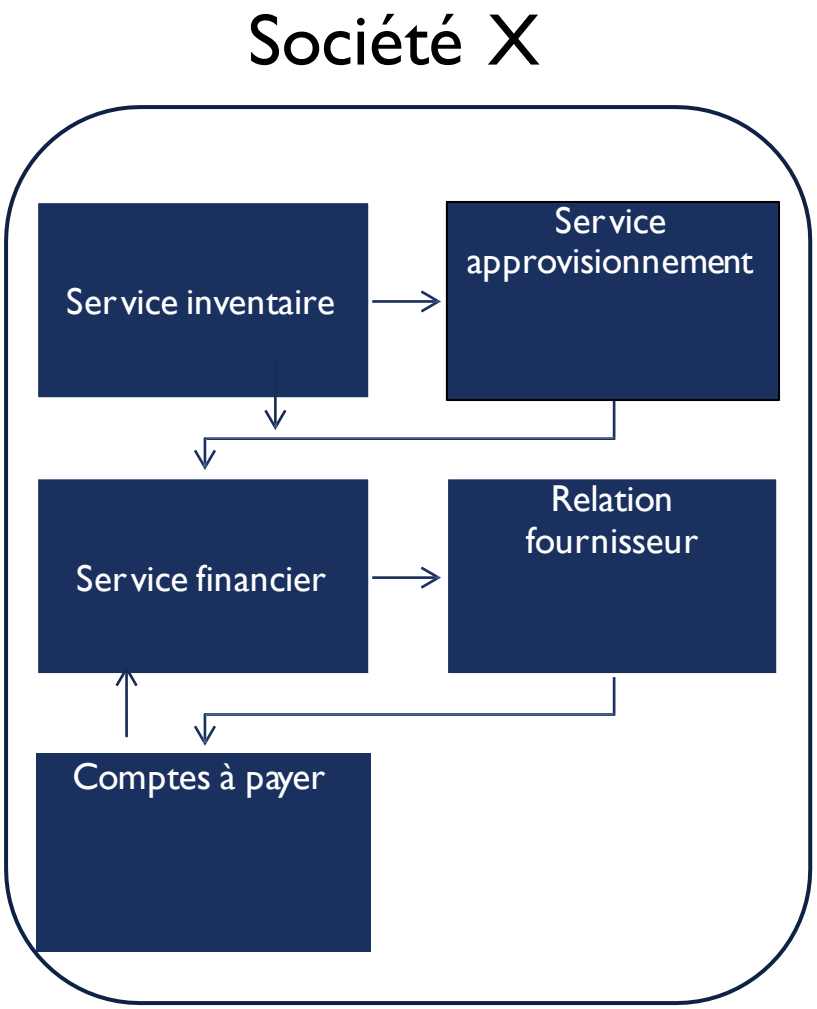
FORMATION WEB SERVICES

Par Dr Hmida ROJBANI

INTRODUCTION AUX ARCHITECTURES ORIENTÉES SERVICES

LA PROBLÉMATIQUE PAR UN EXEMPLE



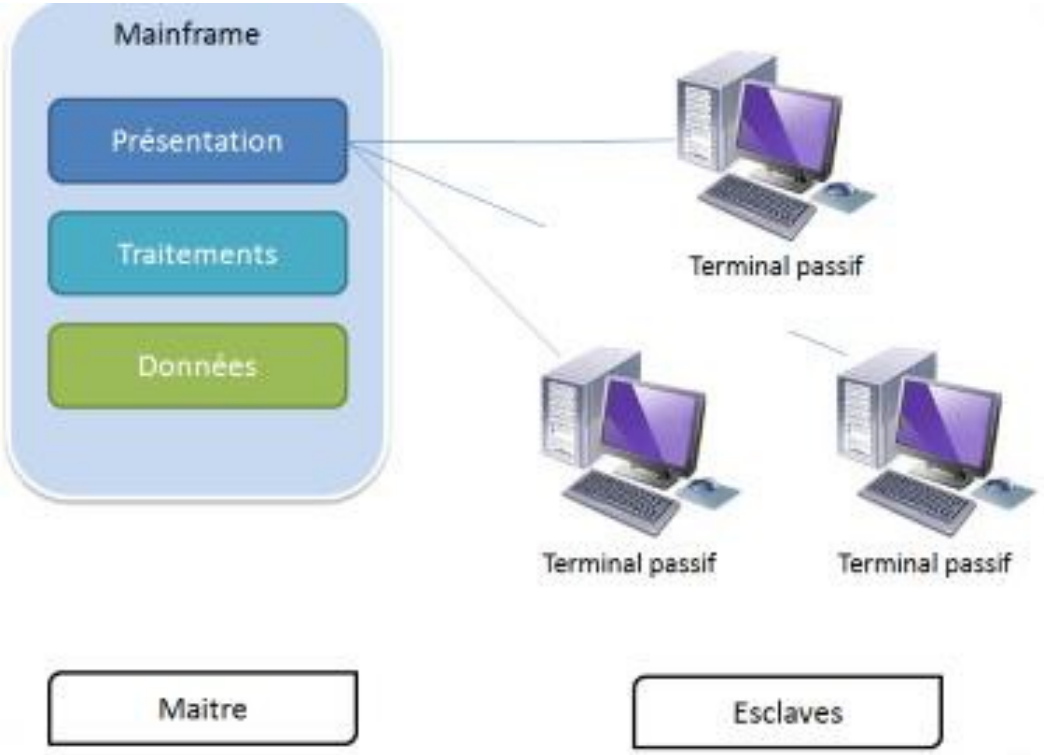
ARCHITECTURE 1-TIER

ARCHITECTURE 1-TIER

- Les trois couches applicatives sont intimement liées et s'exécutent sur le même ordinateur
- On parle d'informatique centralisée (applications sur un site central : Mainframe)
- Les utilisateurs se connectent aux applications exécutées par le serveur central (le mainframe) à l'aide de terminaux passifs

APPLICATION SUR « MAINFRAME »

- C'est le serveur central qui prend en charge l'intégralité des traitements, y compris l'affichage qui est simplement déporté sur des terminaux passifs.

