International Journal of Computer Science and Engineering (IJCSE)
ISSN(P): 2278-9960; ISSN(E): 2278-9979
Vol. 6, Issue 4, Jun - Jul 2017; 1-10
© IASET

International Academy of Science, Engineering and Technology
Connecting Researchers; Nurturing Innovations

RECOMMENDED INTEGRATION PATTERNS FOR COMMON EAI SCENARIOS

SAVEETHA RUDRAMOORTHY

Senior Architect/Middleware Expert, Amadeus Labs, Bangalore, Karnataka, India

ABSTRACT

Enterprise Application Integration (EAI) is a collection of technologies and services which integrates software components, applications, data, and people across the enterprise. The goal of such integration is to provide vendor independent, common façade for seamless communication between applications end to end. Wiring hundreds and thousands of applications in an enterprise requires proper design and implementation guidance. Enterprise Integration Patterns are known design patterns that provide solutions to issues/problems faced during application integration.

This abstract aims in detailing about recommended integration pattern(s) for few common message oriented integration scenarios. Following this, architects and developers can fill the wide gap between the high level vision of integration and actual system implementation.

KEYWORDS: Enterprise Application Integration, Integration Patterns, Message Oriented Middleware (MOM), Channel, Routing, Queue

INTRODUCTION

Integrating enterprise applications via middleware backbone enables one to build a homogenous platform where disparate applications communicate via a common interface. It is essential to take guidance, to build such a complex platform where a huge number of applications are wired via services, data and people. Enterprise Integration Patterns aims in providing design and implementation guidance to all possible classic integration scenarios.

Advantages of integration pattern recommendations are:

- Reduced design and implementation effort, meaning reduced time to market
- Offers standards based system integration, guiding architects to quickly integrate new application
- Reduced integration design and implementation failures, offering potential cost savings
- Reduced design and implementation review re-iterations and ensures successful deployment of loosely coupled integration architecture

There are various integration styles one could follow in order to integrate the applications in an enterprise. Few common styles are File based, Web based, B2B, Remote Procedure Invocations, and Messaging, etc. Most common integration styles in a wide class of enterprise solution is 'Messaging' where applications integrated via Messages over message bus. Hence, in this abstract, we will focus on integration patterns adaptable for Message based Enterprise Application Integration, commonly called as 'Message oriented middleware' (MOM).

Integration via Messaging

Following is the typical flow of messages in any EAI / ESB based solution:

- Messages pushed by sender application (s) is consumed by middleware over a channel. Channels commonly used for message based integration are HTTP, HTTP (S), JMS and JMS over SSL.
- Once the messages are consumed by ESB, messages are processed further in terms of business based orchestration, aggregation, filtering and transformations.
- On top of this basic orchestration, there could be complex integration scenarios like applying rules on repeated events/messages (CEP) or integrated workflow with automated and manual tasks etc.
- Also, any messages flowing into ESB are controlled, monitored and audited for management and reporting purposes

There are different integration patterns guiding above mentioned different steps in message flow. These guides are around:

- *Deciding channels to communicate,* whether it is dedicated channel or shared channel.
- *Message formats*, whether instruction is sent to end application of operations to perform or whether the end application process messages by itself based on business rules
- *Message transformations*, whether enrich or filter the message content
- *Message Routing*, whether the message is routed based on content or route to dedicate list of recipients
- *Message endpoints*, how to dispatch the message to end consumer
- *Overall system management*, which part of message can assist in overall tracking and management

This paper refrains from explaining each integration pattern, rather it focuses on combination of integration patterns one could follow for various integration scenarios.

Enterprise Application Integration Architecture

Typical Application integration Architecture contains the following components:

Message Bus: This is the key component which transfers the messages across MOM components and between providers and consumers as well

Enterprise Service Bus (ESB): Software architecture, providing integration between application and services which are complex in nature. Core feature of ESB can be remembered by acronym ‘VERTO’ – Validation, Enhancement, Routing, Transformation and Orchestration.

