Université Nice SOPHIA ANTIPOLIS

Applications Réparties : REST

* D'après les cours de:

Roger Costello, Stéphane Lavirotte

Introduction

- ✓ Pourquoi des applications réparties ?
  - Besoin de rendre des applications accessibles par le web
    - Elargir l'audience des utilisateurs
    - Vendre des services en ligne
    - Faire communiquer des applications existantes
- ✓ Comment mettre en œuvre des applications réparties ?
  - Quelle architecture choisir ou concevoir ?
  - Quel format utiliser pour échanger des données sur le web ?
  - Doit-on utiliser un protocole applicatif existant ou développer un protocole adapté à nos besoins ?
- ✓ Rendre une application accessible par le web
  - Définir un format de données
  - Définir un protocole applicatif

Interopérabilité

- ✓ Développer des formats de données
  - Tellement de domaines d'application: impossible d'avoir un format unique
  - XML définit des règles de structuration de l'information
- ✓ Développer des protocoles génériques
  - Permettre à toute application communique sur le Web
    - Approche adoptée par XML-RPC et SOAP
  - Paradigme de l'appel de méthode à distance
    - Avantages:
      - Facilite le travail du développeur (modèle d'interaction identique au modèle de l'application)
      - Améliore l'interopérabilité en spécifiant le format dans lequel les informations doivent être envoyées
    - Inconvénient
      - Doit généraliser être généralisé à tous les modèles d'interaction:
        - Communication synchrone ou asynchrone
        - Communication avec ou sans état

REST

RESTful Web Services

Reposant, Paisible

REST

- ✓ REpresentational State Transfer
  - Terme introduit par Roy Fielding dans sa thèse en 2000
  - REST n'est pas un protocole ou un format
  - C'est une manière de construire une application pour les systèmes distribués (style architectural)

- ✓ Architecture Orientée Ressource
  - Toute information qui peut être nommée est une ressource
  - Une ressource est identifiée par un identificateur (URI)

REST n'est pas un Standard

- ✓ REST n'est pas un standard
  - Pas de recommandation du W3C
  - Pas de toolkit éditée par Microsoft, IBM ou Sun
  - Mais utilise des standards: HTTP, URL, XML, HTML, MIME, ...
- ✓ REST est un style d'architecture
  - C'est un design pattern pour l'implémentation d'un système
  - Principes:
    - Les requêtes sont client-serveur
    - Les requêtes sont stateless (sans état)
    - Les clients accèdent à des ressources nommées: une ressources est représentée par une URL.
    - Les clients et serveurs adhèrent à une interface uniforme: toutes les ressources sont accédées via une interface générique: les méthodes HTTP

Avantages de REST

- ✓ Avantages:
  - Application est plus simple à entretenir: les liens sont mieux structurés (les liens sont le moteur de l'état de l'application)
  - Absence de gestion d'état du client sur le serveur:
    - moindre consommation mémoire
    - plus de simplicité (capacité à répondre à plus de requêtes)
    - possibilité de répartir les requêtes (*load balancing*)
      - Meilleure évolutivité et tolérance aux pannes
  - Utilisation des caractéristiques d'HTTP
    - SOAP ne pré-suppose pas de protocole même s'il utilise la plupart du temps HTTP (redites entre SOAP et HTTP)
  - Utilisation des URI pour représenter (nommer) des ressources
    - Possibilité de mise en cache très facilement

Inconvénients de REST

- ✓ Inconvénients:
  - Le client doit conserver toutes les données nécessaires pour le bon déroulement d'une requête
    - Consommation de la bande passante plus importante
  - L'utilisation d'un formulaire HTML envoyant ses données avec une méthode comme DELETE en général pas compris
    - on émule ce comportement avec un champ caché qui transmettra le type de méthode d'envoi des données à l'application

