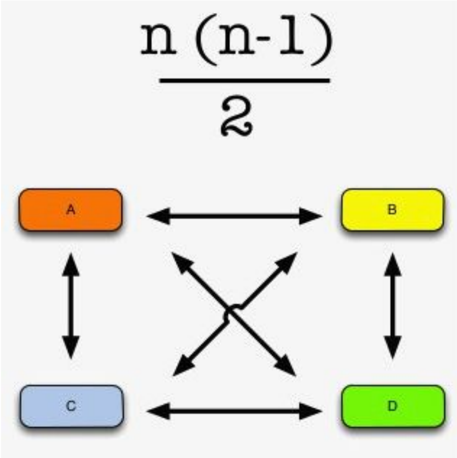
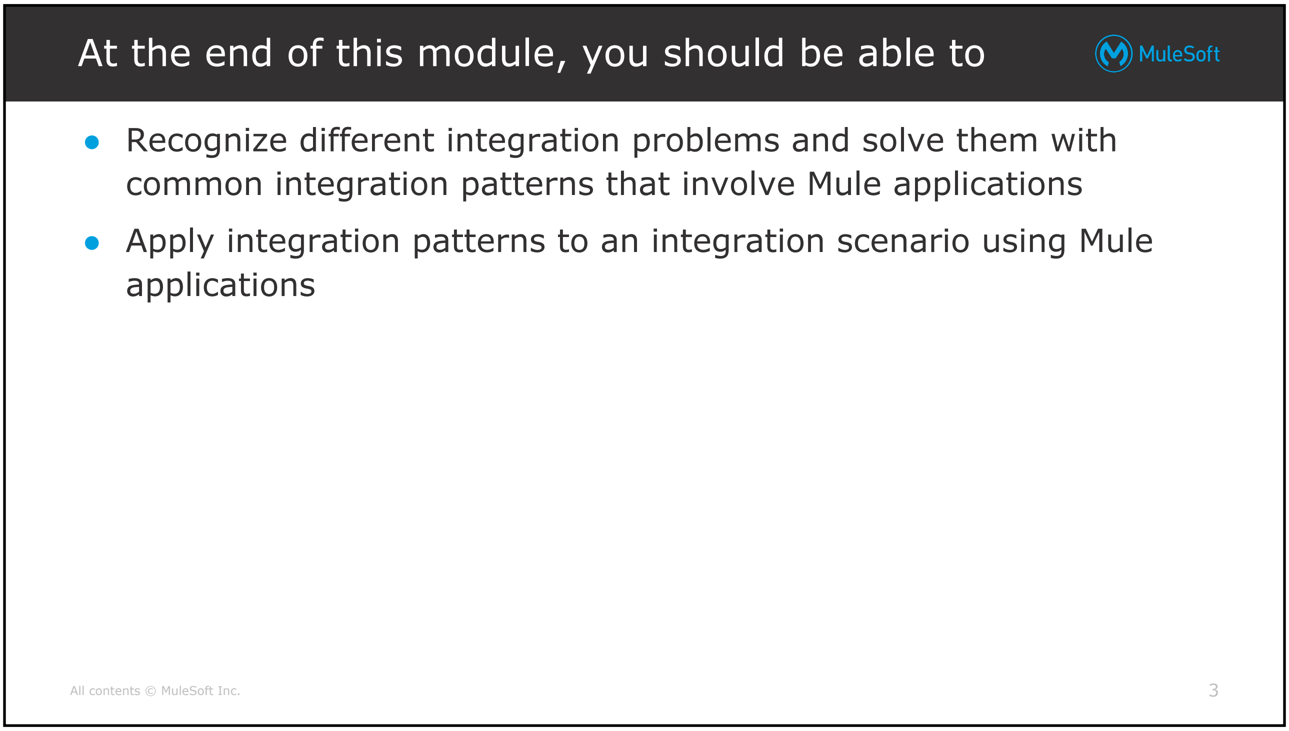
Module 5: Choosing Appropriate Message Transformation and Routing Patterns

Goal





At the end of this module, you should be able to

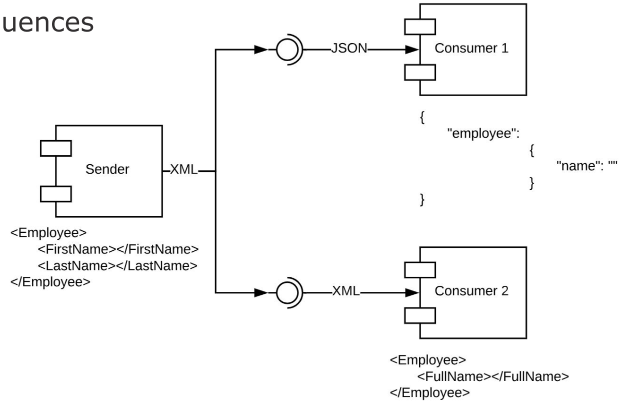
MuleSoft

- Recognize different integration problems and solve them with common integration patterns that involve Mule applications
- Apply integration patterns to an integration scenario using Mule applications

Identifying reusable ways to transform and process events

Why are event transformation and routing patterns needed?

- To identify **reusable ways to transform and process events** between particular enterprise systems
  - Enterprise systems use particular message formats and message schemas
  - Enterprise systems often define business processes that must execute in certain sequences



```
<Employee>
      <FirstName></FirstName>
      <LastName></LastName>
</Employee>
```

```
{
      "employee":
                        {
                              "name": ""
                        }
}
```

```
<Employee>
      <FullName></FullName>
</Employee>
```

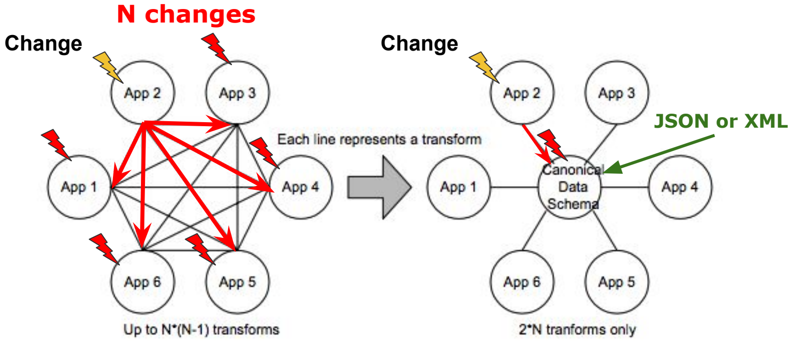
Common message construction patterns implemented in Mule applications

- Common data models
- Message transformation patterns
- Message validation patterns
- Message routing patterns

Converting between data models

Transforming between data models

- With or without a common data model, your Mule applications need to **transform** between different data representations
- MuleSoft recommends using a flexible data representation language to develop and maintain application integrations
  - Should be easy to represent and transform between **XML, JSON, Java POJOs,** and **flat files**

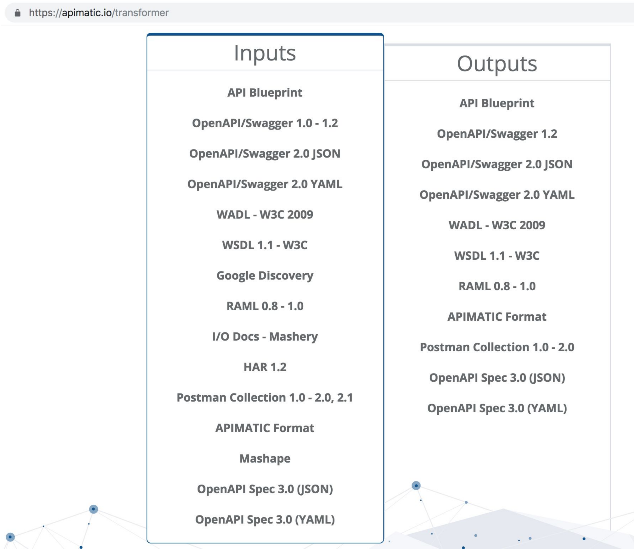


How MuleSoft typically supports modern data types

- Mule application designs and implementations can model data using
  - JSON and XML schema
  - REST API Modeling Language (RAML) data types
  - OpenAPI Specification (OAS) data types
- Both of these schema languages can represent JSON, XML, and other types of objects or flat files
- Anypoint platform supports some interoperability between OAS and RAML
  - Full interoperability is planned in future releases

Converting between popular API specification formats

- There are online tools to convert between various API specification formats
  - https://apimatic.io/
  - This is helpful to convert other formats to RAML/OAS
    - So they can be imported into API designer



https://apimatic.io/transformer

Inputs

API Blueprint
OpenAPI/Swagger 1.0 - 1.2
OpenAPI/Swagger 2.0 JSON
OpenAPI/Swagger 2.0 YAML
WADL - W3C 2009
WSDL 1.1 - W3C
Google Discovery
RAML 0.8 - 1.0
I/O Docs - Mashery
HAR 1.2
Postman Collection 1.0 - 2.0, 2.1
APIMATIC Format
Mashape
OpenAPI Spec 3.0 (JSON)
OpenAPI Spec 3.0 (YAML)

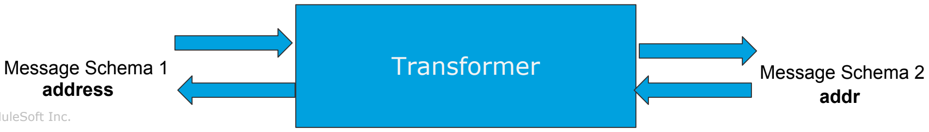
Outputs

API Blueprint
OpenAPI/Swagger 1.2
OpenAPI/Swagger 2.0 JSON
OpenAPI/Swagger 2.0 YAML
WADL - W3C 2009
WSDL 1.1 - W3C
RAML 0.8 - 1.0
APIMATIC Format
Postman Collection 1.0 - 2.0
OpenAPI Spec 3.0 (JSON)
OpenAPI Spec 3.0 (YAML)

Choosing event transformation patterns for Mule applications

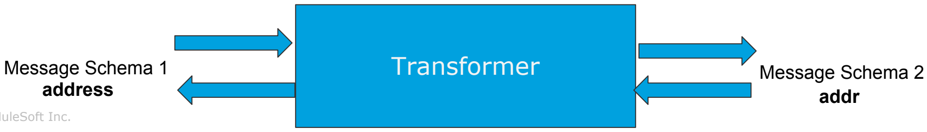
Why event transformation patterns are needed

- Different applications, whether homegrown or from different vendors, can use **different** event **payload** schema to represent even the **same** data
- For example, one application can name a customer attribute **address**, whereas the other may use **addr** to represent the same address.



