Messaging System

Outline

- Application distribution vs. application integration
- Challenges to integration
- Integration styles
- Messaging system
- Advantages of asynchronous messaging
- Messaging patterns overview

Distributed Applications vs. Integration

- An enterprise application often incorporates an n-tier architecture.
- Why is an n-tier architecture considered application distribution and not application integration?
- Communicating parts are tightly coupled.
- Communication between tiers tends to be synchronous.

Challenges to integration

- All integration solutions have to deal with a few fundamental challenges:
- Networks are unreliable.
- Networks are slow.
- Any two applications are different
- Change is inevitable – avoid **avalanche effect**

Integration solutions can easily get caught in an **avalanche effect** of changes – if one system changes, all other systems may be affected. An integration solution needs to minimize the dependencies from one system to another by using loose coupling between applications.

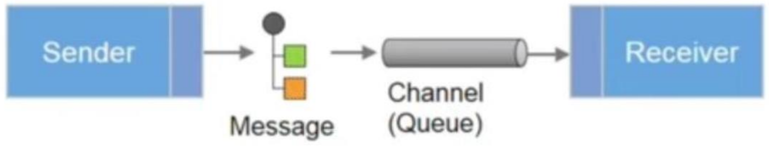
What is messaging?

- Messaging is a technology that enables high-speed, asynchronous, program-to-program communication with reliable delivery.
- Programs communicate by sending packets of data called messages to each other.
- Channels, also known as queues, are logical pathways that connect the programs and convey messages.
- A sender or producer is a program that sends a message by writing the message to a channel. A receiver or consumer is a program that receives a message by reading (and deleting) it from a channel.

Two important messaging concepts

- **Send and forget** — The sending application sends the message to the message channel. Once that send is complete, the sender can be confident that the receiver will eventually receive the message and does not have to wait until that happens.
- **Store and forward** — The messaging system stores the message on the sender's computer. Then, the messaging system delivers the message by forwarding it to the receiver's computer, and then stores the message once again on the receiver's computer. This store-and-forward process may be repeated many times, as the message is moved from one computer to another, until it reaches the receiver's computer.

Asynchronous messaging model



Systems send messages	Simple Interaction
Channels have logical, location-independent addresses	Location Decoupling
Placing a message into the Channel is quick ("fire-and-forget"). The Channel queues messages until the receiving application is ready to consume.	Temporal Decoupling

Asynchronous messaging benefits

Temporal decoupling

- Sender does not have to wait for receiver to process message

Limit Failure Propagation

- Sender not affected by intermittent failure
- Can be reliable over unreliable transports

Throttling

- Receiver can consume messages at its own pace
- Processing units can be tuned independently

Asynchronous messaging benefits

Insertion of intermediaries (Pipes-and-Filters)

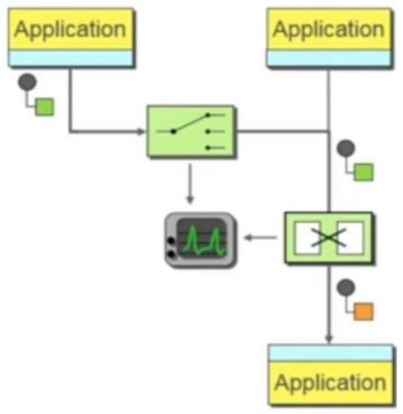
- Composability: Transformation, routing etc.

