Orquestación distribuida con Camunda BPM, Parte 1

Simon Zambrovski
21 de octubre · 9 min de lectura

La automatización de los procesos comerciales es el dominio principal y uno de los propósitos de los motores de procesos. Camunda BPM es un motor de proceso de código abierto muy popular y liviano, no es una excepción. Ejecuta BPMN 2.x directamente y proporciona una cobertura muy alta del estándar BPMN 2.x.

En los últimos años, Microservicios se convirtió en una de las tendencias más importantes en la industria moderna del software. No voy a hablar sobre los pros y los contras, las oportunidades y las trampas; solo observo un cambio en los proyectos de mis clientes para pensar en contextos separados en lugar de monolitos. Mi esfuerzo generalmente está en la creación de una arquitectura lo suficientemente buena para administrar esos contextos. Especialmente, los microservicios (y los sistemas reactivos) dieron un gran impulso a un enfoque antiguo pero importante del diseño impulsado por dominio (DDD). Independientemente de la distribución, fomenta la creación de contextos limitados y se ocupa de la integración entre ellos. Como artesano de BPM, me centro en la automatización de los procesos comerciales ejecutados en dichos entornos.

Para mí, la esencia del cambio realizado por esas tendencias hoy en día se trata de dos características: **autonomía** y **agilidad** . Desde el punto de vista de BPM, especialmente la autonomía crea requisitos adicionales para la Modelación e Implementación de Procesos. La autonomía en sí misma no tiene un requisito estricto sobre la distribución, por lo que podemos discutir este tema incluso sin distribución, pero en un contexto de Procesos de Negocio implementados a lo largo de Contextos Limitados.

Dos términos en Business Process Management están directamente relacionados con la autonomía: **Orquestación** y **Coreografía** . Clásicamente, la Orquestación denota una

ejecución guiada dentro de un contexto cerrado / acotado realizado por alguna entidad, en contraste con la Coreografía, que trata más sobre diferentes contextos múltiples cerrados / acotados que intercambian mensajes, al saber en qué mensaje reaccionar y cómo.

La orquestación se entiende bien dentro de un solo contexto limitado. El tema nació en la década de 1970 y tuvo tiempo suficiente para evolucionar, tanto en la academia como en la industria. Los lenguajes estándar como BPMN 2.x definen la semántica operativa para la orquestación y las herramientas modernas como Camunda BPM lo implementan de manera excelente.

Por otro lado, la idea de coreografía está claramente definida en contextos distribuidos. Tener contextos limitados independientes implementados como sistemas autónomos, administrados por diferentes equipos no permite un acoplamiento estrecho entre ellos. Para lograr un acoplamiento flexible, los mensajes se utilizan para definir la API entre esos sistemas. Los sistemas de mensajería proporcionan una entrega asincrónica que fomenta la transparencia de ubicación y la independencia de disponibilidad.

Desafortunadamente, solo hay pocas técnicas para implementar orquestaciones en la industria: una popular es seguir la Arquitectura dirigida por eventos (EDA) mediante el uso de una infraestructura de mensajería / bus de eventos común, como Apache Kafka o RabbitMQ. Bernd Ruecker de Camunda señaló las debilidades de esta solución con respecto a la complejidad, la escala y la evolución en su publicación . En resumen, funciona en escenarios pequeños y crea un alto acoplamiento entre sistemas / componentes individuales. Bernd propone **usar la orquestación en lugar de la coreografía** y domesticar el caos y generalmente estoy de acuerdo con él.

Al mismo tiempo, solo reemplazar la coreografía por la orquestación no resuelve todos los problemas. Me gustaría compartir mis pensamientos sobre esto, algunos patrones y algunas experiencias de implementación.

. . .

