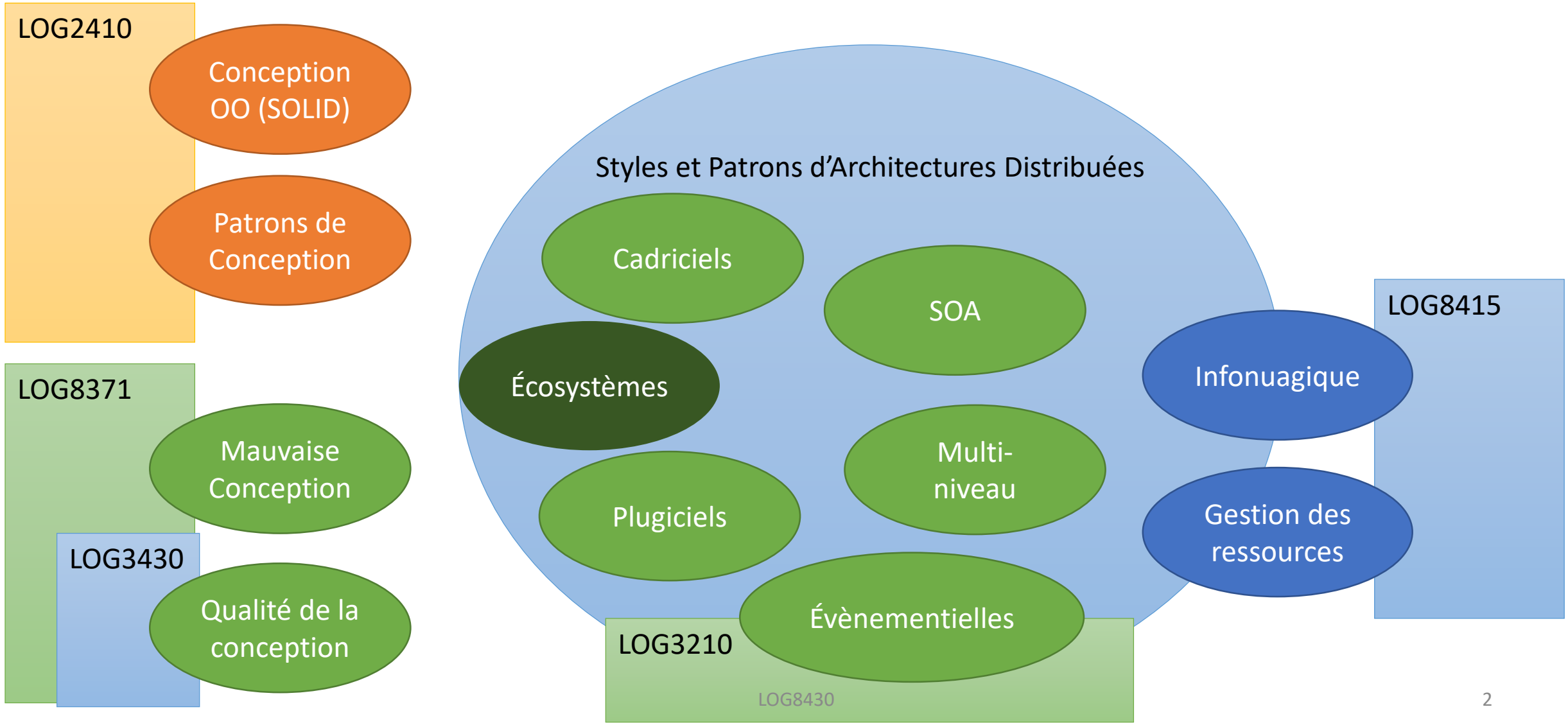
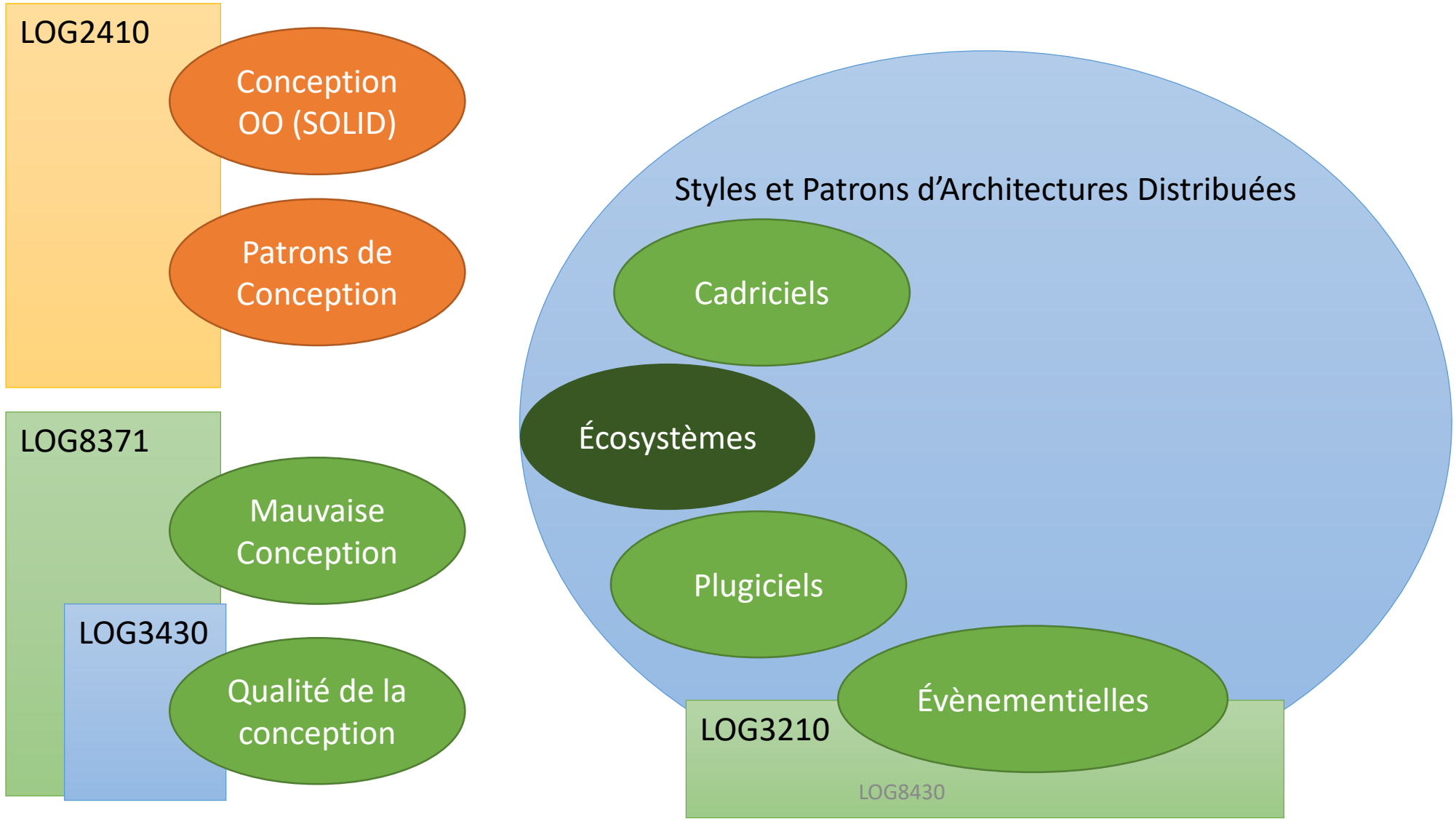
LOG8430: Architectures orientées services

