AARHUS UNIVERSITET

Microservices and DevOps

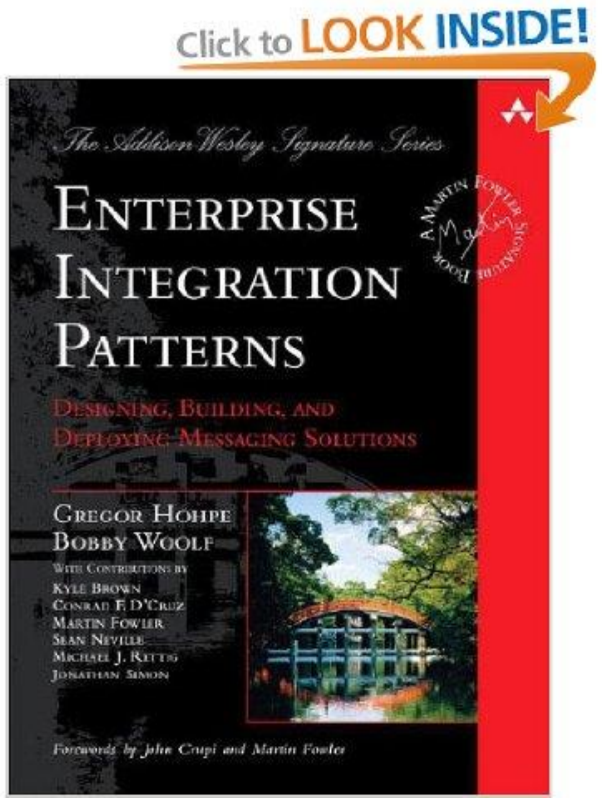
Scalable Microservices

Messaging

Henrik Bærbak Christensen

Literature

- Hohpe & Woolf, 2004
  - We will just scratch the surface

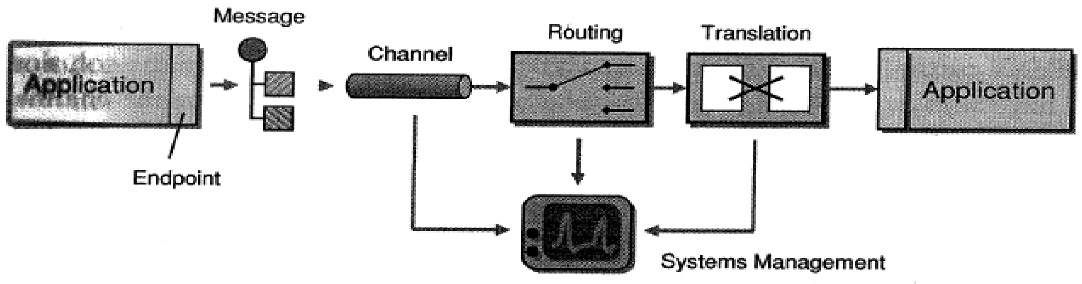

Click to LOOK INSIDE!
The Addison Wesley Signature Series
ENTERPRISE INTEGRATION PATTERNS
DESIGNING, BUILDING, AND DEPLOYING MESSAGING SOLUTIONS
GREGOR HOHPE
BOBBY WOOLF
A MARTIN FOWLER SIGNATURE BOOK
Forewords by John Crupi and Martin Fowler

Example

Messaging in One Minute

Reference Architecture

- Any Messaging system will have this architecture

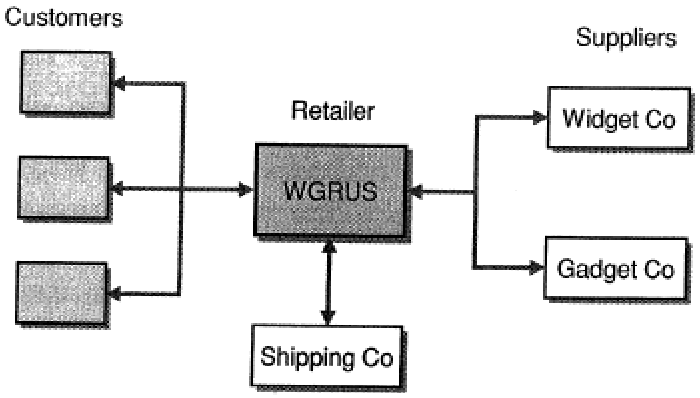


Basic Elements of Message-Based Integration

- Metaphor of Messaging: Mail and mailboxes
  - Message = a letter
  - Channel = mailbox
  - Routing = address, stating who to receive
- Is Asynchronous !

