C24

www.C24.biz

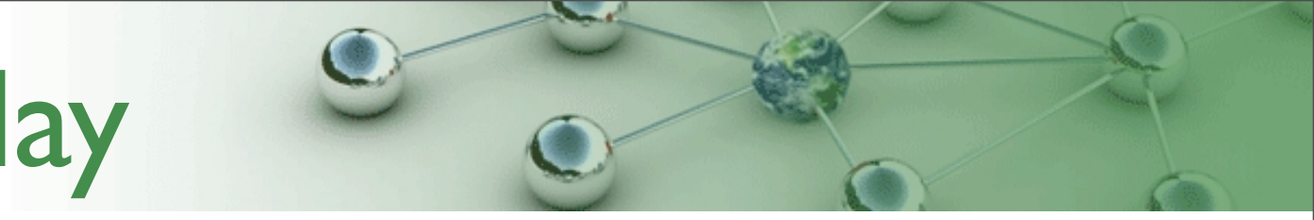
Practical Enterprise Integration

Andrew Elmore

Why so problematic?

- There are a large number of variables
  - Transports - FTP, HTTP, TCP, SMTP, File, ZipFile, JMS, ...
  - Types - CSV, XML, Fixed Length, ...
  - Formats - SWIFT, HL7, EDIFACT, SS7, ...
  - External systems - each with their own interpretation of the above
  - Business rules - what can (and should) be received when
  - SLAs - processing times, cut-off windows, latency & throughput
- Even better, they keep changing
  - External systems evolve (== change their mind & introduce new bugs)
  - Standards evolve
  - New clients/partners need to be integrated
- For many of us, these aren't our area of expertise

Themes for today

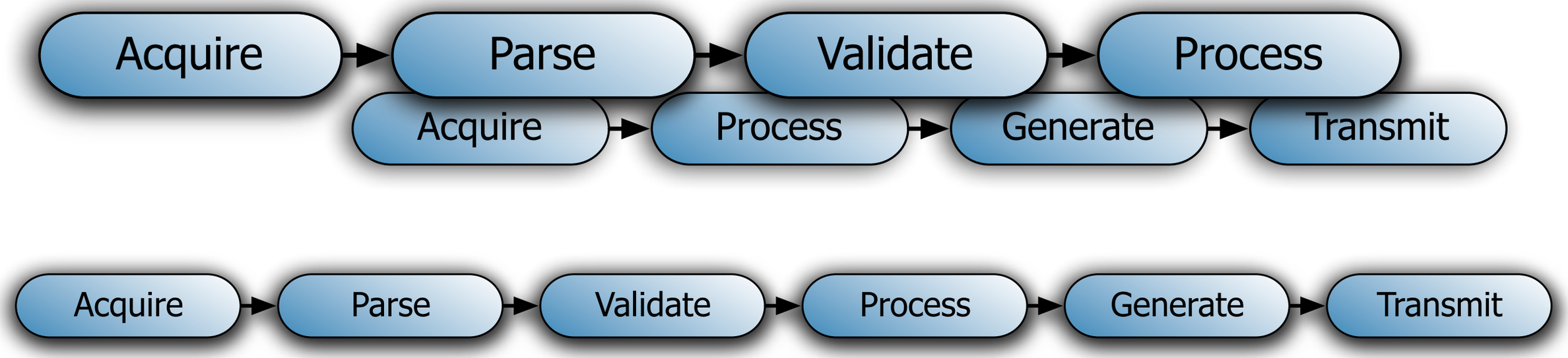


- Offload the problem where possible
  - Don't write your own socket-handling code...
- Solve each problem once
  - Consistency minimises development time and makes it easier to hand over to the Ops team
- Simplicity
  - EI projects have a habit of generating their own complexity
  - Ensure additional complexity is really needed before you add it

Defining the Flow

Incept5

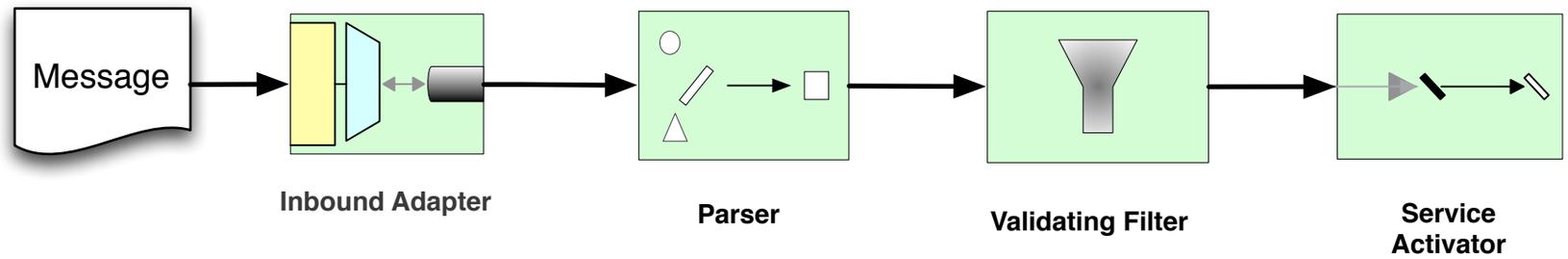
Starter EI Patterns

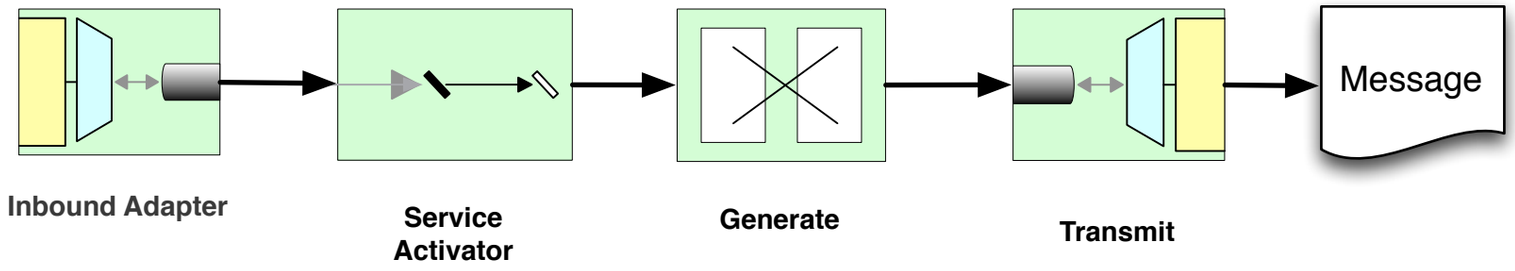


- We see these patterns repeatedly in every EI project
  - And usually multiple times in each project

EIP & Gregorgrams

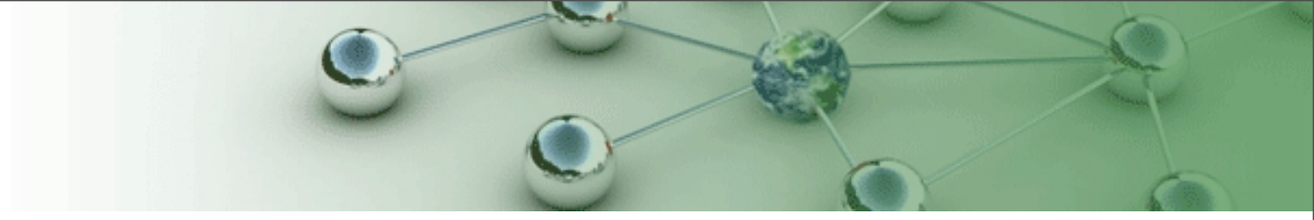
- These patterns are so well known the have been standardised.
  - The common blocks present in all EI projects





- Why should you structure your project this way?
  - It's easier to understand
    - These patterns are well-known and naturally describe common concerns
  - We can leverage tools to help us build them...
  - Promotes re-use

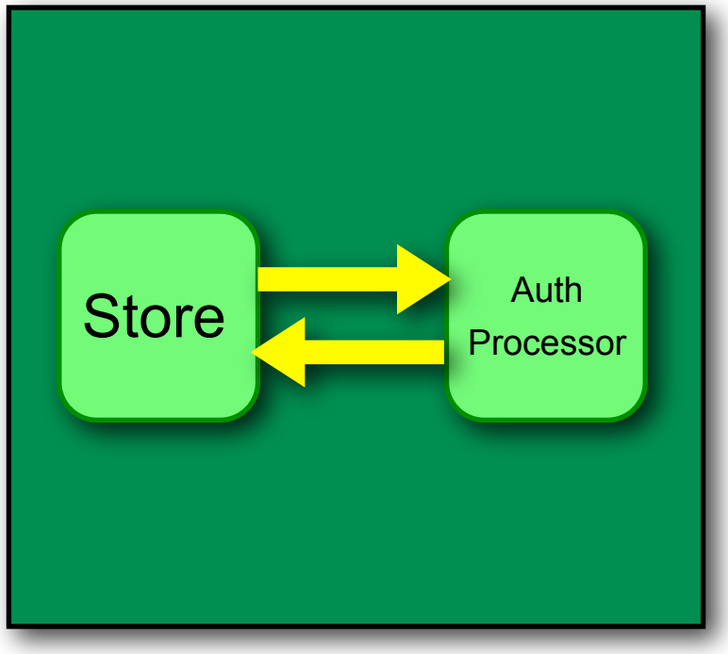
ESBs



- Enterprise Service Buses
  - As lightweight as you want them to be
- Implement the patterns we've just seen
  - Spring Integration
  - Mule
  - Camel / Fuse
  - Synapse
  - ...
- Even better, most contain a wide range of transport adapters
  - File, (S)FTP, JMS, AMQP, SMTP, JDBC, TCP, Web Services, HTTP, ...