Patrones de descomposición en BPMN

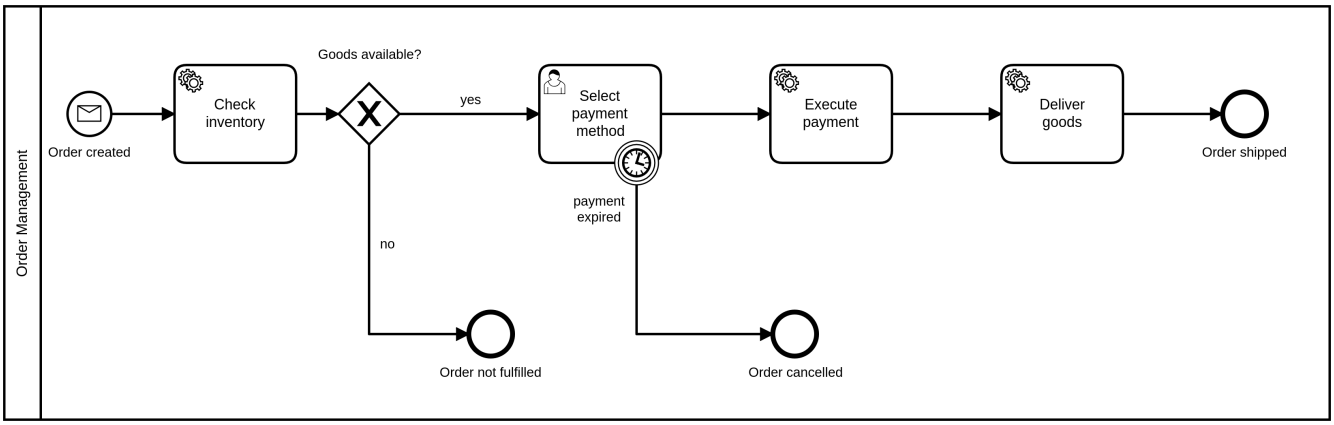
En resumen, el término DDD "Contexto limitado" se refiere al alcance con responsabilidad y el mismo lenguaje, que se utiliza principalmente para descomponer el dominio con el fin de hacer frente a la limitación de tamaño, complejidad e

integridad. BPM utiliza la descomposición de los procesos comerciales por los mismos motivos: para facilitar la gestión, el mantenimiento y el cambio de las piezas.

Por descomposición me refiero a una transformación del enfoque de modelado de procesos del modelo de proceso en otro modelo de proceso que es equivalente en términos de semántica operativa (= significado).

Proceso de ejemplo

Aquí hay un pequeño proceso de ejemplo utilizado para demostrar algunas características.

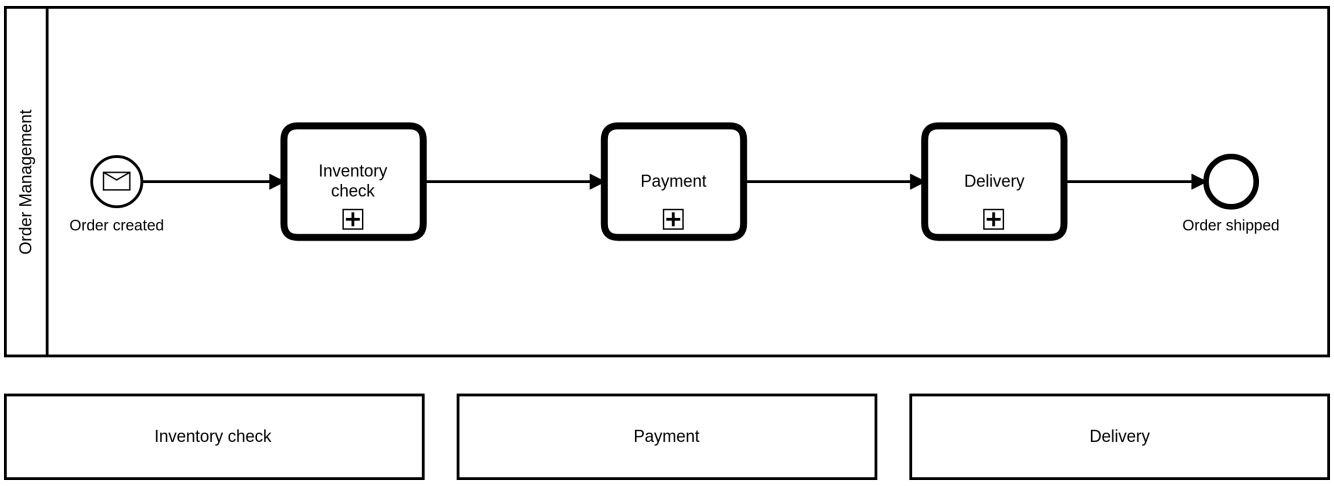


Proceso de gestión de pedidos que abarca múltiples contextos delimitados

El proceso de Gestión de pedidos se inicia en cada pedido creado y organiza varias capacidades comerciales con el objetivo de cumplir el pedido. Comienza con la verificación del inventario, continúa con el pago y finalmente desencadena la entrega.