Example: WGRUS

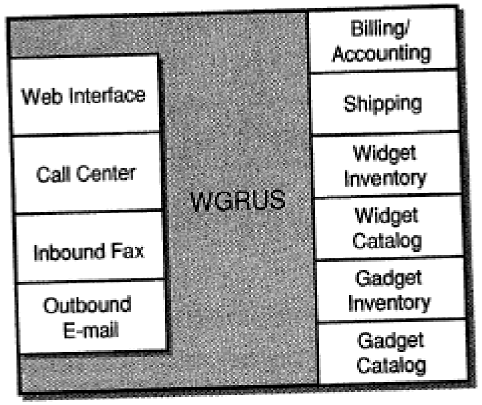
- Retailer selling widgets and gadgets
  - Orders by web, by phone, by fax
  - Processing
    - Check inventory, shipping, invoicing
  - Status check by customer
  - Admin
    - Update prices
    - Update user details



WGRUS Ecosystem

Legacy Systems

- WGRUS is a merged company

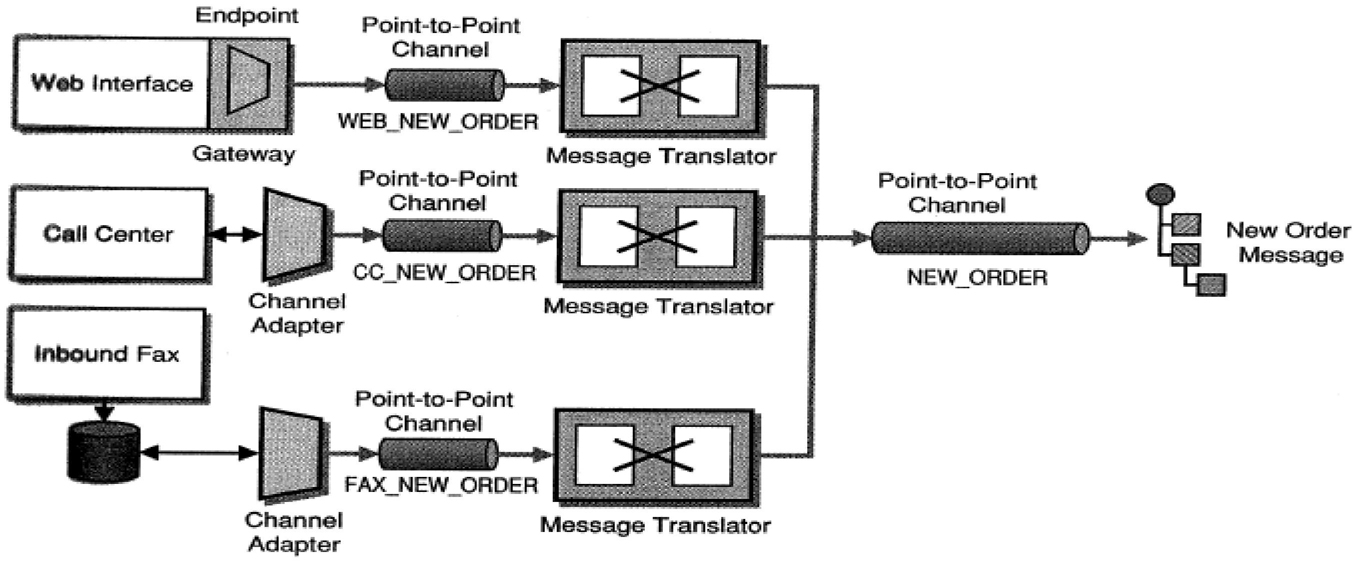


WGRUS IT Infrastructure

- How do we bind all these systems together?

Client side

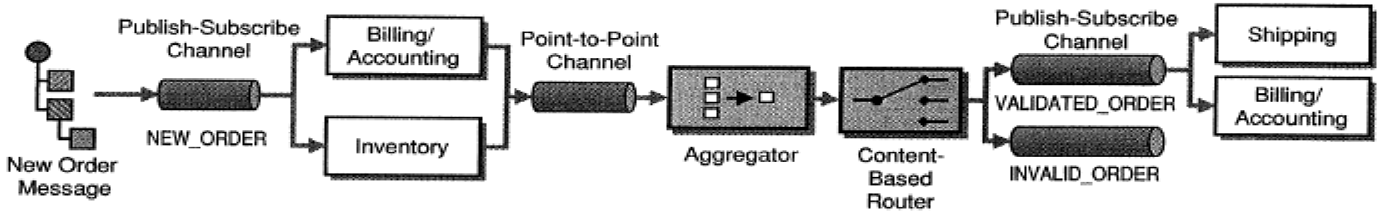
- A MQ based solution
  - How to make ‘a order’ a uniform message from three different systems and processes



Taking Orders from Three Different Channels

Server side

- How do we handle that an order must
  - Update and verify inventory status
  - Be packed and shipped
  - Invoiced
  - Or perhaps rejected?



Order Processing Implementation Using Asynchronous Messaging

Discussion

- *Enterprise Service Bus*
  - *The solution to all integration issue?*

- AntiPattern: Swiss Army Knife
  - Is it super smart? Or one tool that does all jobs equally poor?

