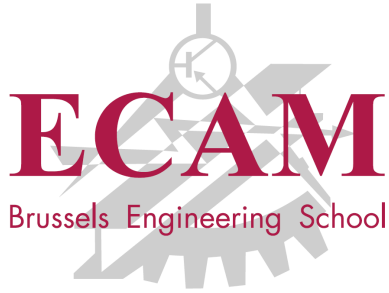


I402A Architecture logicielle

Séance 8

Architecture orientée-services

Sébastien Combéfis

2017–2018

Rappels

- Comprendre les systèmes distribués et leurs architectures
  - Définition et buts des systèmes distribués
  - Communication entre composants à l'aide de middleware
- Trois exemples d'architectures distribuées les plus répandues
  - Architecture client-serveur avec client léger ou lourd
  - Séparation physique avec une architecture multi-tier
  - Objet partagé et appel à distance avec architecture broker

Objectifs

- Survol de l'évolution des architectures jusqu'à aujourd'hui

  Des applications monolithiques aux (micro-)services

- Présentation des architectures orientée-services
  - Caractéristiques de l'orientation services
  - Composants et intervenants dans les services webs

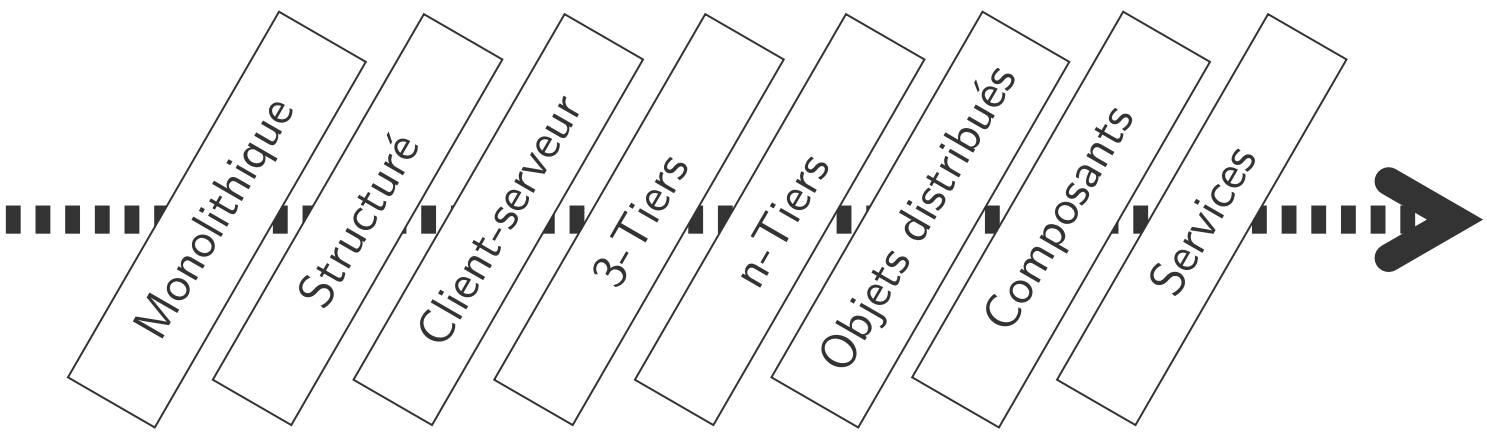
- Architecture de type REST

  Principes généraux et quick tips pour bien les utiliser

Évolution des architectures

Évolution des architectures

- Évolution des architectures avec changements business
  - Divisions business isolées, d'organisation verticale
  - Structures focusées sur les process, d'organisation horizontale
  - Écosystème business avec des composantisé et distribué