Actividades de llamadas

El patrón de descomposición más simple en BPMN es el uso de actividades de llamada:

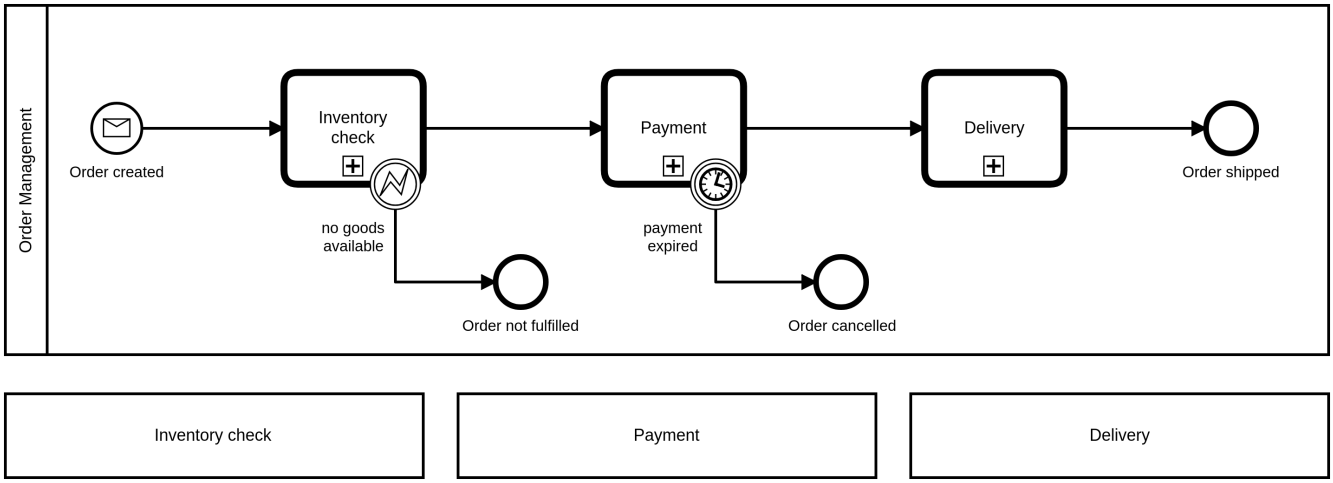


Actividades de llamada

En este sencillo ejemplo, el proceso de Gestión de pedidos delega la ejecución de Verificación de inventario, Pago y Entrega a tres procesos. La ejecución de la Gestión de pedidos se detiene en la actividad de la llamada hasta que finalice y continúe. Desde el punto de vista del alcance y la visibilidad, es irrelevante para la Gestión de pedidos cómo se implementan, ya que se consideran cajas negras.

En Camunda BPM, la actividad de llamada también define una API entre los dos procesos al proporcionar una asignación de entrada y salida variable. Esto permite mapear el alcance variable del proceso del llamante al alcance variable del llamado pasando, renombrando o creando nuevas variables y luego define el mapeo inverso para el resultado de la ejecución.

Además de la asignación de variables, la actividad de la llamada puede tener eventos de límite adjuntos, que influyen en la ejecución y las posibles rutas de retorno. Aquí hay un ejemplo, en el que los procesos de pago e inventario pueden terminar de manera diferente:

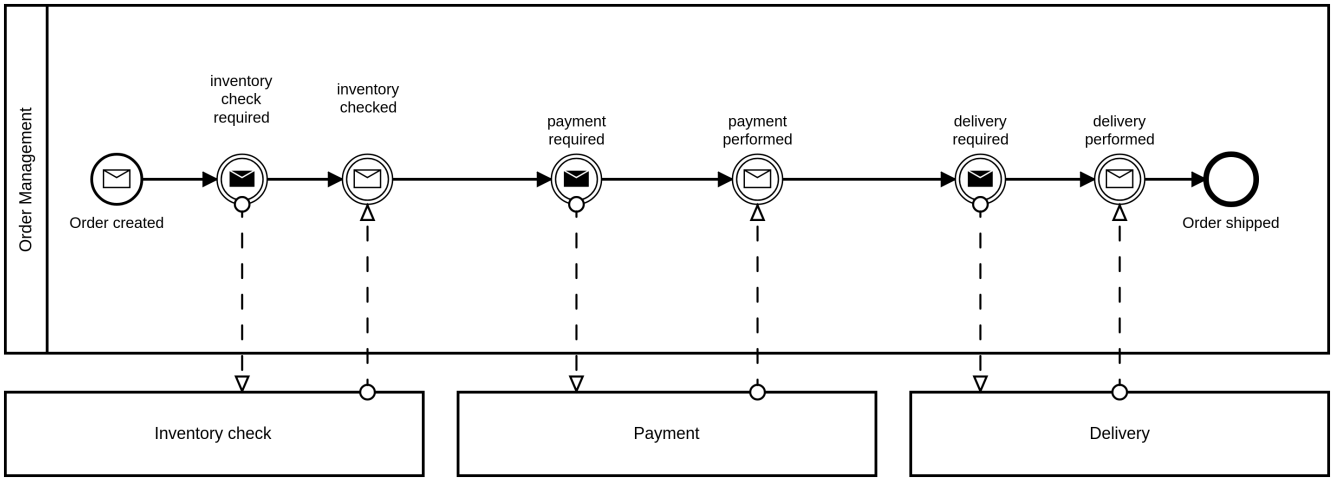


API con eventos límite

Mensajes

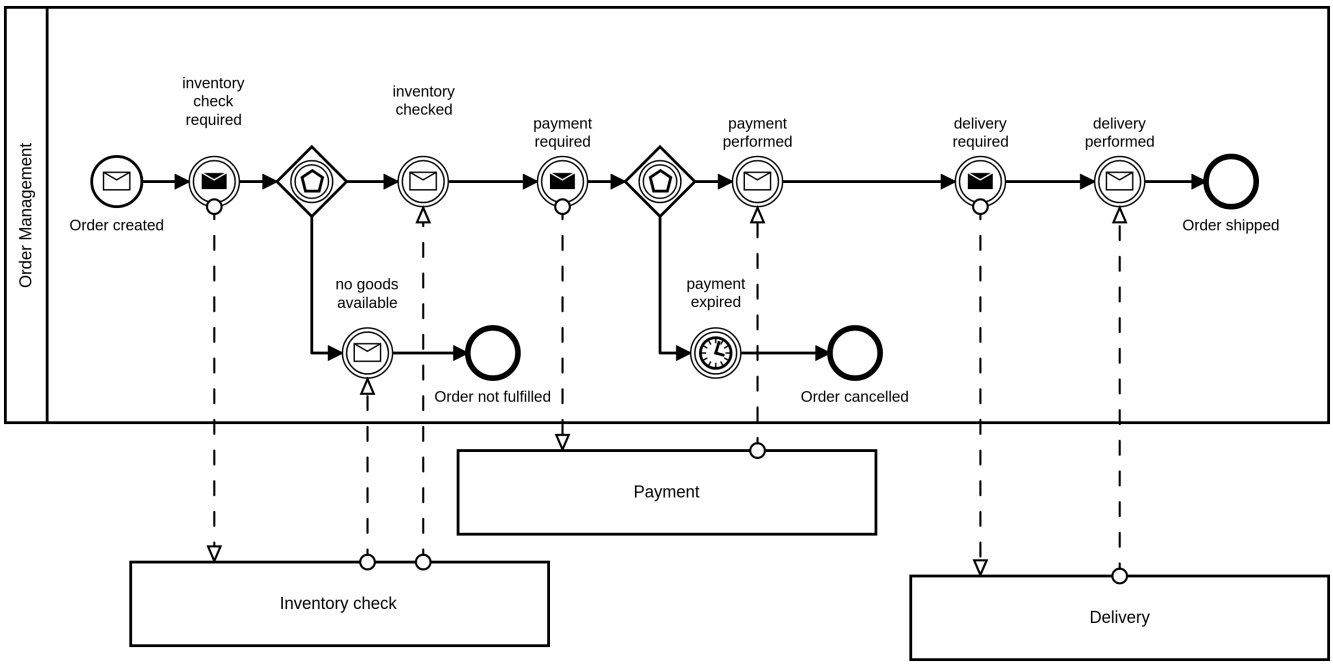
Otro enfoque popular para delegar la ejecución es el uso de mensajes BPMN. A diferencia de las actividades de llamada, la ejecución del proceso de llamada (Gestión de pedidos) no se bloquea durante la ejecución del proceso llamado. Esto crea una ejecución concurrente real a nivel conceptual, lo que puede resultar en una ejecución concurrente a nivel técnico. Si nuestro objetivo es proporcionar la misma semántica

utilizando mensajes proporcionados con actividades de llamada, se deben usar los pares de eventos de mensaje de lanzar / atrapar. Aquí hay un ejemplo:



Simule la actividad de la llamada con mensajes

Dado que la Gestión de pedidos delega toda la lógica a otros procesos, solo sirve como un orquestador puro, utilizado para traducir / mediar entre diferentes contextos. El modelo de proceso se vuelve más interesante si se incluyen las respuestas excepcionales:



Simule la actividad de la llamada con eventos límite con mensajes

Compare esto con el ejemplo de actividad de llamada. No hay ningún evento intermedio de error de captura que pueda asignarse al evento de error de límite, por lo que se debe usar un mensaje para indicar que "no hay productos disponibles".

. . .

Implementando patrones de descomposición

Usar actividades de llamada en el mismo motor

Camunda BPM proporciona soporte de primer nivel para la implementación de la actividad de llamadas. Todo lo que tiene que hacer es implementar ambos modelos de proceso en el mismo motor y especificar el mapeo de entrada / salida en el modelo de proceso de la persona que llama. Si la asignación de "todos" está bien, ya está, pero es una mejor idea **limitar el significado de las variables de proceso** a un modelo de proceso. Si su descomposición se realiza junto con los límites del contexto empresarial / capacidad empresarial, debe definir una asignación en una traducciónentidad para hacer cumplir el desacoplamiento de los modelos y contextos de procesos. Las actividades de llamadas se manejan completamente dentro del motor Camunda y no se requiere un código personalizado. Incluso las rutas excepcionales introducidas por el error BPMN lanzado por el proceso llamado y el tiempo de espera no necesitan intervención del desarrollador, pero se pueden definir en el modelo.

> *Hay un detalle importante de que la API (recuerde que esto consiste en las asignaciones de variables y los eventos de límite) está completamente definida por el proceso de la persona que llama. Usando los tipos de relación de contexto limitado DDD de Vernon, clasificaría esta relación como* ***Conformista.*** *Si se proporciona la asignación de variables, se establece una* ***capa anticorrupción*** *(ACL) entre los procesos.*

De hecho, para la ejecución del proceso llamado generalmente no hay diferencia si ha sido iniciado por otro proceso. Gracias a la API de Camunda, puedes descubrir esto analizando el gráfico de ejecución.

Uso de correlación de mensajes dentro de un motor

Las cosas se vuelven un poco más complicadas si usamos mensajes BPMN para la comunicación entre dos procesos implementados en el mismo motor. Camunda BPM admite eventos de captura de mensajes como elementos de primer nivel que funcionan sin código adicional (el motor creará una suscripción de mensajes y se correlacionará con esto, si ocurrió el evento de mensaje), los mensajes de captura son gratuitos. El envío de mensajes (eventos de lanzamiento de mensajes) funciona solo con la API de Camunda Messaging . El `RuntimeService` ofrece un montón de métodos para la

correlación de mensajes con el funcionamiento de ejemplo (a la espera en la suscripción mensaje) o iniciar una nueva instancia de proceso por mensaje.

> *La definición de API entre el proceso llamante y el proceso llamado se basa en los nombres de los mensajes. Como deben coincidir, el tipo de relación se puede clasificar como* ***Asociación.***

