AARHUS UNIVERSITET

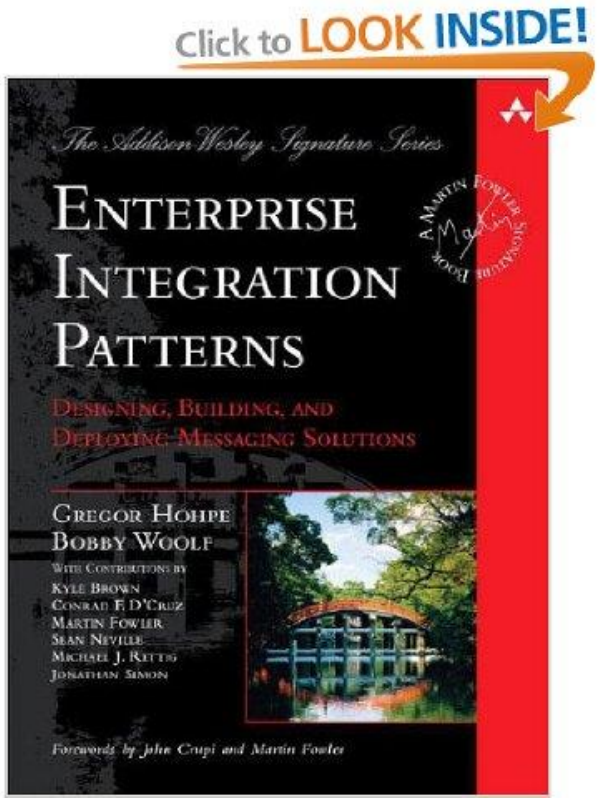
Software Architecture in Practice

Connector: Messaging

Henrik Bærbak Christensen

Literature

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

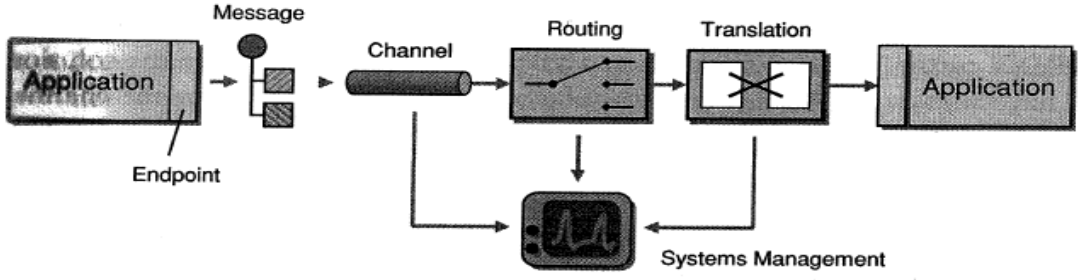


Example

Messaging in One Minute

Reference Architecture

- Any Messaging system will have this architecture

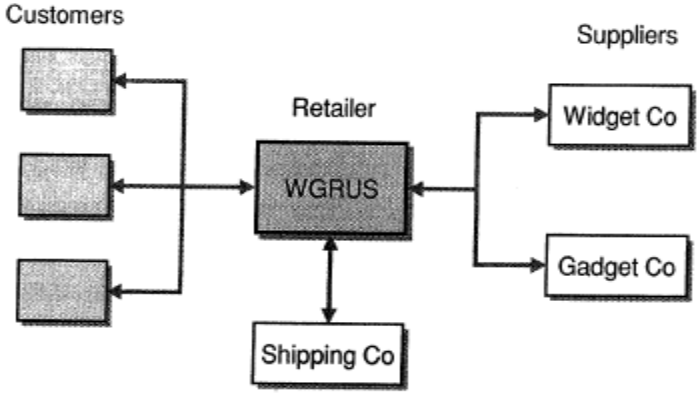


Basic Elements of Message-Based Integration

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

Example: WGRUS

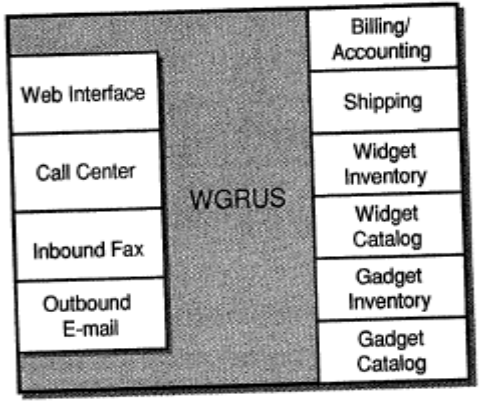
- Retailer selling ‘widgets and gadgets’
  - Orders by web, by phone, by fax
  - Processing
    - Check inventory, shipping, invoicing
  - Status check by customer
  - Admin
    - Update prices
    - Update user details



WGRUS Ecosystem

Legacy Systems

- WGRUS is a merged company
  - Legacy applications, own formats, own processes for order intake

