Composition de services

Enterprise Integration Pattern via Apache Camel Workflows

Composition : A+B → C

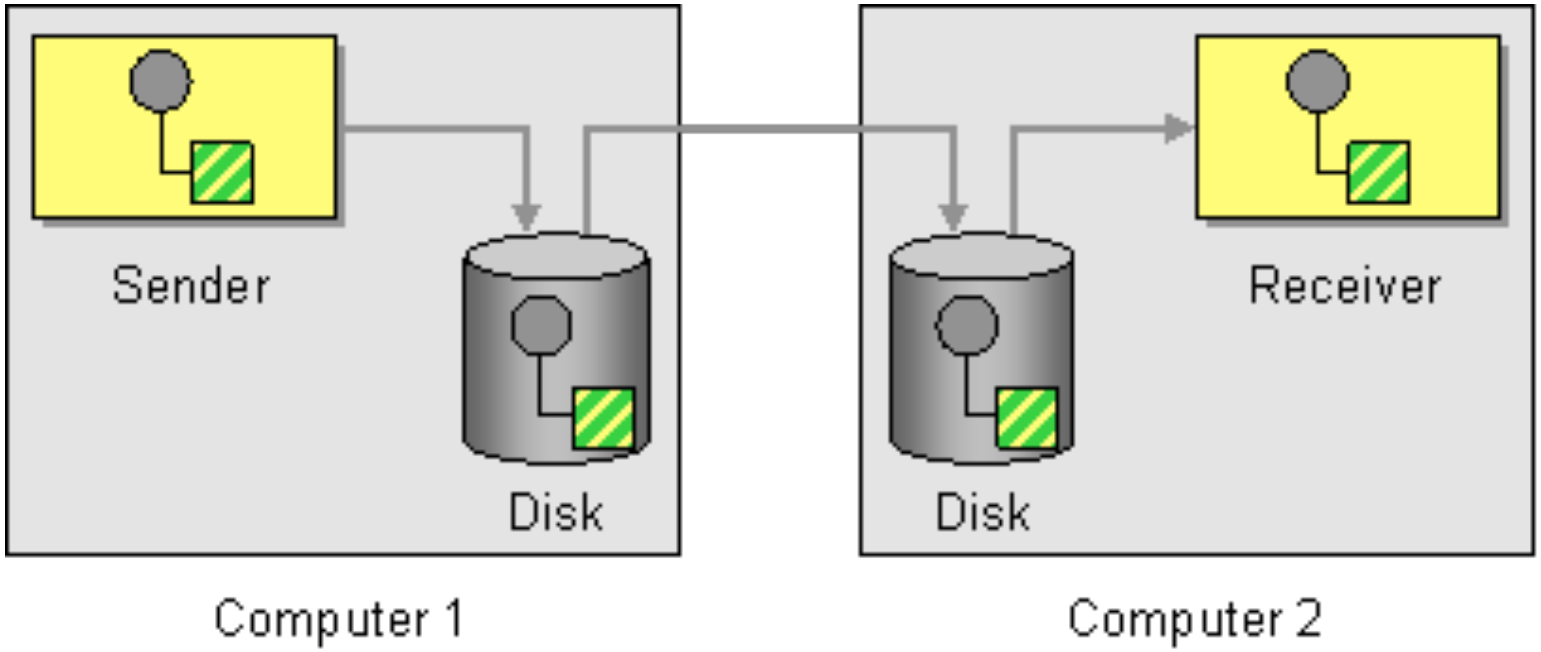
- Dans le cadre de gros systèmes d'information :
  - logique applicative complexe
  - processus décomposables
  - décentralisation des traitements

- Un service peut être vu comme une interface sur:
  - un / des processus métier
  - utilisant à leur tour des services
    - réalisant des actions à plus petit échelle
  - "analyse descendante" vers SOA
    - attention aux limites de l'analogie !