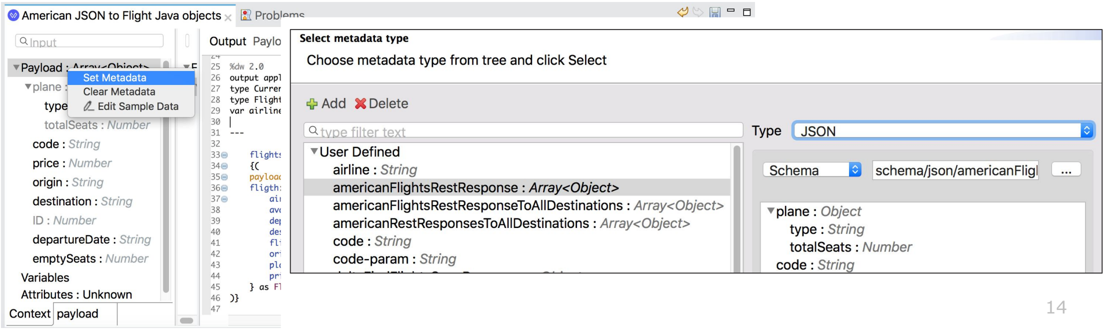
The problem of multiple event transformations

- Schemas used by different systems might contain a **different** number of **fields**
  - And the corresponding fields may differ
  - Such as by having
    - Different field **names** (address vs. addr)
    - Different **types** of value (String vs. Integer)
- To integrate systems, each system's data model must be transformed to the other systems' data model



How **DataSense** makes DataWeave mapping easier

- DataWeave supports **DataSense**
  - When connectors supports DataSense, DataSense can **automatically sense and import metadata** from the connectors to help you visually drag and drop DataWeave mappings between input sources and output targets
  - You can also **import schemas** and **sample documents** to more quickly and easily design your transformations, and assist DataSense



Payload : Array<Object>
plane :
type
totalSeats : Number
code : String
price : Number
origin : String
destination : String
ID : Number
departureDate : String
emptySeats : Number
Variables
Attributes : Unknown

```
%dw 2.0
output appl
type Currer
type Flight
var airline

---

    flights
    {(
    payload
    fligth:
        ai
        av
        de
        de
        fl
        or
        pl
        pr
    } as Fl
)}
```

Select metadata type

Choose metadata type from tree and click Select

Type: JSON

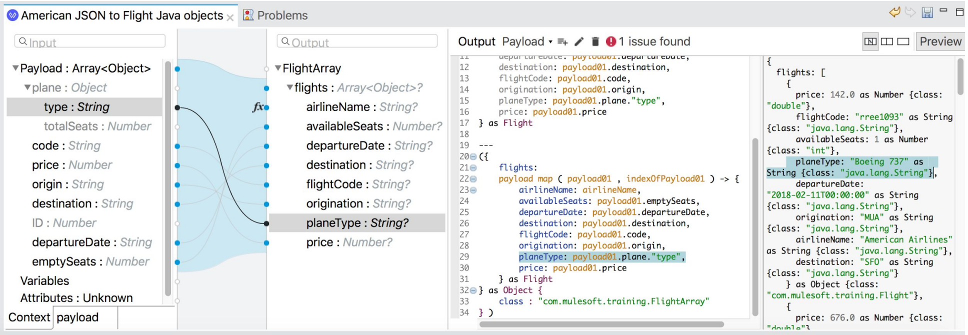
Schema: schema/json/americanFligl

User Defined
airline : String
americanFlightsRestResponse : Array<Object>
americanFlightsRestResponseToAllDestinations : Array<Object>
americanRestResponsesToAllDestinations : Array<Object>
code : String
code-param : String

plane : Object
type : String
totalSeats : Number
code : String

Leveraging DataSense to implement the transformer pattern

- With DataSense, inbound and outbound schemas can be **auto-populated** in the transformation mapping tools
- DataSense assists you at **design time** by providing a live stream of content types while you are coding
  - Provides a **scaffolding** to begin writing mappings
  - But not required, the entire DataWeave code can also be typed out manually



```
        departureDate: payload01.departureDate,
        destination: payload01.destination,
        flightCode: payload01.code,
        origination: payload01.origin,
        planeType: payload01.plane."type",
        price: payload01.price
} as Flight

---
({
    flights:
    payload map ( payload01 , indexOfPayload01 ) -> {
        airlineName: airlineName,
        availableSeats: payload01.emptySeats,
        departureDate: payload01.departureDate,
        destination: payload01.destination,
        flightCode: payload01.code,
        origination: payload01.origin,
        planeType: payload01.plane."type",
        price: payload01.price
    } as Flight
} as Object {
    class : "com.mulesoft.training.FlightArray"
} )
```

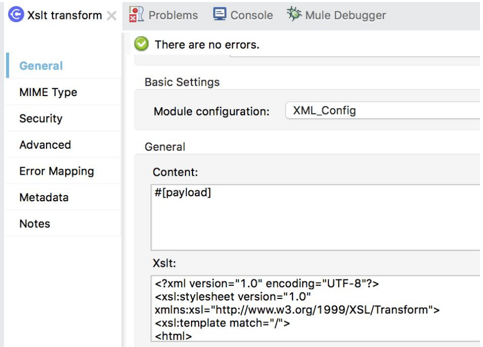
```
{
  flights: [
    {
      price: 142.0 as Number {class:
"double"},
      flightCode: "rree1093" as String
{class: "java.lang.String"},
      availableSeats: 1 as Number
{class: "int"},
      planeType: "Boeing 737" as
String {class: "java.lang.String"},
      departureDate:
"2018-02-11T00:00:00" as String
{class: "java.lang.String"},
      origination: "MUA" as String
{class: "java.lang.String"},
      airlineName: "American Airlines"
as String {class: "java.lang.String"},
      destination: "SFO" as String
{class: "java.lang.String"}
    } as Object {class:
"com.mulesoft.training.Flight"},
    {
      price: 676.0 as Number {class:
"double"},
```

DataWeave supported formats as input or output

Mime Type	Supported Formats
application/csv	CSV
application/dw	DataWeave (for testing a DataWeave expression)
multipart/*	Multipart (Form-Data)
application/flatfile	Flat File, Cobol Copybook, Fixed Width
application/java	Java, Enum Custom Type (for Java)
application/json	JSON
application/octet-stream	
application/xml	XML, CData Custom Type (for XML)
application/xlsx	Excel
application/x-www-form-urlencoded	URL Encoding
text/plain	For plain text.

Using the XML Module's XSLT transformer

- Uses the Extensible Stylesheet Language Transformations (XSLT) language
  - Transforms XML documents to other **XML schemas** or other non-XML formats
  - **XPath** is used to navigate to document structure
- An alternative to using DataWeave transformations



There are no errors.

Basic Settings

Module configuration: XML_Config

General

Content:

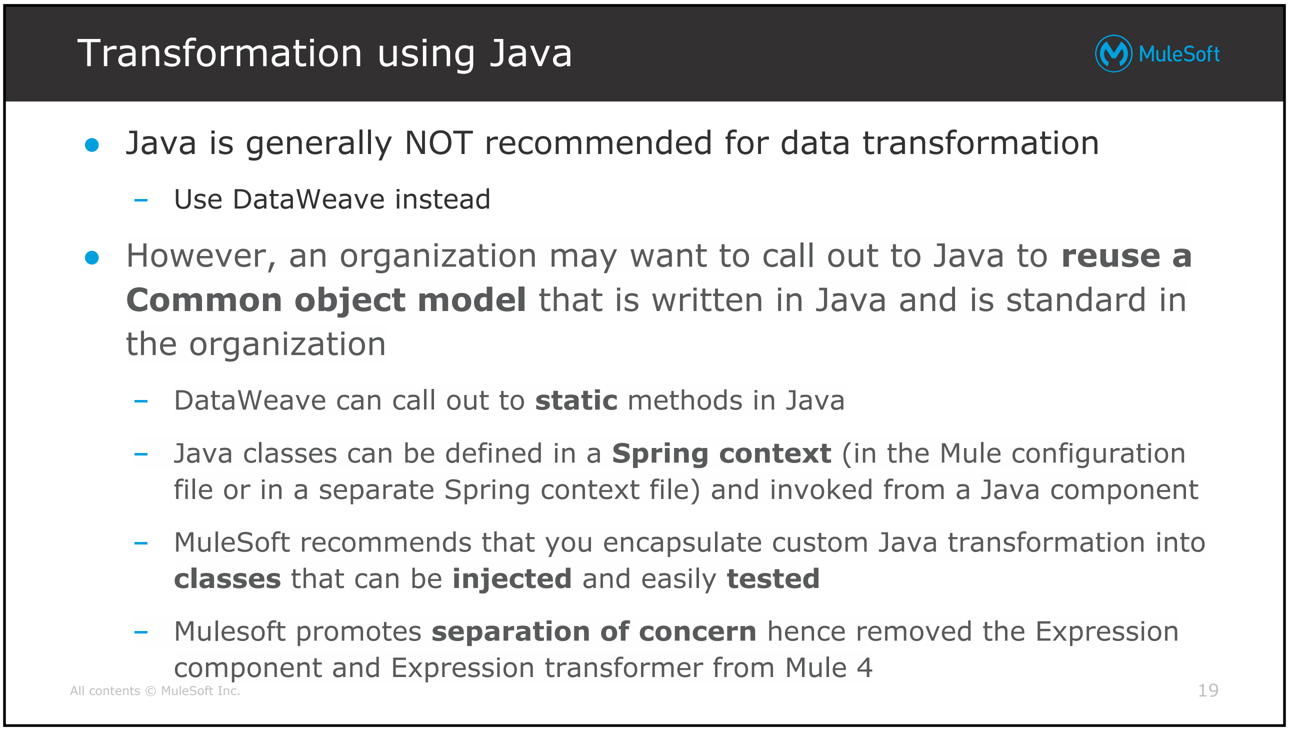
```
#[payload]
```

Xslt:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
```

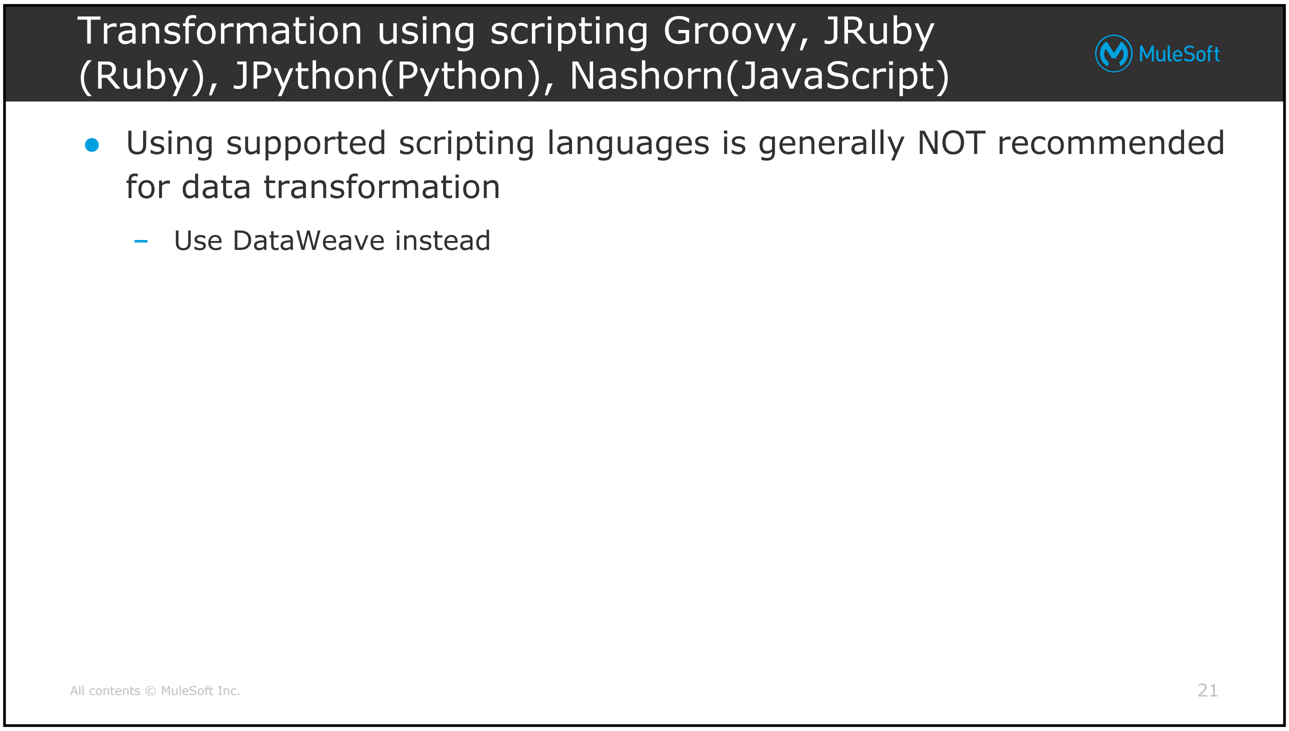
Leveraging other flows and services from within a DataWeave expression

- DataWeave can also access other flows, external services, or Java libraries
- This allows some transformation work to be offloaded from DataWeave to other flows or libraries
  - For example, DataWeave can call out to XSLT transformation flows that use the XML Module
    - Or to a different Java library or script that can handle XSLT transformations



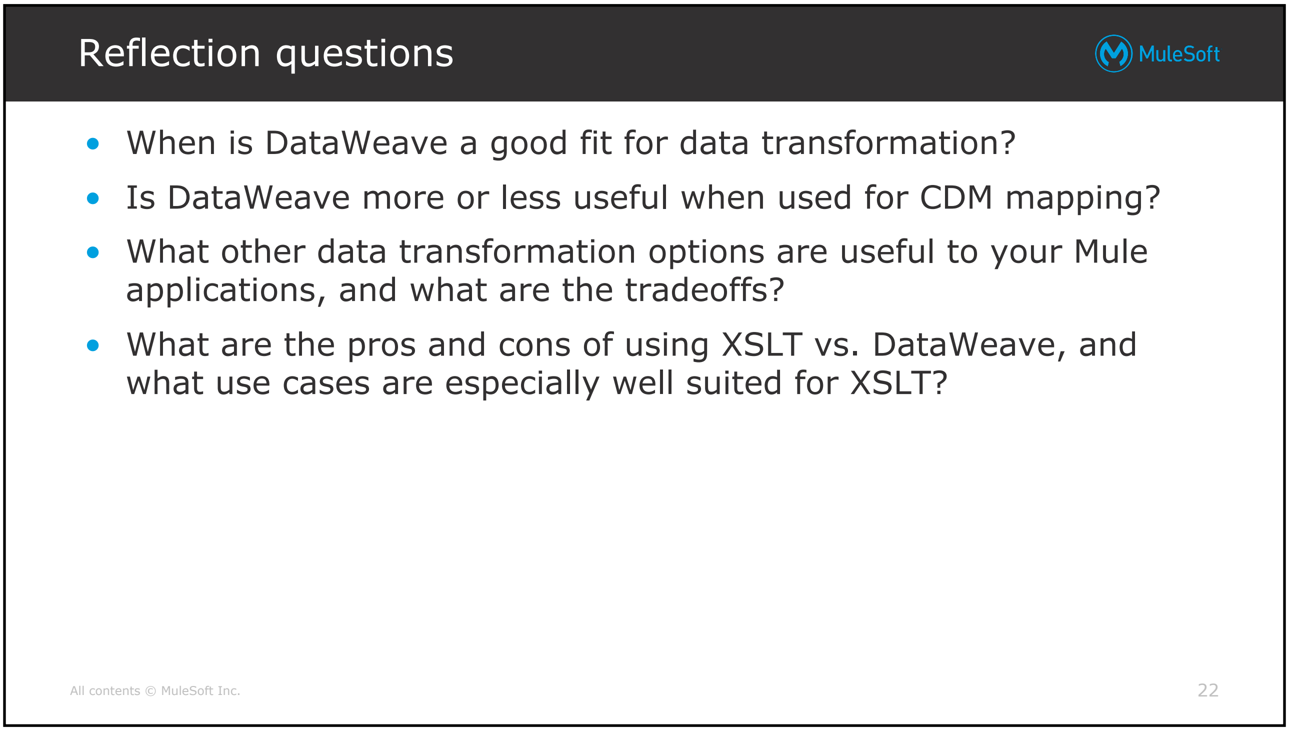
Transformation using Java

- Java is generally NOT recommended for data transformation
  - Use DataWeave instead
- However, an organization may want to call out to Java to **reuse a Common object model** that is written in Java and is standard in the organization
  - DataWeave can call out to **static** methods in Java
  - Java classes can be defined in a **Spring context** (in the Mule configuration file or in a separate Spring context file) and invoked from a Java component
  - MuleSoft recommends that you encapsulate custom Java transformation into **classes** that can be **injected** and easily **tested**
  - Mulesoft promotes **separation of concern** hence removed the Expression component and Expression transformer from Mule 4



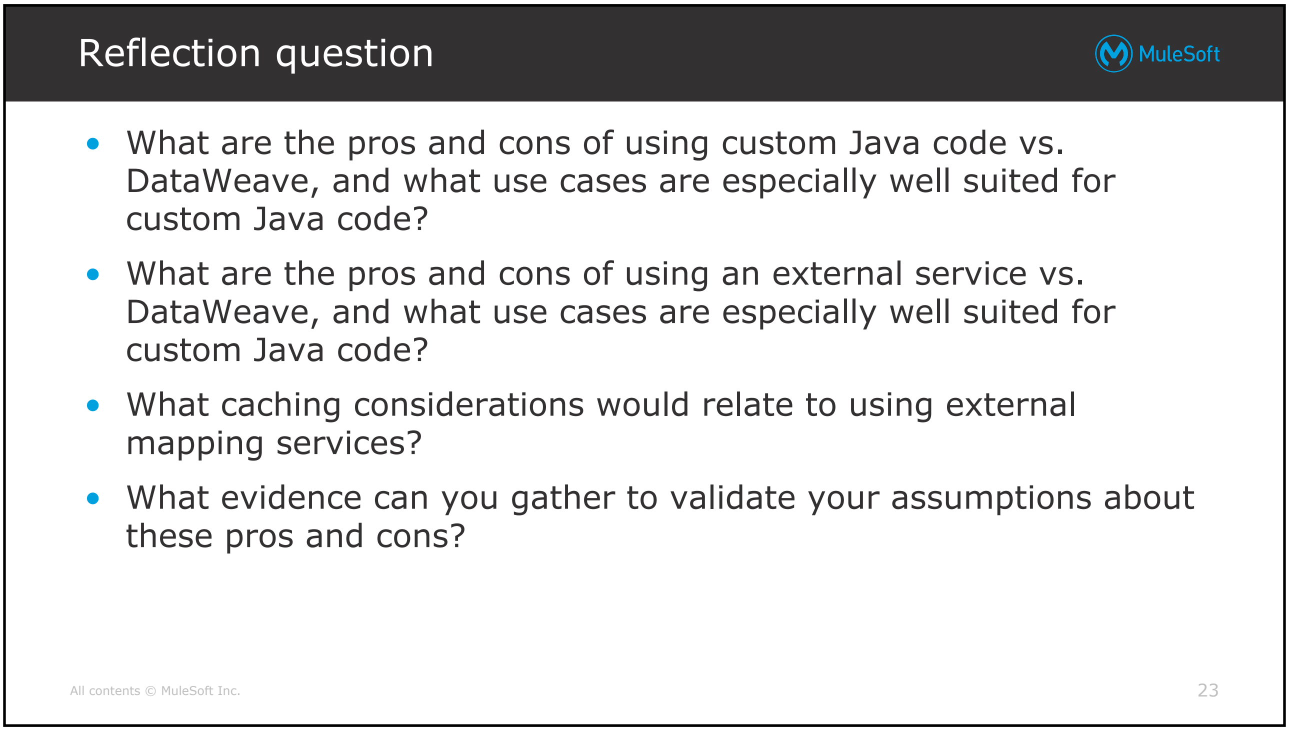
Transformation using scripting Groovy, JRuby (Ruby), JPython(Python), Nashorn(JavaScript)

- Using supported scripting languages is generally NOT recommended for data transformation
  - Use DataWeave instead



Reflection questions

- When is DataWeave a good fit for data transformation?
- Is DataWeave more or less useful when used for CDM mapping?
- What other data transformation options are useful to your Mule applications, and what are the tradeoffs?
- What are the pros and cons of using XSLT vs. DataWeave, and what use cases are especially well suited for XSLT?

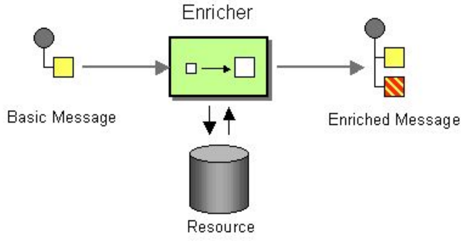


Reflection question

- What are the pros and cons of using custom Java code vs. DataWeave, and what use cases are especially well suited for custom Java code?
- What are the pros and cons of using an external service vs. DataWeave, and what use cases are especially well suited for custom Java code?
- What caching considerations would relate to using external mapping services?
- What evidence can you gather to validate your assumptions about these pros and cons?

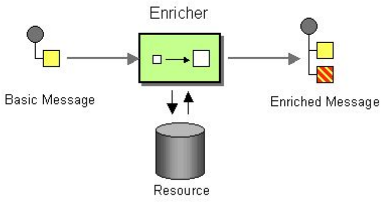
Implementing **event enrichment** patterns (also called content enricher patterns)

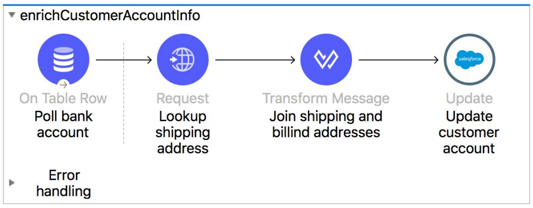
- Context
  - When an event is sent from a source system to a target system, the target system may require **more information** than the source system can provide
- Problem
  - Communication with the target system **may not work** if the event originator does not have all required content in the event
  - For example, security headers (access tokens) or other required data or attributes



Implementing the message enricher pattern with MuleSoft

- Implementation
  - Mule event processors uses information inside the incoming event (e.g. key fields) to retrieve data from an external source
  - After the event processor retrieves the required data from the resource, it appends the data to the payload, or other parts of the message/event
  - Typically the **previous event payload must be stored in a variable, then merged with the new event payload** after the retrieving event processor completes





enrichCustomerAccountInfo

On Table Row
Poll bank account

Request
Lookup shipping address

Transform Message
Join shipping and billind addresses

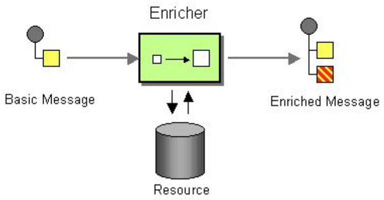
Update
Update customer account

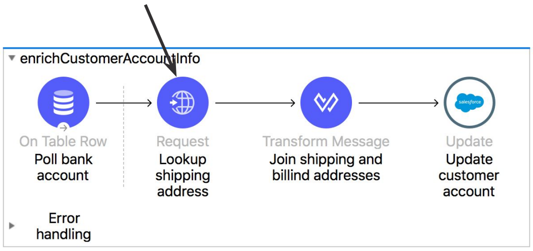
Error handling

Ways to preserve the previous event payload after an event processor executes

- Many event processors have a **target**
  - A target saves you the step of having to store the previous event payload using a Set Variable event processor
  - When a target is set, the event processor result is stored in the variable named by the target
  - The payload from the previous event processor is then left unchanged
  - The target sets its value using a DataWeave expression that can operate on the event processor result (such as the response to a lookup operation)

