Software Development
Group ARRO Lyngby
09/09/2024

Systems Integration

OLA 1

Name	Mail
Andreas Fritzbøger	cph-af167@cphbusiness.dk
Owais Dashti	cph-od42@cphbusiness.dk
Rasmus Taul	cph-rt91@cphbusiness.dk
Rabee Fawzi Abla	cph-ra157@cphbusiness.dk

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Contents

1. Links

1.1. GitHub Repo

GitHub Repo

1.2. Video presentation

Video on YouTube

2. About

This report provides an overview of current practices, patterns, and technologies related to Enterprise Integration. It explores the design and implementation of distributed, scalable applications that rely on various services or modules interacting effectively. The report covers both monolithic architectures and microservices, highlighting their benefits and challenges. Additionally, it explores diagramming standards, integration patterns, and trends in enterprise integration, such as cloud-based integration, data-centric approaches, and runtime analytics.

3. Stacks

3.1. Tech Stack

Version Control Platform: Git + GitHub
Text Editing: Typst, Google Drive
Desk: Microsoft Edge, Google Chrome

3.2. Development Stack

Typescript + Node.js / C#

4. Introduction to Enterprise Integration

Enterprise Integration involves the design and implementation of distributed, scalable applications where various services or modules communicate and interact with each other. These services can be part of a monolithic application, where all components are tightly coupled, or they could be microservices, which are independently deployable modules that interact over a network.

The primary goal of Enterprise Integration is to enable seamless data exchange and automate business processes across different systems within an organization. This process often involves integrating legacy systems, new applications, and cloud services, ensuring they work together to achieve some organizational goal.

4.1. Monolithic vs. Microservices Architectures

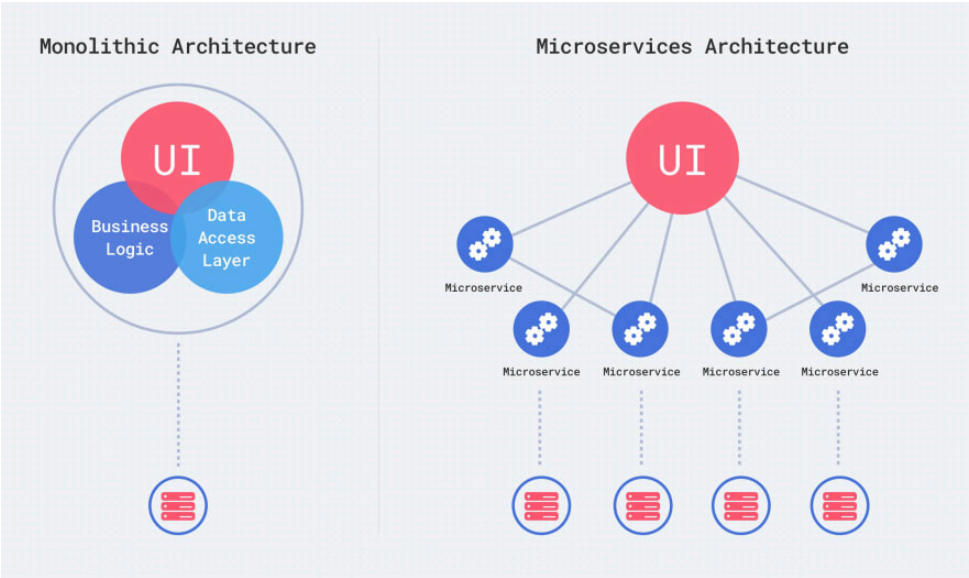


Figure 1: Monolithic Architecture vs Microservices Architecture

In a monolithic architecture, all components of an application are interconnected and interdependent, forming a single, cohesive unit. The application is built and deployed as one complete package, sharing the same codebase, database, and runtime environment. While this approach can be simpler to develop initially, requiring less overhead in terms of infrastructure and communication between components, it often becomes challenging to scale and maintain as the application grows. Updates or changes to one part of the application can cause unexpected effects across the entire system. For example, considering a traditional e-commerce application where the user interface, payment processing, product catalog, and order management are all bundled together in a single codebase. If a developer needs to update the payment processing logic, they must rebuild and redeploy the entire application. This increases the risk of unintended side effects, requires extensive regression testing, and can lead to downtime, especially if the application is large and complex. Additionally, scaling the application to handle increased traffic typically involves duplicating the entire application instance, which can be resource-intensive and inefficient.

