Casualgram

Práctica de Grupo AS
16/17

Índice

1. Equipo de Desarrollo

Natalia IglesiasTorres	natalia.iglesiast@udc.es
Patricia Juane Borreguero	p.juane@udc.es
Guillermo Afonso Varela Chouciño	guillermo.vchoucino@udc.es
Luis Fernando Cruz Ramos	luis.cruz@udc.es
Roberto Insua Brandariz	roberto.insua@udc.es
Yago Fariña Budiño	yago.farina@udc.es
Luis Ogea Borbón	luis.ogea.borbon@udc.es
David De Castro Celard	david.decastro@udc.es
Cristian Díaz Hermida	cristian.diaz.hermida@udc.es

2. Descripción del Sistema

Casualgram es una aplicación de mensajería instantánea.
La aplicación permite que cualquier par de usuarios registrados en el servidor intercambien mensajes, por lo que es probable que el número de usuarios conectados al servidor, que deberá tener una alta disponibilidad, sea alto.
Además, el objetivo de la compañía es obtener un alto beneficio por la comercialización de la aplicación, por lo que la facilidad de uso es un factor importante de cara al usuario.
El sistema debe ser facilmente mantenible debido al escaso equipo de desarrollo, que no podrá mantener actualizaciones constantes.
La aplicación es multiplataforma, y funcionará desde cualquier dispositivo que disponga de conexión a internet, tanto un ordenador de sobremesa o un dispositivo móvil. Por lo tanto, la robustez y la fiabilidad del sistema son puntos a tener en cuenta para el sistema.
Los datos almacenados de los usuarios serán almacenados por la aplicación, y serán los que permitan que se envíen mensajes, por lo que es importante aportar alguna medida de seguridad al almacenar esta información de forma correcta.
Finalmente el envío y recepción de mensajes debe ser rápido y eficaz, para que el usuario no desespere y piense que el sistema está fallando.

Por lo que podemos deducir los siguientes **requisitos funcionales**:

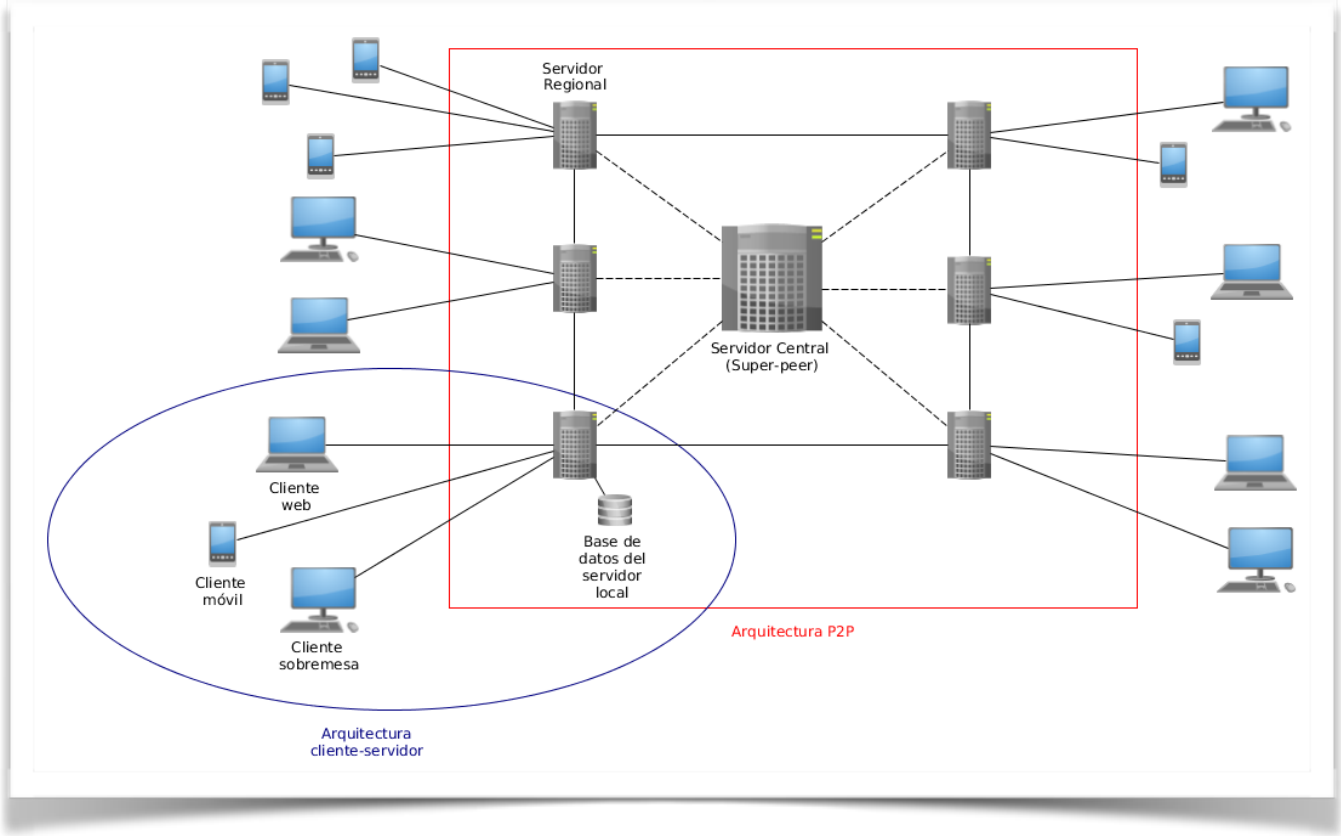
- Registro de usuarios no repetidos
- Conexión de usuarios con el servidor
- Envío de mensajes
- Desconexión de los usuarios