```
<https:request target="shipAddress" targetValue="#[payload.address]" .../>
```





Reflection questions

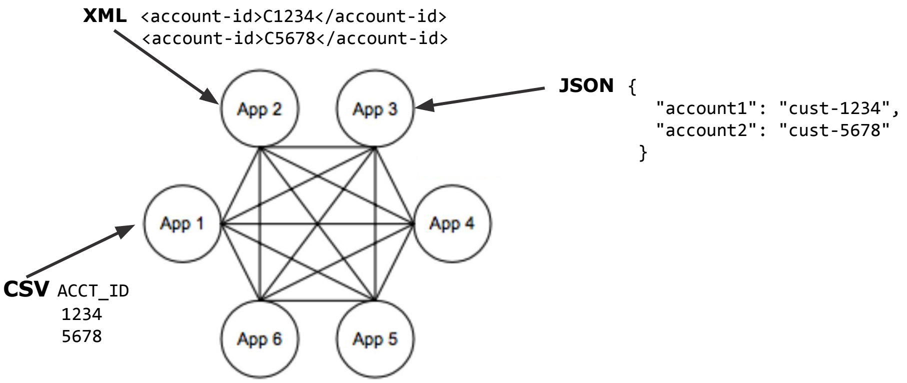
- How can data be joined between previous event processors, and what are the tradeoffs?
- How does data volume, request volume, or latency concerns affect these tradeoffs and choices?
  - You will see more about these tradeoffs later in this module
- Note: CDMs are discussed in more detail in the Anypoint Platform Architecture: Application Networks course

Simplifying and reusing data mappings using common data models

Critical data is often locked up across an enterprise in incompatible silos

MuleSoft

- When an application is integrated with other applications, it needs to know how to
  - Parse data coming in from other applications
  - Format data to be sent out and to be understood by other applications



XML

```
<account-id>C1234</account-id>
<account-id>C5678</account-id>
```

JSON

```
{
   "account1": "cust-1234",
   "account2": "cust-5678"
 }
```

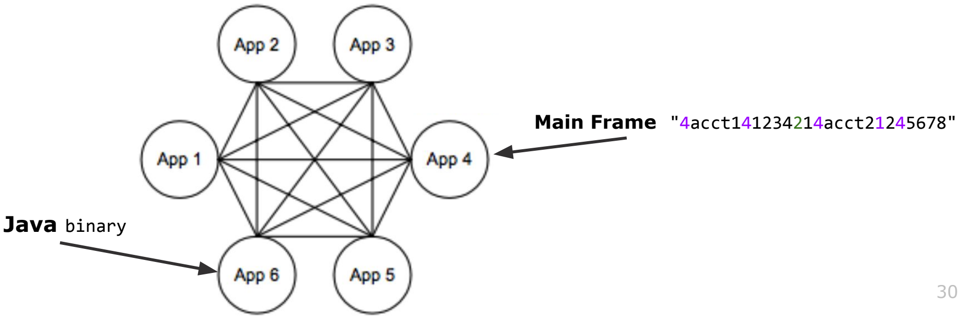
CSV

```
ACCT_ID
 1234
 5678
```

Why separate point-to-point schema mappings are difficult to build and to maintain

MuleSoft

- Ideally, data schema of every app is **known in advance**
- But there can be issues
  - How should data schemas be described and shared?
  - Data represented in one data schema in one system may not be easily transformed into data using another data format and schema type
    - Applications may create **non-standard schema**
    - Data may be a **binary stream**



Main Frame `"4acct141234214acct21245678"`

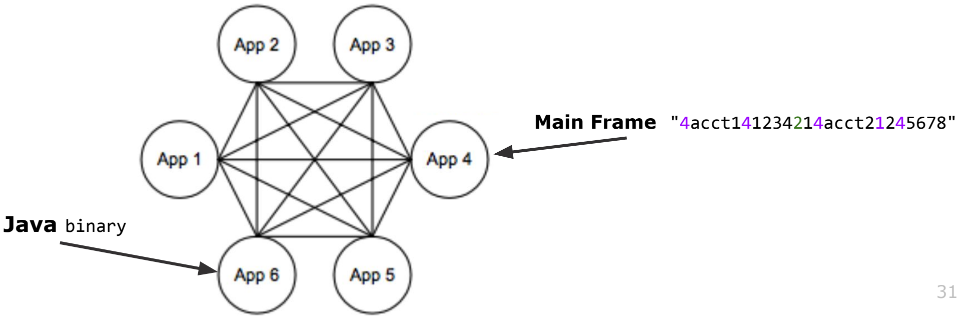
Java `binary`

The schema mapping explosion problem

MuleSoft

- Some mappings may require coupling **two non-standard** and **obsolete** schemas
  - These brittle mappings are difficult and expensive to maintain
- And there are a lot of them
  - In the worst case, N*(N-1) separate and often custom mappings are required, which is O(N^2) complexity

N*(N - 1) mappings



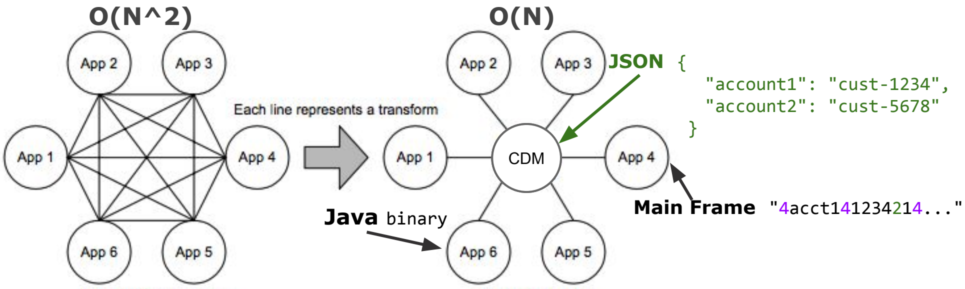
Main Frame `"4acct141234214acct21245678"`

Java `binary`

The solution provided by a common data model (CDM)

MuleSoft

- Build a **common** (also called canonical) data model
  - Specify general rule or acceptable procedure
  - Map from different native application data schema into more common or more neutral data schema
  - In the **worst case, 2*N mappings are required** to support n systems with disparate data schemas, which is O(N) complexity
  - Moreover, each mapping is into an agreed, common, and more open format

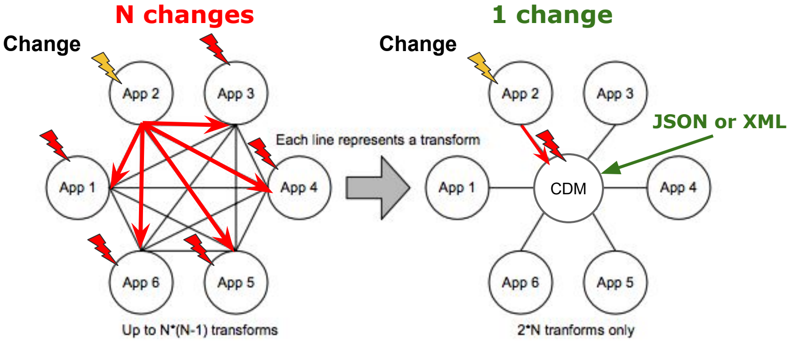