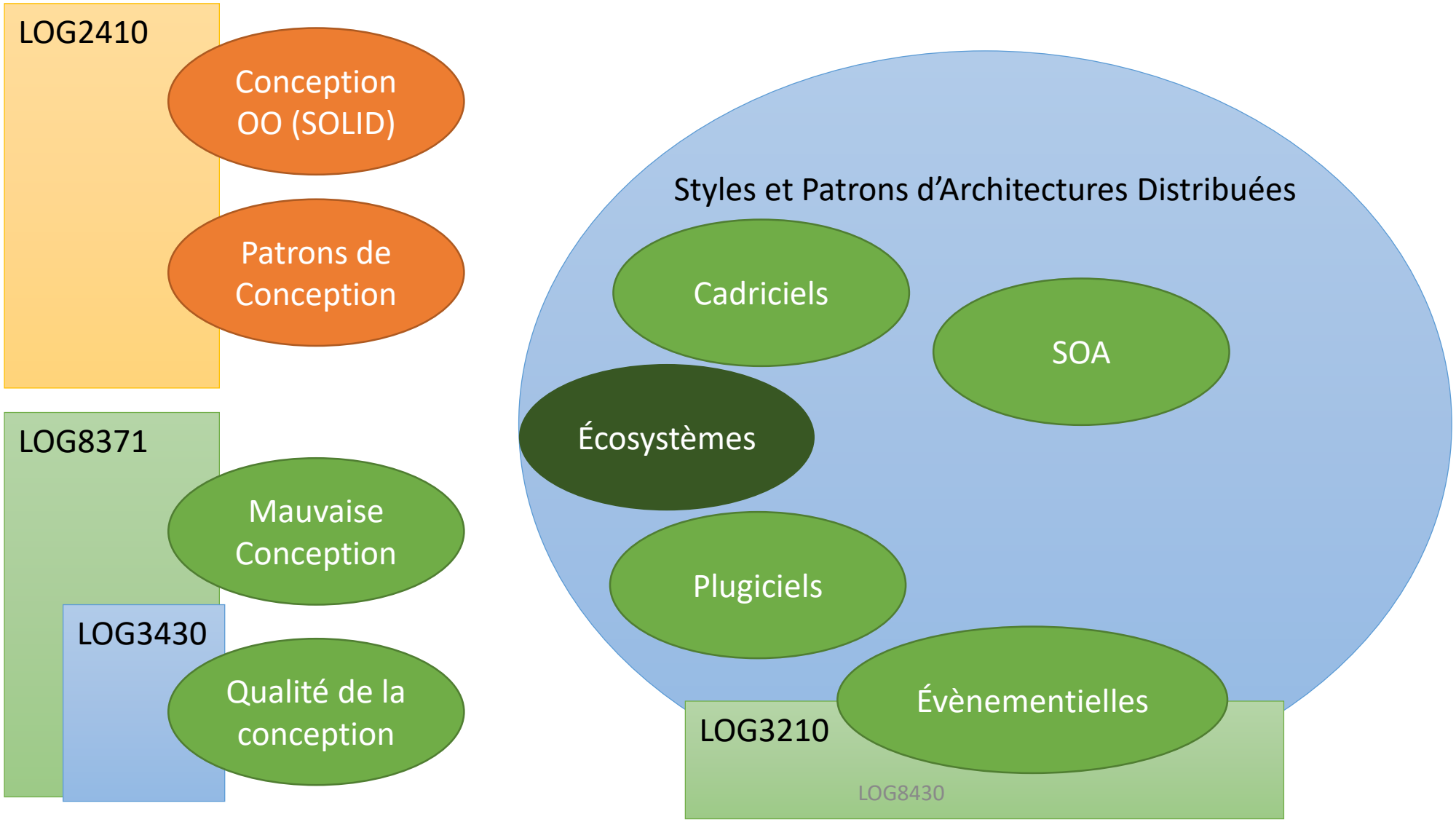
Carte du cours



Précédemment



Aujourd'hui


LOG2410
Conception OO (SOLID)
Patrons de Conception
LOG8371
Mauvaise Conception
LOG3430
Qualité de la conception
Styles et Patrons d'Architectures Distribuées
Cadriciels
SOA
Écosystèmes
Plugiciels
Évènementielles
LOG3210
LOG8430

La naissance des services web

Au début, il y avait trois forces : la modularité, l'interopérabilité et le Web. La modularité disait que l'univers devait être divisé en modules, chacun devant être responsable de quelque chose. L'interopérabilité disait que les modules devaient communiquer entre eux et collaborer pour réaliser leurs tâches. Le Web fournissait un moyen de communication. À la fin, les trois forces se sont rejointent et cela a créé les services, ce qui apporta la paix à l'univers.

