

# SOA

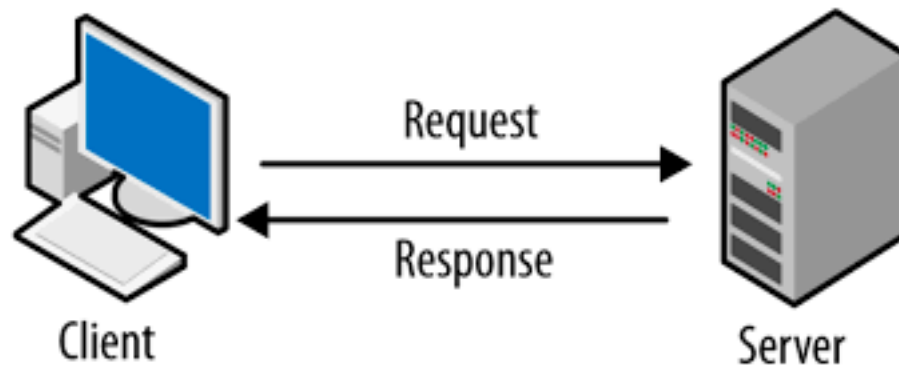
## Restful Web services

# Plan

- \* Web service, Web Api, Rest
- \* Jersey framework
- \* Springboot + Micronaut frameworks
- \* PHP Slim + Guzzle frameworks

# Client/server basics

- \* Client : asks for a resource at URI
- \* Server : return resource at URI with content



# Client/server basics

## URIs (Unique resource identifier)

- \* Identify resources
- \* Are format independent

## Resources

- \* /books/thesecondagemachine
- \* /books/lessformore

## Collection

- \* /books

# Client/server basics

Request :

- \* URI
- \* HTTP verb (or method) describing the action
- \* Some headers describing requirements
- \* A body (data)

Example :

*POST /books/book\_manage HTTP/1.1*  
*Host: iut.ca.fr*

*name1=value1&name2=value2*

# Client/server basics

## HTTP verbs:

- \* POST -> add
- \* PUT -> modify
- \* GET -> read
- \* DELETE -> ?
- \* (HEAD, OPTIONS, PATCH, etc.)