Du monolithique à sa distribution

- Organisation verticales des entreprises avec business isolés

  Ensemble de logiciels par unité, très monolithiques

- Ajout d'une structure pour organisation horizontale

  Modules pour les étapes d'un process, chacun monolithique

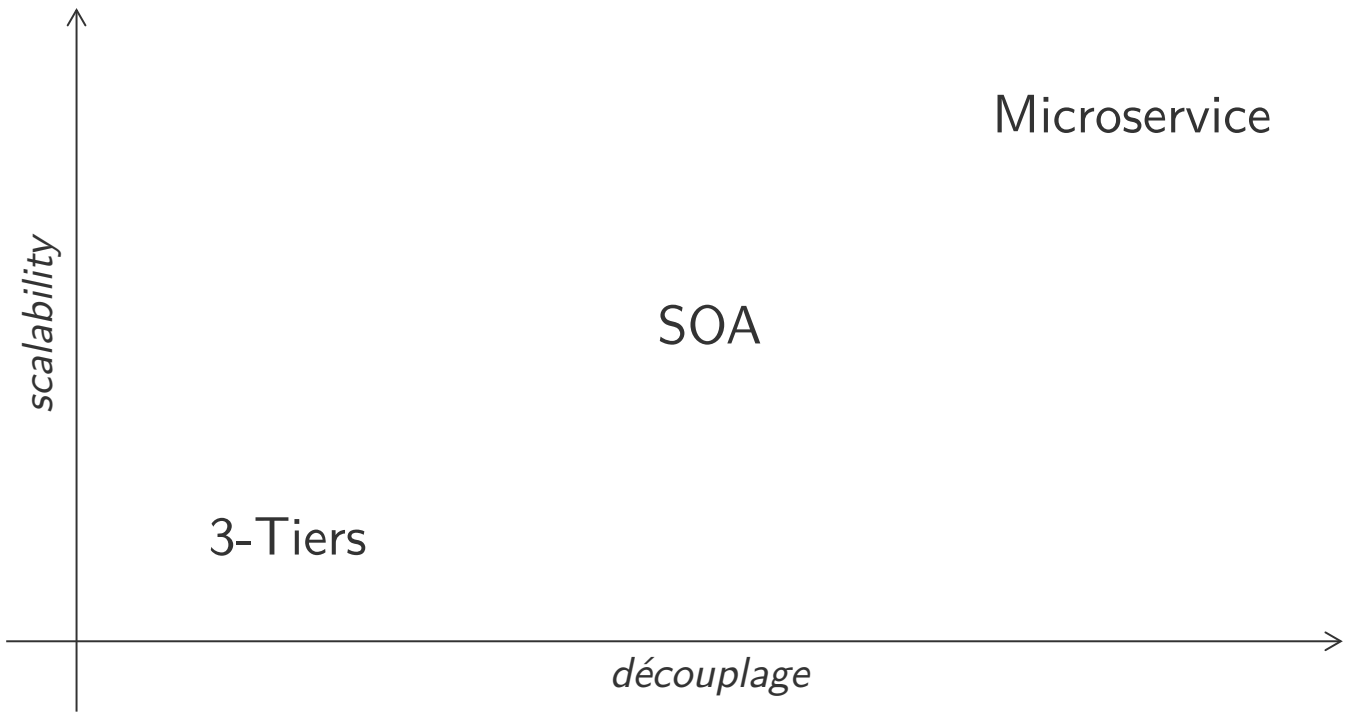
- Apparition d'architecture client-serveur avec les réseaux

  Plusieurs clients pour un même serveur d'applications

Du 3-Tiers aux services (1)

- Augmentation de la scalability et du découplage

  Shift de paradigme avec la croissance des services sur Internet



Du 3-Tiers aux services (2)

- Séparation de trois principales parties avec 3-Tiers

  Frontend, backend et sources de données

- Confinement logique business en composants avec SOA

  Intégration de composants distribués faiblement couplés

- Chaque (micro-)service est une unité atomique de travail

  Plusieurs (micro-)services pour un en SOA

Orienté-services

Architecture orientée-services (1)

