ELECCIONES

TRABAJO PRACTICO ESPECIAL 1

72.42 PROGRAMACIÓN DE OBJETOS DISTRIBUIDOS
GRUPO 9

Alumnos

LUCAS SANZ GOROSTIAGA	56312
TOMAS MAXIMILIANO DALLAS	56436
AGUSTINA OSIMANI	57526
MICAELA BANFI	57293

Instituto Tecnológico de Buenos Aires
ITBA

17-09-2020

1 Introducción

En el siguiente informe se presentará la implementación para un sistema remoto thread-safe para realizar una elección, contabilizando los votos, definiendo a los ganadores y permitiendo la fiscalización del mismo. Se hará una breve descripción de cómo está organizado el sistema, de cómo funciona y se darán explicaciones sobre decisiones de diseño tomadas.

2 Organización

El proyecto está basado en el arquetipo brindado por la cátedra, y consta de las siguientes 3 partes

- API: presenta las interfaces para la comunicación entre los clientes y los servicios.
- Client: aquí se encuentran los 4 clientes pedidos por la consigna, administración, votación, fiscalización y consulta.
- Server: contiene los servicios remotos de administración, votación, fiscalización y consulta. Contiene a la clase `Election`, que es la encargada del manejo general del sistema

3 Decisiones de diseño e implementación de los servicios

El diseño de los servicios del lado servidor se pensó originalmente como una división de una clase, o `Servant`, por servicio, teniendo aparte una clase principal `Election` que alojaría todos los recursos del servidor. Luego, esto se modificó ligeramente, quitándole la responsabilidad al servicio de fiscalización al mover su funcionalidad dentro de `Election` ya que era más fácil poder "lockear" los recursos desde allí. Todos los `Servant`'s son llamados directamente por su interfaz desde el cliente y cuando necesitan actualizar algún recurso llaman a la instancia de `Election` que tiene cada uno al momento de construirse.
Se decidió utilizar `UnicastRemoteObject` en vez de la alternativa `Activation` puesto que para el alcance de este trabajo no se vió necesario el ahorro de recursos del lado servidor ya que no hay un uso intensivo de los mismos.

4 Criterios aplicados para el trabajo concurrente

Decidimos utilizar un *fair Reentrant Lock* para poder leer y escribir todo lo referido al sistema de votación. Cuando se actualiza el estado de alguna variable, activamos el *write lock*, para así evitar que otro servicio pueda leer esa variable que se está modificando. Desde el lado de la lectura, hacemos lo mismo, pero activando el *read lock* para evitar que otro servicio pueda actualizar la variable que se está intentando leer. Decidimos utilizar el Reentrant Lock en vez de un Lock común porque en un principio habíamos diseñado el sistema de una manera tal que cada función está lo más modularizada posible, por lo que cada función se encarga de lockear o unlockear el lock (read o write) correspondiente para su acción. Entonces, una función podía llamar a varias funciones que accedan a un mismo lock varias veces. Por lo que preferimos utilizar este tipo de lock, que beneficia este caso de uso.

5 Potenciales puntos de mejora y/o expansión

- Actualmente, el sistema permite realizar una sola votación. Una mejora sería poder brindar la posibilidad de realizar mas de una votación sin tener que volver a ejecutar el server. Es decir, *limpiar* el sistema, los votos y los fiscales.
- Otra mejora sería poder paralelizar más el calculo del resultado de la votación. Actualmente se usan streams, pero se podría considerar el uso de parallel streams.
- Se podría utilizar RMI Activatable en vez de UnicastRemoteObject para hacer un uso más eficiente de los recursos del servidor sin necesitar que estén los Servants constantemente escuchando pedidos.