Provider and Consumer Applications: End applications who can interface to ESB via SOAP/JMS or ReST/HTTP or SOAP/HTTP.

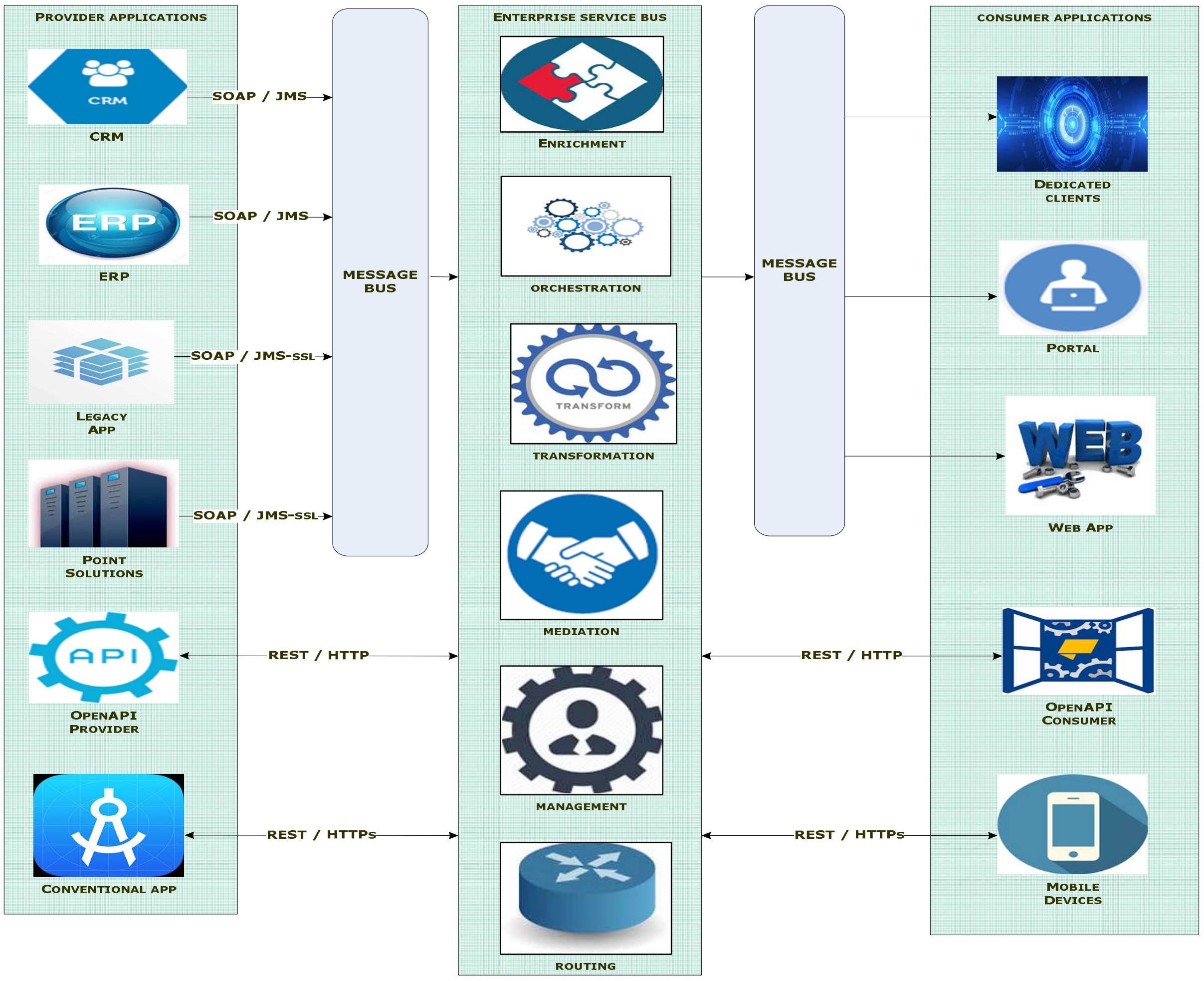


Figure 1: Enterprise Application Integration Architecture

Integration Scenario 1: How to Consume a Message?