```
{
   "account1": "cust-1234",
   "account2": "cust-5678"
 }
```

A common data model decouples multiple transformation steps

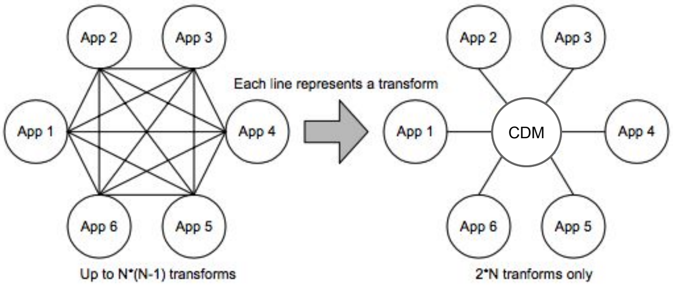
MuleSoft

- Isolates backend systems and supporting Mule applications from change
- Encourages innovation and reuses with existing services
  - Encourage evolution of legacy systems and business processes towards more modern, standardized, and cloud-friendly data formats, like JSON (or even XML)



When should a common data model be used?

- To represent data in new applications that need to **communicate with other internal applications** inside the intranet environment
- When disparate organizations or business units can **agree** on a **common** standard or format, or when the scope is narrow enough
- When the process of defining a common data model does **not delay** implementation indefinitely



Scoping a common data model

- There are tradeoffs on specifying CDMs depending on the intended scope
- Possible scopes include
  - Just to the **current project**
  - To the **business unit** or other organization context
  - To the **entire enterprise**
- Enterprise-wide CDMs may not be helpful, and may significantly delay implementations
  - It may be difficult or impossible to get agreement across business units or even within one business unit
- Note: CDMs are discussed in more detail in the Anypoint Platform Architecture: Application Networks course

Reflection questions

- What are the overall pros and cons of creating a CDM?
- What are some scenarios where a common data model (CDM) is an obvious fit?
- What are some scenarios where a CDM is not a good fit?
- As what project or organizational scope(s) are CDMs typically successfully applied?
- At what project or organizational scope(s) are CDMs often a hinderance or liability?
- What are the tradeoffs to building one unified CDM for the entire enterprise (such as a Master Data Management effort)?

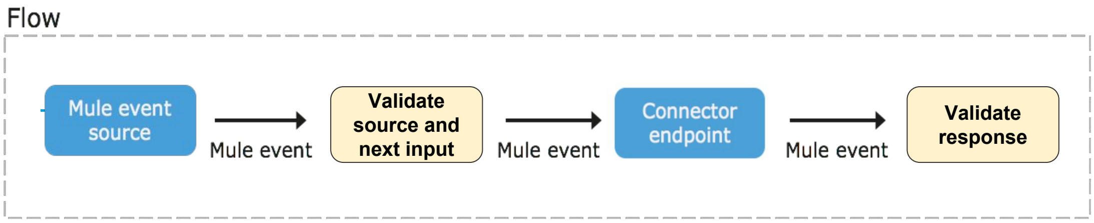
Choosing data validation patterns for Mule applications

The problems addressed by message validation patterns

- Context
  - In a flow, the current Mule event may **contain invalid or incorrect data** that was generated by humans, applications, or during transmission
    - Especially after a response from an external system is returned by a connector
  - The current Mule event may also have data that is **incompatible with data** transformation or connector requests later in the flow
- Problem
  - Incorrect data sent to downstream systems may cause data related **errors**
  - At a minimum this could consume **unnecessary system resources**
  - Even worse, unexpected data errors could **crash** downstream applications

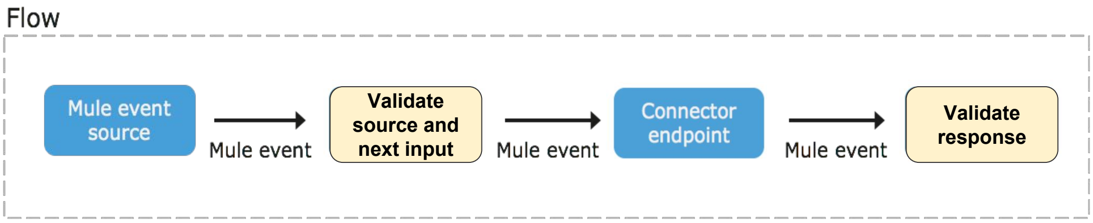
How message validation patterns make Mule applications more predictable and reliable

- Solution
  - Mule applications need to **validate** data structures and content as Mule events are transformed through Mule flows
  - Data validation involves some type of **boolean logic test** to decide if particular **expected conditions** are **true or false** at that point of the flow
  - In particular, Mule applications should **identify invalid or incorrect data** and **handle them**, before sending to external downstream systems



Where data should be tested and checked in Mule flows

- In practice, MuleSoft recommends validating events as early as possible
  - At the **beginning of each flow**, if the event source has incoming data
    - Such as for an HTTP listener, database On Table Row, or JMS listener operation
  - Later in a flow, immediately after an **event is created or enriched**
- In other words, your Mule application should **fail fast**

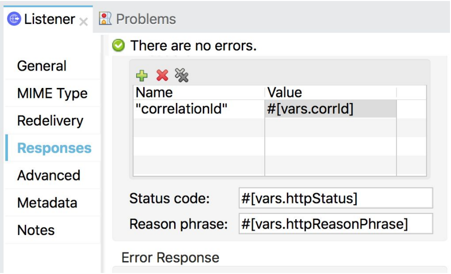


Ways to implement data validation patterns in Mule applications

- Common ways to validate events include
  - **Choice routers**
    - Call different flows based on the previous event processors response
  - **DataWeave code**
    - In any Mule component, write boolean logic to test data or conditions
    - In a Transform Message component, set variables based on results of boolean logic tests
  - **Validation modules**
    - Each validation operation throws an error when a data validation condition is false
    - Core Validation module
    - XML and JSON schema validators
    - APIkit validation
    - HTTP Request success and failure status code validators
  - **Catch and handle errors**
    - If possible, recover from validation errors and continue event processing

Validating and routing response events

- Data validation results can be stored in variables
- HTTP Listener responses can be set from these data validation variables
- A Choice router supports flow control logic to route the Mule event
  - Can call different flows based on the previous event processors response
  - The routing logic can include event validation DataWeave expressions

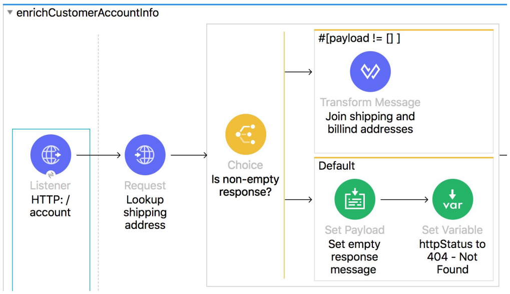


There are no errors.

Name	Value
"correlationId"	#[vars.corrId]

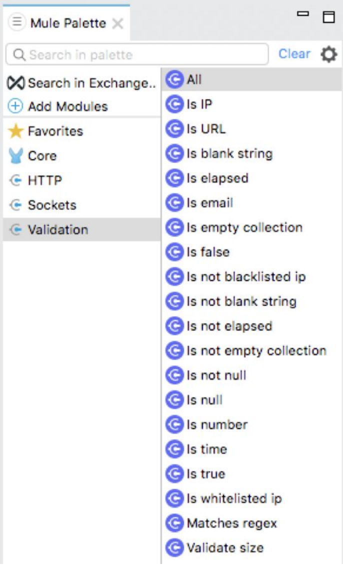
Status code: #[vars.httpStatus]

Reason phrase: #[vars.httpReasonPhrase]



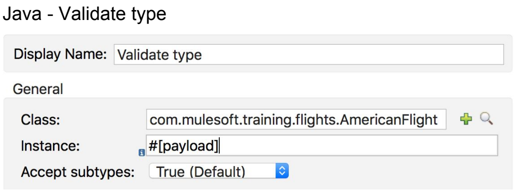
Validating conditions in a flow

- MuleSoft provides various **validation operations** to test (validate) if some conditions are true
  - And throw an error if the condition fails
- The condition is applied to a target DataWeave expression that usually involves the inbound Mule event
- Validation operations also provide an easy way to **throw some specific error type**



Modules that provide validation operations

- Generic validation operations
  - The Validation module
- Other modules provide validation operations for specific schema or data types
  - JSON - Validate schema
  - XML - Validate schema
  - APIkit - JSON validation, throw SOAP faults
  - Java - Validate type
  - HTTP Request operation - Validate response, identify success vs. failure status codes



Java - Validate type

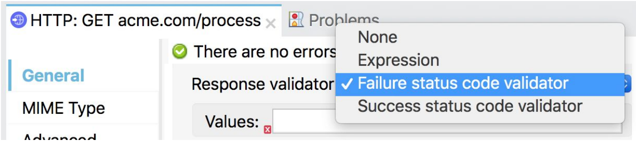
Display Name: Validate type

General

Class: com.mulesoft.training.flights.AmericanFlight

Instance: #[payload]

Accept subtypes: True (Default)



HTTP: GET acme.com/process

Problems

General

MIME Type

Advanced

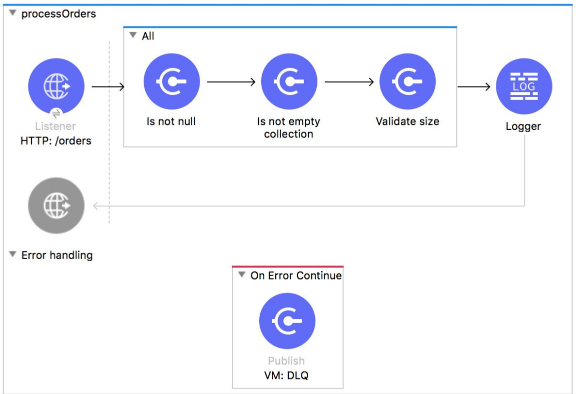
There are no errors

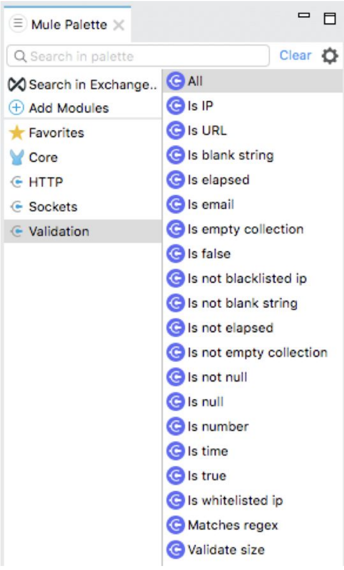
Response validator: ✓ Failure status code validator

Values:

Combining Validation module operations

- The Validation module provides generic validation operations
- The All scope can combine validation operations together





Mule Palette

Search in palette

Clear

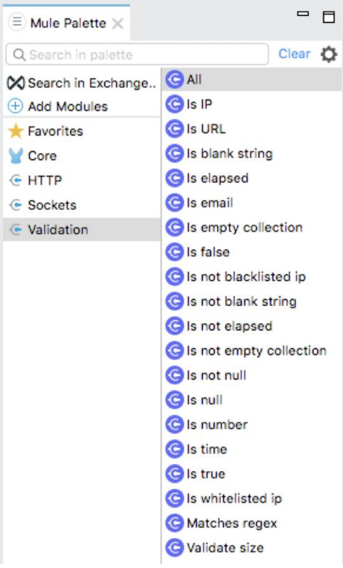
- Search in Exchange..
- Add Modules
- Favorites
- Core
- HTTP
- Sockets
- Validation

- All
- Is IP
- Is URL
- Is blank string
- Is elapsed
- Is email
- Is empty collection
- Is false
- Is not blacklisted ip
- Is not blank string
- Is not elapsed
- Is not empty collection
- Is not null
- Is null
- Is number
- Is time
- Is true
- Is whitelisted ip
- Matches regex
- Validate size

Using the Validation module without throwing an error

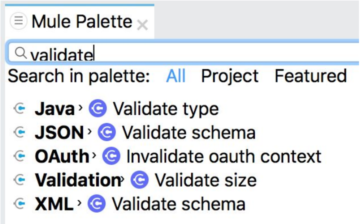
- Validation operations can be used directly inside DataWeave code
- In this case, the operation just returns true/false, it does not throw an error
- You can use this to write Choice router conditions

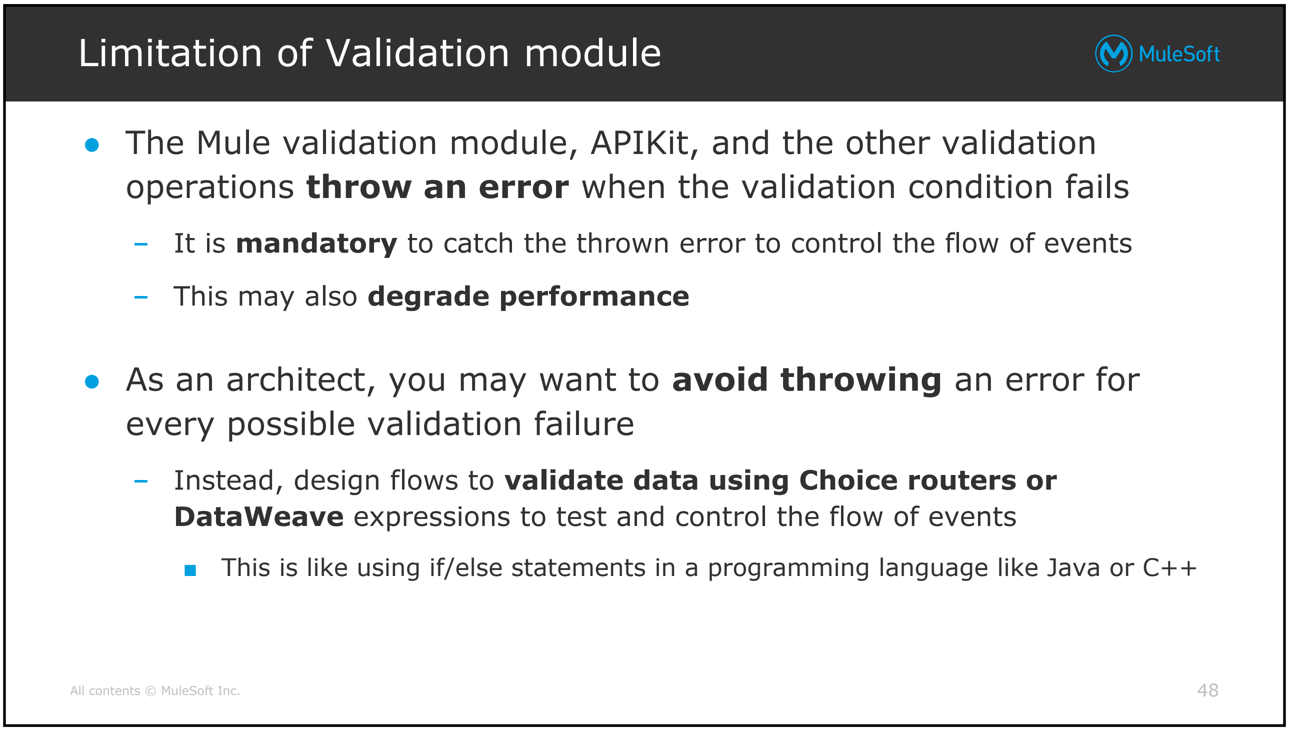
```
<choice>
  <when expression="#[ Validation::isEmail(vars.unknownVariable) ]" >
    <set-payload value='#[vars.unknownVariable ++ " is a valid email."]'/>
  </when>
  <otherwise >
    <set-payload value='#[vars.unknownVariable ++
              " is a not a valid email or a valid url."]'/>
  </otherwise>
</choice>
```



Validating schema in a flow

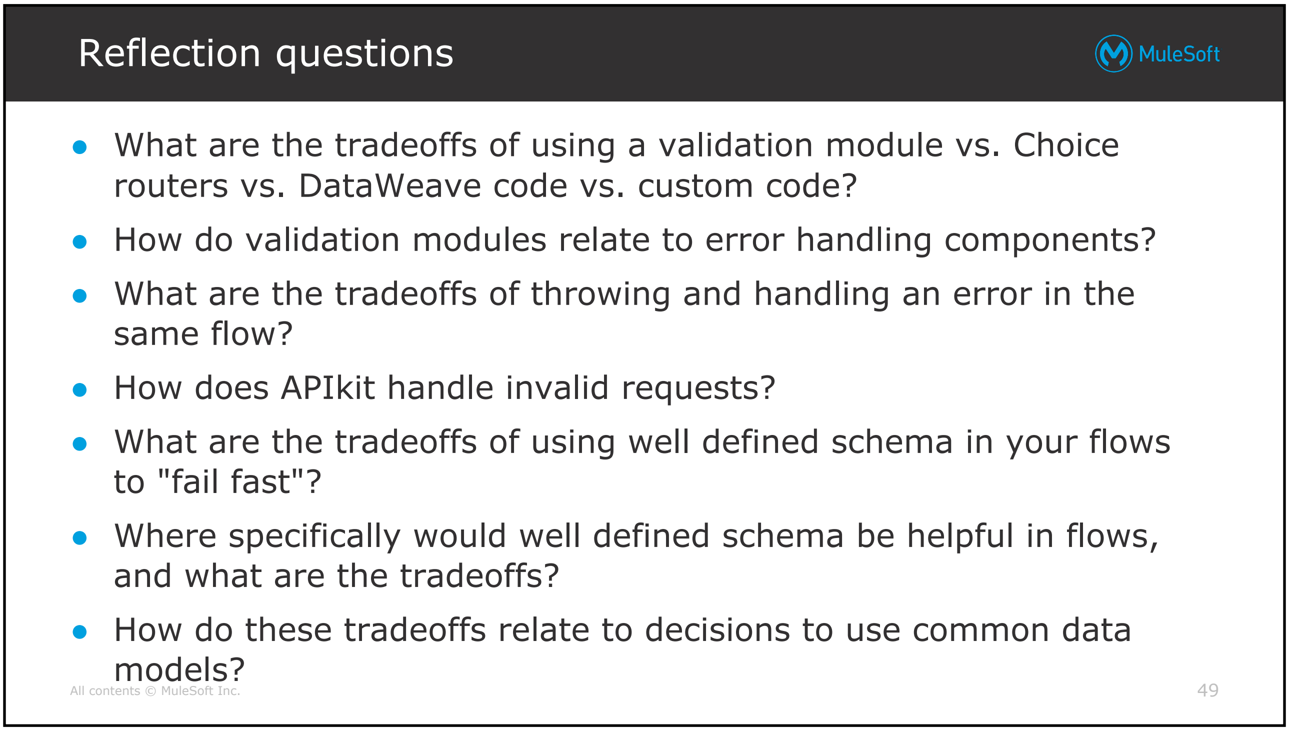
- There are also validation operations in other modules
  - **XML, JSON, and Java**


Mule Palette
validate
Search in palette: All Project Featured
Java › Validate type
JSON › Validate schema
OAuth › Invalidate oauth context
Validation › Validate size
XML › Validate schema



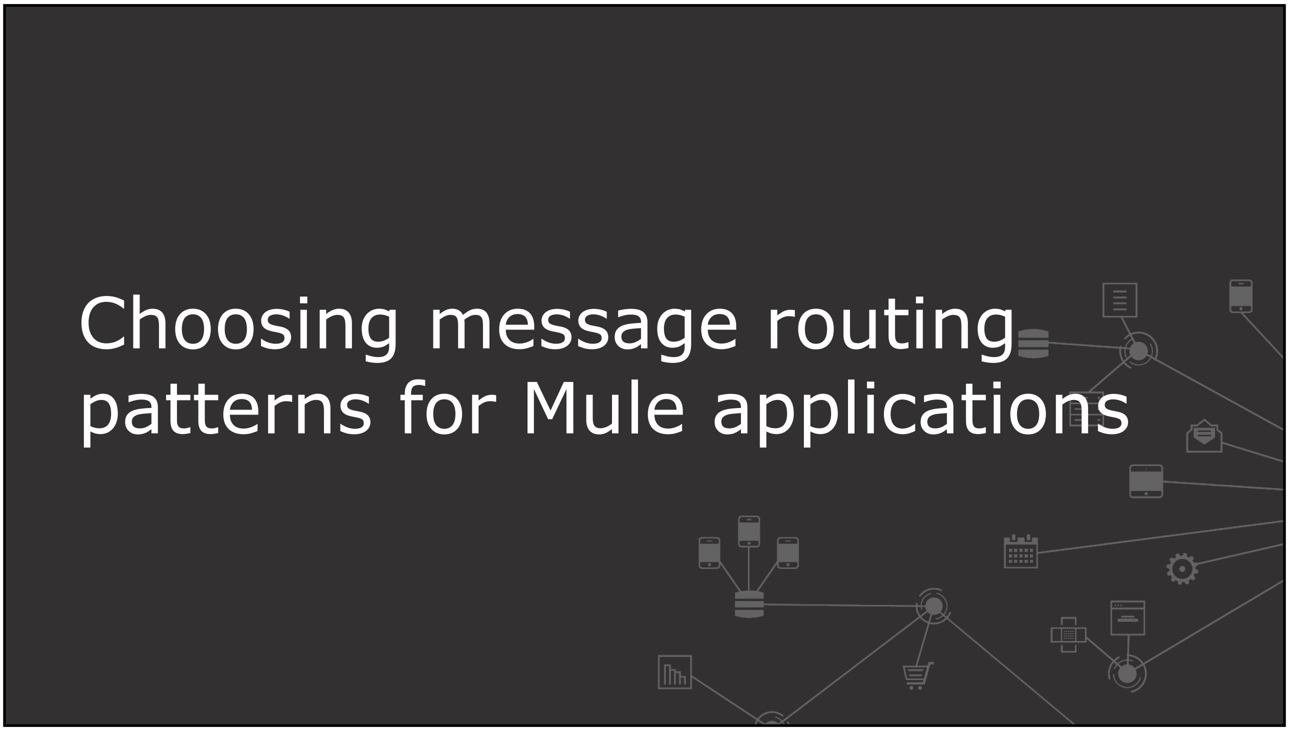
Limitation of Validation module

- The Mule validation module, APIKit, and the other validation operations **throw an error** when the validation condition fails
  - It is **mandatory** to catch the thrown error to control the flow of events
  - This may also **degrade performance**
- As an architect, you may want to **avoid throwing** an error for every possible validation failure
  - Instead, design flows to **validate data using Choice routers or DataWeave** expressions to test and control the flow of events
    - This is like using if/else statements in a programming language like Java or C++

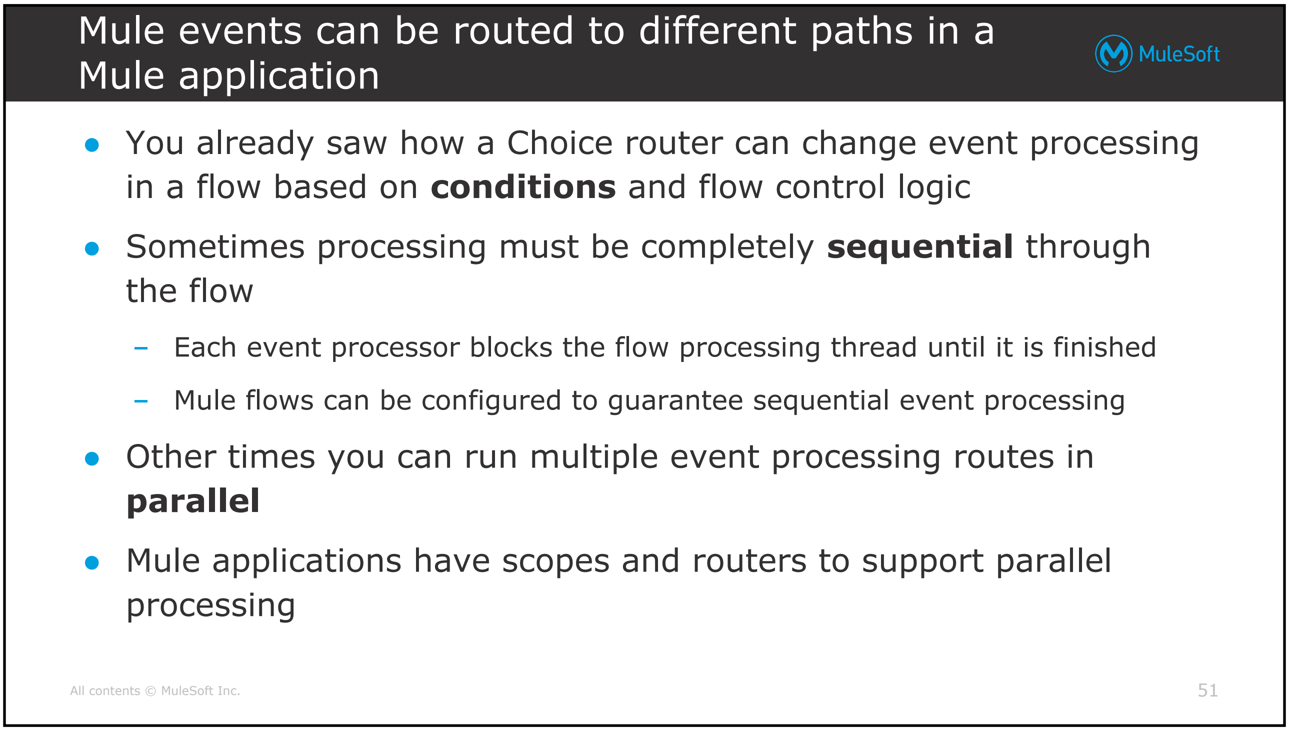


Reflection questions

- What are the tradeoffs of using a validation module vs. Choice routers vs. DataWeave code vs. custom code?
- How do validation modules relate to error handling components?
- What are the tradeoffs of throwing and handling an error in the same flow?
- How does APIkit handle invalid requests?
- What are the tradeoffs of using well defined schema in your flows to "fail fast"?
- Where specifically would well defined schema be helpful in flows, and what are the tradeoffs?
- How do these tradeoffs relate to decisions to use common data models?



Choosing message routing patterns for Mule applications



Mule events can be routed to different paths in a Mule application

- You already saw how a Choice router can change event processing in a flow based on **conditions** and flow control logic
- Sometimes processing must be completely **sequential** through the flow
  - Each event processor blocks the flow processing thread until it is finished
  - Mule flows can be configured to guarantee sequential event processing
- Other times you can run multiple event processing routes in **parallel**
- Mule applications have scopes and routers to support parallel processing

The issue with requiring sequential event processing in a Mule flow

- Context
  - Sometimes business logic requires incoming events to be processed **sequentially**
- Problem
  - Mule runtime is a **multithreaded** that allows processing of incoming events in parallel

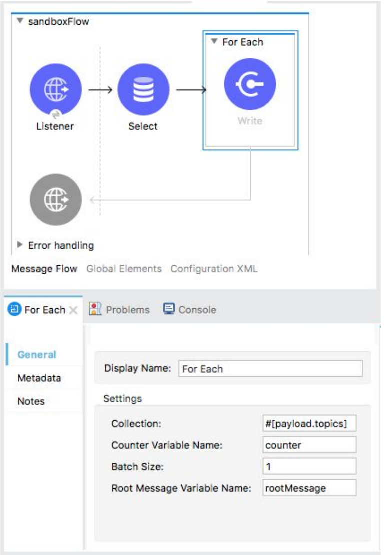
How to configure sequential execution in a Mule flow

- Solution
  - Mule runtime engine allows to specify **max concurrency of 1** which ensures only one flow instance processes events at any point in time
  - The **For Each scope** (foreach) splits a payload into elements and processes them one by one through the components that you place in the scope
  - Akin to **for loops** in other programming languages
- Implementation

