Integration Platform: An **Integration Platform** is defined as a computer software which integrates different applications and services.

An **Integration Platform** tries to create an environment in which engineers can:

- Data (information) Integration: Ensure that they are using the same datasets and can share information. Data management with metadata information and versioning ensures the data is kept consistent.
- Integrate many kinds of applications (independent from platform, programming language or resource) so they can be bound together in workflows and processes to work in conjunction. The different interfaces are hidden by the usage of a uniform interface in the **Integration Platform** (Process Integration).
- Collaborate between distributed and scattered applications and engineers over the network.
- Interoperability between different operating systems and programming languages by the use of similar interfaces.
- Take security considerations into account so that, for example, data is shared only with the right resources.
- Visual guidance by interactive user interfaces and a common facade for all integrated applications.

Common components of integration platform

Integration platform typically contains a set of functional components, such as

- Message bus for enabling reliable messaging between enterprise applications.
- Adapters to transform messages from and to application's proprietary protocol. Adapters often offer connectivity via common standards, like FTP, SFTP or format support, like EDI.
- Transformation engine and visualized data mapping to transform messages or files from one format to another.
- Metadata repository for storing information separated from processes, like business party.
- Process Orchestration Engine for orchestration design and execution. In this context orchestration is a technical workflow that represents a business process or part of it.

- Technical dashboard for tracking messages in a message bus and viewing execution history of orchestrations.
- Scheduler for scheduling orchestrations
- Batch engine for controlling large file transfers, batch jobs, execution of external scripts and other non-messaging based tasks.

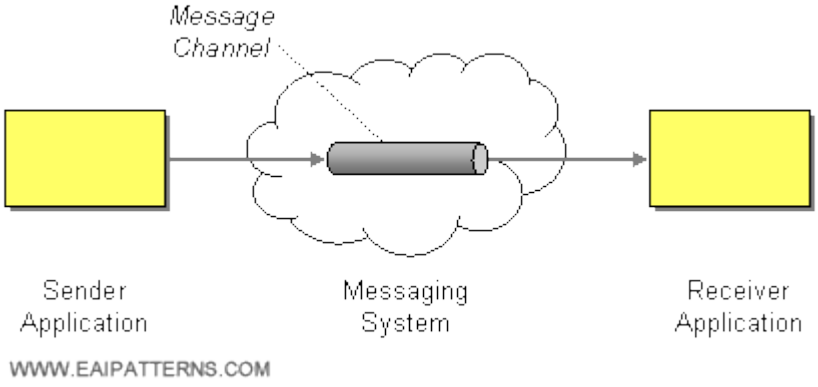
EIP(Enterprise integration patterns):

- As the name implies, these are tested solutions for specific design problems encountered during many years in the development of IT systems. And what is all the more important is that they are technology-agnostic which means it does not matter what programming language or operating system you use.

  Patterns are divided into seven sections:

  1. Messaging Systems,
  2. Messaging Channels,
  3. Message Constructions,
  4. Message Routing,
  5. Message Transformation,
  6. Messaging endpoints,
  7. System management.

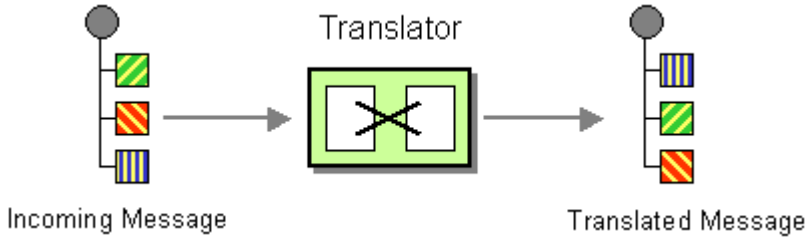
  Message Channel (from Messaging Systems)



  A message channel is a logical channel which is used to connect the applications. One application writes messages to the channel and the other one

