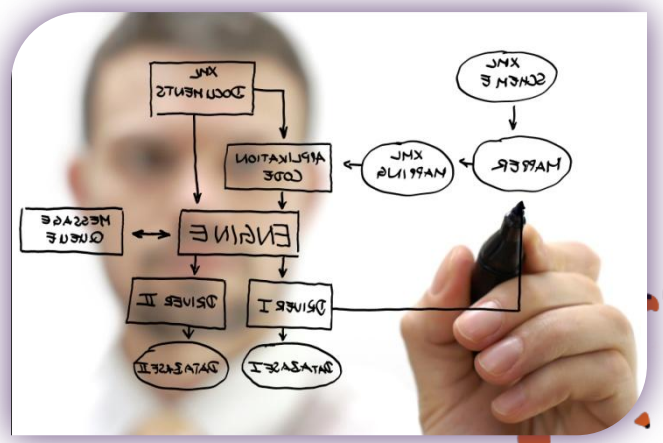
White paper

How to choose your process orchestration technology ?

Gaël Blondelle, Adrien Louis, Marc Dutoo



Version 1.0 – December 2009

Get more information :

www.petalslink.com

Tél : +33 (0)5 62 73 43 80

Mail : contact@petalslink.com

Preamble

Why this white paper ?

This whitepaper is written for architects, project managers, and SOA developers, willing to use SOA orchestration for creating processes. It explains differences between existing orchestration technologies, and summarizes the main criterions for choosing the most appropriate technologies.

The second part focuses on technical details and implementations of orchestration, and routing technologies and patterns.

Authors

Gaël Blondelle is CTO of **Petals Link**, Vice-president of OW2 Open source consortium, Co-founder of Toulouse Java User Group, and public speaker about open source, SOA, and Java.

Adrien Louis is the initial developer of the **Petals ESB project** (former PEtALS), then chief product architect. He has been appreciated as a great trainer for Petals ESB, and he is the author of several articles and whitepapers about SOA and architecture.

Marc Dutoo is Open Source solution architect at **Open Wide**, a leading French Open Source portal integrator. His interests include SOA, BPM and content management. Marc also co-leads the Eclipse Java Workflow Tooling project, and he is one of the first contributors in the Petals ESB project.

Petals Link

Petals Link (a brand of EBM Websourcing) is an open source SOA company, focusing on SOA integration solutions. **Petals ESB**, their flagship **open source ESB**, is a basis for large-scale decentralized architectures. It comes with **Business activity monitoring (BAM)** and **SOA governance** to improve SOA possibilities.

Some Petal Link references

French Social Security Central Agency (ACOSS), French state modernization agency (DGME), Orange, Alcatel-Lucent, Akerys, Academy of Toulouse, Gironde General Council, Limousin Regional Council, City of Lyon, Cegedim Activ, Almerys/Orange Business Services, Thales, EADS, French Defense Infrastructures Department (SID), French Air Force, French General Directorate of Armament (DGA).

Index

How to choose the most adequate orchestration technology?

SOA is a multi-facet approach, whose main pillars are Service infrastructure, Service governance and Service Orchestration. Petals leverages the JBI specification to support the Service Infrastructure approach.

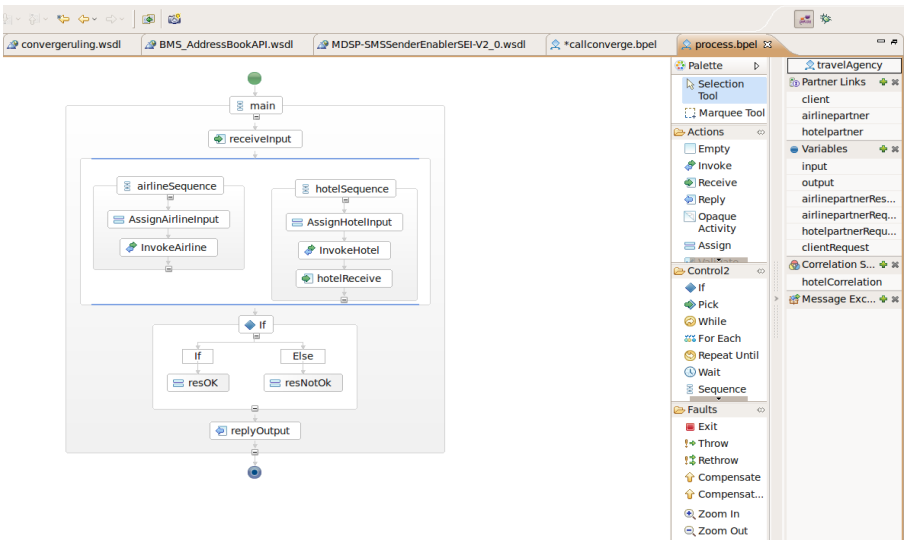
But as soon as you have a Service Oriented Infrastructure layer, you may want to orchestrate these services in order to add value to your overall information system. This first part focuses on the different orchestration means available with Petals. The second part makes a deeper comparison between two which can often offer similar features : service orchestration and routing

Existing technologies

BPEL

Most people closely associate SOA with BPEL due to the fact that WS-BPEL - the real name of the **OASIS specification** accepted in April 2007 - is designed to orchestrate Web Services.