- Un service est un composant d'une fonctionnalité business

  Bien défini, self-contained, indépendant, publié et disponible

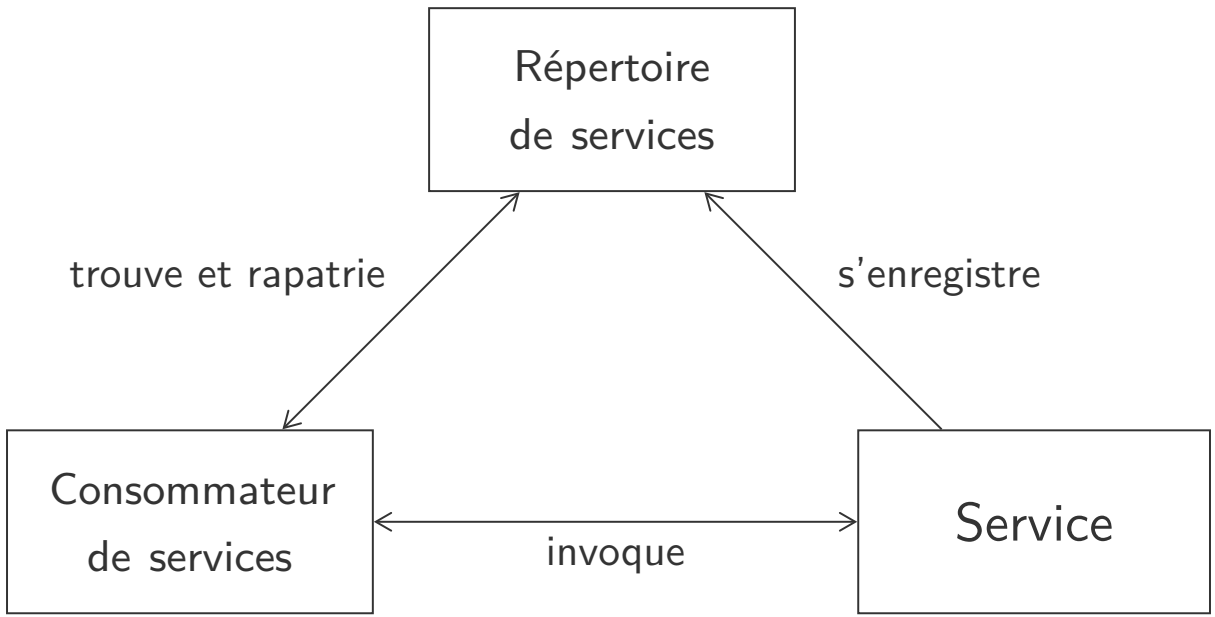
- Connection entre services par protocole d'échange de messages

  Distribue les requêtes et réponses entre les services

- Applications centrées autour de services business-driven IT

  Services software et consommateurs de ces services

Architecture orientée-services (2)



Caractéristiques

- Déploiement distribué des données et services (logique)

  Découplage maximum, découvrable, structuré, coarse-grained, basé sur des standards, unités fonctionnelles sans état

- Composabilité et réutilisabilité des composants

  Assemblage de processus à partir de services existants

- Interopérabilité

  Partager et utiliser services partagés, peu importe technologie

Orientation service

- Paradigme architectural avec quatre principes
  - Les frontières sont explicites
  - Les services sont autonomes
  - Partage de schéma et contrat entre services, pas de classes
  - Compatibilité de services basée sur des politiques
- Architecture orientée service (SOA)

  Doit satisfaire les quatre principes ci-desssus

Service web

- Web service est implémentation plus courante de SOA

  Offre de services sur le web, à travers internet

- Plusieurs technologies existantes utilisables
  - Protocole HTTP ou HTTPS pour réaliser les communications
  - Langage XML ou JSON pour construire les messages
  - WSDL pour décrire les services offerts, entêtes de méthodes
  - UDDI pour enregistrer et chercher des services web
  - Message SOAP pour invoquer un service web

UDDI

- Universal Description, Discovery, and Integration (UDDI)

  Enregistrement et recherche de descriptions de services

- Stockage de quatre informations primaires
  - **businessEntity** décrit le fournisseur de service
  - **businessService** détails non techniques sur le service
  - **bindingTemplate** information technique d'accès au service
  - **tModel** est un modèle technique

WSDL

- Web Services Description Language (WSDL)

  Description des signatures précises des services invoquables

- Document XML qui répond à trois questions sur le service
  - **À propos de quoi** est le service ?
  - **Où** se trouve le service ?
  - **Comment** le service peut-il être invoqué ?