# Client/server basics

## Response :

- \* Status code
- \* Some headers
- \* Content

HTTP/1.x 200 OK

Transfer-Encoding: chunked

Date: Sat, 28 Nov 2019 04:36:25 GMT

Server: iut.ca.fr

Connection: close

Pragma: public

Expires: Sat, 28 Nov 2009 05:36:25 GMT

Cache-Control: max-age=3600, public

Content-Type: text/html; charset=UTF-8

Last-Modified: Sat, 28 Nov 2019 03:50:37 GMT

Content-Encoding: gzip

Vary: Accept-Encoding, Cookie, User-Agent

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>Top 20+ MySQL Best Practices - Nettuts+</title>
<!-- ... rest of the html ... -->
```

# Web service

*A server running on a computer device, listening for requests at a particular port over a network, serving web documents ([HTML](#), [JSON](#), [XML](#), images). Wikipedia*

Evolution of distributed systems -> SOA (Service Oriented Architecture)

Technology introduced by IBM and Microsoft, then formulated by W3C

Web API : web service with REST (see later)



# Web service

Why Web service ?

- \* Over HTTP, HTTP servers are « everywhere »
- \* Can be used from web apps. Mobile apps. Other web services.
- \* Allow parallelism more easily
  - launch X instances (via vm, docker, etc.)
  - paradigm : parallelism through message passing between instances
  - microservice architecture (1 action -> 1 service)

# Web service

2 main protocols:

- \* SOAP
  - \* Stateful ; Stateful
  - \* Protocol based upon XML
  - \* More powerful ; integrates several security protocols
  - \* More difficult to code ; more heavy