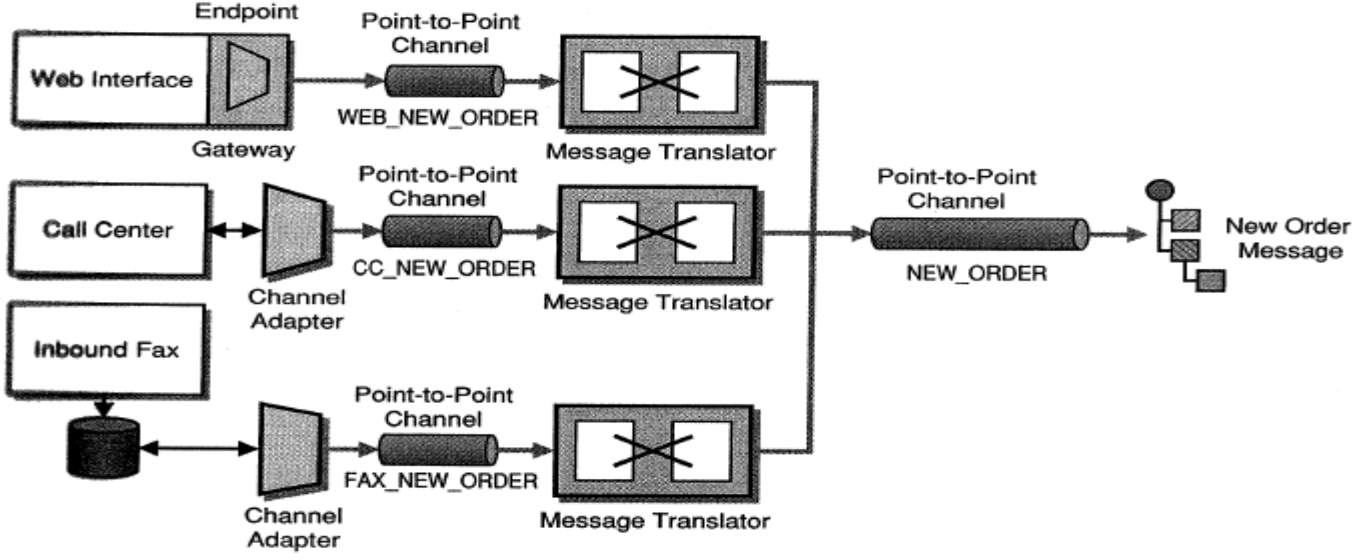


WGRUS IT Infrastructure

- How do we bind all these systems together?

Client side

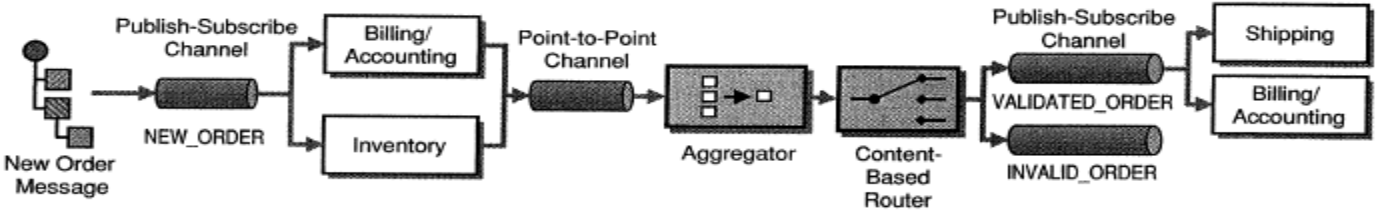
- A MQ based solution
  - How to make ‘a order’ a uniform message from three different systems and processes



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

AARHUS UNIVERSITET

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*
  - Because – where is the logic?
    - Not in Components but in the Connector (=ESB)
    - *Smart services, dump pipes* is way forward

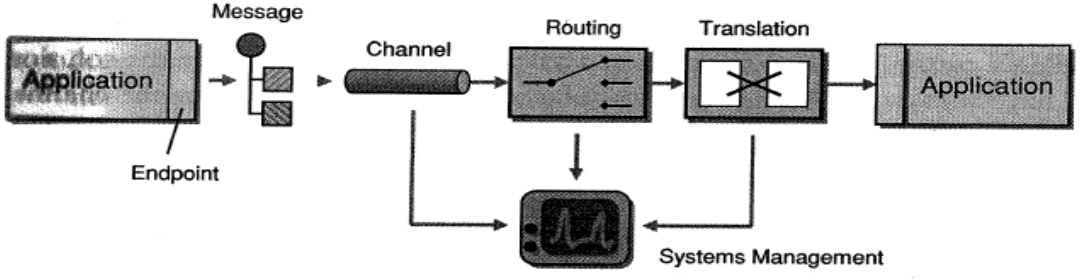
AARHUS UNIVERSITET

Discussion

- But… The merits is in the
  - Loose coupling
    - Producer/consumer just agree on a ‘channel’, no direct URL or addresses involved (except to the MQ ☺)
  - Temporal decoupling
    - MQ acts as a buffer, so a consumer can ‘be away for security updates for 15 minutes’ and then catch up on what happened
    - No direct connection meaning A crashes when B is ‘away’ for some time
  - Workload decoupling
    - If producer makes a *spike* (more messages than consumer can cope with), it is just buffered in the MQ, the consumer is not overwhelmed, but processes in its own pace

