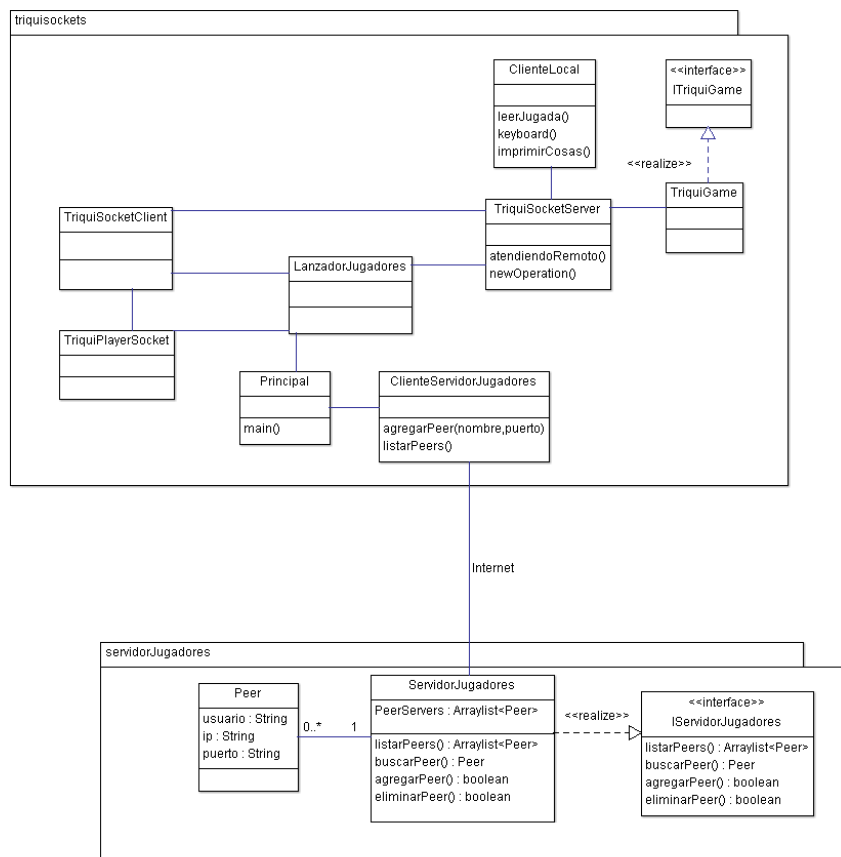
A continuación se presenta los requerimientos del reto 2, los cuales se sintetizan en los siguientes 3:

1. El Servidor debe atender a los diferentes clientes en modo concurrente, es decir, que permita atender varios juegos simultáneamente varias sesiones de juego remotas.
2. Realizar una modificación al diseño e implementación que permita a los 2 jugadores, ESTAR EN DOS CLIENTES O MAQUINAS DISTRIBUIDAS. Es decir, ya el TriquiPlayer solo conecta a un Jugador. Esto implica, que se deben adicionar funcionalidades al servidor para que permita GESTIONAR la conexión y los juegos entre los JUGADORES distribuidos. Este requerimiento es nuevo, y no existía en las versiones 1 y 2 del Triqui.
3. Realizar el diseño y la implementación en una arquitectura P2P, en la cual, el Cliente posea la lógica del juego, e implemente algún mecanismo para coordinar las acciones del juego (un solo juego, y los 2 clientes se conectan, uno local y el otro remoto. O dos juegos, los cuales deben

coordinar las acciones). Se recomienda utilizar una arquitectura P2P híbrida basada en Servidor, en el cual se implementen los servicios de registro, búsqueda y localización de Clientes. La funcionalidad de los servicios ofrecidos por el Servidor DEBEN SER IMPLEMENTADOS POR INVOCACIÓN REMOTA, no por Sockets.