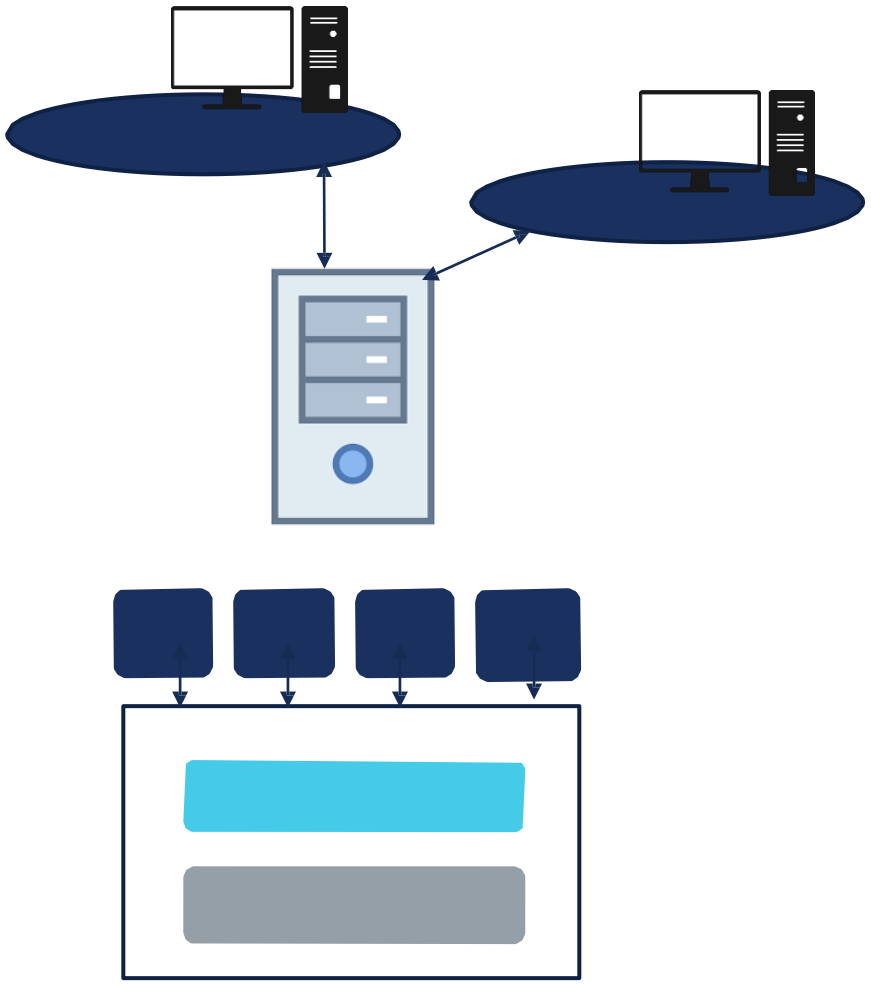


APPLICATION SUR « MAINFRAME »

- Tous les programmes sont déployés sur cet ordinateur unique!
- Tous les programmes travaillent dans le même environnement logiciel et avec les mêmes technologies

- **Mais au début des années 2000…**
- Il fallait intégrer au SI de plus en plus de logiciels à haute valeur ajoutée (ERP, E-commerce, CRM…).
- Chacun de ces systèmes a besoin de son environnement adapté.
- Se ramener à mettre en place un serveur dédié pou chaque logiciel.

ARCHITECTURES 2-TIER :CLIENT / SERVEUR



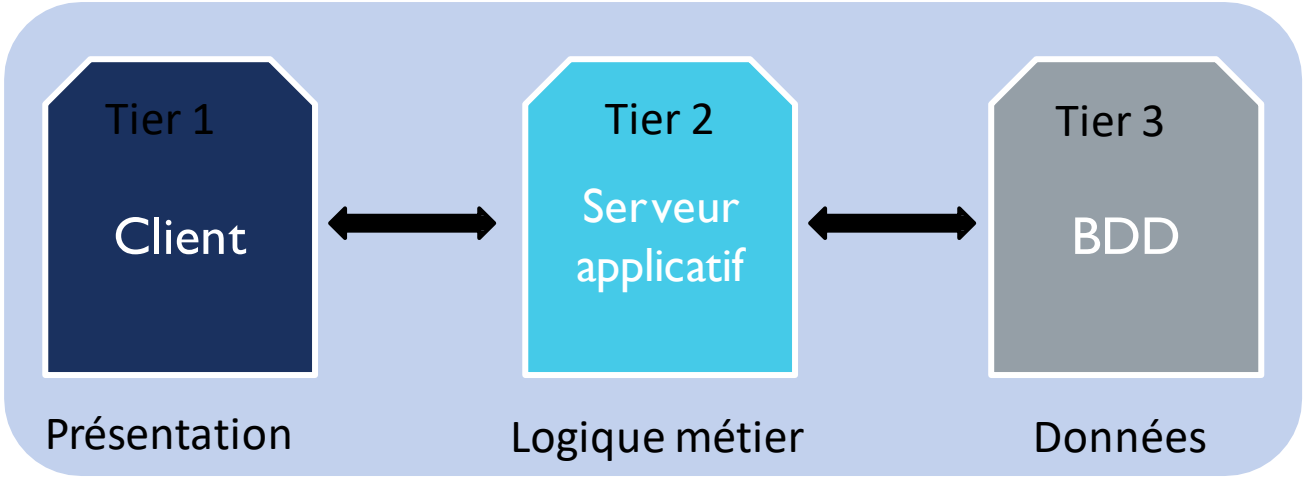
- Avec l'arrivée des « PCs », la couche présentation s'est déplacée au côté client.
- Dans certaines applications, uniquement la base de données est maintenue au côté serveur.
- Les améliorations des puissances des ordinateurs personnels ont permis d'implémenter des couches présentation plus sophistiquées (Choix entre client léger et client lourd).
- Le concept de API « Application Program Interface » est apparu.