Discussion

- And these merits are just using the core MQ patterns
  - Channels (dump connectors/pipes)
  - Routing (specify which messages you pub/subs to)



Basic Elements of Message-Based Integration

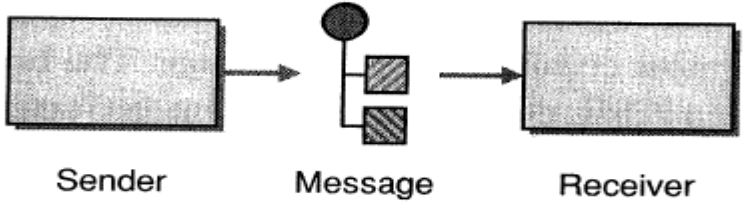
AARHUS UNIVERSITET

Patterns

Just the simple ones...

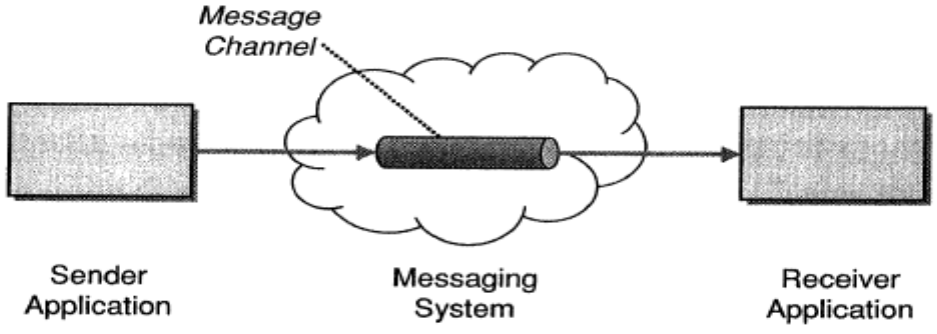
Message

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



Message Channel

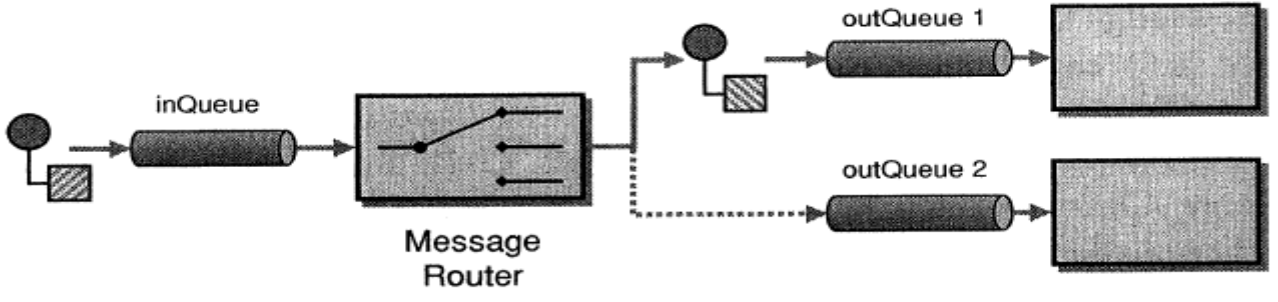
Connect the applications using a *Message Channel*, where one application writes information to the channel and the other one reads that information from the channel.