Scenario

A typical scenario in any EAI is sending messages via channel to dedicated consumer or to multiple consumers at same time. Following question arises:

- Whether to have dedicated or shared channel across consumer applications?
- Whether to create channels to hold messages per data type?
- If so, what would be naming convention?
- Whether to create specific channel and processing for invalid and dead messages?
- Will the consumer receive the message over channel even when messaging system crashes?
- Will the consumer receive the broadcasted messages whenever he wants?

Pattern Recommendation

There are set of patterns one could follow to design such integration scenario and get the queries answered.

Patterns recommended can be depicted as follows:

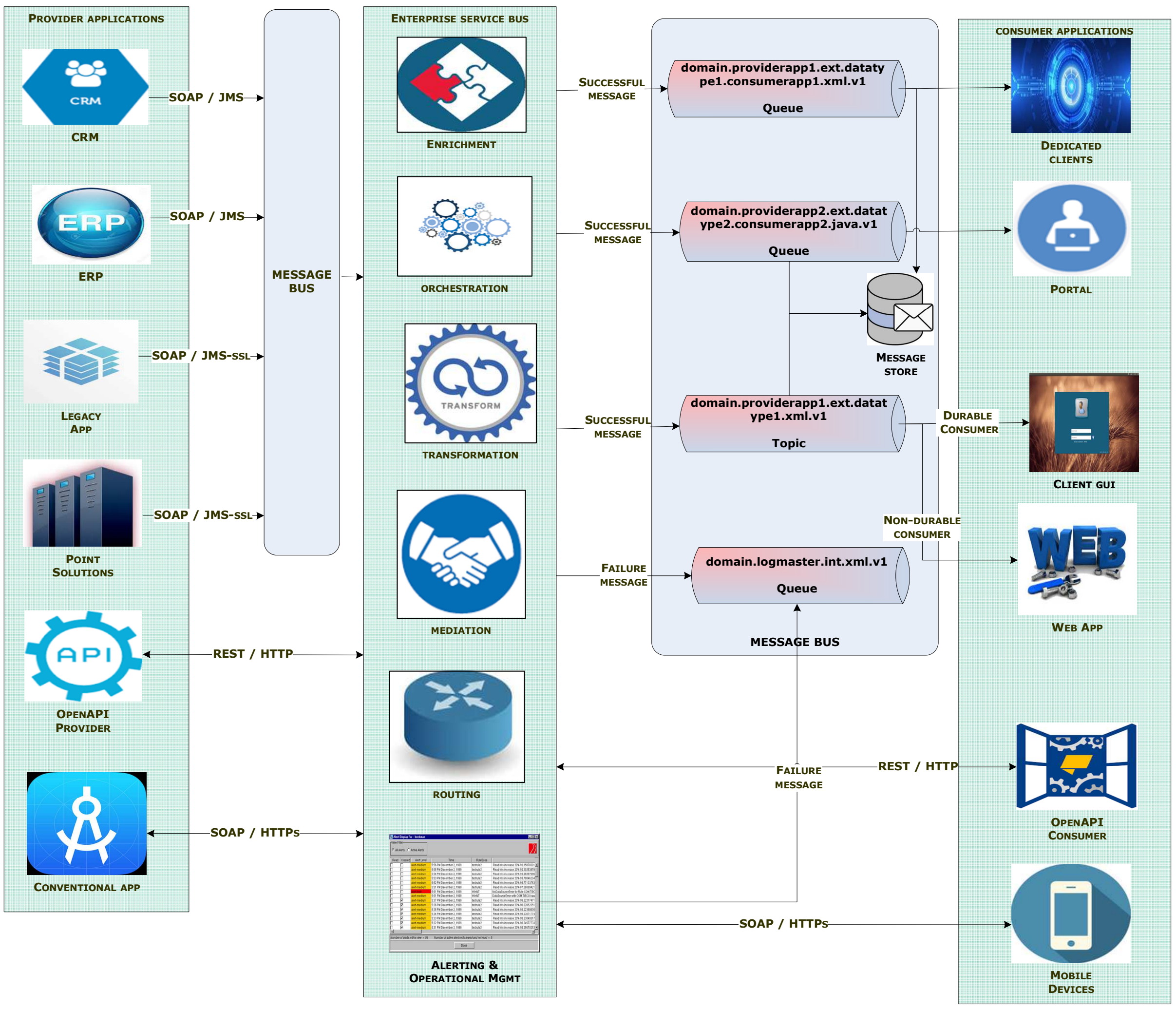


Figure 2: Integration Pattern Recommendation for Channels

If there is dedicated communication to be sent from sender to receiver, then '**Point-to-Point**' channels can be used. Messaging infrastructure used to handle such dedicated traffic is called as 'Queue'. It would be recommended to have dedicated channel per data type, which falls into the category of '**Data type channel'** pattern.

If there is some kind of event which needs to be conveyed to multiple consumers at the same time, then preferred channel to be used in messaging world is 'Topic'. A pattern followed in these scenarios are **'Publish-Subscribe'**. Example cases in Airline domain could be sending flight arrival and departure details to multiple consumers like travel agents, customers, boarding team, baggage team, etc. If a consumer is active during message broadcast, he will listen to the message, otherwise he will lose the message. If the consumer still needs to receive the broadcasted message even when he is inactive, then he has to subscribe to the topic in 'Durable' mode. In this case, messaging infrastructure persists the messages on behalf of the consumer and pushes the messages whenever consumer come alive, hence ensuring '**Guaranteed'** delivery.