In contrast, a microservices architecture breaks down the application into smaller, independent services that communicate with each other through APIs. Each service is responsible for a specific business function, such as user authentication, inventory management, or payment processing. These services can be developed, deployed, and scaled independently, providing greater flexibility and resilience. This architecture offers several benefits, such as improved scalability, where each service can be scaled based on its specific demand, and fault isolation, meaning that if one service fails, it does not necessarily bring down the entire application. Additionally, microservices enable easier deployment and continuous delivery, allowing individual services to be updated or rolled back without affecting the rest of the application. Examples of this architecture can be seen in companies like Netflix and Amazon. For instance, Netflix has over a thousand microservices handling different functionalities, such as user preferences, recommendations, and content delivery (video-streaming). This allows Netflix to scale its services independently, so if a new show is released and a surge in traffic is expected, only the relevant services (like the content delivery) need to be scaled, rather than the entire application. Similarly, Amazon uses microservices to manage its huge e-commerce platform, where services such as payment processing, inventory management, and user authentication are decoupled

and can be modified or scaled independently. However, while microservices provide these advantages, they also introduce some complexities. Managing distributed data across multiple services requires careful planning to ensure consistency and integrity, often involving additional technologies like distributed databases or messaging queues. Communication between services can be error-prone and needs to be designed to handle network latency, retries, and potential failures. Implementing robust monitoring, logging, and security across a distributed system can also be more challenging compared to a monolithic setup.

To implement Enterprise Integration effectively, several tools and frameworks are commonly used to facilitate communication between services and ensure data consistency. Examples include **Apache Kafka**, a distributed event streaming platform that can handle real-time data feeds and integrate various systems in a scalable way, and **RabbitMQ**, a message broker that supports multiple messaging protocol and patterns enabling message delivery and routing between services. These tools help manage the complexities associated with both monolithic and microservices architectures by providing reliable communication, data consistency, and integration between various components.

5. Diagramming Standards

Diagrams are essential tools for visualizing, designing, and understanding how a system or process works. Each type of diagram highlights different aspects of a system and serves various purposes depending on the audience and the objectives of the representation.

5.1. When to use diagrams

Choosing the right diagram depends on a few things:

- Purpose: What is the objective of creating the diagram?
- Audience: Who will be viewing or using the diagram (e.g., customers, stakeholders, developers).
- Focus: Which part of the system do we need to highlight or explain visually?

5.2. When not to use diagrams

We want to avoid using diagrams if:

- The workflow is too simple.
- The system or process lacks components or interactions
- The explanation of how the program works is more straightforward and effective with text.

Below is an overview of popular types of diagrams used in enterprise integration:

5.3. BPMN (Business Process Model and Notation)

BPMN is a standard for mapping and visualizing business processes. It provides a standardized way to describe how tasks are performed and how different elements within a process interact with each other. Represented using flowchart-like diagrams, BPMN helps in understanding, managing, and improving business workflows by clearly illustrating activities, events, and decisions.

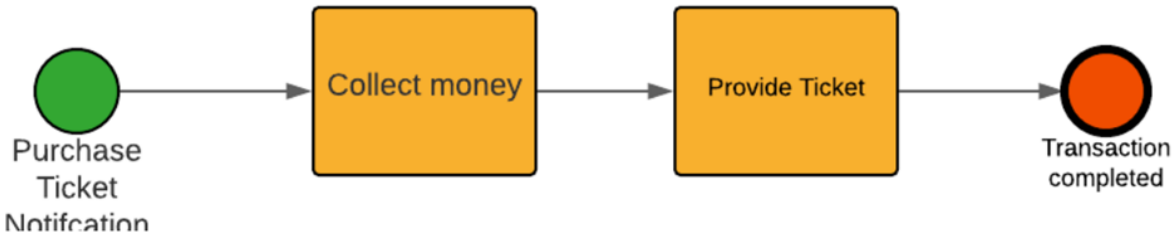
- Focuses on business processes and workflows.
- Not object-oriented; emphasizes the sequence of steps in a process.

Key Elements of a BPMN Diagram:

- Start Event: This shows where the process begins.
- Tasks: Each task represents a step or action in the process.

