Message Construction

Systems Integration

PBA Softwareudvikling/BSc Software Development

Tine Marbjerg

Fall 2018

Today’s Topics

- Follow up on PDF Processing Exercise
- Introduction to RabbitMQ
- Message Construction (EIP chapter 5)
- Message Routing (EIP chapter 7)

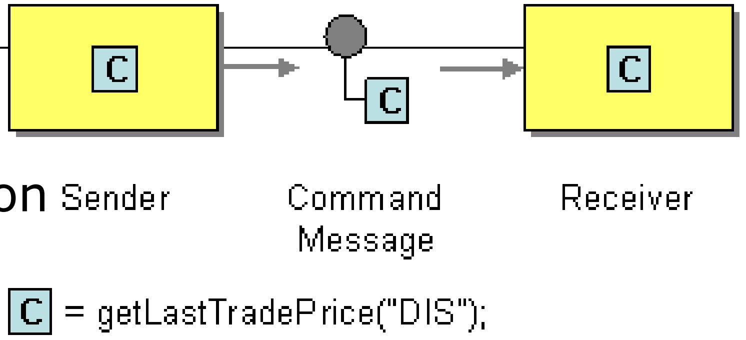
PDF Processing Exercise

- Solution(s)
- Problem(s)
- Comment(s)

Message Issues

• Message intent	
– *Command Message* (145)	invoke function
– *Document Message* (147)	send data
– *Event Message* (151)	send notification
• Returning a response	
– *Request-Reply* (154)	want a reply
– *Return Address* (159)	where to put reply
– *Correlation Identifier* (163)	link request to reply by id
• Large amounts of data	
– *Message Sequence* (170)	break data into manageable chunks
• Slow messages	
– *Message Expiration* (176)	put deadline on time-sensitive messages
• Design data format	
– *Format Indicator* (180)	specification of message format

Command Message

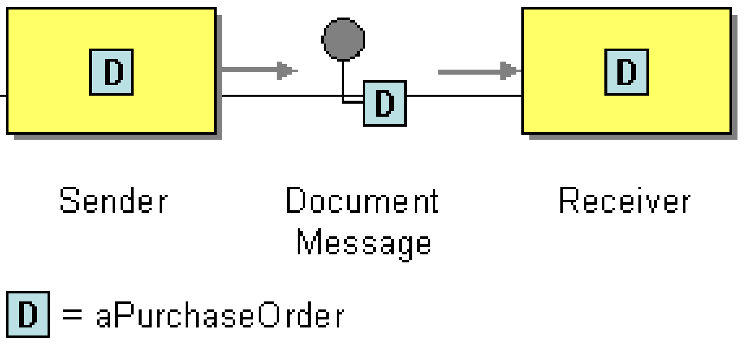


- Invoke functionality in other application

- SOAP and WSDL example (EIP p. 146)
  - RPC-style SOAP message is example of *Command Message* pattern

```
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope"
 soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
  <soap:Body>
    <m:GetLastTradePrice xmlns:m=“Some-URI">
       <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </soap:Body>
 </soap:Envelope>
```