```
<flow maxConcurrency="1">
  <http:listener>
 </flow>
```

Ways to implement sequential execution patterns in Mule applications

- By default For Each tries to split the payload
- For Each can split up iterable types
  - If the payload is a simple Java collection, the For Each scope can directly split the payload without any configuration
  - For non-Java collections, such as XML or JSON, you need to specify the iterable collection to use by setting the **collection** attribute
    - The value is evaluated from a DataWeave expression



Display Name: For Each

Settings

Collection: #[payload.topics]

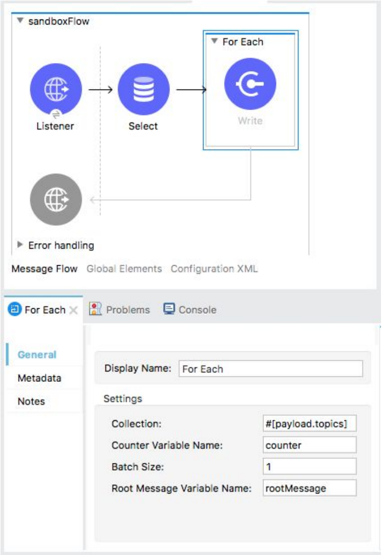
Counter Variable Name: counter

Batch Size: 1

Root Message Variable Name: rootMessage

Using a For Each scope's batch size to improve performance

- The **batch size** aggregates elements in the collection into smaller batch collections
  - Each smaller batch collection is passed together to the For Each scope's event processors
    - Instead of passing individual records one at a time
  - Not to be confused with the **Batch scope**
- A batch size of 1 means that individual elements (not 1-element collections) are passed on



Display Name: For Each

Settings

Collection: #[payload.topics]

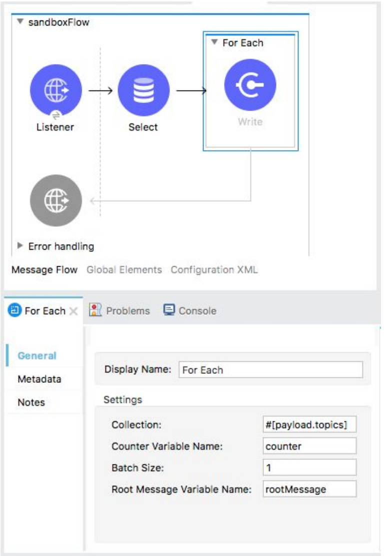
Counter Variable Name: counter

Batch Size: 1

Root Message Variable Name: rootMessage

Using the batch size to improve performance

- The default batch size is 1, so if a collection has 200 elements, the For Each scope iteratively processes 200 elements, each as a separate Mule message
- Using a batch size can reduce the number of iterations of the For Each scope
- Many connectors also support bulk operations, so can handle a collection of records all at once
  - This can avoid rate limits imposed by the backend service or endpoint



Display Name: For Each

Settings

Collection: #[payload.topics]

Counter Variable Name: counter

Batch Size: 1

Root Message Variable Name: rootMessage

Parallel processing

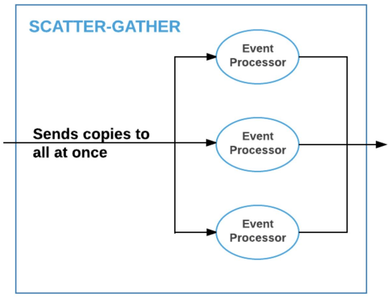
- Context
  - Sometimes business logic requires incoming events to be processed in **parallel**
- Problem
  - In a single flow, processors are **executed in sequence** and each processor execution is dependent on the execution of preceding processor
  - The processor that communicates with an external system might block (sequential) processing of the current Mule event by its flow
    - Such as requesting from a remote server or executing a lengthy database transaction

Supporting parallel event processing in Mule flows

- To execute event processors concurrently and/or in parallel, the Mule runtime provides various processors
  - Scatter-Gather
  - Async scope
  - VM
  - JMS
- Parallel execution occurs using **different** threads from various thread pools
- If one event processor is **blocked**, other processors are able to **continue** processing without waiting for the completion of this processor

Implementing parallel processing using Scatter-Gather

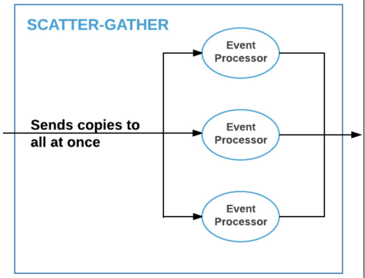
- The Scatter-Gather component is a routing event processor
  - Copies (scatters) the Mule event to multiple routes
  - Each route is processed in parallel
  - Parallel execution of routes may improve performance
- The Scatter-Gather completes after every route completes, or after a configured timeout expires
  - The default timeout is to wait forever
  - The timeout value is applied in parallel to each route
  - If a route's timeout expires, a MULE:TIMEOUT error is thrown for that route



How the results from each route is gathered at the end of a Scatter-Gather

MuleSoft

- If every route succeeds
  - After the last route finishes, each of Mule events are **gathered** into a single Mule event that is passed to the next event processor in the flow
  - The gathered Mule event payload contains each route's resulting Mule event
    - Access each route's result payload from the array `#[payload..payload]`

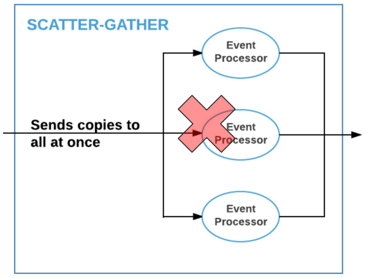


