Messaging Channels

System Integration
PBA Softwareudvikling/BSc Software Development
Tine Marbjerg
Fall 2017

Overall considerations about messaging channels

Message Channel Characteristics

Fixed set of channels

- Number of channels tends to be static - agreed upon at design time
- Possible exception: reply channel in *Request-Reply*

Unidirectional channels

- Channels are like buckets that applications add and take data from, but messages gives direction
- For practical reasons, two-way communication needs two channels (i.e. makes channels unidirectional)

Message Channel Decisions (1)

One-to-one or one-to-many channel?

- Message will be received by only one application (*Point-to-Point*)
- Message copied for each of the receivers (*Publish-Subscribe*)

What type of data on channel

- All data on a channel should be of the same type, i.e. structure, format etc. (*Datatype Channel*)
- Main reason that messaging systems needs lots of channels

Message Channel Decisions (2)

- **Error handling: What happens to invalid and undeliverable messages?**
  - If delivered properly, there is no guarantee the receiver knows what to do with an invalid message
    - Receiver puts the ‘strange’ message on *Invalid Message Channel*
  - Delivery problem
    - Messaging system puts message on *Dead Letter Channel*

- **Must messaging system be crash proof?**
  - By default, channels store messages in memory
  - *Guaranteed Delivery* makes channels persistent so messages are store on disk

Message Channel Decisions (3)

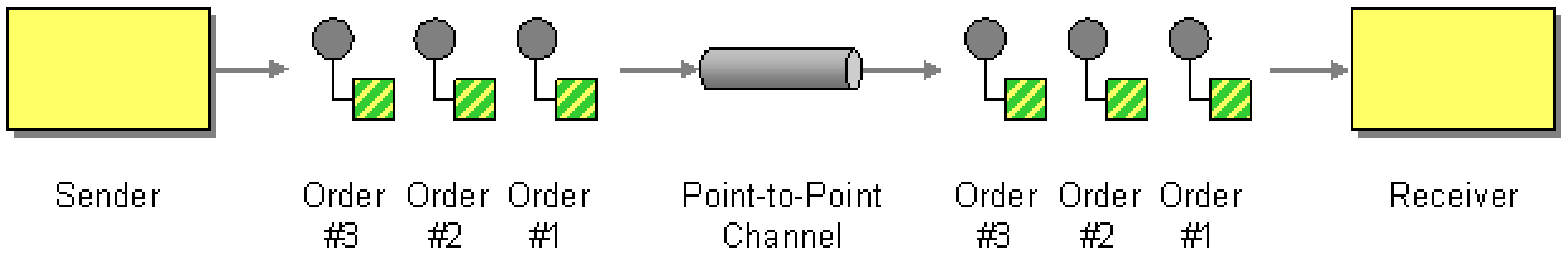
- **What to do with clients not built for messaging?**
  - *Channel Adapter* makes applications that cannot connect to a messaging system able to connect to a channel without modifying application

- **Channels as communication backbone**
  - Messaging system can become a centralized point for shared functionality in the enterprise
  - *Message Bus:* a backbone of channels that gives unified access to an enterprise's applications and makes them share functionality
    - Common data model
    - Command set of commands

Design of messaging channels: Use of Patterns

Point-to-Point Channel (103)

- How can the caller be sure that exactly one receiver will receive the document or perform the call?



