**ITERACIÓN 5 – TRANSACCIONES DISTRIBUIDAS**

**Grupo 3**

**Carlos Felipe Agudelo Ospina 201328150**

**Sergio Yodeb Velásquez Yepes 201315851**

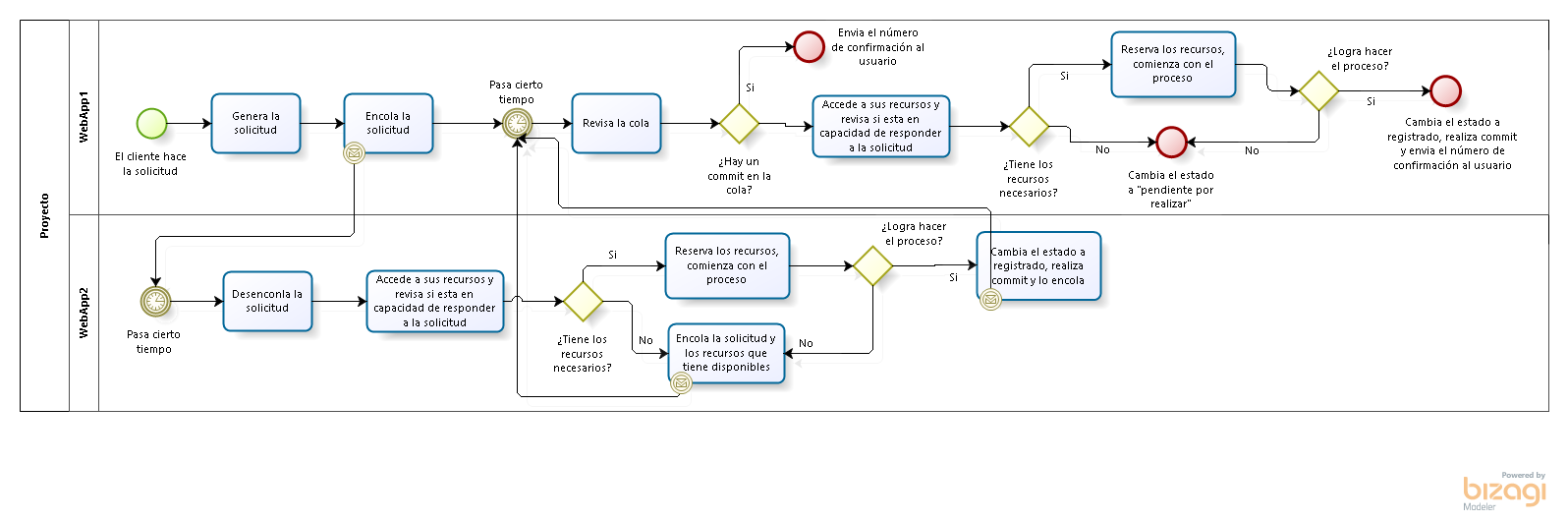
**Restricciones existentes para los dos proyectos**

A continuación se listarán ciertas restricciones encontradas en la implementación de transacciones distribuidas usando el estándar XA y Java Message Service (JMS):

* Uno o ambos sitios no han implementado la interfaz XA de alguna forma.
* La cola de mensajes no se encuentra bien definida, y una o ambas aplicaciones no tienen acceso a esta.
* El protocolo de comunicación a través de la cola de mensajes no está definido e implementado en alguna o ambas aplicaciones.
* El API de JMS puede estar mal configurado en alguna o ambas aplicaciones. Por ejemplo el API puede no tener las garantías de entrega correctas o el modo de confirmación incorrecto, por lo que las características ACID que se esperan lograr de la comunicación podrían no lograrse.
* El orden de commits puede estar mal definido en JMS, lo que puede dañar las características de transaccionalidad deseadas.
* Los recursos pueden estar mal definidos en una o ambas aplicaciones, por lo que los cambios y consultas en las bases de datos no podrían ser llevados a cabo.