Message Router

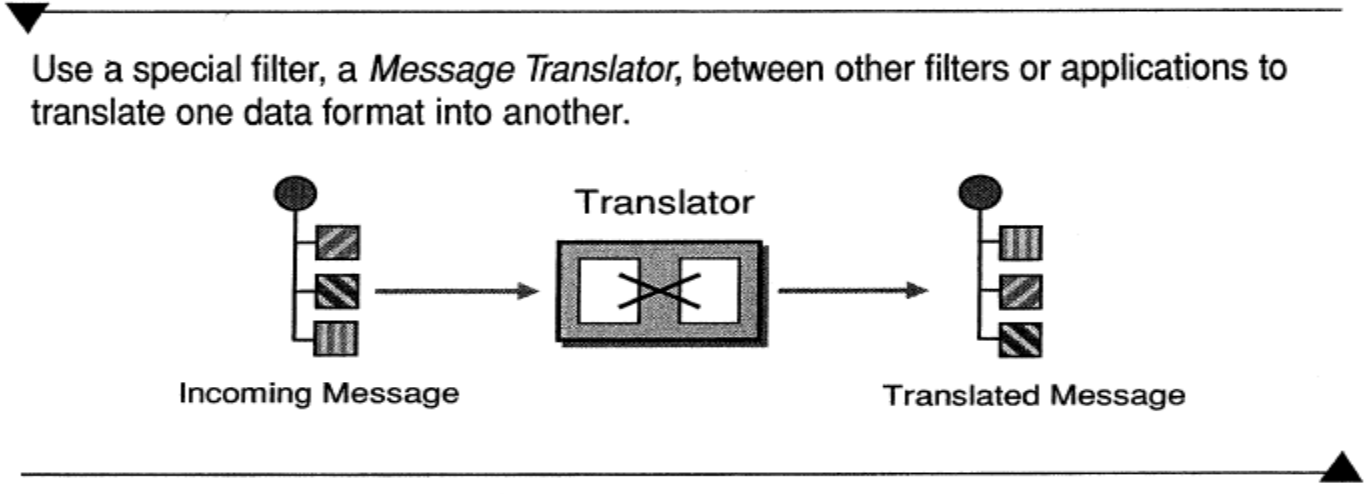
- Also called ‘topic based routing’

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

- (I think you should use with care ☺, or avoid)
  - You encode business logic into the connector



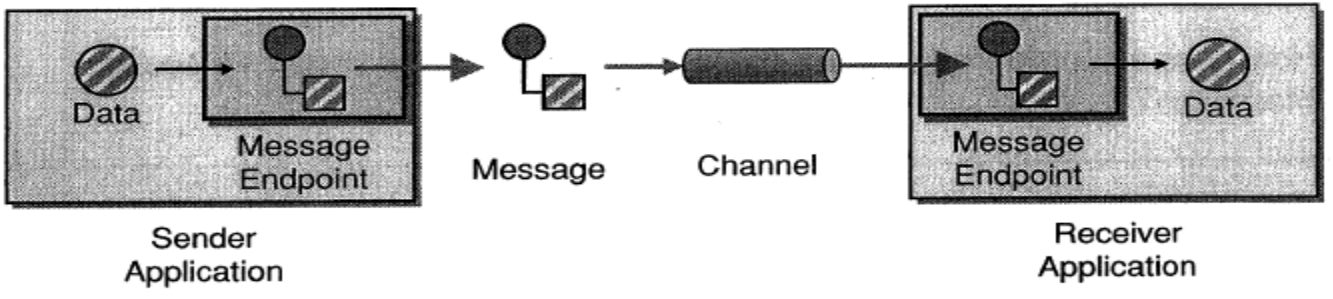
Message Endpoint

- Basically the ‘language-specific driver’
  - Ala the

4. **RabbitMQ Java Client** **700** usages

com.rabbitmq » amqp-client MPL GPL Apache

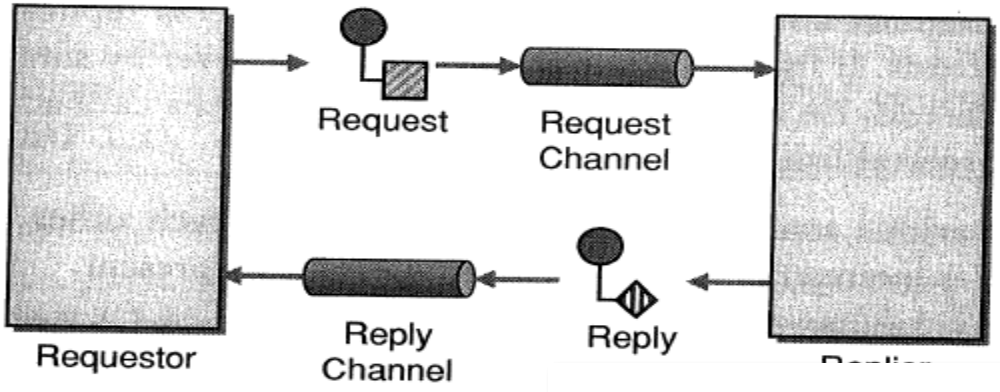
Connect an application to a messaging channel using a *Message Endpoint*, a client of the messaging system that the application can then use to send or receive Messages.