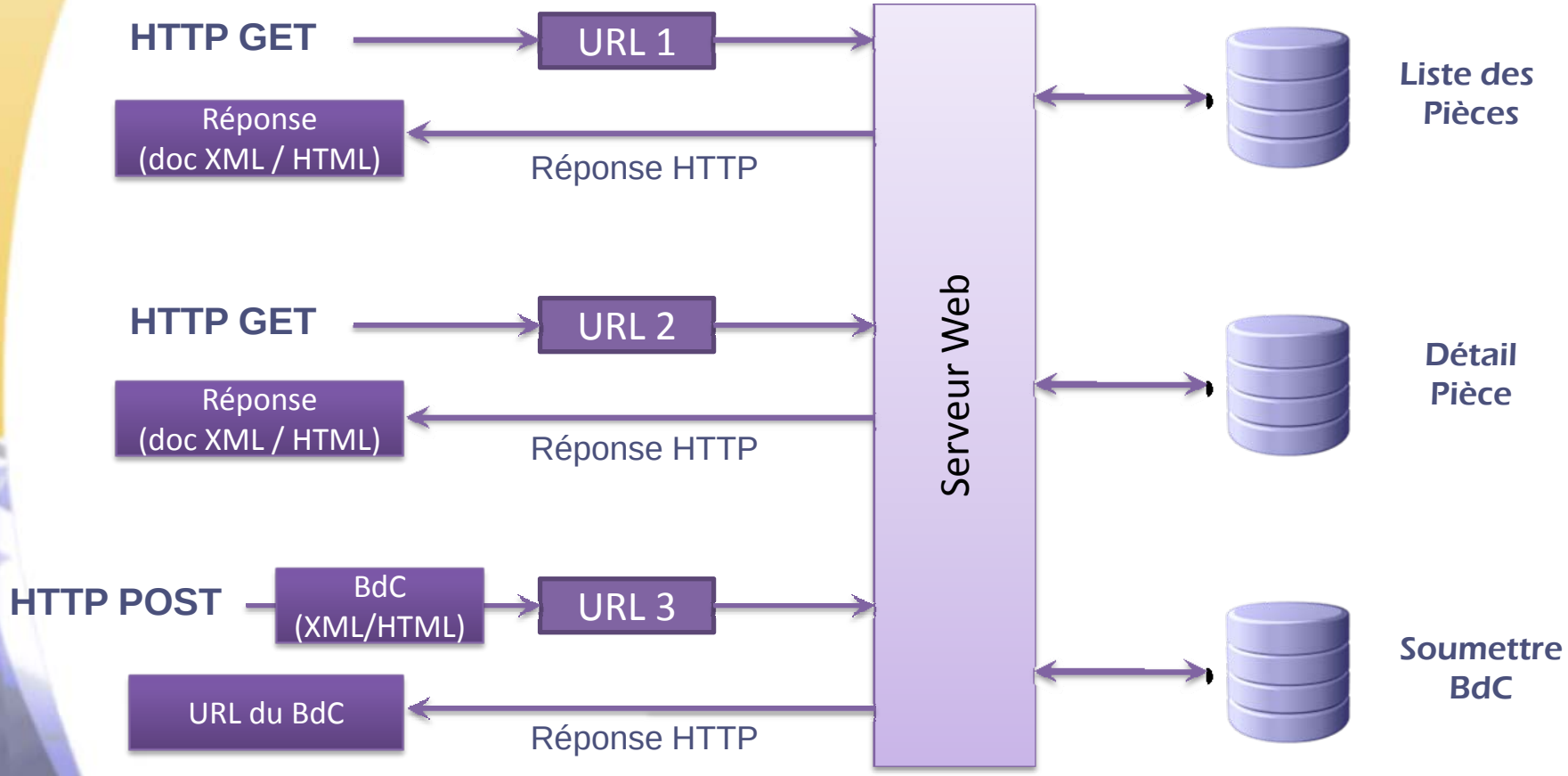
Comparaison de REST vs SOAP

Par l'exemple

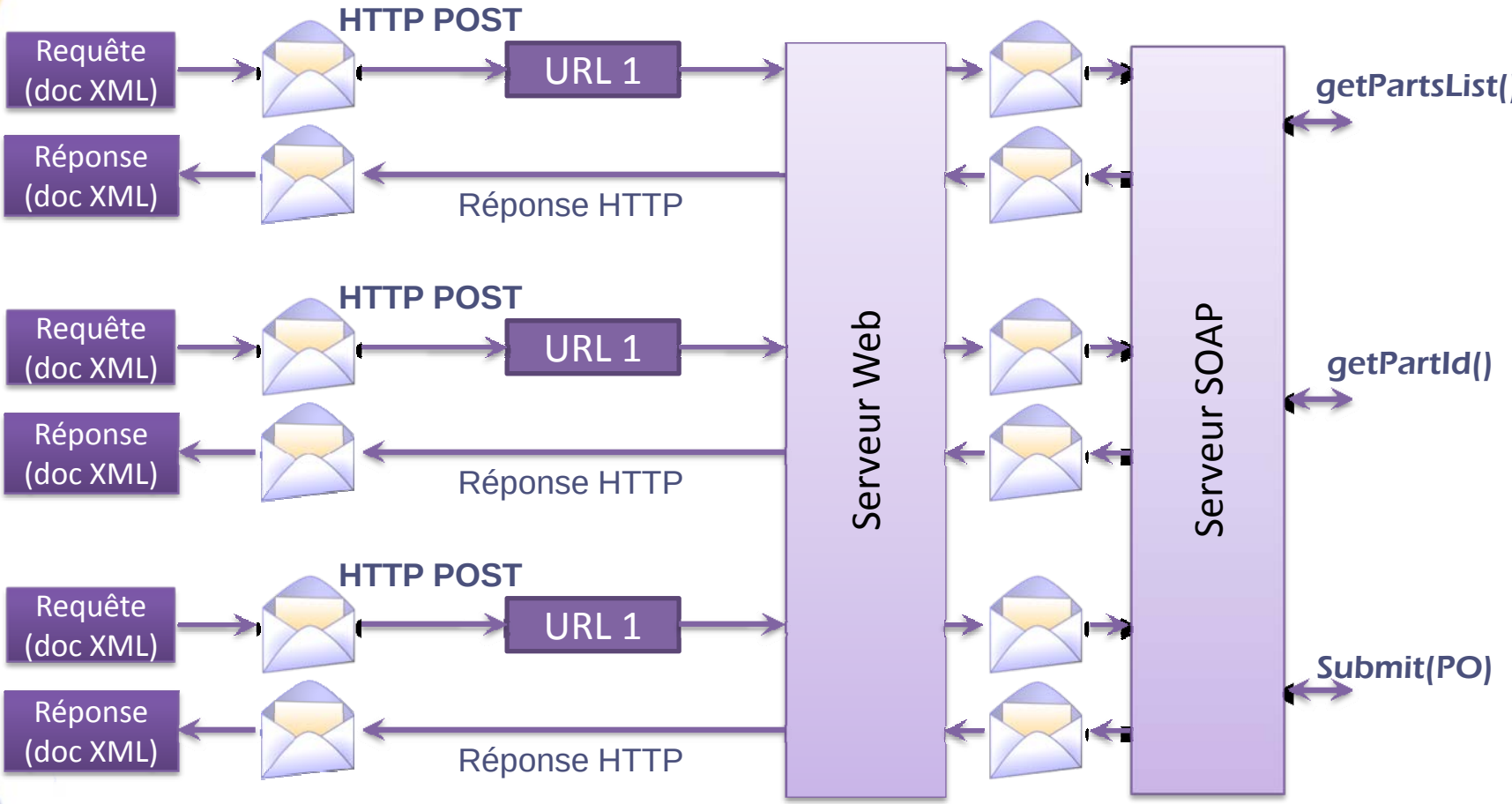
Comprendre REST par l'Exemple

- ✓ Ce type d'architecture se comprend mieux par l'exemple
- ✓ Exemple d'un Entrepôt de Pièces Détachées
  - Services Web accessibles pour les clients
    - Obtenir la liste des pièces détachées
    - Obtenir des informations détaillées sur une pièce donnée
    - Soumettre un bon de commande
- ✓ Nous allons étudier la mise en œuvre de cet exemple
  - Via REST
  - Via SOAP