**Análisis de estrategias sobre RF18**

**Diagrama de flujo**



**Estrategia Two Phase Commit**

Bajo esta estrategia el proceso comienza cuando el nodo coordinador manda la solicitud de pedido a ambas unidades. Al revisar la solicitud, cada unidad del programa debe verificar si posee los recursos necesarios para satisfacer el pedido, si esto no es así, puede que combinando los recursos varias unidades puedan satisfacer el mismo. Si este es el caso, las subunidades deben hacer commit de toda la transacción, reservando los materiales consumidos y registrando las fases de producción pendientes. Si todos estos commits son exitosos el nodo coordinador puede hacer commit de todo el pedido para que este sea registrado. Si ambas unidades no pueden satisfacer juntas el pedido es responsabilidad del nodo coordinador de registrar el pedido como pendiente o imposible de realizar. Si alguna de las unidades durante el proceso descrito antes hace rollback, el nodo coordinador debe asegurarse de darle la orden a todas las unidades de hacer rollback y de registrar el pedido como pendiente o insatisfecho.

**Estrategia cola de mensajes**

La unidad que registró la solicitud de pedido debe encolar el mensaje de solicitud, revisar sus materiales y ver si es capaz de realizar el pedido con sus propios recursos, si esto es así debe encolar este mensaje; de lo contrario debe encolar o bien los recursos que hacen falta o los recursos que dispone. La segunda unidad eventualmente lee la solicitud, y procede a verificar sus materiales, desencolando el mensaje de la primera unidad revisa si entre sus recursos y los otros la solicitud puede ser satisfecha. Si este es el caso hace commit y envía un mensaje que confirme el commit. La primera unidad posteriormente leerá dicho mensaje y hará commit por su parte. Si ambas unidades no pueden satisfacer el pedido la segunda unidad deberá informar esto en la cola de mensajes. Al leer el mensaje de fallo la primera unidad deberá hacer rollback de los cambios que realizó y registrar el pedido como pendiente.

**Análisis de impacto de estrategias**

La estrategia global elegida es cola de mensajes, dado que se considera que el API JMS ofrece más utilidad y garantías para mantener la transaccionalidad en las operaciones. Se considera además que utilizando JMS la implementación de balanceo de carga es más fácil.

**Estrategia para RF19**

La unidad que recibe la solicitud de cambio de una estación de producción encola la solicitud, revisando antes si la estación dada está registrada en su base de datos. Si este es el caso, la estación se debe activar o desactivar según sea el caso. Al desactivarse la estación las etapas de producción que le estaban asignadas deben ser reasignadas, por lo que todas estas se encolan para que posteriormente la primera unidad o la segunda se hagan cargo de estas. Cuando alguna de las unidades lee el mensaje de reasignación se encarga de asignarle la etapa de producción a la estación con menos carga en el mundo y desencolar el mensaje. Cuando todas las etapas que le estaban asignadas a la estación se desencolen se debe encolar un mensaje de commit para que ambas unidades hagan commit de la transacción. Si en algún momento alguna unidad hace rollback debe encolarse el mensaje para que la otra también lo haga.

Al activarse la estación se hace una solicitud adicional para que le sean asignadas distintas etapas de producción pendientes o asignadas a estaciones con mayor carga, al ocurrir esto si una unidad lee el mensaje deberá seleccionar una etapa de producción de la estación de producción con más carga que posea y asignársela. Después de que se le asigna, se vuelve a encolar la solicitud de carga, con un número que indique cuantas etapas de producción la estación lleva asignada. Cuando una unidad lea la solicitud de la cola deberá revisar si el número de etapas de producción que lleva la estación es mayor o igual a la cantidad de etapas que tiene asignada la estación con más etapas de sus recursos, si es mayor se para el proceso con un mensaje de commit, si es menor se continúa asignando. Cuando ambas unidades lleguen a un mensaje de commit se hace commit de la transacción y se termina el proceso.

**Estrategia para RFC12 y RFC13**

Al momento en que una unidad reciba la orden de consulta esta debe encolar la orden para que la otra pueda leerla y responder. Cada unidad deberá encargarse de encolar las respuestas de las consultas, para que la unidad en la que fue realizada la consulta pueda mostrar todos los resultados.

La estrategia tendrá como consecuencia un aumento en el tiempo que toma ejecutar cada una de las transacciones, dado que se tendrá que esperar a que cada una de las unidades lea mensajes de la cola y encole además sus mensajes.