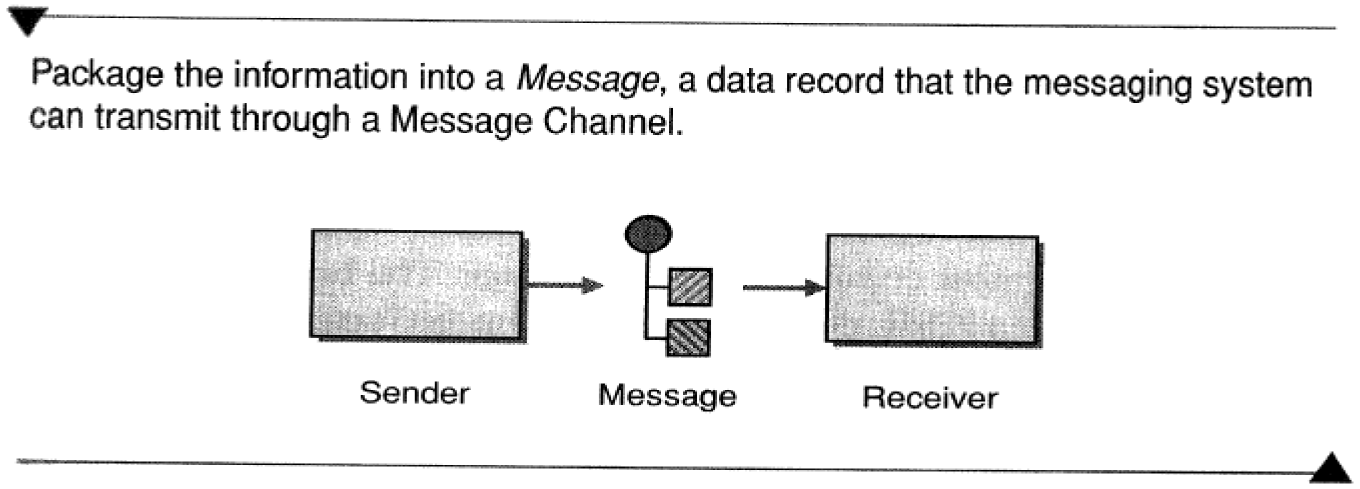
- Jim Webber (REST guru)
  - ESB becomes one *big ball of mud*

AARHUS UNIVERSITET

Patterns

Just the simple ones...

Message

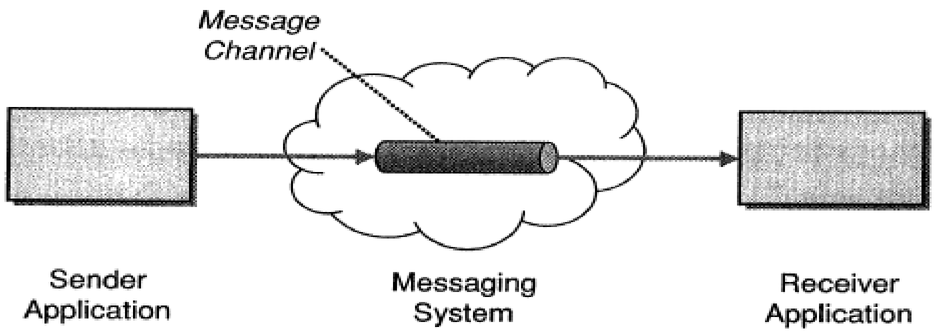


Package the information into a *Message*, a data record that the messaging system can transmit through a Message Channel.

Sender
Message
Receiver

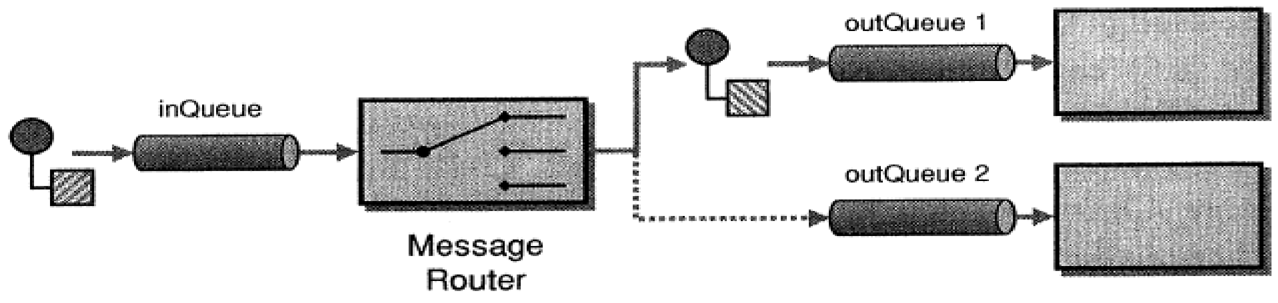
Message Channel

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.



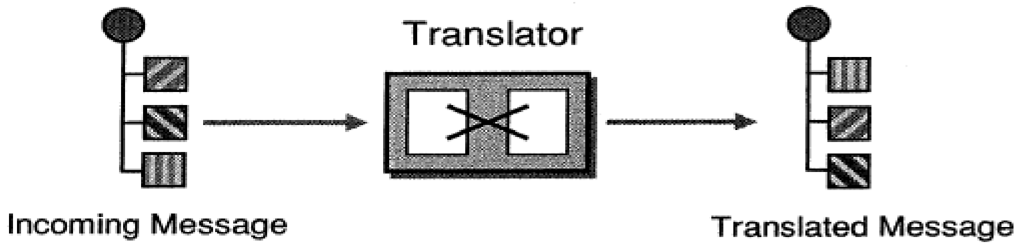
Message Router

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.

