MTAT.03.229
Enterprise System Integration

Lecture 9: Enterprise Integration Patterns

Marlon Dumas
University of Tartu

Based on slides by Luciano García-Bañuelos
The picture

• What if the execution of an enterprise software spans across the boundaries of an organization?
  ◦ Goal: Interoperability
  ◦ Challenges: Heterogeneity, Distribution, Loose-coupling, Scalability, Security, etc.

• Our focus today will be on communication styles and integration patterns
The pitfall of synchronous interactions
The alternative: message-based asynchronous interactions

- It might be a separate tier
- It might add some non-functional features:
  - Persistence of messages
  - Transactional interactions
  - Priority management & load balancing
  - Reliability (e.g., exactly-once delivery)
Main takeaway

• If you architect your integration layer using synchronous interactions you’ll have a hard time achieving:
  ◦ Scalability
  ◦ Reliability
  ◦ Maintainability / Flexibility

• Synchronicity is expensive
Thinking “Asynchronously”

Synchronous

Asynchronous
Messing styles

- **Point-to-Point**
  - built around the concept of **message queues**
  - each message has only one consumer

- **Publish-Subscribe**
  - uses the notion of **message topics** for sending/delivering messages
  - each message has multiple consumers
The promises of messaging

- Channels are separate from applications
  - Removes location dependencies
- Channels are asynchronous & reliable
  - Removes temporal dependencies
- Data is exchanged in self-contained messages
  - Avoids data format dependencies (Payload semantics)
Messaging patterns

- Gregor Hohpe defined a visual pattern language describing message-based enterprise integration solutions.

- The pattern language comprises 65 patterns in 6 categories.

http://www.enterpriseintegrationpatterns.com/eaipatterns.html
Messaging patterns

http://www.enterpriseintegrationpatterns.com/eaipatterns.html
Classic books for the future architect in you...

• Gang of Four (GoF) Design Patterns
  ◦ Heads First Design Patterns is a good supplement to practice (Freeman et al.)

• Refactoring (Fowler, Beck, …)

• Clean Code (Robert Martin)

• Enterprise Integration Patterns (Hohpe)
Request-response patterns
Pattern: Async. request-response

- A pair of endpoints: Consumer and Provider (similar to RPC)
- Channels are unidirectional
- Two asynchronous point-to-point channels
- Separate request and response messages
Scenario: Multiple consumers

- Each consumer has its own reply queue
- But how does the provider send the response?
  - Could send to all consumers (very inefficient)
  - Hard code (violates principle of context-free service)
Pattern: *Return address*

- Consumer specifies *Return Address* (the reply channel) in the request message
- Service provider sends response message to specified channel
Scenario: Load-balanced service providers

- Request message can be handled by multiple providers
- *Point-to-point* channel supports competing services
- Only one service receives each request message
- But what if the response messages are out of order?
Pattern: **Correlation identifier**

- Consumer assigns a unique identifier to each message
  - Identifier can be an arbitrary ID, a GUID, a business key

- Provider copies the ID to the response message

- Consumer can match request and response
Integration patterns in context

AN ORDER PROCESSING SYSTEM
Scenario: Multiple specialized providers

- Each provider can only handle a specific type of message
- Route the request to the "appropriate" provider. But how?
  - *Do not want to burden sender with decision*
  - *Letting providers "pick out" messages requires coordination*
Pattern: *Content-based router*

- Insert a content-based router
- Routers forward incoming messages to different channels
- Message content not changed
- Mostly stateless, but can be stateful, e.g. de-duper
Scenario: Composite messages

• How can we process a message that contains multiple elements?
Patterns: *Splitter + Router*

- Use a splitter to break out the composite message into a series of individual messages
- Then use a router to route the individual messages as before
- Note that two patterns are composed
Scenario: Composing a single response

- How to combine the results of individual but related messages?
  - Messages can be out-of-order, delayed
  - Multiple conversations can be intermixed
Pattern: **Aggregator**

- Collects and stores messages until a complete set has been received (completeness condition)
- Publishes a single message created from the individual messages (aggregation algorithm)
Scenario: Multiple providers