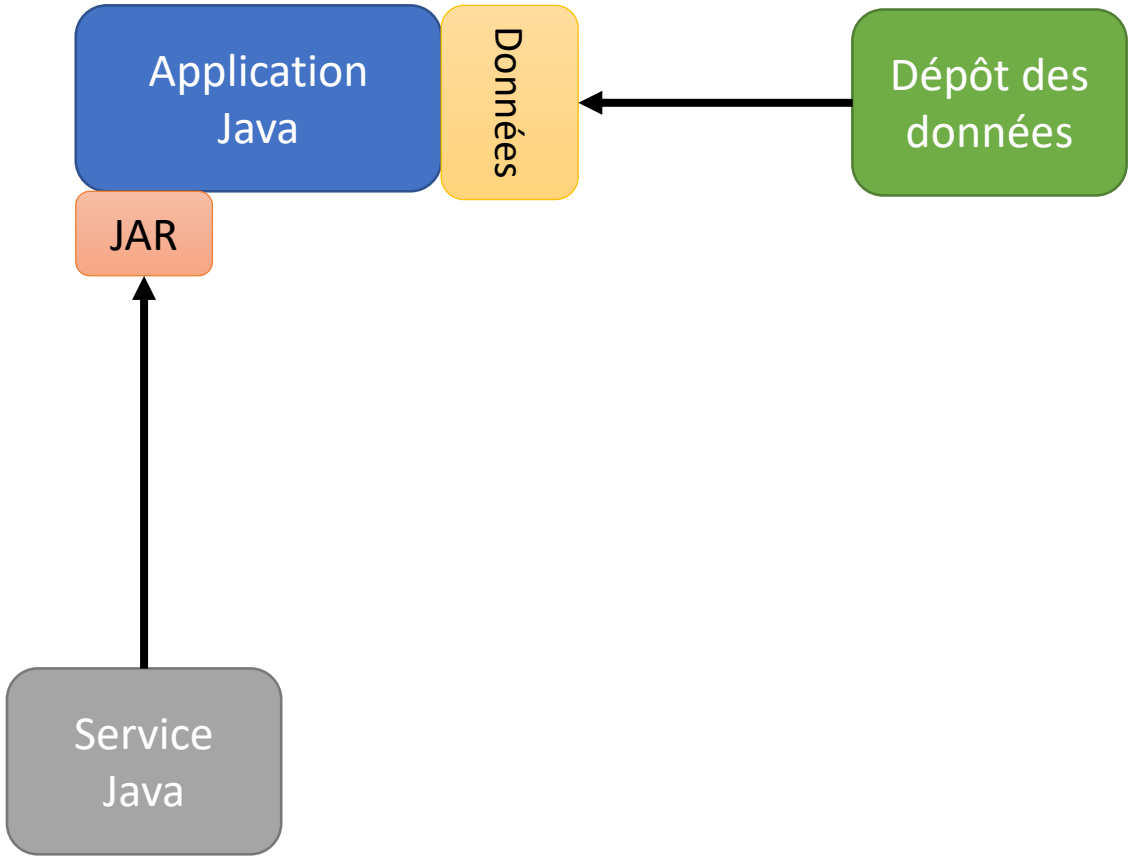
SOA (Services Oriented Architectures)

Motivation : Les affaires

- Un service est compris comme une unité d’affaire.
  - Il accomplit une tâche et il apporte de la valeur.
- Un flux de services définit un processus d’affaires.
- Les services doit être interopérables et partageables.
- Les responsabilités doivent être bien séparées.
- Les services sont conçus afin d’être flexibles et changeables, mais pas nécessairement optimaux.

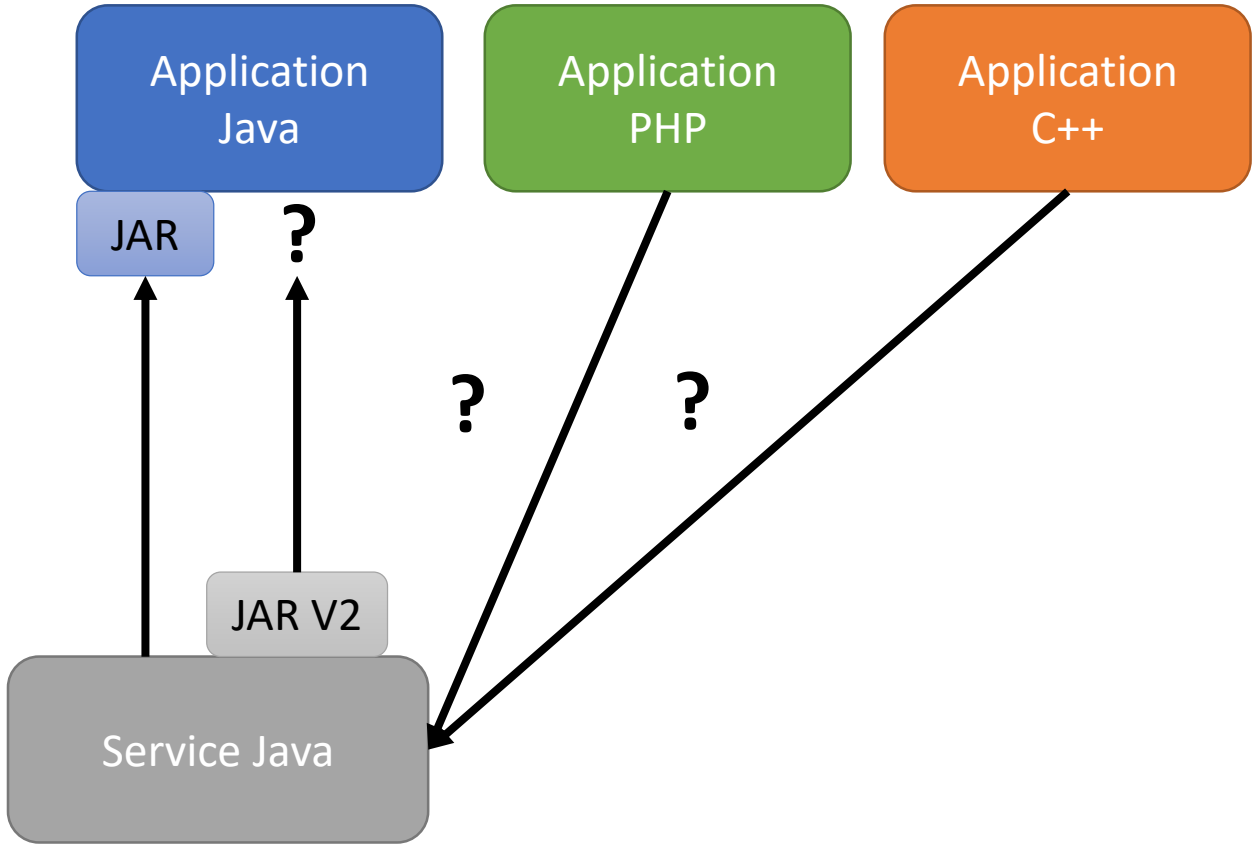
La grâce de la modularité

- « Cette fonctionnalité est déjà implémentée par le fournisseur X ».
- « Ces données sont disponibles par le fournisseur Y ».
- Fonctionnalité réutilisable
  → Minimiser l'effort de développement.
- Faire confiance à l'expertise des autres.
- Du coté du fournisseur, dicter la façon d'utiliser et d'accéder aux données.
- Découpler les composantes.