SOAP

- Simple Object Access Protocol (SOAP)

  Protocole d'invocation d'un service web

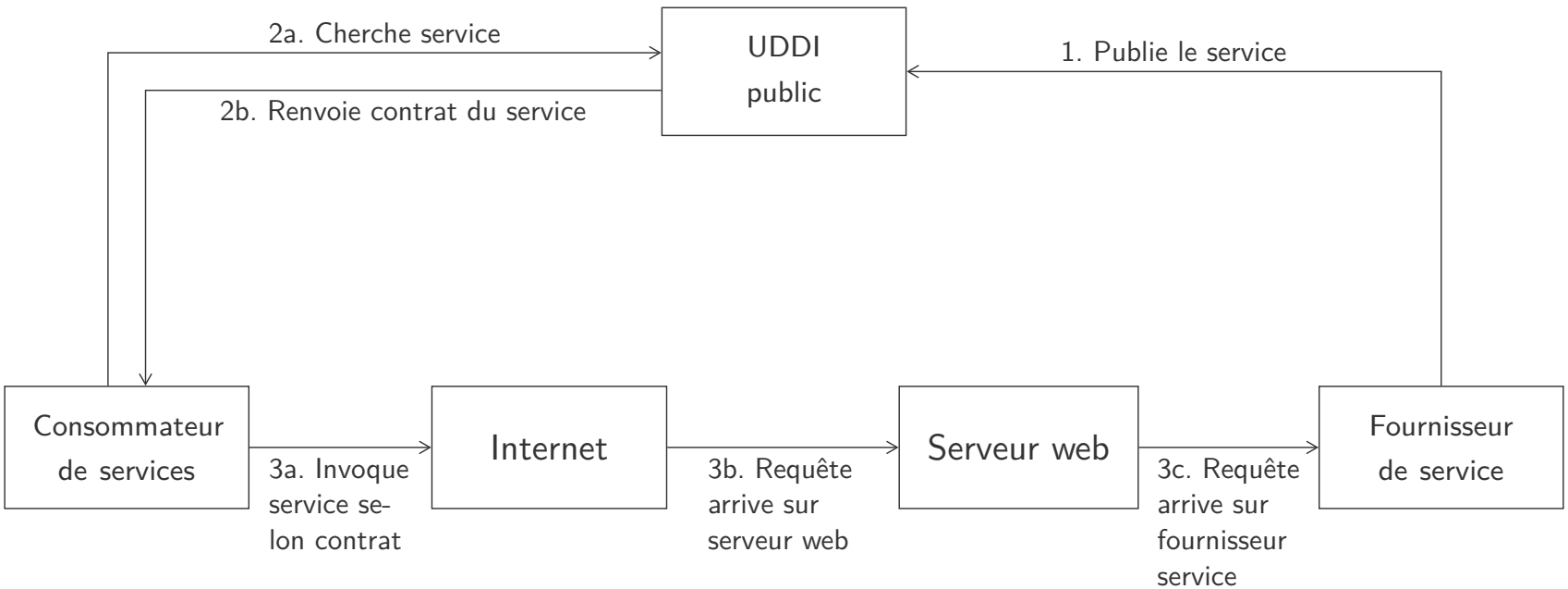
- Indépendant du réseau, transport, langage de programmation