The fact is that BPEL is the standard adopted by the SOA market and supported by the major software providers to orchestrate Web Services. Given the similarities between Web Services and JBI semantics, **BPEL is also a good option to orchestrate JBI services**.



All in all, BPEL can be used in two ways:

- Orchestrate fine-grained services to create coarse-grained services. Such coarse-grained services are typically short-lived.
- Create full-featured long-lived processes, which orchestrate coarse-

grained services or sub-processes. This use case focuses on error-handling and "compensation" features provided by BPEL. "Compensation" is the capability offered by BPEL to deal with long term transactions in the SOA world where services are stateless, non transactional resources. For example, if the process called a service to book an hotel room, the compensation mechanism will call a specific service to unbook the room if something goes wrong in the rest of the process.

However, BPEL is not the only orchestration technology. Other technologies exist and deserve our consideration.

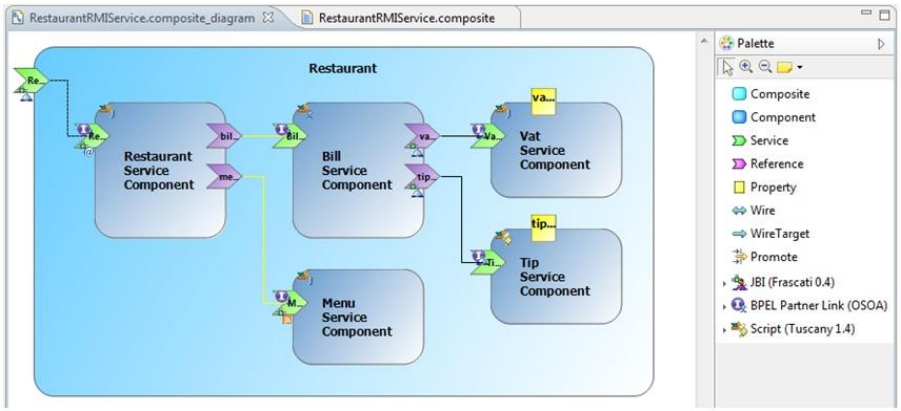
SCA

SCA is about putting components together to create services.

The SCA programming model leverages few concepts:

- A component exposes interfaces that represent entry points to the components and are considered as services
- A component declares references that correspond to dependencies to other components or services.
- A component exposes properties that can be set by the SCA environment to change the component behavior.

SCA is largely used in **an approach which privileges graphical design of components and services with tools like Eclipse STP**.



Rules Engine

Rules engines provide a totally different way to perform orchestration. We call it declarative orchestration because you declare your rules' condition and action parts, and the rules engine computes the real orchestration according to the incoming conditions.

The integration of Drools in a JBI ESB like Petals enables to start a rule on an

incoming JBI message, and to send back messages in the bus when some specific rules are activated.

We think that it's a very elegant way of implementing a declarative orchestration, but it may be hard to support by the average developer as long as these types of technologies are not mainstream.

EIP

Since the first version of Petals, a component implementing common Entreprise Integration Patterns (EIP) is provided to support basic "orchestration" patterns like:

- Pipe : to chain several services
- Split/Aggregate : to run several service processing chains in parallel
- Content Based Routing
- Service based routing (Content based routing, but the routing key is provided by a call to another service)
- Bridge to change from a synchronous to an asynchronous exchange pattern
- WireTap, which gives the capability to "spy" a flow while it passes through the bus

Ad-Hoc JBI component in Java

For very specific orchestration issues, a Java ad-hoc component can be developed.

When possible, we urge our users to consider SCA Java component instead of an adhoc JBI component built on top of the Petals "CDK" because SCA is more standard and should be sufficient.

Finally, writing a new component should be reserved to the case when it is necessary to support a brand new protocol.

Which orchestration technology fits you ?

In summary, we put together the different orchestration means available with Petals:

Orchestration Technology	**Skills needed**	**Specific orchestration pattern supported**
BPEL	Graphical tooling exists. Need some sort of "modeling" skills to get the right level of granularity	Support for compensation, that is the capability to implement long lived transactions with non transactional resources.
SCA (Beta)	Graphical tooling for SCA assembly Average Java skills needed to develop SCA component	Orchestration is developed in plain Java so that Exception handling or similar is easy.
Petals EIP SE	Learn EIP configuration language	Some EIP patterns
Rules Engine (Beta)	Declarative programming can either be considered more natural or more abstract. Nevertheless, it needs specific skills	-
Ad-Hoc component in Java	Expert Java skills	Hard coded. Can be considered both as a bus extension mean or as an orchestration mean. Must be considered when other technologies introduce performance issues.