LIMITES DES ARCHITECTURES 2-TIER

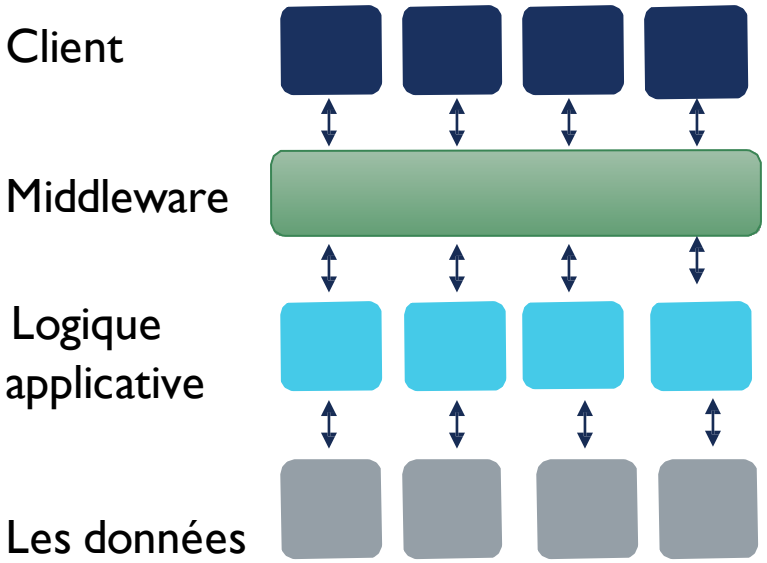
- Pour le client, en cas de besoin d'intégration à différents serveurs, doit maitriser l'API propre à chaque serveur.
- Les serveurs sont totalement séparés.
- Le client se charge de localiser les services, récupérer les réponses, gérer les exceptions … → la charge du client devient de plus en plus importante.

ARCHITECTURE 3-TIERS

- Les données sont toujours gérées de façon centralisée.
- La présentation est toujours prise en charge par le poste client.
- La logique applicative est prise en charge par un serveur intermédiaire.
- Le 3-tiers sont typiquement distribués.



LA COUCHE « MIDDLEWARE »



- Le « middleware » est un niveau d'abstraction entre le client et les autres couche logicielles.
  - Il simplifie la conception côté client en minimisant le nombre d'interfaces.
  - Il se charge de localiser les ressources, y accéder et retourne les résultats.
- Le middleware est un système d'informations comme les autres.Il peut être de type 1-tier,2-tiers,3-tiers… → **architectures N-Tiers**.

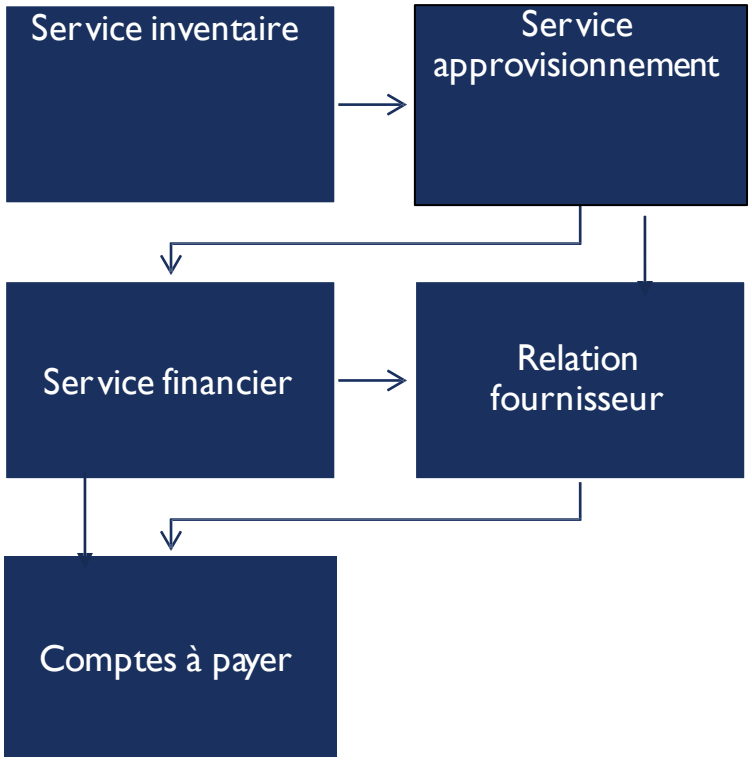
DES NOUVEAUX DÉFIS FACE AUX SI

- Mettre en place chacun des logiciels (ERP, CRM, E-commerce…) dans un serveur séparé a ramené une nouvelle problématique:

Comment faire communiquer ces différents systèmes?

LES CONNECTEURS: UNE PREMIÈRE SOLUTION

- Assurer la communication via des **adaptateurs** (connecteurs)



- Limites de cette solution:
  - **Réduit l'évolutivité du SI**
  - **Flexibilité limitée**
  - **Coût de maintenance**
  - **…**

UNE SECONDE SOLUTION

- Avec l'arrivé du XML en tant que standard, une nouvelle approche a été proposée:
  - Décomposer le SI en des **briques logicielles indépendants** réalisant chacune des opérations **élémentaires.**
  - Ses briques sont appelée des **services.**
  - Les services communiquent ensemble grâce à un protocole standard **SOAP** (basé sur XML)
- Exemple: un service qui gère les comptes des clients d'une banque.
  - La banque se propose d'utiliser ce service dans sa nouvelle application mobile.