Représentation REST

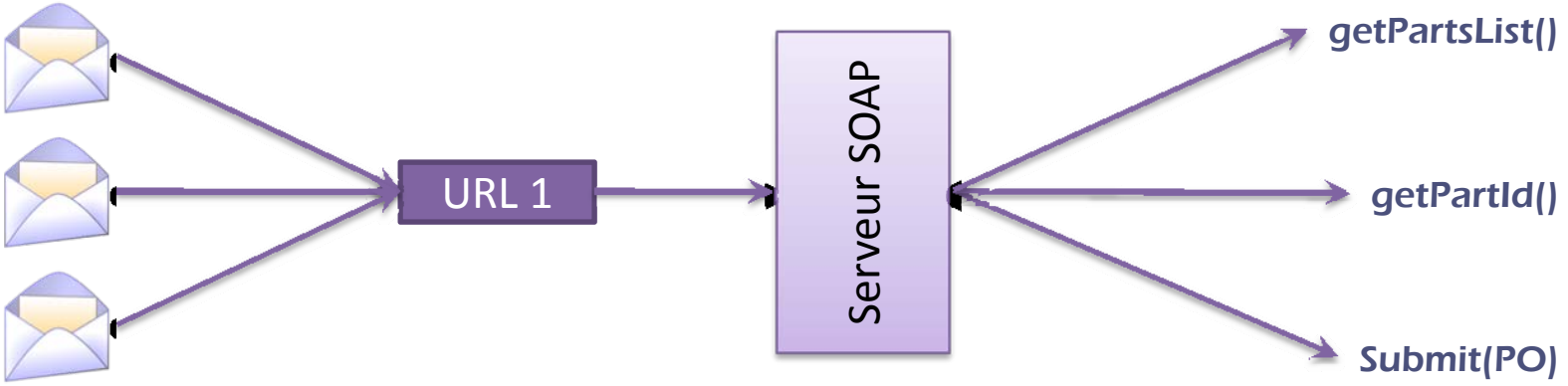


Représentation SOAP



Implémentation du WS avec SOAP

- Même si ce n'est pas une obligation, en SOAP, forme de tunneling sur la même URL



- Exemple de message SOAP:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <p:getPartsList xmlns:p="http://www.parts-depot.com"/>
  </soap:Body>
</soap:Envelope>
```

Données Retournées en REST

- ✓ **Exemple de réponse :**

```
<?xml version="1.0"?>
<p:Parts xmlns:p="http://www.parts-depot.com"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.parts-depot.comhttp://www.parts-
   depot.com/parts.xsd">
  <Part id="00345" xlink:href="http://www.parts-depot.com/parts/00345"/>
  <Part id="00346" xlink:href="http://www.parts-depot.com/parts/00346"/>
  <Part id="00347" xlink:href="http://www.parts-depot.com/parts/00347"/>
  <Part id="00348" xlink:href="http://www.parts-depot.com/parts/00348"/>
</p:Parts>
```

- ✓ **On notera que chaque pièce de la liste a une URL qui permet d'y accéder. C'est un élément clé de REST**

Données Retournées en SOAP

- Exemple de réponse:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <p:getPartsListResponse xmlns:p="http://www.parts-depot.com">
      <Parts>
        <Part-ID>00345</Part-ID>
        <Part-ID>00346</Part-ID>
        <Part-ID>00347</Part-ID>
        <Part-ID>00348</Part-ID>
      </Parts>
    <p:getPartsListResponse>
  </soap:Body>
</soap:Envelope>
```

- Absence de lien vers les pièces retournées
  - A noter que les infos retournées pourraient pointées vers un service REST-ful

REST vs SOAP ou XML-RPC

- ✓ SOAP et XML-RPC
  - Se basent sur des méthodes (appels de méthodes à distance)
- ✓ REST
  - Se base sur les ressources existantes et l'échange de leur valeur via une URL qui la nomme
- ✓ De nombreux autres points de comparaison:
  - Serveurs Proxy (Web intermédiaires)
  - Etat de transition dans une application cliente
  - Cache (i.e., performance)
  - Evolution du Web (Web sémantique)
  - Interface générique (versus interface custom)
  - Interopérabilité
  - ...

Conclusion

Quelques recommandations sur les bonnes pratiques d'une architecture REST

Quelques Recommandations

- ✓ Fournir une URI pour chaque ressource que l'on souhaite (ou souhaitera exposer). Consistant avec l'axiome du Web de Tim Berners-Lee, ainsi que les recommandations du W3C
- ✓ Préférer les URIs logiques aux URI physiques. Permet à l'implémentation de la ressource de changer sans impacter les clients
  - Préférer http://www.airbus.com/airplanes/A320
  - à http://www.airbus.com/airplanes/A320.html
- ✓ Utiliser des noms pour les URI logique plutôt que des verbes. Les ressources sont des choses, pas des actions.
- ✓ Faire des GET HTTP sans effets de bord. Cela rend les choses plus sécurisées

Quelques Recommandations

- ✓ Utiliser des liens dans vos réponses aux requêtes. Ainsi cela connecte les réponses aux autres données. La réponse contient les informations sur les "prochaines opérations à effectuer".
- ✓ Minimiser l'utilisation des attributs HTTP
  - Préférer http://www.parts-depot.com/parts/00345
  - à http://www.parts-depot.com/parts?part-id=00345
- ✓ Utiliser le "/" pour modéliser une relation parent/enfant
- ✓ Utiliser une "méthode de déploiement progressif" pour exposer des données aux clients. Autrement dit, une ressources devrait fournir des liens pour obtenir plus de détails
- ✓ Utiliser toujours la méthode HTTP GET pour récupérer une information et pas la méthode HTTP POST