  Messages SOAP encodés en XML, invocation et réponse

- Pas d'appels par référence, que par valeurs

  Aucune référence stateful à des objets distants

Appel d'un service web



Avantage

- Composants faiblement couplés entre eux

  Définition très autonome et indépendante des services, interface

- Transparence par rapport à la localisation des services

  Identification des machines et des services sur ces dernières

- Indépendance du protocole de communication

  Liberté dans le choix du protocole, n'influence pas le service

Inconvénient

- Pas applicable pour une application avec une GUI

  Nécessiterait beaucoup d'échanges de données potentiels

- Nécessité de haute disponibilité des serveurs impliqués

  Tolérance aux pannes et sécurité plus complexes

- Gros investissement de mise en place de l'infrastructure

  Pas à déployer pour une application standalone


REST

Architecture REST

- Développement de services web RESTful

  REST est une collection de principes, et pas de standards

- Style d'architecture REST définie à partir de six contraintes
  - Interface uniforme *(uniform interface)*
  - Pas d'état *(stateless)*
  - Possibilité de cache *(cacheable)*
  - Client-serveur *(client-server)*
  - Système en couches *(layered system)*
  - Code à la demande *(code on demand)*

Interface uniforme

- Définition d'une interface uniforme entre client et serveur

  Simplifie et découple l'architecture, évolution indépendante

- Quatre principes clés à suivre
  - Ressources individuelles identifiées par un URIs

    Représentation des ressources en HTML, XML, JSON...
  - Manipulation d'une ressource directement sur sa représentation
  - Message contient tout ce qu'il faut pour l'interpréter
  - Hypermedia as the Engine of Application State
    - État client : body, query string, request header, requested URI
    - Réponse service : body, response code et header

Pas d'état

- REpresentational State Transfer (REST) sans état

  La requête contient l'état nécessaire à son traitement

- Deux éléments impliqués lors de l'appel d'un service
  - Ressource sur laquelle opérer identifiée par l'URI
  - État ou changement de l'état contenu dans la body response

- Meilleure mise à l'échelle d'une application

  Pas de session, de maintien d'état, load balancer plus simples

État et ressource

- Serveur se base sur application state pour satisfaire requête
  - Données nécessaire pour la session ou requête en cours
  - Varie par client, et par requête
- Resource state est une représentation d'une ressource
  - Données stockées dans la base de données
  - Constant pour tous les clients
- L'application state ne doit pas s'étendre sur plusieurs requêtes

  Exécution indépendante de requêtes, sans dépendre de l'ordre

Cache

- Réponses du service déclarent la possibilité de cache

  Une ressource peut-elle être cachée et combien de temps

- Empêcher un client d'utiliser des données périmées

- Amélioration des performances par caching

  - Élimination d'interactions client-serveur
  - Amélioration de la mise à l'échelle, évolutivité, performance

Client-serveur et couches

- Séparation nette entre les clients et serveurs

  Séparation des préoccupation, par exemple serveur gère données

- Meilleure portabilité des clients et serveurs
  - Stockage des données interne aux serveurs, clients portables
  - Serveur plus scalable car interface et état user côté client
- Possibilité d'avoir des serveurs intermédiaires
  - Facilitation de load balancer, caches partagées...
  - Gestion d'authentification et politique d'accès

Code à la demande

- Extension ou personnalisation des fonctionnalités d'un client

  Transfert d'une partie logique vers le client

- Envoi par le serveur de code JavaScript, applet Java...