Enterprise Integration Patterns

- *Design patterns* spécifiques à l'intégration dans les systèmes d'information
- Basé sur la manipulation et la transmission de messages métier
- Catégories:
  - Canaux
    - point à point, publish/subscribe, garanti (avec persistance), adaptateurs, ponts, bus
  - **Routage**
  - **Transformation**
  - Système

Canaux persitants



e.g. fichiers, JMS

Transformations

- Enveloppe
- Enrichissement, filtrage de contenu
- Vérifications
- Normalisation

Routage

- Basé sur le contenu
- Dynamique
- Filtrage
- Découpage (/Aggrégation)
- Diffusion/Aggrégation
- (Ré)ordonnancement
  - Ordonnancement de messages
  - Ordonnancement de services
- Workflows

Apache Camel

- Cadre applicatif facilitant la mise en place d'EIP
- Permet de spécifier des *routes* intégrant différents composants de routage et de filtrage
  - Configuration via Java, XML, Scala ...
- Traite des messages depuis/vers des points d'accès (Transports: cxf, http, jms, etc ...)

Camel: mise en place

- création d'un CamelContext
  - fait automatiquement dans certains cas (e.g. déploiement via Spring dans ServiceMix)
- configuration via l'ajout de routes et de points d'accès:

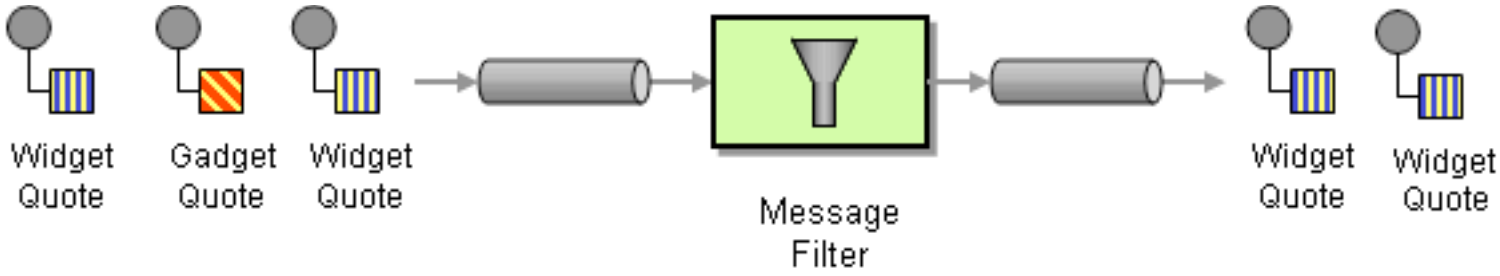
```
public class MyRouteBuilder extends RouteBuilder {
    public void configure() throws Exception {
        from("timer:myTimerEvent?fixedRate=true")
        .setBody(constant("Hello World!")).
        to("log:ExampleCamelRoute");
    }
}
```

Pipeline

- Enchaînement d'appels à des services
  - La sortie d'un service vient en entrée d'un autre
- A combiner avec des filtres/ transformations

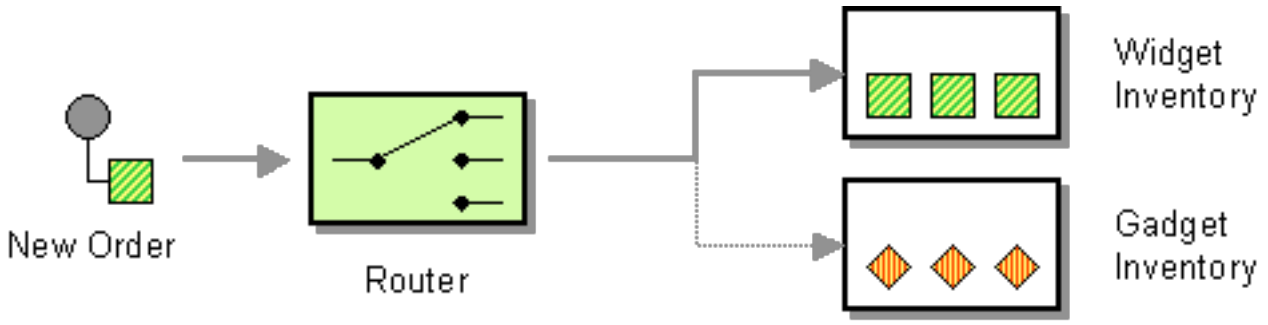
Filtres

- Supprimer certains messages
  - Selon le contenu




```
from("seda:a")
        .filter(header("foo").isEqualTo("bar"))
            .to("seda:b");
```