Y los siguientes **requisitos no funcionales**:

- Portabilidad (multiplataforma)
- Fiabilidad
- Disponibilidad
- Escalabilidad
- Robustez
- Concurrencia
- Facilidad de uso
- Facilidad de mantenimiento
- Seguridad
- Rapidez y facilidad en la transmisión de información

3. Descripción de la Arquitectura

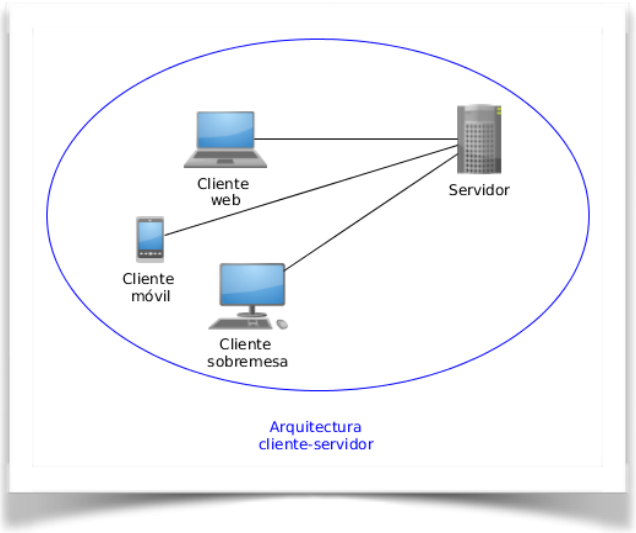
Para implementar nuestro sistema, hemos decidido combinar dos arquitecturas (P2P y Cliente-Servidor), como se muestra en esta vista física de la arquitectura:



Para conectar los distintos servidores del sistema utilizamos una arquitectura P2P, semicentralizada en la que utilizamos un servidor central como super-peer para que los distintos servidores regionales puedan conocer a sus vecinos. Al utilizar esta arquitectura conseguimos aumentar la disponibilidad y la escalabilidad de nuestro sistema, gracias al reparto de carga. En la descripción del sistema se especificó que funcionaría en distintos dispositivos, por lo que decidimos utilizar la arquitectura cliente-servidor para aportar servicios independientes a cada tipo de cliente. Gracias a esta arquitectura conseguimos simplificar el mantenimiento del sistema y aumentamos la robustez, ya que cualquier fallo que ese produzca en un cliente no afectará a los demás.