DES NOUVEAUX DÉFIS

Invoquer le service

Comment indiquer à l'application mobile ou trouver le service de gestion des comptes des clients

WSDL

Localiser le service

Comment les autres services trouvent le documentWSDL

UDDI

Echange de données

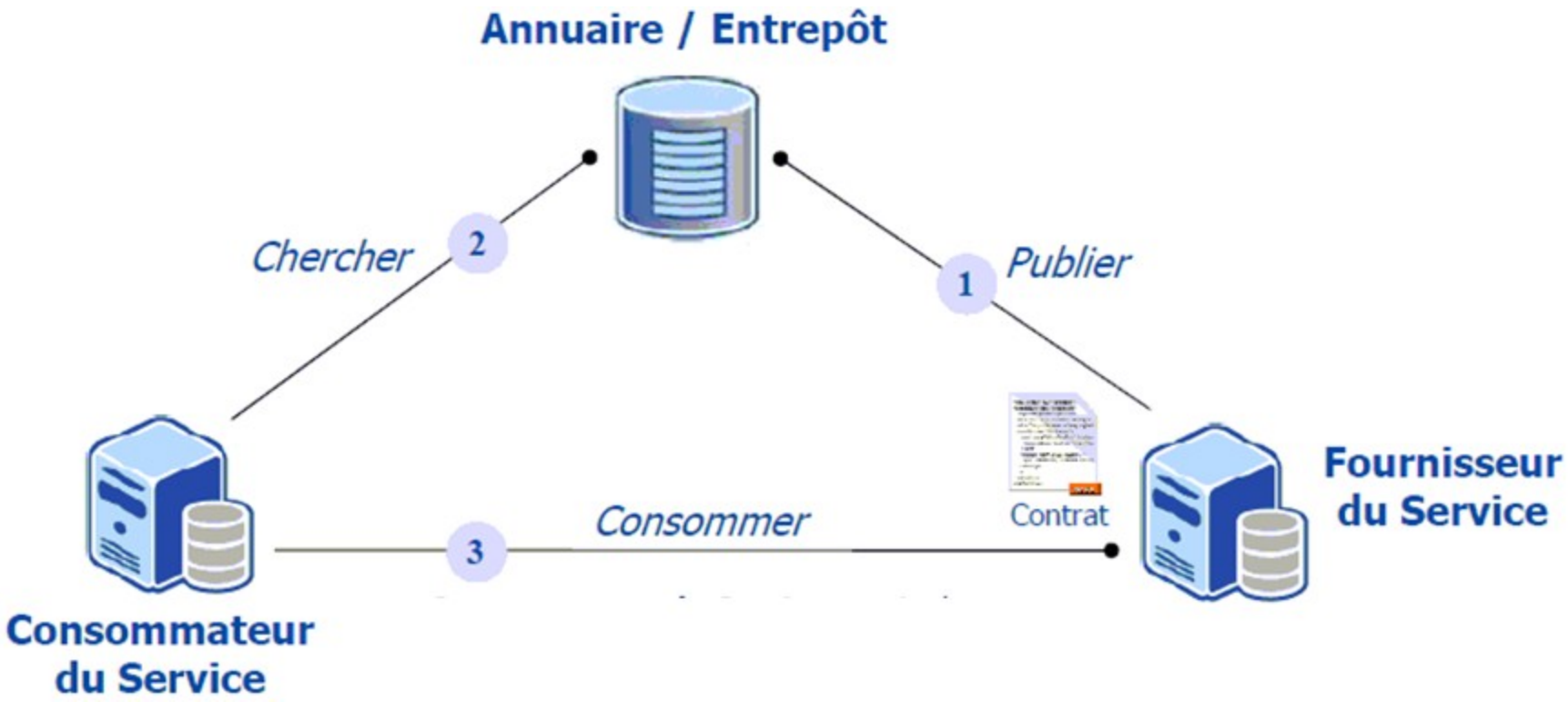
Selon quel format et quelle structure de données les données sont échangées?

SOAP

PLAN DU COURS

- Problématique et contexte
- Evolution des architectures logicielles
- Les composants des architectures orientées services
- Caractéristiques des architectures orientées services

STANDARDS D'UNE AOS



STANDARDS D'UNE AOS

- Le Fournisseur du service:publie son service via le contrat.
- Le consommateur du service (Le client) cherche un service répondant à ses exigences.
- Le consommateur du service envoie des messages au fournisseur de services tout en respectant le contrat.

PLAN DU COURS

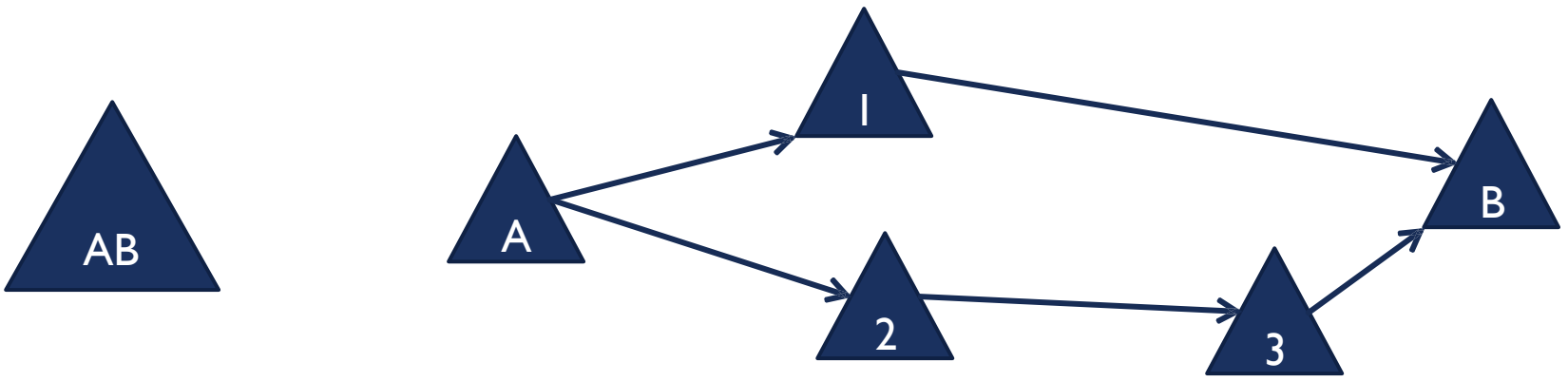
- Problématique et contexte
- Evolution des architectures logicielles
- Les composants des architectures orientées services
- Caractéristiques des architectures orientées services

ÉCHANGE DE MESSAGES

- Comment communiquer ?
- Les fournisseurs de services communiquent avec leurs clients en échangeant des messages.Ils sont définis par les messages qu'ils peuvent accepter et les réponses qu'ils peuvent donner.