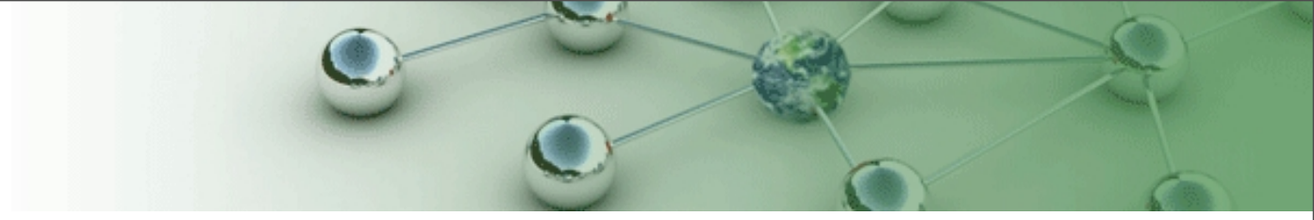
A Simple Example

- The Challenge -
  - Run A & B on separate machines
  - ...without making any changes to either class



Disclaimer - just because you can doesn't mean you should!

Details



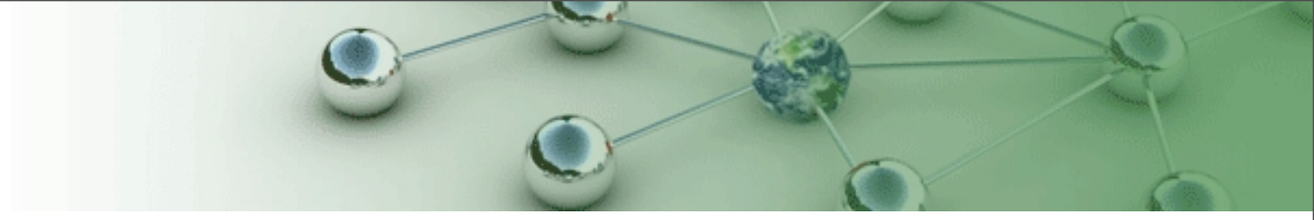
- The AuthProcessor implements a service interface

```
public interface AuthorisationService {

    /*
     * processAuthRequest
     * Attempt to validate and approve a request
     */
    public abstract AuthResponse processAuthRequest(AuthRequest req);

    ...
```

- The Store makes a call to the AuthorisationService

```
public class Store {

    private AuthorisationService server;

    public String authorisePurchase(...) {
        AuthRequest req = ...
        AuthResponse resp = server.processAuthRequest(req);

        return resp.getCode() == ResponseCode.OK? req.getId() : null;
    }
```

C24 Details

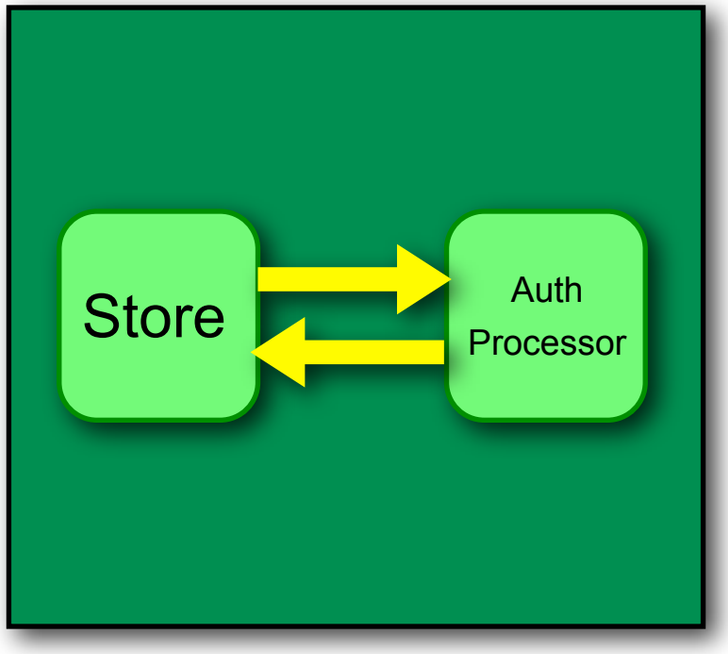


- Spring wires them together

```
<beans:bean id="authProcessor"
    class="com.incept5.rabbitmq.server.AuthorisationProcessor"/>

<beans:bean id="client" class="com.incept5.rabbitmq.client.Store">
    <beans:property name="server" ref="authService"/>
</beans:bean>
```

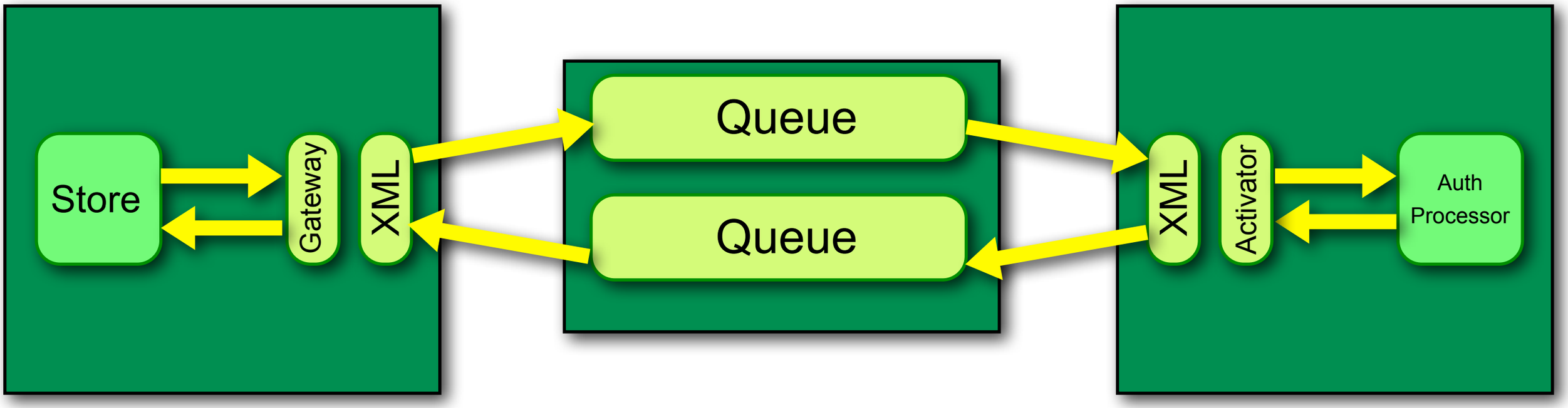
A Simple Example

- The Challenge -
  - Run A & B on separate machines
  - ...without making any changes to either class

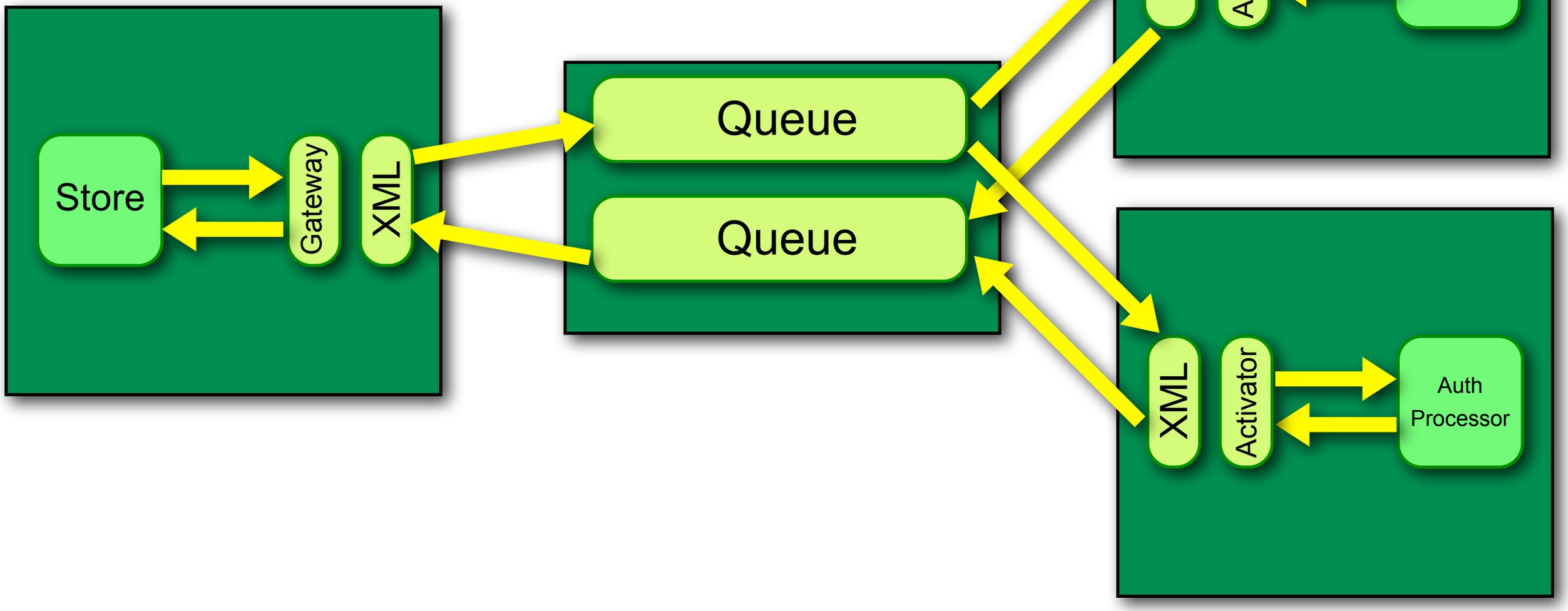


Disclaimer - just because you can doesn't mean you should!

