# “Iteración 5 – Transacciones distribuidas”

Vivian Natalia Gómez Cubillos, Kelly Katherine Peñaranda Rivera

Nixon Fernando Ortiz S., Carlos Alberto Mendoza Patalagua

Anyella Valeria Pérez Buendía, Diego Alejandro Solano

Universidad de los Andes, Bogotá, Colombia

{vn.gomez, kk.penaranda}@uniandes.edu.co

{nf.ortiz, ca.mendoza }@uniandes.edu.co

{av.perezb, da.solano1}@uniandes.edu.co

Fecha de presentación: noviembre 21 de 2017

## Contenido

# 1. Análisis e implementación de transacciones distribuidas

## 1.1 Restricciones sobre los sitios que montan la aplicación RotondAndes.

Usar la interfaz XA de X/Open, permite que se pueda usar el gestor de transacciones global y administrador de recursos, que preserva las propiedades ACID en sus transacciones e implementa el protocolo *Two Phase Commit*(2PC). En este caso la primera restricción en nuestra aplicación es la necesidad de que los recursos se encuentren comprometidos y siempre disponibles, ya que en el caso que algún participante no esté disponible o por algún motivo no responda en alguna de las fases del 2PC, la transacción será revertida. Adicionalmente, otra restricción es la necesidad de que las 3 Rotondas utilicen este protocolo de comunicación e implementen la interfaz de XA, puesto que de lo contrario el protocolo no funcionaría correctamente. Por último, otra restricción es que las transacciones tienen que ser respondidas en un tiempo estimado y considerado para las consultas, ya que el protocolo requiere comunicarse de manera sincrónica y necesita un *timeout* para preservar sus fases y correcto funcionamiento.

Las transacciones distribuidas con colas de mensajes pueden tener restricciones de un tiempo de respuesta adecuado para la solicitud de los requerimientos de la rotonda, ya que, aunque este tipo de protocolo trabaja de manera asincrónica, si es necesario un tiempo de respuesta adecuado para la solicitud de algún tipo de producto del restaurante o la información de cierto restaurante. Adicionalmente, otra restricción podría ser la de dar cierta prioridad en las solicitudes a un determinado restaurante, ya sea porque la solicitud fue realizada en dicho restaurante y este posee en sus menús los productos solicitados o porque hay algún tipo de favoritismo del cliente hacia determinada rotonda.

Especificando las restricciones para cada requerimiento:

- Mismo valor para las constantes en las 3 aplicaciones:

✓ GLOBAL_TOPIC_NAME
✓ LOCAL_TOPIC_NAME
✓ TIME_OUT
✓ REQUEST
✓ REQUEST_ANSWER
✓ Modo AUTO_ACKNOWLEDGE

- Para RF18: REGISTRAR PEDIDO DE PRODUCTOS A UNA MESA

Si el TIME_OUT es violado, se manda un mensaje a la cola de respuesta informando del problema y se hace rollback de la transacción local.

- Para RF19: RETIRAR RESTAURANTE

✓ La cadena debe existir en las 3 aplicaciones.
✓ Cada aplicación recibe la cadena de restaurante que se desea retirar.
  - Para RFC13

Para este requerimiento se va a utilizar Two-phase Commit dado que es necesario mostrar la información de las 3 Rotondas o en caso contrario mandar un mensaje de error.

**Nota**: Por problemas con la cantidad de datos soportados por Postman, se fijó un límite de tuplas aceptadas.

- Para RFC14

Para este requerimiento se utilizará Two-phase Commit dado que es necesario que la información debe ser completa en cuanto a facturación de todas las Rotondas. En caso de que se presente un error se hace RollBack de todas las bases de datos de las Rotondas.