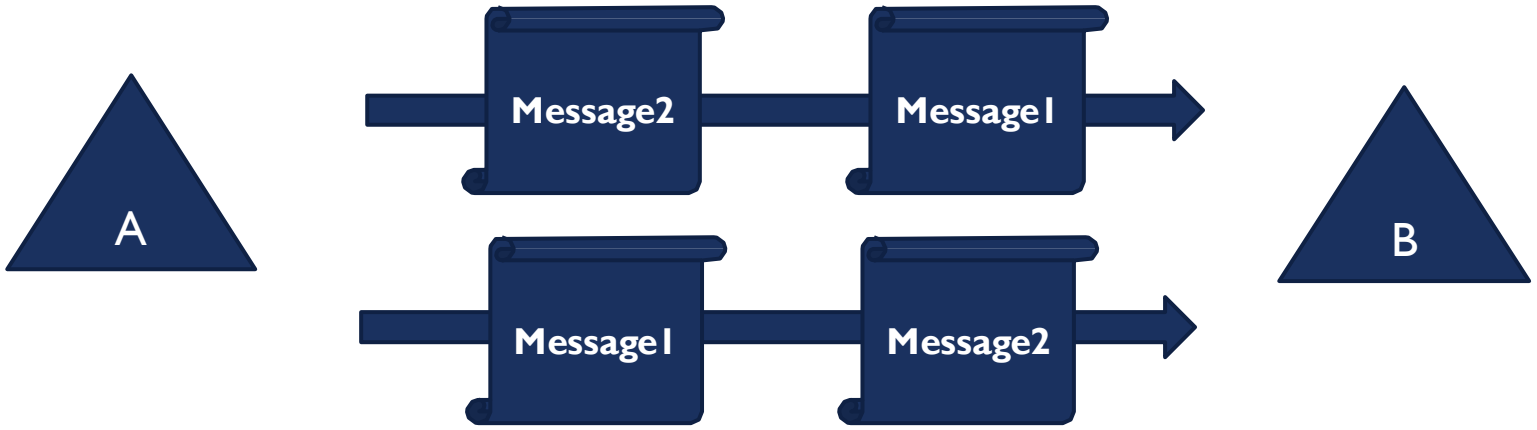
LA COMPOSITION DE SERVICES

- Réutilisation par composition : Les services peuvent être individuellement utiles, ou ils peuvent être intégrés, composés pour fournir des services de haut niveau. Ce qui favorise la réutilisation des fonctionnalités existantes. Cette notion est définie comme « la composition de services »



CHORÉGRAPHIE DE SERVICES

- Ordre d'appel des services : Les services peuvent participer à un workflow, où l'ordre dans lequel les messages sont envoyés et reçus affecte le résultat des opérations effectuées par un service. Cette notion est définie comme « la chorégraphie de services »



DÉPENDANCE

- Les services peuvent être totalement autonome, mais peuvent aussi dépendre de la disponibilité des autres services, ou sur l'existence d'une ressource comme une base de données par exemple.
- Exemples:
  - Service indépendant: Dans le cas le plus simple, un service peut effectuer un calcul comme le calcul de la racine cubique d'un nombre fourni sans avoir besoin de se référer à une ressource externe, ou il peut aussi charger à l'avance toutes les données dont il a besoin pour son exécution.
  - Service dépendant : Inversement, un service qui effectue la conversion des devises devrait accéder en temps réel aux informations des taux de change permettant de dégager des valeurs correctes.

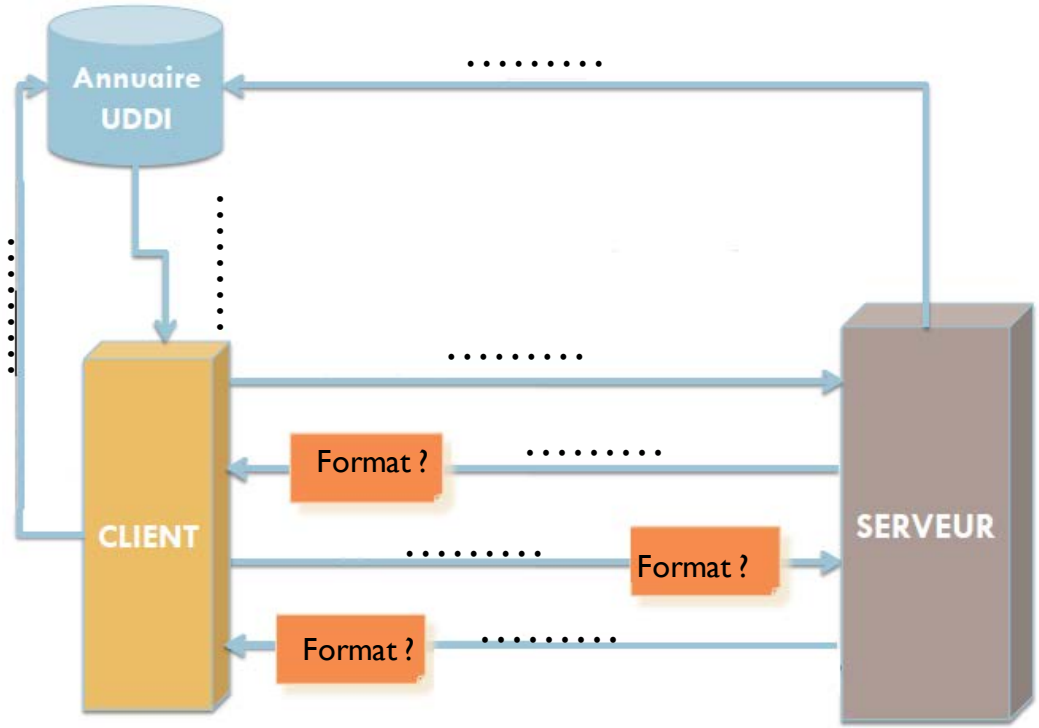
CODE DE DÉVELOPPEMENT INACCESSIBLE

- Les Services publient des détails tels que leurs capacités, les interfaces, les politiques et protocoles pris en charge pour la communication.
- Les détails d'implémentation comme le langage de programmation et la plate-forme d'hébergement ne sont pas préoccupantes pour les clients, et ne sont pas révélées pour des raisons de sécurité.