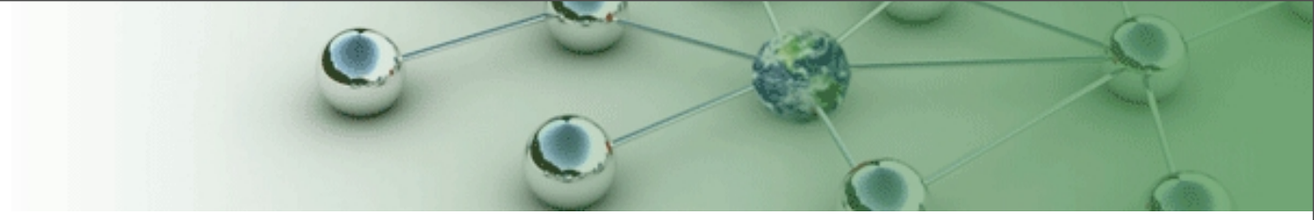
No-Code Remoting



No-Code HA



Client Config



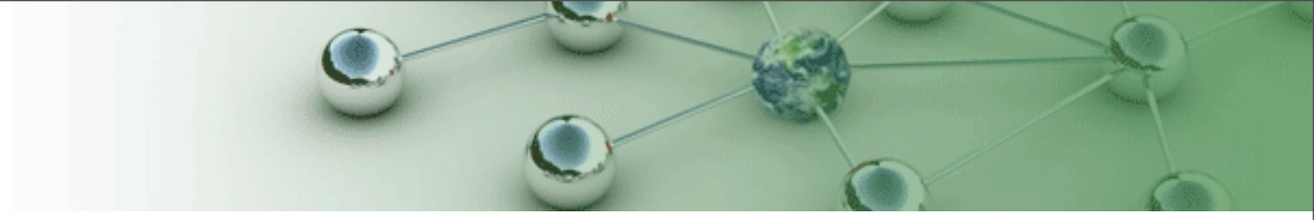
- Proxy the service interface

```
<gateway id="authService"
         service-interface="com.incept5.rabbitmq.model.AuthorisationService"
         default-request-channel="client.request.java"
         default-reply-channel="client.response.java"/>
```

- Convert the AuthRequest to an XML wire format

```
<!-- Request Handling -->
<chain input-channel="client.request.java" output-channel="client.request.xml">
    <int-c24:transformer
        transform-class="com.incept5.c24.transform.AuthRequestToXmlTransform" />
    <int-c24:marshalling-transformer sink-factory="xmlSinkFactory" output-type="STRING"/>
</chain>
```

Client Config



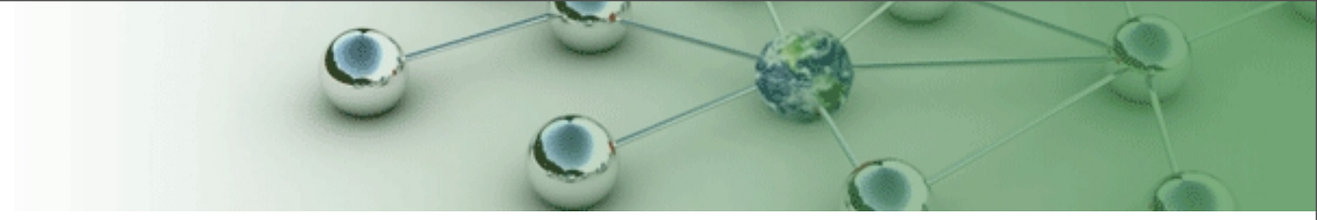
- Send the message to the queue and wait for a response

```
<amqp:outbound-gateway
    request-channel="client.request.xml" reply-channel="client.response.xml"
    exchange-name="PurchaseRequests" routing-key="Europe" amqp-template="amqpTemplate"/>
```

- Transform the response XML to a Java object

```
<!-- Response Handling -->
<chain input-channel="client.response.xml" output-channel="client.response.java">
    <int-c24:unmarshalling-transformer
        source-factory-ref="xmlSourceFactory" model-ref="XmlResponseElement"/>
    <int-c24:transformer
        transform-class="com.incept5.c24.transform.XmlToAuthResponseTransform" />
</chain>
```

Server Config



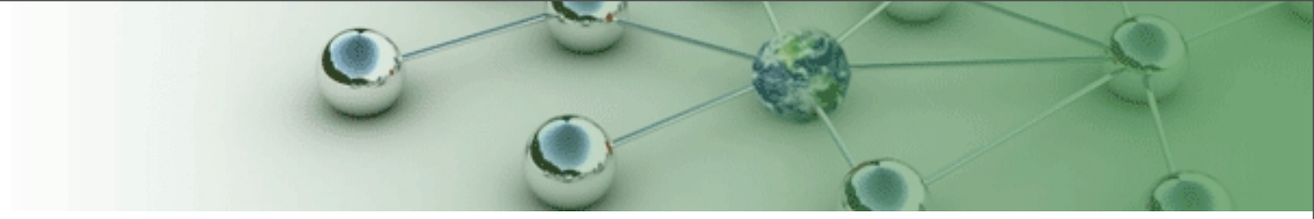
- Receive message from the queue

```
<amqp:inbound-gateway
    request-channel="receive.authrequest" reply-channel="reply.authresponse"
    queue-names="Europe” connection-factory="connectionFactory"
    error-channel="error.authrequest" channel-transacted="false"/>
```

- Parse XML and convert an AuthRequest

```
<!-- Request Handling -->
chain input-channel="receive.authrequest" output-channel="process.authrequest">
    <int-c24:unmarshalling-transformer
        source-factory-ref="xmlSourceFactory" model-ref="XmlRequestElement"/>
    <int-c24:transformer
        transform-class="com.incept5.c24.transform.XmlToAuthRequestTransform" />
</chain>
```

Server Config



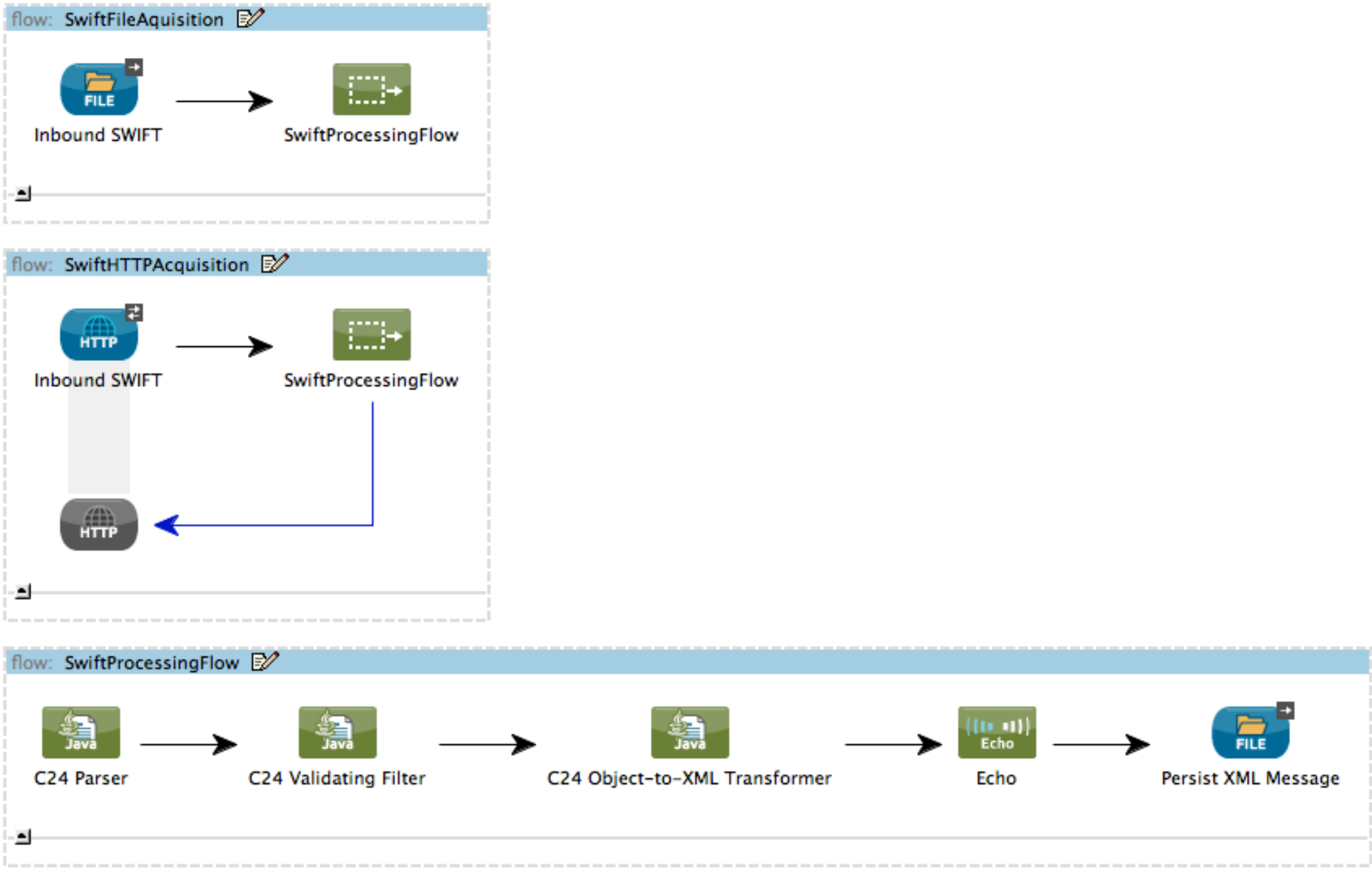
- Invoke the method on the AuthProcessor

```
<service-activator
    input-channel="process.authrequest" output-channel="process.authresponse"
    ref="authProcessor" method="processAuthRequest"/>
```

