Messages, Bus et Patrons d’intégration

Emmanuel Coquery

Composition de services

Dans le cadre de gros systèmes d’information :

- Logique applicative complexe
- Processus décomposables
- Décentralisation des traitements

Un service peut en cacher d’autres (A=B+C)

Un service peut être vu comme une interface sur:

- un / des processus métier
- utilisant à leur tour des services
    - qui réalise des actions à plus petite échelle / moins complexes
- « analyse descendante » vers SOA
    - attention aux limites de l’analogie !

Quelques problématiques autour de la composition

- Liées aux processus métiers complexes
    - Transactions longues, instances multiples, concurrence, distribution
        - Point de vue: Orchestration vs chorégraphies
- Gestion des messages
    - Comment organiser la transmission
    - Quelle efficacité ?

Transactions de longue durée

- Persistance de l'état du processus
  - Stocker l'état (au sens automate) + données de chaque instance
  - Penser le process comme une gestion de ressources (type REST)
    - ! Impact sur la conception !

Transactions métier

- Attention à la répartition (commit à 2 ou 3 phases)
- Pas systématiquement de rollback
    - Impacts d'une action informatique dans le monde réel
        - e.g. livraison effectuée
- **Compensation** au lieu du rollback
    - e.g. renvoi des produits livrés, remboursement, etc
    - une compensation peut être une transaction

Parallélisme et concurrence

- Tâche à exécuter de manière concurrente / parallèles
- Gestion des dépendances entre tâches
    - Moteur de workflow
        - Exceptions dans le flot « normal »
    - Système de règles sur événements et données
        - ECA (Event-Condition-Action)
- Synchronisation des tâches
    - toutes / une /certaines tâches dont on dépend sont terminées

Orchestration vs chorégraphies

- Orchestration
  - Vue centralisée
  - Un service gère le processus et les appels aux autres services
    - e.g. gestion d’un ticket dans un gestionnaire de bugs
- Chorégraphie
  - Processus géré de manière distribuée
  - e.g. authentification type Kerberos, paiement via une tierce partie
- Pas exclusifs
  - Points de vue différents

Instances et corrélation

- Comment savoir si deux messages sont liés ?
    - (méta)données
- Exemple simple: identifiant de session
- Plus complexe:
    - si acteurs non connus au démarrage du processus
        - ou si un nouveau participant peut arriver en cours d’exécution
    - identifiants non partagés
    - interactions entre processus / service partagé entre plusieurs processus

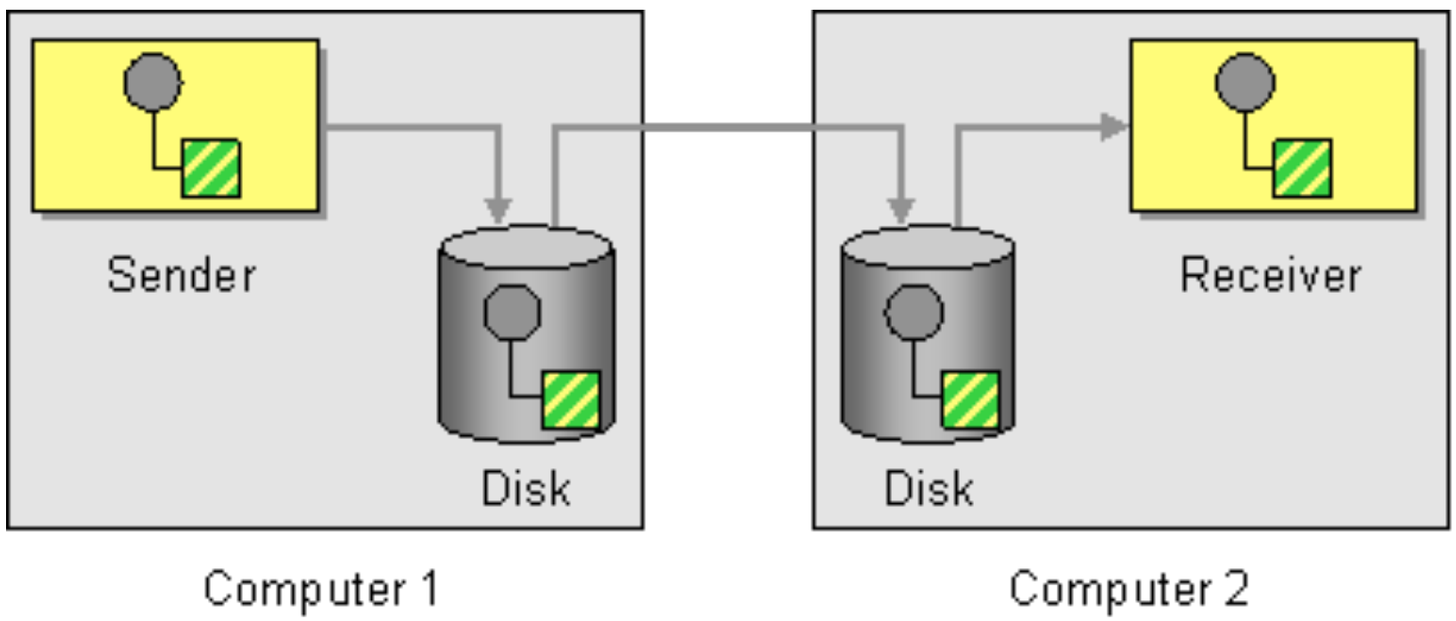
Limites des services simples dans un cadre SOA