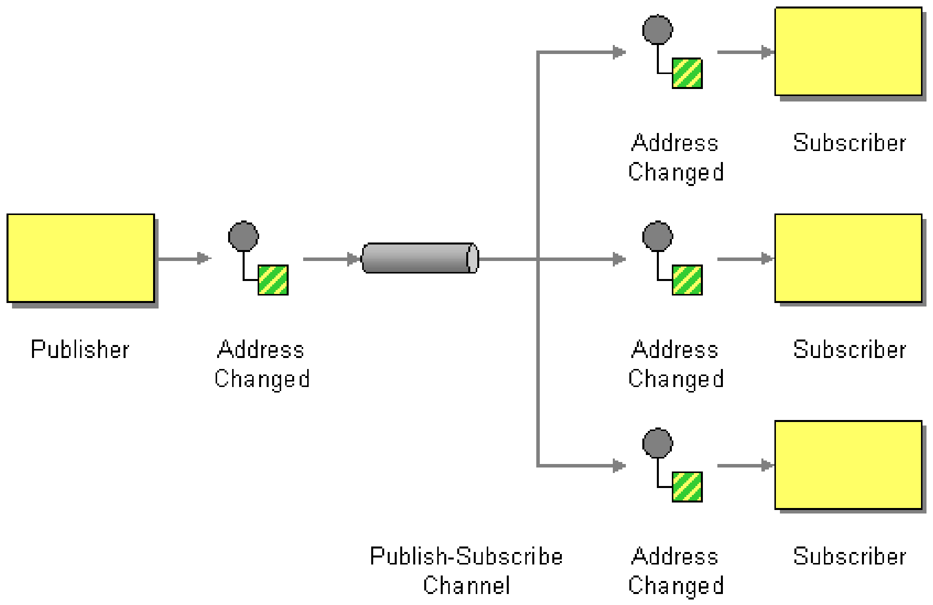
- Send the message on a *Point-to-Point Channel*, which ensures that only one receiver will receive a particular message.

OBS!

- If the channel has multiple receivers, only <u>one</u> of them can successfully consume a particular message.
- The <u>channel</u> <u>ensures</u> that only one of them succeeds, i.e. the receivers do not have to coordinate with each other.

Publish-Subscribe Channel (106)

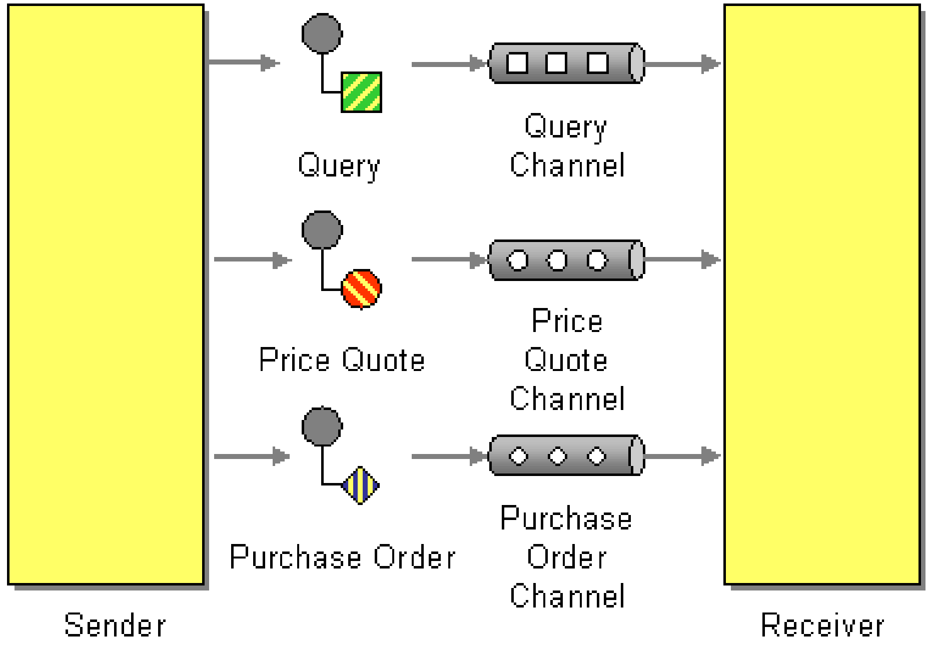
- How can the sender broadcast an event to all interested receivers?



- Send the event on a *Publish-Subscribe Channel*, which delivers a copy of a particular event to each receiver.
  - One input channel splits into multiple output channels
  - Each output channel has only one subscriber
  - MSMQ doesn't support natively (Multicasting support must be used)

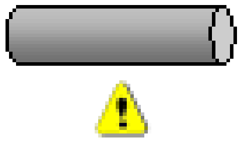
Datatype Channel (111)

- How can the application send a data item such that the receiver will know how to process it?