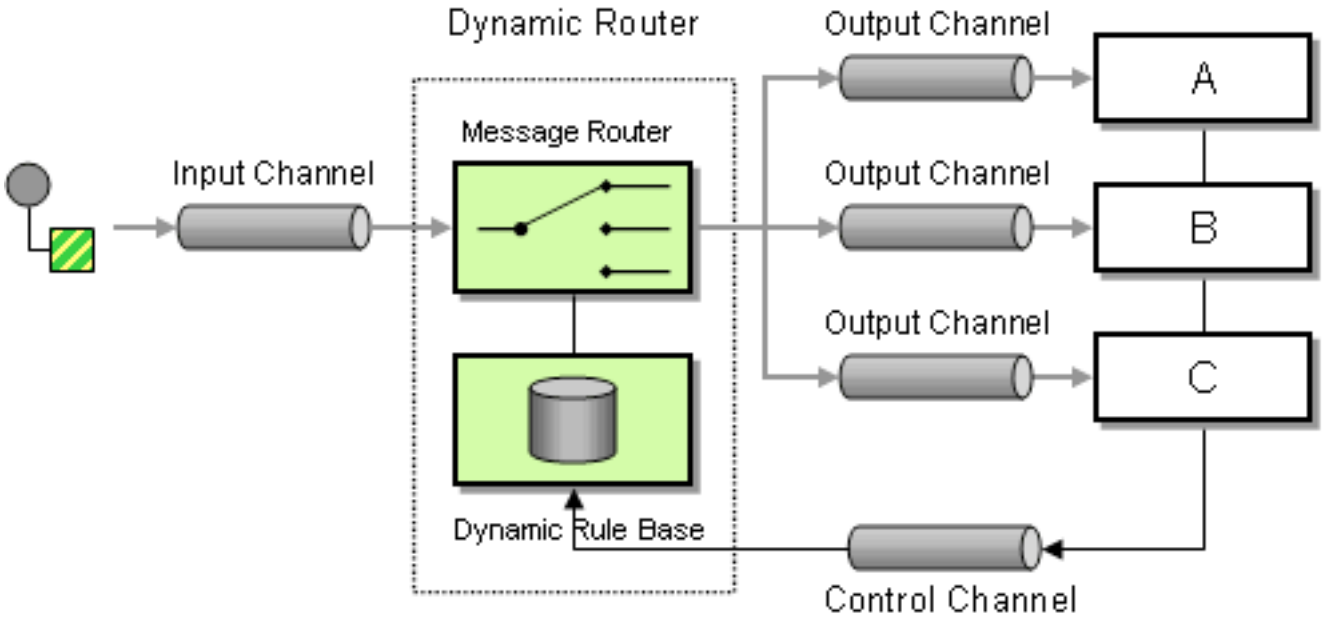
Routage basé sur le contenu




```
from("seda:a")
    .choice()
        .when(header("orderType").isEqualTo("widget"))
            .to("seda:b")
        .otherwise()
            .to("seda:d");
```

Routage dynamique

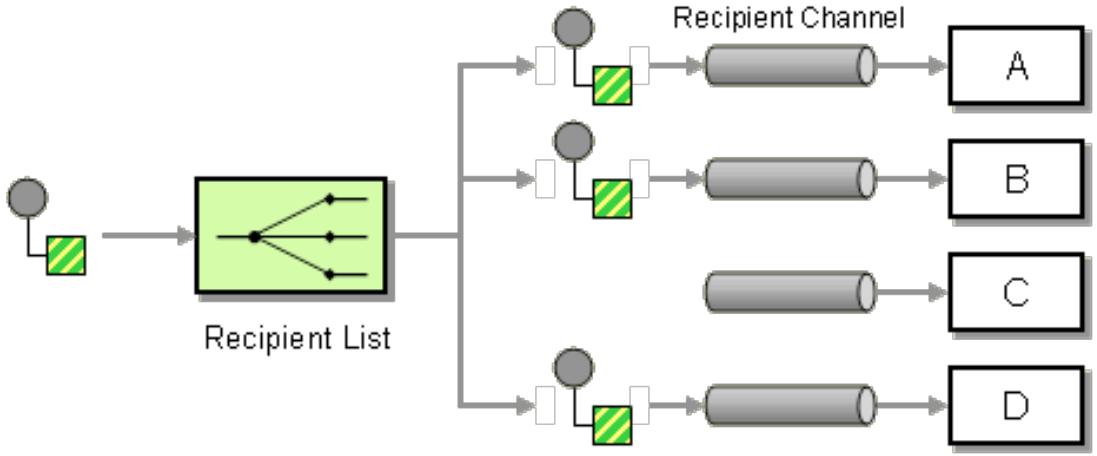
- Choix de la destination selon l'état du système