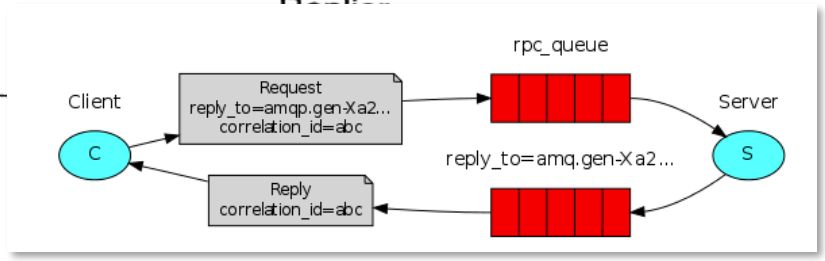


Request-Reply

- Simulate RPC

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

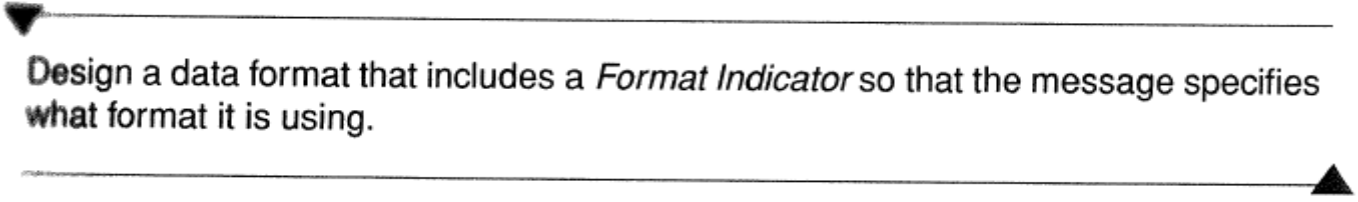




AARHUS UNIVERSITET

Format Indicator

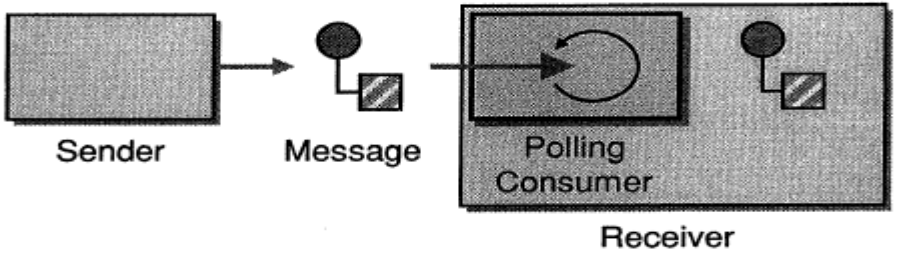
- Messages are opaque, but end-points need to agree on what the contents is and how it is formatted



- Example
  - All EcoSense ‘Karibu’ messages have a header of 6 bytes
    - GFS003: GroundFos dorm Sensor data, version 003