## 1.2 Ajustes de arquitectura y modelo UML

### 1.2.1 Modelo UML – Sitio 1

Para permitir la ejecución de transacciones distribuidas y el uso de estrategias como $Colas\ de\ Mensajes$ y $Two\ Phase\ Commit$ se modificó la arquitectura de cada uno de los proyectos que implementan las 3 rotondas integradas en la nueva aplicación. En este orden de ideas, implementamos los paquetes $jms, dtm\ y\ startup$, los cuales contienen los elementos necesarios para permitir las transacciones distribuidas. En primer lugar, $jms$ contiene los $MDB's$ ($Message\ Driven\ Beans$) que son oyentes de un mensaje para el procesamiento asincrónico de los mismos, estos $MDB$ contienen las constantes de los $Topics$ globales que permiten la comunicación entre las 3 rotondas y los métodos $OnMessage, SendMessage\ y\ OnException$ para recibir y enviar mensajes y manejar excepciones, respectivamente. En segundo lugar, el $dtm$ contiene la clase "manejador" de las transacciones remotas, el $dtm$ se comunica con el $tm$ para unir la información de la respuesta remota con la respuesta local. Finalmente, el paquete $startup$ contiene la clase responsable de permitir que los servicios se resuelvan de manera remota, puesto que esta establece la conexión con $rabbitMQ$.