AOS :VUE D'ENSEMBLE

- Le Service :le fondement du modèle d'interaction entre applications.
- Le paradigme SOA :,**Publier,Chercher** et **Consommer.**
- Le terme **architecture orientée services** fait référence à un style de construction fiables des systèmes distribués qui offrent des fonctionnalités comme les services,en mettant l'accent sur la faiblesse de l'association entre les services qui interagissent (couplage faible= association très faible)

UNE RÉFLEXION : SCÉNARIO FOURNISSEUR / CONSOMMATEUR D'UN SERVICE



Complétez le schéma par les étapes ci-dessous en les numérotant par ordre chronologique.

1. J'appelle ton service.
2. Le client cherche un service.
3. L'annuaire a trouvé le service demandé et renvoie l'information au serveur qu'il héberge.
4. Voici le résultat du service web
5. Quel est le contrat du service web que tu proposes?
6. Le fournisseur publie ses services web.
7. Voici mon contrat.

LES STANDARDS D' ÉCHANGE DE DONNÉES

DÉFIS DANS UNE ARCHITECTURE ORIENTÉE SERVICES → DES STANDARDS

Invoquer le service	Echange de données	Localiser le service

- Comment permettre au client d'invoquer un service distant mise à côté la technologie de la manière la plus transparente

WSDL

- Selon quel format et quelle structure de données les données sont échangées?

SOAP

- Comment localiser le service adéquat parmi une large collection de services disponibles

UDDI

« EXTENSIBLE MARKUP LANGUAGE »

PRÉSENTATION ET SYNTAXE

PRÉSENTATION

- XML (eXtensible Markup Language) :langage à balises extensible
  - permettant de définir de nouvelles balises.
  - permettant de mettre en forme des documents grâce à des balises (markup).
- Contrairement à HTML (langage défini et figé avec un nombre de balises limité),XML est un **métalangage** permettant de définir d'autres langages.
- Peut décrire n'importe quel domaine de données grâce à son extensibilité.
- Permet de structurer,poser le vocabulaire (grammaire) des données

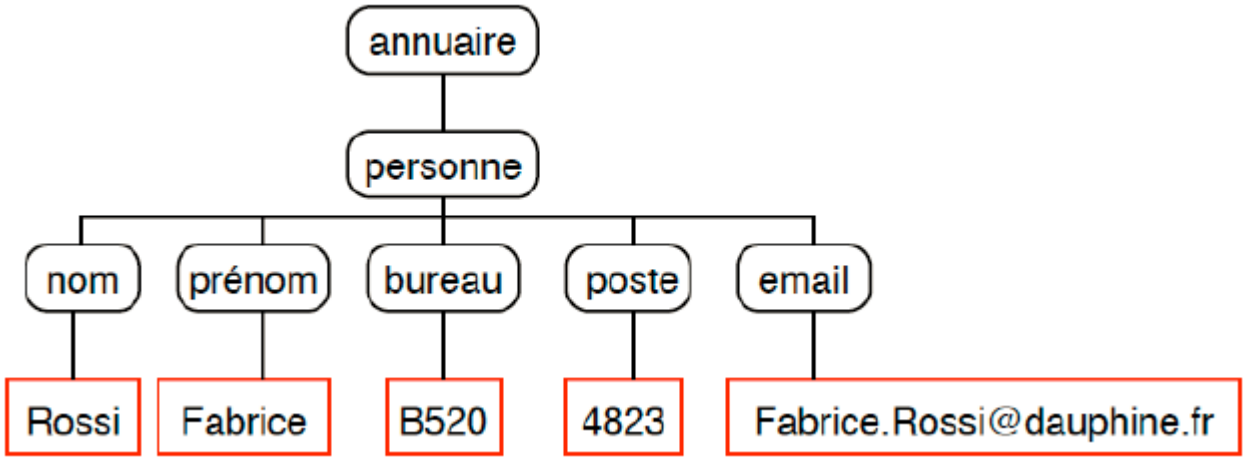
DONNÉES ET STRUCTURE

- XML permet de représenter des données structurées :
  - Données textuelles organisées :
  - Un **document** constitué **d'éléments**
  - Un **élément** peut être constitué de texte ou contenir d'autres éléments (ou un mélange des deux)
  - Un élément peut être associé à des informations complémentaires, les **attributs**
- La structure est celle d'un arbre :
  - Un **document XML** = un **arbre**
  - Un **élément** = un nœud de l'arbre
- Le standard XML indique comment traduire l'arbre en un document XML, pas comment organiser les données

EXEMPLE : ANNUAIRE (1/4)

- **But** : stocker l'annuaire de Dauphine (nom, prénom, bureau, numéro
- de poste, email)
- Le texte du document : les informations !
- Organisation : s'arranger pour que les informations restent correctement groupées (ne pas mélanger les données !)

Solution 1 :



EXEMPLE : ANNUAIRE (2/4)

- Traduction en XML de l'arbre :