### 3. Vista lógica

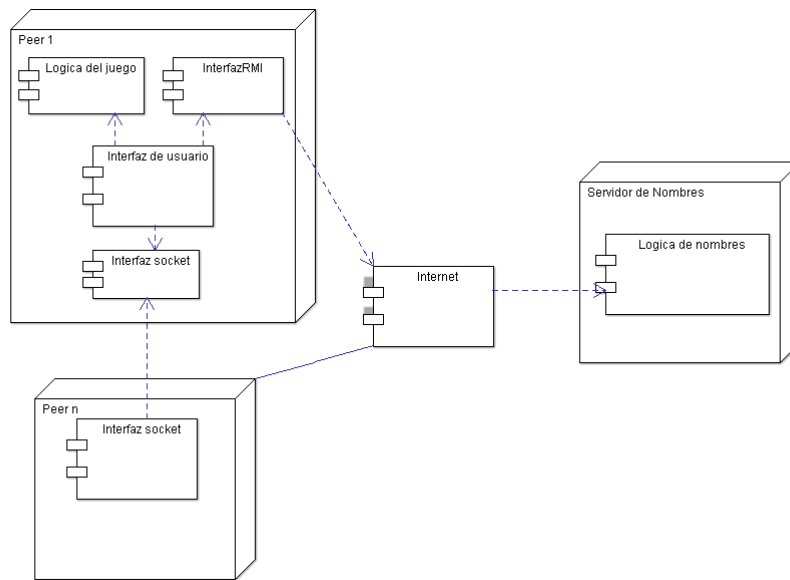
Como podemos observar el software se compone básicamente de dos grandes paquetes, el del servidor, donde se encuentra toda la funcionalidad a ser desplegada para poder invocar los métodos y obtener así la información de los jugadores y por otro lado el de los peers, donde se gestiona el ingreso de usuarios, el decidir quién tiene la lógica local y quien se apeg a esta, etc.



Para mayor detalle ver anexo 1.

### 4. Vista de despliegue

En la vista de despliegue notamos que lo único que estará en el servidor, es la lógica de la invocación remota por medio de la cual los jugadores encuentran otros jugadores, todo lo que es el juego de triqui está en cada uno de los peer, donde por determinados módulos se comunican entre ellos.

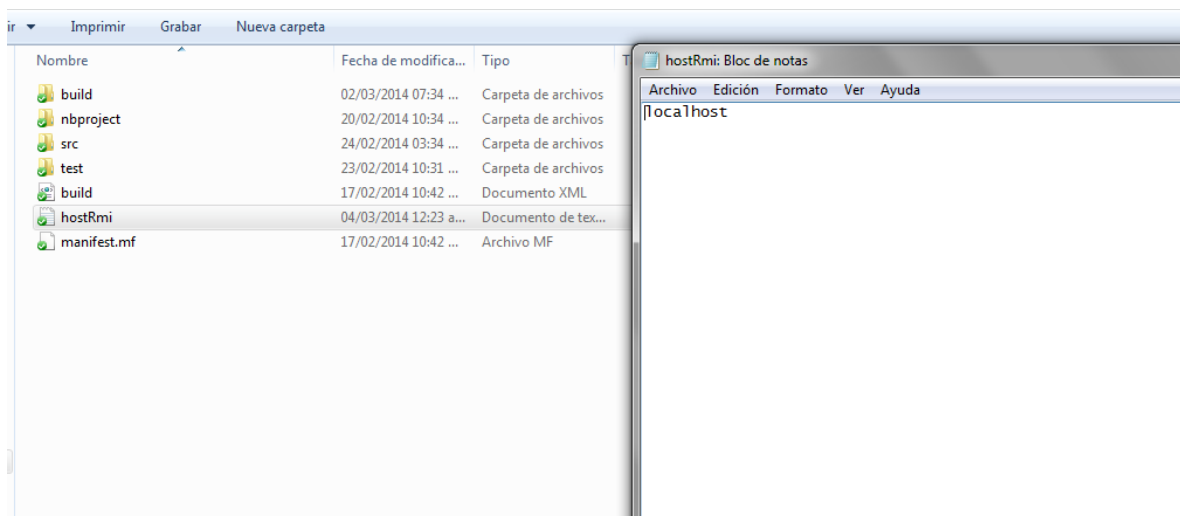


Para mayor detalle ver anexo 2.