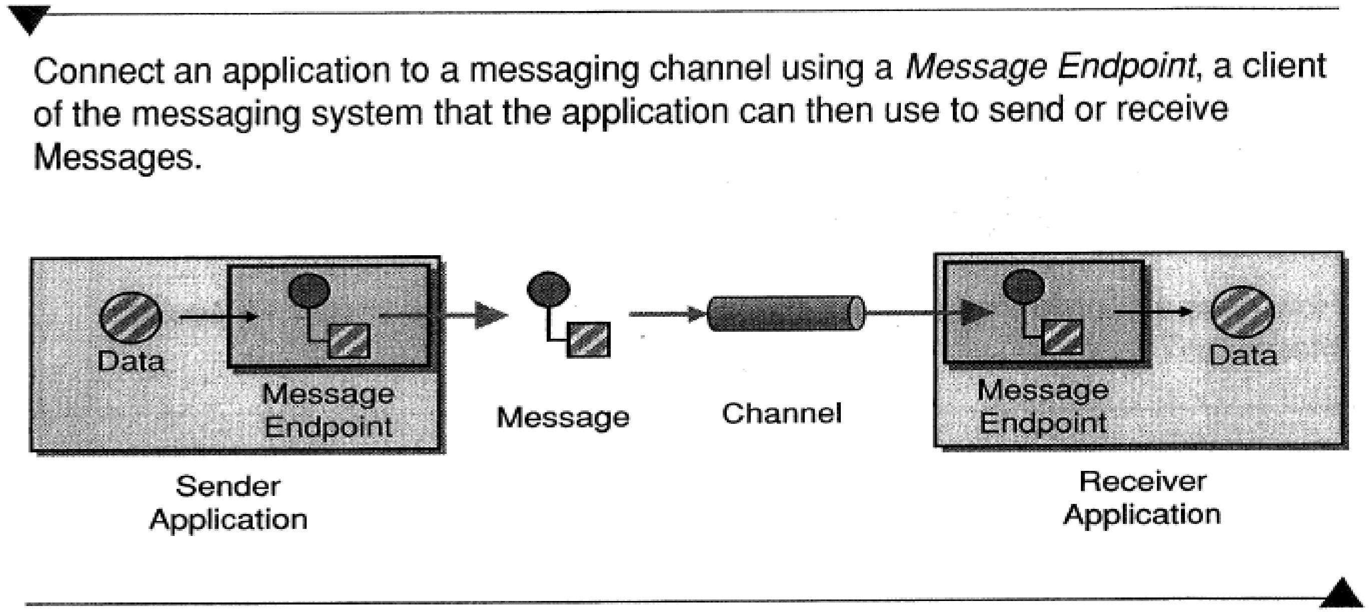


Message Translator

Use a special filter, a *Message Translator*, between other filters or applications to translate one data format into another.

