Processus métier et services : Illustration à travers WS-BPEL

E.Coquery

emmanuel.coquery@univ-lyon1.fr

http://liris.cnrs.fr/~ecoquery
→ Enseignement → TIW5

Processus métier [1]

- Ensemble d'actions
- Visant à réaliser une tâche de haut niveau
- Ayant un sens métier
  - En général pas informatique
- Comment l'implémenter dans une architecture SOA ?

1. Business Process (BP)

Retour sur la commande de glaces

Représenter un processus de traitement d'une commande de glaces

- Commande
- Paiement
- Livraison
- ...

Spécificités

- Plusieurs acteurs
  - Pas toujours connus au démarrage
- Temps d'exécution long
  - Transactions de longue durée
  - Nécessité d'un état persistant pour le processus
  - Plusieurs instances simultanées pour un même processus
- Transactions complexes
  - Multiparties
  - Non annulables, mais compensables
- Vue à haut niveau
  - Détails gérés par les services

WS-BPEL

- Langage pour l'orchestration de services
- Décrit un enchaînement d'actions (workflow)
- Langage et moteur d'exécution dédiés au processus métier :
  - Acteurs multiples et dynamiques
  - Transaction complexes
  - Longue durée

BPEL : structure de base

- Syntaxe XML
  - targetNamespace
  - imports possibles
- Blocs

```
<scope name="toto">...</scope>
```

Glaces : quels acteurs, quelles actions ?

- Client
  - Passe la commande
  - Paie
- Préparateur de glaces
  - Prépare les glaces à livrer
- Livreur
  - Prend les glaces et effectue la livraison
- Banque
  - Effectue le transfert d'argent lors du paiement
- Processus de traitement
  - Orchestre les interactions
  - Prévient le client du bon déroulement des actions

Acteurs en WS-BPEL

Représentés par des services, exposés :

- Par le process lui-même
- Par des services externes

Définition

- Dans le BPEL :
  - partnerLink (= nom + partnerLinkType)
  - Quel role est joué par
    - le process (`myRole`)
    - le partenaire (`partnerRole`)

  Dans le WSDL :
  - partnerLinkType (= mapping role $\leftrightarrow$ portType)

Souvent un seul rôle

- sauf pour les process interactions asynchrones

Exemple glaces : myRole

BPEL :

```
<bpel:partnerLink
    name="Client"
    partnerLinkType="tns:CommandePLT"
    myRole="processRole" />
```

WSDL :

```
<plnk:partnerLinkType name="CommandePLT">
    <plnk:role name="processRole"
               portType="wsdl:CommandPortType" />
</plnk:partnerLinkType>
```

+ Banque

Exemple glaces : partnerRole

BPEL :

```
<bpel:partnerLink
    name="Livreur"
    partnerLinkType="tns:LivreurPLT"
    partnerRole="livreurRole" />
```

WSDL :