- Garder l'aspect **couplage faible**
- Adresses / points d'accès
  - fixes / utilisation d'un annuaire
  - complexe si beaucoup de services
- Interactions complexes codées dans chaque client

Intégration de services: enterprise integration patterns

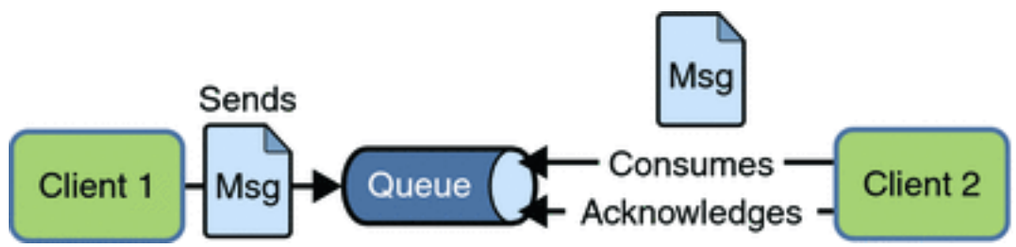
- Design patterns spécifiques à l'intégration dans les systèmes d'information
    - Basé sur la manipulation et la transmission de messages métier
- Catégories:
    - Canaux
        - point à point, publish/subscribe, garanti (avec persistance), adaptateurs, ponts, bus
    - Routage
    - Transformation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

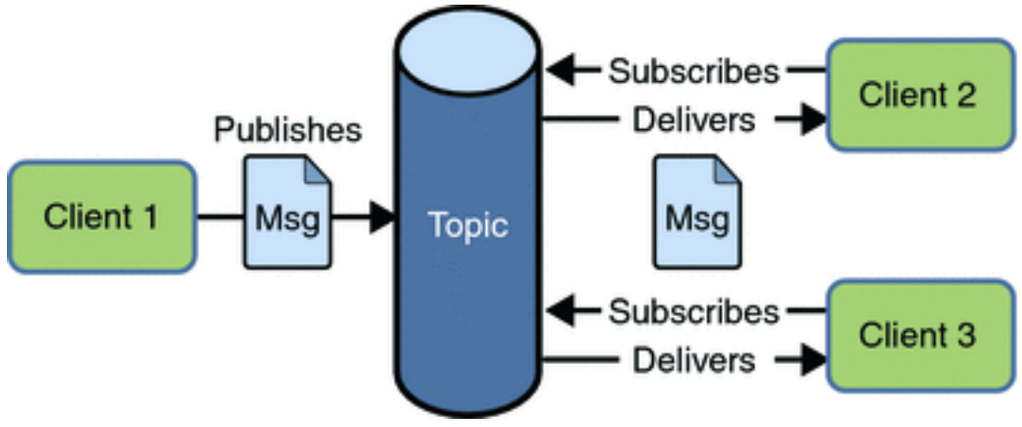
Canaux persistants



- Pour ne pas perdre de message

Canaux: modes de transmission





- Queue: deliver exactly once
- Topic: broadcast

Source: JEE tutorial

Canaux: types d’interaction pour le receiver

- Synchrone
    - Le consommateur récupère explicitement le message avec un appel bloquant
- Asynchrone
    - Le consommateur enregistre un callback

