Software Microservices Patterns
(Orchestration Vs Choreography)

Amit Sengupta (Independent Research)

Email – amits2913@gmail.com
Independent Researcher

Abstract

*The **Microservice Architecture** is a collection of small services with each service having a specific function. Application and business use case is broken up into a set of composable services which promise many technical benefits to enterprise organizations. They are small, lightweight, easy to implement, enable reusability that reduces the cost of developing or changing applications, ensures the efficient use of resources, and makes it easy to scale applications on demand. To implement larger application processes, these microservices must communicate and collaborate. Typically, this follows one of two patterns:*

1. ***Choreography,** in which communication is done via asynchronous message passing.*
2. ***Orchestration,** in which a controller is used to synchronously manage the process flow.*

Choosing the right pattern requires the resolution of some trade-offs concerning coupling, chattiness, visibility, and design. To address this problem, we propose a decision framework for microservices collaboration patterns that helps solution architects to crystallize their goals, compare the key factors, and then choose a pattern using a weighted scoring mechanism. In cases where there is no clear preference, a hybrid pattern is suggested which inherits some strengths of both choreography and orchestration. We are not aware of any existing decision frameworks to guide solution architects in choosing a microservices collaboration pattern.

Keywords: - *Microservices Orchestration, Microservices Choreography, Domain Driven Design, Coupled Vs Decoupled, Distributed Workflow, Event Based, Service Oriented Architecture*

Introduction: -

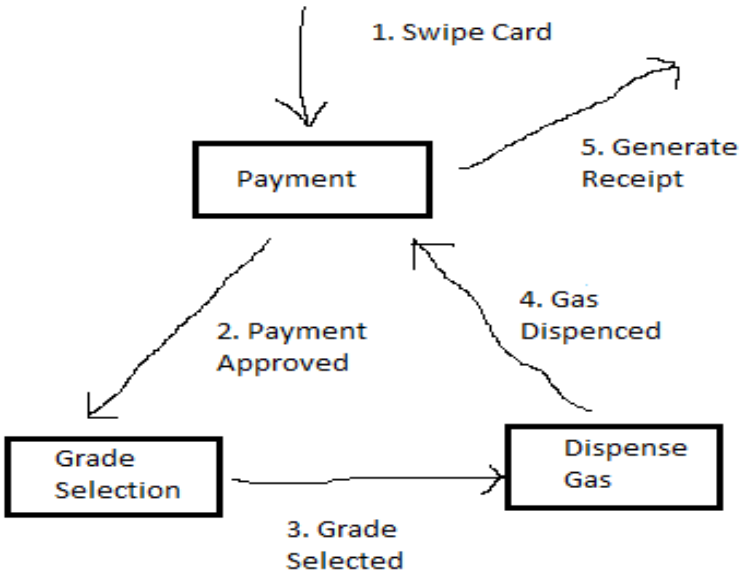
Microservices constitute an implementation approach to Service-Oriented Architecture (SOA) principles and patterns, with emphasis on service development and deployment using modern software engineering tools and practices. Microservices are characterized as being modular, small in size, independently deployable, and organized around business capabilities. These characteristics allow for the components of complex applications to be independently monitored, tested, updated, or scaled; benefits that led to the adoption of microservices by companies such as Amazon, Netflix, and Uber. Microservices have also been deployed in more traditional enterprises, such as banking, due to their effectiveness at integrating or replacing parts of monolithic legacy systems.

Business Context: -

Decomposing a complex application into a collection of microservices also introduces some challenges, most notably, in determining how they should collaborate in order to implement different application processes. In practice, microservices typically collaborate according to one of two different design patterns: -

Choreography: -

In a typical jazz band, there is no conductor, players will improvise other players. Everyone typically knows what they are supposed to be doing, and is required to hit the right notes, as each beat hits. This is exactly how each bounded context work together in choreography. Taking example of filling gas in a car, they would emit an event for every action that was executed on and the typical architecture would look something like below: -



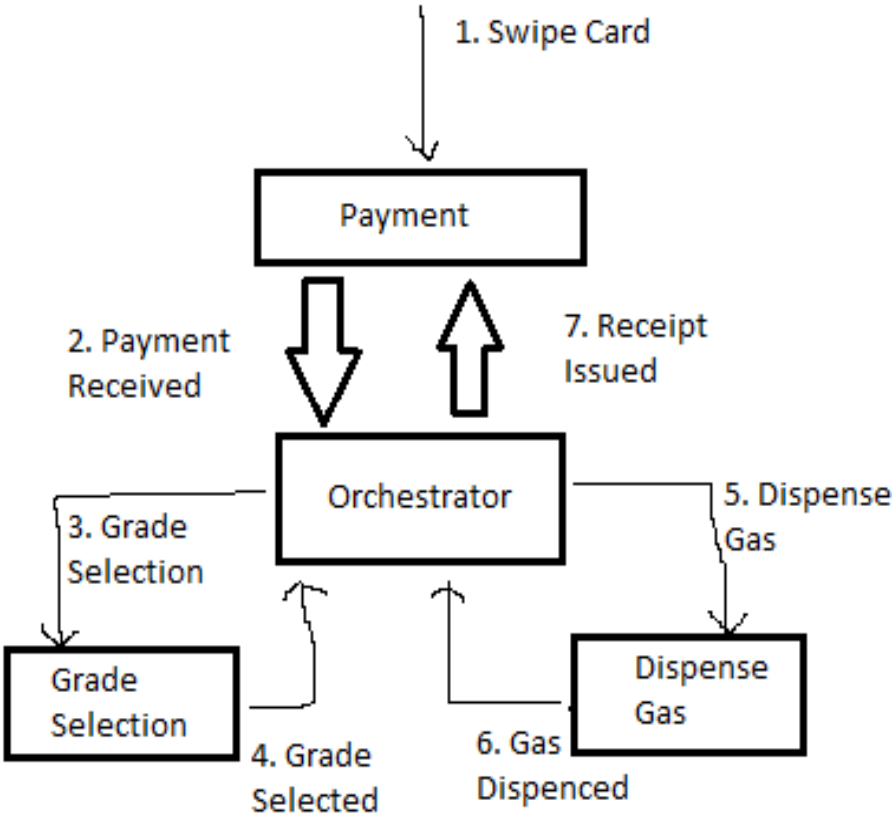
Choreography is driven by events from the various services within a system. Each service consumes events, performs an action, and publishes events. In this pattern, each bounded context would register the events they're interested in. Whenever this event is emitted, the bounded context takes the event and performs certain actions, then emit another event based on the change. Each component only knows about things that they are interested in.