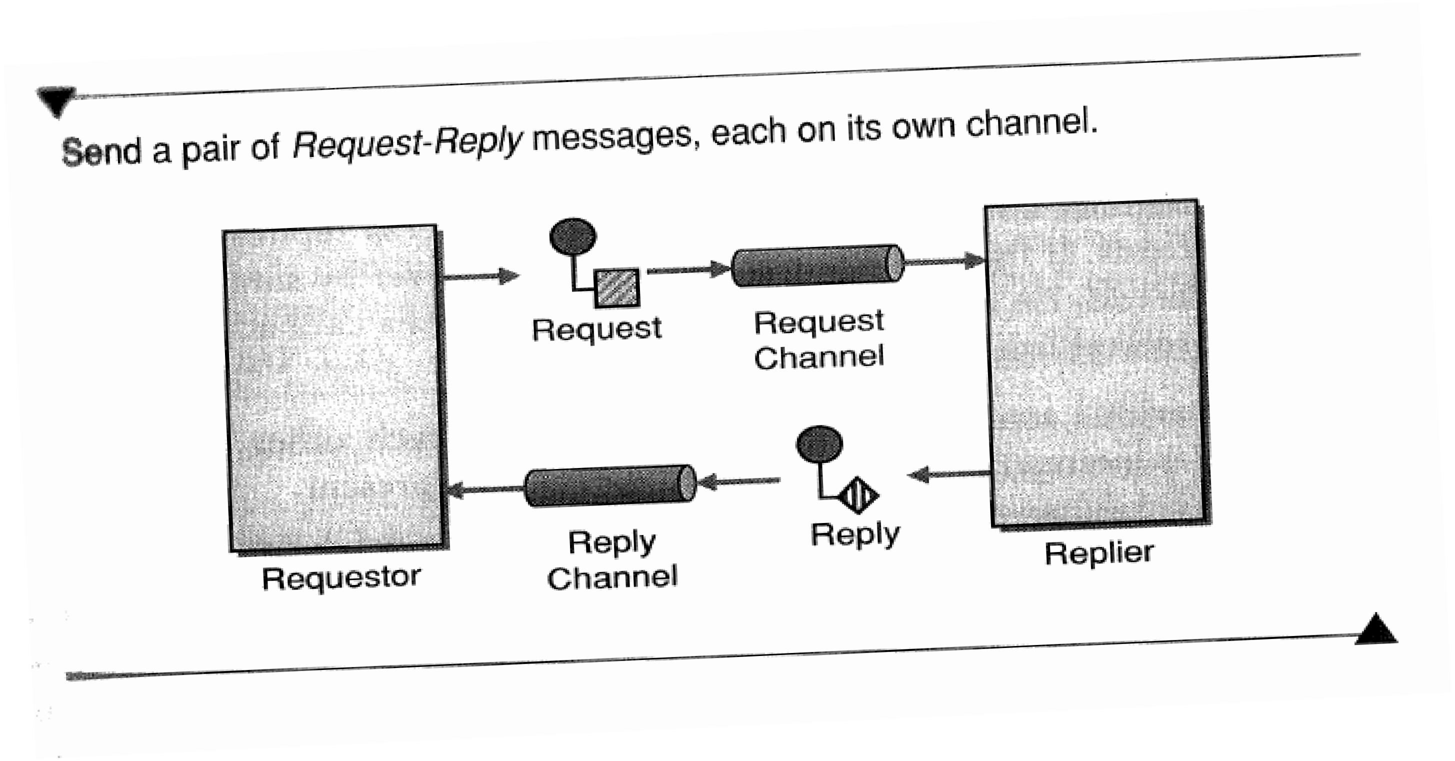


Message Endpoint

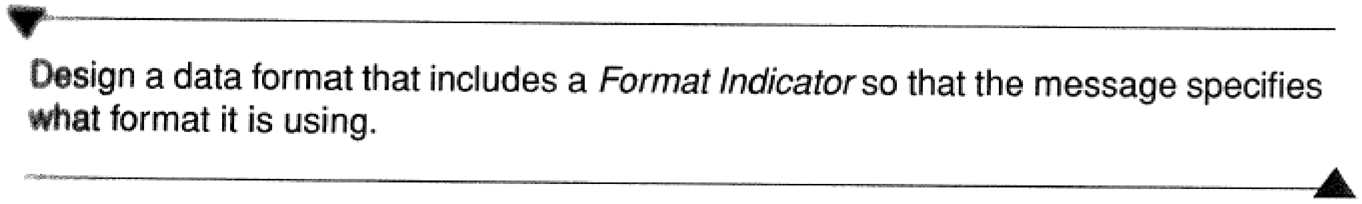


Connect an application to a messaging channel using a *Message Endpoint*, a client of the messaging system that the application can then use to send or receive Messages.

Request-Reply

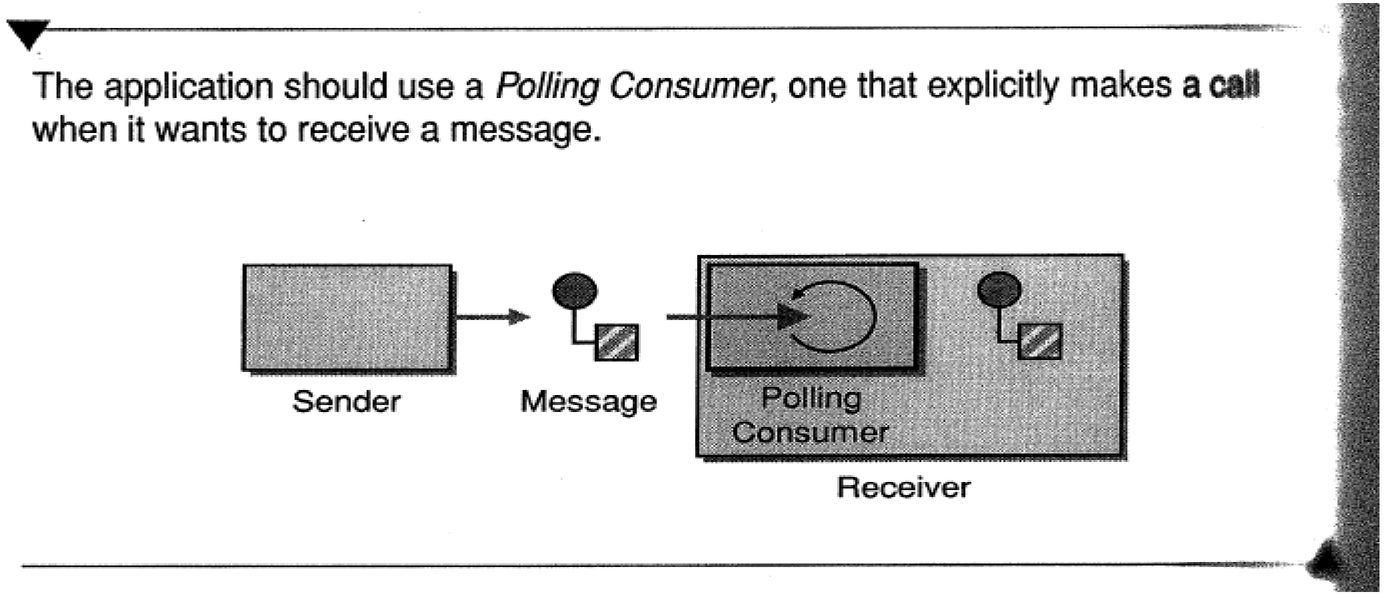

Send a pair of *Request-Reply* messages, each on its own channel.