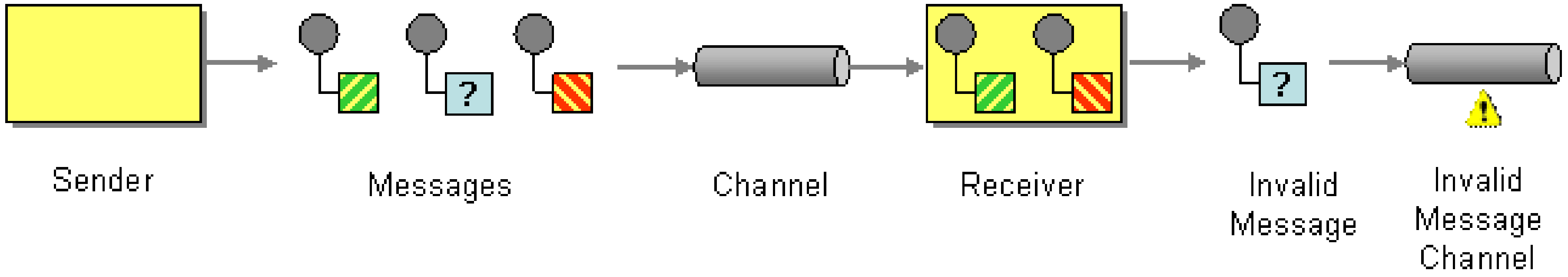


- Use a separate *Datatype Channel* for each data type, so that all data on a particular channel is of the same type.

Invalid Message Channel (115)



- How can a messaging receiver gracefully handle a message that makes no sense?

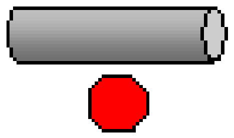


- The receiver should move the improper message to an *Invalid Message Channel*, a special channel for messages that could not be processed by their receivers.

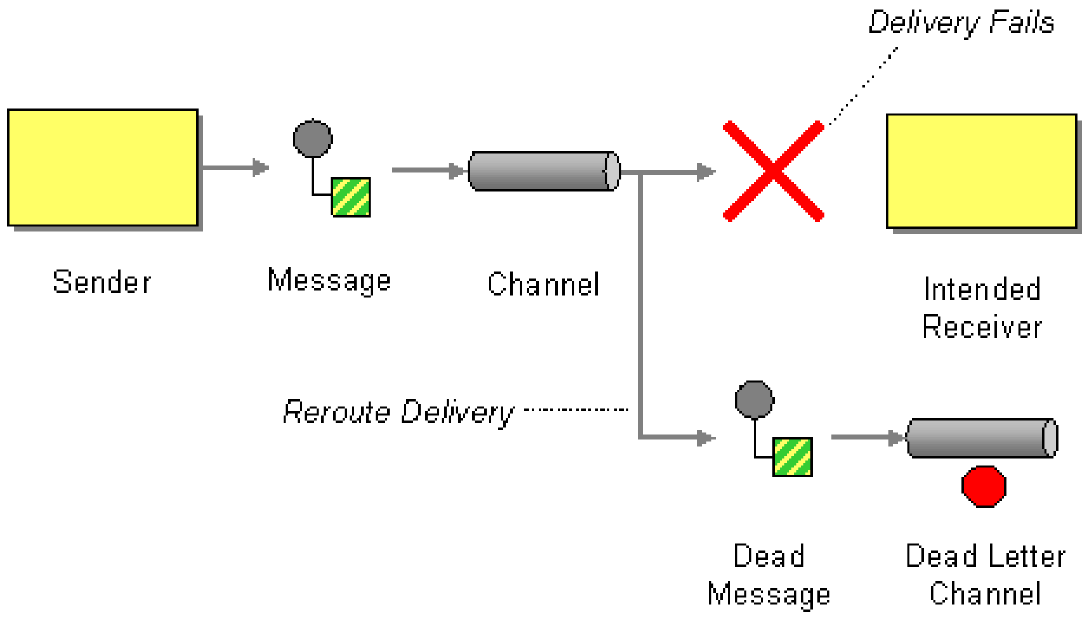
Invalid Message Example

```
//Receiver
...
try {
   // read message
}
catch ( Exception )
{
   //Invalid message detected
   invalidQueue.Send(requestMessage);
}
```

Dead Letter Channel (119)



