wipro

Integration Patterns

Design pattern overview – integration services

Design pattern is a standard solution for common and recurring problems. Design patterns are an effective way to avoid the expensive process of reinventing, rediscovering and revalidating agnostic software artifacts.

Following are some of the key integration patterns that can be followed as a part of integration engagements

Canonical schema pattern

The Canonical schema pattern helps to limit and structure the mapping code required. Canonical means define one and only or the official format for each enterprise business object which contains fields from all the different businesses available in the enterprise. This pattern is applicable in EAI (Enterprise Application Integration) or SOA (Service Oriented Architecture) scenarios

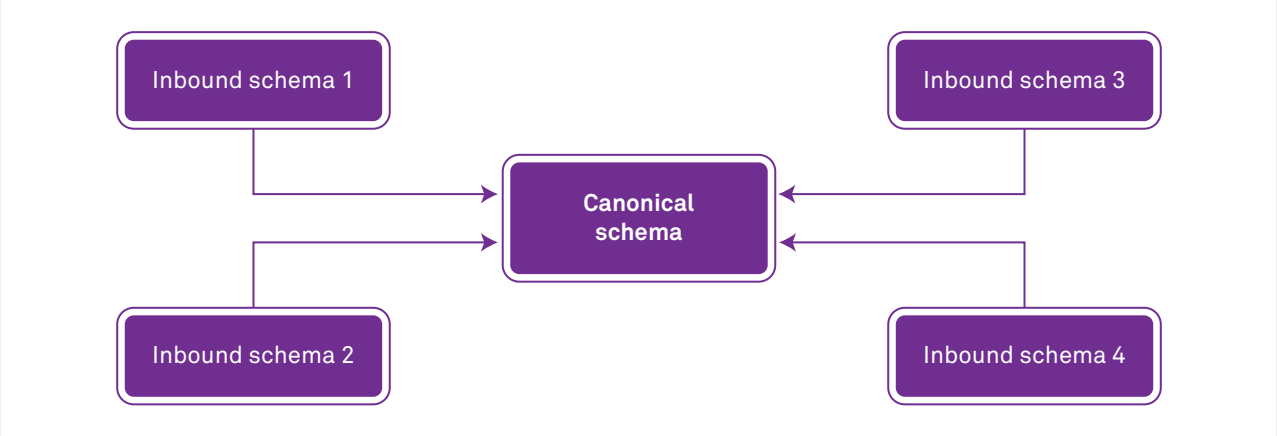

Figure 1: Canonical schema pattern

Content enrichment pattern

Content enrichment pattern enriches the content of received messages for e.g. If the message originator does not have all the required data items available within it, this pattern is used to collect additional information by accessing an external data source. The detailed step required to implement this pattern includes the following steps:

- Construct a request message
- Send the request message to the external source (database, web service, WCF service etc.)
- Receive a response from external source
- Combine the response from external source with the original message

Example: Retrieve data from a database and add the data in a given location in a message