```
"payload" : {
        "0" : {
                "attributes" : {
                        "properties" : { … }
                        "headers" : { … }
                        …
        }
                "payload" : {
                        "firstRoute" : "First Payload"
                }
                …
        }
        "1" : {
                …
                "payload" : {
                        "secondRoute" : "Second Payload"
                }
        }
}
```

How errors from a Scatter-Gather are handled

MuleSoft

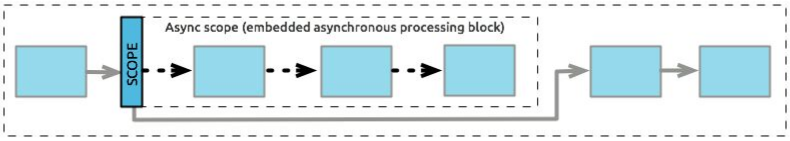
- If any route throws an error
  - An error of type MULE:COMPOSITE_ROUTING is thrown
  - The error object contains the result (Mule event or error) of every route, organized as successes or failures
  - The error object also includes the error from any routes that timed out
  - Event processing does not continue with the next event processor in the flow
  - Instead, the flows error handlers process the error as they would any other error type



- To recover from an individual route error, use a flow reference in each route and handle errors in the referenced flows

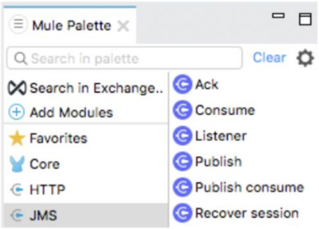
Implementing concurrent processing using an Async scope

- The event processors inside an Async scope execute concurrently with the event processors from the main flow
- **Response** from the Async scope's event processing is **not accessible** nor is it returned to the main flow
- Errors inside the Async scope's event processors do not impact the main flow
  - And **do not use the outer flows error handler**
  - Use a Try scope inside the Async scope to customize error handling
  - Otherwise the default error handler is used



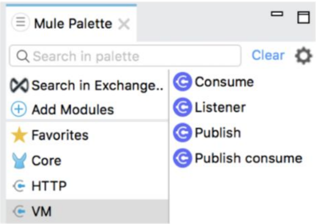
Implementing concurrent processing using JMS destinations

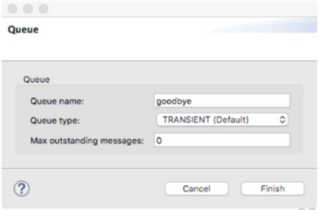
- JMS connector provide async communication for intra and inter application through JMS destinations via a shared message broker
- JMS publish and consume/listener processors can be used to implement concurrent processing
- Supports various messaging models
  - Point-to-point queues
  - One-to-many topics



Implementing concurrent processing using VM queues

- VM connector provides async communication for intra and inter application through asynchronous, inside-the-JVM queues
  - Define the VM connector in a Mule domain to share the VM queue between Mule applications
- The VM module's Publish, Consume, and Listener operations can be used to implement concurrent processing




Queue

Queue

Queue name: goodbye

Queue type: TRANSIENT (Default)

Max outstanding messages: 0

Implementing parallel out-of-order processing with a Batch Job

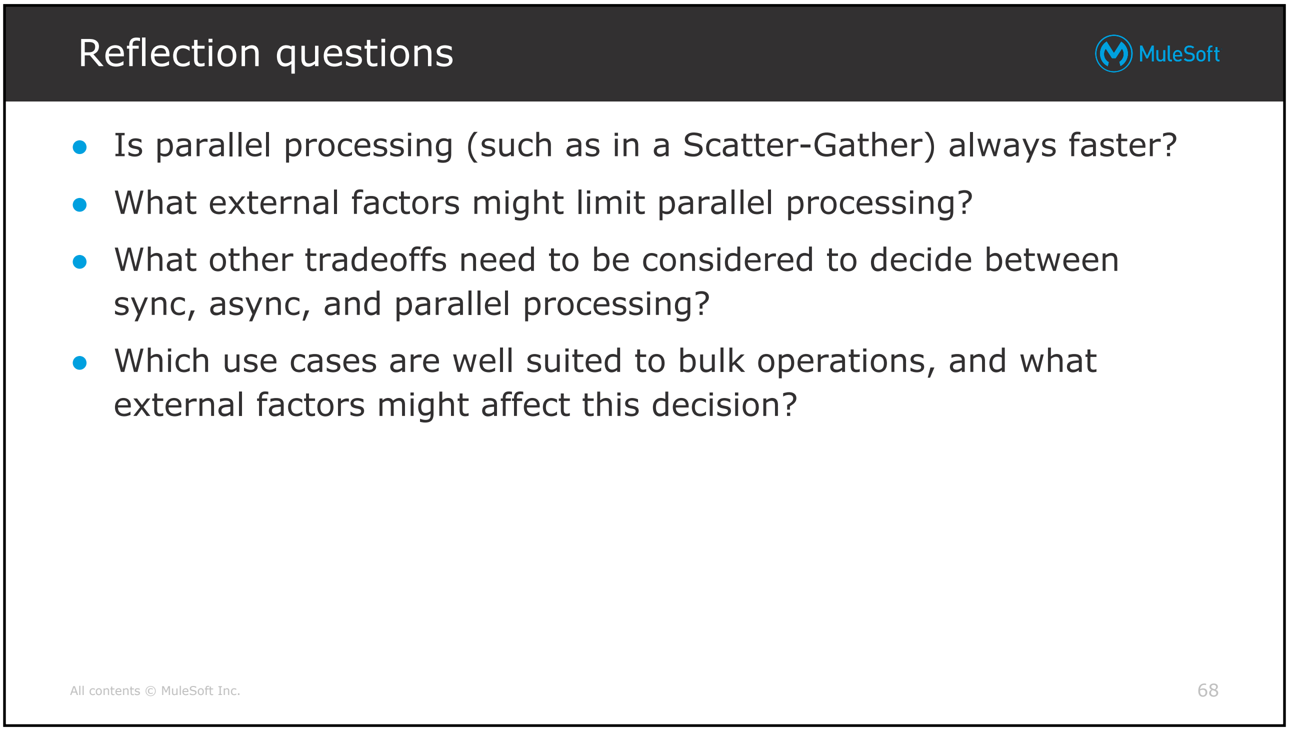
- A Batch Job processes a collection of records in a batch grouping at a time
- Each batch group is processed together through the Batch Job
- Each Batch Step processes in parallel in its own threads
- If a previous record blocks an earlier Batch Step, a later record can skip ahead to the next Batch Step and can complete the entire Batch Job out of order
- However, the Batch Job guarantees that each record moves through each Batch Step in sequential order

Aggregating operations inside a Batch Step scope

- A Batch Step can include a Batch Aggregator scope
- The Batch Step can be configured with an aggregation count
- The Batch Step will collect processed records until the aggregation count is reached
- Then the aggregated records are sent together to the event processors inside the Batch Aggregator scope
- Some connectors support bulk operations that can directly handle the aggregated payload
  - For example, the Salesforce and Database connectors
  - This can increase throughput at the expense of timeliness

Reflection questions

- What are the tradeoffs to process collections of elements/records with a For Each scope vs. a Batch Job vs. all at once with a single bulk operation?
- What are the tradeoffs of using bulk operations inside a Batch Job vs. all at once?
- What are the tradeoffs of using a single bulk operation vs. processing each record in a For Each scope?



Reflection questions

- Is parallel processing (such as in a Scatter-Gather) always faster?
- What external factors might limit parallel processing?
- What other tradeoffs need to be considered to decide between sync, async, and parallel processing?
- Which use cases are well suited to bulk operations, and what external factors might affect this decision?

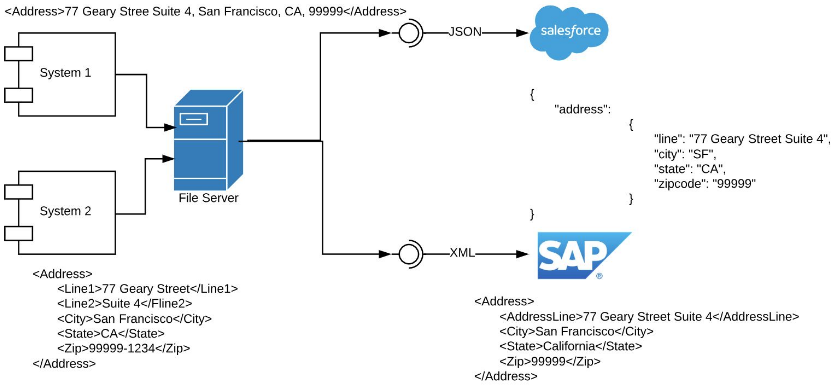
Applying message transformation, validation, and routing patterns

Applying message transformation, validation, and routing patterns to use cases

- Now you can apply the design patterns from this module to some of the course use cases
- This includes deciding when and how to apply design patterns such as
  - The best way to transform data between endpoints
  - When and how to design common data models and data mappings
  - The best way to write defensive flows and the best routing patterns to use
  - The best way to handle errors

Exercise 5-1: Apply message transformation and routing patterns to a systems integration use case

- Design integration between multiple inbound and outbound systems
- Apply enterprise integration design patterns efficiently and effectively to meet requirements of a specific scenario