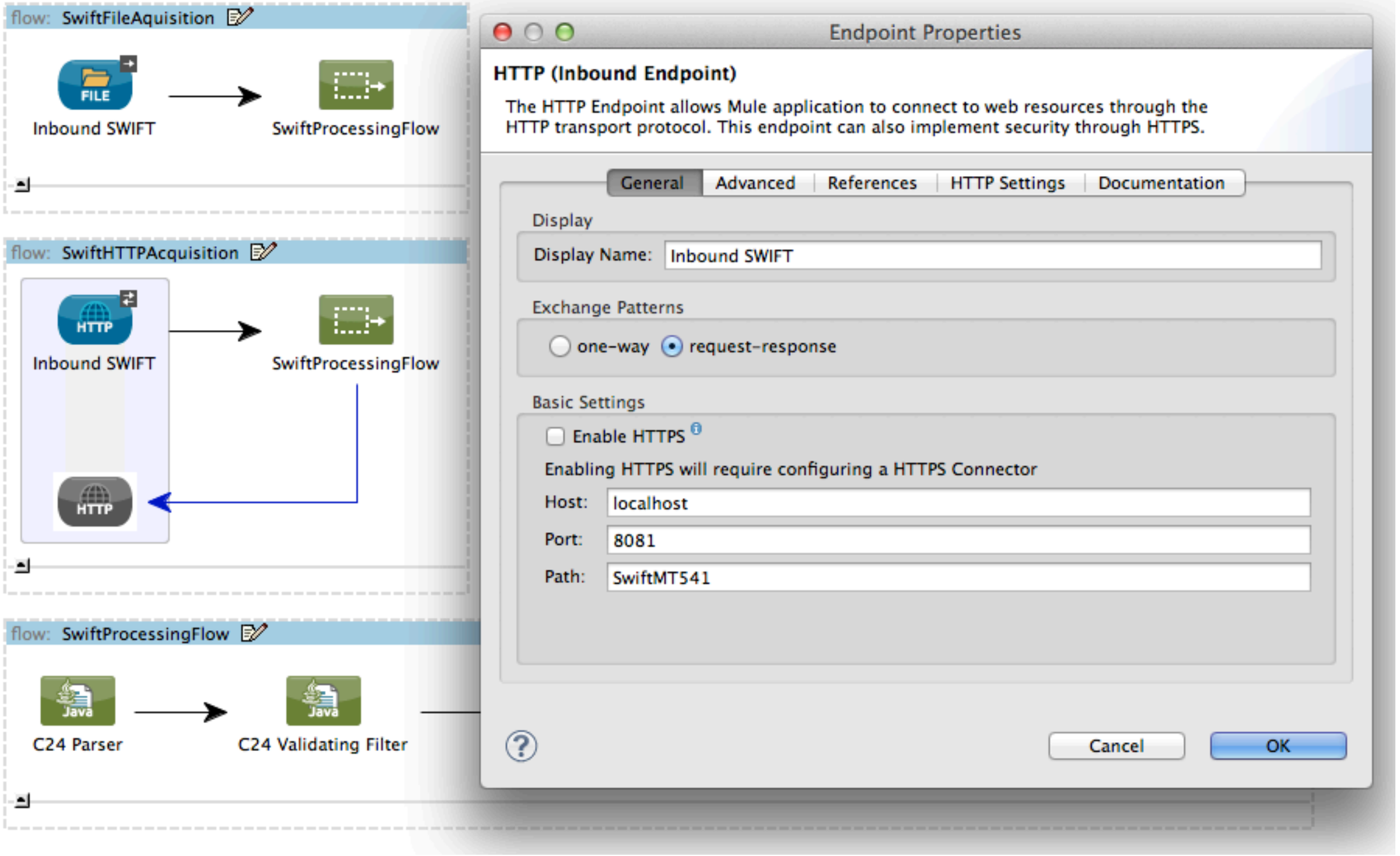
- Convert the AuthResponse to an XML wire format

```
<!-- Response Handling -->
<chain input-channel="process.authresponse" output-channel="reply.authresponse">
    <int-c24:transformer
        transform-class="com.incept5.c24.transform.AuthResponseToXmlTransform" />
    <int-c24:marshalling-transformer sink-factory="xmlSinkFactory" output-type="STRING"/>
</chain>
```

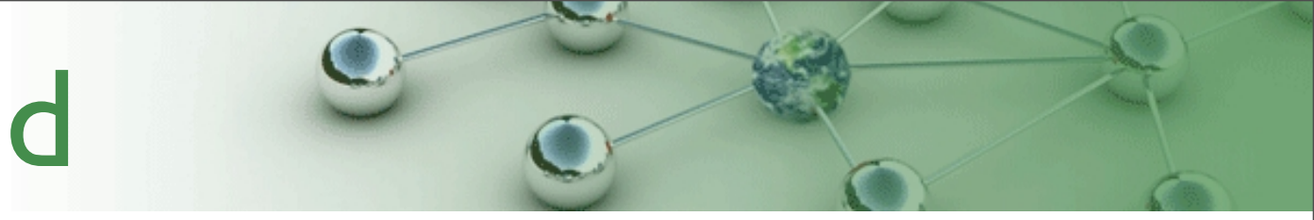
Mule Studio



Configuration



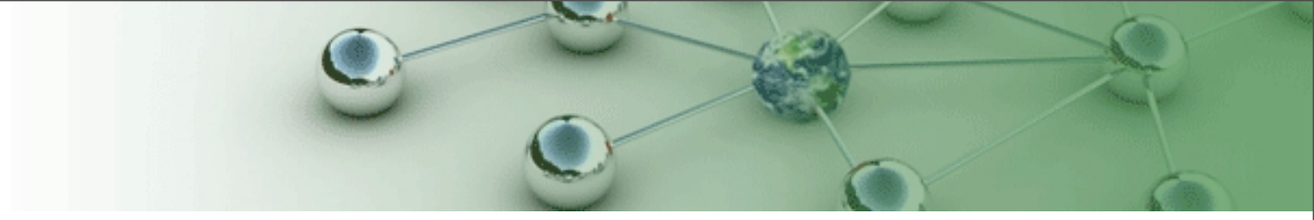
Under the Hood



```
<mule xmlns="...">

    <flow name="SwiftFileAquisition" doc:name="SwiftFileAquisition">
        <file:inbound-endpoint path="/tmp/mule/in" doc:name="Inbound SWIFT">
            <file:filename-regex-filter pattern=".*.dat" caseSensitive="true"/>
        </file:inbound-endpoint>
        <flow-ref name="SwiftProcessingFlow" doc:name="SwiftProcessingFlow"/>
    </flow>

    <flow name="SwiftProcessingFlow" doc:name="SwiftProcessingFlow">
        <component doc:name="C24 Parser">
            <singleton-object class="biz.c24.io.mule.C24Parser">
                <property key="element" value="biz.c24.io.swift2012.MT541Element"/>
                <property key="encoding" value="UTF-8"/>
            </singleton-object>
        </component>

    ...
</mule>
```

Other wiring languages are available... Java, Scala, ...

SI Scala DSL



- Acquire from a file

```
file.poll("directory").atFixedRate(1000) -->

handle {m: Message[File] => println(m.getPayload().getCanonicalPath()); m} -->
```

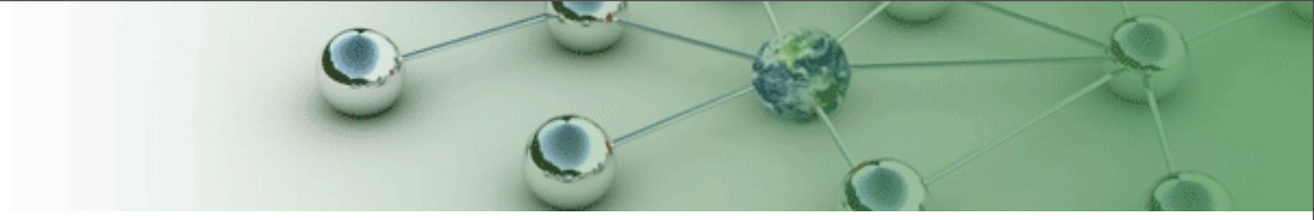
- Parse, validate & filter

```
transform.apply {m: Message[File] => parser.transform(m)} -->

transform.apply {m: Message[ComplexDataObject] => validator.transform(m)} -->

filter.apply {m: Message[ComplexDataObject] =>
    m.getHeaders.containsKey("c24_valid")}.
      additionalAttributes(exceptionOnRejection=true) -->
```

Reading the Message

Incept5

Parsing



- Parse & validate as soon as possible
  - Reduces time spent processing invalid messages
  - The earlier we spot a problem, the easier to identify the source
- Most message formats are structurally simple
  - Writing a parser isn't hard...
- ...but there can be hundreds of messages
  - Each with its own nuances
  - Correct structure of a message is often content dependent
- Standards are written by people who don't have to implement them
  - C10: If field 23B contains the code SPRI, field 56a must not be present (Error code(s): E16). If field 23B contains one of the codes SSTD or SPAY, field 56a may be used with either option A or option C. If option C is used, it must contain a clearing code (Error code(s):E17)

Receipt CSV

51482eba-8d90-4f97-a2f0-633eeec56162,4810,2012-02-29T00:56:41 ← Receipt Header

51482eba-8d90-4f97-a2f0-633eeec56162,286,28,111.72
51482eba-8d90-4f97-a2f0-633eeec56162,321,20,59.00
51482eba-8d90-4f97-a2f0-633eeec56162,329,33,49.5
51482eba-8d90-4f97-a2f0-633eeec56162,2,15,34.35
51482eba-8d90-4f97-a2f0-633eeec56162,203,3,19.5

← Receipt Items

a23107b4-4a2d-41dd-a81d-8e78d0821cc7,11710,2012-02-29T00:27:33
a23107b4-4a2d-41dd-a81d-8e78d0821cc7,286,14,55.86
a23107b4-4a2d-41dd-a81d-8e78d0821cc7,336,9,33.21
a23107b4-4a2d-41dd-a81d-8e78d0821cc7,354,3,8.61
a23107b4-4a2d-41dd-a81d-8e78d0821cc7,320,7,24.43
a23107b4-4a2d-41dd-a81d-8e78d0821cc7,333,2,3.98
a23107b4-4a2d-41dd-a81d-8e78d0821cc7,2,10,22.90
a23107b4-4a2d-41dd-a81d-8e78d0821cc7,174,2,9.98

Import into iO

Steps

1. Import File
2. Target Directory
3. Profiles
4. Model Name & Target Namespace
5. **Record Types**
6. ReceiptHeader
7. ReceiptItem

Record Types

C24 Integration Objects has analyzed your data and determined that it is composed of the following types of record. If this is correct, click 'Next' otherwise select the record types that you wish to change and modify their properties.
Setting tick in header column updates column names of the next record type from values of the curent record.