## 5. Funcionamiento

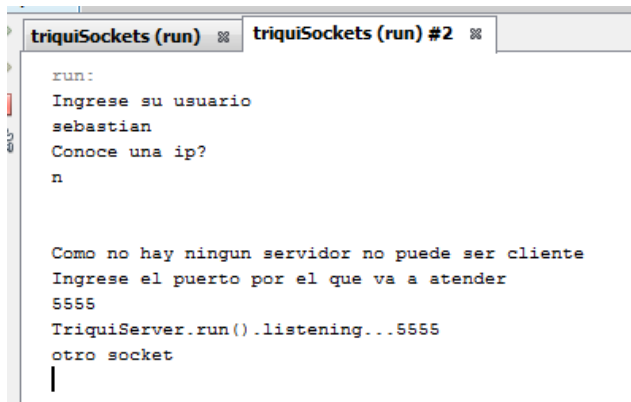
Antes que nada se debe lanzar el servidor de nombre donde sea que se desea, aunque aun sin este servidor es posible conectarse directamente conociendo la ip y el puerto como debe ser una arquitectura p2p.

En el archivo hostRmi.txt ubicado en la raíz del proyecto se debe escribir el host del servidor, de esta manera se comunica e invoca los métodos remotos. Para este ejemplo usamos el localhost para también podría ser una ip o algo como sistemas.eafit.edu.co



En primer lugar se debe ingresar el usuario, en caso de que ya se conozca una ip y puerto donde hay un peer servidor se puede ingresar directamente y de esta manera podemos jugar incluso si el servidor de nombres esta caído, este no es el caso.

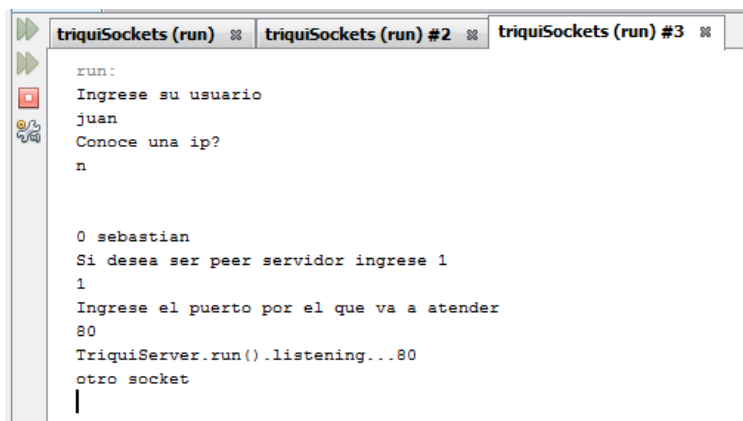
Como aun no hay ningún peer servidor nos vemos obligados a ser uno, así que debemos ingresar el puerto por el que vamos a atender.



```
triquiSockets (run)  triquiSockets (run) #2
run:
Ingrese su usuario
sebastian
Conoce una ip?
n

Como no hay ningun servidor no puede ser cliente
Ingrese el puerto por el que va a atender
5555
TriquiServer.run().listening...5555
otro socket
|
```

Este es otro peer, pero como ya hay un servidor se nos da la opción de ser peer servidor o peer cliente.



```
triquiSockets (run)  triquiSockets (run) #2  triquiSockets (run) #3
run:
Ingrese su usuario
juan
Conoce una ip?
n

0 sebastian
Si desea ser peer servidor ingrese 1
1
Ingrese el puerto por el que va a atender
80
TriquiServer.run().listening...80
otro socket
|
```