Routage dynamique - 2

```
from("direct:start")
    // use a bean as the dynamic router
    .dynamicRouter(bean(DynamicRouterTest.class, "slip"));

public String slip(String body) {
    bodies.add(body);
    invoked++;
    if (invoked == 1)
        return "mock:a";
    else if (invoked == 2)
        return "mock:b,mock:c";
    else if (invoked == 3)
        return "direct:foo";
    else
        return "mock:result";
}
```

Diffusion

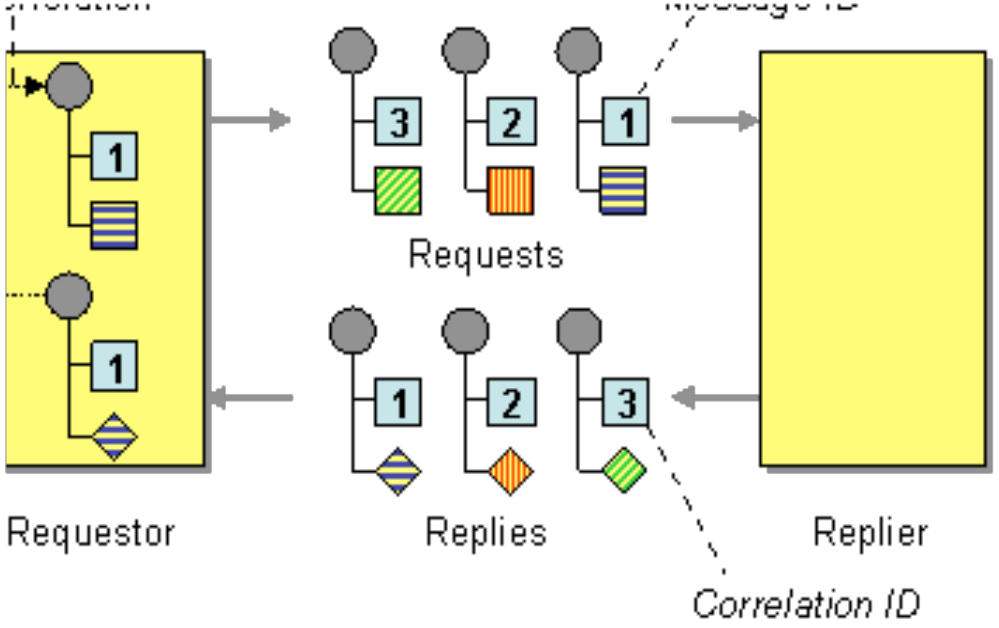


- La liste peut être statique ou dynamique

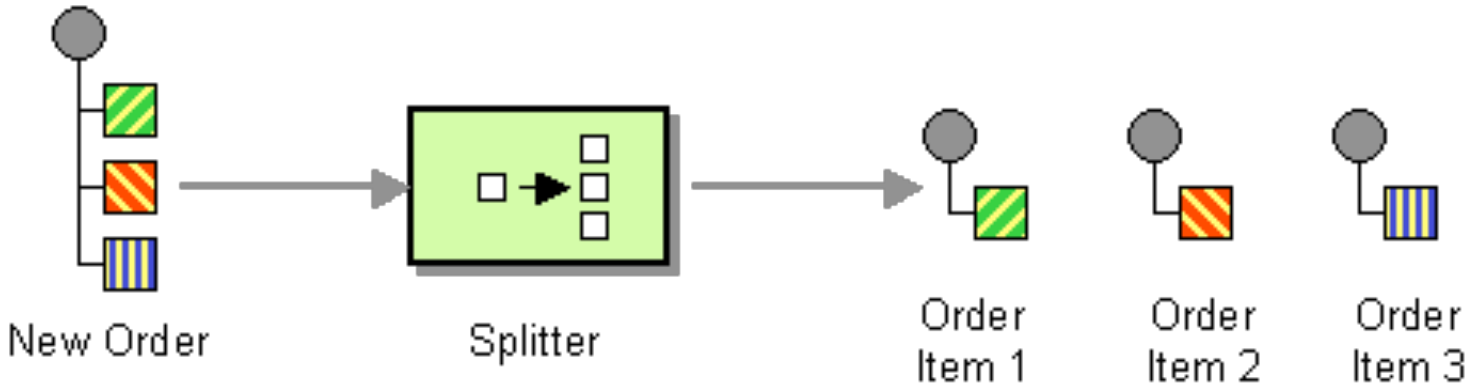
```
from("seda:sender")
        .multicast().to("seda:a", "seda:b", "seda:d");
from("seda:a")
        .recipientList(header("foo"));
```