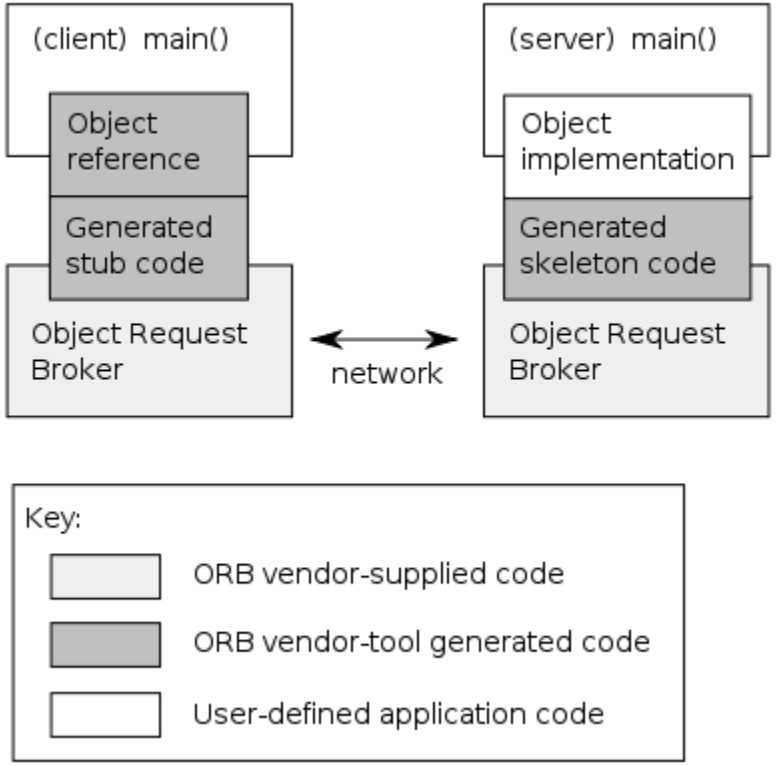
Le malheur de la modularité

- La modularité par des librairies inclut le partage de code.
- Cela implique des dépendances technologiques (OS, langage etc.)
- Il y a aussi des dépendances temporelles.
  - Les clients téléchargent une version et ils doivent mettre à jour la librairie manuellement lorsqu'il y a une nouvelle version.



Les premiers efforts pour l'interopérabilité CORBA

- Une méthode pour partager des objets.
- Le courtier (ORB) achemine les requêtes aux objets indépendamment du langage ou du système d'exploitation.
- Cela permet de partager des objets entre des applications développées dans des langage différents, et de communiquer par réseau.
- Des outils existent pour traduire l'interface d'un objet vers une implémentation locale.



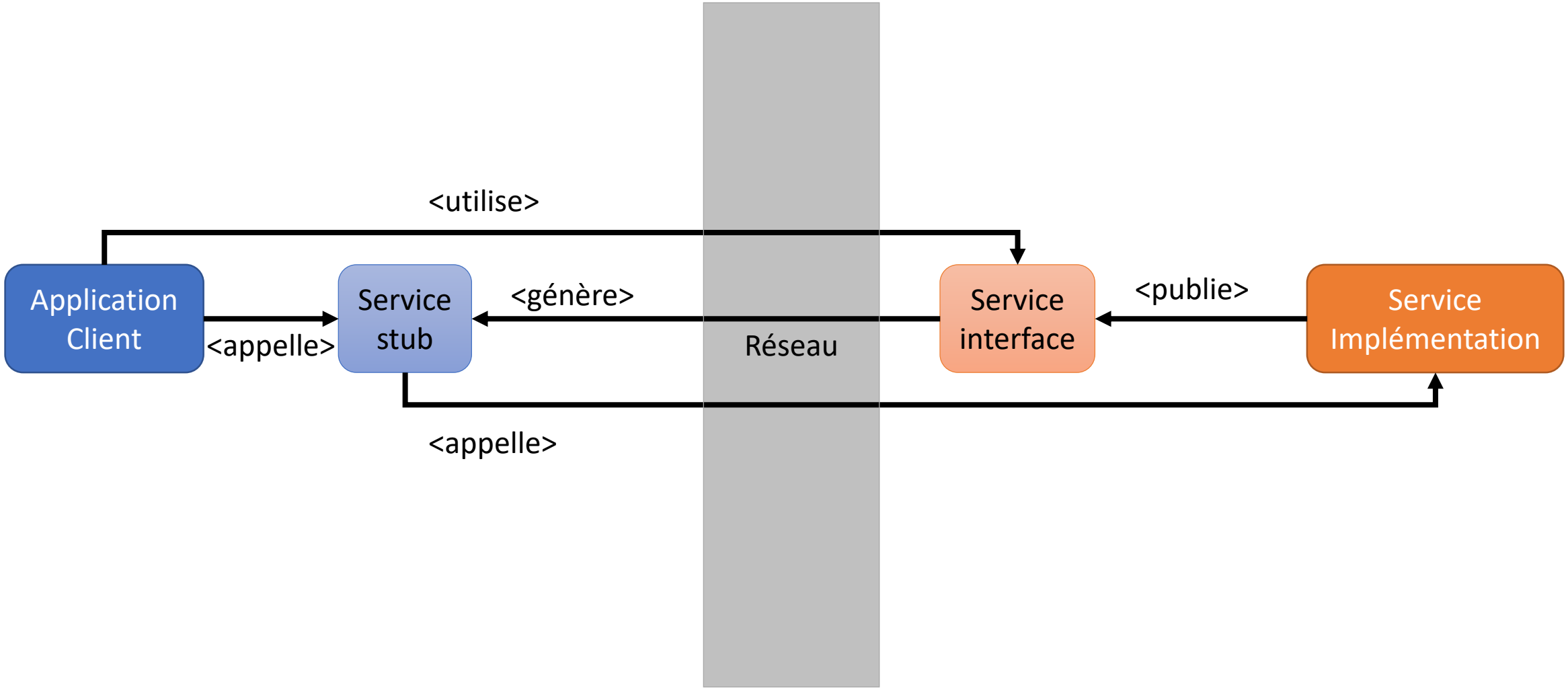
Mais pourquoi on n’utilise pas CORBA?

- Problèmes d’implémentation.
- Problèmes de conception.
  - Conception par comité : beaucoup de conflits et manque de consensus
  - Manque de normes.
- Transparence de la localisation.
  - Les objets sont traités à la même façon indépendamment de leur localisation.
  - Cela crée des problèmes de performance.