- Gateways: These are decision points. They show where the process might go in different directions based on a “yes” or “no” decision.
- Flows: Arrows show the order in which steps happen, connecting the tasks and events.
- End Event: This shows where the process finishes.
- Swimlanes: These are like lanes on a racetrack, separating different participants (customer and system) to show who does what in the process.

Using these simple elements, BPMN diagrams make business processes easy to understand and improve.



5.4. UML (Unified Modeling Language)

UML is an object-oriented language used to model and design software applications. It visualizes the interactions between objects (like classes and instances) within a system. UML is primarily focused on software and system design and helps developers and stakeholders understand the architecture and progress of the system. UML has some benefits:

- Simplifies complex ideas and systems.
- Represents complicated lines of code visually.
- Helps developers comprehend system design and development.

UML has different types of diagrams:

- UML Activity Diagram.
- UML State Machine Diagram.
- UML Class Diagram.
- UML Component Diagram.
- UML Communication Diagram.

5.5. Enterprise Integration Patterns (EIPs)

Enterprise Integration Patterns (EIPs) are a set of design patterns for integrating different systems and applications within a large business that operates in various sectors. It can be used if we have an online store that uses different systems for handling orders, managing inventory, and providing customer service. EIPs help these systems communicate smoothly with each other by using some common patterns and Components. Common Patterns:

- Message Router
- Message Transformer
- Message Filter
- Message Aggregator
- Message Splitter
- Content-Based Router

Key Components:

- Message Channels

- Message Endpoints
- Message Flows

5.6. C4-Model

The C4 Model uses an "abstraction-first" approach to simplify the understanding and communication of complex software architectures. It starts with a high-level overview and progressively adds detail, ensuring each layer of abstraction builds upon the previous one.

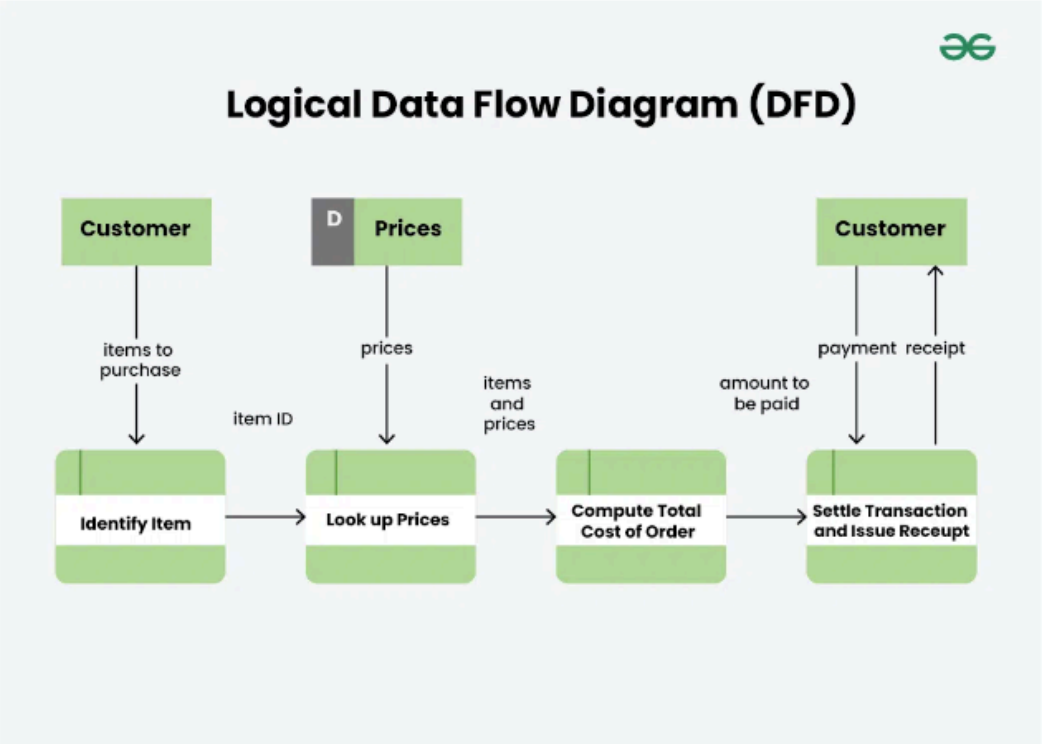
Levels of C4 Model Diagrams:

- **Level 1: A System Context diagram**
  - Represents the system as a whole and its interactions with external users or systems.
- **Level 2: A Container diagram**
  - Breaks the system down into its major components, such as applications, databases, or services.
- **Level 3: A Component diagram**
  - Provides an in-depth look at each container, showing internal components (such as modules or microservices) and their interactions.
- **Level 4: A code (UML class) diagram**
  - Details the code structure, including classes, interfaces, and methods.