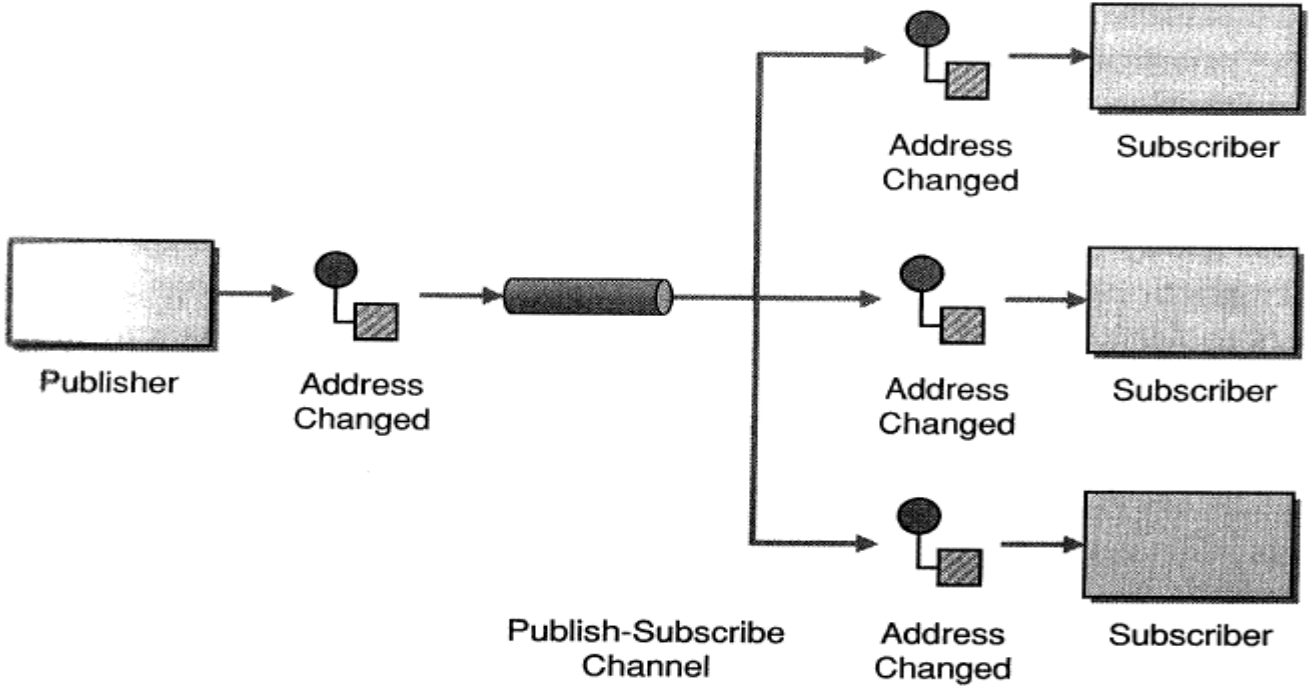
Polling Consumer

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



Publish-Subscribe Channel

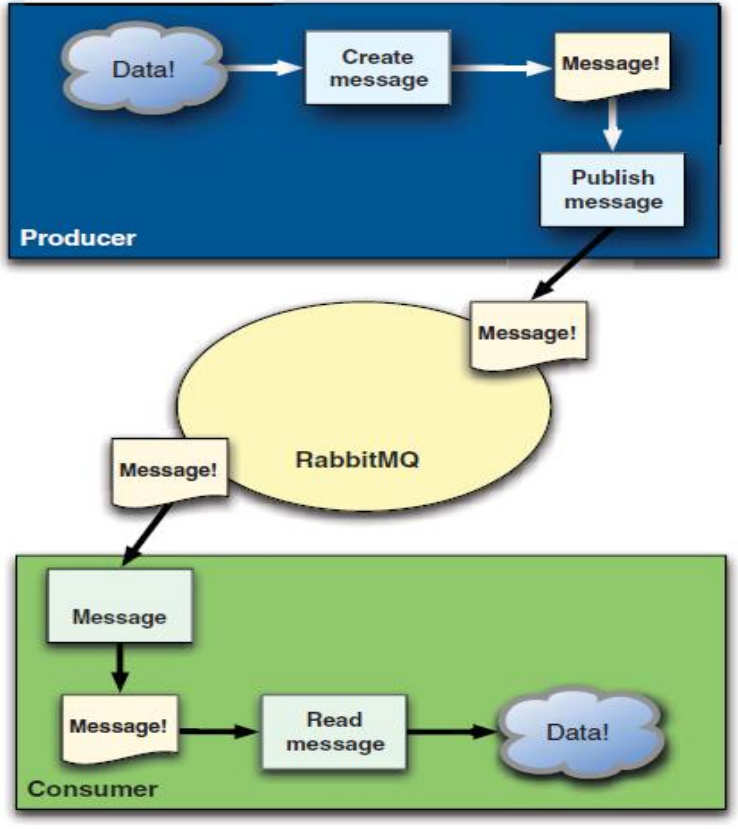
Send the event on a *Publish-Subscribe Channel*, which delivers a copy of a particular event to each receiver.