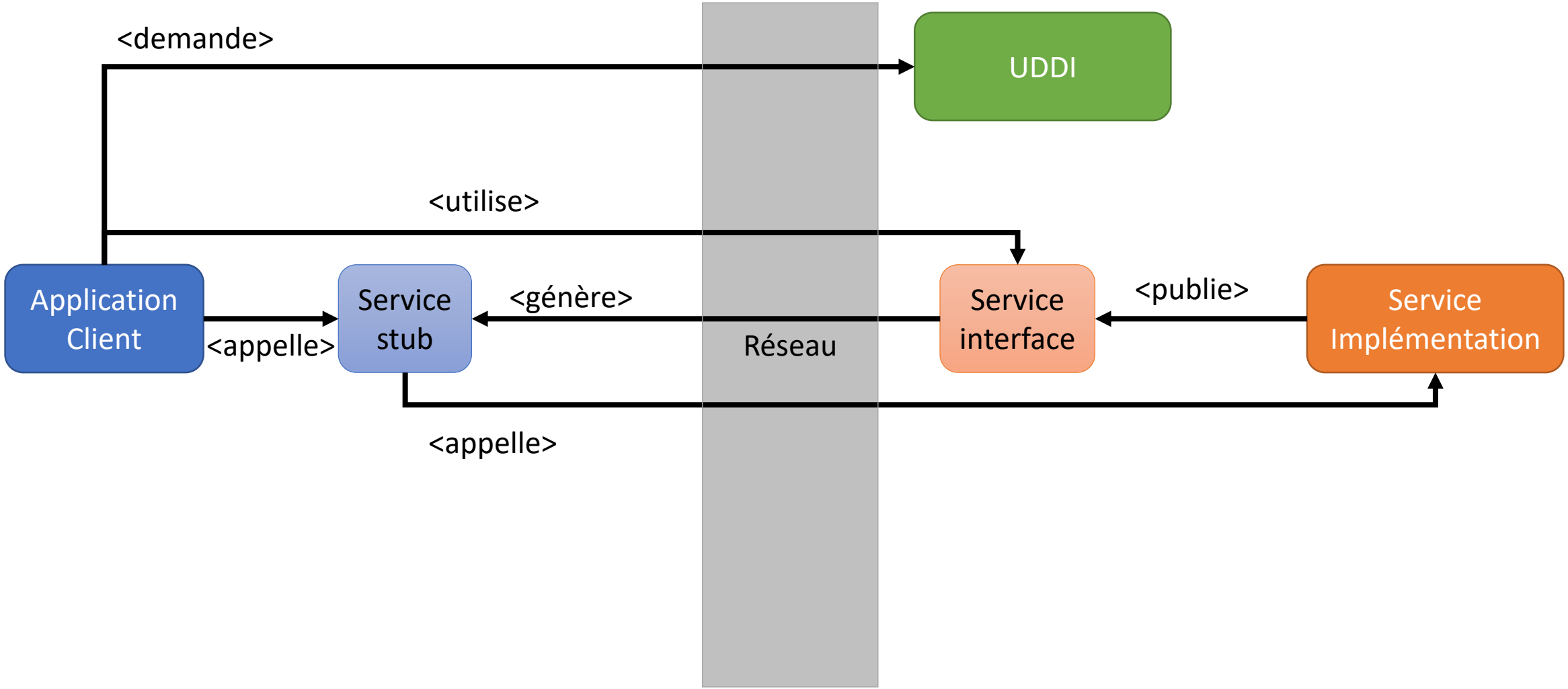
Définition des services web

- Un service est une composante logicielle qui est :
  - Indépendante : elle n’exige aucun autre logiciel pour fonctionner.
  - Atomique : elle produit des résultats complets.
  - Publique : elle expose une expertise/fonctionnalité ou des données de manière explicite.
- Un service publie une interface.
  - C’est une boite noire pour les clients.
- Les services composent des architectures modulaires et distribuées.