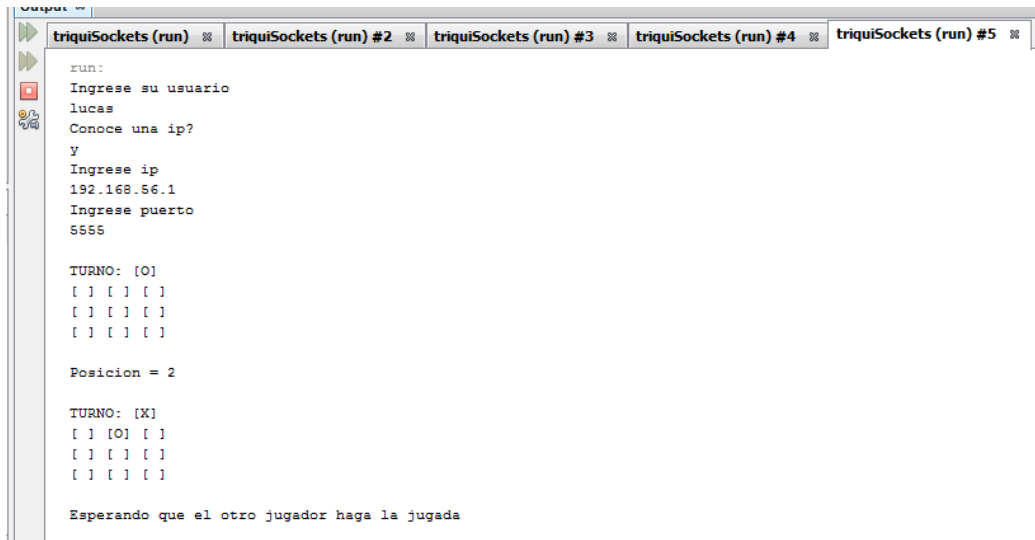
En este otro peer decidimos ser clientes, primero listamos los peer servidores disponibles y luego elegimos el deseado, y entonces empezamos a jugar directamente peer to peer.

```
Output %  
triquiSockets (run) % triquiSockets (run) #2 % triquiSockets (run) #3 %  
Si desea ser peer servidor ingrese 1  
2  
0 sebastian  
1 juan  
Ingrese el numero del peer servidor con el que quiere jugar  
0  
  
TURNO: [O]  
[ ] [ ] [ ]  
[ ] [ ] [ ]  
[ ] [ ] [ ]  
  
Posicion = 1  
  
TURNO: [X]  
[O] [ ] [ ]  
[ ] [ ] [ ]  
[ ] [ ] [ ]
```

Esta es la apariencia del servidor de nombres, aquí vemos que servidores hay, así como sus ip y puertos, información con la que los clientes pueden establecer conexión (para este ejemplo es la misma ip pues esta local, pero para los remotos también funciona).

```
Output %  
triquiSockets (run) % triquiSockets (run) #2 % triquiSockets (run) #3 % triquiSockets (run) #4 %  
run:  
El servidor esta listo  
  
sebastian 192.168.56.1 5555  
juan 192.168.56.1 80
```

Aquí tenemos otro peer el cual se conecta directamente a Sebastian sin intermediación del servidor de nombres, favoreciendo así la independencia y la arquitectura p2p.



```
run:
Ingrese su usuario
lucas
Conoce una ip?
Y
Ingrese ip
192.168.56.1
Ingrese puerto
5555

TURNO: [O]
[ ] [ ] [ ]
[ ] [ ] [ ]
[ ] [ ] [ ]

Posicion = 2

TURNO: [X]
[ ] [O] [ ]
[ ] [ ] [ ]
[ ] [ ] [ ]

Esperando que el otro jugador haga la jugada
```

## 6. Posibles mejoras.

- Se desea hacer la interfaz inicial en modo consola pues puede ser mas cómoda
- Cuan el peer tiene un servicio de virtualización como virtual box la ip que manda es la de la tarjeta emulada y no la real, esto se puede mejorar para que siempre tome la ip principal.
- En caso de que un peer servidor falle inesperadamente se puede hacer que los peers clientes también los puedan eliminar, es decir, si un peer cliente trata de jugar y no puede, que avise al servidor de nombres que ese peer servidor ya no funciona.

## 7. Enlace repositorio GIT

<https://bitbucket.org/sebasgverde/reto2toptelematica>

## 8. Bibliografía.

[http://es.wikipedia.org/wiki/Java\\_Remote\\_Method\\_Invocation](http://es.wikipedia.org/wiki/Java_Remote_Method_Invocation)

<http://cs.mty.itesm.mx/profesores/raul.perez/DAD/ejercicios/RMI/RMI-netbeans.pdf>

<http://download.java.net/jdk8/docs/technotes/guides/rmi/hello/hello-world.html>