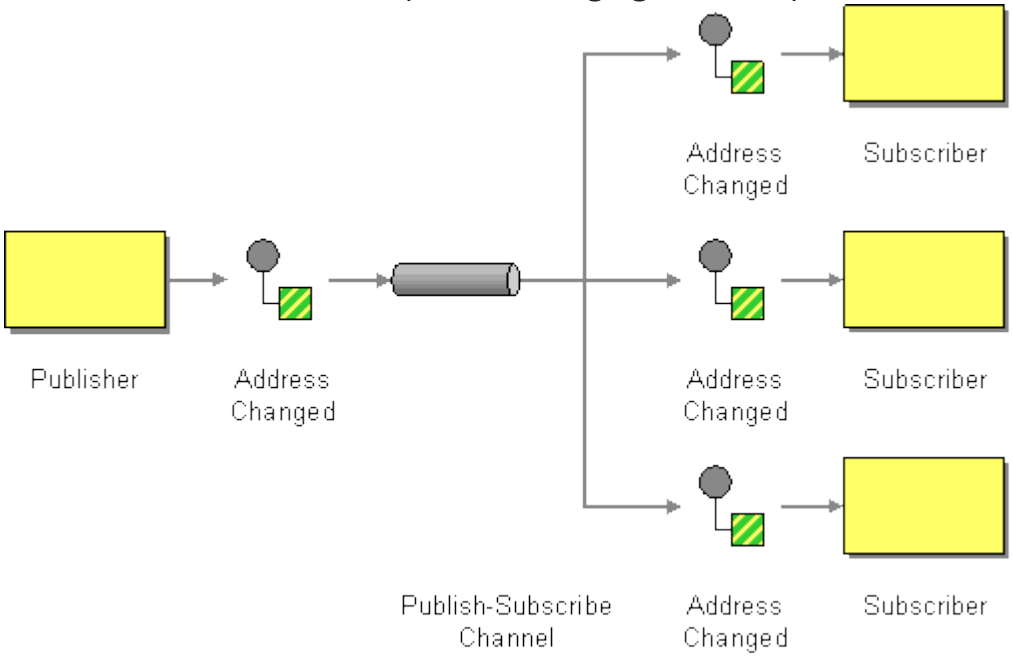
(or others) reads that message from the channel. Message queue and message topic are examples of message channels.

Message Translator (from Messaging Systems)



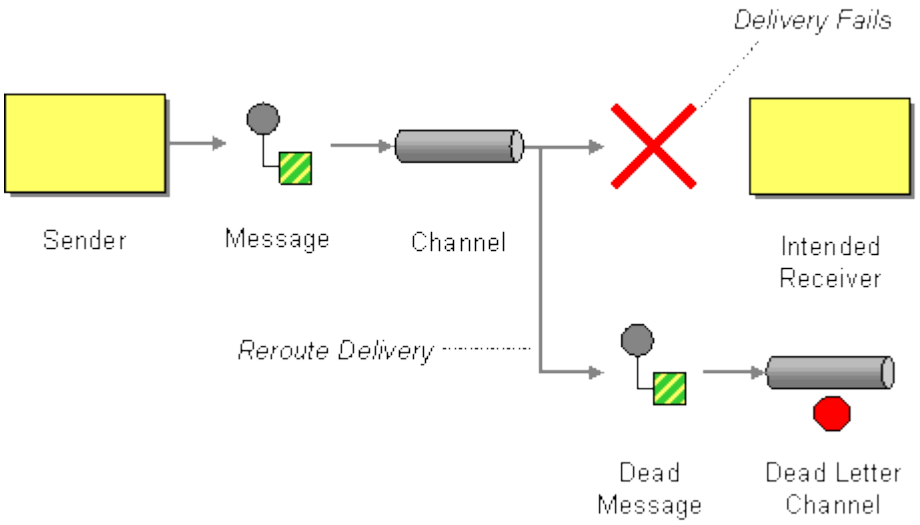
Message translator transforms messages in one format to another. For example one application sends a message in XML format, but the other accepts only JSON messages so one of the parties (or mediator) has to transform XML data to JSON. This is probably the most widely used integration pattern.

Publish-Subscribe Channel (from Messaging Channels)



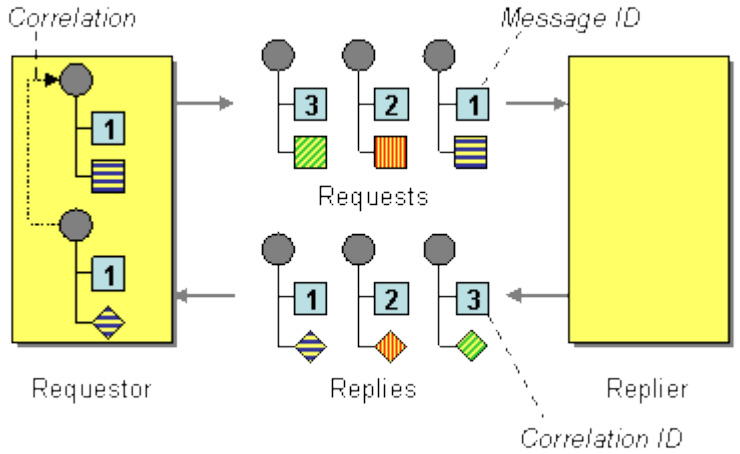
This type of channel broadcasts an event or notification to all subscribed receivers. This is in contrast with a point-to-point channel . Each subscriber receive the message once and next copy of this message is deleted from channel. The most common implementation of this patter is messaging topic.