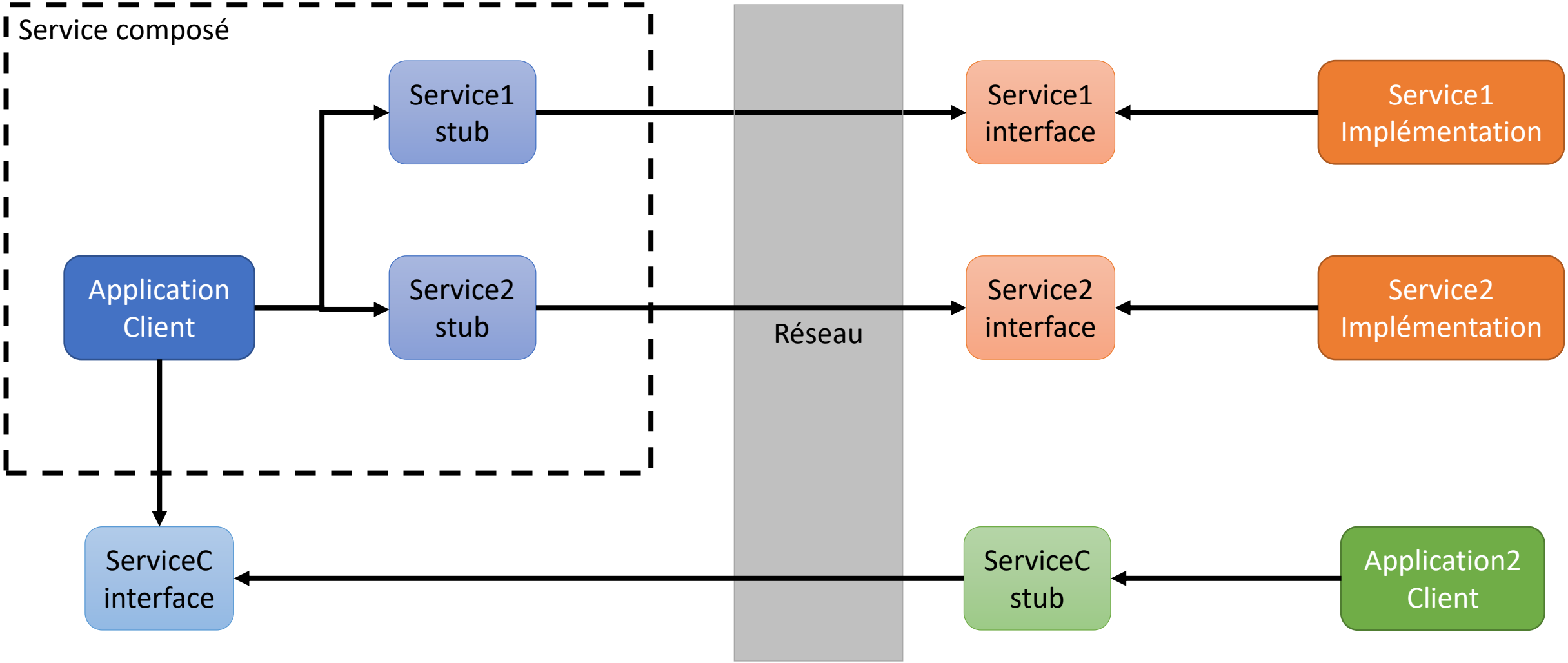
Architecture des systèmes de service



Architecture des systèmes de service

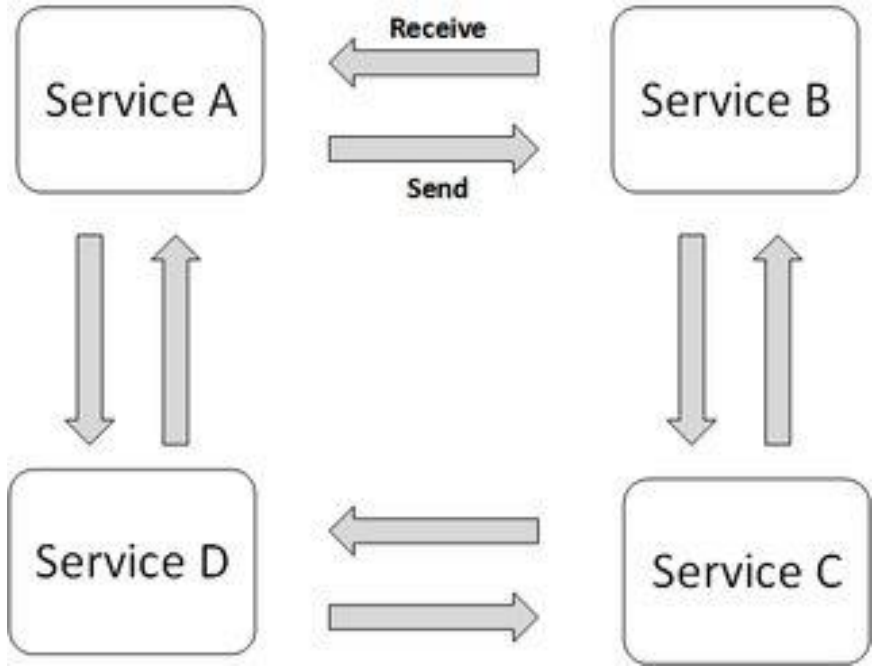


Architecture des systèmes de service

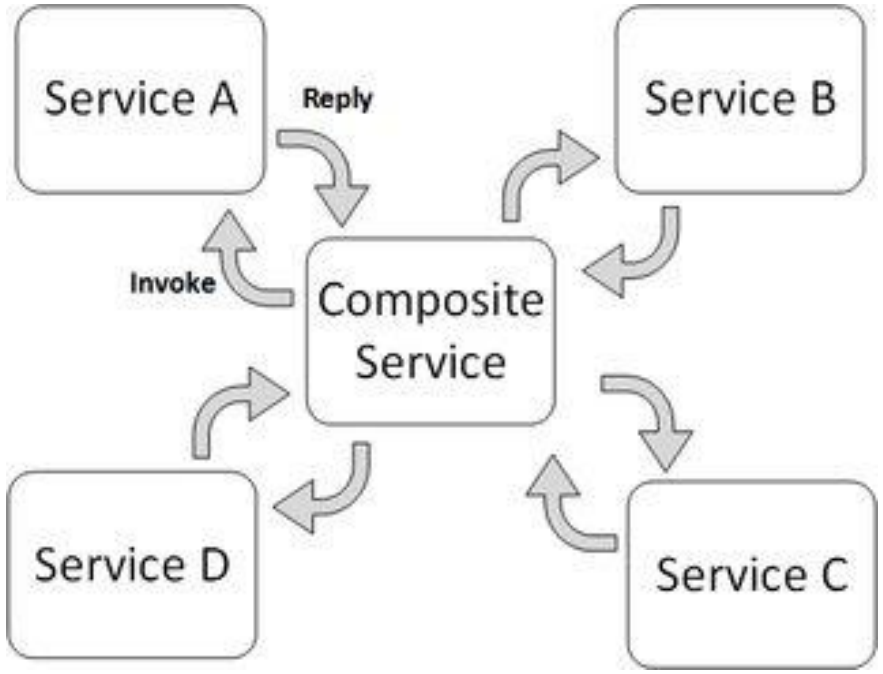


Patrons de composition

Chorégraphie



Orchestration



SOA Composantes

- Entités
  - Fournisseur
  - Client
  - Réseau
- Artefacts
  - Implémentation service
  - Interface service
  - Implémentation client
  - Service stub
  - Protocole de communication
  - Entente de niveau de service (SLA)

SOA : Fournisseur

- **Rôle** : Le fournisseur…fournit le service! Il est responsable pour son implémentation et la publication de son interface.
- **Objectifs** : Le fournisseur expose ses fonctionnalités et ses données propriétaires, mais d'une façon qu'il contrôle explicitement.
- **Responsabilités** : Le fournisseur est responsable du bon fonctionnement du service, de son déploiement, de sa disponibilité et de sa qualité. Il est aussi responsable de la bonne conception de l'interface du service.

SOA : Client

- **Rôle** : Le client utilise le service. Il développe l'application-client qui consomme les données fournies par le service.
- **Objectifs** : Le client utilise le service pour obtenir des données externes ou pour accéder aux fonctionnalités implémentées par d'autres. Le client s'attend au niveau de qualité promis par le fournisseur.
- **Responsabilités** : Le client doit suivre les termes d'utilisation imposés par le fournisseur. Il doit générer un stub dans le langage de l'application locale. Le client est responsable d'adapter l'application en cas d'évolution du service.

SOA : Réseau

- Le réseau est le mécanisme de communication entre le service et les application-clients.
- Parfois, on parle de l'Internet, mais on peut aussi avoir recours à un réseau privé dans une entreprise ou à une infrastructure infonuagique.
- Le fournisseur est responsable d'exposer son service sur un réseau et le client est responsable de maintenir une connexion à ce réseau.
- Les défaillances du réseau sont la responsabilité du fournisseur du service.
  - Cela affecte la SLA.

SOA : Service et Application

- Le service et l'application-client sont des applications normales.
- Leur développement est indépendant.
  - Le langage d'implémentation peut être différent.
- Pour être déployées comme des services, les implémentations doivent avoir des annotations spéciales ou dépendre de librairies pour permettre à des outils spéciaux de générer les interfaces et le code nécessaire au déploiement.
- Pour l'application-client, des librairies sont aussi nécessaires pour accéder aux services sur un réseau.
- Les librairies sont spécifiques au langage d'implémentation.

SOA : Interface du service

- Comme dans le cas de l'interface publique d'une classe ou d'une API, l'interface d'un service spécifie ce qui est réalisable par ce service.
- Elle spécifie les opérations qui peuvent être appelées par un client.
  - Nom, paramètres, type de retour, exceptions etc.
- Elle spécifie les données qui sont disponibles ou qui sont utilisées par les opérations.
  - Nom, type, structure etc.
- L'interface spécifie aussi l'endroit où se trouve le service, sa version, ses dépendances potentielles et d'autres *méta-informations*.
- Contrairement à l'interface d'une classe, l'interface d'un service est parfois un document formel et elle peut être consommée par des outils automatique.
  - Génération du stub.
  - Découverte du service.
  - Identification des différences entre des versions ou des services différents.
- L'interface est spécifiée dans un langage abstrait, p.ex., XML, HTML etc.

