Support de cours

SOA

SERVICE ORIENTED ARCHITECTURE

Objectifs du cours

- Comprendre le concept service et les principes de l'architecture SOA
- Comprendre l'intérêt de l'architecture SOA
- Comprendre le concept service Web et apprendre à utiliser ou interpréter les standards des services Web
- Maîtriser le développement de services Web par l'utilisation de l'API JAX-WS

Plan

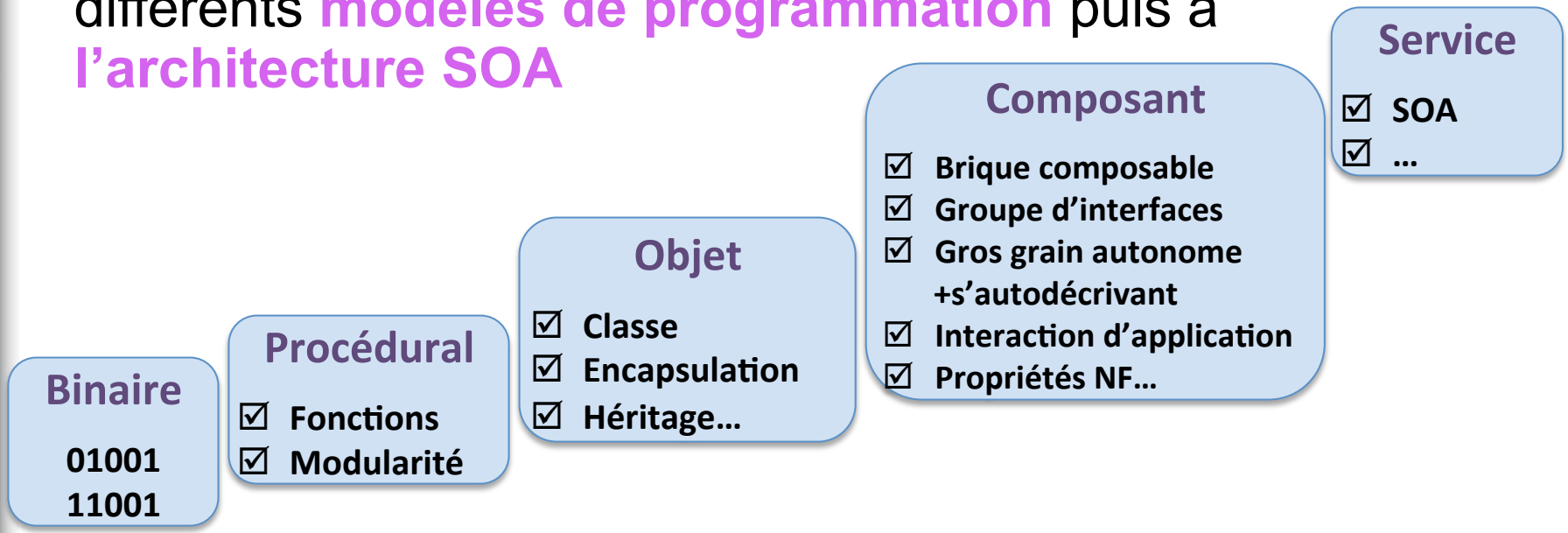
- Le concept Service
- L'architecture SOA
- Le concept Service Web
- Les standards des services Web
- L'API JAX-Web

Ch1. Le concept Service

- Evolution des paradigmes de développement
- Qu'est ce qu'un service?
- Orchestration des services
- Types de services
- Propriétés du service

Evolution des paradigmes de développement

- La conception d'un programme informatique s'effectue conformément à un paradigme de développement (PD)
- Un PD définit un concept pour représenter le monde et des techniques pour traiter ce concept
- Différents PD ont vu le jour et ont évolué du binaire, à différents **modèles de programmation** puis à **l'architecture SOA**



Concept Service

- **Composant logiciel** qui exécute une action pour le compte d'un client
- Il traduit le **niveau logique** d'accès aux traitements, plutôt que le niveau physique d'implémentation (EJB, Servlet…)

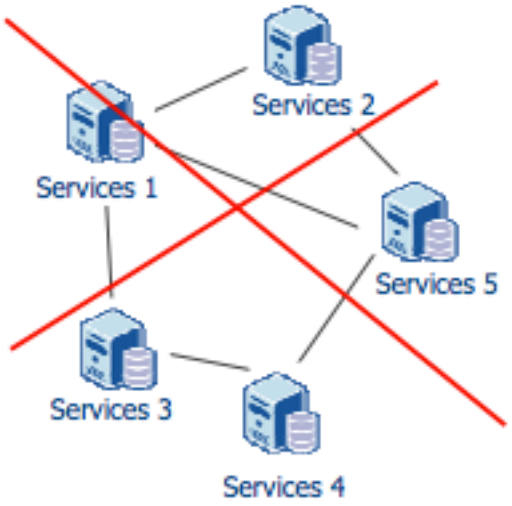
Définition du Service

- **Composant logiciel** :