Dead Letter Channel (from Messaging Channels)



The Dead Letter Channel describe scenario, what to do if the messaging system determines that it cannot deliver a message to the specified recipient. This may be caused for example by connection problems or other exception like overflowed memory or disc space. Usually, before sending the message to the Dead Letter Channel, multiple attempts to redeliver message are taken.

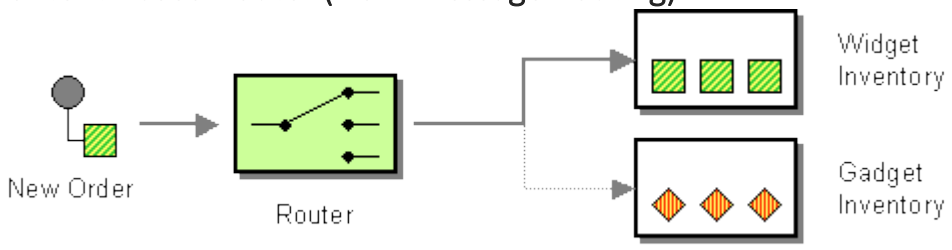
Correlation Identifier (from Message Construction)



Correlation Identifier gives the possibility to match request and reply message when asynchronous messaging system is used. This is usually accomplished in the following way:
Producer: Generate unique correlation identifier.
Producer: Send message with attached generated correlation identifier.
Consumer: Process messages and send reply with attached correlation

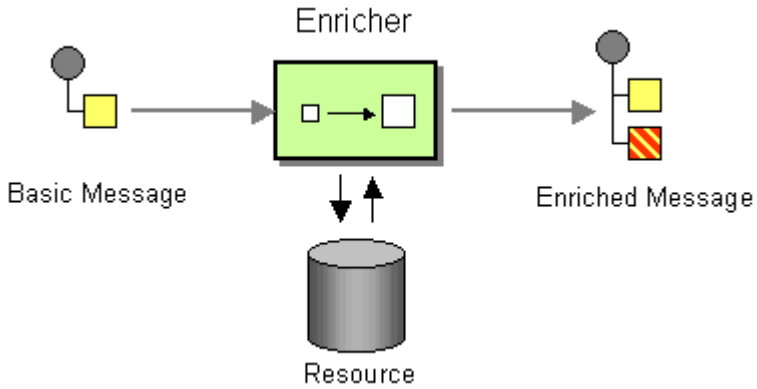
identifier given in request message.
Producer: Correlate request and reply message based on correlation identifier.

Content-Based Router (from Message Routing)



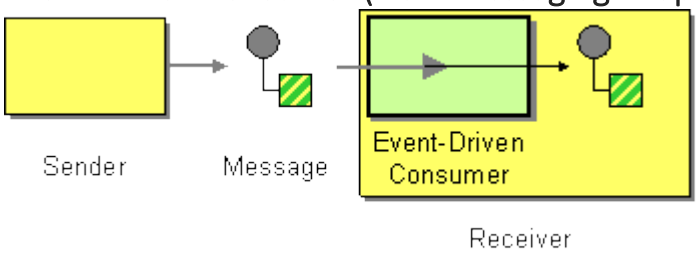
Content-Based Router examines message contents and route messages based on data contained in the message.

Content Enricher (from Message Transformation)



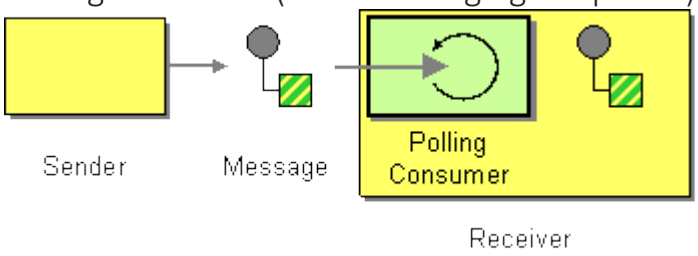
Content Enricher as the name suggests enrich message with missing information. Usually external data source like database or web service is used.