Throughput over latency

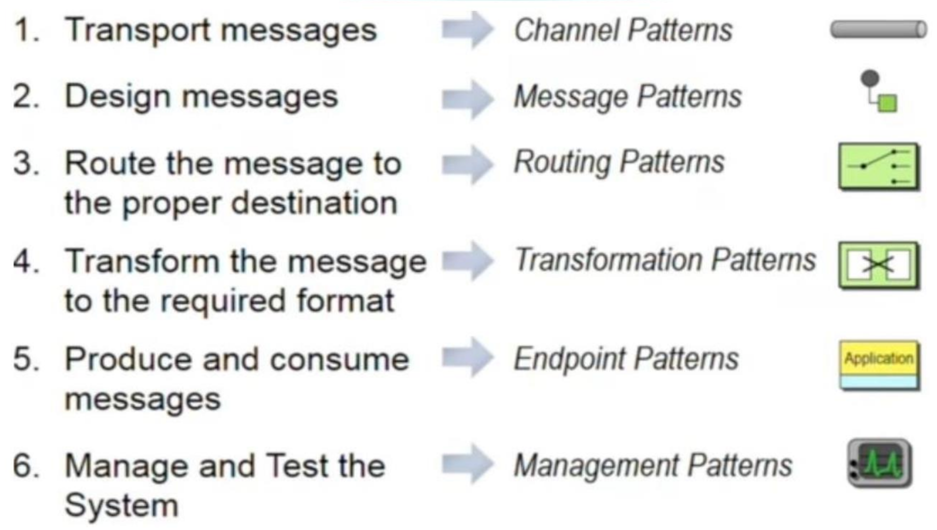
- "Wider bridges not faster cars"

Messaging systems

1. Transport messages
2. Design messages
3. Route the message to the proper destination
4. Transform the message to the required format
5. Produce and consume messages
6. Manage and Test the System



Messaging systems patterns



1. Transport messages	→	*Channel Patterns*
2. Design messages	→	*Message Patterns*
3. Route the message to the proper destination	→	*Routing Patterns*
4. Transform the message to the required format	→	*Transformation Patterns*
5. Produce and consume messages	→	*Endpoint Patterns*
6. Manage and Test the System	→	*Management Patterns*

Application

Messaging systems patterns

The messaging patterns are basically patterns for each of those categories:

1- **Channel patterns** describe how messages are transported across a channel.

2- **Message construction patterns** describe the intent, form and content of the messages that travel across the messaging system.

3- **Routing patterns** discuss how messages are routed from a sender to the correct receiver. Routing patterns consume a message from one channel and republish it, usually without modification, to another channel based on a set of conditions.

4- **Transformation patterns** change the content of a message, for example to accommodate different data formats used by the sending and the receiving system. Data may have to be added, taken away or existing data may have to be rearranged.

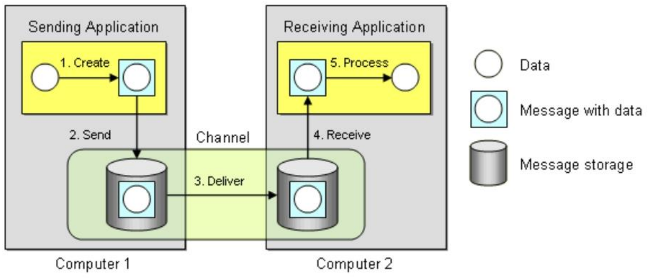
5- **Endpoint patterns** describe how messaging system clients produce or consume messages.

6- **Management patterns** describe the tools to keep a complex message-based system running, including dealing with error conditions and performance bottlenecks.

Messaging System

Messaging system

- A messaging system manages messaging the way a database system manages data persistence.
- The main task of a messaging system is to move messages from the sender's computer to the receiver's computer in a reliable fashion.
- A message is transmitted in five steps: create, send, deliver, receive, and process.



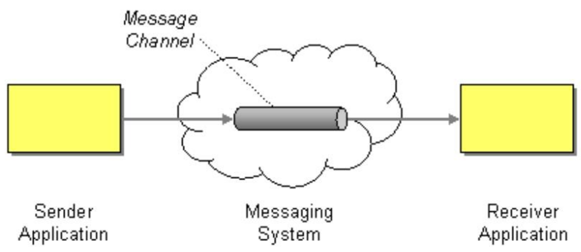
Messaging system

- A message is transmitted in five steps:

1. Create — The sender creates the message and populates it with data.
2. Send — The sender adds the message to a channel.
3. Deliver — The messaging system moves the message from the sender's computer to the receiver's computer, making it available to the receiver.
4. Receive — The receiver reads the message from the channel.
5. Process — The receiver extracts the data from the message.