Name	Sample Data	Cardinality	Included	Header	Type
ReceiptHeader	79951b37-7af4-4e6a-a05b-a25fd99746c9,2204,2012-02-29T00:41:05	1..1	☑	☐	Delimited
ReceiptItem	79951b37-7af4-4e6a-a05b-a25fd99746c9,258,4,23.48	1..*	☑	☐	Delimited

Advanced | Merge Selected | Split Selected... | Reset Selected

< Back | Next > | Finish | Cancel

Header

Steps

1. Import File
2. Target Directory
3. Profiles
4. Model Name & Target Namespace
5. Record Types
6. **ReceiptHeader**
7. ReceiptItem

ReceiptHeader

This screen lets you specify the syntax properties associated with each type of record found in your sample data

Delimiters

Delimiters appear for empty fields

Delimiter , 2c

Preview – Column Data Types

ReceiptId (String)	CustomerId (long)	DateTime (Generic date)
79951b37-7af4-4e6a-a05b-a25fd99746c9	2204	2012-02-29T00:41:05

Selected Column Name: DateTime

Selected Column Data Type: Generic date

Advanced Syntax Options

Advanced

< Back | Next > | Finish | Cancel

Item

Steps

1. Import File
2. Target Directory
3. Profiles
4. Model Name & Target Namespace
5. Record Types
6. ReceiptHeader
7. **ReceiptItem**

ReceiptItem

This screen lets you specify the syntax properties associated with each type of record found in your sample data

Delimiters

Delimiters appear for empty fields

Delimiter , 2c

Preview – Column Data Types

ReceiptId (String)	ProductId (long)	Quantity (long)	price (double)
79951b37-7af4-4e6a-a05b-a25fd99746c9	258	4	23.48
79951b37-7af4-4e6a-a05b-a25fd99746c9	242	4	21.6
79951b37-7af4-4e6a-a05b-a25fd99746c9	289	5	17.45
79951b37-7af4-4e6a-a05b-a25fd99746c9	281	1	3.99
79951b37-7af4-4e6a-a05b-a25fd99746c9	288	5	17.45
79951b37-7af4-4e6a-a05b-a25fd99746c9	280	1	3.4
79951b37-7af4-4e6a-a05b-a25fd99746c9	283	2	6.58
79951b37-7af4-4e6a-a05b-a25fd99746c9	338	4	11.96
79951b37-7af4-4e6a-a05b-a25fd99746c9	319	4	5.00
79951b37-7af4-4e6a-a05b-a25fd99746c9	341	4	11.96
79951b37-7af4-4e6a-a05b-a25fd99746c9	320	3	10.47

Selected Column Name: price

Selected Column Data Type: double

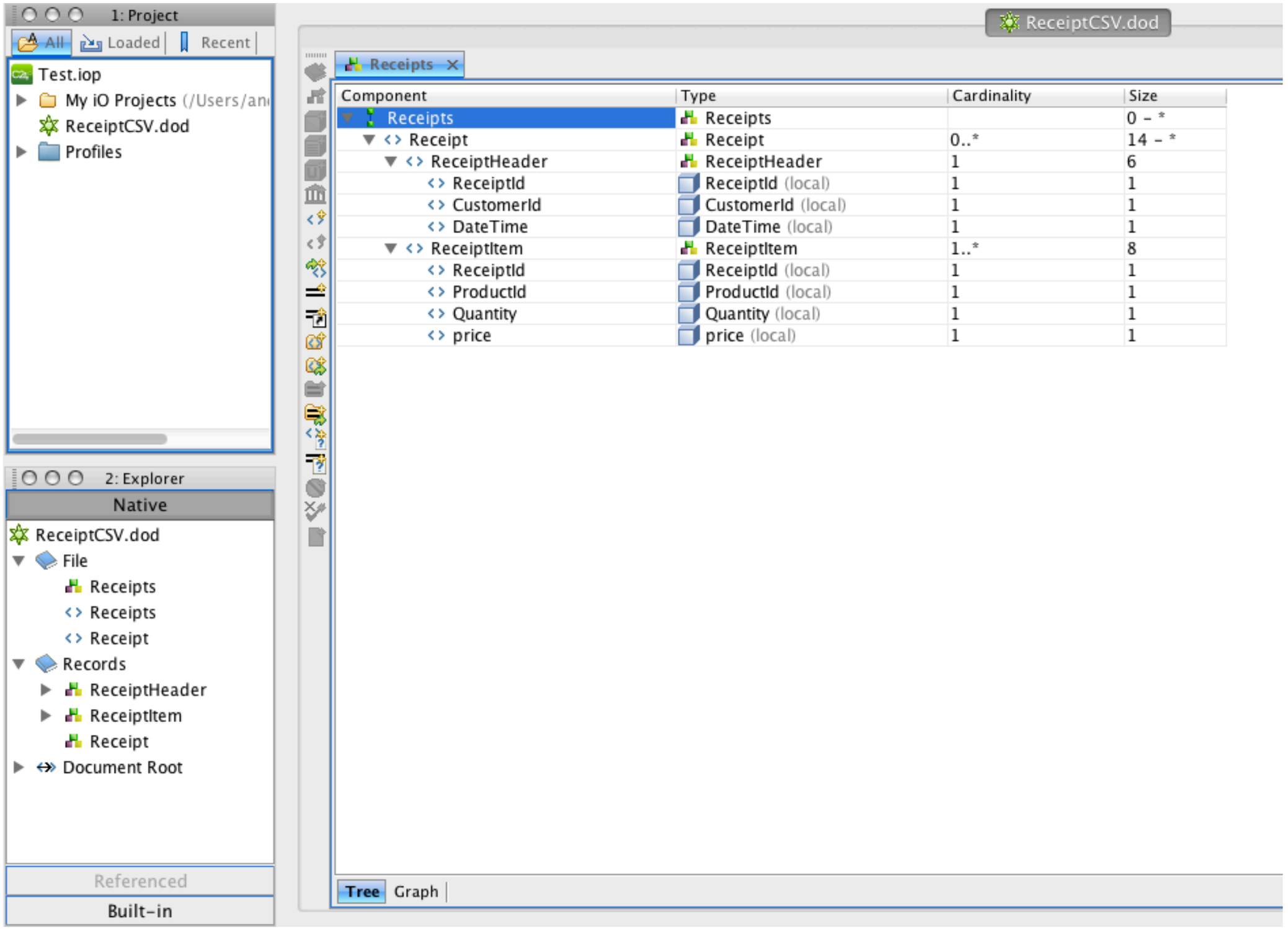
Advanced Syntax Options

Advanced

< Back | Next > | Finish | Cancel

C24 Model

www.C24.biz



C24
www.C24.biz

Generate

- Build in the GUI or via Ant & Maven tasks

```
/**
 * ReceiptItem. <p/>
 * This object is composed of the following <i>elements</i>:
 * <ul>
 * <li><b>ReceiptId</b> of type {@link java.lang.String} (1)</li>
 * <li><b>ProductId</b> of type <code>long</code> (1)</li>
 * <li><b>Quantity</b> of type <code>long</code> (1)</li>
 * <li><b>price</b> of type {@link java.math.BigDecimal} (1)</li>
 * </ul>
 * <br><strong>Produced by C24 Integration Objects (http://www.c24.biz)</br>
 * @author C24 Integration Objects;
 * @see biz.c24.retaildemo.model.csv.ReceiptItemClass
 **/
public class ReceiptItem extends biz.c24.io.api.data.ComplexDataObject {

    /**
     * Gets the value of ProductId (1).
     * @return The value.
     **/
    public long getProductId()
    {
        return this.productId;
    }

    /**
     * Sets the value of ProductId (1).
     * @param value The new value.
     **/
    public void setProductId(long value)
    {
        this.productId = value;
        this.isproductIdSet = true;
    }
```

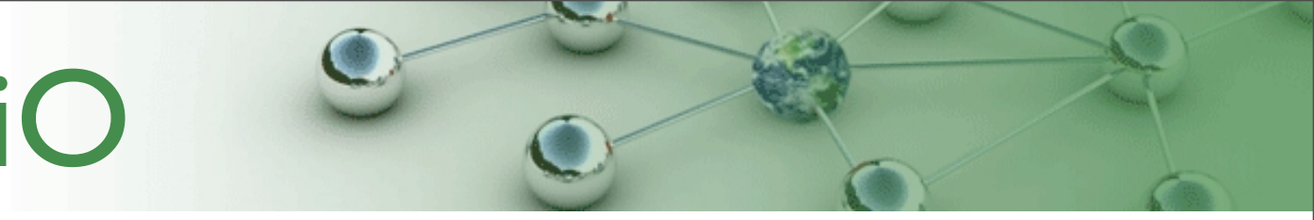
Parsing with C24 iO

- We could have defined the model by
  - Explicitly creating it in Java
  - Describing it in the GUI
  - Importing an XSD, RelaxNG schema, a Java class, a CSV, ...