Format Indicator



Design a data format that includes a *Format Indicator* so that the message specifies what format it is using.

Polling Consumer

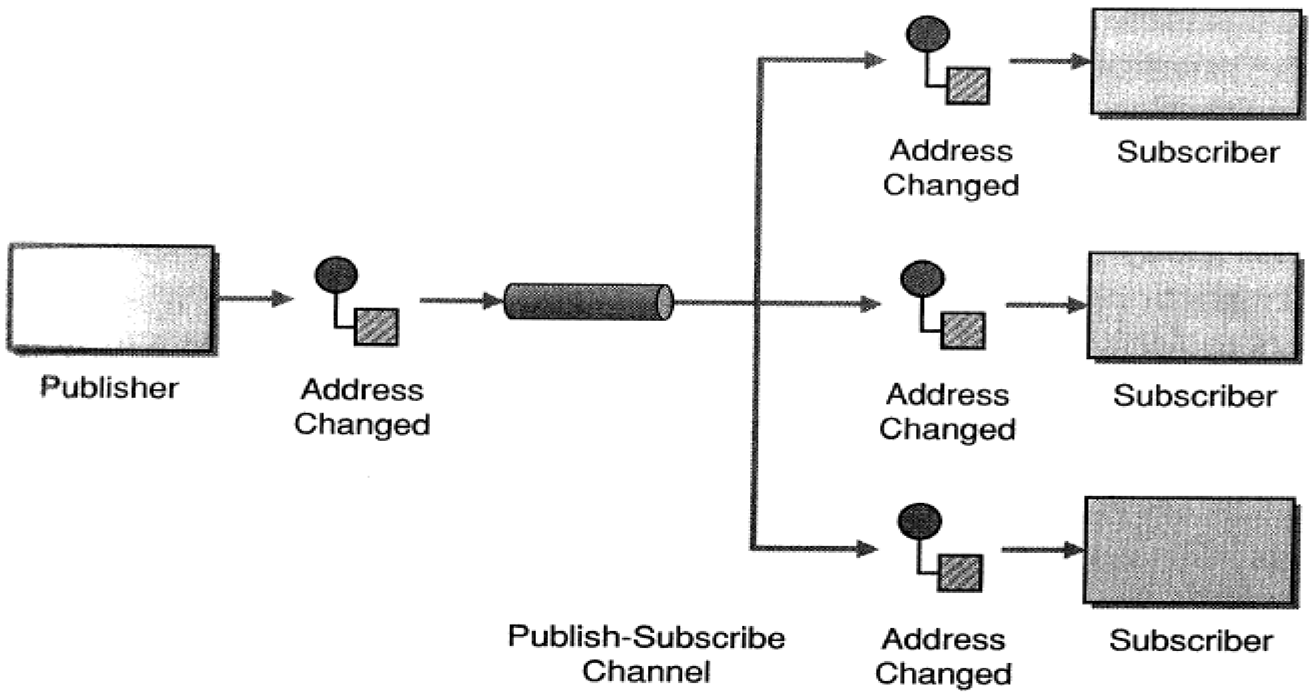


The application should use a *Polling Consumer*, one that explicitly makes a call when it wants to receive a message.

Sender
Message
Polling Consumer
Receiver

Publish-Subscribe Channel

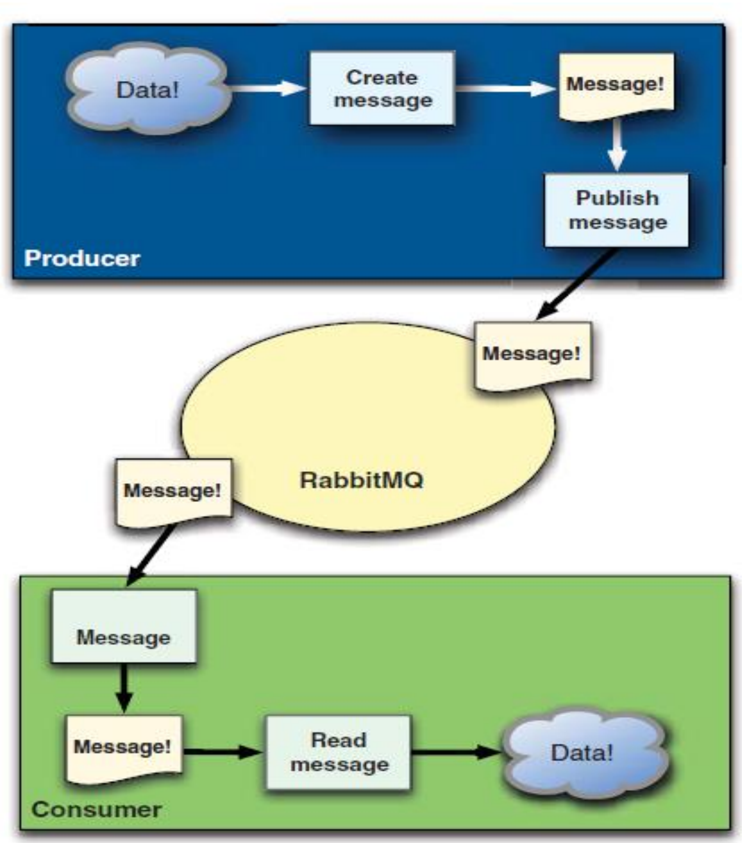
Send the event on a *Publish-Subscribe Channel*, which delivers a copy of a particular event to each receiver.