Event-Driven Consumer (from Messaging Endpoints)



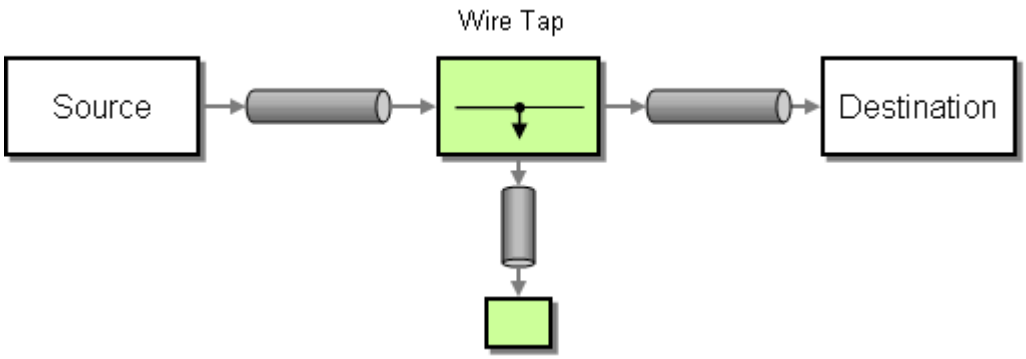
Event-Driver Consumer enables you to provide a action that is called automatically by the messaging channel or transport layer. It is asynchronous type of pattern because receiver does not have a running thread until a callback thread delivers a message.

Polling Consumer (from Messaging Endpoints)



Polling Consumer is used when we want receiver to poll for a message, process it and next poll for another. What is very important is that this pattern is synchronous because it blocks thread until a message is received. This is in contrast with a event-driven consumer. An example of using this pattern is file polling.

Wire Tap (from System Management)



- Wire Tap copy a message and route it to a separate channel, while the original message is forwarded to the destination channel. Usually Wire Tap is used to inspect message or for analysis purposes.

	WSO2 ESB and SOA Platform	Mule ESB	FuseSource ESB	Adroit Logic UltraESB	JBoss ESB and SOA Platform	Tibco ActiveMatrix
Supports Enterprise Integration Patterns	Yes	Yes	Yes	Yes	Yes	Yes
Delivers all required ESB features (i.e. web services, message transformation, protocol mediation, content routing)	Yes	Yes	Yes	Yes	Yes	Yes
Offers a complete and cohesive SOA Platform (i.e. ESB, Message Broker, Governance Registry, Business Process Server, Data Services Server, Application Server)	Yes	No	No	No	Yes	Yes
SOA Governance	Yes	No	No	No	No	Yes
Graphical ESB Development Workbench	Yes	Yes	Yes	No	Yes	Yes
Based on a composable architecture	Yes	No	No	No	No	No
Cloud integration platform offering (iPaaS)	Yes	Yes	No	No	No	Yes
Cloud Connectors and Legacy Adapters	Yes	Yes	No	No	Yes	Yes
Performance	High	Moderate	Moderate	High	Moderate	High
Security and Identity Management	Yes	Limited	Limited	Limited	Limited	Limited
Open Business Model	Yes	Yes	Yes	Yes	No	No

Why I should use Mulesoft?

MuleSoft's Anypoint Platform offers a number of tools and services, including:

- Mule ESB: a platform that allows developers to connect SaaS and on-premises applications using pre-built connectors, integration templates and drag-and-drop tools.
- API Designer: a web-based tool for creating APIs that also includes a console and a JavaScript scripting notebook. It also allows users to share their API design to receive feedback from other users.
- API Manager: an API management tool that allows organizations to manage users, traffic, service-level agreements and API security. The API Manager also includes API Gateway -- an API security service that runs on-premises or in the cloud -- along with the API Policy Manager and API Contract Manager.
- Anypoint Studio: a graphical design environment for building, editing and debugging integrations.
- API Portal: a developer portal offering interactive documents, tutorials and code snippets. Also included in the API Portal are MuleSoft's Portal Designer, Developer Onramp and Access Controller tools.
- API Analytics: an analytics tool that allows users to track API metrics, such as performance and usage. The API Analytics service also includes visualization tools, such as API Dashboards and API Charts.
- CloudHub: a multi-tenant integration platform as a service that connects SaaS applications and on-premises applications. The iPaaS includes a hybrid deployment option, disaster recovery and high availability.
- Mule Enterprise Management: a management tool for servers, workflows and endpoints

Spring Integration VS Camel VS Mule:

Spring Integration

- The APIs of Spring Integration framework is known as adapters. Written in Java, this framework is light in weight and has an event-driven architecture. Easy to learn, this framework supports various Java environments including FTP, HTTP, JMS, and TCP. Even though spring integration is considered easy, with more than 3000 classes it is repelling the developers by not providing a clear path.