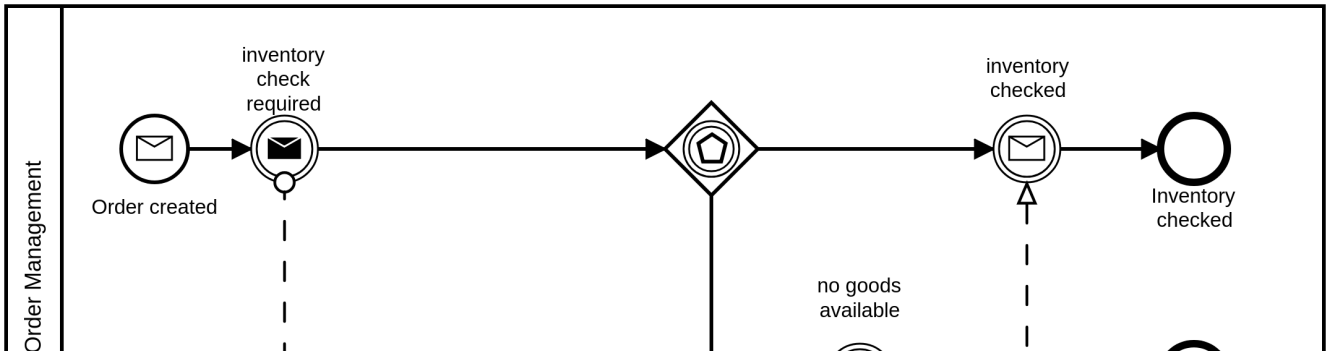
Al mismo tiempo, las variables se pueden establecer (y generalmente se establecen) durante la correlación de mensajes, lo que significa que el estado del proceso llamado cambia el estado interno del proceso llamado. La característica permite una integración fluida entre los procesos que se ejecutan en el mismo contexto acotado, pero debe usarse cuidadosamente a través de los límites del contexto. Este patrón introduce una violación del principio de encapsulación, en caso de actividades de llamada simuladas, en ambas direcciones. Esto se puede evitar mediante la creación de una capa anticorrupción (ACL), que consta de las siguientes reglas:

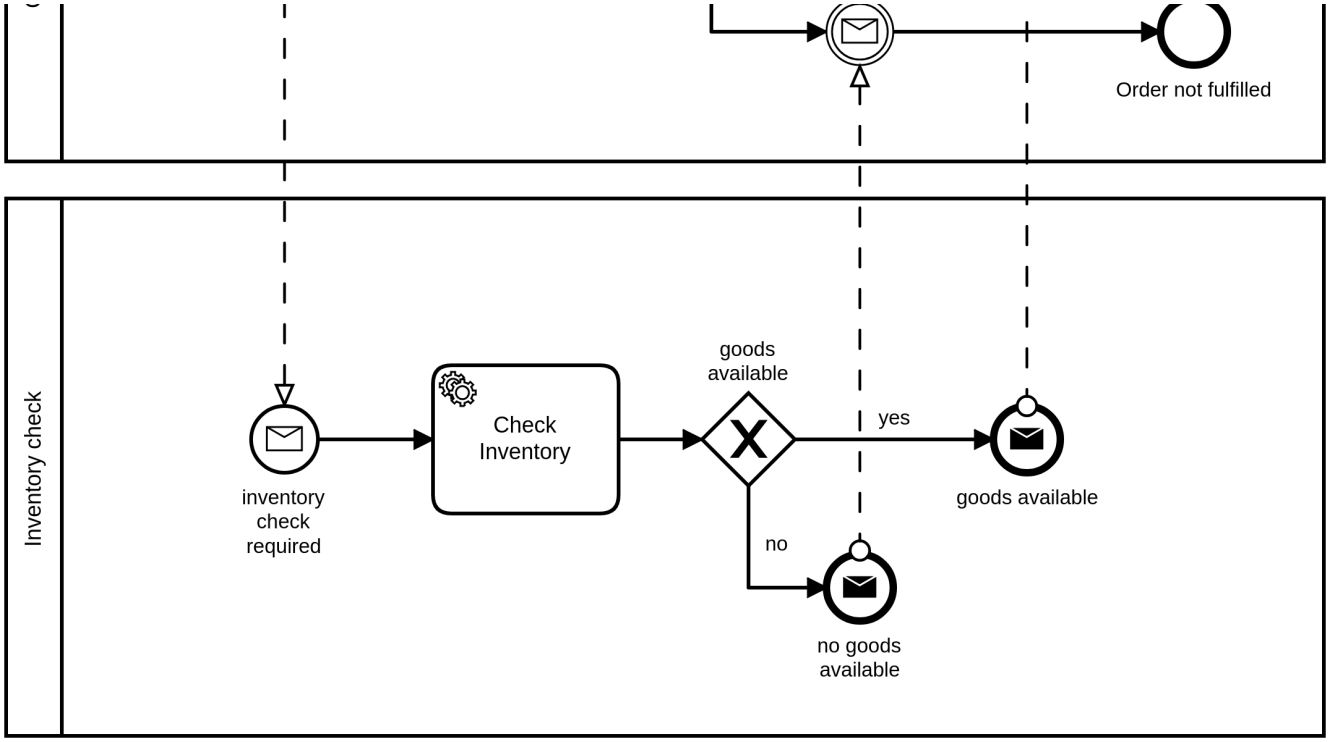
- La correlación no debe establecer variables de proceso (globales)
- La correlación puede establecer variables de proceso locales
- Un oyente especial de ACL conectado directamente al elemento de evento Message Catch se traduce entre esas variables locales y las variables del proceso de recepción

Esta solución conduce a un mejor desacoplamiento de los contextos de proceso, pero a costa de una mayor complejidad técnica.