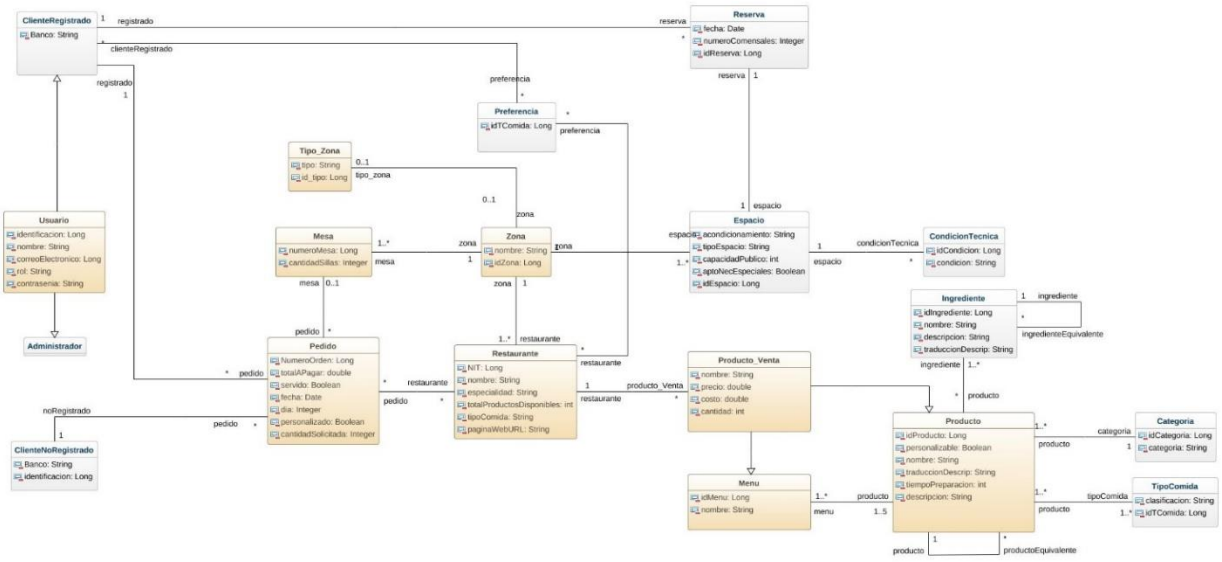


Figura 1. Modelo conceptual sitio 1

### 1.2.2 Modelo UML – Sitio 2

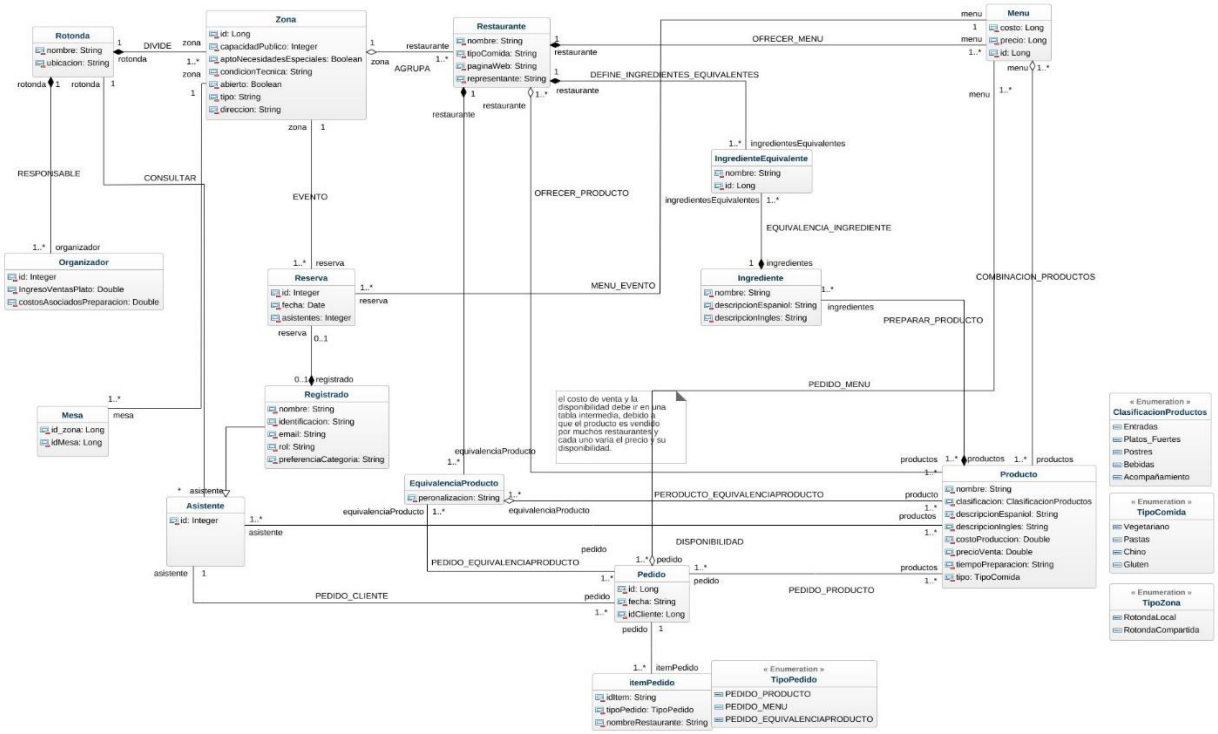


Figura 2. Modelo conceptual sitio 2

### 1.2.1 Modelo UML – Sitio 3

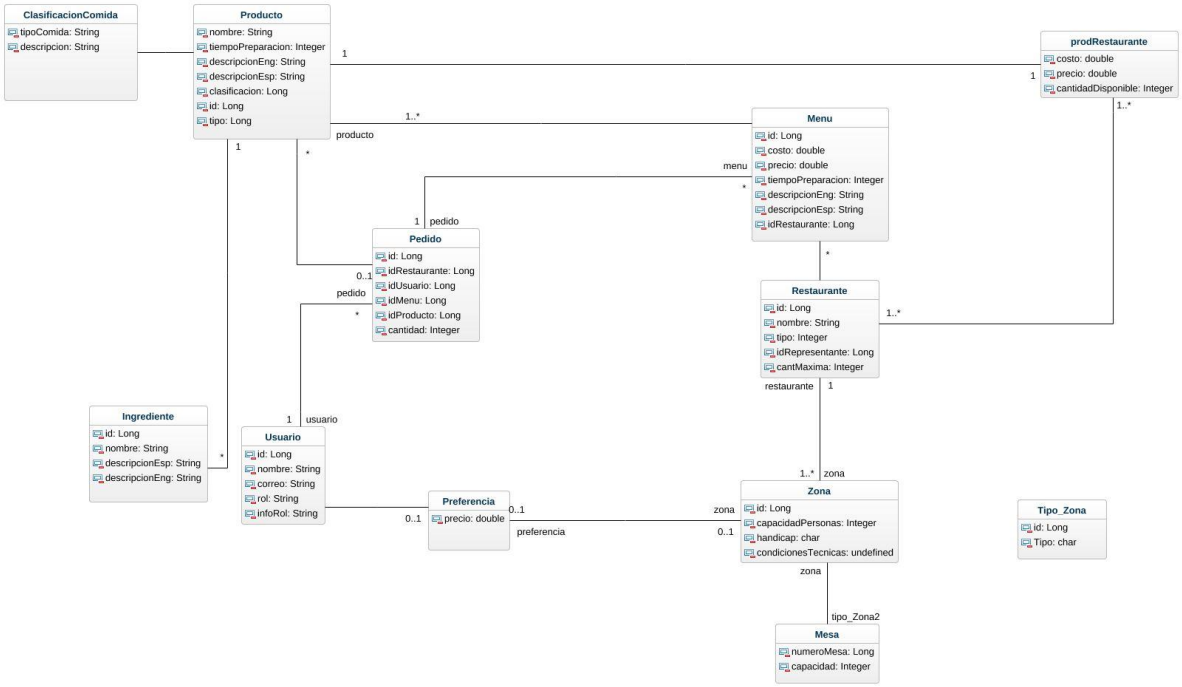


Figura 3. Modelo conceptual sitio 3