Similaire à synchrone vs asynchrone en programmation client/serveur

Frameworks de (transmission de) messages

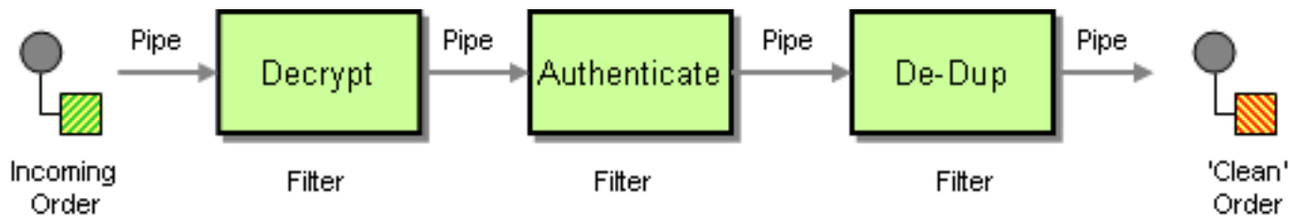
Aller au delà de REST / SOAP+HTTP

- Performance (binaire, protocole dédié, etc)
- Intégration de
  - capacités de routage
  - garanties sur les messages (exactly once, persistance)
- Rôle de tampon en cas de charge temporaire

Quelques implémentations

- RabbitMQ: queues, deliver exactly once, clusters, routing, AMPQ, langages variés
- Kafka: topics, clusters, traitement de flux, langages variés, bonne capacité d'absorption des pics de messages
- ZeroMQ: basique, léger, efficace, langages variés
- ActiveMQ: topics, queues, apache, EIP via intégration avec Apache Camel, AMPQ, JMS, langages variés
- JMS: standard Java
- AMPQ: standard neutre vàv plateforme et langage

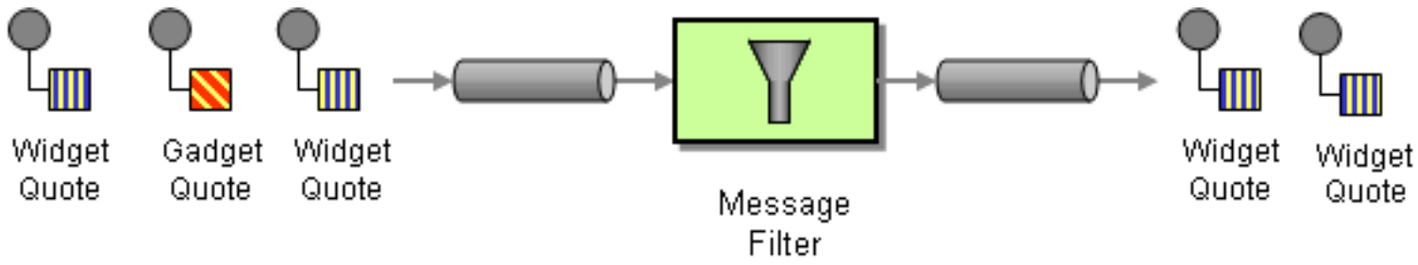
Pipeline de services

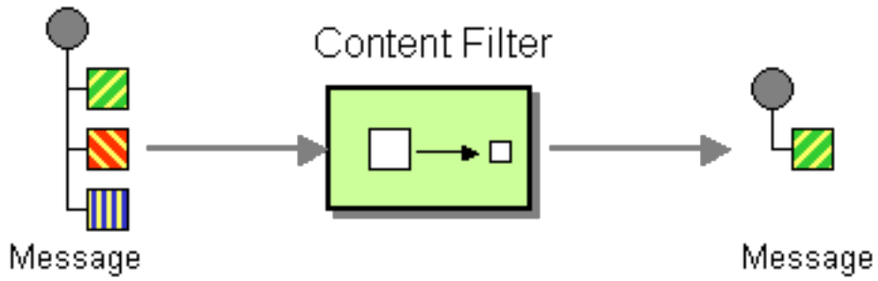


Traitement composite de messages

- 1 étape = 1 appel à un service
- réponse d’un service = entrée du service suivant