  - \* Specialised to business transactions
- \* Rest
  - \* See now

# API

API (application Programming Interface) definitions :

- \* *« A set of subroutine definitions, protocols, and tools for building application software » Wikipedia*
- \* *Une API est une interface logicielle qui permet de « connecter » un logiciel ou un service à un autre logiciel ou service afin d'échanger des données et des fonctionnalités. CNIL*

# Rest

REST is the underlying architectural principle of the web, formalized as a set of constraints, described in **Roy Fielding's dissertation**.

An API that adheres to the principles of REST **does not require the client to know anything about the structure of this API**.

Rather, the server needs to provide whatever information the client needs to interact with the service.

The key abstraction of information in REST is a resource. Any information that can be named can be a resource, and is identified by a URI.

Rest heavily relies on the HTTP protocol: RFC 2616

# Rest

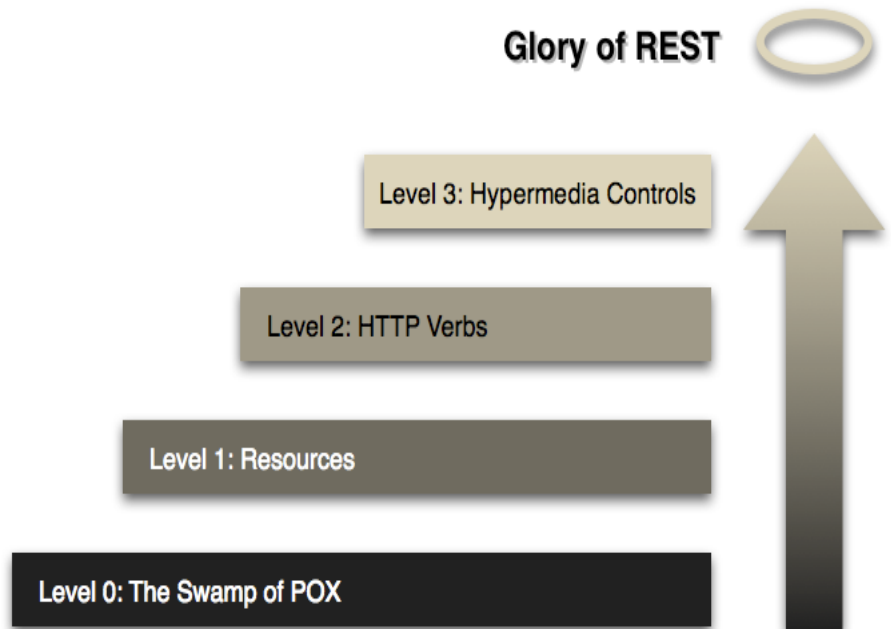
## Richardson maturity model

Level 0:

HTTP as a transport system

HTTP as a tunneling  
mechanism

Using headers content



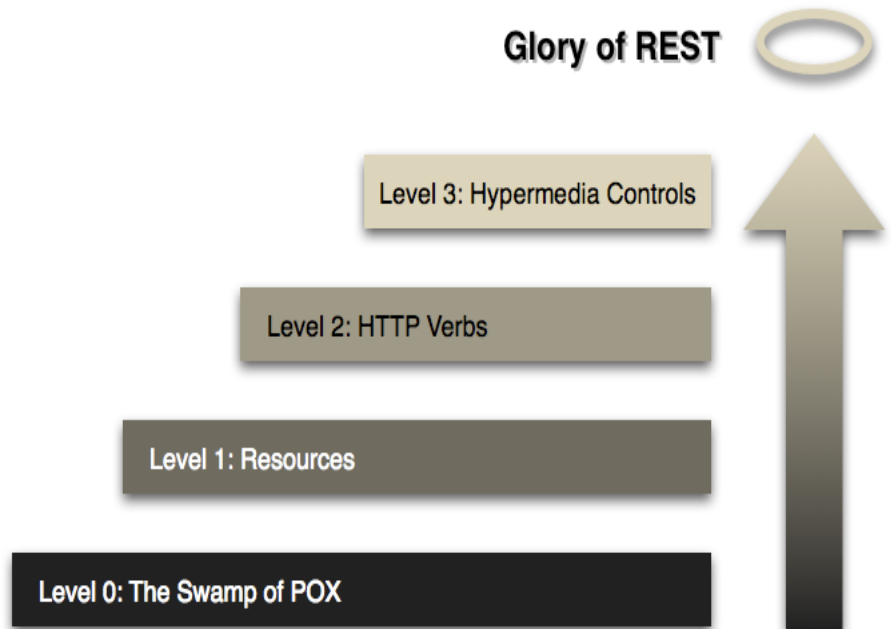
# Rest

## Richardson maturity model

### Level 1:

start talking to individual resources.

*POST /slots/1234 HTTP/1.1  
[various other headers]*



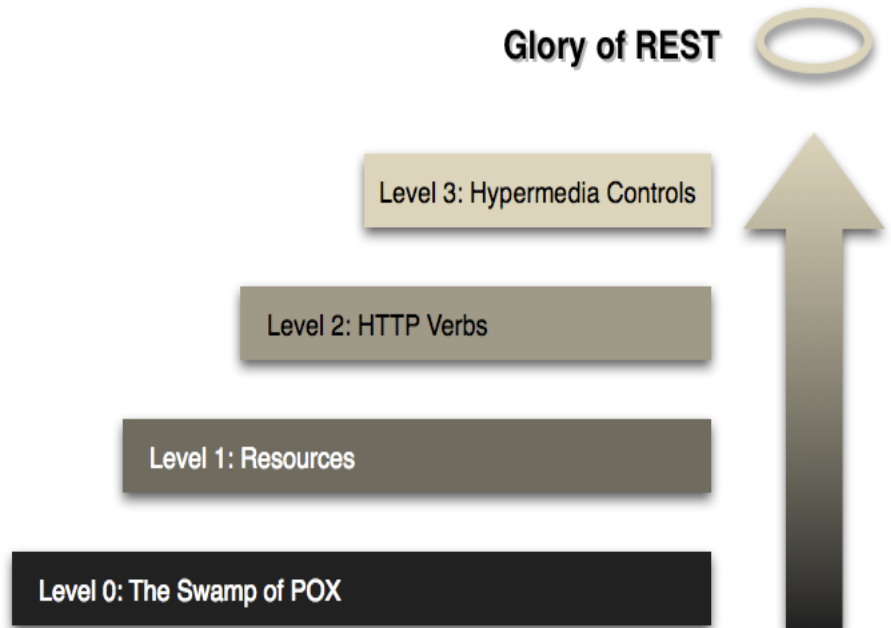
# Rest

## Richardson maturity model

Level 2:

Client uses HTTP verbs

*Servers uses HTTP status*



# Rest

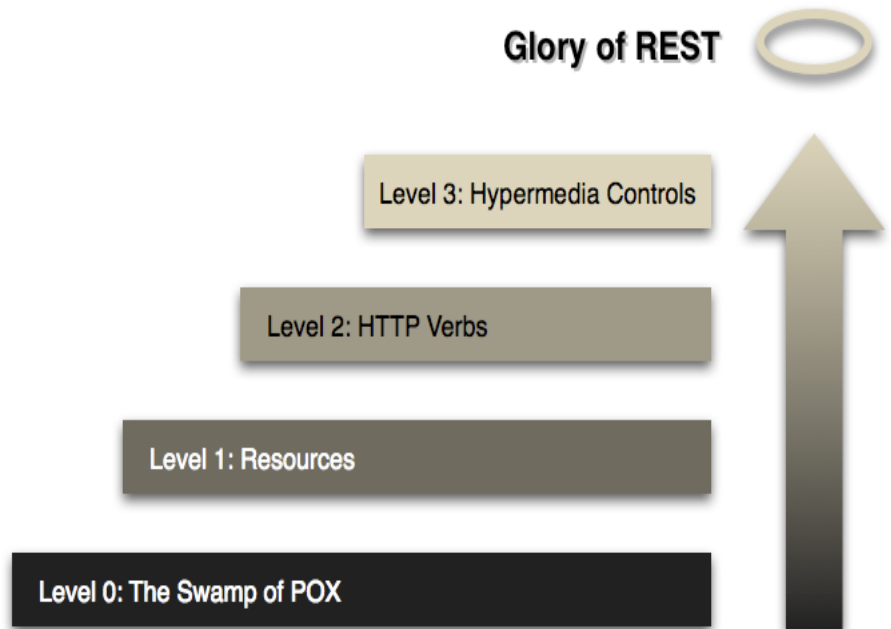