Each orchestration has its pro and cons. We consider that they must be evaluated according to the specific integration context (integration patterns to support, standards, skills needed …) before a choice can be made.

Choosing between Routing and Orchestration in an ESB

Enterprise Service Buses are nowadays indeed useful solutions that combine an array of tools, and allow solving practical problems in the field of application and service integration. However, they present the same mild inconvenience that a toolbox does to its user who knows that the solution to his problem has to be in the box, but for the sake of him can't figure out which one it is!

The goal of this article is to help ESB users choose the right answer according to their needs, when confronted with the most complex and diverse of ESB concepts: routing and orchestration. Instead of abstract theorizing we will ground our efforts and reasoning in simple, real-world examples with the JBI compliant ESB[1] : OW2 Petals ESB, in an attempt to fill the void between low-level routing and global, business service orchestration. In other words: we will try to uncover how the different layers of routing and orchestration build up.

From Enterprise Service Bus to the routing problem

ESBs have a lot of fields of application, including implementing information system-wide Service Oriented Architectures (SOAs). But at the lowest level they all aim to ease application and service integration - that is, letting one application or service call another. This very simple and common endeavour has various additional levels of complexity:

- "routing", when there is not one but many source services where calls originate from or target services to choose between ;
- "protocol bridges", when services are exposed on another protocol, belong to other servers or even other information systems ;
- "transformations", when service messages do not have the same data format – which is rule rather than exception.

Those three: routing, protocol, transformation have a range of close siblings, but may nonetheless be considered the main ESB concepts. In this article we will focus on the first one and how it relates to a close sibling of his: orchestration. As a short introduction, let us say that routing is fundamentally low-level, near or in the ESB core, and relies on technical configuration (like service deployment descriptors) to provide technical decisions on where a message has to be sent. Orchestration can be seen as combining service calls to create higher-level, more useful composite services, but also often has a definitive "business-level" ring,

[1] **Petals ESB**, the OW2 ESB. Petals ESB provide a leading open source ESB to support SOA. It is a lightweight, highly distributed and scalable platform for both A2A and B2B integration. Thanks to its specific distributed architecture and the tools provided, such as administration, BAM, Eclipse IDE and Governance, Petals ESB offers a very competitive integration solution with support of a large number of protocols, formats and integration features.

and in this case is shorthand for implementing business-level processes combining business-specific services across applications and information systems.

Routing versus orchestration: neither a "one size fits all" nor a "black and white" world

So how are orchestration needs addressed in an ESB? It would seem logical to use an orchestration engine provided with the middleware solution. However, this is far too simple an answer to a complex question. Let us consider the following example.

Displaying a list of items

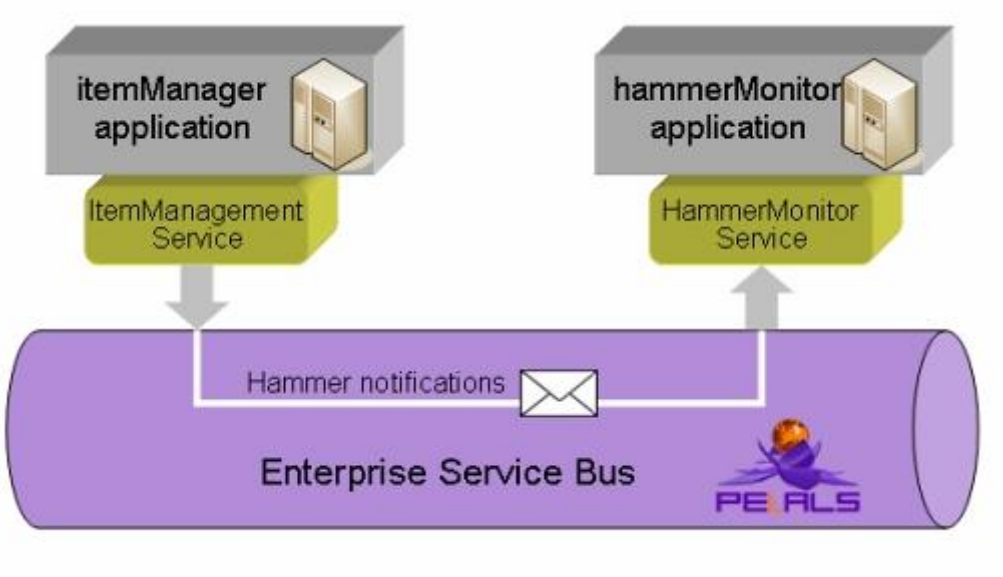
The "ItemManager" application is designed to manage items through operations like creation, update, deletion. This application is connected to an "ItemManagementListener" service, that publishes notifications when an item is updated.