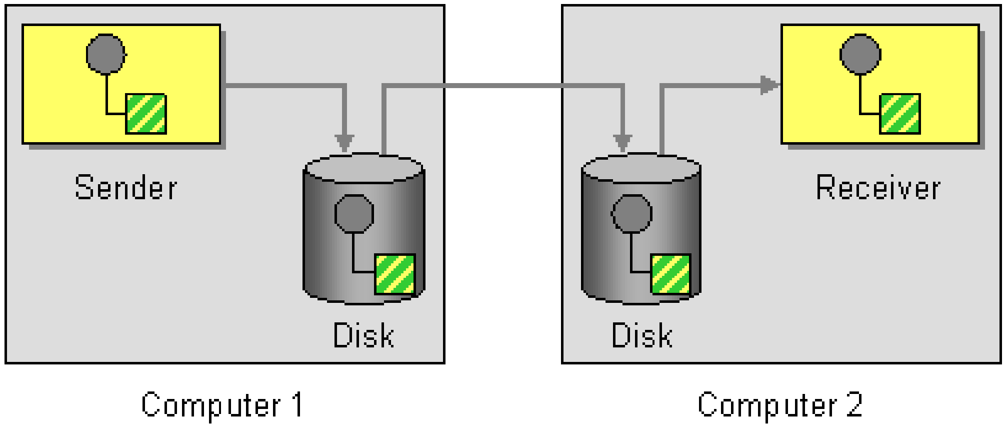
- What will the messaging system do with a message it cannot deliver?



- When a messaging system determines that it cannot deliver a message, it can move the message to a *Dead Letter Channel*.

Guaranteed Delivery (122)

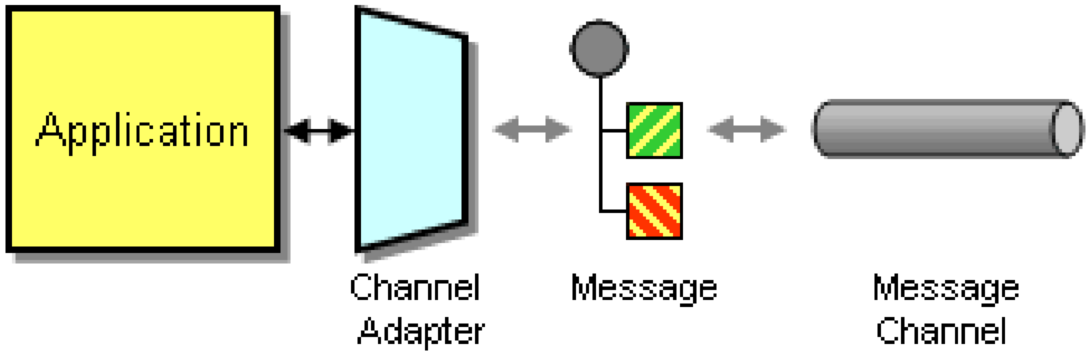
- How can the sender make sure that a message will be delivered, even if the messaging system fails?



- Use *Guaranteed Delivery* to make messages persistent so that they are not lost even if the messaging system crashes.
  - The msg. system uses a built-in data store to persist messages.
  - Each computer that the messaging system is installed on has its own data store so that the messages can be stored locally until successfully forwarded to and stored in the next data store
  - Hurts performance, but more reliable

Channel Adapter (127)