Document Message

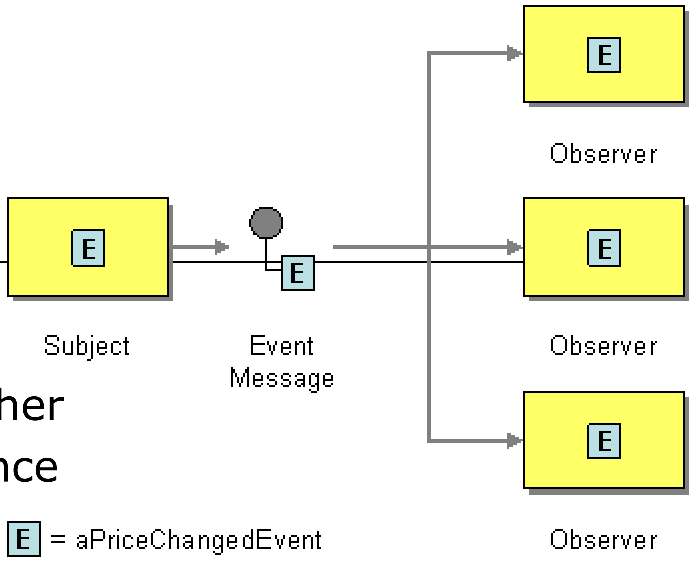


- Transfer data to other application

- SOAP and WSDL example (EIP p. 150)

```
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope"
 soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
  <soap:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
       <symbol>34.5</symbol>
    </m:GetLastTradePriceResponse>
  </soap:Body>
</soap:Envelope>
```

Event Message



- Transmit event from one application to another
- Many events are empty; their mere occurrence tells the observer to react

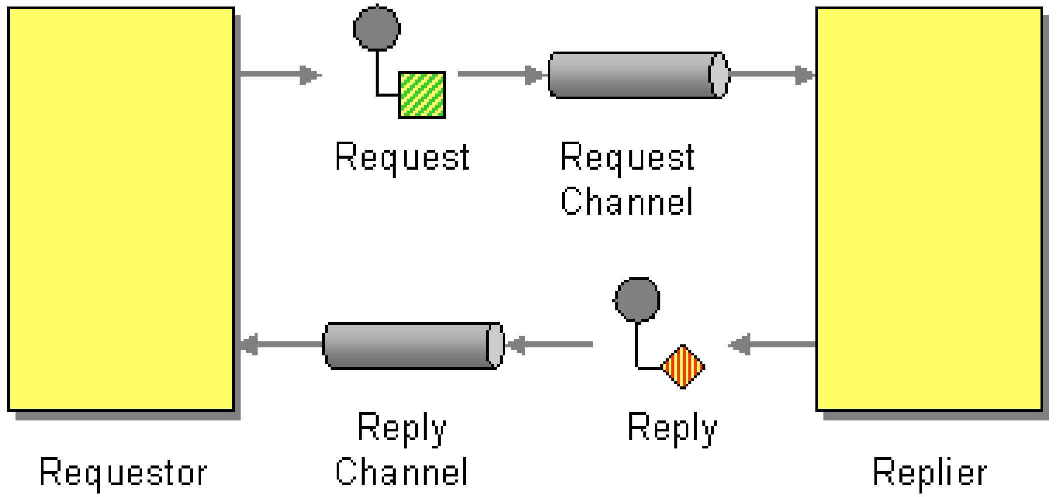
Observer Pattern

- The **push model** sends information about the change as part of the update
  - combined *Document/Event message*

- The **pull model** sends minimal information and observers can afterwards request state from the subject
  1. *Event Message* to notify observer about **update**
  2. *Command Message* send from observer to subject (**state request)**
  3. *Document Message* from subject to observer (**state reply**)