5.7. Data Flow Diagrams (DFDs)

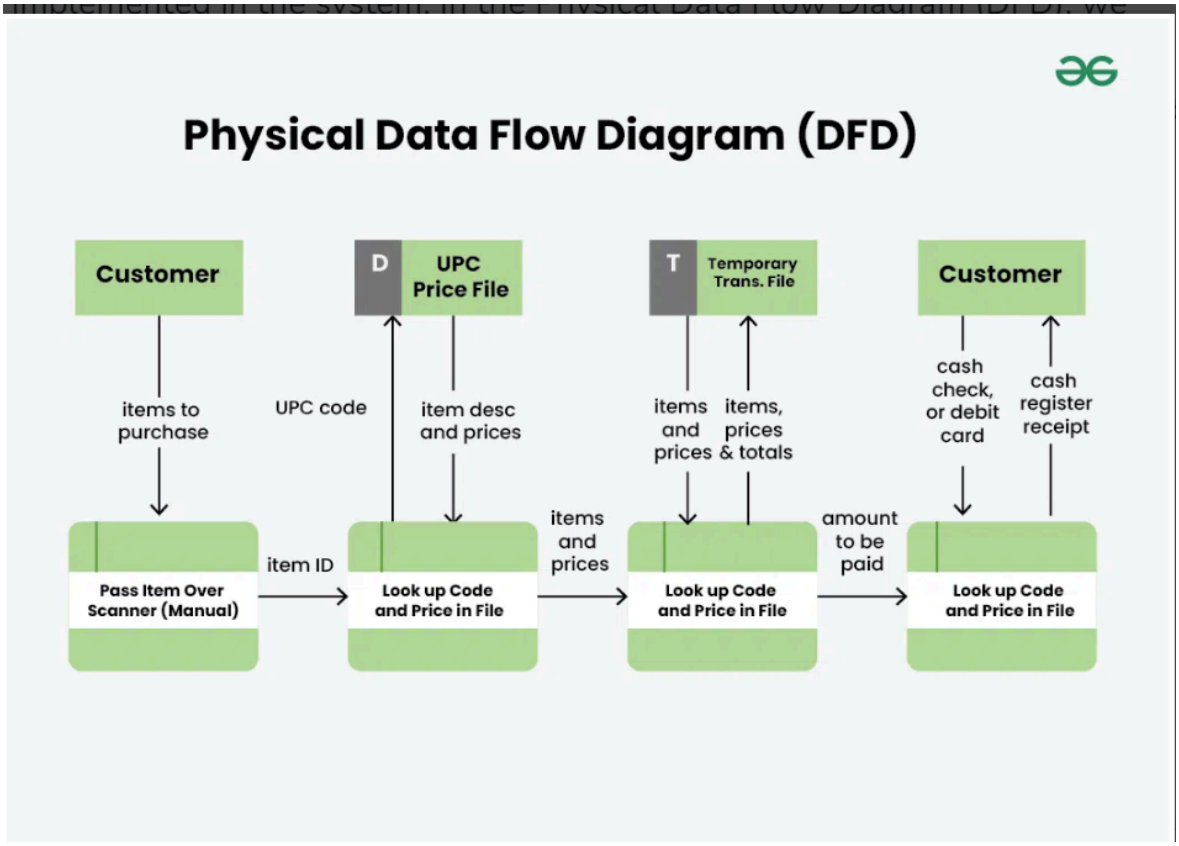
Data Flow Diagrams (DFDs) are used to visualize the flow of data within a system. It helps developers, customers, and users to work together during the analysis and specification of requirements, so they can understand how data is processed, stored, and transferred between different parts of a system. It focuses only on the movement of data between processes, data stores, and external users within a specific system. It can be used in the analysis and design of the data processing within a software application, such as how user input is handled, how data is stored, and how outputs are generated. There are two types of DFDs:

1. Logical Data Flow Diagram, which focuses on the system process and shows how data flows in the system



Logical Data Flow Diagram of Online Grocery Store

2. Physical Data Flow Diagram, which illustrates how data flows are implemented in the system.



6. Integration Patterns

Enterprise integration patterns provide blueprints for connecting and coordinating multiple applications and systems within an organization. These patterns facilitate smooth data flow and communication across various components.