A typical naming convention for these channels are:

<<Domain>>.<<ProviderApp>>.<<External/Internal>>.<<DataType>>.<<ConsumerApp>>.<<MessageType>>.<<Version>>
Example: airline.reservation.external.resData.callcenter.xml.v1
Note: External channels are used for communication with external applications. Internal channels are used for communication within ESB and are specific to ESB.

Middleware business is very critical, losing a message because of processing issues is not acceptable. Hence, the recommendation here is to design **'Invalid Message Channel'**. The very purpose of this channel is to handle all failure messages. Whenever there are processing failures throughout the ESB infrastructure, push those complete payloads including failure reasons into these dedicated channels. Special processing can be built on top of this invalid message so that functional and technical failures can be handled intelligently.

Integration Scenario 2: How to Construct a Message?

Scenario

The message is the backbone of any integration solution. It is critical for architects to know about the ideal structure of a message which can be used in EAI, wherein the following questions are answered:

- *Message intent* – Sender letting the receiver know what is the intention of the message sent
- *Returning a response* – To let the receiver know whether the sender is expecting the return back. If so, what is the return address?
- *Huge amounts of data* –Sequencing the messages in order to support sending huge message structure
- *Slow messages* – Ensuring receiver not receiving the messages after expiry, which means messages could be meaningless for the business