Richardson maturity model

Level3:

Use of HATEOAS (Hypertext As  
The Engine Of Application State).

Service discovery

*Hypermedia requests and  
responses*





# HATEOAS

## Media types

- \* Media type is a format of a request or response body data.

[RFC 6838](#) :

1. application/json
2. application/xml
3. application/x-www-form-urlencoded
4. text/plain; charset=utf-8
5. text/html

- \* Hypermedia types

Adds links into media types

Ex: application/hal+json : add links into json data

# HATEOAS

It means that hypertext should be used to find your way through the API

*~ application is a kind of state machine*

# HATEOAS

GET /account/12345 HTTP/1.1

Host: somebank.org

Accept: application/xml

...

HTTP/1.1 200 OK

Content-Type: application/xml

Content-Length: ...

**<?xml version="1.0"?>**

**<account>**

**<account\_number>12345</account\_number>**

**<balance currency="usd">100.00</balance>**

**<link rel="deposit"**

href="/account/12345/deposit" />

**<link rel="withdraw"**

href="/account/12345/withdraw" />

**<link rel="transfer"**

href="/account/12345/transfer" />

**<link rel="close" href="/account/12345/close"**

**/>**

**</account>**

Plus tard...

\* HTTP/1.1 200 OK

\* Content-Type: application/xml

\* Content-Length: ...

\*

\* **<?xml version="1.0"?>**

\* **<account>**

\*

**<account\_number>12345</account\_number>**

\* **<balance currency="usd">-25.00</balance>**

\* **<link rel="deposit"**  
href="/account/12345/deposit" />

\*

**</account>**

# HATEOAS

Exemple simple (voir TP)