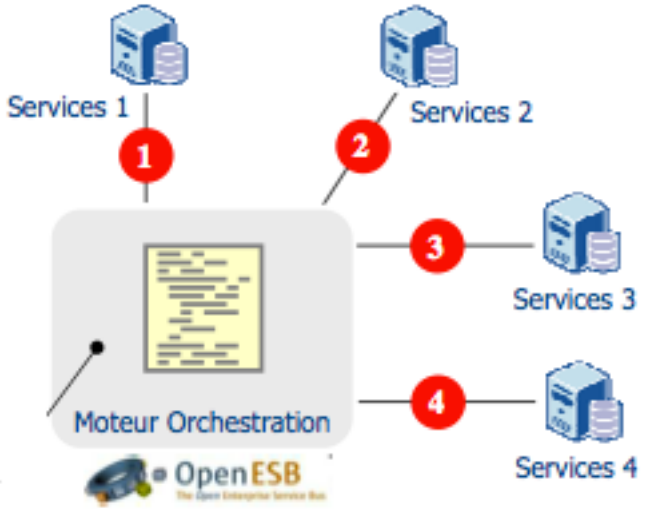
  - **Mutualisé** (partagé puisqu'il est réutilisable et interopérable)
  - **Référencé** dans un annuaire (où il est identifié)
  - **Normalisé** (toutes ses fonctions sont appelés de la même façon via des paramètres, conformément à un **contrat**)
  - Décrit par une **interface d'appel** (par un langage indépendant des technologies)
  - Communicant avec le client par le biais de **messages (E/S)**
  - **Neutre** (son utilisation est indépendante de son implémentation ou évolution tant que le contrat est respecté)
    - **Couplage faible :** interface isole le client du service
  - **Déployé** (physiquement) sur un **serveur**

Orchestration des services

- Les services peuvent être composés (agrégés) dans le but de réaliser un processus donné
- L'orchestration leur permet de **communiquer sans avoir à se connaître** pour préserver leur couplage lâche (leur indépendance)
- **Un moteur d'orchestration se charge d'appeler les services** selon l'enchaînement désiré



Couplage fort



Couplage lâche

Types de services

- **Applicatif**
- **Fonctionnel**
- **Entité (CRUD) :** Create, Read, Update and Delete
- **Transverse (Infrastructure)**
- **Host**

Service Applicatif

- Il traduit la **logique applicative d'une application**, exprimée par les uses cases ou les processus métier
  - C'est un **service de la couche applicative**, qui n'est en général utilisé que dans le contexte de l'application où il a été créé
- Il peut être modélisé par :
  - **UML** : Uses cases et Diagramme d'activité
  - **MERISE** : Modèle Organisationnel de Traitements
  - **BPMN** (Business Process Modeling Notation)
- Ses opérations peuvent être déclenchées selon que des **préconditions** sont vérifiées ou non
- Ses résultats peuvent être émises selon que des **postconditions** sont vérifiées ou non
- Il est **context-aware** (son comportement s'adapte aux besoins des clients et au contexte d'exécution)

Service CRUD, Transverse et Host

- **CRUD : service élémentaire** permettant de créer, rechercher, lire, màj ou exporter vers un format (pdf, excel…), un objet métier

- **Transverse (Infrastructure) :** exécute un **traitement métier spécifique** (ex : services de log, gestion du Contexte Utilisateur…)

- **Host :** permet aux applications distribuées d'utiliser une application **Mainframe** du Host de l'entreprise
  - Il peut être encapsulé dans et rendu accessible par des logiciels d'infrastructure destinés à la Gestion ou Contrôle d'information (ex : IMS, CICS…)

Service fonctionnel

- C'est un **service de la couche Services, réutilisable** dans des contextes variables
- Il **exécute un traitement métier** (fonction), et peut être invoqué par différent services applicatifs
- Il **invoque des services CRUD** et/ou **Transverses** pour pouvoir manipuler des objets métiers
- Il peut aussi invoquer des services plus élémentaires ou externes (partenaires) pour assurer **l'orchestration** (gérer leur enchaînement de manière à réaliser un processus qui apporte une valeur ajoutée)
- Il peut aussi servir à la gestion de la sécurité, des règles métiers, etc.