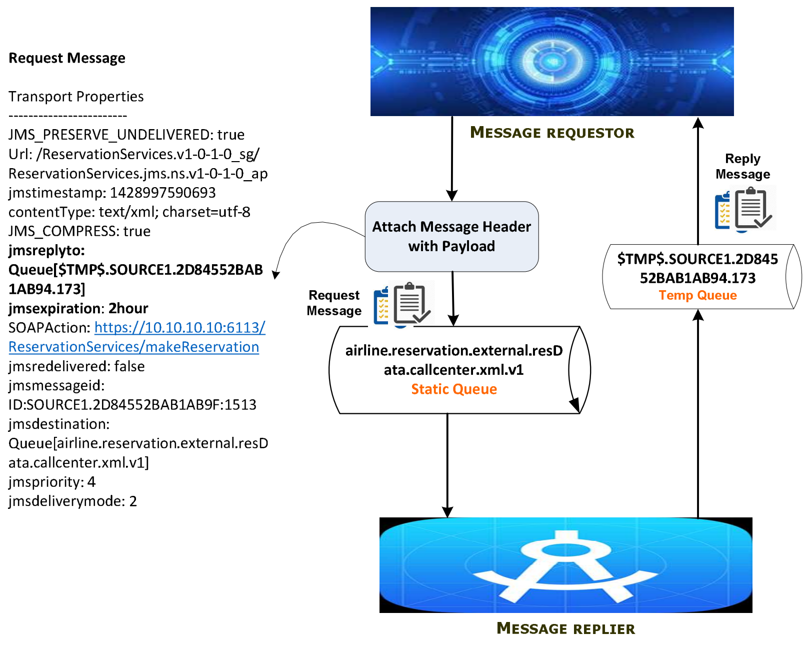
Pattern Recommendation

Sender led execution - If the integration scenario is more synchronous in nature, like SOAP messages sent to execute particular functionality, then **Command message** pattern can be used.

Receiver led execution - **Document messages** can be used when sender passes a complete document and receiver determines the functionality to perform based on business requirements.

Event messages are asynchronous in nature where notifications are sent on a channel where all the listeners hook and consume. The typical use case for applying this pattern is when a Notification message has to be sent to all consumers when platform maintenance is going on.

Request-Reply pattern is recommended for all secure synchronous communications. Here, Requestor and Replier uses their own channels to securely communicate messages over dedicated channel. Requestor always receives reply from consumer for every request he sends. If message queues are used for this communication, then Requestor passes the reply queue name (could be temporary queue) in the message header as follows:



Request Message

Transport Properties

JMS_PRESERVE_UNDELIVERED: true
Url: /ReservationServices.v1-0-1-0_sg/ReservationServices.jms.ns.v1-0-1-0_ap
jmstimestamp: 1428997590693
contentType: text/xml; charset=utf-8
JMS_COMPRESS: true
jmsreplyto: Queue[TMP.SOURCE1.2D84552BAB1AB94.173]
jmsexpiration: 2hour
SOAPAction: https://10.10.10.10:6113/ReservationServices/makeReservation
jmsredelivered: false
jmsmessageid: ID:SOURCE1.2D84552BAB1AB9F:1513
jmsdestination: Queue[airline.reservation.external.resData.callcenter.xml.v1]
jmspriority: 4
jmsdeliverymode: 2

MESSAGE REQUESTOR
Reply Message
Attach Message Header with Payload
TMP.SOURCE1.2D84552BAB1AB94.173
Temp Queue
Request Message
airline.reservation.external.resData.callcenter.xml.v1
Static Queue
MESSAGE REPLIER

Figure 3: Request / Reply Integration

Typically middleware message bus won't serve as indefinite storage. Rather messages on message bus should be consumed by consumer as soon as it arrives. There are situations where consumers may not be active and online always. In those cases middleware can retain messages for specific duration after which messages can be expired. **Message Expiry** parameter can be set on channel in order to instruct the messaging infrastructure to retain the messages only for specific time interval, post which those can be dropped.

airline.reservation.external.resData.callcenter.xml.v1
Queue Properties: secure,failsafe,maxbytes=50MB,overflowPolicy=discardOld,expiration=2hour
Queue properties indicates that any message on queue can be persisted for 2 hours. Post that messages present in the queue for more that 2 hours are discarded (FIFO).

Integration Scenario 3: Where to Route a Message?

Scenario

The value of an integer is obtained only when the applications are loosely coupled and when messages are routed between loosely coupled applications. Following queries need an answer when one attempt to route the message to destinations:

- How can I determine that application route the messages to the right destination?
- Can I send the messages 'As-is' and send it or Do I need to strengthen the message?
- Do I send right set of information while I truncate the message for bandwidth reasons?