- Seule contrainte optionnelle d'une architecture REST

  Un viol d'une des 5 premières fait perdre le titre RESTful

REST Quick Tips (1)

- Utilisation des verbes HTTP pour clarifier les requêtes
  - GET, POST, PUT, DELETE
  - Par exemple, GET ne modifie pas les données d'une ressource
- Choisir des bons noms pour identifier les ressources
  - Query string plutôt utiliser pour filter qu'identifier
  - Possibilité d'organisation hiérarchique des ressources
  - Utilisation de nom, et pas verbe, pour identifier les ressources
  - /posts/23 au lieu de /api ?type=posts&id=23

REST Quick Tips (2)

- Préférer JSON par défaut, et offrir XML en plus si pas cher

  Définition du format JSON/XML fait partie du contrat

- Commencer avec des ressources fine-grained
  - Proche du domaine et de l'architecture base de données
  - Puis agrégation avec service sur plusieurs ressources
  - Commencer par fournir le CRUD
- Offrir de la connectivité grâce à des liens hypermédias
  - Liens vers d'autres ressources dans les réponses
  - Parfois, ajout d'une self-référence, liens de pagination

Propriété

- Idempotence d'une opération (appel de service)
  - Même appels produisent toujours même résultat
  - Même résultat sur serveur, mais réponse peut être différente
- Sécurité d'un appel de service
  - Appel de méthode ne produit pas d'effets de bord sur le serveur
  - Un appel sûr sera idempotent, par définition

Verbe HTTP (1)

- Lecture d'une représentation d'une ressource avec GET
  - Représentation XML ou JSON et response code de 200
  - Erreur avec typiquement 400 ou 404
  - Opération idempotente et sure, appelable sans risques
- Mise à jour d'une ressource avec PUT
  - Envoi de la nouvelle représentation d'une ressource
  - Parfois utilisé pour créer une nouvelle ressource
  - Modification réussie avec 200 ou 204, création réussie avec 201
  - Opération idempotente, mais pas sure

Verbe HTTP (2)

- Création de nouvelles ressources avec POST
  - Post vers le parent crée une ressource fille associée
  - Création réussie avec 201
  - URI de la nouvelle ressource transmise en header Location
  - Pas idempotent, ni sûr
- Suppression d'une ressource avec DELETE
  - Suppression réussie avec 200 ou 204

Exemple de ressources (1)

- Créer un nouveau client

  POST http://www.example.com/customers

- Récupérer le client 1234

  GET http://www.example.com/customers/1234

- Créer une commande pour le client 1234

  POST http://www.example.com/customers/1234/orders

- Récupérer la liste des commandes du client 1234

  GET http://www.example.com/customers/1234/orders

Exemple de ressources (2)

- Récupérer la commande numéro 5678

  GET http://www.example.com/orders/5678

- Ajouter un élément à la commande 5678

  POST http://www.example.com/orders/5678/lineitems

- Récupérer le premier élément de la commande 5678

  GET http://www.example.com/orders/5678/lineitems/1

Bonne pratique de nommage

- Par rapport aux noms des ressources
  - Ne pas utiliser le query string pour définir l'opération
  - GET http://api.example.com/s?op=updcust&id=1234&fmt=json
  - GET http://api.example.com/update_customer/1234
  - GET http://api.example.com/customers/1234/update
  - PUT http://api.example.com/customers/1234/update
- Par rapport à la pluralisation
  - Toujours utiliser le pluriel pour les noms de ressources
  - Singulier réservé aux ressources singleton (pas collections)

Wrapped response

- Code de retour HTTP pas toujours renvoyé au développeur

  En JavaScript par exemple, pas directement accessible

- Emballage des réponses dans une structure à quatre élément
  - **code** statut HTTP de la réponse
  - **status** de la réponse (success, fail ou error)
  - **message** en cas d'échec ou d'erreur
  - **data** avec les données du corps

Filtre

- Query-string utilisé pour passer des paramètres de filtrage
  - Limiter le nombre de résultats : ?offset=0&limit=25
  - Pagination : ?offset=25&limit=25
  - Filtrage des données : ?filter="name::blah|sex::female"
  - Tri des données : ?sort=lastname|-age
- Possibilité de combiner plusieurs paramètres

  Par exemple pour trier des résultats après filtre

Crédits

- https://www.flickr.com/photos/deshaunicus/8443225552
- https://www.flickr.com/photos/jeremybrooks/4164721895
- https://www.flickr.com/photos/aftab/6278574520