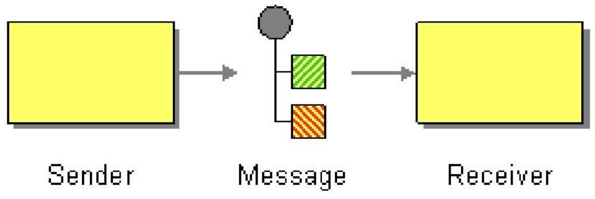
Message Channel

- How does one application communicate with another using messaging?
- Connect the applications using a Message Channel, where one application writes information to the channel and the other one reads that information from the channel.



Message

- How can two applications connected by a message channel exchange a piece of information?
- Package the information into a Message, a data record that the messaging system can transmit through a message channel.



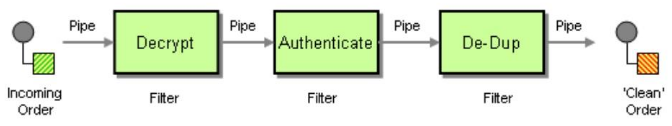
- A message consists of two basic parts:

  1- **Header** – Information used by the messaging system that describes the data being transmitted, its origin, its destination, and so on.

  2- **Body** – The data being transmitted; generally ignored by the messaging system and simply transmitted as-is.

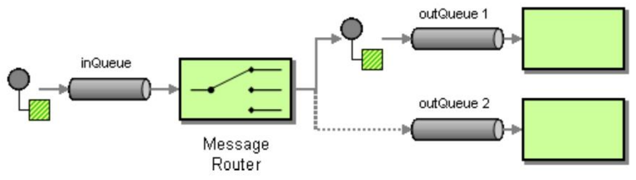
Pipes and Filters

- How can we perform complex processing on a message while maintaining independence and flexibility?
- Use the Pipes and Filters architectural style to divide a larger processing task into a sequence of smaller, independent processing steps (Filters) that are connected by channels (Pipes).



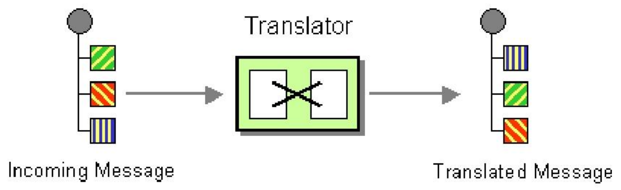
Message Router

- How can you decouple individual processing steps so that messages can be passed to different filters depending on a set of conditions?
- Insert a special filter, a Message Router, which consumes a Message from one Message Channel and republishes it to a different Message Channel depending on a set of conditions.



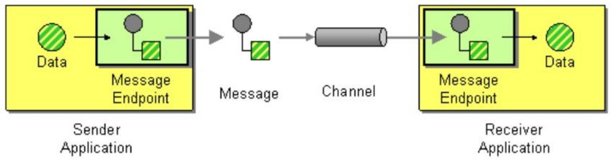
Message Translator

- How can systems using different data formats communicate with each other using messaging?
- Use a special filter, a Message Translator, between other filters or applications to translate one data format into another.



Message Endpoint

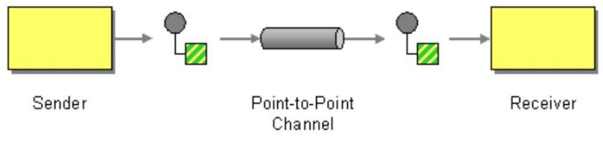
- How does an application connect to a messaging channel to send and receive messages?
- Connect an application to a messaging channel using a Message Endpoint, so that the application can then use it to send or receive messages.



