

In WSO2 Micro Integrator, the message flow between a client and a service is handled using a series of sequences that manage the message journey. Here’s a simple, step-by-step explanation with an example:

**Explanation of Message Flow in WSO2 Micro Integrator**

WSO2 Micro Integrator acts as a bridge between a **client** (someone or something requesting a service) and the **service** (which processes the request and provides a response). This interaction is controlled through three main sequences:

1. **In Sequence**
2. **Out Sequence**
3. **Fault Sequence**

**Step-by-Step Message Flow**

1. **Client Request (Step 1)**
   * The client sends a request, like asking for the current temperature from a weather service API.
2. **In Sequence (Step 2)**
   * The message flows through the **In Sequence** in the Micro Integrator. This sequence is where the initial processing occurs, such as logging the request, authenticating the client, or transforming data formats to match the service’s requirements.
   * For example, if the client’s request format is JSON, and the service expects XML, the In Sequence can handle this transformation.
3. **Service Interaction (Step 3)**
   * After processing in the In Sequence, the message is forwarded to the target **service** (e.g., the weather service API). The service then processes the request.
4. **Service Response (Step 4)**
   * The service generates a response, such as returning the current temperature for the requested location.
5. **Out Sequence (Step 5)**
   * The response goes through the **Out Sequence** in the Micro Integrator, where it may be further processed. For example, the response format might be converted back to JSON if that’s what the client expects, or additional response logging might occur.
6. **Client Delivery (Step 6)**
   * Finally, the processed response is delivered to the client, completing the interaction.

**Error Handling with Fault Sequence**

* If any error occurs (e.g., the service is unavailable or there’s a data format issue), the **Fault Sequence** handles the error.
* By default, the Fault Sequence logs the error details, including the payload and any exceptions encountered, and stops further processing.
* You can configure the Fault Sequence to perform custom error handling actions, such as retrying the request, notifying an administrator, or returning a custom error message to the client instead of simply dropping the message.

**Example Scenario**

Imagine a client requests data from a currency conversion service through the WSO2 Micro Integrator.

* **In Sequence**: The integrator validates the request format and converts currency values if needed.
* **Service Interaction**: The request reaches the conversion service.
* **Out Sequence**: The service response (e.g., the conversion result) is formatted or logged before being sent back.
* **Fault Sequence**: If the service is down, the Fault Sequence could return an error message like, "Service temporarily unavailable. Please try again later."

This structured flow ensures that messages are handled consistently, with custom options for handling errors if they arise.

**Work flow to send request to the backend first we create Endpoint that points to backend service url and create Rest API that consume or call the predefined endpoint and return the response to the client**

**Switch Mediator (Request Routing based on payload content)**

Based on the client request payload redirecting or routing the request to different endpoints where we have to already create different endpoints which points to different backend service

* + - The main purpose using property mediators in this case is to set new message property that will be used in message routing to identify which backend service should be called to make the appointment
    - In other word this property mediator will extract the hospital name from the request payload

In WSO2, the **Switch Mediator** is used to route messages to different sequences or endpoints based on the value of a specified expression, known as the **Source XPath**.

**Source XPath in Switch Mediator**

* **Source XPath** is the expression used by the Switch Mediator to evaluate the message content and determine which route (or case) to follow.
* XPath (XML Path Language) is commonly used to locate elements or attributes within an XML message.
* In the Switch Mediator, the Source XPath points to a specific part of the incoming message that you want to evaluate, such as a particular tag or attribute value.

For example, if you want to route messages based on the value of a <status> element in the XML payload, you can set the Source XPath to //status. The Switch Mediator will check the value of this element to decide which case sequence to execute.

**Purpose of the Switch Mediator**

The **Switch Mediator** is used for conditional routing. It allows you to define multiple "cases" based on the value of the Source XPath expression. Each case has a specific sequence of actions or endpoints that the message will follow if the evaluated expression matches the case value.

**Example Scenario**

Imagine you have an XML message that contains a <type> element indicating the type of service requested:

xml

Copy code

<request>

<type>medical</type>

</request>

In this case:

1. Set the **Source XPath** in the Switch Mediator to //type.
2. Define cases based on possible values of <type>, such as "medical" or "surgical".
3. If <type> equals "medical", the Switch Mediator can route the message to a sequence or endpoint that handles medical requests. If it equals "surgical", it routes the message to another sequence for surgical requests.

**Why Use Switch Mediator?**

The Switch Mediator is useful when you need to direct messages to different processing flows based on certain criteria within the message content. It’s particularly helpful for:

* Routing messages to different services based on a message attribute.
* Handling different business rules based on the message content.
* Reducing complexity by keeping all routing logic within one mediator, rather than using multiple filters or conditional mediators.

This approach keeps your routing logic organized and ensures that each type of message is directed to the correct process or service.