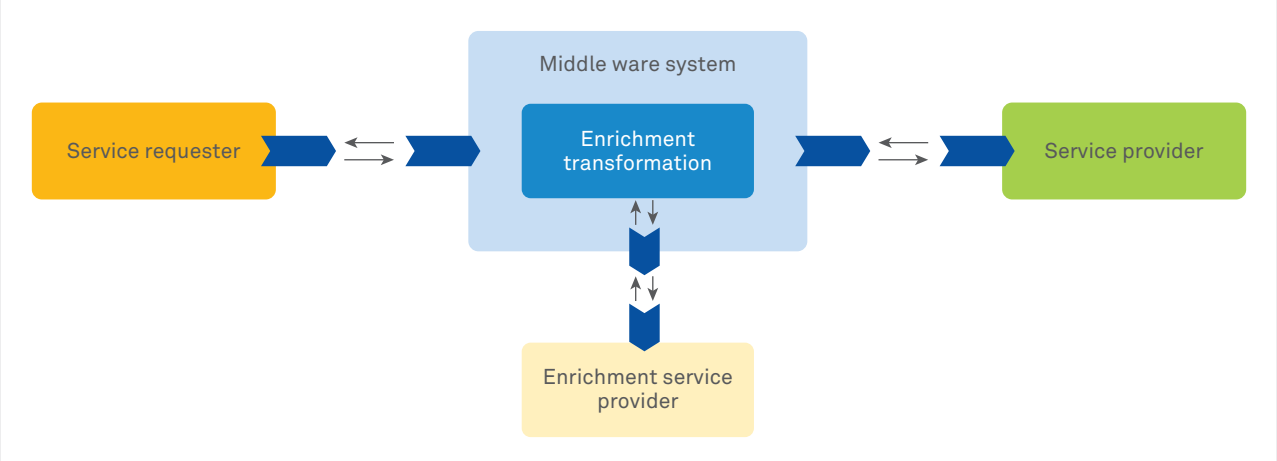

Figure 2: Content enrichment pattern

Content-based router pattern

Content-based routing routes the message based on the content of the message, rather than by an explicitly specified destination. The Content-based router pattern checks the content of the received message and then uses these details to get the routing information for the message. Based on the routing information, a different channel is available to send the data.

There can be a number of criteria such as existence of fields, specific field values, etc. for routing the message. To implement the Content-based router pattern, values must be available at the run time to decide the destination of the message. In this pattern, usage of business rules and dynamic send port configuration plays a key role.

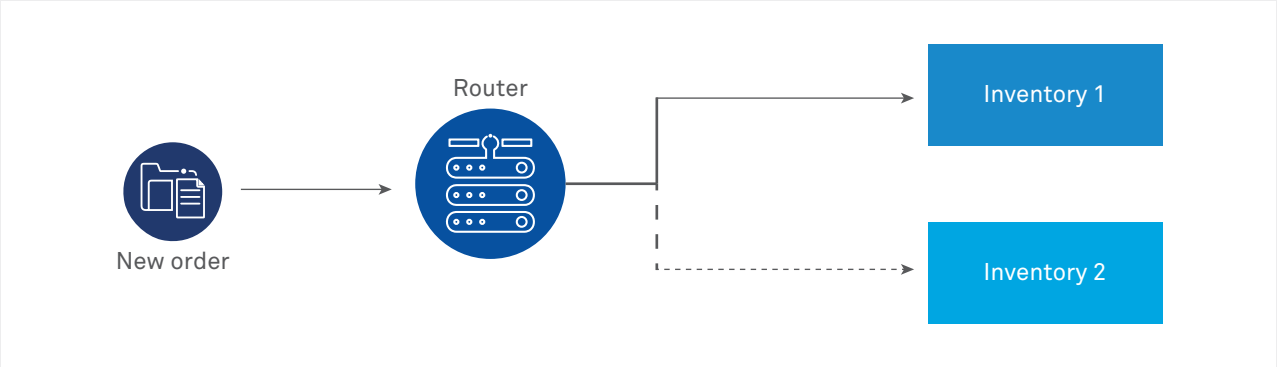

Figure 3: Content-based router pattern

Correlation identifier pattern

In this pattern, each message has a unique identifier attached with the request message. The unique identifier is either generated after the message is received (for e.g. Message ID, GUID (Globally Unique ID)) or should be a part of the request message (for e.g. Business key like Order ID or any primary key through which the message can be identified). Once the message is received by the destination, it fetches the unique identifier and copies that ID to the response message. And finally, the source system can match the ID with the request and response.