```
isbn:      1
name:      "book"
author:    "Bauer"
▼ _links:
  ▼ self:
    href:   "http://localhost:8080/books2/1"
  ▼ books:
    href:   "http://localhost:8080/books"
```

# REST API

## REST : (RE) **State Transfer**

- \* Server doesn't keep state related to the client session
  - \* Server is stateless (can serve any client any time)
  - \* Client transfers the state
- 
- \* Stateless -> how HTTP is designed, how web is designed in general

# REST API

Rest : (RE) **State Transfer**

\* Some principles :

1. Give every “thing” an ID
2. Link things together
3. Use standard verbs
4. Resources with multiple representations
5. Communicate statelessly

Source <https://www.infoq.com/articles/rest-introduction/>

# Micro-services ?

- \* Microservices = services fonctionnels précis conçus pour réaliser parfaitement une seule chose.
- \* *Chaque service est élastique, résilient, composable, minimal et complet*
- \* Application = composition de micro-services
- \* Concept qui s'adapte bien aux conteneurs (Docker, etc.)

# Et les micro-services ?

- \* Plusieurs frameworks légers:
- \* comme [Dropwizard](#), [Spring Boot](#), [Spotify Apollo](#), [Spark](#) (Java), [Kumuluzee](#) (J2EE), [Flask](#) (Python), [Sinatra](#) (Ruby) ou [Vert.x](#) (Polyglotte).
- \* Micronaut
- \* Quarkus



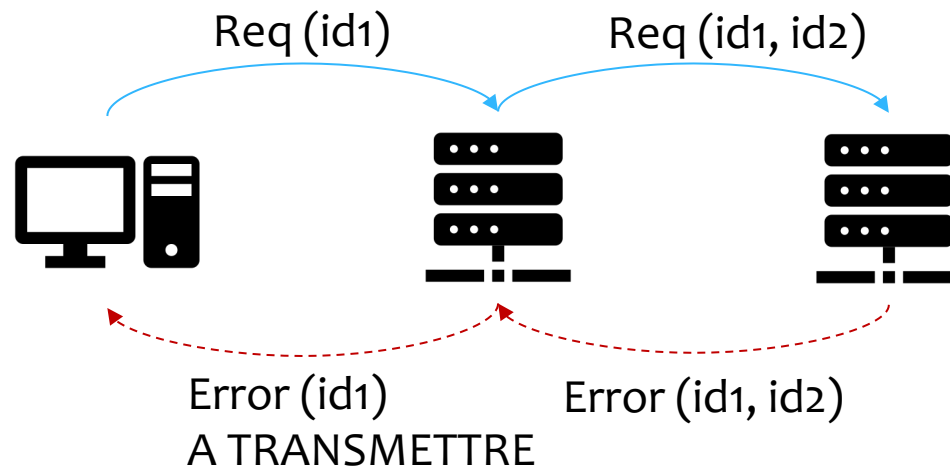
# Web service composition

# Composition de services Web

- \* Composition
  - \* Faire interagir des services Web ensemble
  - \* Déployés sur le même serveur, ou sur des Clouds/serveurs différents
- \* La composition appelée service composite, services invoqués appelé des composants de service
- \* D'un point de vue Client, service composite = service
- \* 2 types de composition
  - \* Orchestration
  - \* chorégraphie

# Composition de services Web

## \* Gestion des erreurs ; transfert des erreurs



# Composition de services Web

- \* Difficultés:
  - \* Gestion des erreurs
  - \* Si 1 composant remonte une erreur, elle doit être gérée par le service appelant pour un retour vers le client
- \* 2 types de composition
  - \* Orchestration
  - \* chorégraphie

# Gestion des services web

## Orchestration des services

- Lorsqu'un service web coordonne d'autres services
