Relatório: Padrão de Projeto *Process Orchestrator*

Christian E. Barbosa[1], **Elizadora M. da Silva**[1],
Augusto C. A. de Oliveira[1], **Lazaro L. Junior**[1], **Venicius F. da Silva**[1]

[1]Universidade Federal do Ceará
Quixadá – Ceará – Brasil

christianestevam@alu.ufc.br, elizadora.mendonca@alu.ufc.br,
augustces@alu.ufc.br, lazarojunior@alu.ufc.br,
venifeitosa@alu.ufc.br

Abstract. *This report presents the design pattern called* Process Orchestrator*, which aims to solve the problem of coordinating interdependent processes in a microservices architecture without direct coupling. The pattern uses asynchronous events (e.g., RabbitMQ [RabbitMQ 2023]) to manage workflows. The primary example discussed is the e-commerce order processing system.*

Resumo. *Este relatório apresenta o padrão de projeto denominado* Process Orchestrator*, voltado para solucionar o problema de coordenação de processos interdependentes em uma arquitetura de microserviços sem acoplamento direto. O padrão utiliza eventos assíncronos (por exemplo, RabbitMQ [RabbitMQ 2023]) para gerenciar workflows. O exemplo principal discutido é o processamento de pedidos em um sistema de e-commerce.*

1. Introdução

Em sistemas distribuídos baseados em microserviços [Newman 2015], é comum que diferentes serviços precisem executar etapas de um processo de forma interdependente. Um exemplo clássico é o fluxo de um pedido em e-commerce, que envolve desde a criação do pedido até a emissão da nota fiscal. A implementação tradicional, que utiliza chamadas diretas entre serviços, pode gerar alto acoplamento, dificultar manutenções e tornar o sistema mais vulnerável a falhas.

O padrão *Process Orchestrator* surge como uma solução para centralizar o controle do fluxo de execução, delegando a coordenação das etapas a um componente centralizado que opera de forma desacoplada dos serviços envolvidos.

2. Problema

Em ambientes de microserviços, o acoplamento excessivo entre os serviços pode levar a:

- Dificuldade para modificar ou estender o fluxo de execução.
- Problemas de disponibilidade, onde a falha de um serviço pode comprometer o processo inteiro.
- Complexidade na gestão de dependências entre os componentes.

No cenário de um sistema de pedidos de e-commerce, por exemplo, a criação de um pedido envolve várias etapas: validação do estoque, processamento do pagamento, emissão da nota fiscal e envio de notificações. Se cada etapa chamar diretamente a próxima, a manutenção e evolução do sistema tornam-se complicadas.

3. Solução Proposta: Process Orchestrator

O padrão *Process Orchestrator* propõe a centralização da coordenação dos processos através de um orquestrador, que gerencia o fluxo de execução por meio de eventos. Os principais componentes do padrão são:

- **ProcessOrchestrator**: Responsável por coordenar e disparar as etapas do processo.
- **ProcessStep**: Interface que define uma etapa do processo.
- **ConcreteStep**: Implementações concretas das etapas.
- **StepRegistry**: Componente que armazena e organiza a sequência de execução das etapas.
- **EventQueue**: Mecanismo de mensageria (como o RabbitMQ) utilizado para envio e recepção dos eventos que desencadeiam cada etapa.

A vantagem desta abordagem é a redução do acoplamento entre os serviços. Cada etapa é acionada por um evento e o orquestrador decide qual a próxima ação, permitindo implementar estratégias de retry ou de fallback em caso de falha.

4. Arquitetura e Implementação

A implementação apresentada utiliza Java com Spring Boot e integra o RabbitMQ para gerenciar a comunicação assíncrona entre as etapas do processo. A API REST permite consultas sobre o status do pedido, do pagamento e da nota fiscal.