## 1.3 Lógica del proceso del RF18.

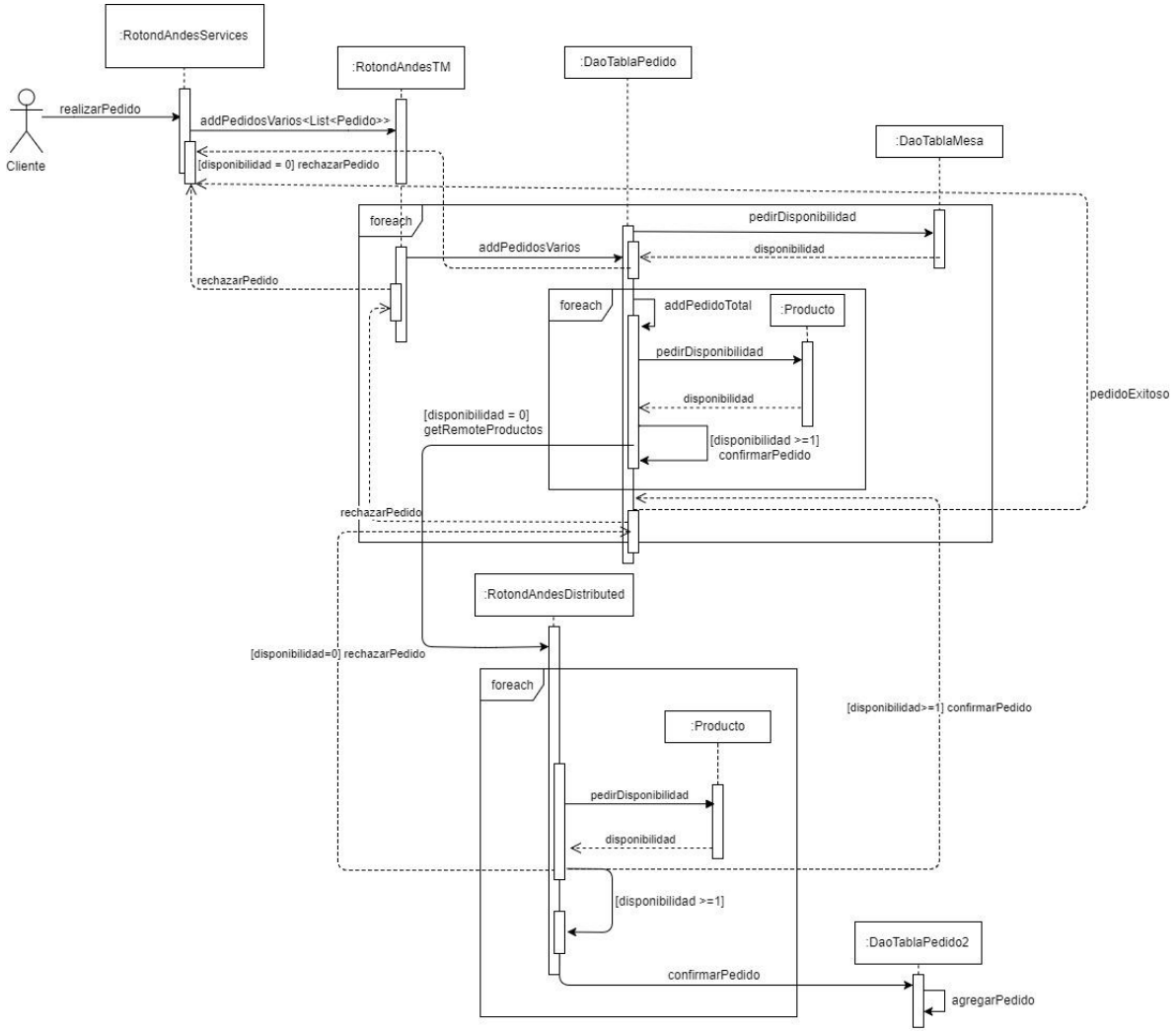


Figura 4. Modelo conceptual sitio 3

La ejecución del requerimiento REGISTRAR PEDIDO DE PRODUCTOS A UNA MESA sigue la siguiente lógica y flujo de eventos:

**1.** Un cliente realiza un pedido, especificando los productos deseados con su respectiva cantidad, la zona y la mesa en la que los desea.

