Iteración 5

Juan C. Corrales, Daniel Perilla
Iteración 5 proyecto RotondAndes
Universidad de los Andes, Bogotá, Colombia
{jc.corrales, d.perilla11} @uniandes.edu.co

Fecha de presentación: Diciembre 12 de 2017

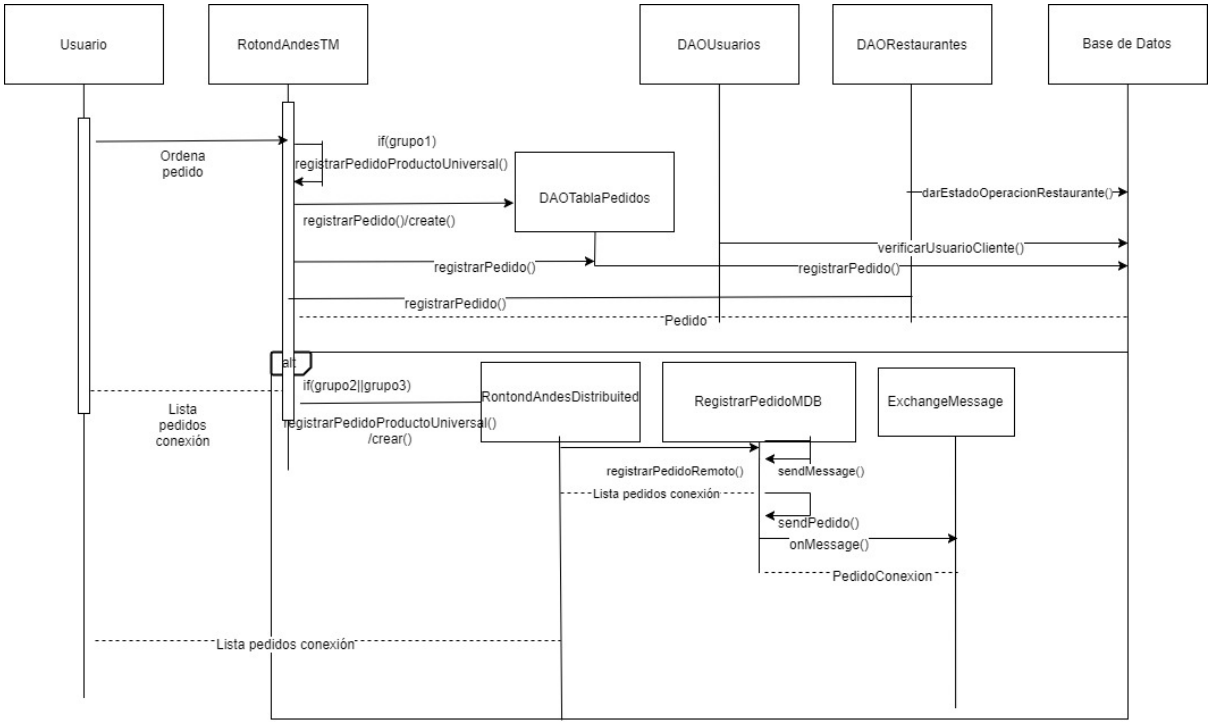
Tabla de contenido

1. Análisis de la implementación de transacciones distribuidas

1.1 (5%) El XO es un estándar para el procesamiento de transacciones distribuidas (DTP). En él se describe la interfaz entre el gestor de transacciones (componente que procesa información descomponiéndola de forma unitaria en operaciones indivisibles, llamadas transacciones) global y el administrador de los recursos locales. Esto toma un papel importante cuando se tiene en cuenta que el sistema responde a lo que es una petición del usuario, con esta petición, el sistema debería estar en capacidad de traducir la información a cada una de sus entidades miembros para que así puedan realizar sus funciones respectivas. Es importante aclarar que este estándar de comunicación de entrada tiene que ser similar a uno de salida(respuesta) con tal de que el eje central entienda las respuestas.

1.2 (10%) Primero toco cambiar buena parte de la arquitectura de la base de datos con el fin de implementar el sistema compartido, se creó la tabla de cliente externo que tenía la capacidad de realizar pedidos de sin tener que estas suscrito a ninguna rotonda en específico y que pudiera ser atendidos con cualquiera de las rotondas, en cambio también se puede identificar. En segunda instancia para un pedido creamos una orden, le asignamos la orden en cuestión, cuando se crea un producto en otras palabras se crea un nuevo método para manejar las ordenes desde las otras rotondas. RentabilidadRestaurante es una clase del VOS temporal que se encargaba de cumplir con el RFC14.

1.3(8%)



1.4 (15%) Para la cola de mensajes la estrategia que se utilizó fue la de utilizar una clase llamada RegistrarPedidoMDB está implementaba lo que es un MessageListener. Esta clase cumplía con lo que conocemos como una Queue FIFO de mensajes, está era necesaria para cumplir con los requerimientos de cola de mensajes pues sin esta no existiría lo que es la cola de mensajes. Esta clase cumplía con la función de almacenar y enviar los mensajes según era notificada, de esta manera nos permitía escuchar mientras hubiera rotondas no disponibles y trabajar en los pedidos cuando era posible. Con esta estrategia nos fue capaz no tener que estar siempre pendiente de los pedidos del cliente, sino que podíamos almacenar las órdenes, así se prevenía la caída del sistema por una gran cantidad de peticiones a la vez.