RabbitMQ

AARHUS UNIVERSITET

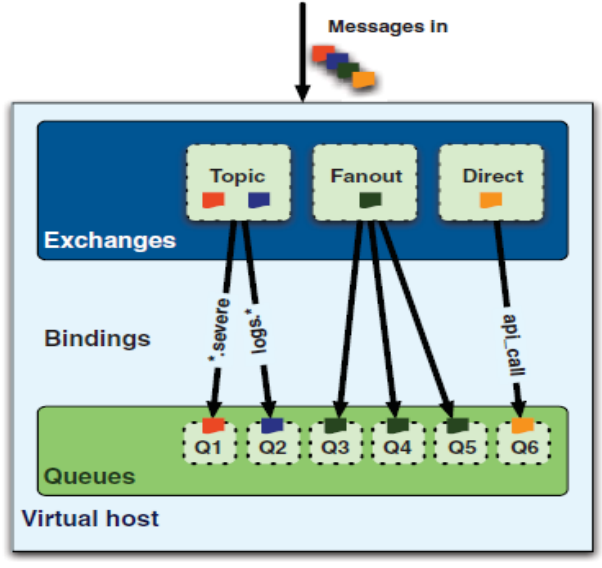
The Basic Architecture



AARHUS UNIVERSITET

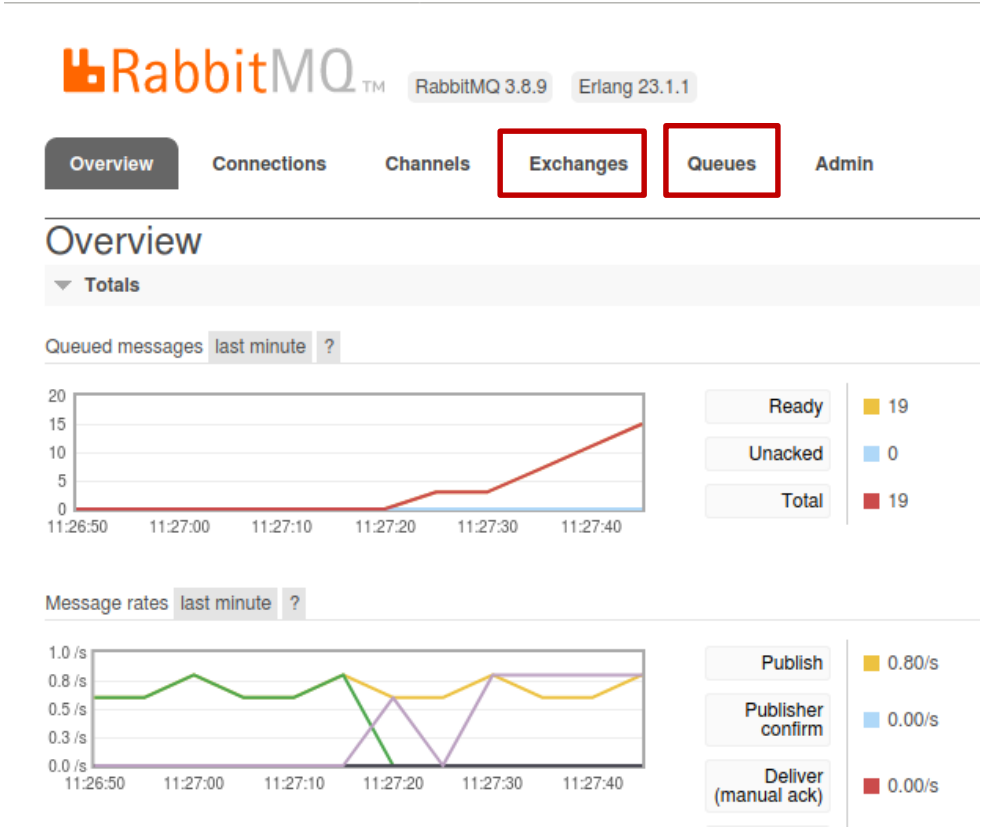
Exchanges and Queues

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



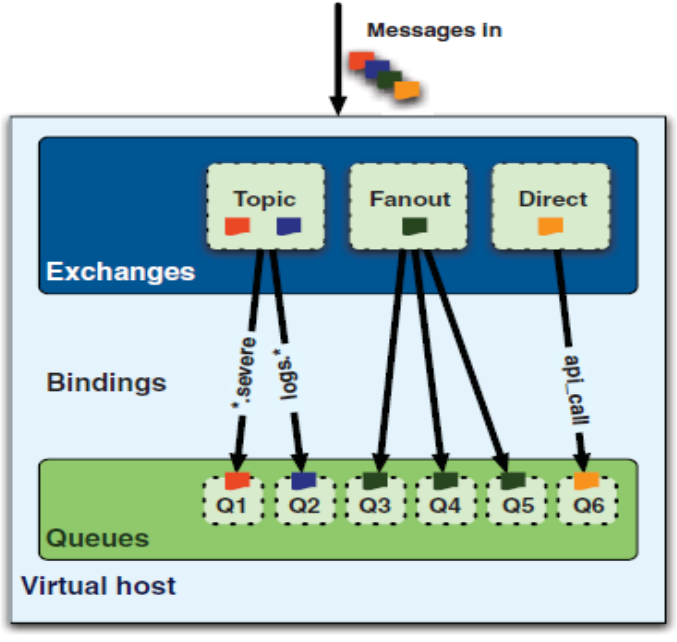
AARHUS UNIVERSITET

Visible in the Dashboard



Direct

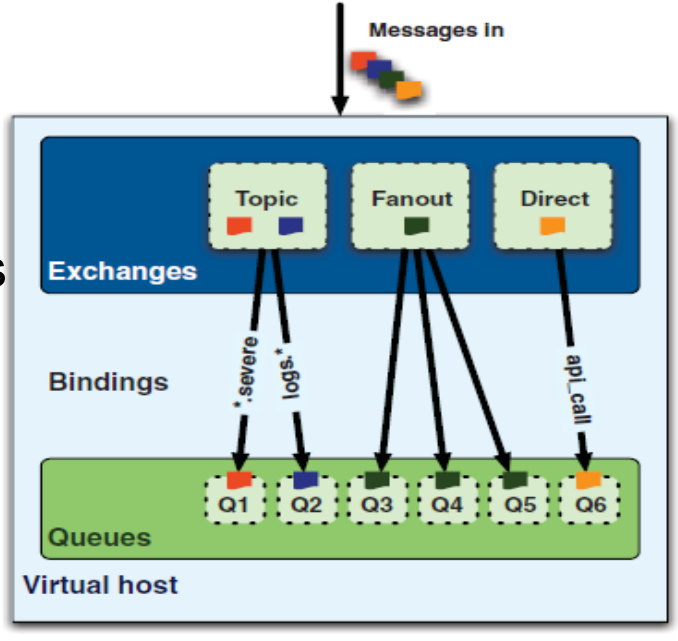
- *Direct* Send directly to receiver (explicit invocation)
  - Essentially it *seems* there is no exchange, because our message ends up on the queue right away
  - Exchange name = queue name