RabbitMQ

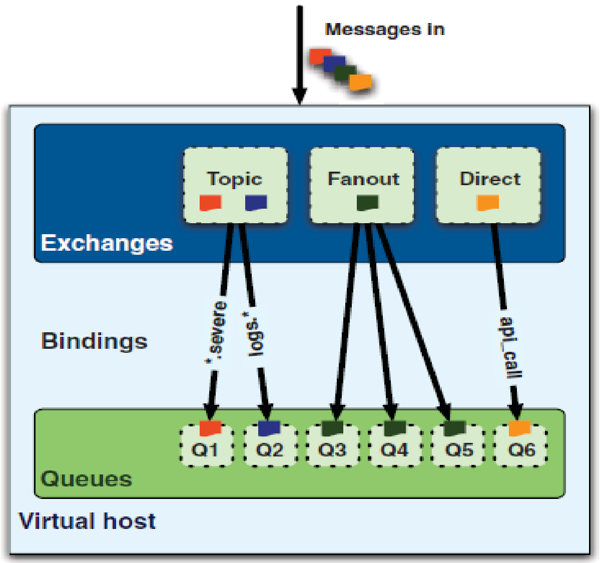
AARHUS UNIVERSITET

The Basic Architecture



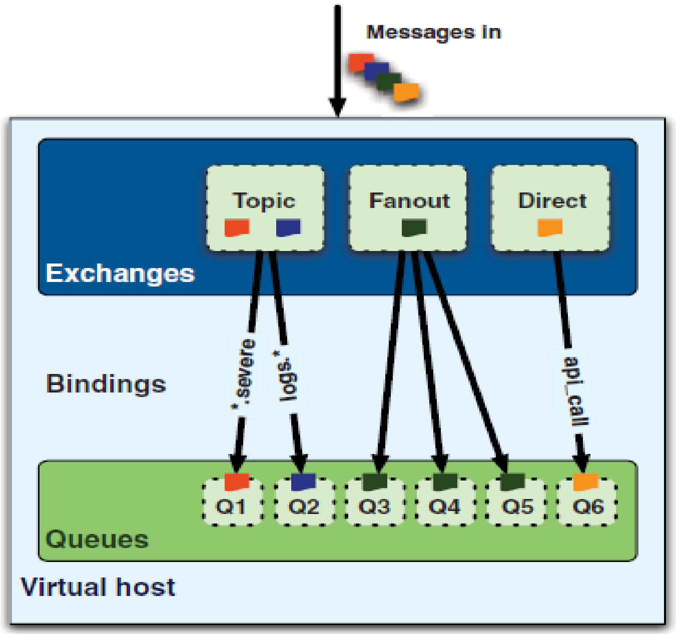
Exchanges and Queues

- Clients push messages to *exchanges*
- Serves pulls messages from *queues*
- *Bindings* govern how exchanges moves messages to queues



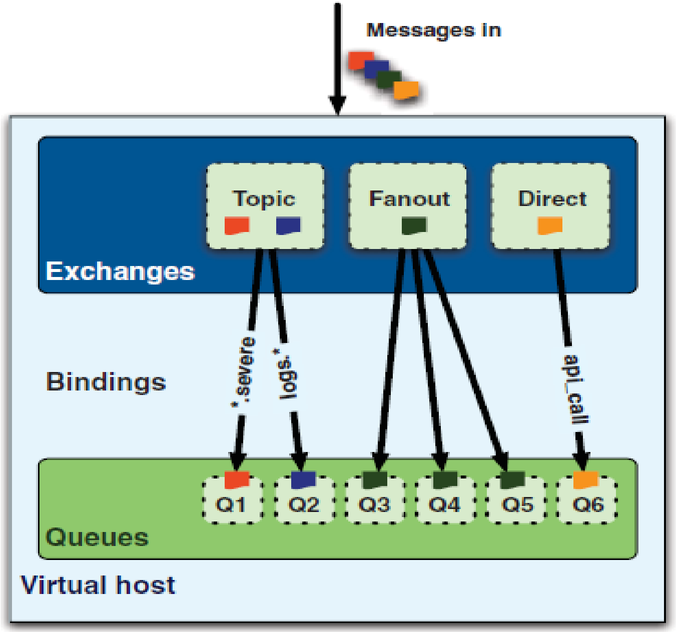
Direct

- *Direct* is ‘point-to-point’ communication ala Broker/REST
  - Essentially it *seems* there is no exchange, because our message ends up on the queue right away