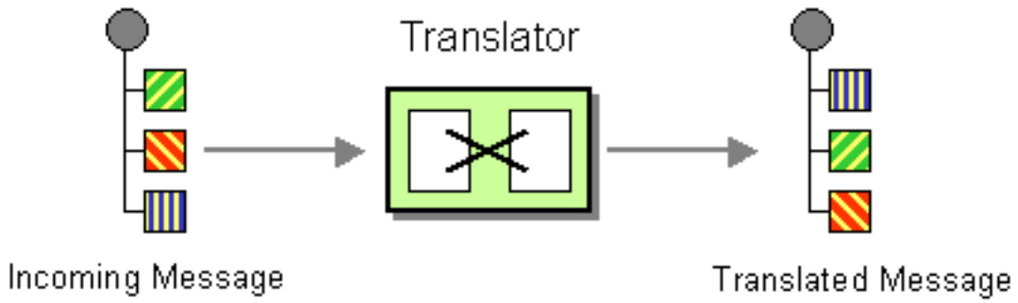
Filtres

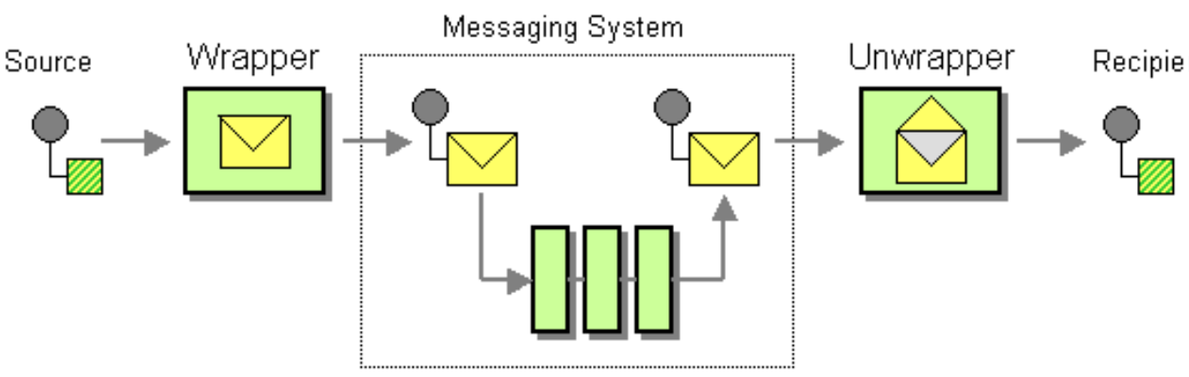


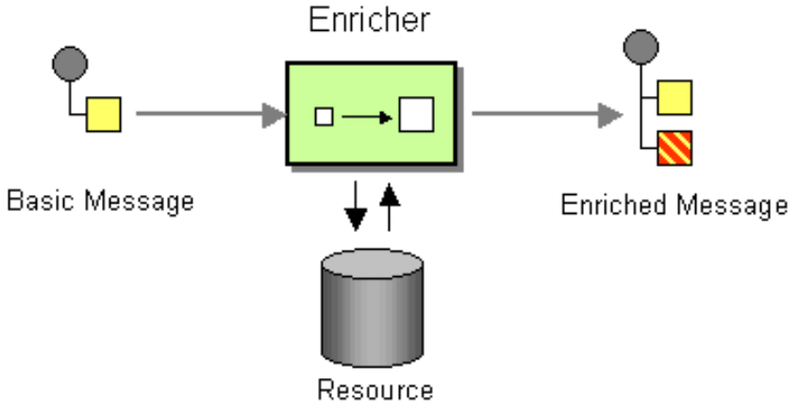


- Peut supprimer (des parties d’) un message
- Se combine bien avec les pipelines