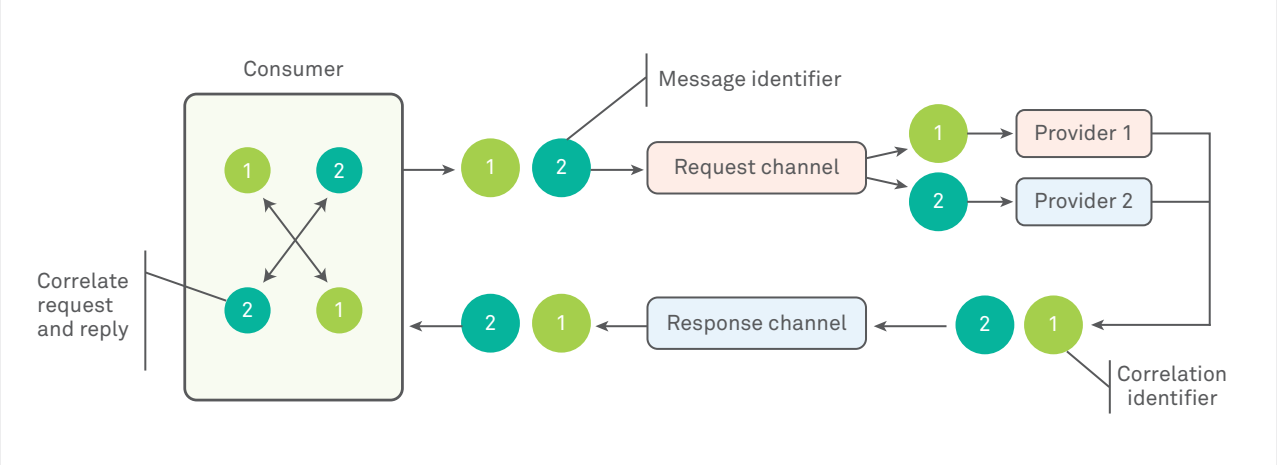

Figure 4: Correlation identifier pattern

Request-reply pattern

This pattern is based on the service provider and consumer. As the name indicates, it is asynchronous messaging between the source and target. The request message should be sent through unidirectional channels. To complete the request and response flow, two asynchronous point-to-point channels should be maintained. The message format is different for request and reply.

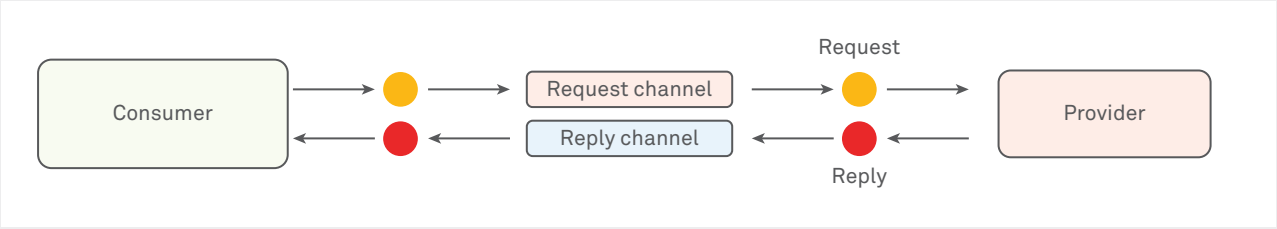

Figure 5: Request-reply pattern

Error handling pattern

Error handling pattern is used to automate the message failure case. It should be developed and used as a common reusable error handling component. This provides the flexibility to send a message to the designated location in case of failures. Failed messages can be routed to the subscribing port for reporting or processing. This can also be used for alert. This pattern is mainly used in any one of the below cases:

- Routing failed messages in ports
- Catching failed messages with send port
- Catching failed messages with orchestration
- Handling exceptions inside orchestration

Suspend with retry pattern

The suspend with retry pattern enables the orchestration to suspend a message when there is an error. The suspension occurs within a loop so that the orchestration suspends, asks for intervention, and then retries the operation a fixed number of times.

Splitter pattern

The splitter pattern splits the received message into multiple messages. Each of the individual messages after split will be processed separately.

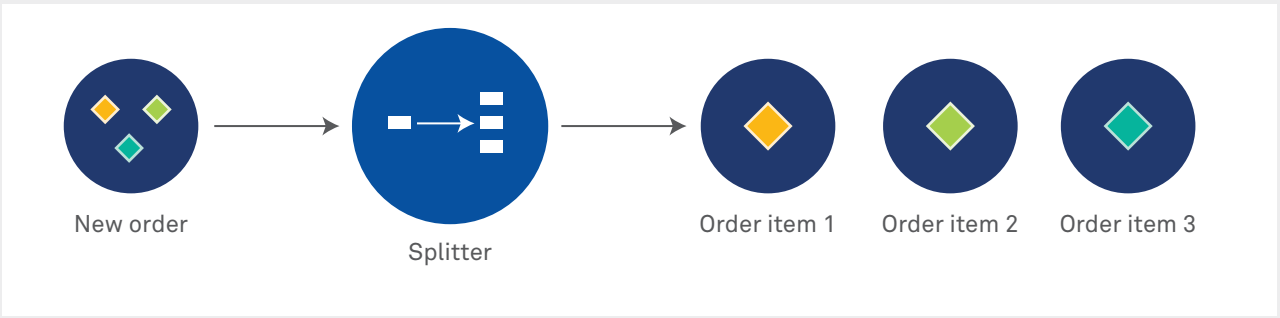

Figure 6: Splitter pattern

Sequential and parallel convoy:

A sequential convoy is a set of related messages that have a predefined order. In a sequential convoy, all the types of messages are received in an ordered sequence. The first receive shape in the orchestration initializes the correlation set and the other receive shapes in the orchestration follow the correlation set.