Another application, the "HammerMonitor" application, is a monitoring tool that displays information on item updates that are specifically about hammers. This application exposes a "HammerMonitor" service with a "display" operation that receives these notifications.

Both services are exposed on an ESB. What we want is to let the HammerMonitor display hammers that are known to the ItemManagement application.

In order to connect the ItemManagementService to the HammerMonitorService, we need to configure the ESB connectors (aka "binding components"). One connector is linked to the ItemManager application, the other one is linked to the HammerMonitor application.

Moreover the connector linked to the HammerMonitor application is configured to expose, inside the ESB, an endpoint whose name can be "hammerMonitorService". Thus, a simple way to achieve our goal is to configure the connector linked to the ItemManager application so that it calls, inside the ESB, the endpoint "hammerMonitorService" whenever it receives a message from the ItemManager application.



However, as often in the real world, let us say both services have different data formats. This is not a barrier to SOA, as SOA defines a loosely coupled architecture (i.e. it is not mandatory for a service consumer to fit to the service provider definition).

The ItemManagement application provides to the ItemManagementListenerService the following message:

```
<items>
    <item type="Hammer" name="hammer1"/>
 </item>
```

And the ItemMonitorService has an operation "display" using the following format:

```
<hammers>
    <hammer hammerName="hammer1"/>
 </hammers>
```

At this point, a mere call does not work anymore to link both services. Data provided by the ItemManagement application needs to be first transformed. This is actually a very simple, local need of orchestration that has nothing to do with the business level.

A first way to address this would be to use a common, well-known orchestration solution like full blown, externally deployed, BPEL-supporting orchestration engine[2]. This would work, but in this case this would be akin to use a hammer (pun intended) to open a nut: either all transformed messages would have to go through a single central, remote orchestration engine, in a manner akin to the obsolete "hub" integration architecture, or there would have to be an

[2] like **EasyBPEL**, used as Petals BPEL engine component

orchestration engine deployed on each node – an obviously far too heavy solution for this simple problem.

So it appears a single, global, business-level answer to orchestration needs is not enough: what about the "dirty" work that has to be done between the routing and the business level, when generic routing provided by the bus is not enough and the main concern is not yet to implement business rules or processes by manipulating SOA-managed business services, but merely to combine technical, "behind-the-scene" services so they "get the work done"?

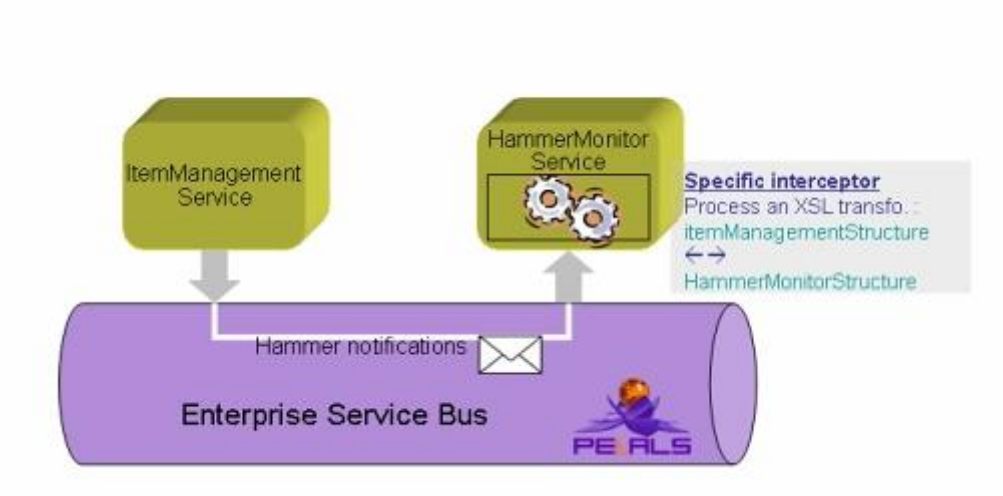
The bus-level, specific development approach: interceptors

The lowest level answer to technical routing and orchestration needs lies in enhancing the ESB's built-in features.

In the case of our previous example, a direct way to circumvent the problem of data consistency between the application that sends the message and the application that receives it is to add some logic in the connectors (i.e. the binding components of the ESB).

For instance, the binding components provided by the PEtALS ESB can be extended with "interceptors". An interceptor is a piece of Java code that is executed in the "sender" binding component before a message is sent into the bus, or in the "receiver" component, when a message is delivered.

In our example, this code can call an XSL transformation to adapt the ItemManagement message format to the HammerMonitor format.