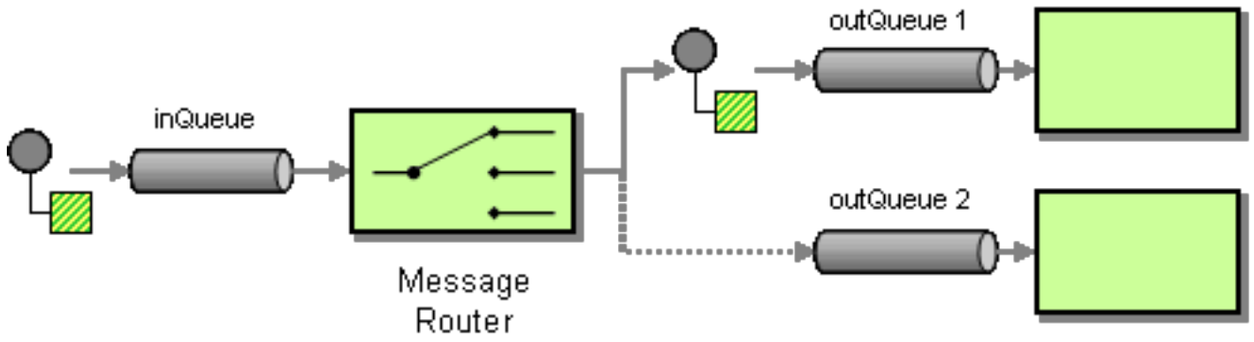
Transformations

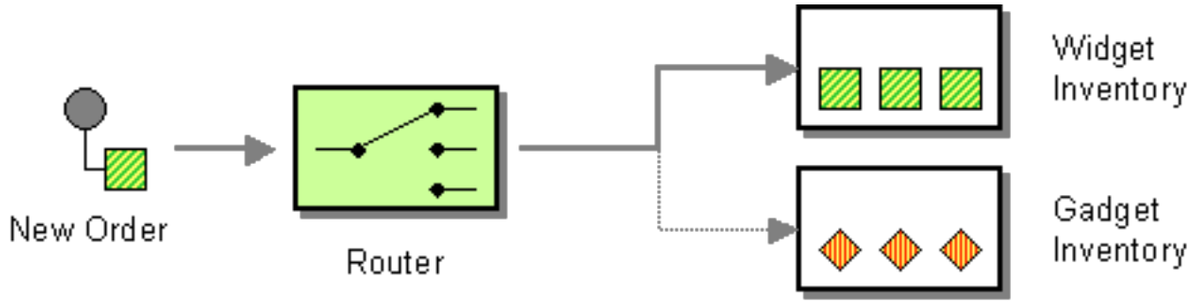






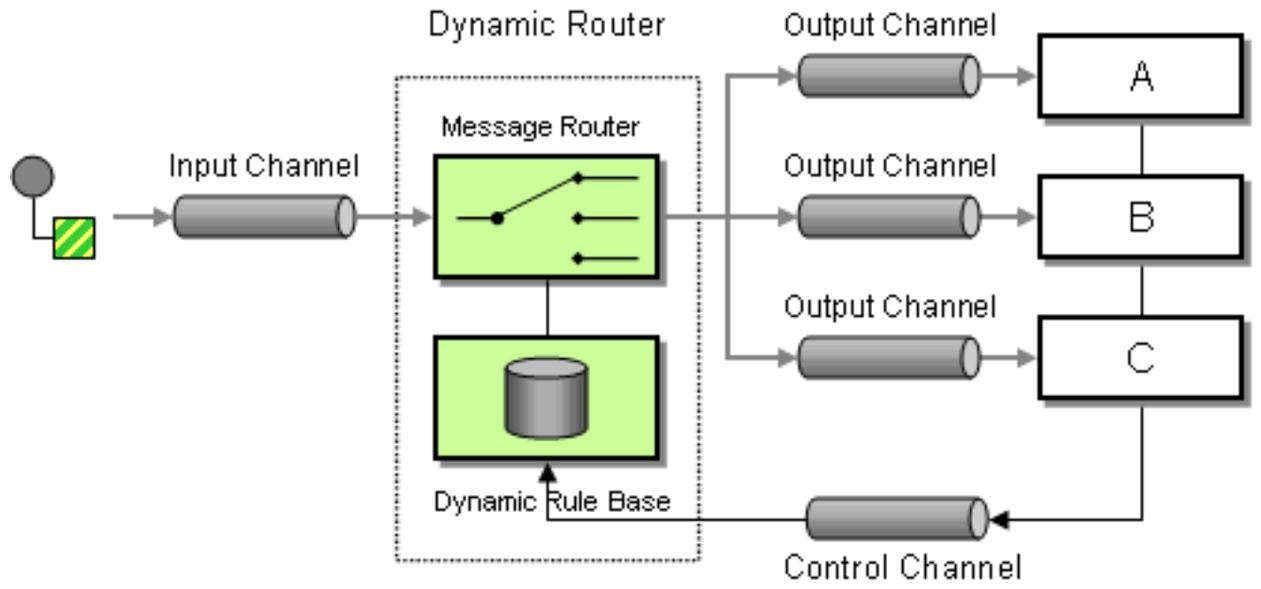
Routage basés sur les messages