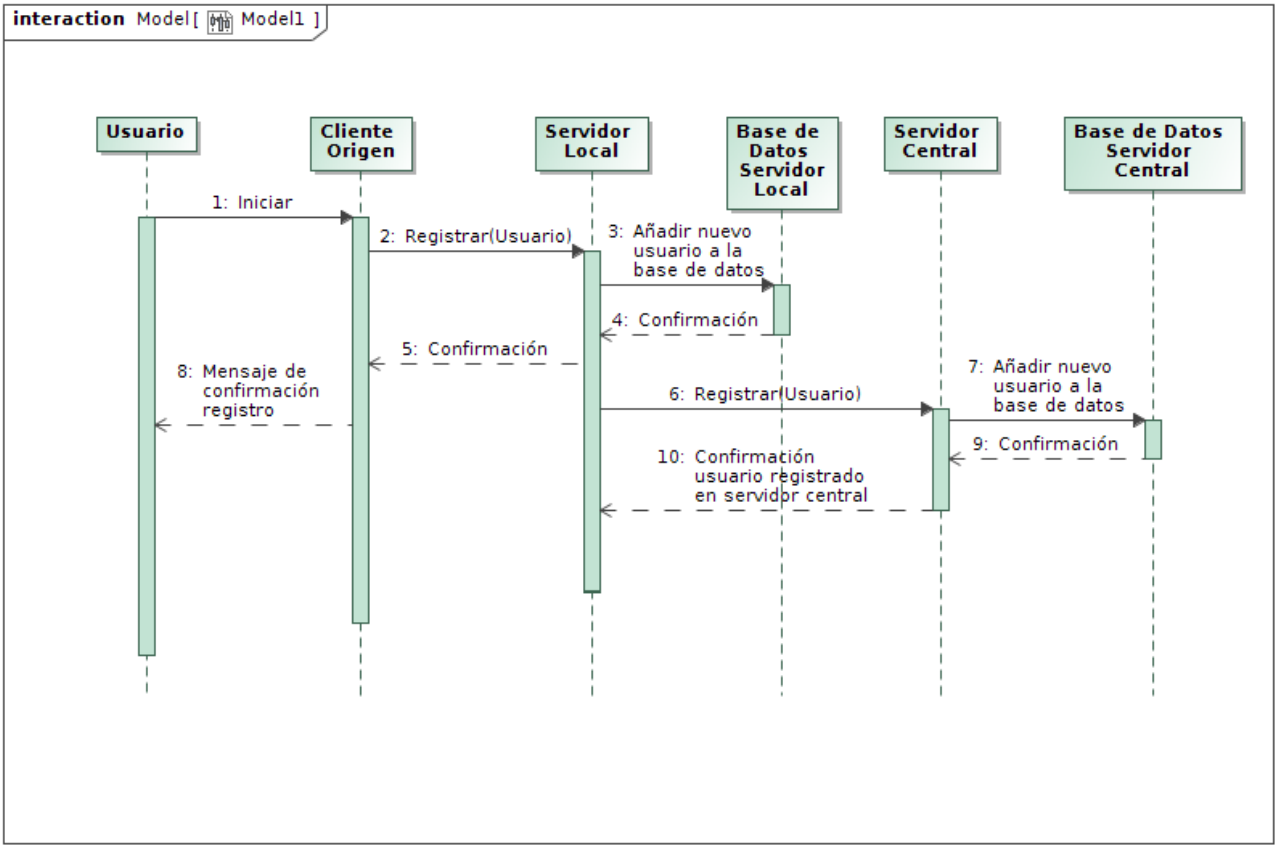
4. Arquitectura Implementada

Hemos considerado que implementar la totalidad de la arquitectura escaparía del ámbito de la asignatura por lo que decidimos implementar la siguiente parte:

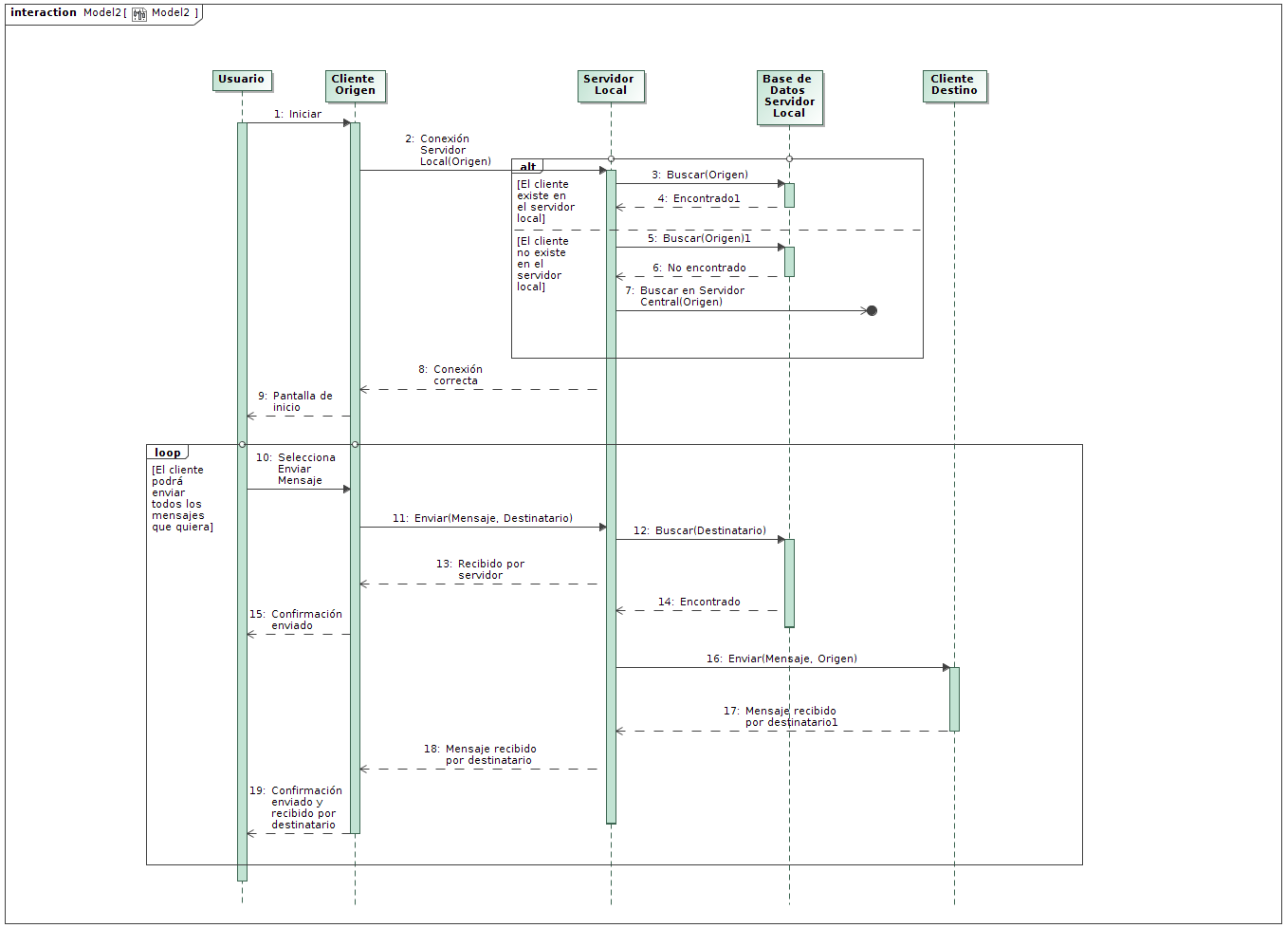


A continuación mostramos las principales funcionalidad del sistema:

- **Registro de usuarios**



- **Envío de mensajes**



5. Registro de Pruebas

Para comprobar el correcto funcionamiento del sistema implementado, hemos llevado a cabo las siguientes pruebas:

- Conexión y desconexión del servidor
- Registro de un nuevo cliente en el servidor
- Registro de un cliente con un login repetido
- Enviar un mensaje a un usuario inexistente
- Enviar un mensaje a un usuario existente
- Caída del servidor

En cualquier caso, el sistema se asegura de proporcionar mensajes informativos (de error, normalmente) al usuario en cada uno de estos casos.

6. Aspectos Relacionados

Durante el desarrollo del sistema, se han intentado seguir los siguiente patrones:

- Acceptor-Connector
- Façade
- Adapter
- Half-Sync/Half-Async

El patrón acceptor-connector ha sido utilizado para gestionar las conexiones de los distintos clientes con sus respectivos servidores regionales. Los patrones façade y adaptar se utilizan para ocultar a los clientes la lógica de operaciones utilizadas, a los clientes se les proporcionan unas funciones básicas (registrar, enviar_mensaje, etc) e ignoran el trabajo realizado por el servidor. También ha intentado seguirse el patrón half-sync/half-async para gestionar la concurrencia de peticiones al servidor.

7. Repositorio

https://github.com/cristiandiazhermida/practica_grupo_as_2017