SOA : Service stub

- C’est le code qui rend possible l’accès au service.
- Le code est généré automatiquement ou il est disponible en tant que librairie dans le langage de l’application-client.
- Le code peut être complexe et difficile à comprendre, mais il n’est pas nécessaire de le comprendre ou de le modifier.
- Les informations définies dans l’interface du service sont suffisantes pour générer un stub.
- Les outils de génération complètent ces informations avec des configurations pour l’accès au réseau.

SOA : Protocole de communication

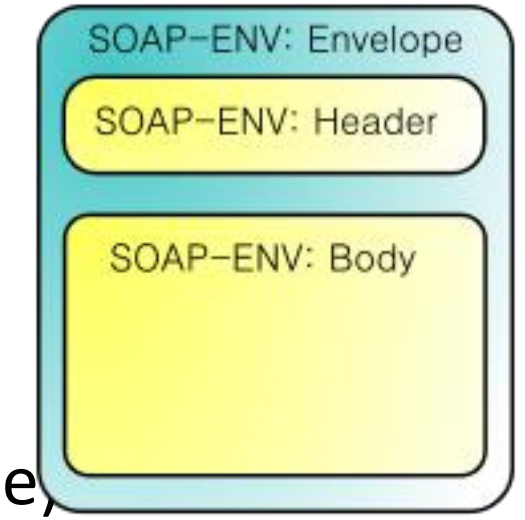
- Le protocole de communication dicte la façon de communiquer entre les modules sur le réseau, les données qui transitent entre les modules, leur format et le type de connexion entre les modules.
- Il y a plusieurs protocoles avec des intentions diverses:
  - HTTP – données (parfois textuelles)
  - SMTP – courriels
  - FTP – fichiers
- Il y a des types de services qui fonctionnent avec plusieurs protocoles (p.ex., SOAP) ou juste un protocole (p.ex., REST)

SOA : SLA

- La SLA est une entente entre le fournisseur et le consommateur du service qui concerne la livraison et la qualité du service.
- Dans le document, on spécifie :
  - Les types de services disponibles
  - La performance désirée du service
  - Les méthodes de monitorage
  - Comment rapporter les problèmes
  - Le temps nécessaire pour répondre et résoudre les problèmes
  - Les répercussions et les pénalités pour le fournisseur si la qualité du service n'est pas telle que prévue.
- Des métriques populaires incluent:
  - Temps de réponse pour une demande
  - Nombres de demandes servies par unité du temps.
  - Taux de disponibilité

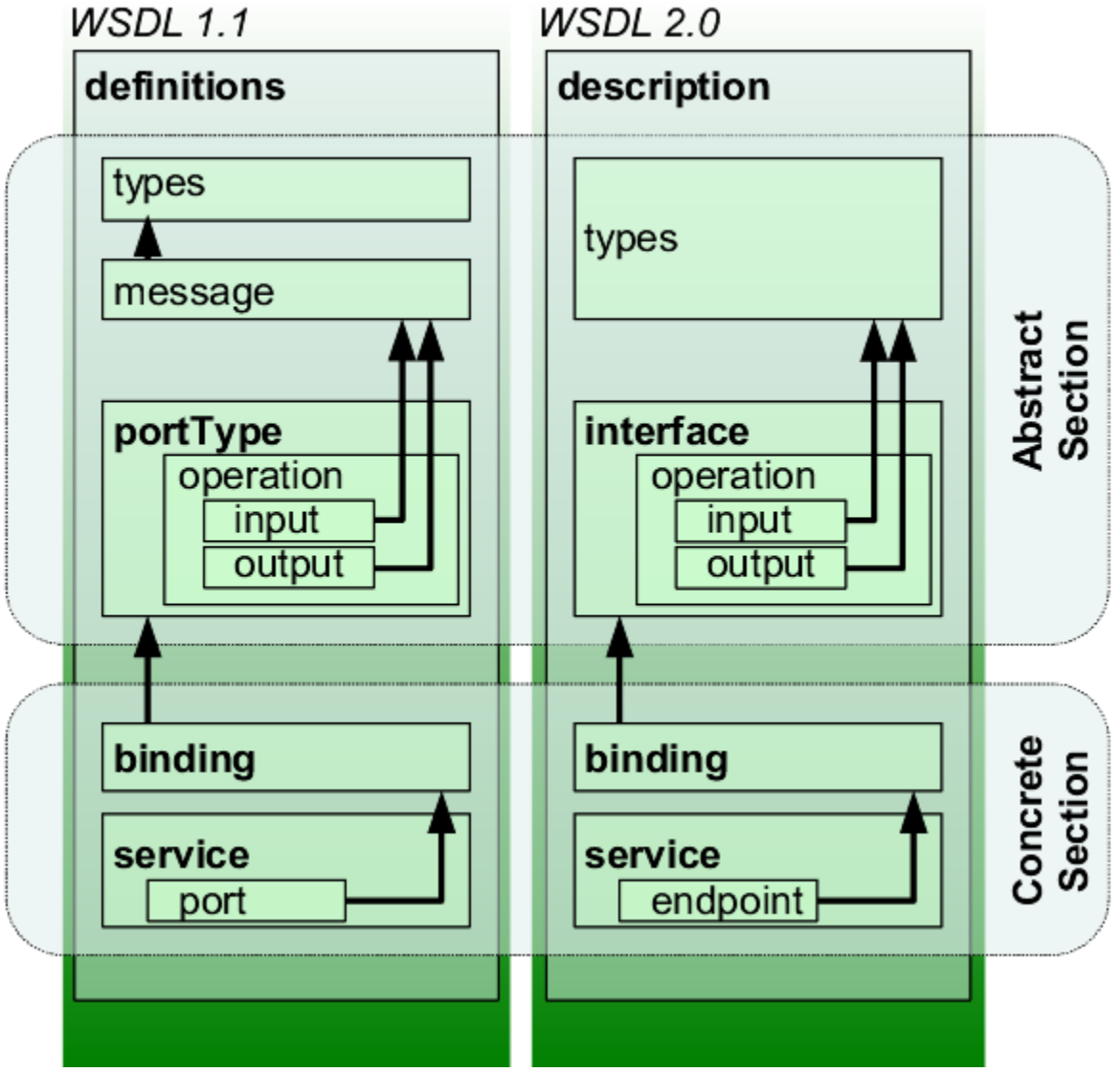
Simple Object Access Protocol (SOAP) et les WS-* services

- SOAP est un protocole de messages.
- Il est indépendant du langage, mais aussi du protocole de communication.
- SOAP utilise XML pour spécifier les données.
- SOAP inclue le message (en XML) et les détails de la communication (comme une demande HTTP, par exemple).
- Les services qui utilise SOAP utilisent les normes WS.
  - Web Service Definition Language (WSDL)
  - WS-Security
  - WS-Policy



WSDL

- Les interfaces de services SOAP sont définies en WSDL.
- WSDL est un langage basé sur le XML.
- Le fichier WSDL spécifie les opérations, leurs types de retour, et les types de leurs paramètres.
- Il spécifie aussi la localisation du service et comment on peut y accéder (la liaison et le protocole de communication).