Orchestration: -

In an orchestra, each musician is awaiting command from the conductor. They are each an expert at playing their instrument, whether it be a violin, bass drum or clarinet, have the sheet music for their part, however they'd be collectively lost without the conductor.

This is similar to the orchestration pattern in the sense that typically there would be a process manager (conductor) that handles all incoming events. It would know what to do with every event. It then converts the event to a command and sends it to the relevant bounded context.

With the same example, an orchestration pattern would look something as highlighted below: -



An extra Orchestrator component is introduced compared to choreography. This component is sometimes also called *Process Manager*. It is responsible for all the events from all bounded contexts that are responsible for a particular scenario. In this case, the whole system. Whenever an event is received, it converts this event to a command and sends the command to the relevant bounded context. In this pattern, each bounded context now knows nothing about any other. In fact, it doesn't even need to know about Orchestrator, however Orchestrator knows the entire process.

Decision Framework for Pattern Adoption: -

We have come up with a framework for deciding which microservices collaboration pattern to use. The framework employs a set of six collaboration factors derived from factors like –

1. Distribution of microservices across the wide area network (WAN), impacting 'chattiness'
2. Predictability of response time from the application process
3. Loose coupling of microservices to the process, impacting deployment
4. Reusability of microservices
5. Overall Complexity of the microservice design
6. Runtime visibility of the application process flow.

As a first step, the company's architecture team, or lead architect, is meant to assess the general ability of each microservices collaboration pattern to handle/provide the six collaboration factors, and then establish this assessment as an internal **Technology Architecture Standard** to guide on-going solution architectures. In terms of 'Ability', each collaboration factor is to be assessed using a five-point Likert scale; **1**-Incabable, **2**-Slightly Incapable, **3**-Neutral, **4**- Capable, **5**-Very Capable. A worked example for a particular enterprise is provided in table below –

Decision Factors	Choreography	Orchestration
Distribution Across Network	5	2
Predictability Of Response Time	3	5
Loose Coupling	5	3
Reusability	1	5
Design Complexity	2	5
Runtime Process Visibility	2	4

For each on-going solution, the solution architect together with business users are meant to assess the priority of each of the six collaboration factors, in satisfying the business need of the solution. In terms of business 'Priority', each collaboration factor is to be assessed using a five-point Likert scale; 1-Not a Priority, 2-Low Priority, 3-Medium Priority, 4- High Priority, 5-Essential. For each of the six collaboration factors, the assessed priority is then multiplied by the assessed ability for each collaboration pattern. The total score for each collaboration pattern then serves as an indication to the solution architect as to which collaboration pattern should be selected for the solution.

The framework proposed in this paper is meant to guide the solution architect in deciding which microservice collaboration pattern to use for specific application processes. However, it is important to note that both collaboration patterns, as well as the 'hybrid' pattern, may be employed concurrently across a company's multiple application processes, on a case-by-case basis, depending on the design goals of each application.

Conclusion: -

RESTful API-only approach is getting old, for architecture that delivers business services faster and more reliably, and that scales with ease. Orchestration and choreography don't have to be mutually exclusive. The rule-of-thumb, when it comes to implementing business workflows would be to use orchestration within the bounded context of a microservice but use choreography between bounded-contexts. The main point to remember is to try and minimize/remove direct point to point communication between the business services.

In this article, we have compared the two main microservice collaboration patterns - choreography and orchestration and have proposed a decision framework to help solution architects choose appropriate patterns for their applications. For future work, we intend to test our decision framework with industry partners who are in the process of migrating from a legacy monolithic application architecture to a microservices-based architecture.

Reference:

1. https://doi.org/10.1109/ACCESS.2020.2980891
2. https://doi.org/10.1007/s10270-017-0653-2
3. https://doi.org/10.1109/TSC.2017.2732988
4. https://doi.org/10.1016/j.ifacol.2020.12.1961
5. https://www.wallarm.com/what/orchestration-vs-choreography
6. https://github.com/AdyKalra/technolgytrends/blob/master/Architecture%20trends/Choreography%20vs%20Orchestration%20-%20serverless.md