- Now we can parse them:

```
// Use String or InputStream or Reader or ...
InputStream stream = ...;
ReceiptElement element = ReceiptElement.getInstance();

Source source = element.getModel().source();
source.setEncoding("UTF-8");
source.setInputStream(stream);

Receipt msg = source.readObject(element);
```

Validating with iO



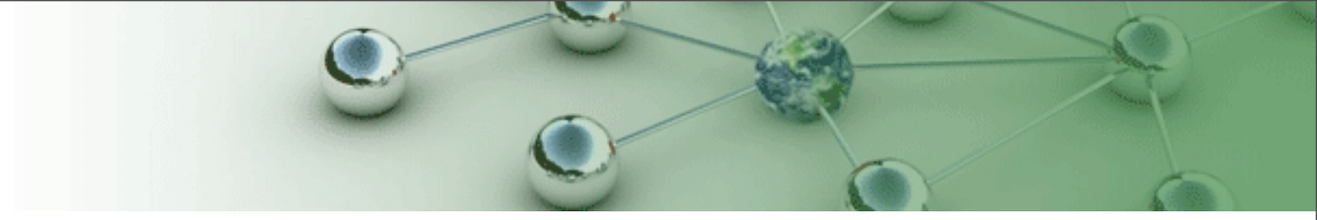
- Fail first - abort on first failure

```
ValidationManager mgr = new ValidationManager();
try {
    mgr.validateByException(msg);
    // It's valid, continue processing
    ...
} catch(ValidationException ex) {
    // Message is invalid. Hand off to error processing
    ...
}
```

- Or capture all errors within the message

```
ValidationManager mgr = new ValidationManager();
ValidationEventCollector vec = new ValidationEventCollector();
mgr.addValidationListener(vec);

if(mgr.validateByEvents(msg)) {
    // It's valid, continue processing
    ...
} else {
    // Message is invalid. Hand off to error processing
    vec.getFailEvents()...;
}
```

Processing



- Resulting object is a Java Bean with getters and setters corresponding to the names in your model

```
msg.getReceiptHeader().getDateTime();

msg.getReceiptItems().length;
```

- Equally we query the object via XPath

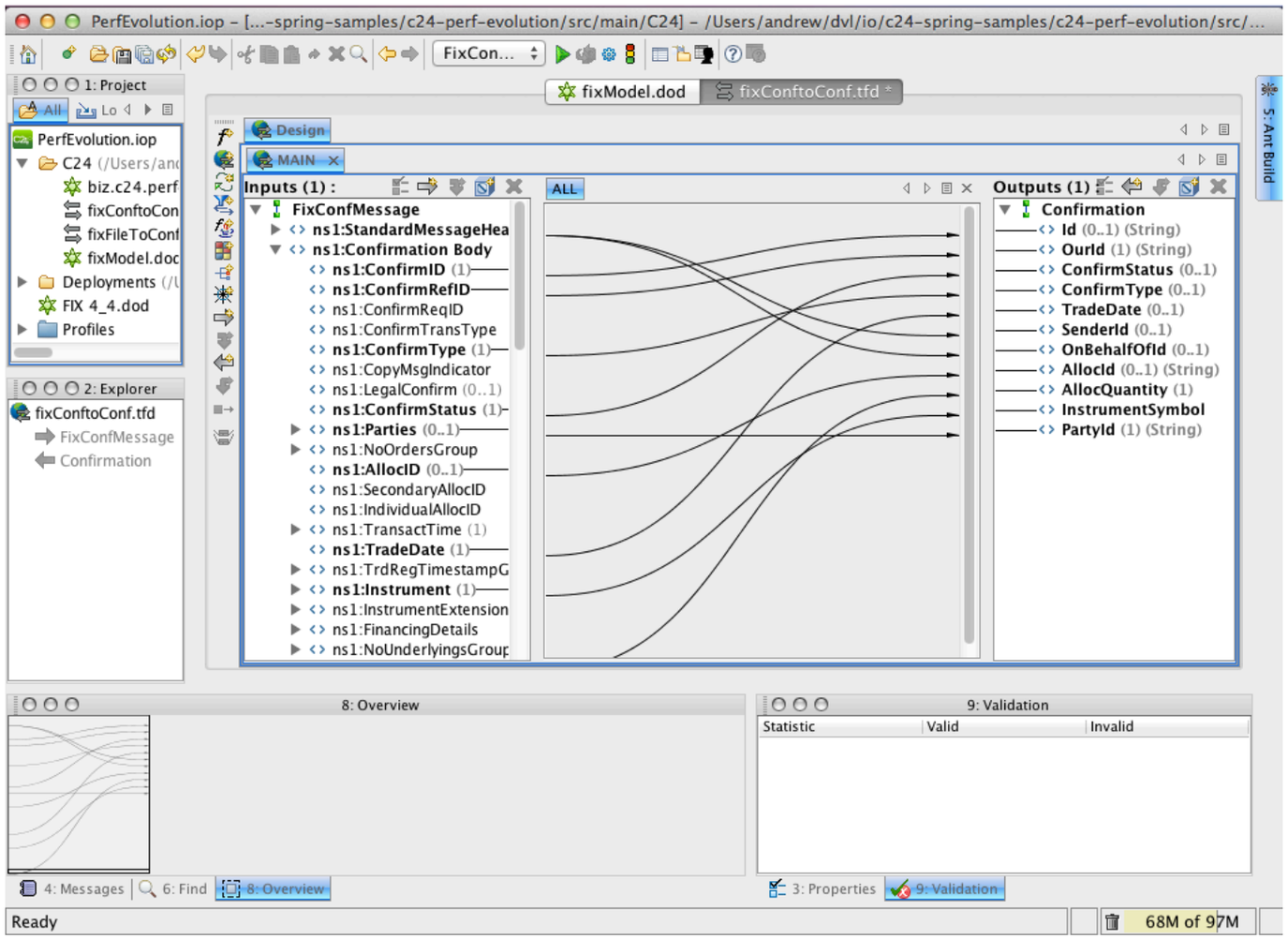
```
IOXPath xPath = IOXPathFactory.getInstance("/ReceiptHeader/ReceiptID");
String id = xPath.getString(msg);
```

  - Results are strongly typed and we can search for individual results or anything matching the criteria

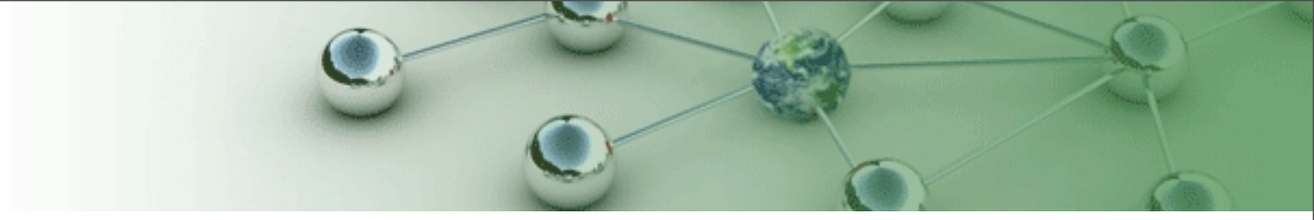
Normalising

- Receiving the same message in different formats?
  - Normalise post-validation to reduce downstream duplication

- iO allows you to define transforms directly in Java, or it can do it for you



Plan Ahead



- Try to get a real set of messages / a real system to develop against
  - Simulators and handcrafted test messages usually lack the nuances of the real system

- Prepare for standards evolution
  - Ensure you have test systems and development time in place
  - ...or license the messages from someone else who will do the work for you

Preparing for Failure

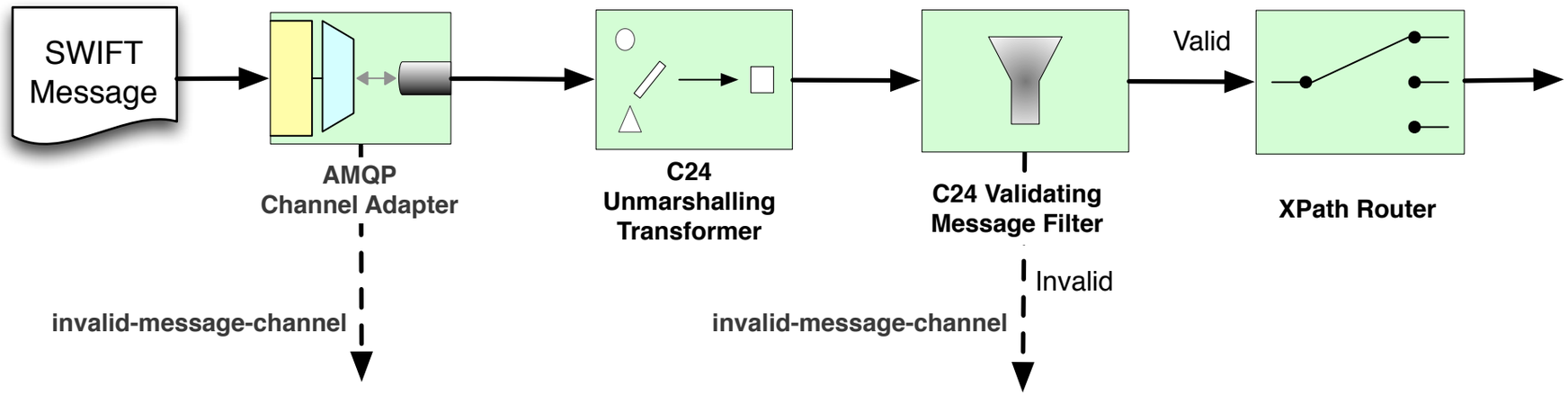
Incept5

Error Handling

- What happens if we encounter an unexpected processing failure?
  - If it's a request-response process the caller might get the exception back
  - The message could be silently lost

- ESBs are great for catching errors and redirecting to an error queue
  - Ensure source message in original format is available
  - Include as much context as possible