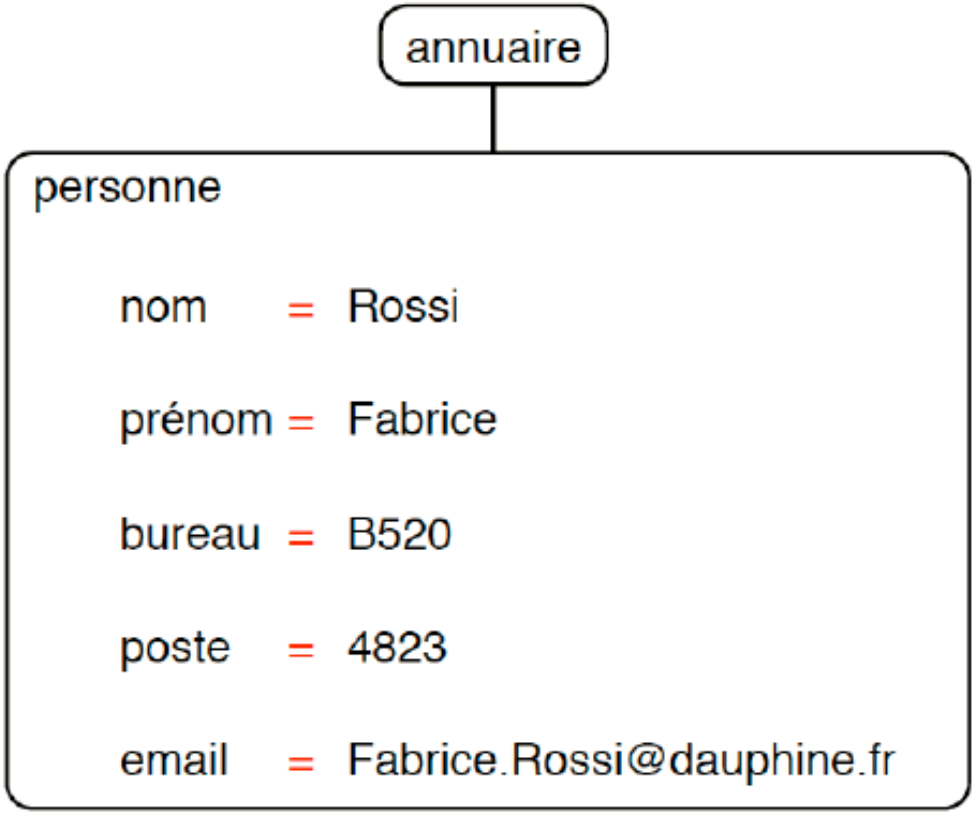
annuaire1.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <annuaire>
3      <personne>
4          <nom>Rossi</nom>
5          <prenom>Fabrice</prenom>
6          <bureau>B520</bureau>
7          <poste>4823</poste>
8          <email>Fabrice.Rossi@dauphine.fr</email>
9      </personne>
10     <!-- suite de l'annuaire -->
11 </annuaire>
```

- Inclusion textuelle ⇒ relation mère-fille dans l'arbre
- Balise ouvrante ou fermante ⇒ nom d'un nœud
- Texte => feuille de l'arbre
- Ne pas confondre les éléments (information) et les balises (syntaxe).

EXEMPLE :ANNUAIRE (3/4)

Solution2:



EXEMPLE : ANNUAIRE (4/4)

- Traduction en XML de l'arbre :

annuaire2.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <annuaire>
3      <personne    nom="Rossi"
4                   prenom="Fabrice"
5                   bureau="B520"
6                   poste="4823"
7                   email="Fabrice.Rossi@dauphine.fr"/>
8      <!-- suite de l'annuaire -->
9  </annuaire>
10
```

- Attributs ➔ annotations d'un nœud
- élément vide ➔ feuille

ORGANISATION D'UN DOCUMENT XML

- On organise les données en décidant de la structure de l'arbre :
  - le nom des éléments
  - l'ordre des éléments
  - les relations d'inclusion
  - la position des données (c'est-à-dire du texte)
  - les contraintes sur les données (texte quelconque, valeur numérique, etc.)
  - les attributs
- Une organisation particulière forme un **vocabulaire** XML, par exemple:
  - **MathML** : pour décrire des équations
  - **SVG** : dessin vectoriel…

LE SYNTAXE XML

- Deux niveaux syntaxiques :
  - bas niveau : document **bien formé**
  - haut niveau : document **valide** (haut niveau => bas niveau)
- Du point de vue utilisateur/concepteur :
  - le bas niveau est **obligatoire** : **mal formé** => pas XML
  - le bas niveau est **fixé** par la norme
  - le haut niveau est **facultatif** : **bien formé** => XML
  - le haut niveau est entièrement de la responsabilité du concepteur : il définit les contraintes syntaxiques (noms de éléments, organisation, etc.)
  - le haut niveau peut se mettre en œuvre de différentes façons (DTD, schémas W3C, Relax NG, etc.)

DOCUMENTS XML BIEN FORMÉS (1/3)

- Les éléments :
- <truc> : balise ouvrante :
  1. doit toujours correspondre à une balise fermante (</truc>)
  2. le texte entre <> est le nom de l'élément : constitué de lettres, chiffres, '.', '-', '_' et ' :'
- </quantite> : balise fermante (depuis une balise ouvrante jusqu'à une balise fermante : le contenu d'un élément, un nœud de l'arbre)
- <et_hop/> : balise mixte, ouvrante et fermante, pour les éléments vides
- **Exemples : Ces balises sont bien ou mal formées ?**
  - <a> — **mal formé**
  - bla, bla, bla
bla, bla, bla — **mal formé**
  - <nom.pas:tres_bien-choisit/> — **bien formé**

DOCUMENTS XML BIEN FORMÉS (2/3)

- **Les attributs** :
  - Exemple : <font name="times">
    - name est un attribut de l'élément font, de valeur times.
  - ne peut apparaître que dans une balise ouvrante ou mixte
  - doit **toujours** avoir une valeur
  - la valeur est **toujours** délimitée par des guillemets " ou des apostrophes '
  - dans la valeur, < est interdit
  - pour le nom d'un attribut, même contrainte que pour les éléments
  - dans une même balise ouvrante ou mixte, chaque attribut ne peut apparaître qu'une fois

DOCUMENTS XML BIEN FORMÉS (3/3)

- **Grammaire de base :**
  - Un document XML est un arbre d'éléments :
    - la racine est **unique**
    - le contenu d'un élément est :
      - d'autres éléments
      - du texte (< et & sont interdits)
    - **Exemples : Ces balises sont bien ou mal formées ?**
      - <a> — **mal formé**
      - <a>3<2 — **Mal formé**
      - <a>3>2 — **Bien formé (déconseillé)**
      - <a>bla
bla — **Bien formé**