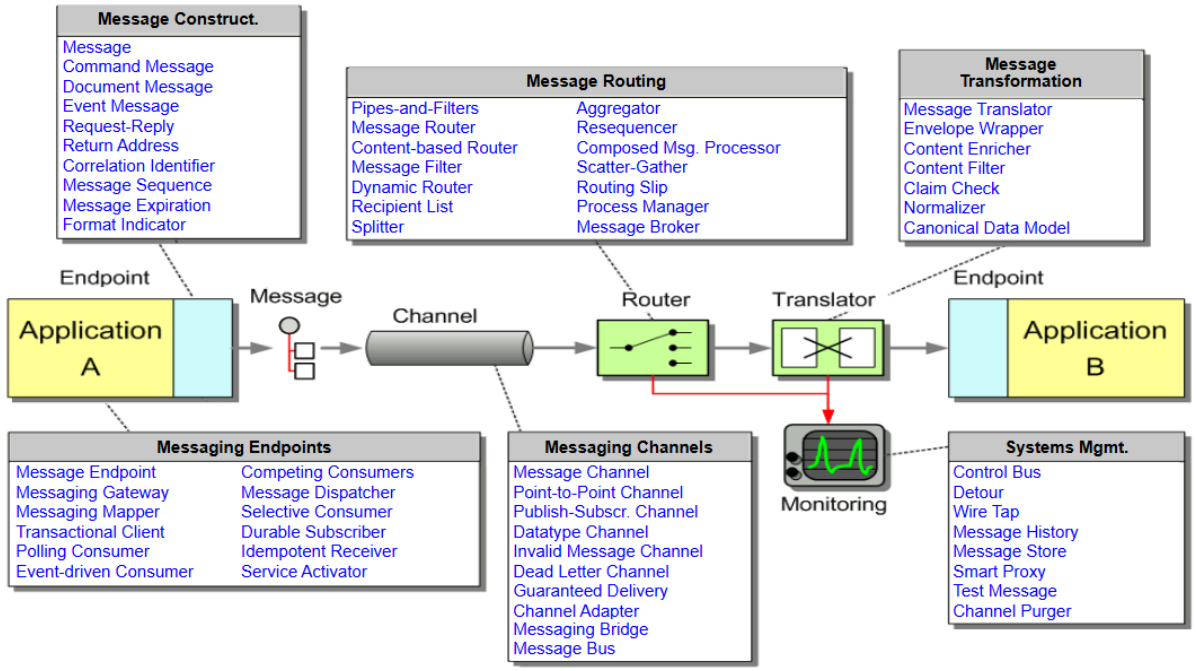


Figure 5: Different kind of patterns used for Enterprise Integration [1]

6.1. Pipes and Filters Patterns

The Pipes and Filters pattern is commonly used in systems where data must be processed through a series of steps. In this pattern:

- Filters represent each processing step. They perform specific operations on the data, such as transformation, validation, or filtering. Filters are typically designed to handle one specific task and can range from simple to complex operations.
- Pipes are the connections that pass data between filters. They can be simple in-memory queues, HTTP requests, or other data transfer mechanisms. Pipes ensure smooth and efficient data flow from one filter to the next.

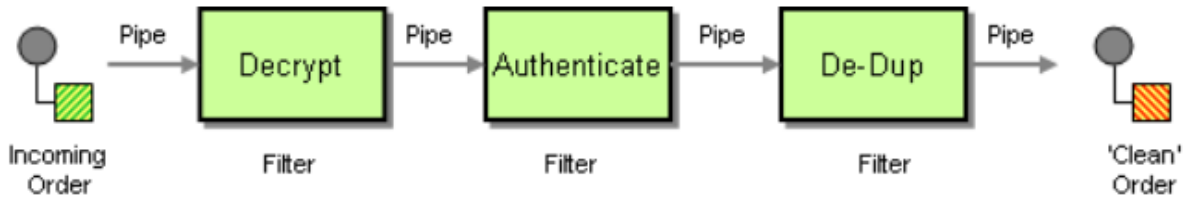


Figure 6: Pipeline using Pipes and Filters Pattern [2]

The illustration above shows a typical Pipes and Filters pattern. The pipeline takes an incoming order and processes it through several filters, each performing a specific function. Pipes transport the data from one filter to the next until the final output is produced.