Request-Reply

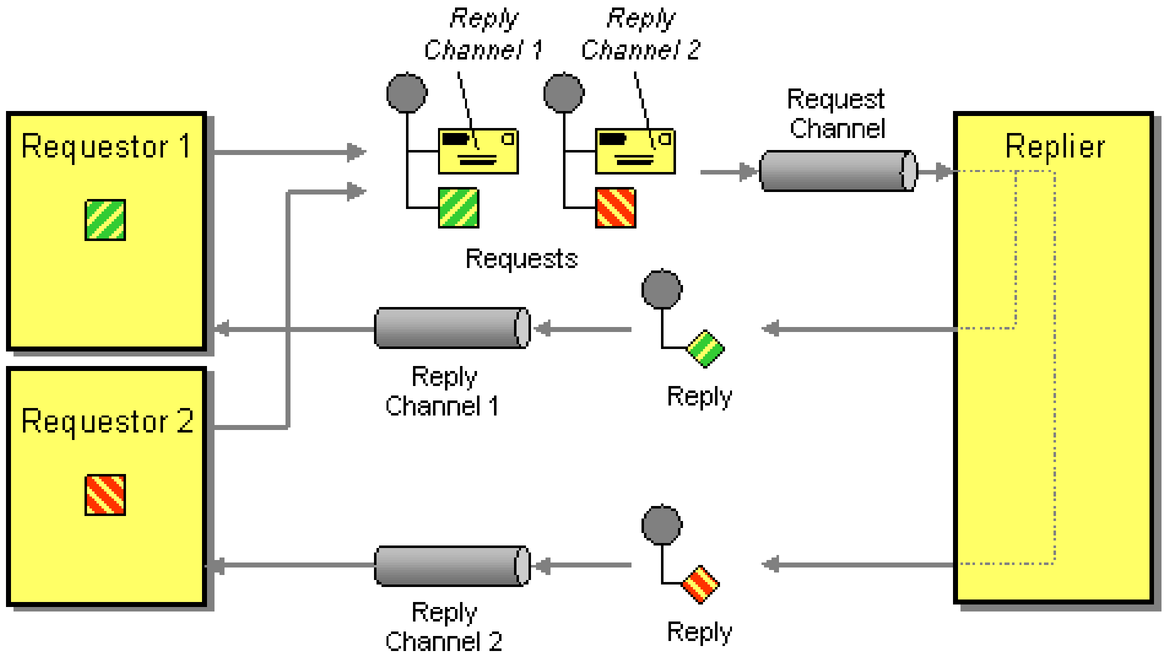
- ***When an application sends a message, how can it get a response from the receiver?***
- Send a pair of *Request-Reply* messages, each on its own channel



- The request channel can be a *Point-to-Point Channel* or a *Publish-Subscribe Channel*
- The reply channel is almost always point-to-point– reply should only be returned to the requestor

Return Address

- **How does a replier know where to send the reply?**



- The request message should contain a *Return Address* that indicates where to send the reply message.

.NET Request-Reply example

.NET Sender code

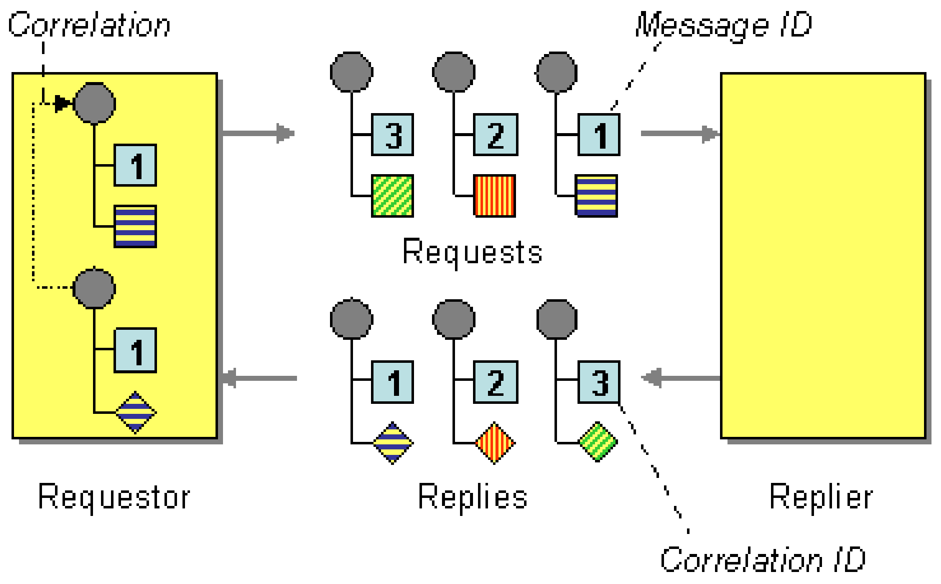
```
Message requestMessage = new Message();
requestMessage.Body = "Hello world.";
requestMessage.ResponseQueue = replyQueue;
requestQueue.Send(requestMessage);
```

.NET Receiver code

```
MessageQueue replyQueue = requestMessage.ResponseQueue;
Message replyMessage = new Message();
replyMessage.Body = // specify message
replyQueue.Send(replyMessage);
```