Experimentando problemas de concurrencia dentro de un motor

Además, la implementación de tales procesos en el motor no es trivial con respecto a la concurrencia y el tiempo. Imagine el siguiente fragmento de modelo de proceso tomado del ejemplo anterior utilizando correlaciones de mensajes para la descomposición del proceso:





Fallo debido a razones de concurrencia

El evento de lanzamiento de mensajes de la Gestión de pedidos tendría una implementación delegada (simplificada y escrita en Kotlin):

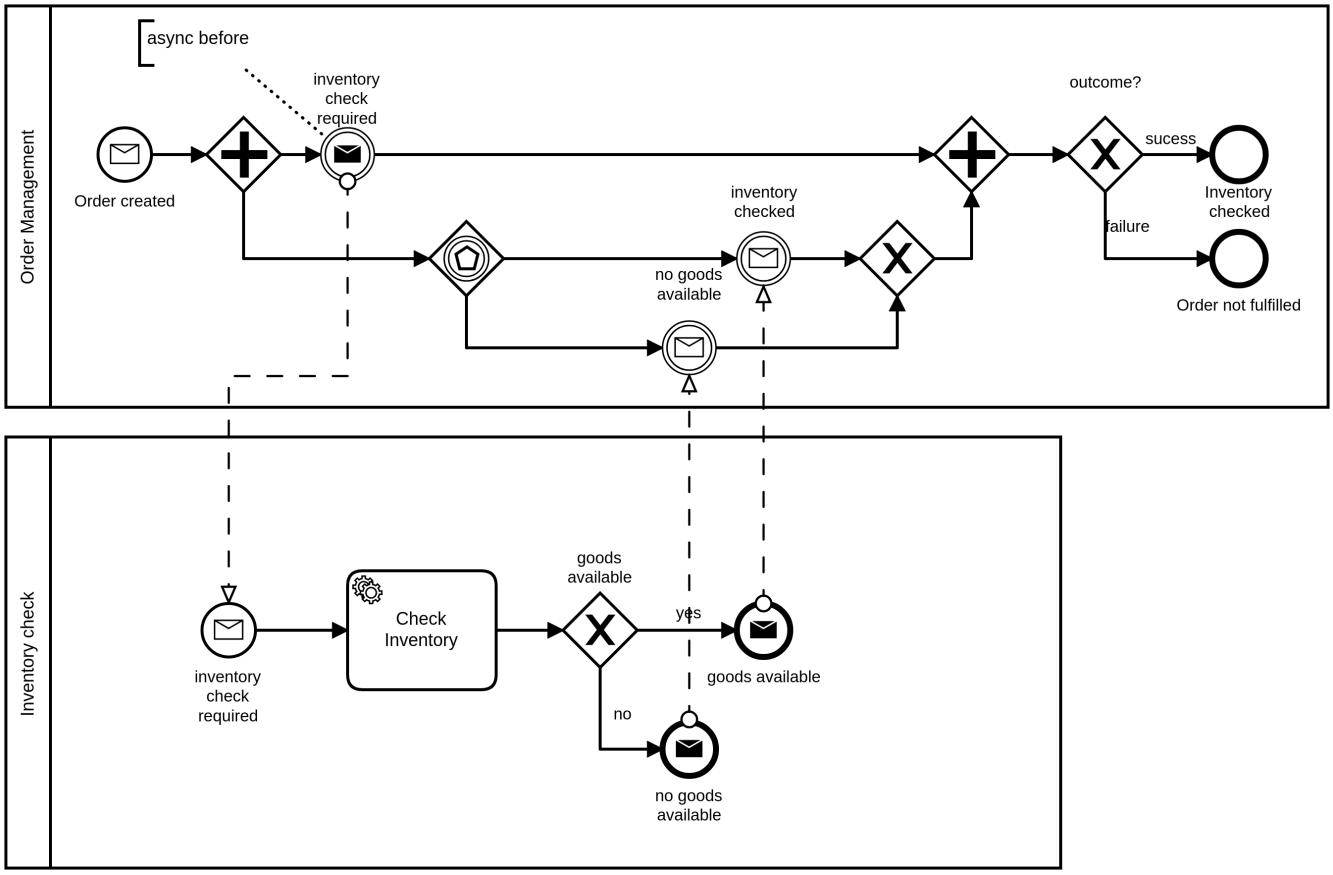
```
@Bean
fun sendInventoryCheckRequired (runtimeService: RuntimeService) =
JavaDelegate { ejecución ->
  val variables = mapa (ejecución. Variables )
  runtimeService
    .createMessageCorrelation ( "Inventory_check_required" )
    .processInstanceBusinessKey ( execute . ProcessBusinessKey )
    .setMarket
    )
.
```