**2.** Por cada pedido, se realizan dos verificaciones principales:

- La disponibilidad de la mesa con base en el número de sillas de la misma.
- La disponibilidad de los productos con base en la cantidad solicitada.

Para la primera verificación, si la mesa deseada no está disponible, entonces se envía un mensaje al cliente indicando dicha situación.
Para la segunda verificación, si los productos no están disponibles en la rotonda en la cual está la mesa deseada, se procede a solicitar los productos de otra rotonda. Sin embargo, si en ninguna rotonda hay disponibilidad, entonces se envía un mensaje al cliente especificando dicha situación.

**3.** Dado el caso en el que hay disponibilidad de la mesa y la cantidad de productos deseada, se procede a asignar el pedido a los restaurantes que ofrecen los productos deseados en la rotonda o rotondas, según sea el caso.

## 1.4 Comparación estrategias: Two-Phase-Commit vs Colas de mensajes

| Requerimiento Funcional | Colas de Mensajes | Two Phase Commit |
|---|---|---|
| **RF18**: Registrar Pedidos de productos en una mesa | -Permite a los usuarios realizar varios pedidos a diferentes restaurantes de manera asincrónica, es decir, no deben esperar a que uno de estos haya sido entregado para poder solicitar otro.<br><br>-Permite a los restaurantes pueden recibir múltiples pedidos así se encuentren ocupados manejando alguna solicitud. | Gracias a la implementación de este protocolo es posible rechazar totalmente el pedido cuando uno de los productos que en él se solicitan no se encuentra disponible. Esto debido a que las rotondas participantes deben confirmar si disponen de los productos, si para algún producto ninguna de ellas lo hace se realiza un $abort$ a la transacción global. |
| **RF19**: Retirar restaurante | La implementación de este requerimiento a través de este protocolo le permite a la rotonda recibir la solitud de retiro de un restaurante, es decir, este solicita ser eliminado y queda en espera de su respuesta. Sin embargo, pueden haber inconsistencias, puesto que al no ser retirado de inmediato, este puede seguir recibiendo pedidos. | Se evita que un restaurante que quiere retirarse de una rotonda siga recibiendo pedidos y que los productos que este ofrecía les aparezcan con disponibilidad a los clientes, puesto que se garantiza un retiro inmediato. |
| **RFC13**: Consultar productos disponibles | No sería muy útil, puesto que se solicita una confirmación inmediata, no es factible dejar en espera al usuario para este tipo de requerimiento. | Facilita brindar una respuesta inmediata a cliente. El estilo del protocolo permite dar una respuesta consistente, pues se solicita la información a cada Rotonda participante y se une en una respuesta unificada por el coordinador del protocolo. |

| | | |
|---|---|---|
| **RFC14**: Consultar rentabilidad de un restaurante. | No sería muy útil, puesto que se solicita una confirmación inmediata, no es factible dejar en espera al usuario para este tipo de requerimiento. | Facilita brindar una respuesta inmediata a cliente. El estilo del protocolo permite dar una respuesta consistente, pues se solicita la información a cada Rotonda participante y se une en una respuesta unificada por el coordinador del protocolo. |

# 2. Especificación e implementación de transacciones distribuidas

## 2.1 Reglas de negocio asociadas al retiro de un restaurante

### 2.1.1 Reglas de negocio asociadas al retiro de un restaurante – Sitio 1

Nuestra Rotonda tiene las siguientes reglas de negocio asociadas al retiro de un restaurante:

- El restaurante no debe tener productos disponibles. Esta regla es solicitada debido a que si el restaurante aún tiene productos disponibles y no se retira de inmediato (caso de implementación por medio de Cola de Mensajes) puede generar inconsistencias, puesto que esto permitiría que un usuario solicite sus productos.
- El restaurante no debe tener pedidos pendientes, todos sus pedidos en la base de datos deben estar servidos o cerrados. Esta regla es importante puesto que el restaurante debe cumplir con todos sus pedidos antes de retirarse de la rotonda.
- Aquel que desea realizar el retiro es un usuario “Restaurante” y tiene una identificación válida a partir de la cual desea solicitar el retiro.