Correlation Identifier

- **How does a requestor that has received a reply know which request this is the reply for?**



- Each reply message should contain a *Correlation Identifier*, a unique identifier that indicates which request message this reply is for

RabbitMQ – Request-Reply example

- Reply to queue is set in the properties of the message.
- Client sends request and receives response

```
AMQP.BasicProperties props = new AMQP.BasicProperties.
                Builder()
                .correlationId(corrId)
                .replyTo(replyQueueName)
                .build();

channel.basicPublish("", RPC_QUEUE_NAME, props, message.getBytes());
```

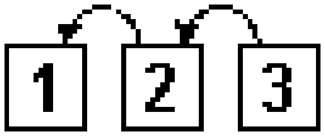
```
channel.basicConsume(replyQueueName, true, new DefaultConsumer(channel) {
  @Override
  public void handleDelivery(String consumerTag, Envelope envelope,
       AMQP.BasicProperties properties, byte[] body) throws IOException {
       if (properties.getCorrelationId().equals(corrId)) {
          response.offer(new String(body, "UTF-8"));
          String message = new String(body, "UTF-8");
          System.out.println(" [x] Received '" + message + "'");
       }
   }
});
```

RabbitMQ – Request-Reply example

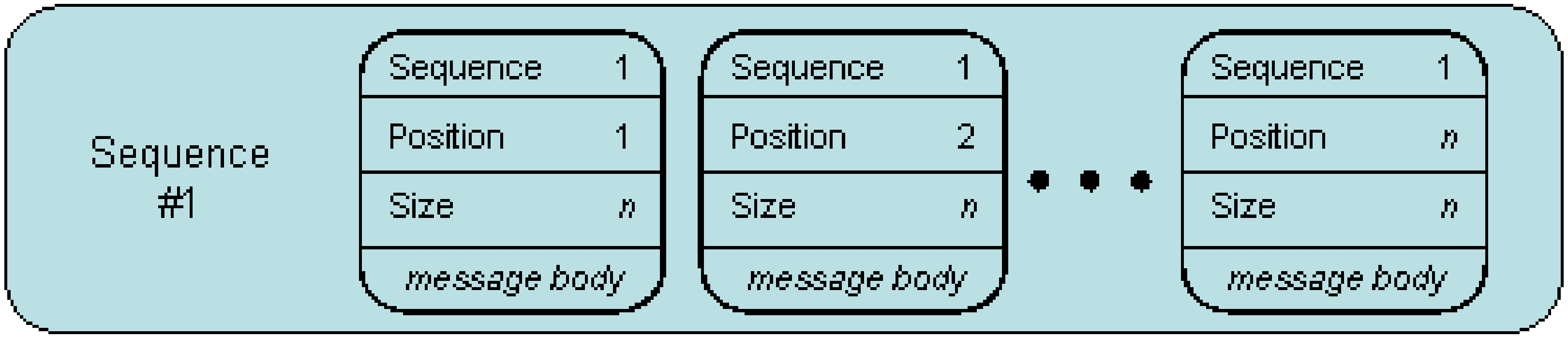
- Server handles request and sends response

```
...
Consumer consumer = new DefaultConsumer(channel) {
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
        AMQP.BasicProperties properties, byte[] body) throws
        IOException {

        AMQP.BasicProperties replyProps =
                new AMQP.BasicProperties
                .Builder()
                .correlationId(properties.getCorrelationId())
                .build();

        // make reply

        channel.basicPublish("", properties.getReplyTo(), replyProps,
        response.getBytes("UTF-8"));
        channel.basicAck(envelope.getDeliveryTag(), false);
        // close
```