Pattern Recommendation

There could be a scenario where during build time ESB don't know where to route the message or business requires dynamic routing based on message content. Key pattern playing an important role here is '**Content based Router'**. Simple set of rules can be stored to derive these routing end points based on message content. Few pseudo rules could be

- **If message source is ‘AppX’ or ‘AppY’, then route to End Point X**
- **If message carries end point URL, then route to End Point URL as mentioned**
- **If message header contains backend Id, then route to specific backend**
- **If Payload size > ’X’ MB, then route to backendInstance A else route to backendInstance B**

Another common integration scenario designer could face is to invoke an end application based on the results from other invocations. Input is resultant **aggregated** from multiple invocations. Also, ESB, **Composed message processor** splits the message, routes the sub-message to different processing units and aggregates the responses into a single message. This can be notated as follows:

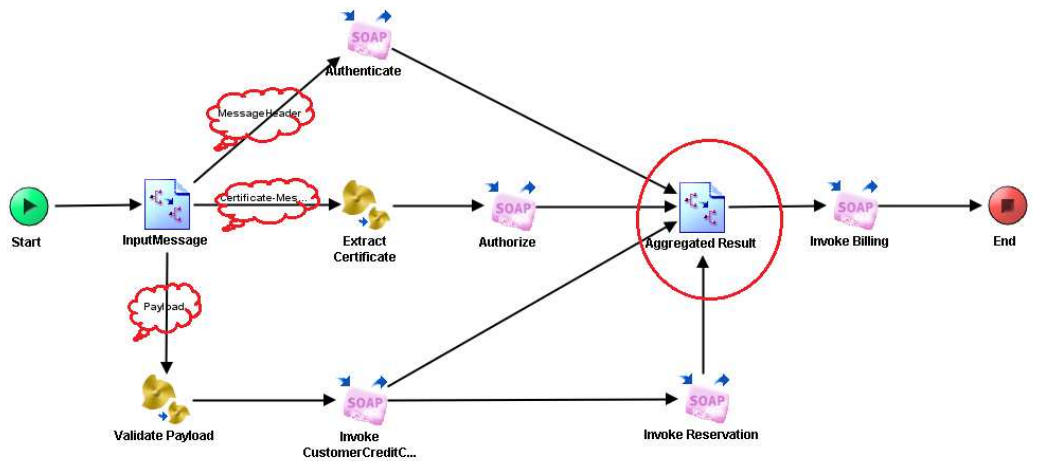


Figure 4: ESB as Composed Message Processor

Here, multiple calls are made in order to derive aggregated input for final invocation. Also, portion of messages are split and processed before making subsequent invocations.

Integration Scenario 4: How to Qualify Messages for Consumption?

Scenario

Message from providers are not always consumed in the same format by consumers. Here comes role of ESB to convert message into the format consumable by end consumer. Key questions for an integrator here are:

- How can I wrap the message in such a way that some portion is used by the ESB for management and rest of payload consumed by end application?
- Whether message needs some enhancement or pruning before reaching the consumer?
- The data model can be normalized in a way that all consumers receive same domain specific data structure?

Pattern Recommendation

Most of the messages involved in enterprise integration contain Header and Body. Mostly header information is of interest in Messaging infrastructure, wherein the end application is interested only on actual payload or message body. The ideal way to pass the complete message including the header is to wrap all parts of messages into **an envelope**.

Enriching the message: There could be cases where consumer application wanted a detailed information wherein provider application can provide message summary. In such cases, it is the responsibility of an ESB layer to enhance this message before it reaches the consumer.

An ESB can enhance the message in any of the following ways, keeping '**Content Enrichment'** as a goal:

- Obtain a set of pre-built information from database or file
- Invoke another application via web service and obtain the results
- Reach out to 3rd party open API services using ReST API and obtain the results
- Send out the message summary to another ESB application and obtain details etc

Filtering the message: Alternatively, end consumer may need message summary wherein provider can only supply only detailed information. In such cases, ESB has to **filter** few elements of the message in order to obtain end result.