Steps to implement via BizTalk:

Sequential convoys can be implemented by using the "sequential correlated receives" messaging design pattern in BizTalk Server.

Create the orchestration with the first receive shape activate property to true and other as false. In the first receive shape, initialize the correlation set so when the first message is received, this starts a new orchestration instance and initializes a correlation set.

Parallel convoy is useful when the same type of messages is supposed to be received from multiple locations and sources. The sequence of messages to be received is also not known. However, the processing of messages starts only when all the required messages will be received by the orchestration.

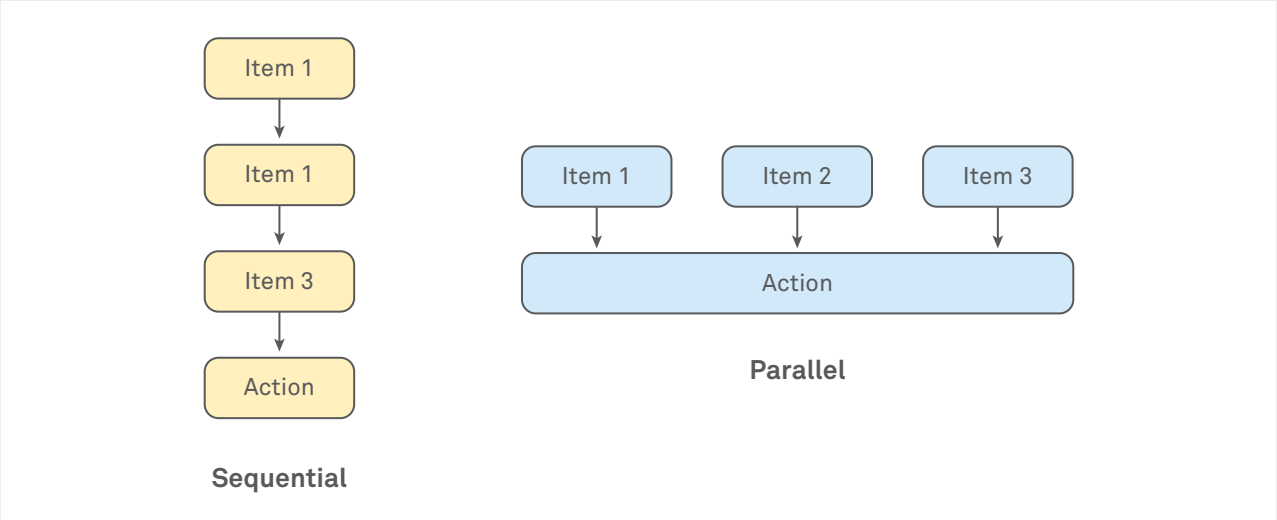

Figure 7: Sequential and parallel convoy

Scatter and gather

This pattern is a combination of the splitter pattern and the aggregator pattern. So it can be implemented by using the aggregator pattern to assemble the results from using the splitter pattern and put them under a parallel actions shape. The scatter and gather pattern is a method for broadcasting and processing messages in parallel. The "scatter" portion distributes a series of messages all at the same time, and then an aggregator "gathers" the messages back into the main response before continuing

About the author

Saikat Kundu
Practice Manager, MAS AED - Microsoft Presales,
Wipro Limited.

Saikat is a TOGAF certified architect and Microsoft Certified Professional (MCP), with 15+ years' experience in analysis, design, development, testing, support and maintenance of enterprise applications in Microsoft cloud and native technologies for global clients. He currently plays the role of Practice Manager in the Microsoft presales team for the O365/Azure/SharePoint/BizTalk/.NET cluster.

Wipro Limited
Doddakannelli, Sarjapur Road,
Bangalore-560 035,
India

Tel: +91 (80) 2844 0011
Fax: +91 (80) 2844 0256
wipro.com

Wipro Limited (NYSE: WIT, BSE: 507685, NSE: WIPRO) is a leading global information technology, consulting and business process services company. We harness the power of cognitive computing, hyper-automation, robotics, cloud, analytics and emerging technologies to help our clients adapt to the digital world and make them successful. A company recognized globally for its comprehensive portfolio of services, strong commitment to sustainability and good corporate citizenship, we have over 175,000 dedicated employees serving clients across six continents. Together, we discover ideas and connect the dots to build a better and a bold new future.

For more information, please write to us at **info@wipro.com**

IND/B&T/JUN-NOV2019