**Apache Camel Architecture:**

Apache Camel implements the **Enterprise Integration Patterns** (EIPs) and offers a standardized, internal **domain-specific language** (DSL) to integrate applications. **EIPs** can be used to split integration problems into smaller pieces and model them using standardized graphics. Apache Camel runs on the Java Virtual Machine (JVM). The core of **Apache Camel** is very small and just contains commonly used **Components** (i.e. connectors to several technologies and APIs) such as **Log, File, Mock or Timer**. The core functionality of Apache Camel is its **routing engine**. It allocates messages based on the related routes. A route contains flow and integration logic. It is implemented using **EIPs** and a specific **DSL.** Each message contains a body, several headers and optional attachments. The messages are sent from a provider to a consumer. In between, the messages may be processed, e.g. **filtered or transformed**.

Messages between a provider and a consumer are managed by a **message exchange container**, which contains a unique message id, exception information, incoming and outgoing messages (i.e. request and response), and the used **message exchange pattern** (MEP).

In Only MEP is used for one-way messages such as JMS whereas In Out MEP executes request-response communication such as a client-side HTTP based request and its response from the server side.

**Apache Camel** offers multiple **DSL** (domain specific language) such as **Java, Scala, Groovy**. The purpose of the DSL is to allow the developer to focus on the integration problem rather than on the tool—the programming language. Although Camel is written mostly in Java, it does support mixing multiple programming languages.

Here are **some examples** of the **DSL** using different languages and staying functionally equivalent.

**Java DSL**

from("file:data/inbox").to("jms:queue:order");

**Spring DSL**

<route>

<from uri="file:data/inbox"/>

<to uri="jms:queue:order"/>

</route>

**Scala DSL**

from "file:data/inbox" -> "jms:queue:order"

**They show how easily you can route files from a folder to a JMS queue using different DSL.**

**CamelContext:** Every camel application will have at least **one CamelContext**. This is the place where we add camel routes. It is similar to **ApplicationContext of Spring**. **Camel context** can be thought as a **container** which keeps all things together. One camel context can have multiple routes inside it.

**Routes**: **CamelContext** may contain one or more **routes**. Routes are the integration logic which defines how data will flow in camel context from one endpoint to another.

**Endpoint**: Endpoint is end of channel through which system can send or receive messages. This is what we call as destination or source in communication language.

**Components**: Components are point of extension in Camel. Components can be an interface to technology, data format, transformers, etc. They may also act as a factory for endpoints.

Camel supports most of the **Enterprise Integration Patterns**.

**Content Based Router Pattern**: CBR patterns allow us to route data as per the content of the input file.

**Splitter Pattern**: A splitter pattern is used to split input data into smaller chunks.

**Recipient List:** A recipient list pattern is used when a list of recipients needs to be retrieved from the message body itself.

**Some of commonly used EIP are as mentioned below.**

**1.Log −** To log complete message or part of it

**2.Message Filter −** Filtering contents of messages

**3.Re-Sequencer −** To get all tokens in sequence

**4.Wiretap −** To inspect travelling messages

Figure shows the architecture of Apache Camel. A **CamelContext** provides the runtime system. Inside, **processors** handle things in between endpoints like routing or transformation. **Endpoints** connect several technologies to be integrated. Apache Camel offers different **DSLs** to realize the integration problem. The **CamelContext** is the runtime system of Apache Camel and connects its different concepts such as **routes, components or endpoints**. Usually, the CamelContext is started when loading the application and stopped at shutdown. The runtime system can be included anywhere in the JVM environment, including web container (e.g. Tomcat), JEE application server (e.g. IBM WebSphere AS), OSGi container, or even in the cloud.

**Components**

In the meantime, over 100 components are available. Besides widespread technologies such as HTTP, FTP, JMS or JDBC, many more technologies are supported, including cloud services from Amazon, Google, GoGrid, and others. New components are added in each release. Often, also the community builds new custom components because it is very easy.

The most amazing feature of Apache Camel is its uniformity. All components use the same syntax and concepts. Every integration and even its automatic unit tests look the same. Thus, complexity is reduced a lot. While components offer the interface to technologies, Processors and Beans can be used to add custom integration logic to a route.

**Processors and Beans**

Besides using EIPs, you have to add individual integration logic, often. This is very easy and again uses the same concepts always: Processors or Beans. Both were used in the routes.

Processor is a simple Java interface with one single method: process. Inside this method, you can do whatever you need to solve your integration problem, e.g. transform the incoming message, call other services, and so on.