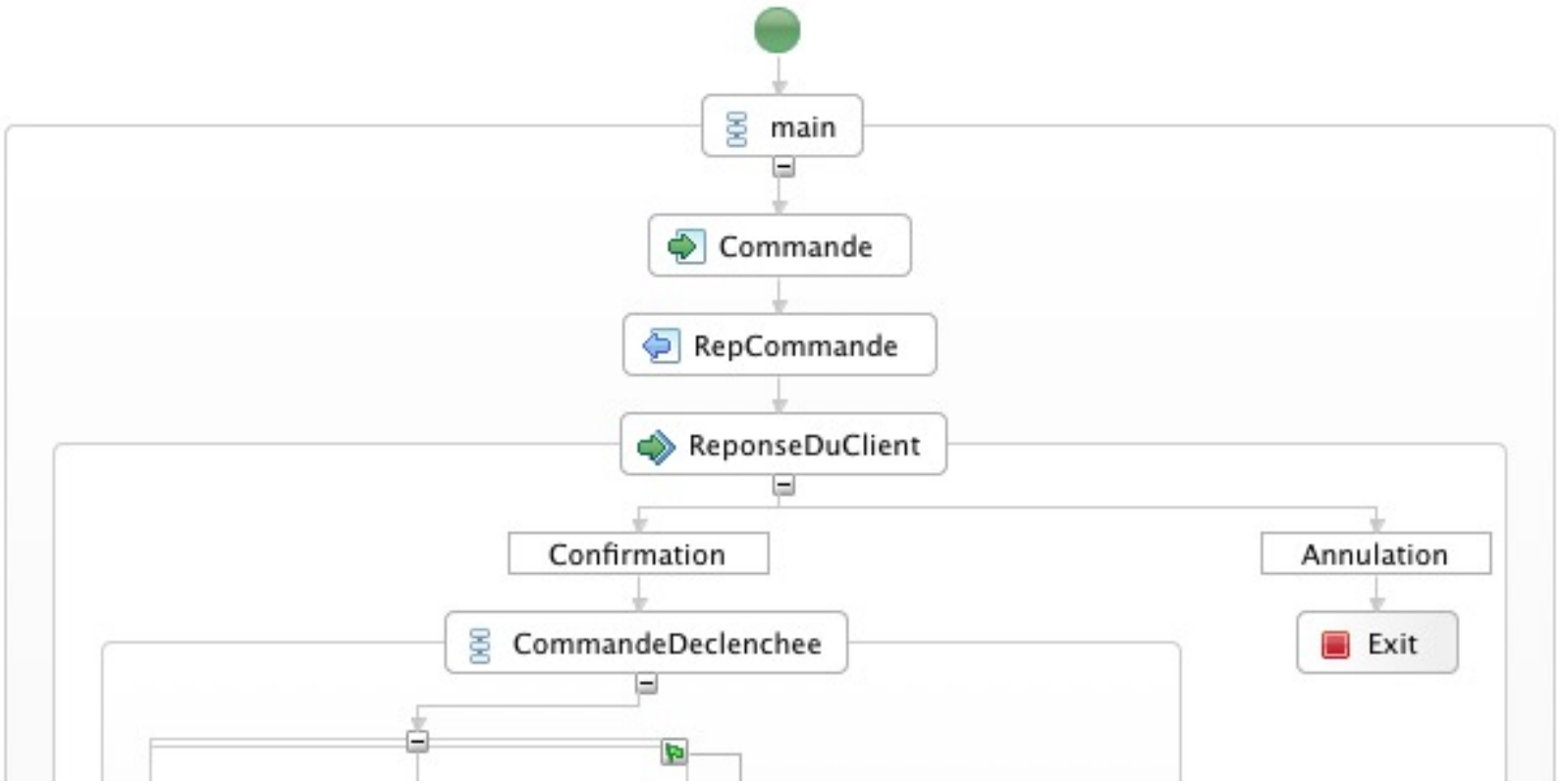
```
<plnk:partnerLinkType name="LivreurPLT">
    <plnk:role name="livreurRole"
                portType="wsdl:LivraisonPortType" />
</plnk:partnerLinkType>
```

