Apache Camel is one of my favorite open source frameworks in the JVM / Java environment. It enables easy integration of different applications which use several protocols and technologies. This article shows when to use Apache Camel and when to use other alternatives.

The Problem: Enterprise Application Integration (EAI)

Enterprise application integration is necessary in almost every company due to new products and applications. Integrating these applications creates several problems. New paradigms come up every decade, for example client / server communication, Service-oriented Architecture (SOA) or Cloud Computing.

Besides, different interfaces, protocols and technologies emerge. Instead of storing data in files in the past (many years ago), SQL databases are used often today. Sometimes, even NoSQL databases are required in some usecases. Synchronous remote procedure calls or asynchronous messaging is used to communicate via several technologies such as RMI, SOAP Web Services, REST or JMS. A lot of software silos exists. Nevertheless, all applications and products of these decades have to communicate with each other to work together perfectly.

Enterprise Integration Patterns (EIP)

Of course, you could reinvent the wheel for each problem, write some spaghetti code and let the applications work together. Unfortunately, your management will not like the long-term perspective of this solution.

Enterprise Integration Patterns (www.eaipatterns.com) help to fragment problems and use standardized ways to integrate applications. Using these, you always use the same concepts to transform and route messages. Thus, it is a good idea to forget about reinventing the wheel each time you have a problem.

Alternatives for Integrating Systems

Three alternatives exist for integrating applications. EIPs can be used in each solution.

Solution 1: Own custom Solution

Implement a individual solution that works for your problem without separating problems into little pieces. This works and is probably the fastest alternative for small use cases. You have to code all by yourself. Maintenance will probably be high if team members change.

Solution 2: Integration Framework

Use a framework which helps to integrate applications in a standardized way using several integration patterns. It reduces efforts a lot. Every developer will easily understand what you did (if he knows the used framework).

Solution 3: Enterprise Service Bus (ESB)

Use an enterprise service bus to integrate your applications. Under the hood, the ESB also uses an integration framework. But there is much more functionality, such as business process management, a registry or business activity monitoring. You can usually configure routing and such stuff within a graphical user interface – you have to decide at your own if that reduces complexity and efforts. Usually, an ESB is a complex product. The learning curve is much higher. But therefore you get a very powerful tool which should offer all your needs.

What is Apache Camel?

Apache Camel is a lightweight integration framework which implements all EIPs. Thus, you can easily integrate different applications using the required patterns. You can use Java, Spring XML, Scala or Groovy. Almost every technology you can imagine is available, for example HTTP, FTP, JMS, EJB, JPA, RMI, JMS, JMX, LDAP, Netty, and many, many more (of course most ESBs also offer support for them). Besides, own custom components can be created very easily.

You can deploy Apache Camel as standalone application, in a web container (e.g. Tomcat or Jetty), in a JEE application Server (e.g. JBoss AS or WebSphere AS), in an OSGi environment or in combination with a Spring container.

If you need more information about Apache Camel, please go to its web site as starting point: *http://camel.apache.org*. This article is no technical introduction J

When to Use Apache Camel?

Apache Camel is awesome if you want to integrate several applications with different protocols and technologies. Why? There is one feature (besides supporting so many technologies and besides supporting different programming languages) which I really appreciate a lot: **Every integration uses the same concepts!** No matter which protocol you use. No matter which technology you use. No matter which domain specific language (DSL) you use – it can be Java, Scala, Groovy or Spring XML. You do it the same way. Always! There is a producer, there is a consumer, there are endpoints, there are EIPs, there are custom processors / beans (e.g. for custom transformation) and there are parameters (e.g. for credentials).

Here is one example which contains all of these concepts using the Java DSL:

from(„activeMQ:orderQueue“)..transaction().log(„processing order“).to(mock:“notYetExistingInterface“)

Now let’s look at another example using the Scala DSL:

„file:incomingOrders?noop=true“ process(new TransformationProcessor) to „jdbc:orderDatastore“

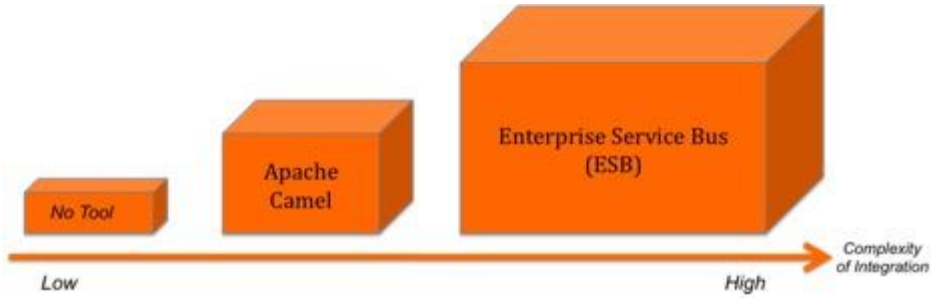
If you are a developer, you should be able to recognize what these routes do, don’t you?

Two other very important features are the support for error-handling (e.g. using a dead letter queue) and automatic testing. You can test EVERYTHING very easily using a Camel-extension of JUnit! And again, you always use the same concepts, no matter which technology you have to support.

Apache Camel is mature and production ready. It offers scalability, transaction support, concurrency and monitoring. Commercial support is available by FuseSource: *http://fusesource.com/products/enterprise-camel*

When NOT to Use Apache Camel?

Well, yes, there exist some use cases where I would not use Apache Camel. I have illustrated this in the following graphic (remember the three alternatives I mentioned above: own custom integration, integration framework, enterprise service bus).



If you have to integrate just one or two technologies, e.g. reading a file or sending a JMS message, it is probably much easier and faster to use some well known libraries such as Apache Commons IO or Spring JmsTemplate. But please do always use these helper classes, pure File or JMS integration with try-catch-error is soooo ugly!

Although FuseSource offers commercial support, I would not use Apache Camel for very large integration projects. An ESB is the right tool for this job in most cases. It offers many additional features such as BPM or BAM. Of course, you could also use several single frameworks or products and „create“ your own ESB, but this is a waste of time and money (in my opinion).

Several production-ready ESBs are already available. Usually, open source solutions are more lightweight than commercial products such as WebSphere Message Broker (you probably need a day or two just to install the evaluation version of this product)! Well-known open source ESBs are Apache ServiceMix, Mule ESB and WSO2 ESB. By the way: Did you know that some ESB base on the Apache Camel framework (e.g. Apache Service Mix and the Talend ESB). Thus, if you like Apache Camel, you could also use Apache ServiceMix or the commercial Fuse ESB which is based on ServiceMix.

Conclusion

Apache Camel is an awesome framework to integrate applications with different technologies. The best thing is that you always use the same concepts. Besides, support for many many technologies, good error handling and easy automatic testing make it ready for integration projects.

Because the number of applications and technologies in each company will increase further, Apache Camel has a great future. Today we have application silos, in ten years we will probably have cloud silos which are deployed in Goggle App Engine, CloudFoundry, Amazon EC3, or any other cloud service. So I hope that Apache Camel will not oversleep to be ready for the cloud era, too (e.g. by offering components to connect to cloud frameworks easily). But that's future… At the moment you really should try this framework out, if you have to integrate applications in the JVM / Java environment.