A Figura 1 ilustra um diagrama de classes simplificado do padrão:

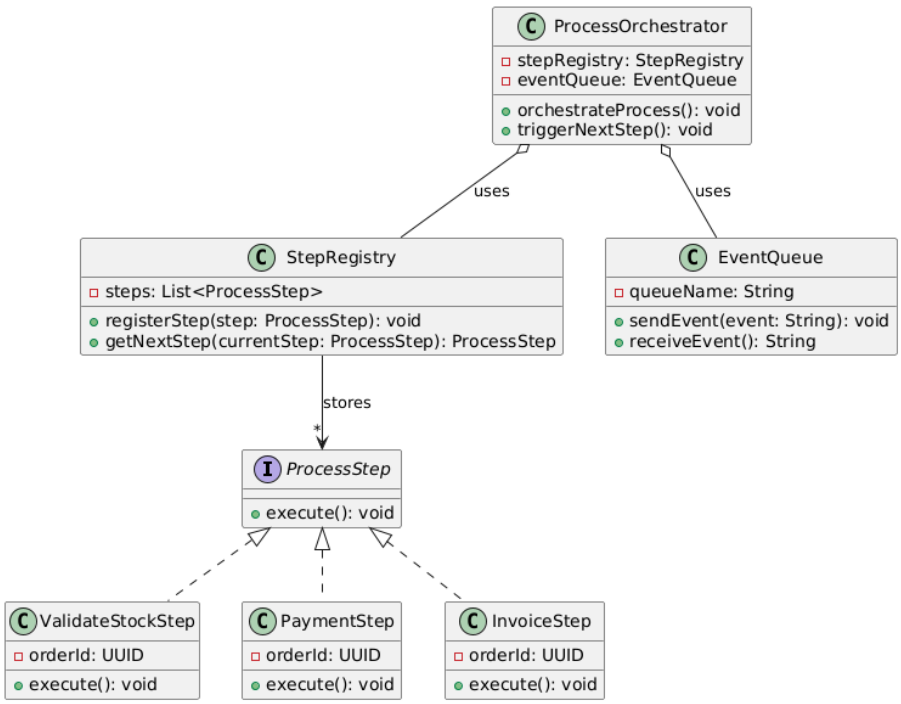

Figura 1. Diagrama de classes simplificado do padrão *Process Orchestrator*

Neste diagrama, podemos observar a centralização do fluxo no `ProcessOrchestrator` e as diversas implementações da interface `ProcessStep`, que representam as etapas do processo.

5. Exemplos de Uso

Atualmente, o exemplo de e-commerce demonstra a coordenação das seguintes etapas:

1. Criação do Pedido.
2. Validação do Estoque.
3. Processamento do Pagamento.
4. Emissão da Nota Fiscal.

Para expandir a demonstração do padrão, outros cenários podem ser considerados:

- **Workflow de Aprovação de Documentos**: Um documento passa por revisão automática e, se necessário, por revisão humana, antes de ser aprovado ou rejeitado.
- **Integração de Sistemas de Saúde**: Coordenação entre serviços que validam, processam e armazenam informações de pacientes, garantindo a consistência dos dados.

Esses exemplos demonstrariam a versatilidade do padrão *Process Orchestrator*, adaptando-se a diferentes cenários que requerem a coordenação de processos de forma desacoplada e robusta.

6. Conclusão

O padrão de projeto *Process Orchestrator* traz uma solução robusta e escalável para a coordenação de processos em ambientes distribuídos e baseados em microserviços. Ao centralizar o controle do fluxo e utilizar eventos assíncronos para disparar cada etapa, ele promove um desacoplamento eficiente entre os serviços, facilitando a manutenção do sistema. A implementação do exemplo de e-commerce ilustra claramente como o padrão pode ser aplicado para gerenciar workflows complexos, como o processamento de pedidos, de forma modular e confiável.

Referências

Newman, S. (2015). *Building Microservices*. O'Reilly Media, 1st edition.

RabbitMQ (2023). Rabbitmq – the open source message broker. `https://www.rabbitmq.com/`. Acessado em: 24 de fevereiro de 2025.