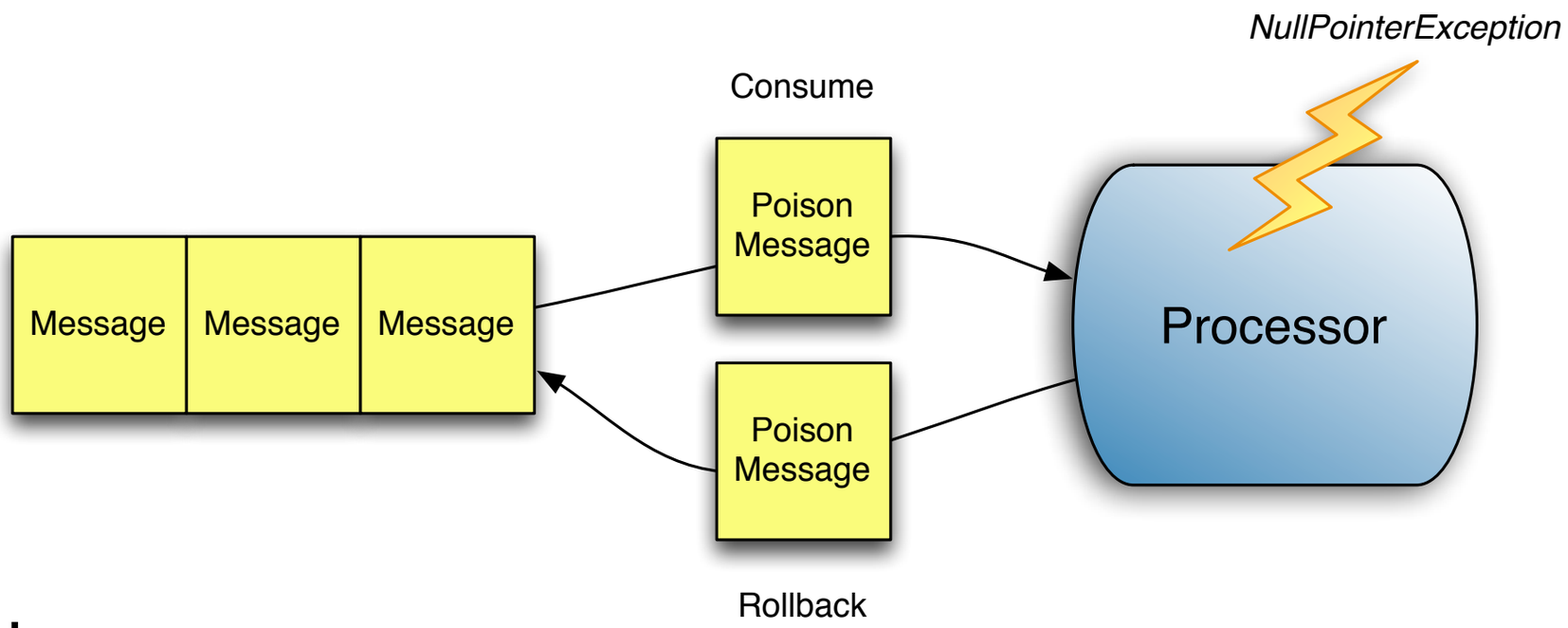
Spring Integration




```
<!-- Consume messages from the queue -->
<amqp:inbound-channel-adapter channel="parse-message-channel"
                              queue-names="SwiftMessages"
                              connection-factory="rabbitConnectionFactory"
                              error-channel="invalid-message-channel"
                              prefetch-count="100"/>

<chain input-channel="parse-message-channel" output-channel="process-valid-channel">
    <int-c24:unmarshalling-transformer
        model-ref="swiftMessageModel" source-factory-ref="textualSourceFactory"/>
    <filter ref="c24Validator" discard-channel="invalid-message-channel"/>
</chain>

<int-c24:xpath-router
    input-channel="process-valid-541-channel" xpath-statement="//Block1/SeqA/...
```

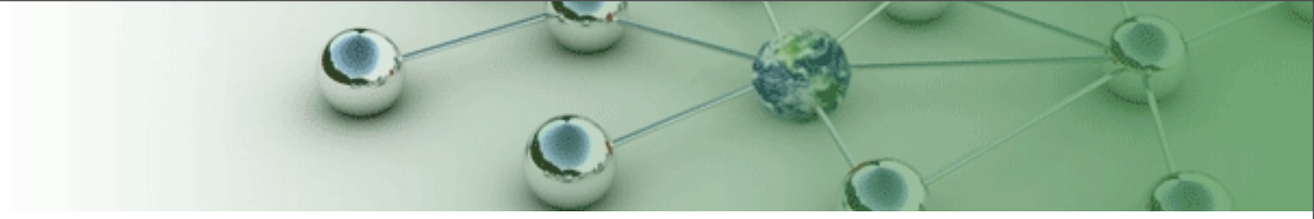
Poison Messages

- Not all poison messages intentionally malicious
  - Anything you fail to handle properly can cause lock-up



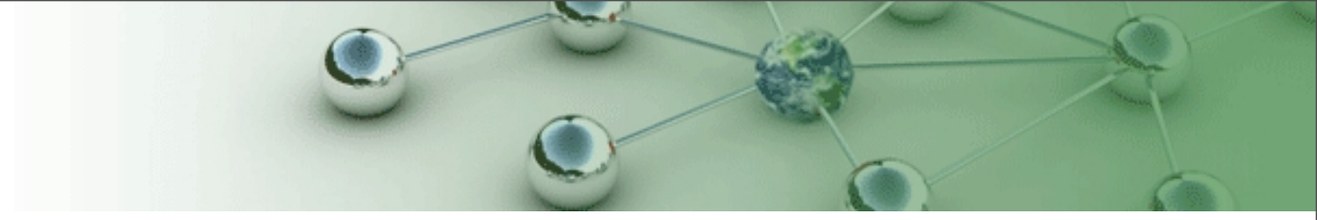
- Distinguish between
  - Transient failure - someone else will successfully process it
  - Permanent failure - message cannot be processed
- Ensure either
  - Anything that causes failure moves message to an error queue and commits the source, or
  - Implement a retry count - once a message has failed N times it goes to the error queue

Security



- Ensure security of sensitive message contents
  - Regulated for some content (e.g. PCI-DSS)
- Message contents can be implicitly persisted to disk
  - Intermediate message brokers might write to disk for reliability or to manage RAM usage
  - OS might swap memory out to disk
- Sensitive data should be encrypted

iO Interception

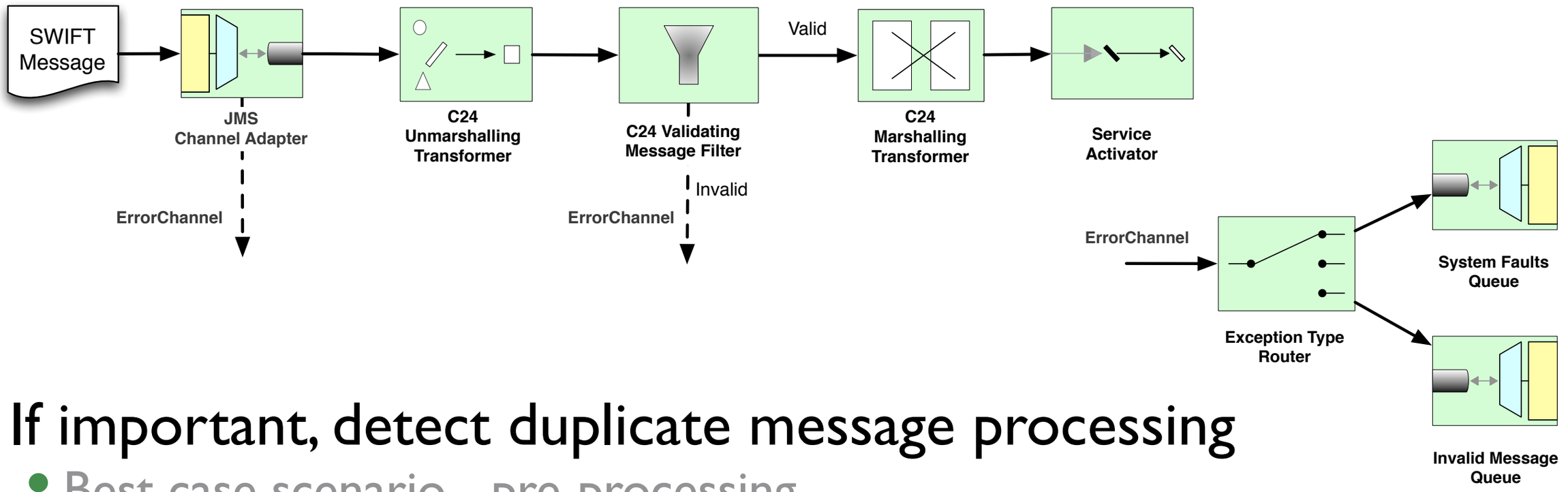


- We can override both parsing and formatting:

```
public class CardPan extends GenericStringDataType {

    @Override
    public Object parseObjectExternal(String str, ParsePosition pp) {
        // We receive PANs encrypted. Decrypt using HSM
        String plaintextPan = hsm.decrypt(str);

        // Handle errors
        ...

        return plaintextPan;

    }

    @Override
    public String formatObjectExternal(Object obj) {
        String pan = obj.toString();

        // We always mask the PAN before displaying it
        String maskedPan = pan.replaceAll("[0-9]", "*");

        return maskedPan;
    }
```

C24 Do...

- Ensure each flow has a catch-all error flow
- Ensure that flow 'commits the transaction' from the source's perspective

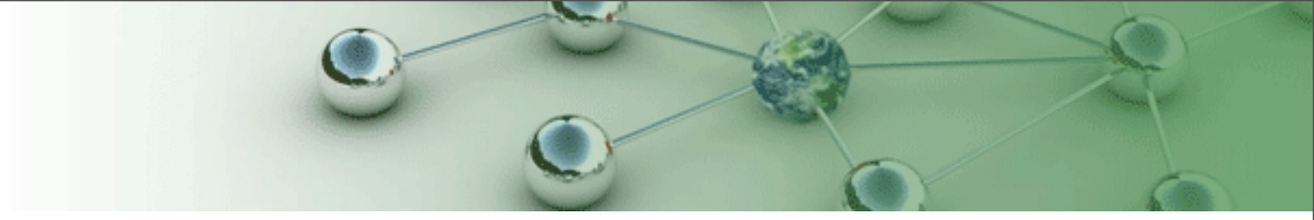


- If important, detect duplicate message processing
  - Best case scenario - pre-processing
  - Aim for idempotent services

Keeping it Running

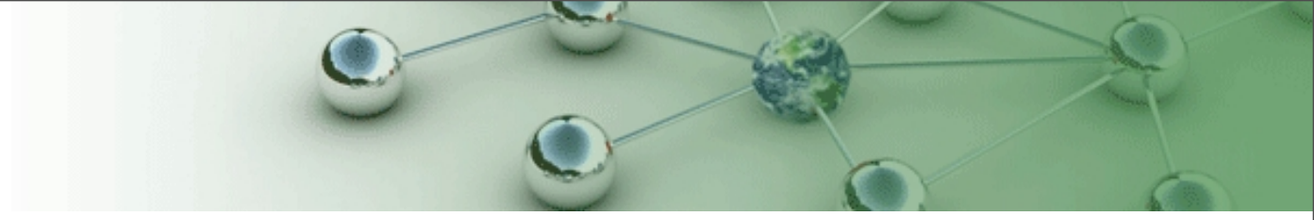
Incept5

SLAs



- Monitoring
  - Ensuring system is performing within tolerance
- Alerting
  - Warn of (impending) violations & problems
- Forecasting
  - Predict future traffic volumes
  - Proactively manage capacity

