Trabajo Práctico 1

Elecciones

Programación de objetos distribuidos

Segundo Cuatrimestre 2019

Grupo 2

Integrantes

Della Sala Rocío - 56507

Giorgi, María Florencia - 56239

Rodríguez, Ariel Andrés - 56030

Santoflaminio, Alejandro - 57042

Decisiones de diseño e implementación

Lo primero que se realizó fue la separación en 3 módulos; *client, api y server.*
En el primer módulo se encuentran las clases para los 4 clientes pedidos en la consigna: `ManagementClient, VotingClient, FiscalClient y QueryClient.`
En el segundo módulo, se encuentran las interfaces remotas que se le ofrecen al cliente, las cuales implementará el servidor. Todas las interfaces extienden de `Remote`. Además, en la api tenemos algunas clases que usarán tanto cliente como servidor como es la clase `Vote`, enums como `Party, Province, Action` o `ElectionState` y excepciones para el cliente.
En el tercer módulo tenemos tanto la clase `Server` para el servidor como los servants, es decir 4 clases que implementan los métodos de las interfaces remotas. Además tenemos 2 clases y una interfaz más de las ya mencionadas en este módulo. En primer lugar, `VotingSystem` que es la clase que calcula todos los ganadores según el método escogido (FPTP, AV y STV) y además convierte el resultado a string. En segundo lugar, una clase central llamada `ElectionCentral` que es aquella clase que almacena el estado de la votación, los votos y fiscales. Y por último una interfaz `Fiscal Monitor` cuya utilización se explicará luego cuando hablemos sobre cómo se implementó el fiscal en particular.

Lo primero que se ejecuta es el servidor que lo que hace es el binding de los objetos remotos (los servants) al servidor de nombres provisto por Java, llamado RMI Registry. De esta forma podrá ser accedido luego por los clientes. Lo que se realizó fue la exportación del objeto `UnicastRemoteObject` en vez de hacer que la implementación extienda de esta clase.
Los clientes en primer lugar guardan las propiedades del sistema que necesitan para continuar su ejecución. Luego pasan a ejecutar su tarea principal. Por ejemplo el `ManagementClient` busca a través del método lookup el servicio en el RMI Registry, y al encontrarlo llama a los métodos sobre los cuales tiene acceso, es decir aquellos declarados en la api. Es interesante destacar de forma particular el funcionamiento e implementación de `FiscalClient`.

Para la implementación del fiscal, lo que se realizó fue pasarle al servant una función de callback para que le comunique al cliente cada vez que un voto era registrado. Para ello se definió una interfaz remota `Detector` en la api. En el cliente se implementa esta interfaz en un objeto remoto de callback que lo llamamos `ElectionFiscal`. El fiscal instancia este objeto y lo exporta como un `UnicastRemoteObject` y de este modo se quedara escuchando pedidos remotos. El cliente llama al servicio pasando este objeto como parámetro. Al terminar su proceso, el servidor notifica al cliente del evento.

En cuanto al uso de colecciones para la implementación tenemos dos listas (instanciadas como `ArrayList`) en `ElectionCentral` una de fiscales registrados y otra de votos totales Además, tenemos dos colecciones más. Para el servant del cliente de consulta, tenemos un mapa de partidos y resultados el cual es usado como una instancia media hasta la conversión a un string. Para el servant del cliente de fiscalización tenemos un mapa de mapas donde se guardan los fiscales registrados por mesa por partido, pudiendo tener únicamente un fiscal del mismo partido por mesa. Este mapa se instancio como ConcurrentHashMap para la concurrencia.

Ahora pasaremos a hablar de la implementación de los distintos sistemas de votación implementados. La primera decisión importante fue que en STV se trabajó con porcentajes mientras que en AV con la cantidad de votos. Esto porque debido a la forma de calcular la transferencia de votos en STV, podrían quedarnos fracciones de votos, por lo cual se trabajó con porcentajes.
En AV, la meta porcentual está establecida en 50%. Si ninguno de los candidatos llegara a dicha meta, se elimina según el procedimiento pero puede haber casos en los que nunca se llegue. AV siempre imprime como resultado el ganador, o en caso de empate técnico aquellos con mismo porcentaje de votos.

Tanto en AV como en STV, si hay igual cantidad de votos para un partido (o porcentaje en STV), se toma uno de forma aleatoria para la evaluación de aquellos que son los menos votados. En STV se toma de forma aleatoria alguno que haya sobrepasado la meta del 20%.
En STV se reparten los votos hasta que eventualmente deja de ser posible y entonces es posible que queden representantes con porcentaje menor a 20%. También puede que en los resultados algún partido tenga más del 20% de los votos, esto porque al tomarlo como representante, el partido tenía votos intransferibles.
Para calcular el porcentaje a transferir cuando se sobrepasa la meta se toman en cuenta todos los votos de ese partido como primera opción y por igual a aquellos votos que tenían a dicho partido como segunda o tercera opción y habían sido transferidos previamente. Luego se reparte el porcentaje de forma equitativa a aquellos partidos como siguiente opción a aquellos votos (en el caso de tercera opcion no se transfieren, pues no hay una cuarta).
En STV, aquellos votos que tenían como segunda opción algún partido que quedó fuera de carrera (ya sea por eliminación por ser el menos votado o por haber sido elegido representante) son descartados al tener que ser transferidos de su primera opción a la segunda ya desaparecida. Lo mismo sucede en AV si al eliminar el menos votado, la segunda opción de aquellos votos fue eliminada previamente. Esta decisión se tomó para mantener cierta consistencia entre ambos sistemas.

Criterios aplicados para concurrencia

Para la concurrencia se decidió hacer la aplicación cliente-servidor lo más thread safe posible aplicando diferentes técnicas vistas en la teoría:

- Por un lado, se dispuso que algunos bloques de código utilizen la keyword `synchronized` para evitar que dos clientes estén utilizando el mismo bloque de código al mismo tiempo.
- Además de lo anterior, se utilizó un `ReentrantReadWriteLock` para ciertas zonas clave del código que eran constantemente modificadas por los clientes. Principalmente, se lo utilizó para bloquear la escritura en la lista de votos, para así, evitar que dos clientes escriban sobre la lista al mismo tiempo, pudiéndose perder votos.
- Como medida adicional, se decidió usar un `ConcurrentHashMap` para el registro de los fiscales ya que permite escritura utilizando un `lock` internamente lo cual evita la pérdida de registros de fiscales.

Potenciales puntos de mejora y/o expansión

- En cuanto al diseño, se pudo haber rediseñado el código de STV y AV para que estuviera más modularizado y de esa manera poderlo hacer más eficiente y a la vez más entendible para que en un futuro pudiera mejorarse o escalarse a otros tipos de votaciones.
- En cuanto a la concurrencia, hubiera sido deseable tener tests que asegurasen al 100% que la aplicación cliente-servidor fuera thread safe para garantizar que no se pierdan votos ni registros de fiscales, siendo esto el punto central del trabajo práctico