El Two phase commit se utilizó en cambio en lo que fue los onMessage(), dentro de estos habían llamados a métodos que tenían dentro de sí operaciones de rollback en caso de que algunas de las transacciones pendientes de esta u otras rotondas no llegara a pasar. Para que esto funcionara de manera específica se tuvieron que tener rutas de precaución para estos casos, por ellos en muchos de los métodos para los nuevos requerimientos salieron muchos ifs de escape de métodos, de manera que no se presentaran cuellos de botella ni excepciones a no presentarse las condiciones esperadas. Gracias a esta estrategia podíamos preservar cierto tipo de consistencia en el sistema por el hecho de que no se despachaban pedidos si no existían rotondas capaces de cumplir con las demandas de los clientes.

2. Especificación e implementación de transacciones distribuidas

2.1 (10%) RF18 – Registrar pedido de productos a una mesa

- Impacto estrategias: Debido a que los pedidos pueden contener productos de distintos RotondAndes, es necesario consultar restaurante por restaurante cada uno de los productos ordenados, de manera que se crea como una lista de cosas con las que se tienen que cumplir, y a medida que se piden los productos por Rontoda se van quitando elementos de esta lista.

- Estrategia global más indicada: Se hace un recorrido por las diferentes rotondas, para ir quitando del pedido los productos que se pueden ir ordenando por rotonda. De esta manera se hace capas pedir todos los productos. Es importante en este punto explorar la importancia del Two phase commit, si luego de revisar todas las rotondas se da a la realización que no esta disponible un producto en específico, se espera que se haga un rollback a toda la operación en todos los restaurantes de manera de hacer todo congruente con la vida real.

RF19 – Retirar restaurante

- Impacto estrategias: Para que las informaciones de todas las tablas de datos sean congruentes, toca consultar todas las bases de datos para determinar si existen o no el restaurante buscado para cada RotondAndes, de esta manera se elimina secuencialmente el restaurante, siempre y cuando este exista en la base de datos de Rotondandes
- Estrategia global más indicada: Para que el resultado sea congruente es necesario realizar la acción de búsqueda y eliminación respectiva para cada una de las bases de datos presente en el sistema. Con el fin de que se pueda realizar el método de manera que sea congruente en todas las bases de datos, en caso de que no se pueda eliminar el restaurante en todas las bases de datos, se espera que se haga un rollback a toda la operación, esto se hace mediante la implementación de un Two phase commit en este requisito.

RFC13 – Consultar los productos disponibles en Rotondandes

- Impacto estrategias: Como cada entidad perteneciente a cada uno de los RotondAndes puede ofrecer productos distintos, es necesario reconocer en este caso, primeramente cuál de los restaurantes de los RotondAndes, contiene el producto buscado por el cliente. Por ello es necesario buscar entre todas las bases de datos para determinar cuál tiene el producto buscado.
- Estrategia global más indicada: Es necesario hacer un recorrido por todas las bases de datos disponibles para determinar cuál de estas tiene el producto indicado. Esto se debe a que pensar en crear otra base de datos con una tabla que relacione RotondAndes con productos es no posible.

RFC14 – Consultar la rentabilidad de un restaurante

- Impacto estrategias: Con el fin de cumplir con este requerimiento es necesario actualizar la base de datos de la entidad para poder calcular la rentabilidad de un restaurante, con el fin de que esto funcione sin inconvenientes ni irregularidades, la base de datos de cada uno de los RotondAndes necesita actualizarse con el resto de las bases de datos, y así actualizar de manera correcta la información con respecto a la realidad.
- Estrategia global más indicada: Para implementar la información de tal manera que las bases de datos estén actualizándose de manera constante, se utiliza conexiones entre las bases de datos a modo de colas de mensajes, en este caso RabbitMQ.

RNF9 – Distribución 2

- Impacto estrategias: Con el fin de que la aplicación se pueda utilizar desde cualquier sitio de manera que parezca una solo entidad en vez de tres, utilizamos lo que es RabbitMQ, un software de mensajería que actúa como middleware (un software que presta servicios más allá de lo que son los sistemas operacionales que permite que las diferentes entidades de un sistema distribuido se comuniquen y manejen información)

entre emisores y destinatarios, gracias esto las entidades se unen las unas otras mediante MQ y una central para antender al cliente.

- Estrategia global más indicada: Para realizar este requerimiento es primordial hacerla instalación correcta de RabbitMQ, software que se encargará de crear las colas de mensajes con los usuarios.

RNF10 - Disponibilidad

- Impacto estrategias: Para implementar la disponibilidad constante en la aplicación decidimos diseñar una cola de mensajes, esta nos da notificaciones de las acciones del cliente cuando el sistema no se encuentra disponible. Luego cuando ya este activo, el sistema revisa las acciones presentes en la cola de estilo FIFO de manera que no solo es congruente con la temporalidad con la que los clientes realizaron las acciones en la aplicación, sino que además es capaz de atender a sus acciones así no se encuentre disponible en el momento.
- Estrategia global más indicada: Debido a que este es un nuevo sistema lo que se hace es implementarlo en la aplicación general que funciona para los tres grupos, esto con el fin de generar una cola de mensajes en caso de que ninguna o alguna de los grupos en especial no tenga la aplicación disponible. Creando así no solo una cola de mensajes para el grupo como un todo sino uno para cada subgrupo.