Fanout

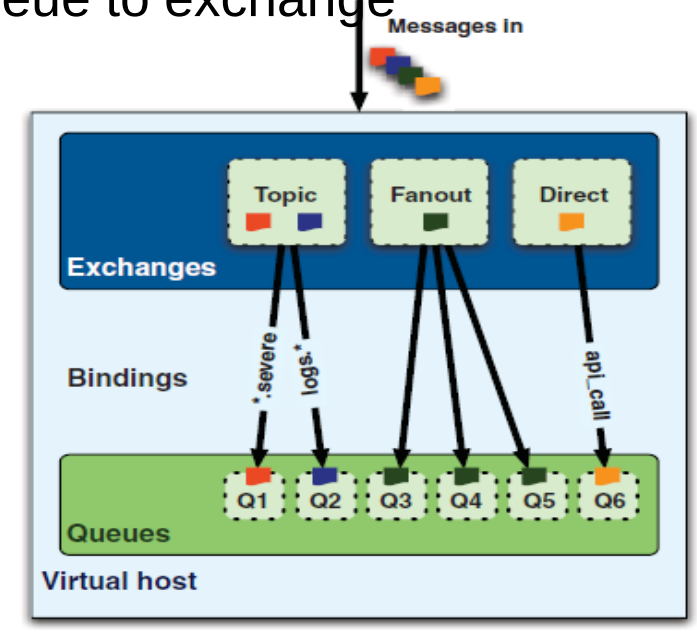
- *Fanout* Publish to subscribers (impl. Invocation/PubSub)
  - Clients pushes to a *named* exchange (ex: ”logs”)
  - Queues are *bound* to a named exhange (ex: Q3-logs)
  - Server pull from named queue

- Note the duplication of letters
  - You send ‘one letter’ but N receivers get each their copy of it…



Topics

- *Topics:* *Message Router* pattern
  - 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.
- Durability/Delivery mode
  - Queues/Exchanges default to *transient,* but can be *durable*
    - They will survive MQ restarts and crash
  - Messages can be *persistent*
    - They are written to disk, survive MQ restart

Lots of Options

- Message Time-to-live
  - TTL: messages that expire after given time

- Topic based messaging = many options
  - Cluster serves queues bound to '*.*.CRITICAL'
    - Nygard 'bulkhead' pattern – reserve computing capacity for most critical services (ala 'degraded mode')
  - Bound to '*.server7.*'
    - Allows 'sticky session' tactic, all messages are routed to a single server, which can then keep session state internally
      - Hm, not good for scalability…

Discussion

Availability

- Different time – Different process

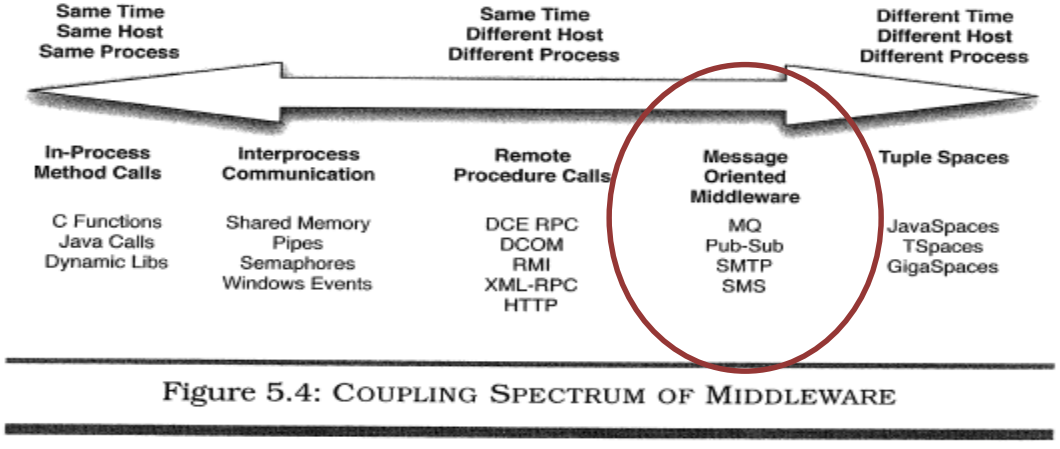


Figure 5.4: COUPLING SPECTRUM OF MIDDLEWARE

  - => 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 *queued* 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

Availability

- Can provide ‘elasticity’ during *impulses* to counter *unbalanced capacities (Nygard terminology)*
  - 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?
- WarStory:
  - Karibu daemon crash => 20 h of (70Kb/sec) data in queues

AARHUS UNIVERSITET

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…*

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…