- 1 processus global avec appel vers d'autres services, gestion des erreurs
- Compositions simples en Java etc.
- Compositions complexe, besoin de meta langages -> BPEL,

## Orchestration des services

- Langages BPMN / BPEL

- processus BPEL (processus écrit en XML qui décrit comment interagissent les WS suivant des stimuli extérieurs)

- Besoin d'un serveur qui exécute les processus BPEL  
la gestion des erreurs doit être gérée par le processus  
(mécanisme de replis, re-exécution du processus)

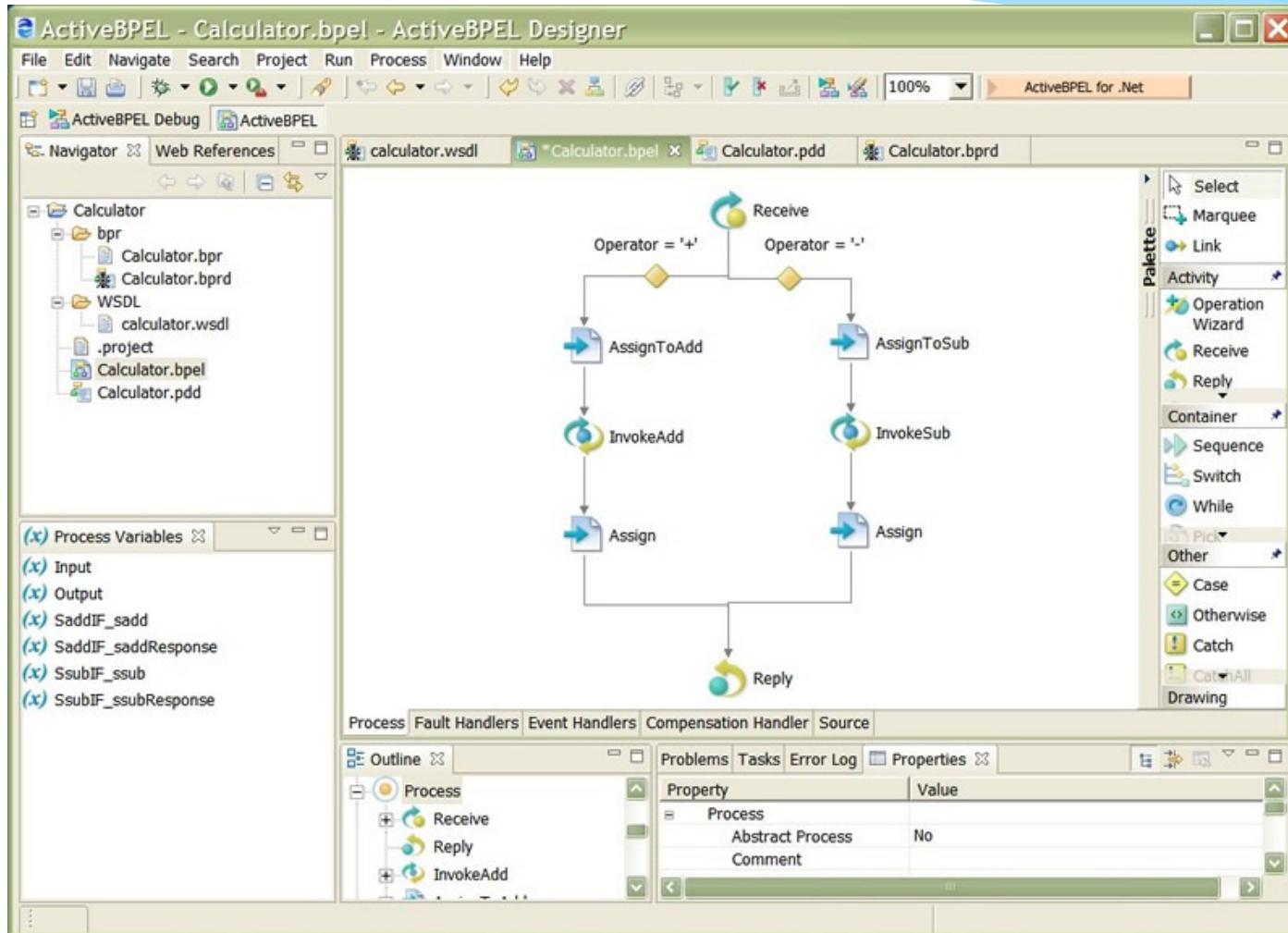
- Langage de programmation de processus mais aussi interface graphique (boîtes)

# Apercu de WSBPEL

- Definition des partenaires
- Utilisation de variables, assignation de valeurs (assign)
- Activités basiques (invoke, receive, reply, wait, throw)
- Activités structurés (while, switch, sequence, pick (temporisation))
- Correlation = session
- Scope découpage d'un processus en plusieurs parties
  - Pl. handler possibles par scope (compensation, fault, event )

# Aperçu de WSBPEL

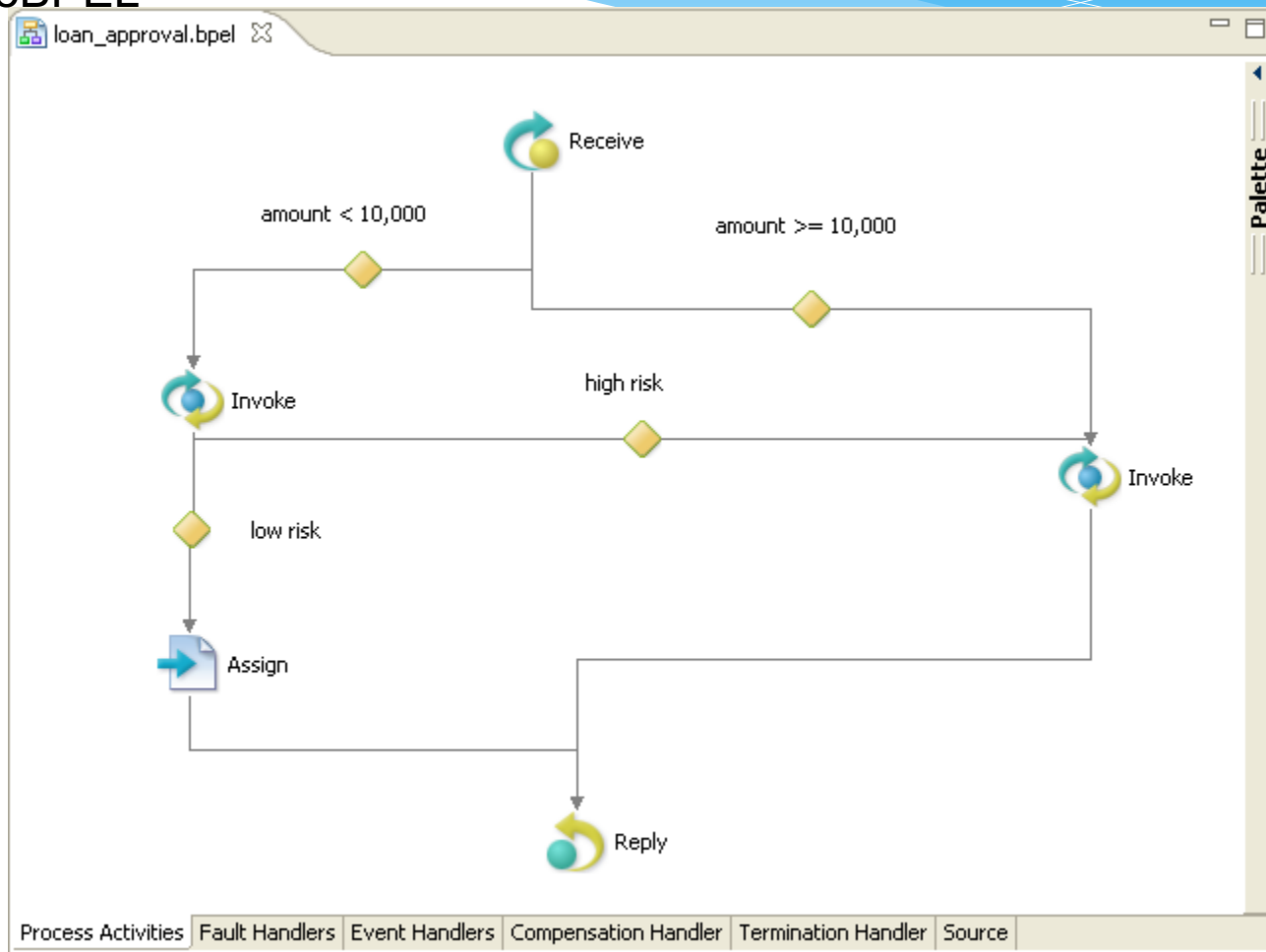