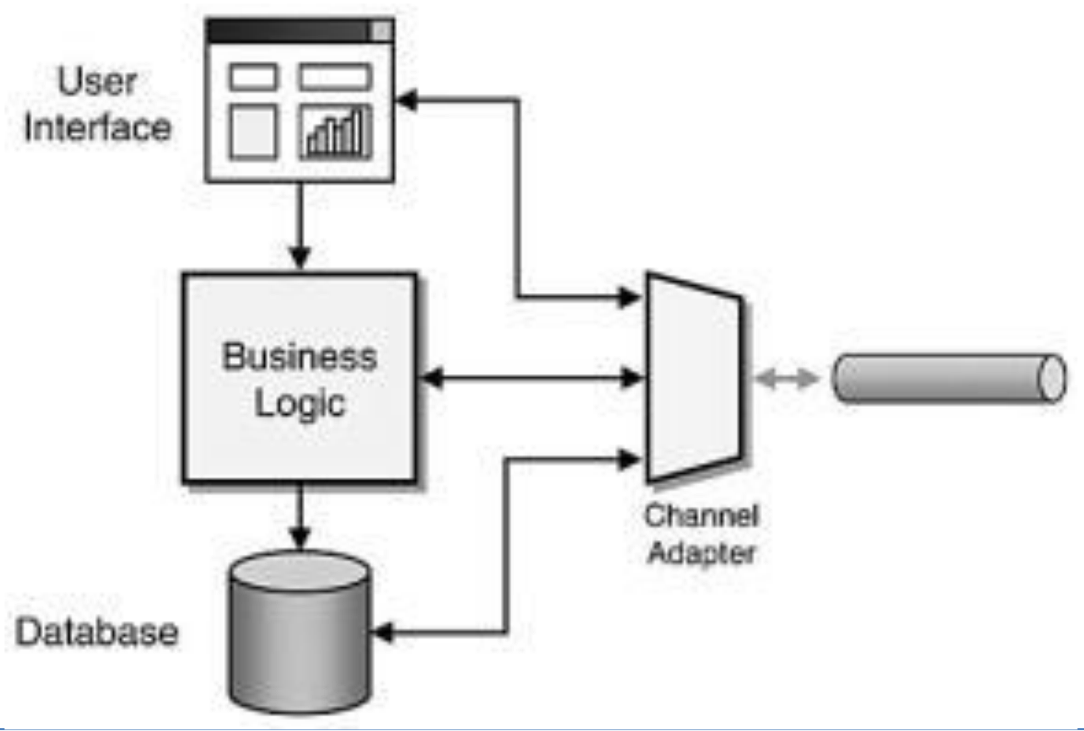
- How can you connect an application to the messaging system so that it can send and receive messages?



- Use a *Channel Adapter* that can access the application's API or data to publish messages on a channel based on this data, and that likewise can receive messages and invoke functionality inside the application.

Adapter - Connect to different layers

- Depending on application architecture, the Channel Adapter can connect to different layers in application:



Examples of Data Extraction

- Camel - A routing engine with domain specific languages

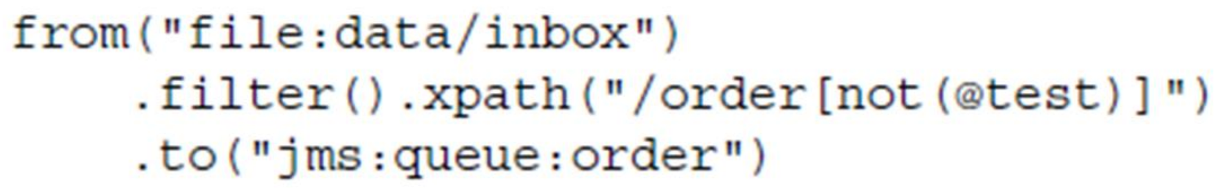
  ❑ **Java example 1**

  Define route that consumes files from a file endpoint to JMS channel:

  ```
  from("file:data/inbox").to("jms:queue:order");
  ```

  ❑ **Java example 2**

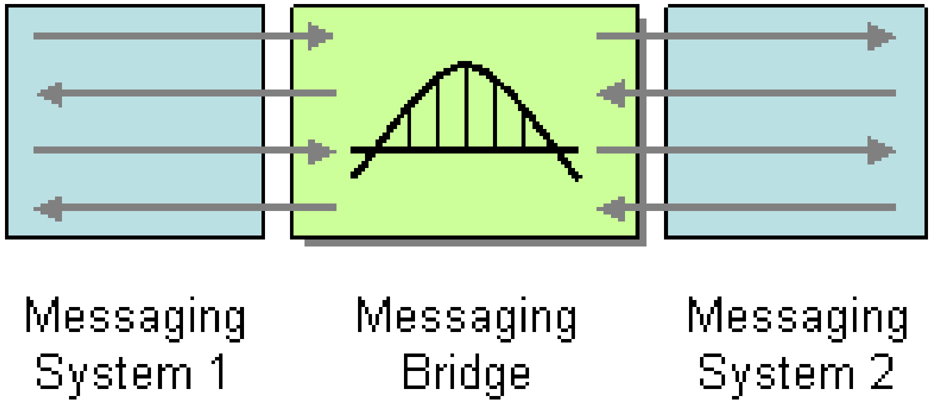
  Messages are routed to a filter, which uses XPath to check whether the message is a test order or not. If message passes the check, it routes to JMS endpoint.



```
from("file:data/inbox")
     .filter().xpath("/order[not(@test)]")
     .to("jms:queue:order")
```

Messaging Bridge (133)

- How can multiple messaging systems be connected so that messages available on one system are also available on the others?