Advantages of the Pipes and Filters Pattern:

- Easy to create new pipelines with existing filters.
- Simplifies updates or replacements of individual filters.
- Allows for reordering of filters as needed.

- Enables running filters on different hardware or in parallel.

In real-world scenarios, the Pipes and Filters pattern is particularly effective in microservices architectures, where each microservice can act as a filter in the pipeline. This pattern is ideal for scenarios where data needs to undergo multiple, distinct processing steps.

6.2. Other Integration Patterns

Other integration patterns include Enterprise Service Bus (ESB) and Remote Procedure Call (RPC), which provide different methods for application communication and integration.

6.2.1. Enterprise Service Bus (ESB)

Enterprise Service Bus (ESB) is another integration pattern that allows communication among multiple applications through services exposed by a middleware component. The 'bus' has applications and services plugged into it, which enables a rapid integration and reduces dependencies. The application integration becomes simplified. ESB is also popularly used in mircoservices architectures.

6.2.2. Remote Procedure Call (RPC)

The Remote Procedure Call (RPC) pattern is a request-response mechanism that allows a client to execute functions on a remote server as if they were local procedures. Communication between the client and server is synchronous; the client waits for the server's response before proceeding. Although RPC is relatively easy to implement, it can suffer from performance issues if the client becomes unresponsive or the server takes too long to respond.

All the patterns overall purpose for enterprise integration are to serve as blueprints for effective application integration, ensuring a smooth data flow between systems and applications.

7. Current Trends in Enterprise Integration

7.1. Cloud-Based Application Integration

Cloud-based application integration connects diverse software and systems, whether hosted in the cloud or on local servers, ensuring seamless business operations across various functions —from inventory management to customer service. By using tools such as iPaaS (Integration Platform as a Service) and APIs, businesses can avoid data errors, automate routine tasks, and make more informed decisions with real-time, synchronized data. This integration improves adaptability to change, allowing businesses to respond more quickly to market demands and opportunities.

7.2. Data-Centric Approach

A data-centric approach prioritizes data as a primary and critical asset in all aspects of business operations and decision-making processes. Ensuring data is accurate, available, and secure allows it to be effectively used across the entire organization. This is essential for integrating various disparate systems into a single unified system and provides a broader, more comprehensive view of the business rather than limited, localized snapshots.

Benefits of Being Data-Centric in Enterprise Integration:

- Unified Data Management: Data from different departments, such as sales, finance, and customer service, is made available in a format that everyone can easily understand and use. This unified approach aids in making better decisions since everyone accesses the same comprehensive data set.

- Improves Data Quality: Integrating all systems reduces data errors. High-quality data ensures that reports, analytics, and decisions based on this data are more reliable.
- Easier Access to Data: With centralized or well-connected data through integration, information becomes more accessible to everyone, quickly and securely. This facilitates faster operations and enhances collaboration across different departments.
- Better Scalability: A data-centric approach allows businesses to grow more efficiently. New systems and data can be seamlessly added without disrupting existing operations, maintaining consistent and well-integrated data.
- Supports Compliance and Security: Managing data centrally and focusing on enterprise-wide data simplifies implementing security measures and complying with regulatory requirements.

7.3. Runtime Analytics

Runtime analytics involves analyzing real-time data collected during system operations. This method enables companies to make quick and informed decisions as the data is generated, avoiding the delays typical of traditional data storage and analysis. Real-time analysis allows for more immediate and dynamic decision-making.

Real-time data is often visualized on dashboards—such as those in Microsoft Power BI—which provide administrators with actionable insights to make timely and informed decisions.

Bibliography

[1] "Enterprise Integration Patterns." Accessed: Sep. 05, 2024. [Online]. Available: https://www.enterpriseintegrationpatterns.com/patterns/messaging/

[2] "Pipes and Filters." Accessed: Sep. 05, 2024. [Online]. Available: https://www.enterpriseintegrationpatterns.com/patterns/messaging/PipesAndFilters.html