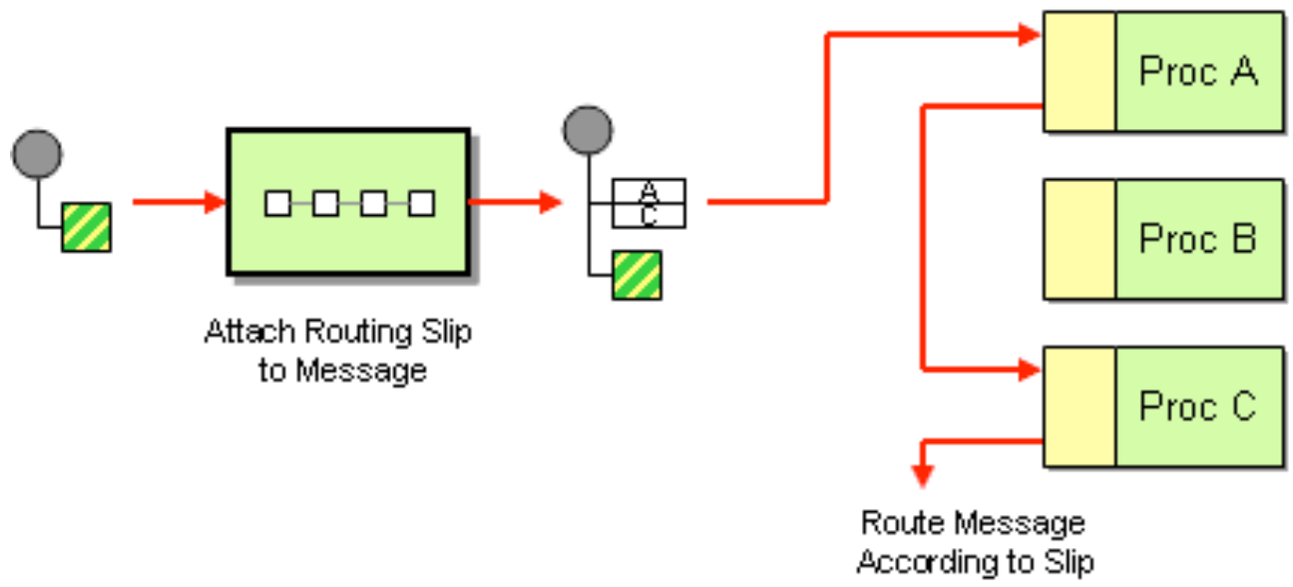
- Seul le contenu du message (header / payload) est utilisé pour décider de la destination
- Possibilité de transformation préalable pour gérer les cas complexes