SOAP : Avantages et Désavantages

- **Avantages**
  - Indépendant du protocole de communication
    - C’est personnalisé.
  - Il est basé sur XML, donc tous les outils du langage sont disponibles (chercher, comparer, etc.).
  - Il est possible de définir plusieurs normes et spécifications à nouveau.
- **Désavantages**
  - SOAP n’est pas efficace.
  - Les interfaces de service et les formats des messages sont longs et complexes.
  - Parcourir les fichiers est couteux.

Representational State Transfer (REST)

- REST est un style architectural pour créer des services web.
- REST est basé essentiellement sur HTTP.
- REST est utilisé pour accéder aux ressources disponibles sur le web.
- Toutes les ressources ont un identificateur unique.
- On peut accéder à une ressource spécifique ou à une collection de ressources par une URL.
  - http://www.myservice.com/books
  - http://www.myservice.com/books/978-3-16-148410-0
- On peut appliquer cinq opérations sur ces ressources : GET, PUT, POST, PATCH, DELETE
- Les interfaces des services REST sont parfois publiées comme des pages HTML, même s'il y a quelques propositions de spécifications (pas très populaires).
- Les requêtes des clients au service sont atomiques, c.-à-d. il y a une réponse (ou un exception) par requête.
  - Aucune information du client est stocké sur le serveur.
  - Authentification? Token!

REST : Avantages et Désavantages

- Avantages
  - Une interface simple (juste 5 méthodes).
  - La performance est la priorité.
  - Modificabilité, fiabilité, évolutivité, portabilité augmentées.
- Désavantages
  - Le manque de normes et de spécifications formelles ajoutent des défis aux tâches périphériques pour la maintenance des systèmes et des services (découverte, recherche, composition, évolution, adaptation).
  - Les fonctionnalités plus complexes ne sont pas toujours disponibles.
  - Il faut une organisation des données.

Remote procedure calls (XML/JSON RPC)

- Si on un appel comme ça : http://www.myservice.com/calculatePi?numberOfDigits=9
- S’agit-il d’un appel à un service REST?

Remote procedure calls (XML/JSON RPC)

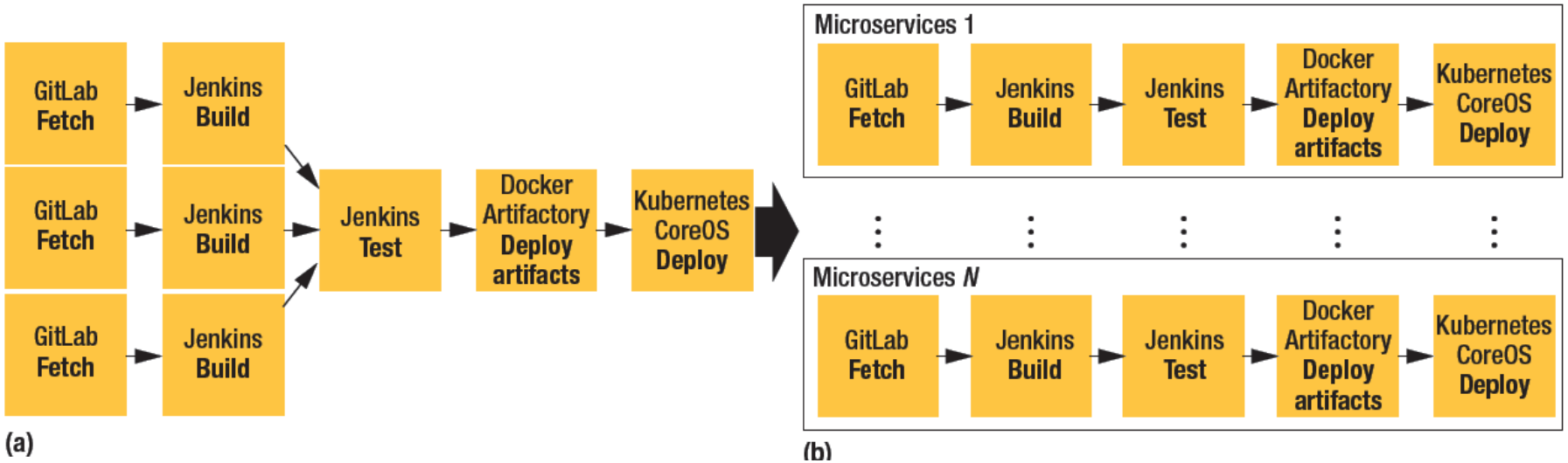
- Si on un appel comme ça : http://www.myservice.com/calculatePi?numberOfDigits=9
- S'agit-il d'un appel à un service REST?
- NON!
- Les services REST fonctionnent sur des ressources et juste avec des méthodes simples.
- Les « RPC » peuvent appeler n'importe quelle méthode (comme sur un objet).
- Elles utilisent des formats standards pour transférer les données (XML ou JSON).
- En fait, SOAP est une évolution et CORBA est une implémentation des RPC.
- Les avantages et désavantages sont presque les mêmes que pour REST, mais avec plusieurs capacités au niveau des méthodes disponibles, et sans imposer une organisation des données.

Microservices

- Une version …plus petite de SOA.
- Chaque service est juste une opération avec une seule responsabilité.
- Les microservices gardent toutes les autres propriétés de SOA : réutilisabilité, portabilité, interopérabilité.
- Ils sont plus faciles à composer et à remplacer.
- Les microservices sont déployés et gérés indépendamment.
- En combinaison avec les conteneurs, les microservices sont la technologie fondamentale pour DevOps.
- Ils promeuvent l'automatisation, l'évolution et la livraison continue.
- Défis
  - Transfert de charge cognitive
    - Le logiciel est plus simple, mais les connexions dans le réseau deviennent beaucoup plus nombreuses et complexes.
    - Nanoservices : quelle est la taille appropriée pour les services?

Migration vers une architecture de microservices

- Motivation
  - La réutilisabilité
  - La gestion des données décentralisée
  - Le déploiement automatique
  - L'évolutivité



Transformations du processus de développement

- L'organisation des équipes suit maintenant l'architecture du système.
- On combine des experts pour chaque microservice.
- On définit une équipe-cœur pour gérer le logiciel au niveau du système.