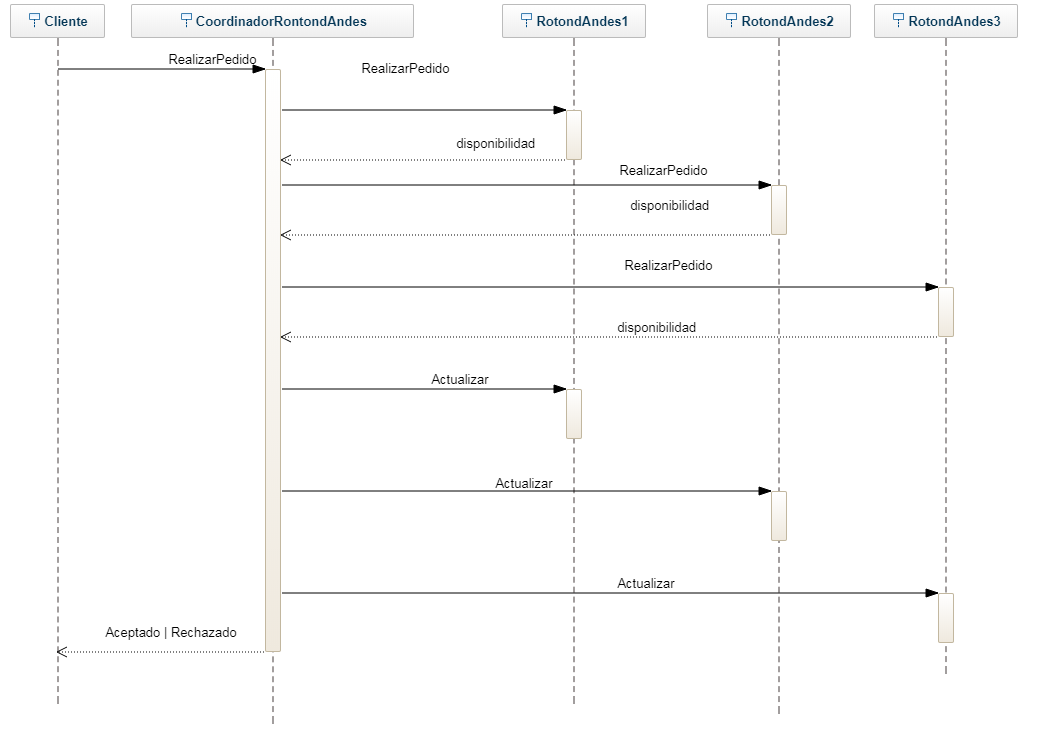
**Documentación iteración 5**

**PARTE 1)**

**Restricciones Planteadas**

Con base a los artículos adjuntos se establecieron las siguientes restricciones:

1. Todas las rotondas deben usar la funcionalidad de XA en sus servidores ya que esta evita que, al momento de llevar a cabo el protocolo de Colas de mensajes, el mensaje no se pierda o se duplique gracias a que se llevará a cabo también el reconocimiento transaccional.
2. Cada implementación de un XA resources deberá empezar, preparar, finalizar, hacer commit o rollback a las operaciones de las transacciones de cada rama corriendo en la sesión de la instancia del XA resources asociado.
3. Ya que cada instancia de XA resources corresponde a una sesión de Oracle (una rotonda particular), solo debe haber una transacción activa al mismo tiempo en la instancia de XA resourse asociado a esta. Además, puede haber transacciones adicionales pero suspendidas esperando.

**Diagrama de secuencia**

El diagrama se divide en 4 etapas principales:

1. El cliente realiza el pedido, en el cual se incluyen los productos seleccionados por este. El pedido es recibido por el coordinador de la Super Rotonda.
2. Una vez el Coordinador recibe la solicitud, este consulta, para cada una de las rotondas, la disponibilidad de cada uno de los productos dentro de estas. Acto seguido, cada una de ellas otorga una repuesta al coordinador de la Super Rotonda.
3. Tan pronto el Coordinador tenga todas las repuestas necesarias, entonces se encarga se actualizar para cada Rotonda, la cantidad de productos disponibles según la respuesta otorgada.
4. Por último, el coordinador se encarga de responderle al cliente con un mensaje de “Aceptado” o “Rechazado” correspondiente al pedido realizado, según se encuentren los productos solicitados disponibles o no.

**Análisis de estrategias**

Para el funcionamiento apropiado de los requerimientos de la SuperRotondAndes se determinan ventajas y desventajas sobre el uso de uno de los dos protocolos para cada requerimiento.

* **RF18:**

1. **‘Two phase commit’**: Teniendo en cuenta que este protocolo es estricto en cuanto a las reglas de negocio que se plantean, este ofrecería resultados que cumplirían fácilmente con las exigencias del requerimiento (confirmación de disponibilidad de productos).

Para la implementación apropiada (distribuida) se requiere una comprobación, obtener los elementos de todas las rotondas y verificar la disponibilidad, y la creación del pedido en las rotondas correspondientes (las que poseen los productos solicitados), para esto se pedirán los resultados de disponibilidad junto a un mensaje de identificación de la rotonda. Se ajusta entonces a lo que pide el requerimiento en cuanto a “rechazado” o “aceptado” como respuesta al cliente, respuestas otorgadas por el coordinador quien toma la decisión una vez tiene una respuesta de todas las rotondas. Debido a esto, se perdería eficiencia en cuanto a tiempo ya que para poder realizar tanto una transacción global, como para realizar la consulta, ya que se tendrá que esperar que cada una de las rotondas realicen el proceso ya descrito, entonces si una de ellas se demora y las otras no, la transacción global si lo hará y todas las demás tendrán que esperar.