Routage dynamique

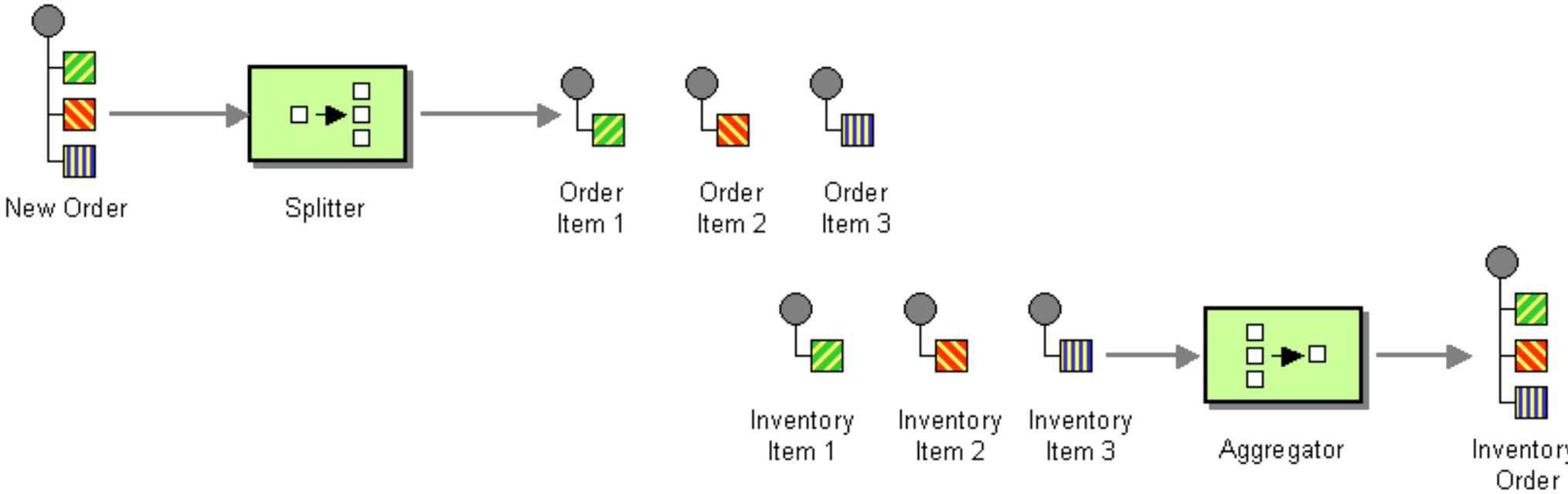


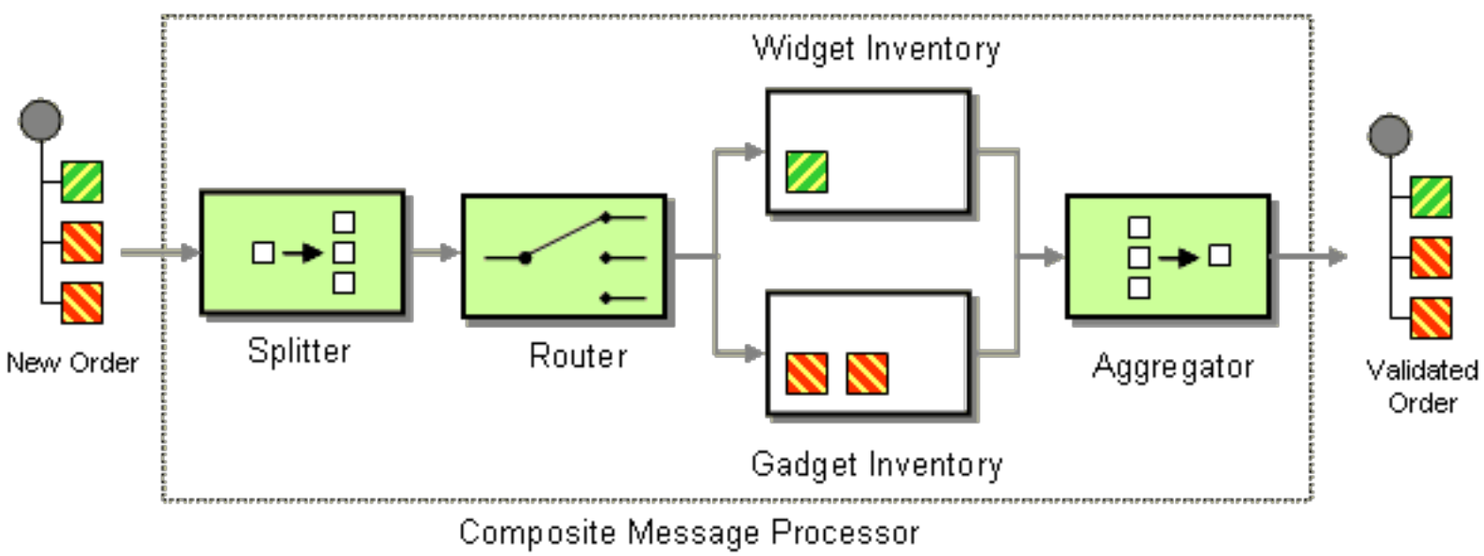
- Choix de la destination selon l'état du système
  - e.g. équilibrage de charge