### 2.1.2 Reglas de negocio asociadas al retiro de un restaurante – Sitio 2

Para que un restaurante que solicita su retiro de la rotonda pueda ser eliminado exitosamente es necesario que éste cumpla con las siguientes condiciones:

- No puede tener pedidos activos, es decir, el restaurante no debe tener facturas que no han sido pagadas ni pedidos que falten por entregar. Estas 2 condiciones con el objetivo de que el restaurante no pierda dinero de ventas que concretó previamente y que los clientes puedan recibir los pedidos que solicitaron cuando el restaurante aún hacía parte de la rotonda.
- No ofrecer ningún producto. Es indispensable que el restaurante quite de la lista de productos ofrecidos por la rotonda aquellos de los que era responsable incluyendo sus menús y sus equivalencias de productos para así evitar inconsistencias y que los clientes soliciten productos que no les podrán ser servidos.

### 2.1.3 Reglas de negocio asociadas al retiro de un restaurante – Sitio 3

Para la Rotonda número 3 se manejaron las siguientes reglas de negocio en el proceso de retirar un restaurante asociado:

- El establecimiento no puede tener pedidos pendientes, es decir, debe haber entregado todos los pedidos pendientes a los respectivos clientes.
- El usuario que solicita el retiro de su establecimiento debe ser de tipo restaurante.
- Al retirarse todos los recursos dependientes en cascada como menús, productos, etc. Deberán ser eliminados.

## 2.2 Impacto de las estrategias transaccionales distribuidas en RotondAndes.

La implementación de este requerimiento mediante el protocolo de cola de mensajes facilita la concurrencia de la aplicación en cuanto a la solicitud de pedidos por parte de los clientes, ya que permite que la rotonda o los restaurantes pertenecientes a ésta específicamente puedan recibir múltiples pedidos y puedan manejarlos apenas cuenten con los recursos disponibles. Es decir, pueden recibir pedidos a pesar de que se encuentren ocupados con otras transacciones. Por otro lado, el uso de Two Phase Commit aporta y asegura a la aplicación más atomicidad en el manejo de los pedidos ya que en este protocolo se valida la regla de negocio que establece que para que un pedido pueda ser completado exitosamente es necesario que al menos una de las rotondas cuente con disponibilidad para cada uno de los productos que pertenecen a este pedido, si esto no ocurre se cancela totalmente el pedido del usuario.

Las transacciones distribuidas en la aplicación RotondAndes permiten expandir el mercado y la capacidad de soportar pedidos y diferentes operaciones por parte de la aplicación. De esta manera, la aplicación permite hacer operaciones con una mayor concurrencia y su manejo se distribuye entre los participantes según es solicitado por cada requerimiento. A lo largo de la implementación, pudimos notar que las diferentes estrategias, $Two\ Phase\ Commit$ y $Colas\ de\ mensajes$ en nuestro caso, se acoplan mejor a ciertos requerimientos, dependiendo de las reglas de negocio. Por ejemplo, en el caso de las consultas, es mejor implementar la estrategia $Two\ Phase\ Commit$, pues cuando un cliente solicita información, desea que se garantice su entrega inmediata y no quedar en espera; adicionalmente, la posibilidad de bloqueo en este tipo de requerimiento no es muy alta, pues solo se solicita información más no se modifica esta misma. Por otro lado, de acuerdo a nuestras reglas de negocio, la solicitud de pedidos es mejor implementarlas con colas de mensajes, pues le permite al cliente realizar diferentes operaciones en la rotonda mientras espera la confirmación de su pedido.

En cuanto a una estrategia global escogida como la “más indicada” para RotondAndes, nos parece que no existe una única estrategia para esta aplicación, pues como bien se ha dicho a lo largo de este documento, la estrategia a implementar depende completamente de las reglas de negocio asociadas a cada requerimiento.