+ LivraisonLookup, ClientInfo

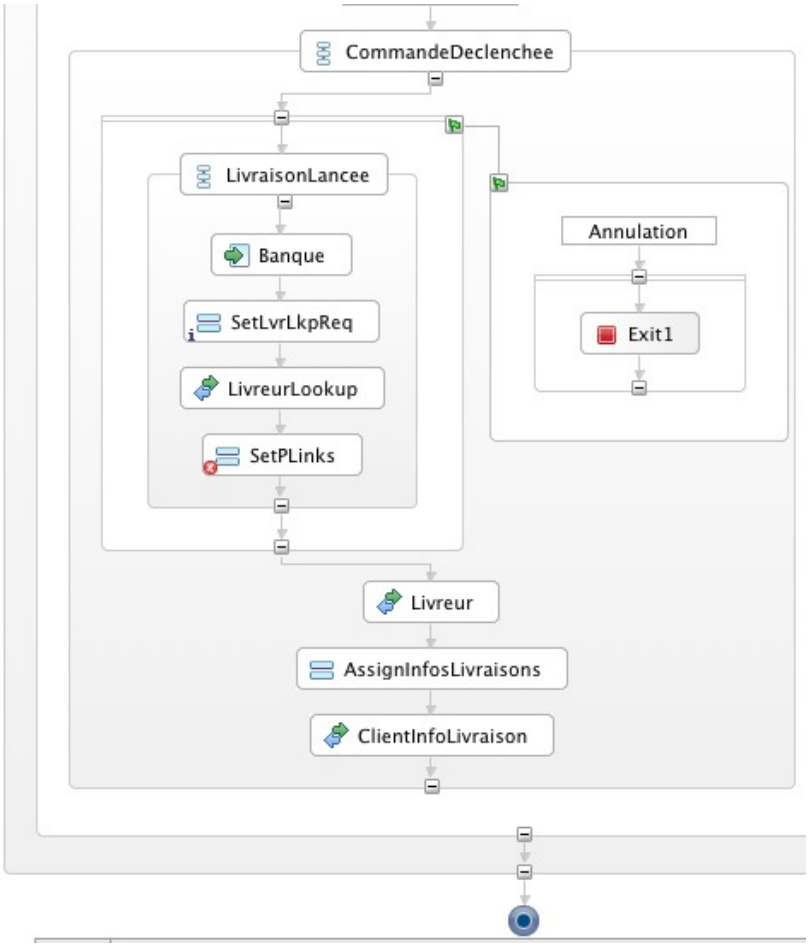
Workflow en BPEL

- Machine à états
- 1 état $\leftrightarrow$ une action
  - ou un ensemble d'action : nesting via les `scope`
- Transitions = relation de dépendance
- Instructions de contrôle classiques en plus
  - pour simplifier la modélisation

Workflow des glaces - 1



Workflow des glaces - 2



Première action

- Engendre une nouvelle instance
- Déclenchement par un événement
  - En général : réception d'un message
- Attribut `createInstance="yes"`
- Glace : réception d'un message de commande

Variables

- 4 sortes :
  - message
  - élément XML
  - type complexe
  - type simple

```
<bpel:variable name="CommandeV"
    messageType="ns:CommandeGlaceRequest" />
```

Contrôle

- Séquences
  `<sequence> ... </sequence>`
- If then else
  `<if><condition>...</condition>...`
  `<elseif><condition>...</condition>...</elseif>`
  `<else>...</else></if>`
- Boucles (while/ repeat until)
  `<while><condition>...</condition>...</while>`
  `<repeatUntil>...<condition>...</condition>`
  `</repeatUntil>`

Interactions avec les *partners*

Roles

- Appel à un service (`partnerRole`)
  - Nécessairement synchrone et bloquant
  - `<invoke ...`
- Implémentation d'un endpoint (`myRole`)
  - Synchrone également
  - `<receive ...`
    - bloquant
  - `<reply ...`

Valeurs en entrée et en sortie

- Variables
- + XPath / XQuery / ...

Exemple glaces : réception de commande

```
<bpel:receive  name="Commande"
     partnerLink="Client"
     operation="CommandeGlace"
     createInstance="yes"
     variable="CommandeV" />

<bpel:reply  name="RepCommande"
     partnerLink="Client"
     operation="CommandeGlace"
     variable="RepCommandeV">
 ...
</bpel:reply>
```

Exemple glaces : recherche d'un livreur

```
<bpel:invoke  name="LivreurLookup"
    partnerLink="LivraisonLookup"
    operation="RechercheLivreur"
    inputVariable="LivraisonLookupRequestV"
    outputVariable="LivraisonLookupResponseV" />
```

receive à plusieurs possibilités : pick

```
<bpel:pick name="ReponseDuClient">
    <bpel:onMessage partnerLink="Client"
        operation="Confirmation"
        portType="ns:CommandPortType"
        variable="ConfirmationV">
...
    </bpel:onMessage>
    <bpel:onMessage partnerLink="Client"
        operation="Annulation"
        portType="ns:CommandPortType"
        variable="AnnulationV">
        <bpel:exit name="Exit" />
    </bpel:onMessage>
</bpel:pick>
```

Affectations

`<assign><copy><from>...</from><to>../<to></copy>...</assign>`

Depuis / vers :

- Variables
- Messages
- PartnerLinks

Expressions

- XPath / XQuery
- Pour la source et la destination

Atomicité

- `<assign>` = ensemble d'affectation
- `<assign>` $\leftrightarrow$ transaction ACID

Adresses de services

- `<sref:service-ref>` : élément contenant l'adresse
- peut contenir une adresse au format WS-Addressing
  - `<wsa:EnpointReference>`

- Utile pour
  - les *callback*
  - la parallélisation d'appels
  - l'utilisation d'annuaires de services (lookupLivraison)

Corrélations

Valeurs utilisées pour aiguiller les messages sur la bonne instance

- Propriété (= nom + type) :
  - valeur utilisée pour des corrélations
  - $\in$ WSDL
- Alias (= propriété + message + emplacement)
  - où trouver la propriété dans le message
  - aussi bien en entrée qu'en sortie
  - $\in$ WSDL
- *Correlation set* (= ensemble de propriétés)
  - tuple de valeurs à utiliser pour une corrélation
  - peut être référencé dans une interaction
  - `initiate="yes"`
  - $\in$ BPEL