Messaging Channels

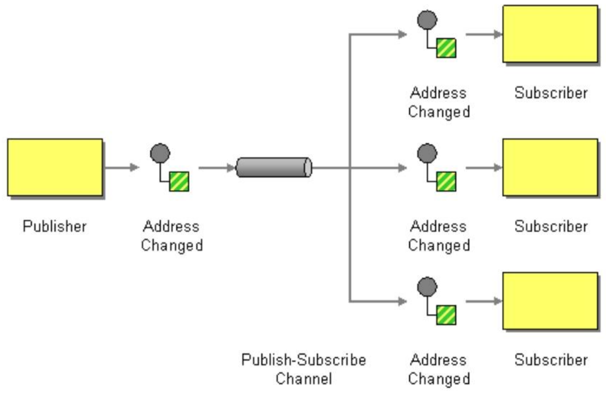
Point-to-Point Channel

- How can the caller be sure that exactly one receiver will receive the document or perform the call?
- Send the message on a Point-to-Point Channel, which ensures that only one receiver will receive a particular message.



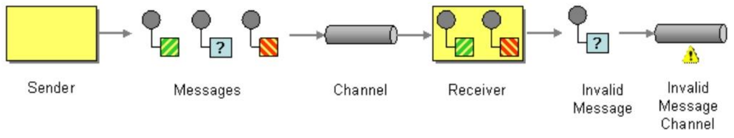
Publish-Subscribe Channel

- How can the sender broadcast an event to all interested receivers?
- Send the event on a Publish-Subscribe Channel, which delivers a copy of a particular event to each receiver.



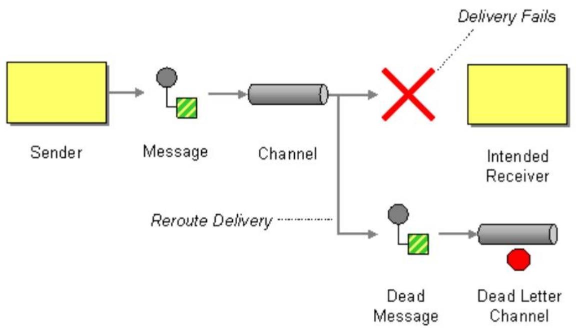
Invalid Message Channel

- How can a messaging receiver gracefully handle receiving a message that makes no sense?
- The receiver should move the improper message to an Invalid Message Channel, a special channel for messages that could not be processed by their receivers.



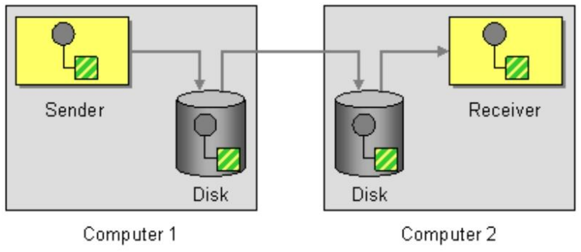
Dead Letter Channel

- What will the messaging system do with a message it cannot deliver?
- When a messaging system determines that it cannot or should not deliver a message, it may elect to move the message to a Dead Letter Channel.



Guaranteed Delivery

- How can the sender make sure that a message will be delivered, even if the messaging system fails?
- Use Guaranteed Delivery to make messages persistent so that they are not lost even if the messaging system crashes.



Questions

- Which of the following best describes a message in the context of messaging systems?
  - a) A set of commands to be executed by a system
  - b) A structured piece of data exchanged between systems
  - c) A log entry in a database
  - d) A visual representation of a system's workflow

Questions

- In messaging systems, what is the main advantage of using asynchronous messaging?
  - a) It requires less programming effort
  - b) It reduces the cost of network hardware
  - c) It decouples systems, allowing them to operate independently
  - d) It ensures that messages are never lost

Questions

- What is a "message channel" in the context of messaging systems?
  - a) A medium for transmitting messages between two systems
  - b) A storage location for archived messages
  - c) A graphical user interface for composing messages
  - d) A software tool for monitoring system performance