Following picture depicts all these patterns in action:

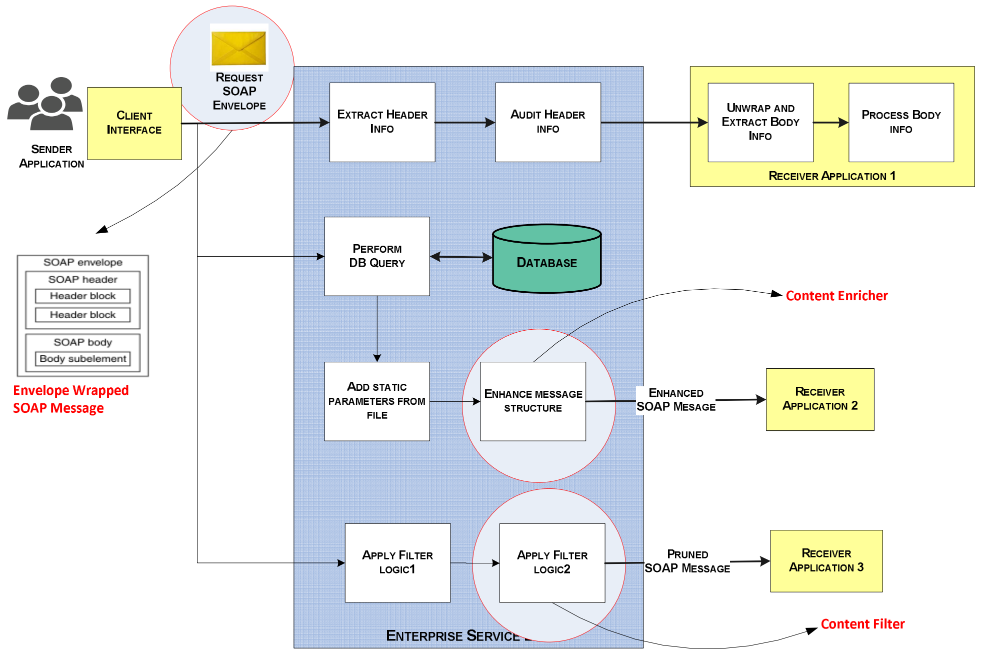


Figure 5: Message Transformation Patterns in Action

Integration Scenario 5: How Can a Message Be Monitored and Controlled?

Scenario

Now that ESB ensure message construction and delivery, how can we ensure that messages are managed and enable operational intelligence? Key management aspects are:

- To administer the data passed across the ESB and applications end to end
- How to obtain sample set of messages which is passing through ESB for debugging purposes?
- How to create alternative flows for operation monitoring wherein main business flow is not impacted?
- Do we need to check the application and component status by sending 'Heartbeat' messages?

Pattern Recommendation

Message Detour: In a process orchestration, one could design set of flows. Main flow means for message validation, transformation and routing and other flows are meant for debugging and monitoring purposes. The **Detour** routes a message through intermediate steps to perform validation, testing, or debugging functions.