Nevertheless, this approach is very restrictive and not extensive. If the XSL transformation is performed in the "receiver" connector (linked to the HammerMonitor), it assumes that all messages received have the ItemMangement XML structure. If a message comes from another application, it can have a different structure, and in this case the XSL transformation may fail.

The interceptor could check the incoming message structure and choose one

XSL transformation or another, depending on the message, but would still remain very coupled to the sender. This approach does not respect the loose coupling concept of SOA. Moreover any other need besides transformation would imply developing another set of specific features within the ESB engine, and that can't be expected from ESB users, nor should it.

The component ("building block") oriented approach: the EIP toolset

ESBs offer integration facilities by providing integration components. These components can do a range of small, useful, flexible operations between a consumer and a service provider. They typically implement several Enterprise Integration Patterns (made well known by Gregor Hohpe[3]) and are the Swiss knife of ESB users.

Independent of the service descriptions (WSDL and others), these EIP Components just perform small things. The most known are:

- The "pipe" pattern: a single event triggers a sequence of processing steps, each performing a specific function. The EIP Component sequences the calls.
- The "content based router" pattern: the EIP Component examines the message content and routes the message onto a different channel, based on data contained in the message.
- The "message dispatcher" pattern: the EIP Component sends the message to a list of service providers (multipoint)
- The "scatter gather" pattern: the EIP Component routes a request message to a number of service providers. It then aggregates all the responses into a single response message

The knowledge of all EIP Component operations allows the developer to combine business applications (consumers and service providers) with several "integration pattern bricks". The final result is a composite integration. Each brick of the integration is a service.

Of course, in order to design this composite integration, a dedicated graphical IDE is paramount since it brings, in addition to ease of use, a centralized view of the configuration of all the bricks. For instance, the following samples are designed by the PEtALS ESB integration tool.

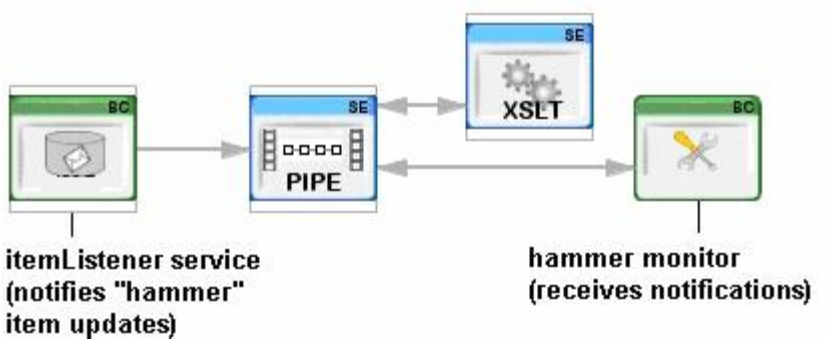
The pipeline

The pipeline pattern is used to "pipe" an incoming message to several services. The message is sent to the first one, and its response is sent to the second one, whose response is itself sent to the third one, and so on.

[3] Gregor Hohpe's **Enteprise Integration Patterns**

Adaptation between a consumer and a service provider

The ItemManagement use-case that we described previously can be designed with this kind of assembly, with a transformation component and a "pipe" brick.



Management of service version evolutions

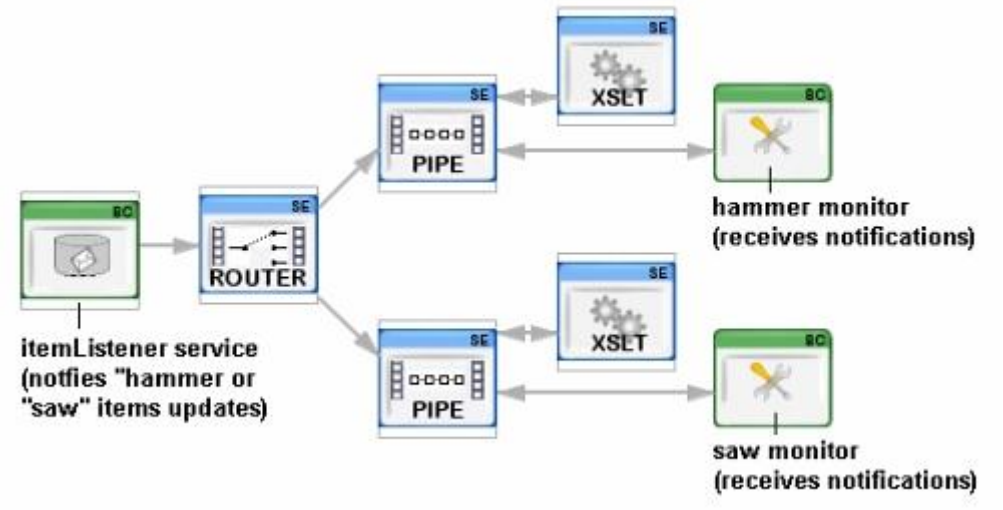
The same behavior can be used to manage service version evolution, in the following way. A consumer always sends the same message structure to the "pipe" brick, which is a proxy to the real service. When the service signature changes, the "pipe" brick sends the consumer message first to an XSL transformation (to adapt the consumer's message to the new service format), then it sends it to the new version of the service. And nothing has changed for the consumer.