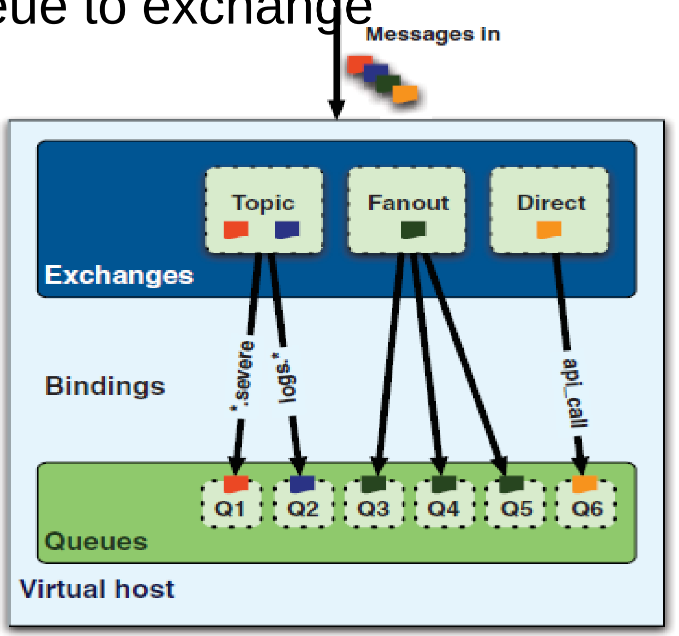
Fanout

- Publish-subscribe uses *fanout*
  - Clients pushes to a *named* exchange (ex: ”logs”)
  - Queues are *bound* to a named exhange (ex: Q3-logs)
  - Server pull from named queue



Topics

- *Message Router* is configured using *topics*
  - Clients push msg with a specific topic to exchange
    - Topic = ”grundfos.reading.store”
  - *Routing key* use macthing to bind queue to exchange
    - Store_queue: ”*.*.store”
  - Any msg with topic that match routing key is put into that queue
  - Server pull from named queue



Lots of Options

- RabbitMQ uses **round-robin load balancing**
  - 2 servers connect to queue ‘Q’
    - Msg1 to server1, msg2 to server2, msg3 to server1, …
- Acknowledgement system
  - Default off, but server may *acknowledge* message is processed
    - No new message delivered until message has been ack.
- Topic based messaging = many options
  - Cluster serves queues bound to ‘*.*.AALBORG’
    - Exercise: Which Nygard pattern does that implement?
  - Bound to ‘*.server7.*’
    - Exercise: What session management does that implement?

AARHUS UNIVERSITET

Discussion

Reliability and Availability?

- So – how does messaging help make more reliable and available systems?

- ?

Availability

- Different time – Different process

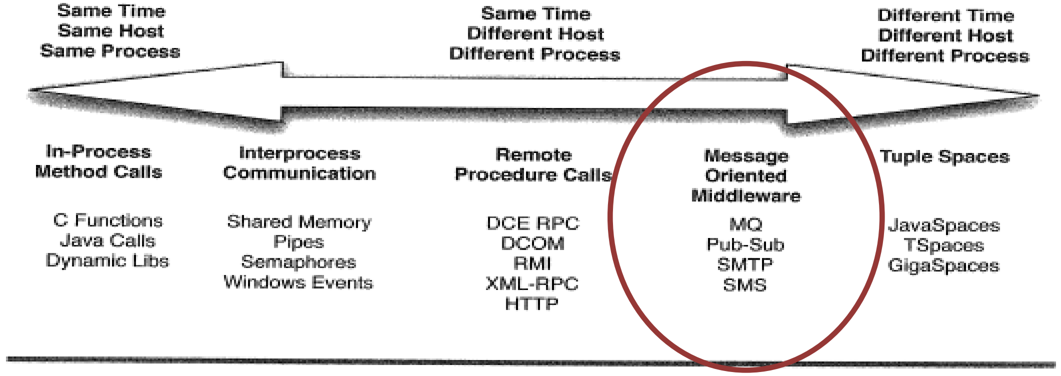


Figure 5.4: Coupling Spectrum of Middleware

  - => Loose coupling at *integration points*
  - Awareness that we are dealing with remote nodes
    - Cmp Java RMI, Corba, .NET remoting
      - Tries to hide that a call is remote

Availability

- Messages are queue in case no consumer
  - If the clients do not need an immediate answer...
    - Read: data collection systems
  - ... Then back-end systems can be maintained while the MQ system just queue up messages for later processing
- Message brokers can be clustered
  - Replication of queues
- Queues can be persisted
  - Messages survive crashed nodes/brokers

AARHUS UNIVERSITET

Availability

- Can provide ‘elasticity’ during *impulses* to counter *unbalanced capacities*
  - During a sudden peak of messages, the MQ serves as a queue, until the consumers can catch up
  - **Key point: The servers set the pace, not the clients!**
- Exercise:
  - How will the clients experience such a situation?

Liabilities

- Instead of
  - Client + server as in the REST / Web case
- … we have
  - Client + message broker + server
- Message broker becomes single-point of failure
  - Counter measure: Clustering
    - But clustering works less well for RabbitMQ (!)
- Message broker becomes bottle-neck
  - Kafka… *Rumors has it that it is extremely fast…*

AARHUS UNIVERSITET

Summary

- Messaging
  - A mail and letterbox metaphor for message exchange
  - Allows flexibility in delivery and content change
  - Decouples producers and consumers over time
  - Asynchronous
- RabbitMQ
  - Exchanges and Queues are bound at run-time
  - Round robin load balancing of queue fetch
- Availability and Stability
  - Handles *impulses* well; not strain…