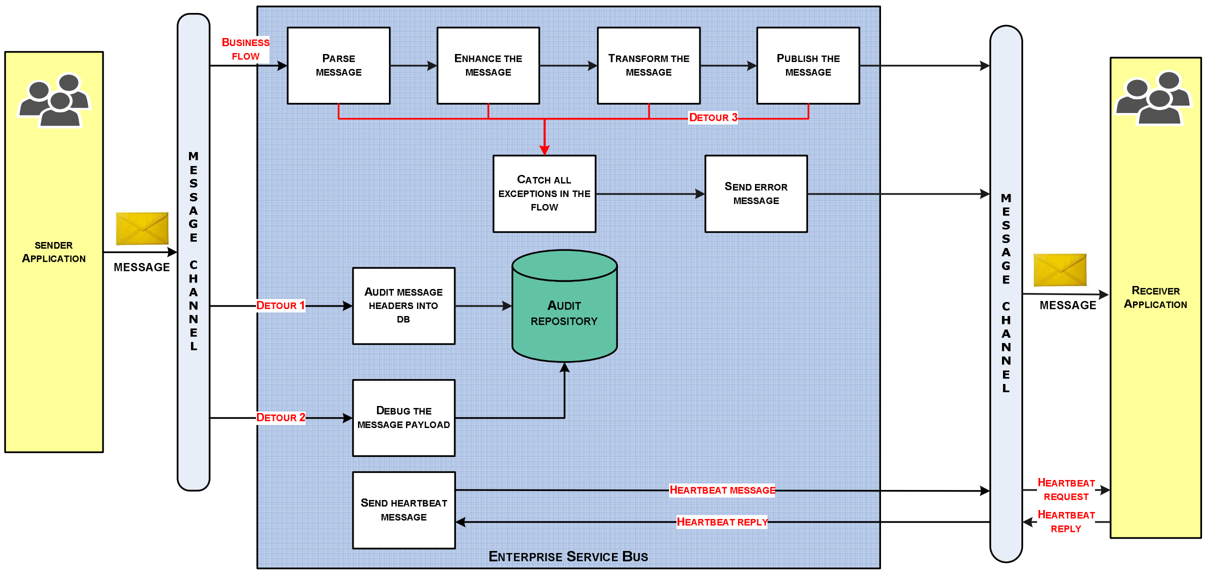


Figure 6: Patterns Managing Message Flow

As highlighted in one of the sample process above, there are 4 Detour states.

- Main flow, which orchestrates business activities.
- Detour flow1 which audits the message headers for key tracking purposes.
- Detour flow2 which debugs the message payload
- Detour flow3 which catches all the exceptions occurred in the message orchestration and report accordingly

Wiretapping Messages: In the production environment, typically messages were not allowed to be tapped. But, in non-productive environments, whenever messages need to be inspected then ideal way to deal with this scenario is to wiretap the messages into an additional temporary channel. This wiretapping can be done by constructing routes and bridges between channels. Routes are constructed between channels if messages have to be transferred across multiple messaging servers via same channel type (between queue to queue or topic to topic). Bridges can be constructed between channels of same server, between queue to the topic (or) topic to topic (or) topic to queue (or) queue to queue.

Heartbeat Test Messages: The Test Message pattern ensures the health of middleware components is in check by preventing situations such as garbling outgoing messages due to an internal fault. “Heartbeat” messages can be sent to components to check the health state of those components and end applications. Result of heartbeat check can be wired to any monitoring and alerting systems.

CONCLUSIONS

With the assistance of core integration patterns mentioned in this white paper architects and integration designers can decide on suitable pattern combinations for their integration scenarios.

Proper application of these patterns in their design and implementation will lead to

- Successful design and implementation of different integration scenarios
- Guidance card to design various simple and complex integration scenarios
- Reduces common mistakes during design phase to a greater extent
- Quickly finalize the design decisions
- Reduces implementation flaws and reduces the gap between EAI design strategy and implementation.

REFERENCES

1. http://www.enterpriseintegrationpatterns.com/
2. https://docs.wso2.com/display/IntegrationPatterns/Enterprise+Integration+Patterns+with+WSO2+ESB
3. Gregor Hohpe and Bobby Woolf, "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions 1st Edition"

AUTHOR DETAIL

Saveetha Rudramoorthy received the B.E. Degree from the University of Madras, India, in 2000. Saveetha is a Senior Architect, Middleware Expert with Amadeus Labs based out of Bangalore, Karnataka, India. She is Tibco and WebMethods certified with over 16 years of experience in enterprise integration and middleware across domains like Telecom and Airline.