Also, it has a lot of dependency on XML due to which the coding process requires huge amount of time.
- With undocumented approach, Spring Integration is considered as both time-consuming and complex.

Mule ESB

- As the name itself suggest, this product is not only an integration framework but also an ESB, which stands for Enterprise Service Bus. Based on Java, mule is highly scalable as it comes with stage-driven architecture and is best suited for robust application integrations.
- Operating environments, the transport protocols used are never the concerning issues for mule, as it provides unparalleled support throughout the process. Also, one of the main advantages of Mule is the various connectors which are offered. But on the damper side, mule does not support OSGI.
- Mule ESB is not considered as the best option for organizations, whose project is limited to HTTP or web services. Because ongoing with Mule ESB the complexity may increase staggeringly.

Apache Camel

- Apache Camel is comparable to Mule ESB in numerous aspects. The best highlighting feature of this framework is, it supports multiple DSLs. Be it Java, Scala, Groovy or Spring XML, Apache Camel ensures unconditional unified performance. Also, organizations where the integration between applications is required that have dissimilar data formats and protocols then Apache Camel is the best possible option.

Need for ESB:

- Point-to-point communication normally has issues with scalability. These issues are further compounded with increased systems. ESB, a middleware technology, is a Bus-like architecture used to integrate heterogeneous systems. In ESB, each application is independent and yet able to communicate with other systems. It, thus, prevents scalability issues and ensures that communication happens only through it.

ESB's guiding principles are:

- Orchestration - integrates two or more applications and services to synchronize data and process.
- Transformation - transforms data from canonical to application specific format.
- Transportation - protocol negotiation between multiple formats like HTTP, JDBC, JMS, and FTP etc.
- Mediation - multiple interfaces for supporting multiple versions of a service.
- Non-functional consistency - transaction management and security.

When to use ESB architecture

The first step when opting for ESB architecture is to map its value to requirements. Given below are some usage guidelines:

- When system integration points grow beyond two, with additional integration requirements.
- When using multiple protocols such as FTP, HTTP, Web Service, and JMS etc.
- When there is a requirement for message routing based on message content and similar parameters.

ESB architecture Implementation Rules

- Messaging services like JMS can be used to de-couple applications
- XML format when canonical data is used for communication
- Using an adapter which is responsible for marshalling and un-marshalling data. The adapter is also responsible for communicating with the application and bus. It is then used to transform data from application format to bus format
- Non-functional activities like securities, transaction management are also performed by the adapter in ESB.
- To summarise, ESB provides for a flexible architecture. It enables multiple application communication and provides easy integration with other systems.

ESB selection checklist

Mule and other ESBs offer real value in scenarios where there are at least a few integration points or at least 3 applications to integrate. They are also well suited to scenarios where loose coupling, scalability, and robustness are required.

Here is a quick ESB selection checklist –

- Are you integrating 3 or more applications/services? If you only need to communicate between 2 applications, using point-to-point integration is going to be easier.
- Will you really need to plug in more applications in the future? Try and avoid YNNI in your architecture. It's better to keep things simple re-architect later if needed.
- Do you need to use more than one type of communication protocol? If you are just using HTTP/Web Services or just JMS, you're not going to get any of the benefits if cross-protocol messaging and transformation that Mule provides.
- Do you need message routing capabilities such as forking and aggregating message flows, or content-based routing? Many applications do not need these capabilities.
- Do you need to publish services for consumption by other applications? This is a good fit for Mule as it provides a robust and scalable service container, but in Erik's use case, all they needed was an HTTP client from their front-end Struts application.
- Do you have more than 10 applications to integrate? Avoid big-bang projects and consider breaking the project down into smaller parts. Pilot your architecture on just 3 or 4 systems first and iron out any wrinkles before impacting other systems.
- Do you really need the scalability of an ESB? It's very easy to over-architect scalability requirements of an application. Mule scales down as well as up making it a popular choice for 'building in' scalability. However, there is a price to be paid for this since you are adding a new technology to the architecture.
- Do you understand exactly what you want to achieve with your architecture? Vendors often draw an ESB as a box in the middle with lots

of applications hanging off it. In reality, it does not work like that. There is a lot of details that need to be understood first around the integration points, protocols, data formats, IT infrastructure, security, etc. Starting small helps to keep the scope of the problem manageable and keep the fuckupery to a minimum. Until you understand your architecture and scope it properly, you can't really make a decision as to whether an ESB is right for you.