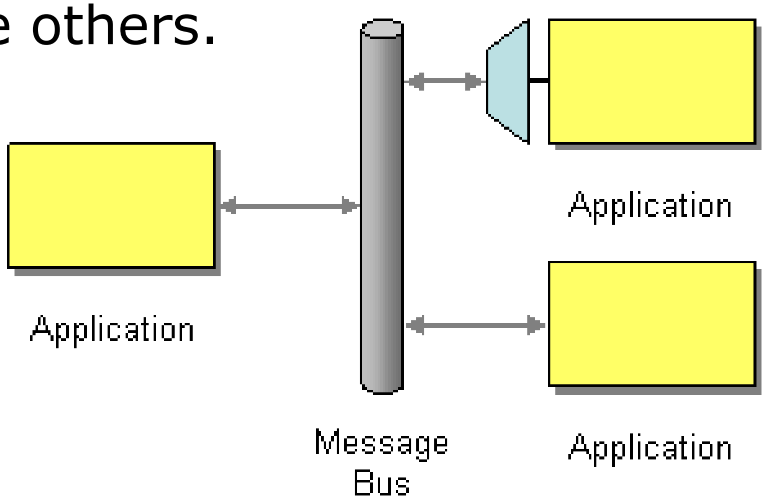


- Use a *Messaging Bridge*, a connection between messaging systems, to replicate messages between systems.

- The bridge acts as map from one set of channels to the other, and transforms the message format of one system to the other

Message Bus (137)

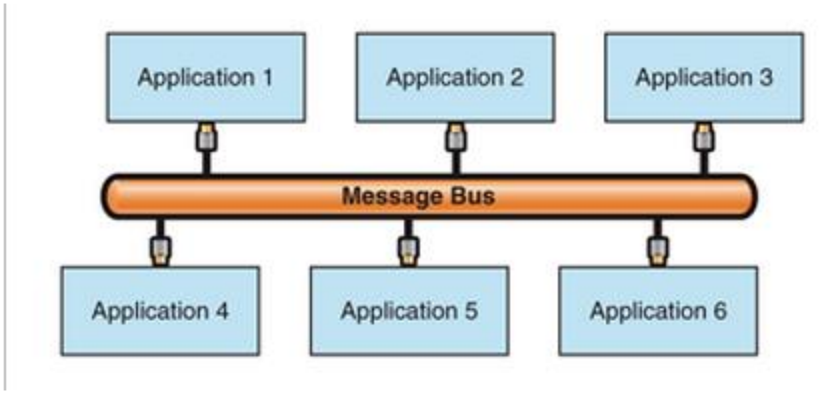
- Architecture that enables separate applications to work together, but in a decoupled fashion such that applications can be easily added or removed without affecting the others.

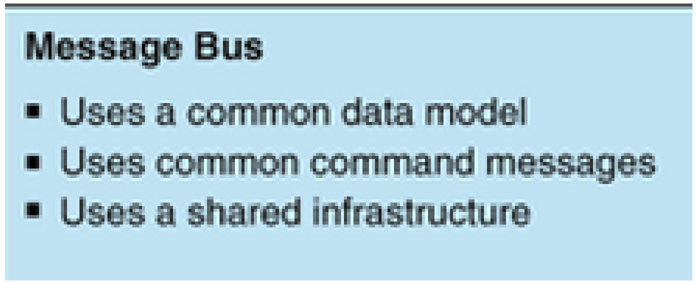


- *Message Bus* enables applications to work together using messaging.
  - Common data model
  - Command set of commands

Applications communicating through bus

- Application that sends messages must prepare the messages so that they comply with the type of messages the bus expects.
- Application that receives messages must be able to understand (syntactically) the message types.
- If all applications in the integration solution implement the bus interface, adding applications or removing applications from the bus incurs no changes.




Message Bus

- Uses a common data model
- Uses common command messages
- Uses a shared infrastructure