Este código **NUNCA** funcionará si no hay `async` un evento de Inicio del Proceso de inventario o `Check Inventory` tarea de servicio. Esto sucede porque la llamada de la API de correlación hará que se ejecute todo el proceso de Verificación de inventario en una transacción **ANTES de** regresar y fallará al correlacionar el mensaje resultante, ya que la Administración de pedidos aún no tiene la oportunidad de crear una suscripción de Mensaje. Por lo tanto, no hay ningún problema de concurrencia aquí, pero el proceso no es ejecutable.

Este código **POTENCIALMENTE** fallará si solo inserta el `async` atributo en el evento de inicio, porque al hacerlo está creando una condición de carrera entre los dos procesos y

supone que el proceso de Gestión de pedidos es más rápido que Verificar inventario. En este caso, existe un problema de concurrencia y se resuelve mediante la implementación de Camunda BPM en función de muchos factores como Job Executor, etc.

Hay una solución que funciona en Camunda BPM por el hecho de que la puerta de enlace paralela se implementa de cierta manera. Aquí está cómo modelarlo:



Posible solución para enviar / recibir en el mismo motor

Esta vez, el `async before` evento Lanzamiento de mensaje forzará al motor a enviar el evento en la próxima transacción, pero creará las suscripciones a los mensajes en la misma transacción. Dado que recibimos mensajes en la rama de una ejecución paralela, debemos unirnos y luego decidir el resultado. Como puede ver, el intercambio de mensajes simple resultante del ejemplo de Actividad de llamada está contaminado tanto en el modelo BPMN como en la implementación técnica.

Conclusión

En este artículo, compartí mis pensamientos sobre el uso de procesos de negocios en el contexto de Microservicios o simplemente Contextos delimitados. Veo dos patrones

principales de BPMN sobre cómo se puede lograr la descomposición del proceso de negocio en partes más pequeñas:

- utilizando actividades de llamada
- usando correlaciones de mensajes

Las implementaciones de esos patrones, incluso en un solo motor o clúster homogéneo (lo cual está bien para evitar que el sistema sea seguro, implementa el mismo software en todos los nodos que comparten la misma base de datos) conduce a algunos desafíos conceptuales y de implementación:

- El uso de actividades de llamada establece una relación "conformista" entre el proceso de orquestación y el proceso llamado. Esto es especialmente un problema si desea evitar el monolito de procesos de negocio y tiene un gran proceso de orquestación que llama a muchos procesos pequeños.
- La correlación de mensajes dentro de Camunda BPM se basa (intencionalmente) en el enfoque que viola el principio de encapsulación. Esta característica es muy útil en un contexto monolítico coherente o solo dentro de un contexto limitado, pero es una limitación si se aplica a la comunicación entre diferentes contextos. Se puede evitar a costa de una mayor complejidad técnica. Además, no es fácil de implementar con respecto a los problemas de concurrencia, que conllevan el costo de la contaminación del modelo de proceso con detalles de implementación.

Ambos resultados no son sorprendentes. BPMN no está diseñado para expresar o abordar problemas relacionados con la distribución o la ejecución concurrente. Al mismo tiempo, Camunda BPM es un motor de proceso, históricamente diseñado para ser un orquestador central (el modelo de operación de "motor compartido" todavía está disponible y es utilizado por los clientes). No es compatible ni aborda ningún problema de los sistemas distribuidos y especialmente de los microservicios, pero aún puede utilizarse en estos entornos con éxito, si se aplican ciertas pautas.