Corrélation

- Identifiant permettant attribuer une réponse à un client

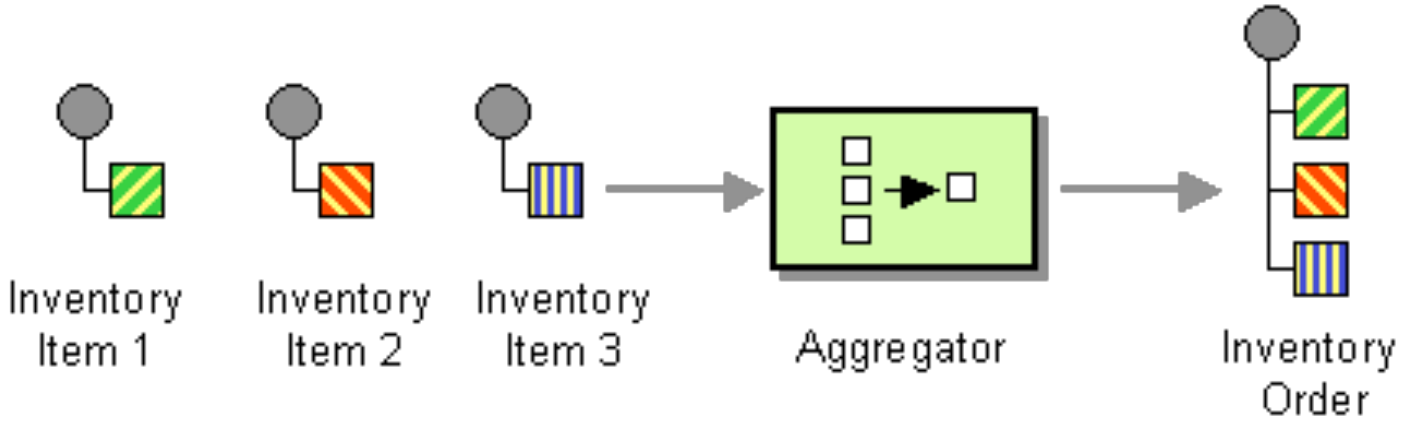


Division




```
from("activemq:my.queue")
    .split(xpath("//foo/bar"))
    .convertBodyTo(String.class)
    .to("file://some/directory")
```