1. **Colas de mensajes**: Para este protocolo Se deberían agregar los ‘Topics’ necesarios para realizar la transacción, en este caso un topic para consultar la disponibilidad, en caso de que esta sea adecuada, generar un mensaje con topic ‘generarPedido’ para que las rotondas generen los pedidos correspondientes. Este método dificulta el cumplimiento de reglas de negocio (no lo imposibilita). También habría que manejar una parte del mensaje como la identificación de la rotonda para generar el pedido a la rotonda correspondiente

* **RF19:**

1. **‘Two Phase Commit’**: En este requerimiento sería ideal llevar a cabo este protocolo, para asegurar que en todas las rotondas se ha borrado el restaurante indicado. Ya que gracias a Two Phase Commit, por su naturaleza, se sabe si se logró realizar la transacción en cada uno de los destinos, y para este requerimiento es necesario que el restaurante sea borrado no solo en una de las rotondas sino en todas. Para proceder con este protocolo habría que enviar la solicitud a las rotondas involucradas con el restaurante (donde este se encuentre), posteriormente se esperaría respuesta de quienes lo pueden hacer para que finalmente el coordinador logre decidir si es debido o no eliminarlo en todas las rotondas. Como fue mencionado anteriormente, la ventaja es que se cumple el hecho de que se eliminaría el restaurante en todas las rotondas involucradas, o en ninguna en caso de que alguna no lo pueda llevar a cabo, es decir, no dejaría que en una rotonda se haya eliminado y en otra no. Y por el lado de las desventajas es también lo ya mencionado, el tiempo que este pueda tardar gracias a que una de las rotondas o sub transacciones se demore y las otras, como también el coordinador, tendrían que esperar, y esto retrasa la transacción global.
2. **Colas de mensajes**: Este protocolo supondría que en el topic se envíe el mensaje de borrado asociado, y que para cada rotonda se ejecute la transacción. Lo cual no asegura por completo que en cada una de las rotondas se logre llevar a cabo el proceso, ya sea por tiempo o por ejecución misma de esta. Como ventajas se podría decir que tardaría menos tiempo al implantar los topics haciendo que en cada rotonda que escuche el mensaje se lleve a cabo la eliminación solicitada sin depender de la otra, el tiempo que demore o los recursos que consuma.

**-RC13:**

1) **‘Two phase commit’**: Teniendo en cuenta que el protocolo ‘two phase commit’ es estricto en cuanto a las reglas de negocio que se plantean, esta cumpliría fácilmente con las exigencias del requerimiento, aunque no sería necesario puesto que no se necesita ninguna comprobación de nada, sino solo obtener la información. Por lo que sería ineficiente este protocolo.

2) **Colas de mensajes:** Para este protocolo Se deberían agregar los ‘Topics’ necesarios para realizar la transacción, en este caso un topic para consultar la disponibilidad, en este caso sería el ideal dado que se podría pasar lo que se necesita de cada aplicación y sería mucho más eficiente en tiempo y daría los mismos resultados.

**-RFC14:**

1) **‘Two Phase Commit’**: En este requerimiento no sería necesario un two phase commit, puesto que solo se requiere la información de cada DB y luego mezclarla. Por lo tanto, esperar que cada una de ellas responda para poder dar una respuesta definitiva no sería óptimo ni lo indicado, se perdería información en caso de que alguna no logre llevar a cabo bien la consulta.

2) **Colas de mensajes:** Este protocolo supondría que en el topic envíe el mensaje que contenga la información de cada base de datos y se retornaría. Lo cual a comparación del anterior es mucho mejor, cada uno responde sin depender del otro, se une todo lo obtenido, independientemente si alguno respondió o no, y se genera el mensaje de respuesta con base en ello.

**PARTE 2)**

- RQF18: Cuando se trata de una transacción distribuida, el proceso cambia debido a que ahora la disponibilidad no solo se tiene que comprobar en una rotonda, luego el pedido tiene que ser generado por cada rotonda y puede que haya inconsistencias en cuanto a los productos generados en una rotonda y en otra, Adicionalmente, antes de rechazar el pedido por falta de disponibilidad se deben acordar las 3 rotondas para hacer rollback de los productos asociados al pedido que si fueron aceptados. En cuanto la estrategia global, se definió como regla de negocio que todos los productos asociados al pedido deben estar disponibles para que el pedido sea aceptado, si al menos un producto no está disponible (en ningún restaurante de ninguna rotonda hay disponibilidad) no se acepta el pedido, para manejar esto lo mejor es usar el protocolo Two Phase Commit, para verificar el cumplimiento de todas las subtransacciones (pedidos a cada rotonda).

- RQF19: En este caso de transacción distribuida, al igual que en la anterior, la eliminación de un restaurante debe poder validarse en cada rotonda antes de poder ser validada en la superrotonda, de otro modo se podrían producir inconsistencias en futuros pedidos, además de verificar los pedidos con productos de ese restaurante existentes para eliminar inconsistencias. De la misma manera se la mejor estrategia es un Two Phase Commit, para validad cada subtransacción de eliminar el restaurante en cada rotonda.

- RFC13: Para una transacción distribuida en una consulta así el problema no es más grave que cada rotonda sea capaz de hacer esa consulta en su base de datos, ya que después no se hacen cambios y no se tienen que validar transacciones adicionales en esta consulta

- RFC14: Para una transacción distribuida en una consulta así el problema no es más grave que cada rotonda sea capaz de hacer esa consulta en su base de datos, ya que después no se hacen cambios y no se tienen que validar transacciones adicionales en esta consulta