Message Sequence



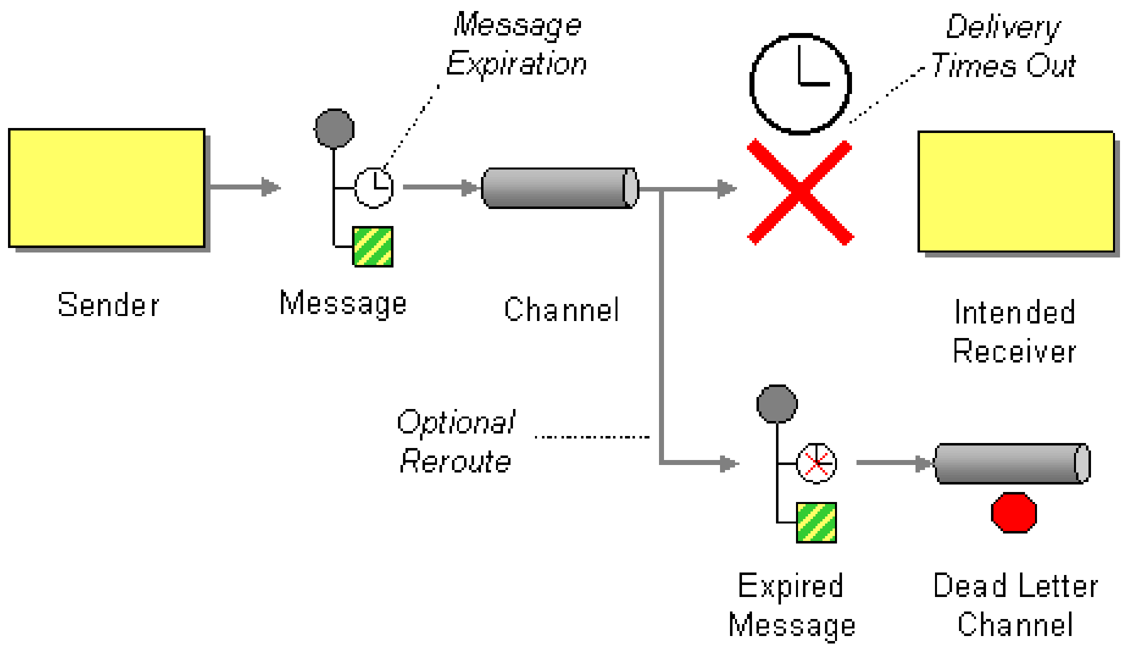
- **How can messaging transmit an arbitrarily large amount of data?**



- Whenever a large set of data may need to be broken into message-size chunks, send the data as a *Message Sequence* and mark each message with sequence identification fields.

Message Expiration

- **How can a sender indicate when a message should be considered stale and thus shouldn’t be processed?**

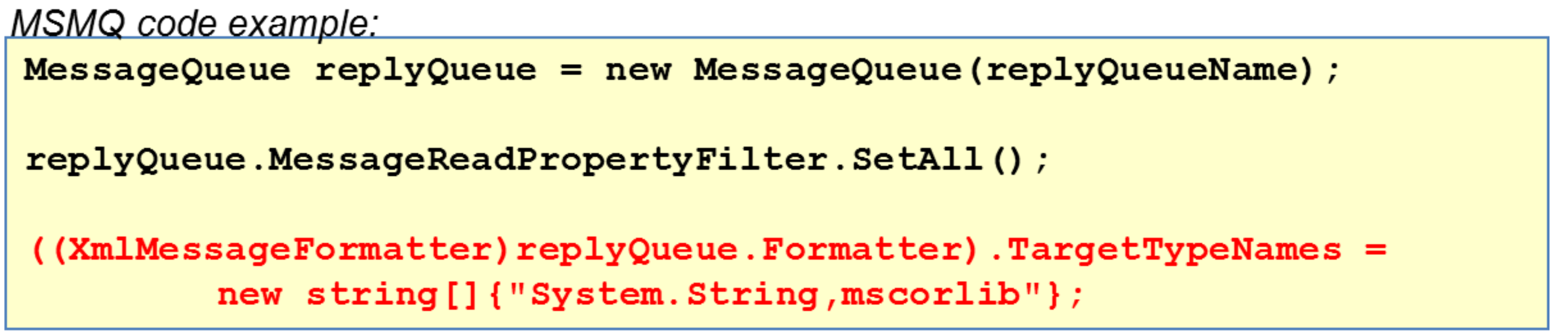


- Set the *Message Expiration* time stamp on the message
  - Like the expiration date on a milk carton ☺

Format Indicator

- **How can a message’s data format be designed to allow for possible future changes?**
- Design a data format that includes a *Format Indicator*, so that the message specifies what format it is using.
- Enables the sender to tell the receiver the format of the message.
- A receiver expecting several possible formats knows which one a message is using and therefore how to interpret the message’s contents.