Monitoring



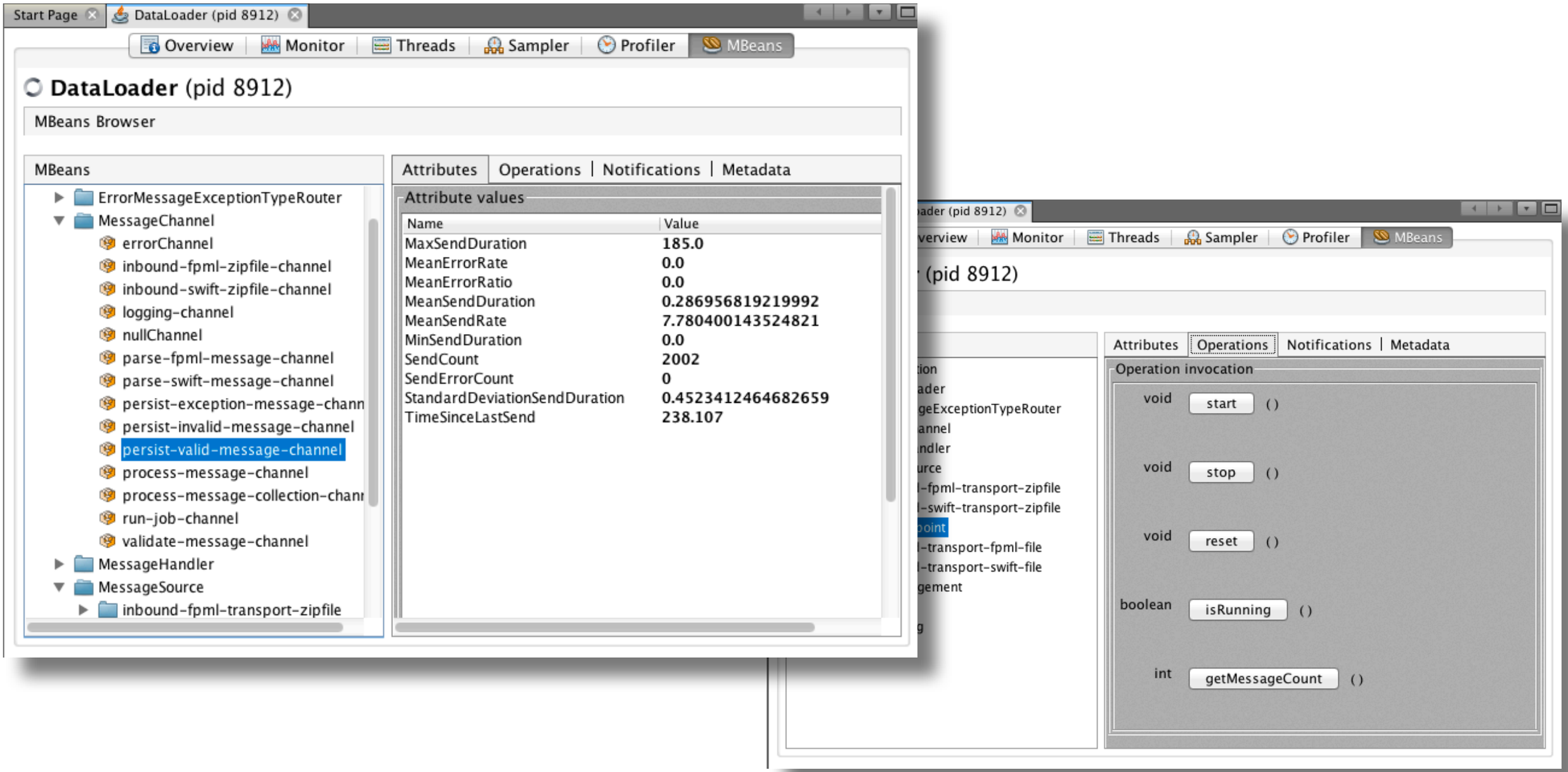
- While the metrics are business-specific, the measurement is often common
  - How many messages are we receiving from X an hour?
  - When did we last receive a message from X?
  - How do today's traffic volumes compare with historical levels?
  - How much capacity do we have left in the system?
  - What is our average processing time per message?

- As our logic is flow-based, it is easy to insert a component at key points to tally message flow
  - Individual message stats can be measured too - enrich the headers with relevant information

- Strive for the same measurements & controls on every flow
  - Consistent systems are easier to operate

Metrics for Free

- Some ESB implementations already contain basic stats and control features



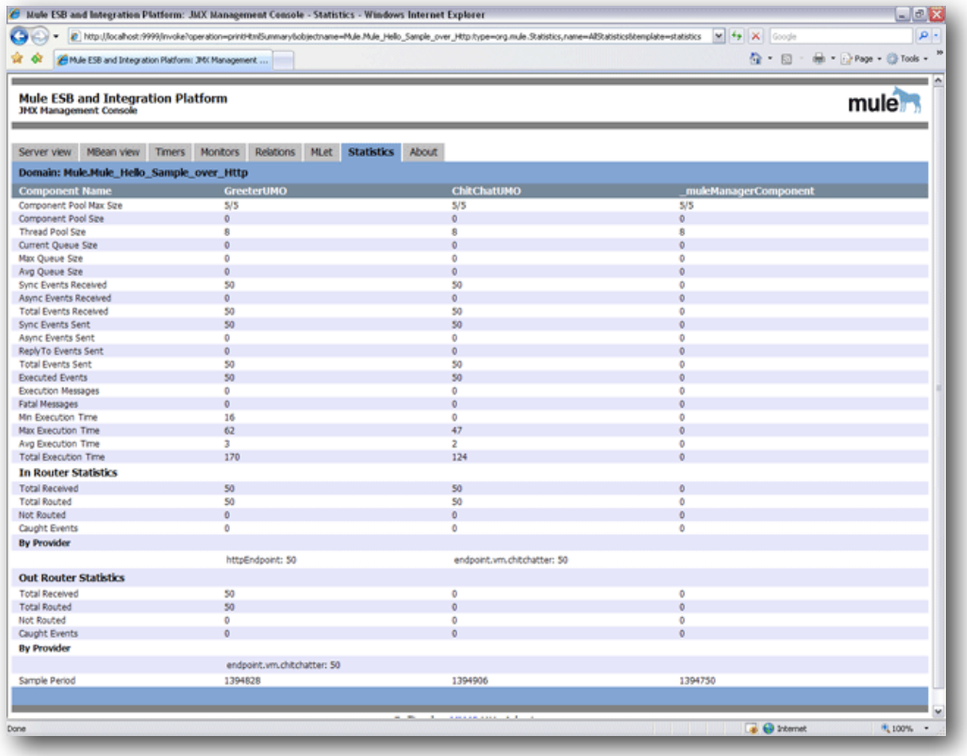
For Example

- Spring Integration

```
<!-- Export our SI components stats over JMX -->
<jmx:mbean-export default-domain="biz.c24.dataloader" server="mbeanServer"/>

<bean id="mbeanServer" class="org.springframework.jmx.support.MBeanServerFactoryBean">
    <property name="locateExistingServerIfPossible" value="true"/>
</bean>
```

- Mule

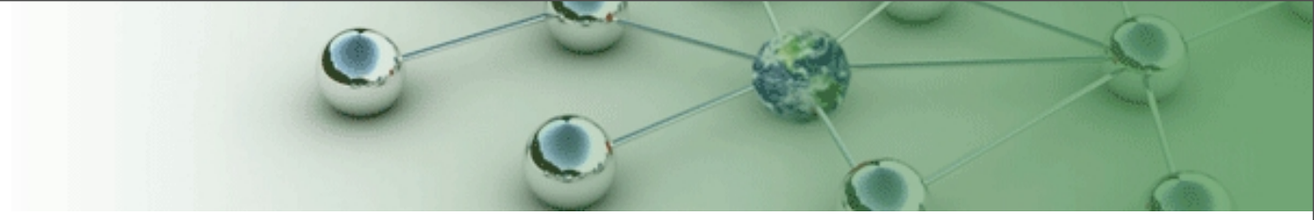
```
<management:jmx-default-config .../>
<management:jmx-server .../>
```



Performance

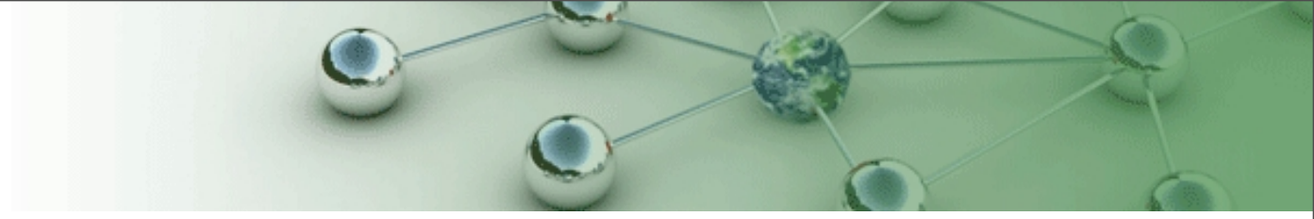
Incept5

Performance



- All flows discussed so far are message-based
  - Visit the C24 stand to talk about batch-processing too...
- Unless we have ordering requirements we can simply cluster horizontally
  - ...or add a thread pool to our inbound adapter
- If we do have detectable sequencing
  - Hold out-of-sequence messages in a (durable) store
  - Server that processes blocking message also processes all stored & unblocked messages

Performance



- Unless you have significant asynchronous waits (e.g. IO) it is generally simpler to use a single thread to process each message
  - Crossing thread boundaries will break transactionality
- Avoid locking & transactions
  - Typically hundreds of times faster
  - Prefer optimistic locking and compensation
- Idempotent services make this easier
  - May be required in a distributed system anyway

C24

www.C24.biz