Pipeline dynamique



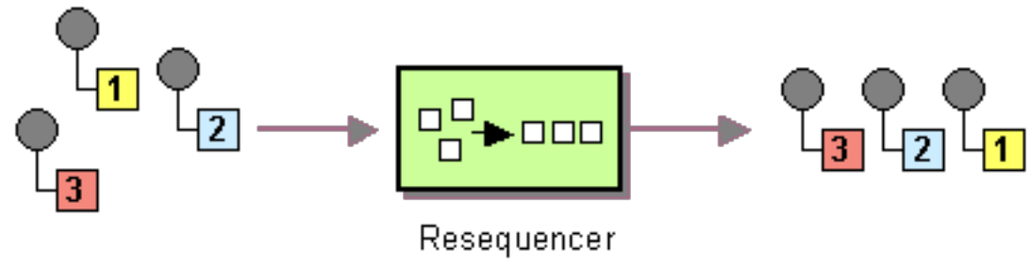
Division / Aggrégation





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



Apache Camel

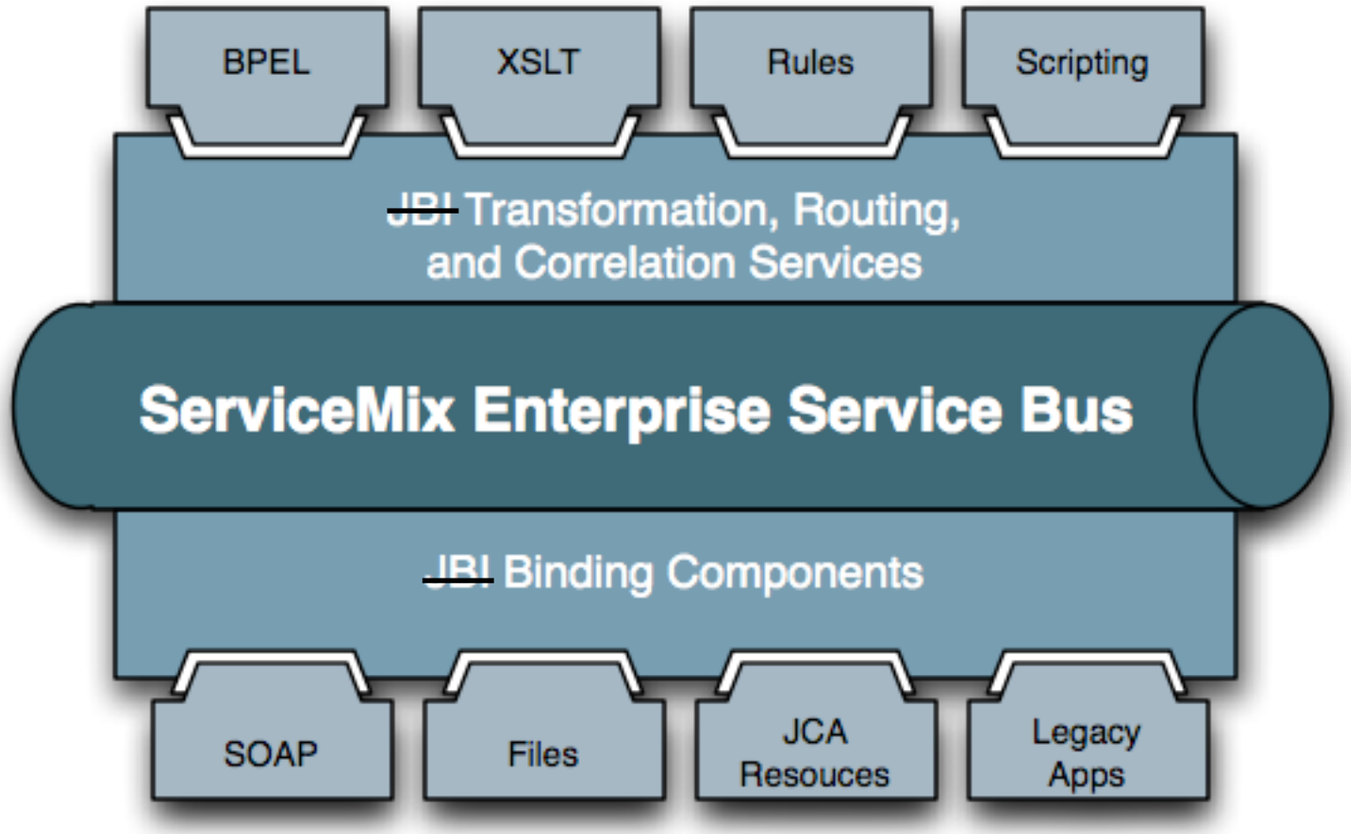
- Implémentation d’un certain nombre d’EIP
  - à travers une API/DSL Java / Scala
- Intégré à de nombreux frameworks/plateformes Java
  - CXF, ActiveMQ, ServiceMix

Bus de services

- Serveur / cluster
  - Hébergeant des services / composants légers (filtres, etc)
  - Fournissant / s'intégrant avec des frameworks de messages
  - Capacités de routage
  - Intègre de nombreux connecteurs

« système de gestion de services »

ServiceMix



- Bus de services
- Basé sur OSGi (Karaf)
- Intègre Camel, ActiveMQ, CXF, Activiti (workflows)

http://servicemix.apache.org

Références

- http://www.enterpriseintegrationpatterns.com/
  - Images  Gregor Hohpe and Bobby Woolf