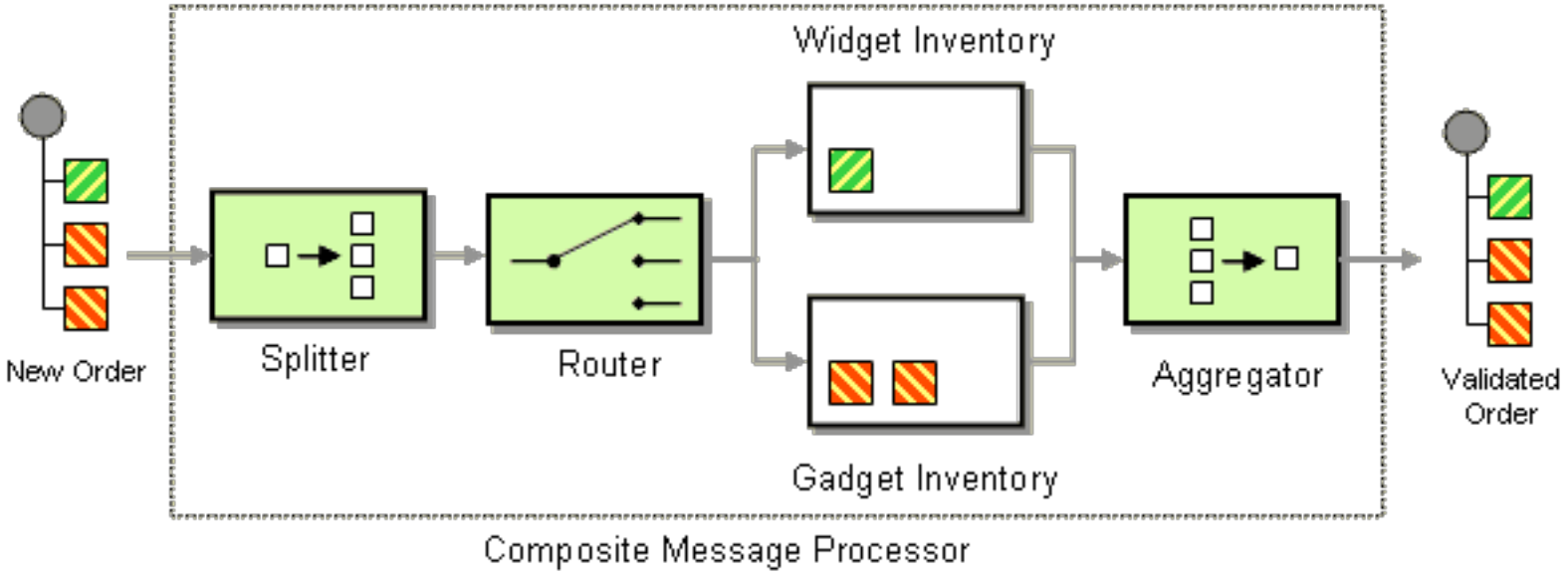
Aggrégation



- Création d'un message à partir de plusieurs
  - en général générés par différent traitements sur le message de départ

Traitements par parties

- chaque partie est traitée par le service approprié



Route Camel

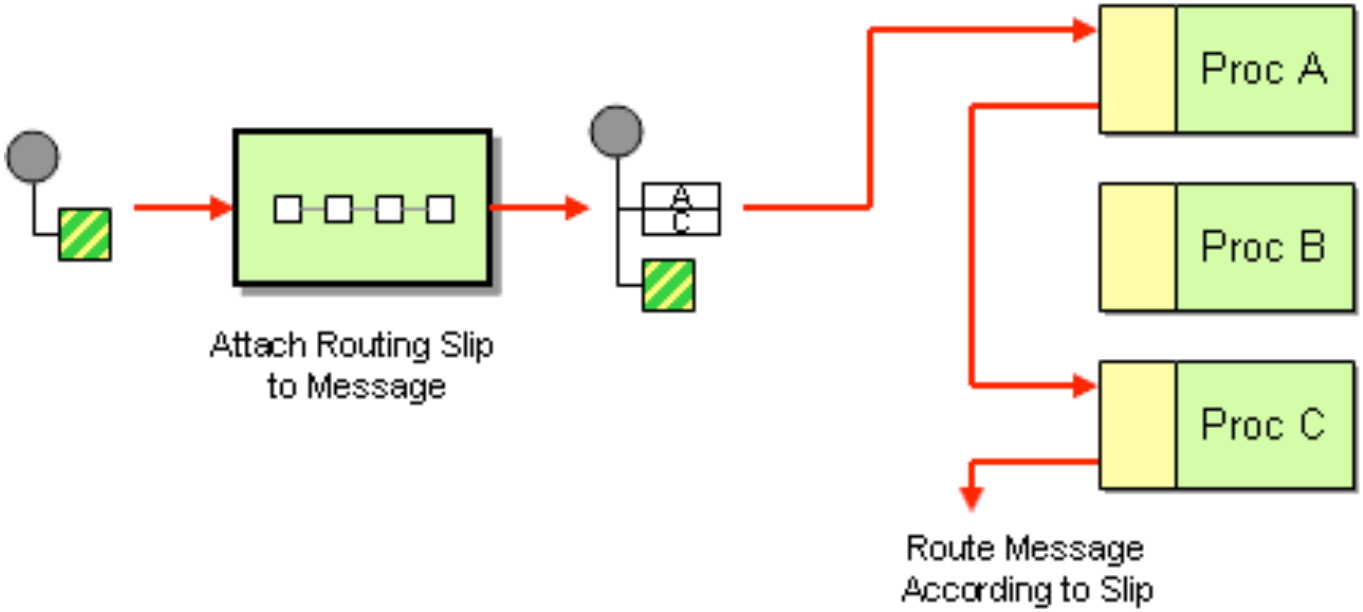
```
// split up the order so individual OrderItems can be validated by the
appropriate bean
from("direct:start")
    .split().body()
    .choice()
        .when().method("orderItemHelper", "isWidget")
            .to("bean:widgetInventory")
        .otherwise()
            .to("bean:gadgetInventory")
    .end()
    .to("seda:aggregate");

// collect and re-assemble the validated OrderItems into an order again
from("seda:aggregate")
    .aggregate(new MyOrderAggregationStrategy()).header
("orderId").completionTimeout(1000L)
        .to("mock:result");
```