Simple MSMQ code example

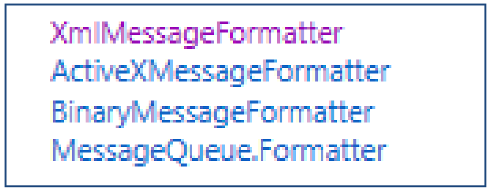


MSMQ code example:

```
MessageQueue replyQueue = new MessageQueue(replyQueueName);

replyQueue.MessageReadPropertyFilter.SetAll();

((XmlMessageFormatter)replyQueue.Formatter).TargetTypeNames =
         new string[]{"System.String,mscorlib"};
```

.NET Message Formatting Examples

`IMessageFormatter` produces a stream to be written to or read from the message body.


XmlMessageFormatter
ActiveXMessageFormatter
BinaryMessageFormatter
MessageQueue.Formatter

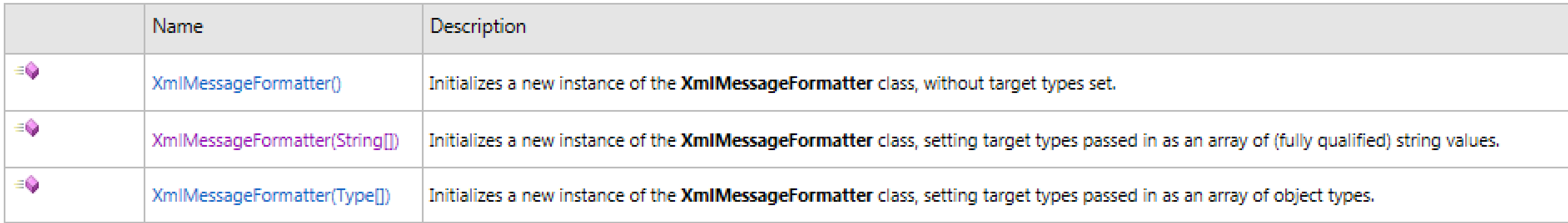
Examples

XML formatter on queue

```
myQueue.Formatter = new XmlMessageFormatter(new Type[]
        {typeof(MyProject.Order)});
```

XML formatter on message

```
myMessage.Formatter = new XmlMessageFormatter(new String[]
        { "System.String,mscorlib" })
```



	Name	Description
◆	XmlMessageFormatter()	Initializes a new instance of the **XmlMessageFormatter** class, without target types set.
◆	XmlMessageFormatter(String[])	Initializes a new instance of the **XmlMessageFormatter** class, setting target types passed in as an array of (fully qualified) string values.
◆	XmlMessageFormatter(Type[])	Initializes a new instance of the **XmlMessageFormatter** class, setting target types passed in as an array of object types.

RabbitMQ example

- Sender

```
AMQP.BasicProperties props = new AMQP.BasicProperties
                 .Builder()
                 .contentType("application/json")
                 .build();
```

RabbitMQ example

- Receiver

```
Consumer consumer = new DefaultConsumer(channel) {

   @Override
   public void handleDelivery(String consumerTag, Envelope
           envelope, AMQP.BasicProperties properties, byte[]    body)
           throws IOException {

    String message = new String(body, "UTF-8");

    System.out.println("content type"+properties.getContentType());
   }
};
```

Alternative Implementations

- **Version Number.** Number or string that uniquely identifies the format
- **Foreign Key**. Unique ID (filename, URL etc.) that specifies a format document
- **Format Document**. Schema that describes the data format. Embedded in the message –not referenced by a number or a key
- Version Number and Foreign Key can be stored in the header field and has to be agreed upon