Content based routing

We've seen how to compose several services into a single one. But the dynamic process aspect is not solved. Here again comes the routing challenge: how to call one service among many?

How to switch a call to one service between many services? Well, the router brick may perform some tests to switch the request to one version or to the other one.

For instance, the ItemManagementListener can send notifications for hammer and saw items to a "content based routing" Component. This component tests the name of the item in the message, and sends it to the correct monitoring services (HammerMonitorService or SawMonitorService). As each service defines a different format, two different transformations have to be performed before sending the message to the correct service. So we compose the "routing" brick with "pipe" and "transformation" bricks.

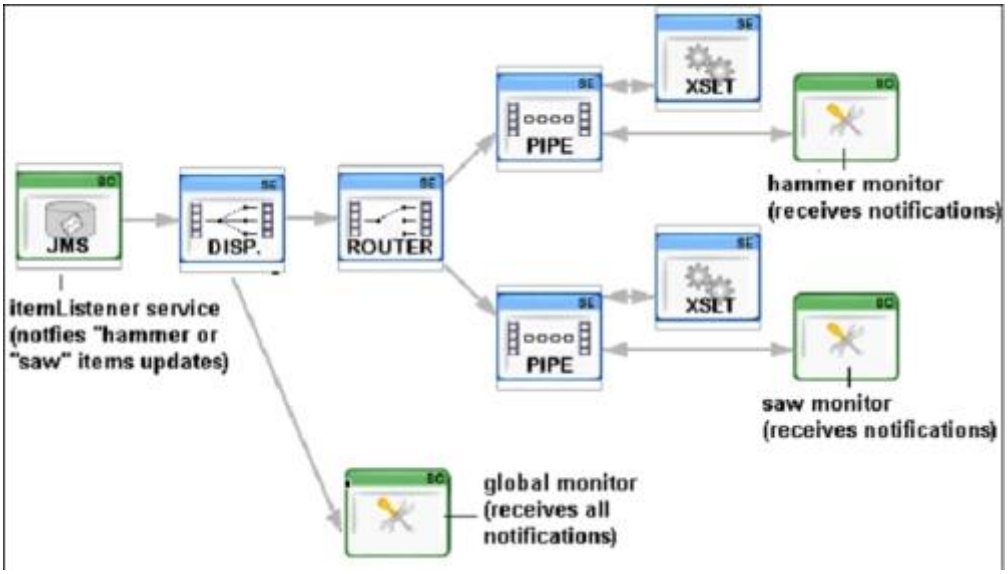


Dispatcher

Another integration need could be to send a request to several services (multi point communication). For example, when an item order is sent from a front application to the ordering system, an email can also be sent to the customer for confirmation. For example, the message is sent to an ordering service and to an SMTP service.

We can imagine that the ItemManangementListener service, which sends notifications from the ItemManagement application, has to publish the notifications to the HammerMonitor, to the SawMonitor and to a global monitoring tool (which receives all notifications).

A "dispatcher" integration brick can be added to the previous composite integration to send the message to the "routing" brick and to the global monitoring service.



The DSL-based approach : the light orchestrator

Where patterns end, the light orchestrator starts

Enterprise Integration Patterns are great concepts that help architecting routing and orchestration solutions, and the EIP component is a great tool allowing to actually design solutions to those problems. However, in complex integration cases, the composite assembly approach easily leads to too scattered and over-designed configurations. Moreover, like all patterns, EI Patterns are limited in numbers, while the real world is full of unexpected cases that call for a more flexible solution.

The answer is to use a light orchestration-specialized DSL (Domain Specific Language), which is what the "light orchestrator" or "Enterprise Integration Orchestration" component provides in PEtALS.

- When is it the right time to use such a component? It depends on a lot of things, including development practices, but here are a few hints:
- When, as we've just said, it is hard to envision a solution using only straight, "by the book" patterns,
- When "routing" and multiplexing patterns such as the one previously described become commonplace (this might also hint at using a rules engine component),
- When there are many layers of embedded "bricks" in an EIP-based system,
- When an orchestration subsystem is best understood and maintained when being solved in one single place rather than scattered across several, albeit simple, EIP "bricks"
- When there is a need for rarer EI Patterns that is not supported by the EIP component (fully dynamic routing, Return Address, Content Enricher, Normalizer…)

EIOrchestration use case : complex dynamic routing

In order to showcase the EIOrchestration component, let's focus on our system's extensibility.