Réordonnancement de messages

- Tri des messages selon une valeur
- Batch
  - Attente de messages
  - Puis tri et traitement
  - Déclenchement sur taille / timeout
- Stream
  - File de priorité
  - taille max / timeout

Ordonnancement dynamique de services

- Associer à un message une liste de services




```
from("direct:c")
  .routingSlip(header("aRoutingSlipHeader"),
                      "#");
```

Processus métier et *workflow*

- Workflow :
  - Ensemble d'actions / de tâches
  - Avec des dépendances
  - Décrit un processus métier

- Formalismes / langages:
  - Diagrammes de Gantt
  - Diagrammes de séquence
  - Petri-nets
  - BPMN (Business Process Model and Notation)
  - WS-BPEL

Processus métier

- Temps d'exécution long
  - Transactions de longue durée
  - Nécessité de rendre persistant l'état du processus
  - Plusieurs instances d'un même processus
- Vue à haut niveau
  - Détails gérés par les services

Parallélisme / concurrence dans un même processus

- Certaines parties du workflow doivent pouvoir s'exécuter de manière concurrente, éventuellement en parallèle
- Dépendance entre tâches
  - e.g. Gantt
  - Peut être conditionnée par l'état/les données du processus
- Synchronisation en fin de tâche
  - Toutes les dépendances sont terminées
  - Une dépendance est terminée
  - Combinaison

Événements concurrents

- Messages arrivant hors du flot normal
  - Erreurs
  - Demandes particulières
  - Annulations

- Prise en compte possiblement complexe
  - Interaction avec flot courant ?
    - Interruption
    - Modification

Transactions et ~~*rollback*~~

- Transaction
  - comme en BD
  - réparties (si processus distribué)
- Il n'est pas toujours possible de défaire une transaction
  - Impacts dans le monde "réel"
    - e.g. livraison effectuée
  - On peut en revanche la **compenser**:
    - e.g. renvoi des produits livrés
    - la compensation peut être une transaction

Processus distribués

- Exécution distribuée
- Potentiellement de manière concurrente
- Problématiques:
  - Expression du processus entier
  - Transactions
    - Début/Fin (~ commit 2/3 phases)
    - Compensation éventuellement distribuée
  - Corrélation

Orchestration vs Chorégraphie

- Orchestration
  - Un service gère le processus et appelle les autres services
  - Vue centralisée
  - Utilisation possible des langages de workflow
  - e.g. gestion d'un ticket dans un gestionnaire de bugs
- Chorégraphie
  - La gestion du processus est distribuée
  - BPMN permet en partie de les décrire
  - e.g. authentification à la Kerberos, paiement via une tierce partie en commerce électronique
- Pas forcément opposés
  - Points de vue différents sur un même processus
  - e.g. gestion des commandes chez un fournisseur

Instances et corrélation

- Exemple simple : identifiant de session
- Services non connus au démarrage de l'instance
- Dans le cas de processus distribués:
  - Identifiant pas nécessairement partagé
  - Un service peut intervenir dans plusieurs processus différents
  - Un nouvel intervenant doit pouvoir être aiguillé sur la bonne instance
    - e.g. nouvel arrivant sur une enchère en ligne

Data driven processes

- Alternative aux workflow
- État entièrement codé par les données
- Évolution exprimée à travers des règles modifiant les données
- A comparer avec la condition d'être *stateless* dans REST

Sources

- http://www.enterpriseintegrationpatterns.com/
  - Images  Gregor Hohpe and Bobby Woolf
- http://camel.apache.org
- http://www.oasis-open.org/committees/download.php/23964/wsbpel-v2.0-primer.htm