WORKFLOW OF THE SWITCH MEDIATOR

* + - INTIALLY CREATE DIFFERENT ENDPOINT WHICH IS MAPPED WITH DIFFERENT BACKEND SERVICE URL THEN IN REST API
    - FIRST ADD LOG START 🡪
    - SET PROPERTY USED FOR READING WHICH HOSPITAL IS SPECIFIED IN THE REQUEST PAYLOAD AND IT WILL SET THE VALUE PASS TO NEXT STEP
    - IN THE SWITCH CASE CREATE DIFFERENT NUMBER OF CASE BASED ON THE NUMBER OF ENDPOINT AND AT THE LAST MAKE IT DEFAULT
* This is the command in order to call the API for the POST command using curl –
* 🡪 F:\WSO2 Resources\Training Document\BackEnd Service>curl -v POST --data @patientExample.json "http://localhost:8290/healthcare/categories/surgery/reserve" --header "Content-Type:application/json" -w "\n"

[Routing Requests based on Message Content - WSO2 API Manager Documentation 4.2.0](https://apim.docs.wso2.com/en/4.2.0/tutorials/integration-tutorials/routing-requests-based-on-message-content/)

**Message Transformation in WSO2**

Message transformation is essential for converting and adapting data formats between services. WSO2 provides several mediators for this purpose:

1. **Data Mapper Mediator**:
   * A visual tool for transforming data across formats like JSON, XML, and CSV.
   * Integrates with mediation sequences to apply transformations within workflows.
   * Allows you to map, structure, and manipulate data elements graphically.
   * **Example**: Transforming an incoming JSON payload to XML for another service, e.g., converting { "name": "Alice" } to <User><Name>Alice</Name></User>.
2. **Payload Factory Mediator**:
   * Allows custom message content creation with a specified format.
   * **Example**: Setting a default response format or constructing a custom error message.
3. **XSLT Mediator** and **FastXSLT Mediator**:
   * Use XSLT (eXtensible Stylesheet Language Transformations) to transform XML-based messages.
   * **Example**: Converting XML <User><Name>Alice</Name></User> into a different XML structure using XSLT stylesheets.
4. **Smooks Mediator**:
   * Supports complex transformations using Smooks, especially for EDI formats.
   * **Example**: Transforming EDI data to XML.
5. **XQuery Mediator**:
   * Allows transformations using XQuery, suitable for querying and restructuring XML.
   * **Example**: Querying specific nodes in XML and transforming the structure.

**Data Mapper Mediator**

The **Data Mapper Mediator** is particularly useful for its visual approach to transformation:

* **Input Schema**: Defines the expected structure of incoming data (e.g., JSON, XML).
* **Output Schema**: Defines the structure of the outgoing data.
* **Mapping Configuration**: Specifies the transformations and mappings from the input to the output.

**Example of Data Mapper Usage**

Let’s say a service receives data in JSON format and requires XML output.

1. **Input Schema**:

json

Copy code

{

"user": {

"name": "Alice",

"age": 30

}

}

1. **Output Schema**:

xml

Copy code

<User>

<FullName></FullName>

<Age></Age>

</User>

1. **Mapping Configuration**:
   * Map user.name to User.FullName.
   * Map user.age to User.Age.

**Result**: The data mapper will transform the incoming JSON:

json

Copy code

{ "user": { "name": "Alice", "age": 30 } }

into the required XML:

xml

Copy code

<User>

<FullName>Alice</FullName>

<Age>30</Age>

</User>

The Data Mapper provides a powerful, visual approach to handling such transformations, making it easier to manage complex integrations across different data formats.

**The Data Mapper**

**Reusing Sequence Template**

The **HospitalRoutingSQ** sequence template in the **HealthcareProject**  repository defines a routing sequence for directing messages to a specific hospital based on a parameter, setHospital. Here’s a breakdown of how it works and what it does:

1. **Parameter Declaration**:
   * The template declares a parameter named setHospital which is not mandatory and has a default empty value. This parameter is expected to contain the hospital’s identifier to which the routing will occur.
2. **Sequence Definition**:
   * Within the <sequence> tag, it performs two main tasks:
     + **Logging**: It logs a custom message with the format "Routing to [hospital\_name]" by concatenating the text "Routing to " with the setHospital parameter value.
     + **Setting a Property**: It sets the property **uri.var.hospital** with the value of setHospital, which can be used later in the message flow to direct the message to the specified hospital.
3. **Usage**:
   * This template is likely used as a reusable component that can be called with a specific setHospital value to dynamically route requests to different hospitals within the system. This makes the routing configuration modular, allowing different endpoints to use the same sequence template by passing different hospital values.

The main purpose of the HospitalRoutingSQ sequence template is to enable dynamic routing based on the hospital specified in the setHospital parameter, enhancing flexibility in handling various hospital-specific requests.

**Call Mediator**  which is used to expose multiple services in a single aggregated service

In this case we first create an appointment fee endpoint and then create settlement endpoint and for this endpoint the first endpoint output will be used as input for this one and it will be called after and the first is GET method and the second is POST method

The difference between the **Call Mediator and Send Mediator is**

THE CALL MEDIATOR SENDS THE MESSAGE RESPONSE TO THE NEXT MEDIATOR IN THE SAME MEDIATION FLOW

WHILE THE SEND MEDIATOR PASS THE MESSAGE RESPONSE BACK TO THE OUT SEQUENCE FLOW

The **Call Mediator** and **Send Mediator** in WSO2 serve different purposes for handling message flow in integration scenarios:

1. **Call Mediator**:
   * **Purpose**: Synchronously invokes a backend service and waits for a response before proceeding.
   * **Use Case**: Typically used when the response from the backend is required immediately for further processing.
   * **Behavior**: Blocks the sequence until the backend response is received.
2. **Send Mediator**:
   * **Purpose**: Sends a message to a backend service asynchronously (non-blocking).
   * **Use Case**: Useful for fire-and-forget scenarios where a response from the backend is not immediately needed.
   * **Behavior**: Does not wait for a response; continues processing immediately after sending the message.

In summary:

* **Call Mediator**: Synchronous, waits for a response.
* **Send Mediator**: Asynchronous, does not wait for a response.

curl -v POST --data @PatientExample.json "http://localhost:8290/healthcare/categories/surgery/reserve" -w "\n"