We've already seen how to add a saw-specific monitoring feature to a system that was initially only able to handle hammers. We could add other tool-specific abilities the same way. However this would require reconfiguring them again each time we want to add another tool type. So what if we want the people using our bus to be able to add their own tool types and specific monitoring abilities?

Example: Our customer wants to be able to dynamically add a ScrewdriverMonitorService for tools of type Screwdriver, and DrillerMonitorService for Drillers, and so on.

We could tell them to mention within each message the name of the tool-specific monitor service it must be sent to, and add dynamic routing capabilities to our system.

Example: We enhance the ItemManagement application so it provides the

following message body to the ItemManagementListenerService:

```
<items>
    <item type="Screwdriver" name="screwdriver1"
          customMonitorService="ScrewdriverMonitorService"/>
 </item>
```

where customMonitorService is an additional data field that may be provided by the customer through the ItemManagement application.

In an ESB, routing such a message can be done by dynamically choosing its recipient service according to the "customMonitorService" attribute. For example, this can be done in PEtALS using the EI Orchestration component, using its "get-calls-by-xpath" feature:

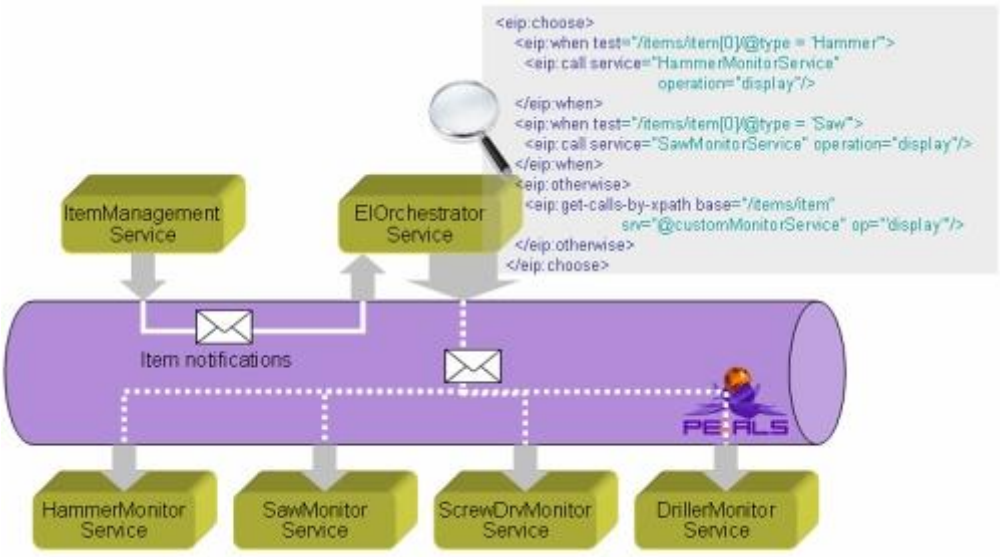
```
<eip:get-calls-by-xpath base="/items/item"
service="@customMonitorService"
       operation="'display'"/>
```

Which, in our example, will call the ScrewdriverMonitorService with the previous message.

A complete EIOrchestration sample for PEtALS

We've said at the beginning that the PEtALS EIOrchestration component allows to handle process complexity well. So here is an example that gathers in a single configuration everything we've seen in this article: piping ("eip:chain" element) and transformations, simple content based routing ("eip:choose" element) and finally dynamic routing ("eip:get-calls-by-xpath" element), while still being quite readable:

```
<eip:eip>
    <eip:chain>
       <eip:choose>
          <eip:when test="/items/item[0]/@type = 'Hammer'">
             <eip:call service="ItemToHammerService"
operation="transform"/>
             <eip:call service="HammerMonitorService"
operation="display"/>
          </eip:when>
          <eip:when test="/items/item[0]/@type = 'Saw'">
             <eip:call service="ItemToSawService"
operation="transform"/>
             <eip:call service="SawMonitorService"
operation="display"/>
          </eip:when>
          <eip:otherwise>
             <eip:get-calls-by-xpath base="/items/item"
                   service="@customMonitorService"
operation="'display'"/>
          </eip:otherwise>
       </eip:choose>
    </eip:chain>
 </eip:eip>
```



Bridging up with Business Process Management concepts

And what about full-fledged, business-level orchestration?

Another way of thinking up integration is the top-down approach, where enterprise business processes are defined. In this approach, business processes drive the definition of business services. Thus, a bridge is needed between what services are offered by existing applications and what the business process wants to orchestrate. Such a bridge is manifested in the set of all managed business-level services within the enterprise information system, i.e. its SOA (Service Oriented Architecture), which acts as a protecting layer both for lower-level, technical services on the bus and for the actual business processes.