Avec ActiveBPEL





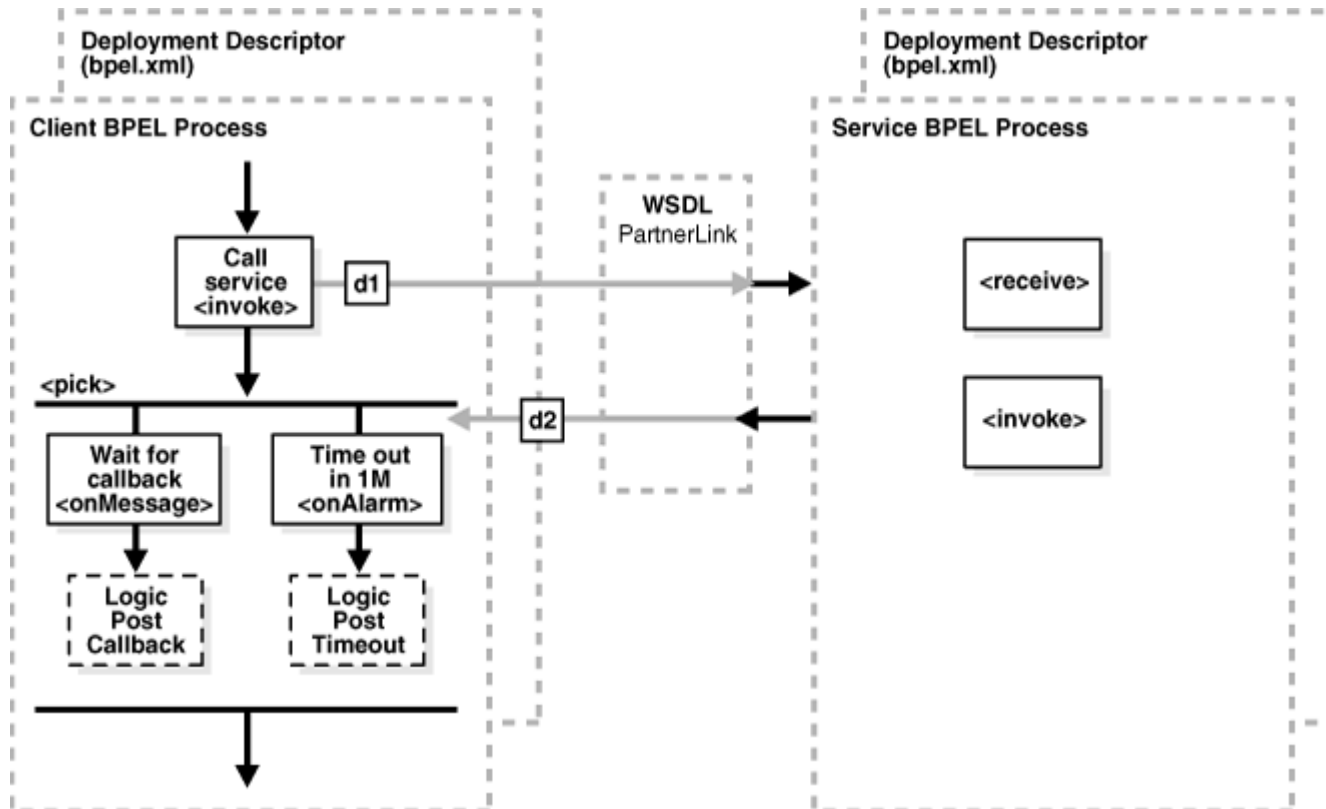
# Aperçu de WSBPEL

Avec ActiveBPEL



# Aperçu de WSBPEL

## Avec ActiveBPEL Le pick



# WS-BPEL

pl. moteurs

- Websphere, bpel process manager, biztalk, bpelmaestro
- Activebpel, pxe, twister

• Limitations

- Service composite centralisé (si moteur tombe en panne, ...)
- Schéma de composition statique

## Chorégraphie de services

- Comportement global basé sur les interactions des services entre eux.
- Chaque service web mêlée dans la chorégraphie connaît exactement quand ses opérations doivent être exécutées et avec qui l'interaction doit avoir lieu.
- Services font office de services et de clients vers d'autres services

## Chorégraphie de services

- Description des interactions de service uniquement de pair à pair
- Pas de processus, chaque service connaît les actions à effectuer par rapport aux messages reçus
- Langages standards de description de choregraphies
  - en XML WS-CL ou WSCI
  - Description des messages
  - Ordre des messages
  - ne définit pas un processus global
- Travaux de recherche sur composition dynamique