- How to send a message to a dynamic set of recipients?
- And then return a single response message?
Pattern: *Scatter & gather*

- Send message to a pub-sub channel
- Interested recipients subscribe to a "topic"
- Aggregator collects individual response messages
  - may not wait for all quotes, only returns one quote
• Receive an order message
• Use splitter to create one message per item
• Send to scatter/gather which returns "best quote" message
• Aggregate to create quoted order message
Hands-on with Spring Integration
Spring Integration

• Concrete implementation of Enterprise Integration Patterns
  ◦ Lightweight intra-application messaging
  ◦ Leveraging middleware for system integration

• Provides declarative adapters as the building blocks
  ◦ High-level abstraction over Spring's support for remoting, messaging, and scheduling

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-integration</artifactId>
</dependency>
```

```
<dependency>
  <groupId>org.springframework.integration</groupId>
  <artifactId>spring-integration-xml</artifactId>
</dependency>
```

```
<dependency>
  <groupId>org.springframework.integration</groupId>
  <artifactId>spring-integration-mail</artifactId>
</dependency>
```

```
<dependency>
  <groupId>javax.mail</groupId>
  <artifactId>javax.mail-api</artifactId>
  <version>1.5.3</version>
</dependency>
```
Invoice processing

- Invoices can be submitted via a REST endpoint
- Invoices can be submitted via email
- Invoice processing is routed based on the invoice total

FILE:  src/main/resources/META-INF/spring/applicationContext-InvoiceProcessing.xml
Email inbound-channels

```xml
<int-mail:inbound-channel-adapter id="inboundMailAdapter"
    store-url="imaps://USERNAME:PASSWORD@imap.gmail.com:993/INBOX"
    channel="receiveEmailChannel"
    should-delete-messages="false"
    should-mark-messages-as-read="true"
    auto-startup="true">
    <int:poller
        max-messages-per-poll="1" time-unit="SECONDS" fixed-delay="10"/>
</int-mail:inbound-channel-adapter>
```
Extraction of email attachment

```java
@Component
public class InvoiceMailPreprocessor {
    @ServiceActivator
    public Document process(Message msg) throws Exception {
        Document invoiceXML = null;
        Object _content = msg.getContent();
        if (_content instanceof Multipart) {
            Multipart content = (Multipart) _content;
            for (int i = 0; i < content.getCount(); i++) {
                BodyPart part = content.getBodyPart(i);
                if (part.getContentType().startsWith("text/xml")) {
                    String fileName = part.getFileName();
                    if (fileName.startsWith("invoice")) {
                        DocumentBuilder builder = DocumentBuilderFactory.newInstance();
                        invoiceXML = builder.parse(part.getInputStream());
                        break;
                    }
                }
            }
        }
        if (invoiceXML == null)
            throw new IOException("No invoice was found!");
        return invoiceXML;
    }
}
```
```java
@RestController
@RequestMapping("/rest/phrs")
public class PlantHireRequestRESTController {

@Autowired
InvoiceGateway gateway;

@RequestMapping(method = RequestMethod.POST, value = "/{phrId}/invoice",
consumes="application/xml")
@ResponseStatus(HttpStatus.CREATED)
public void pushInvoice(@PathVariable("phrId") Long phrId,
@RequestBody String string) throws Exception {
    DocumentBuilder parser = DocumentBuilderFactory.newInstance().newDocumentBuilder();
    Document invoice = parser.parse(newByteArrayInputStream(string.getBytes()));
    gateway.process(invoice);
}

public interface InvoiceGateway {
    @Gateway
    void process(Document invoice);
}
```
Invoice routing

```xml
<int-xl:xpath-filter id="invoiceFilter"
  input-channel="invoiceChannel"
  output-channel="fastTrackPathChannel"
  discard-channel="normalPathChannel">  
  <int-xl:xpath-expression expression="//total &lt;= 1000.00"/>
</int-xl:xpath-filter>
```
Acknowledgements

• These slides are based on material from
  ◦ Gregor Hohpe and Bobby Woolf
    Enterprise Integration Patterns
    Addison-Wesley Professional, 2003

  ◦ Erik Doernenburg