Propriétés des services

- **Réutilisables** et possèdent des **contrats standardisés**
- Communiquent par **messages** à travers des **interfaces adressables**
- **Abstraits et prédictibles**
- **Modulaires** et de **large granularité**
- **Autonomes** et **sans état (stateless)**
- Moyens pour assurer une haute **Interopérabilité**
- **Faiblement Couplés**
- **Découvrables** (dynamiquement)
- **Composables**

Réutilisabilité par contrat

- Le service est réutilisable conformément à un contrat entre le fournisseur et le consommateur
- Le **contrat** décrit :
  - La syntaxe du service : opération, input, output, format, protocole…
  - La sémantique de son utilisation: pré-conditions, post-conditions…
  - Sa QOS : temps de réponse attendu, temps de reprise après interruption…
- Le contrat est généralement décrit au moyen du standard WSDL
- Plusieurs contrats peuvent être définis pour répondre aux besoins différents des consommateurs (ex : service avec haute disponibilité/disponibilité normale)
- Le contrat est utilisé au design-time (génération de code) et au run-time (contrôle du respect du contrat)

Interface adressable et communication par message

- Chaque consommateur peut invoquer un service via son **adresse dans le réseau** à n'importe quel moment
  - Le consommateur peut accéder localement au service pour augmenter la performance, s'ils sont hébergés dans la même machine
- Les services communiquent **uniquement par messages**
  - Appels via le réseau vu que les services sont distribués en SOA
- Pour augmenter la performance, les concepteurs doivent penser à augmenter la granularité des interfaces de services pour diminuer le nombre d'appels réseau

Abstraction et Prédictibilité

- Le service fonctionne en « **boîte noire** »
  - Seul le contrat du service (informations nécessaires pour l'invocation) est exposé au consommateur du service
  - le fonctionnement interne du service (sa logique métier et son implémentation) ne sont pas visibles
- Il est **Prédictible**
  - Son comportement et sa réponse lors de la réception d'une requête ne varient pas

Large granularité et modularité

- **Large granularité :** Le service est un **gros grain** qui regroupe un **ensemble d'interfaces cohérentes** se rapportant à un même module fonctionnel
  - Principe à respecter lors de la conception

- **Modularité :** Il peut être **déployé de façon atomique** bien avant le développement ou déploiement d'applications consommatrices
  - Principe différent du principe du paradigme OO où un programme OO est une unité indivisible

Autonomie et statelessness

- **Autonomie :**
  - Le service est **Indépendant des services externes** : son comportement est indépendant du contexte fonctionnel et technique dans lequel il a été invoqué

- **Statelessness :** Il est **sans état** (stateless) càd il n'intègre pas la gestion de contexte (puisqu'il est autonome)
  - But : Ne pas compliquer la maintenance, préserver la réutilisabilité (Indépendance d'un enchaînement particulier) et assurer la performance (minimiser la consommation de ressources systèmes, utilisées pour le stockage d'états)

Interopérabilité

- **Possibilité de communiquer avec un système hétérogène**
- Le service **précise un type de connecteur** (càd protocole et format de données) que ses clients potentiels doivent utiliser pour pouvoir invoquer l'interface qu'il fournit
- Une **spécification de médiation** permettra de réaliser le mapping au cas où le client adopte un format et types de données hétérogènes
  - Mapping entre deux jeux de caractères comme l'ASCII et EBCDIC, et mapping de types de données
  - Exemples de spécification de médiation : Les API JAX-RPC et JAXM pour le mapping des types de données Java aux types de données SOAP et XML dans le cas d'un service Web

Couplage faible (lâche)

- **Dépendance faible entre le consommateur et le service**
  - **Dépendance du contrat** et non pas de l'implémentation
  - Echange à travers des **messages**
  - **Orchestration** assure l'indépendance des services vu qu'elle leur permet de communiquer pour réaliser un processus, sans avoir à se connaître
- Avantage : **Maintenance facile**
  - un changement dans le service suscite peu de changements dans ses consommateurs (juste ceux relatifs au respect du contrat)

Découvrabilité

- Il est **publié** par le fournisseur dans un annuaire : décrit par un ensemble de métadonnées qui permettent de l'identifier et qu'il est possible de màj
- Le consommateur peut **chercher** un service selon un ensemble de critères à partir de l'annuaire :
  - L'annuaire renvoie au consommateur la liste des services (adresses, frais…) qui répondent à sa requête
  - Tous les arguments nécessaires à l'exécution du service sélectionné (opérations, paramètres…) sont accessibles à partir de son contrat

Composabilité

- Un service peut **participer à des compositions** de services
  - Un ensemble de services peuvent être composés à travers leur orchestration pour répondre à un besoin complexe

- Avantages :
  - Apport de **valeur ajoutée** (répondre à un nouveau besoin complexe)
  - Augmentation de la **modularité** : vu qu'un service complexe peut être décomposé en services simples pouvant être déployés chacun de façon atomique