```
<Address>77 Geary Stree Suite 4, San Francisco, CA, 99999</Address>
```

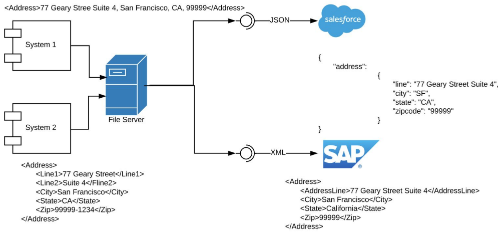
```
{
    "address":
        {
            "line": "77 Geary Street Suite 4",
            "city": "SF",
            "state": "CA",
            "zipcode": "99999"
        }
}
```

```
<Address>
    <Line1>77 Geary Street</Line1>
    <Line2>Suite 4</Fline2>
    <City>San Francisco</City>
    <State>CA</State>
    <Zip>99999-1234</Zip>
</Address>
```

```
<Address>
    <AddressLine>77 Geary Street Suite 4</AddressLine>
    <City>San Francisco</City>
    <State>California</State>
    <Zip>99999</Zip>
</Address>
```

Exercise context

- Enterprise integration requirements
  - Two systems **System 1** and **System 2** send files with the same kind of information to a **shared file server**
  - System 1 and System 2 both **write XML files**, but each system uses a different data model to represent an **Address** object
  - Data from both systems must also be **sent** to **SAP** and **Salesforce** systems
    - Which again use **different data models** for **Address** objects



```
<Address>77 Geary Stree Suite 4, San Francisco, CA, 99999</Address>
```

```
{
    "address":
            {
                "line": "77 Geary Street Suite 4",
                "city": "SF",
                "state": "CA",
                "zipcode": "99999"
            }
}
```

```
<Address>
    <Line1>77 Geary Street</Line1>
    <Line2>Suite 4</Fline2>
    <City>San Francisco</City>
    <State>CA</State>
    <Zip>99999-1234</Zip>
</Address>
```

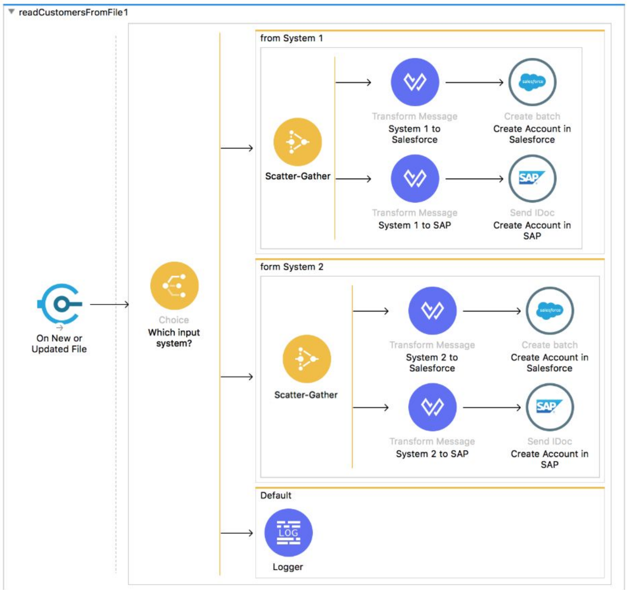
```
<Address>
    <AddressLine>77 Geary Street Suite 4</AddressLine>
    <City>San Francisco</City>
    <State>California</State>
    <Zip>99999</Zip>
</Address>
```

Exercise steps

- Decide which data transformation processes are required or not
  - From System 1 to Salesforce
  - From System 1 to SAP
  - From System 2 to Salesforce
  - From System 2 to SAP
- Decide ways to reduce multiple transformations
  - Can a standardized CDM across the systems reduce the number of transformations?
  - How might a CDM decouple future schema changes in System 1 or System 2 or the SAP and Salesforce systems?
  - How might a CDM help if a new System 3 needs to be connected?
- Design Mule integration flow(s) that use a CDM for this scenario

Exercise solution: Without a CDM

MuleSoft

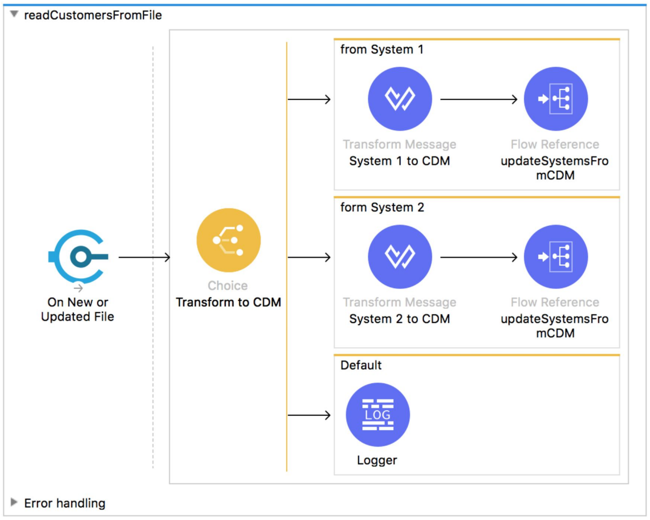


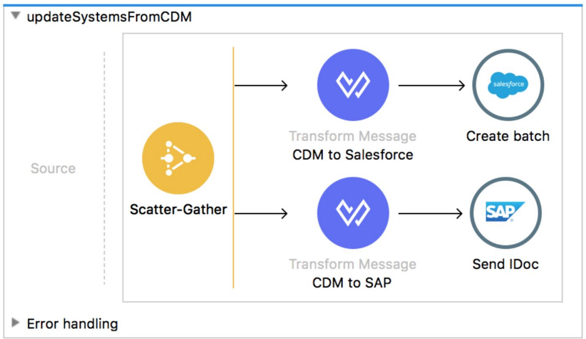
- Have to create different transformations for 2x2 routes
- Have to create separate transformations to Salesforce and SAP twice
  - These mappings tend to be complicated and vendor specific

Exercise solution: With a CDM

MuleSoft

- Transformation to Salesforce and SAP are reused

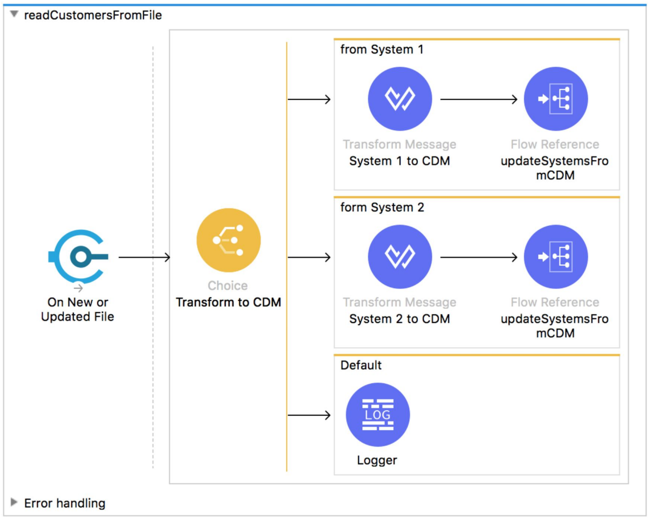


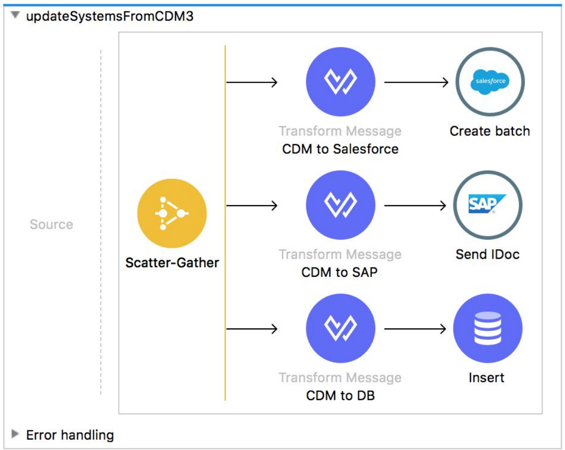


Exercise solution: Adding a third output endpoint

MuleSoft

- Adding a third endpoint is now decoupled from the CDM step





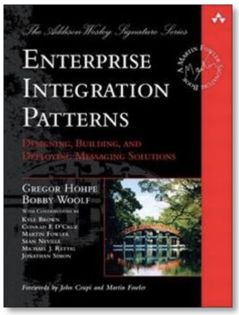
Summary

Summary

- DataWeave can simplify translating between common input and output data types
- Mule events often need to be validated and routed within a flow
- Defensive code is implemented with a combination of choice routers, validator components and DataWeave expressions, and error handling
- Scatter-Gather, Async scopes, and Message queues can help process events in parallel

References

- Hohpe & Woolf, 2003
  - Describes 65 patterns for EAI and MoM
  - http://www.eaipatterns.com/
  - Mule runtime has implemented many of these patterns


The Addison-Wesley Signature Series
Enterprise Integration Patterns
Designing, Building, and Deploying Messaging Solutions
Gregor Hohpe
Bobby Woolf
With Contributions by
Kyle Brown
Conrad F. D'Cruz
Martin Fowler
Sean Neville
Michael J. Rettig
Jonathan Simon
Forewords by John Crupi and Martin Fowler
A Martin Fowler Signature Book