Corrélation de réponses

Plusieurs messages

- sur un même parnerLink
- peuvent être reçus
- avant que la réponse du premier ne soit envoyée

→ `messageExchange="..."` pour corréler un `receive` avec un `reply`

Exemple glaces : correlation sets, WSDL

```
<vprop:property  name="CmdIdPty"
    type="xsd:string" />

<vprop:propertyAlias
    messageType="wsdl:CommandeGlaceResponse"
    part="parameters"
    propertyName="tns:CmdIdPty">
     <vprop:query><![CDATA[@cmdid]]></vprop:query>
</vprop:propertyAlias>
```

Exemple glaces : correlation sets, BPEL

```
<bpel:correlationSet  name="CmdIdSet"
    properties="tns:CmdIdPty" />
 ...
<bpel:reply  name="RepCommande"
    partnerLink="Client"
    operation="CommandeGlace"
    variable="RepCommandeV">
     <bpel:correlations>
          <bpel:correlation  initiate="yes"
              set="CmdIdSet" />
     </bpel:correlations>
</bpel:reply>
```

Parallélisme

- ```
  <forEach counterName="N" parallel="yes">
      <startCounterValue>1</startCounterValue>
      <finalCounterValue>5</finalCounterValue>
      <completionCondition>
          <branches>3</branches>
      </completionCondition>
      ...
  </forEach>
  ```
- La valeur du compteur peut être utilisée dans une expression XPath
  - `$lookupResponse/services/service[$N]`
  ```

Concurrence : Flow

- `<flow>`
- Lien (= dépendance) `<link>`
- Actions :
  - source : action terminée → activation d'un lien
  - cible : combinaison de liens activés → action démarrée
- Combinaison : expression booléenne
  - variables : liens
  - valeur : vrai si lien activé
- Valeur expression :
  - vrai : déclenchement de l'action
  - indéterminée : rien ne se passe
  - faux : erreur
- `suppressJoinFailure`
  - `no` : erreur si une action ne peut être déclenchée
  - `yes` : pas d'erreur si une combinaison s'évalue à `false`

Handlers

- `<faultHandler>`
  - $\leftrightarrow$ try/catch
  - faute SOAP $\leftrightarrow$ exception
- `<terminationHandler>`
  - Appelé à la fin d'un `<scope>`
- `<eventHandlers>` $\rightarrow$ `<onEvent>`
  - similaire au `pick`
  - pas de point d'attente bloquant

Exemple glace : eventHandlers

```
<bpel:eventHandlers>
    <bpel:onEvent partnerLink="Client"
        operation="Annulation"
        portType="ns:CommandPortType"
        variable="AnnulationV"
        messageType="ns:AnnulationRequest">
        <bpel:scope>
            <bpel:exit name="Exit1" />
        </bpel:scope>
        <bpel:correlations>
            <bpel:correlation set="CmdIdSet"
                initiate="no" />
        </bpel:correlations>
    </bpel:onEvent>
</bpel:eventHandlers>
```

Versions de processus

- Exécution de longue durée
- Mise à jour lors d'une exécution
- Migration parfois impossible :
  - Changement des données / d'un schéma
  - Disparition d'états (actions)
- Plusieurs version d'un même process simultanément

Déploiement

Informations

- Essentiellement les partnerLinks :
  - Les partenaires distants (e.g. lookupLivraison)
  - Les interfaces du process
    - partenerLinks avec `myRole=...`
- Persistence de l'état/des données
  - In memory : plus rapide
  - Dans un SGBD
- Propriété (constantes) pour la configuration du process *e.g.* :
  - valeur de seuil utilisée dans une condition métier
  - niveau de parallélisme /seuil de déblocage d'un forEach parallèle

Processus abstraits

- Process avec des parties non définies
  - Correspondrait à des trous dans le process
  - `<opaqueActivity>`
- Peut être instancié en un ≪ processus exécutable ≫.
  - sans `opaqueActivity`
- Possibilité d'utilisation sous forme de patron (*template*) de services

Références

- BPEL Primer
  `https://www.oasis-open.org/committees/download.php/23964/wsbpel-v2.0-primer.htm`
- Documentation d'Apache ODE
  `http://ode.apache.org/userguide/`