The standard way of executing processes in the SOA world is the use of a BPEL engine. It can invoke several services and do some business logic on the flow and on XML documents, while also being able to handle data mapping issues. In this approach, business service definitions are the key of the orchestration: no BPEL orchestration can be done without the definition (WSDL typically) of all services, ensuring cleaner (however costlier) service composition.

An overview of the orchestration setup, when using BPEL in an ESB, is available in the article written by Adrien LOUIS, "build an SOA application from existing services"[4].

Human intervention in business processes: workflows

Now what if in our tool monitoring example we'd need a supervisor's approval before actually displaying information in monitoring applications? It would require a manual intervention from a dedicated operator. This is another

[4] Introduction to SOA and ESB : **"Build an SOA application from existing services"** (Adrien Louis)

face of Business Process Management: workflows, which are business processes allowing the involvement of manual, human operation, either for manual business tasks or manual supervision, through a graphical user interface that may be provided within a business portal, or a more technical administration interface.

A key point is that workflows follow the opposite paradigm of state-based approach rather than a flow-based one like BPEL orchestrators, making them better adapted to long-lived processes, without being restricted from sitting on top of orchestrated services. Hence workflow servers are usefully complemented by "straight" orchestrators, though that means deploying two business process-oriented servers – a constraint addressed by interesting new initiatives like jBoss & Bull's "Process Virtual Machine" and the Eclipse Java Workflow Tooling project[5].

[5] Unifying orchestration and workflow : **The Process Virtual Machine** (Tom Baeyens and Miguel Valdes Faura), and **The Eclipse Java Workflow Tooling project**

Conclusion

We have seen several ways to connect business services with each other, going from low-level ones like customized routing, to high-level ones using business oriented approaches like workflow and orchestration. Most importantly, we've exposed how ESB integrators have very common middle-level needs for composing local, technical services, and how a range of "glue", "Swiss knife"-like features allow them to simply "get the job done".

In summary:

- For a range of **simple integration scenarios** like the connection between two heterogeneous applications, customizing routing through ESB-specific features, e.g. adapting message data format by adding an XSL transformation in the connectors linked to the application, is actually the easiest way (the interceptor approach).
- When a strategy is needed to send the message to the right receiver and when **operations on messages have to be chained**, we can use and assemble simple, pattern-oriented integration bricks typically to perform static routings, chained with transformations (the EIP approach).
- In order to solve complex routing strategies, comprising **dynamic routing or complex imbrications**, a light orchestration component can be used to centralize the routing logic (the LightOrchestrator approach).
- At a **global, business level**, well managed, consistently defined, business-oriented services are worth the effort of being composed using orchestration like WSDL-based BPEL, and made interact with people using workflow solutions.

Different technologies exist for Orchestration or routing. Among them, EIP, SCA, BPEL, Rules engine, or Plain old Java object (POJO). Each one is a way to orchestrate, and choosing needs to look at your project specific needs, and available competencies.

Bibliography

- **Enteprise Integration Patterns** (Gregor Hope)
- Introduction to SOA and ESB : **"Build an SOA application from existing services"** (Adrien Louis)
- Unifying orchestration and workflow : **The Process Virtual Machine** (Tom Baeyens and Miguel Valdes Faura), and **The Eclipse Java Workflow Tooling project**
- **Your ESB from scratch, back to basics** ("routing, transformation, transport, security"): (Balwinder Sodhi)
- **Building composite services, from the trenches** (Jesus Rodriguez): *"However, building composite services still represents a challenge for most of the ESBs in the market. This is mostly given to the fact that most ESBs don't provide a transparent way to express the routing logic into new services that can be consumed by other applications. Instead, developers find themselves trying to implement services that behind the scenes utilize the routing capabilities built into the ESB engine."*

Petals Link

Petals Link (a brand of EBM Websourcing) is an open source SOA company, focusing on SOA integration solutions. **Petals ESB**, their flagship **open source ESB**, is a base for large-scale decentralized architectures. It comes with **Business activity monitoring (BAM)** and **SOA governance** to improve SOA possibilities.

Websites and projects

- **http://petalslink.com** – Petals Link open source software company
- **http://petals.ow2.org** – Petals ESB, the open source ESB for large organizations
- **http://dragon.ow2.org** – Petals Master (ex Dragon), governance tools integrated with Petals ESB.
- **http://opensuit.ow2.org** - Service-oriented web UI framework.
- **http://easywsdl.ow2.org** – WSDL 1.1 and 2.0 parser
- **http://easybpel.petalslink.com** – BPEL Engine

Contacts

contact@petalslink.com

+33 5 62 73 43 80

EBM Websourcing – Petals Link,

4 Rue Amélie,

31000 Toulouse, France