CONSTRUCTIONS UTILES

- Commentaires :
  - <!-- ce qu'on veut sauf deux - à la suite -->
- CDATA (texte) :

cdata.xml

```
<![CDATA[ <a>contenu<b> non interprete,
 non analyse, ne fait pas</a> partie
de l'arbre</b> ]]>
```

L'ENTÊTE DU FICHIER XML

- il est **conseillé** de commencer un document XML par :
  - < ?xml version="1.0" ?>
- l'attribut **encoding** permet d'indiquer la représentation physique des caractères du fichier :
  - < ?xml version="1.0" encoding="UTF-16" ?>
  - < ?xml version="1.0" encoding="UTF-8" ?> par défaut
  - < ?xml version="1.0" encoding="ISO-8859-1" ?> sous linux
- < ?nom ?> : une **Processing Instruction** indique aux logiciels comment traiter le document:
  - encodage
  - associer une feuille de style à un document

LES ESPACES DE NOM

- Un document XML peut contenir des éléments et des attributs qui correspondent à plusieurs domaines distincts (i.e.,à plusieurs dialectes).

```
<widget type="gadget">
        <head size="medium"/>
        <big><subwidget ref="subInfo"/></big>
        <info>
                <head>
                        <title>Description of gadget</title>
                </head>
                <body>
                        <h1>Gadget</h1>
                        A gadget contains a big idea
                </body>
        </info>
</widget>
```

Problème : comment gérer les collisions ?

LES ESPACES DE NOM (2)

- **Solution** => Les espaces de noms (**namespaces**) :
  - permet d'introduire des collections de noms utilisables pour les éléments et les attributs d'un document XML
  - **principes** :
    - chaque collection est identifiée par un URI
    - un préfixe est associé à un URI

LES ESPACES DE NOM (3)

```
<widget type="gadget" xmlns="http://www.widget.inc">
    <head size="medium"/>
    <big><subwidget ref="subInfo"/></big>
    <info xmlns:xhtml="http://www.w3.org/TR/xhtml1">
        <xhtml:head>
            <xhtml:title>Description of gadget</xhtml:title>
        </xhtml:head>
        <xhtml:body>
        <xhtml:h1>Gadget</xhtml:h1>
            A gadget contains a big idea
        </xhtml:body>
    </info>
</widget>
```

LES ESPACES DE NOM (4)

- Déclaration d'un namespace :
  - **xmlns:préfixe="URI"** : association du préfixe à l'URI
  - **xmlns="URI"** : définition de l'URI associé à l'espace de noms par défaut (sans préfixe)
- Nom qualifié :
  - **préfixe : nom local**
  - peut être utilisé pour les attributs et les éléments
  - le préfixe doit être déclaré par un ascendant
- Remarque :
  - C'est l'URI qui assure l'absence d'ambiguïté, pas le préfixe

QUELQUES « NAMESPACES » CLASSIQUES

- Quelques espaces de noms classiques
  - XML:
  - MathML: http://www.w3.org/XML/1998/namespace
  - XHTML: http://www.w3.org/1998/Math/MathML
  - SVG: http://www.w3.org/1999/xhtml
  - XSLT: http://www.w3.org/2000/svg
    http://www.w3.org/1999/XSL/Transform

EXEMPLE :« CHEMICAL MARKUP LANGUAGE »

```
<molecule id="METHANOL">
    <atomArray>
        <stringArray builtin="id">a1 a2 a3 a4 a5 a6</stringArray>
        <stringArray builtin="elementType">C O H H H H</stringArray>
        <floatArray builtin="x3" units="pm">
            -0.748 0.558 ...
        </floatArray>
        <floatArray builtin="y3" units="pm">
            -0.015 0.420 ...
        </floatArray>
        <floatArray builtin="z3" units="pm">
            0.024 -0.278 ...
        </floatArray>
    </atomArray>
</molecule>
```

EXEMPLE 2 :DOCUMENT SVG

Une ellipse blanche de centre (220,100) et de rayons (190,20) à l'intérieur d'une ellipse jaune de centre (240,100) et de rayons (220,30)

```
<svg width="100%" height="100%" version="1.1" xmlns="http://www.w3.org /2000/svg">
      <ellipse cx="240" cy="100" rx="220" ry="30" style="fill:yellow"/>
      <ellipse cx="220" cy="100" rx="190" ry="20" style="fill:white"/>
</svg>
```


Résultat obtenu par un interpréteur SVG

XML EST UN SUCCÈS !

- StandardW3C
- La syntaxe XML ne contient que peu de mot clef: Simplicité
- XML est indépendant des plates-formes: Portabilité de fichiers plats.
- XML est un méta-langage, il est possible de créer ses propres balises: Extensibilité
- Outils disponibles (et gratuits)

Largement utilisé pour les échanges inter-applications

« EXTENSIBLE MARKUP LANGUAGE »

SCHÉMAS (XSD)

PRÉSENTATION DE XSD

- **Un XSD permet de**
  - Définir les éléments
  - Définir les attributs
  - Déclarer (et utiliser) des types
    - Simples (string, integer, …)
    - Complexes, description de sous-arbres
- **XSD est un langage lui-même basé sur XML**
- XSD est défini par le w3c : http://www.w3.org/XML/Schema
- Actuellement, XSD est la manière la plus répandue de décrire un document XML

PRINCIPES DES SCHÉMAS

- Un schéma XML est un document XML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
    <!-- Déclaration de deux types d 'éléments -->
    <xsd:element name="nom" type="xsd:string" />
    <xsd:element name= "prenom" type="xsd:string" />
</xsd:schema>
```

PRINCIPES DES SCHÉMAS

Document XML (Adresse.xml)

```
<?xml version="1.0"?>
<Addresse_postale_France pays="France">
        <nom>Mr Jean Dupont</nom>
        <rue>rue Camille Desmoulins</rue>
        <ville>Paris</ville>
        <departement>Seine</departement>
        <code_postal>75600</code_postal>
</Addresse_postale_france >
```

Schéma (Adresse.xsd)

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
<xsd:complexType name="Addresse_postale_france" >
          <xsd:sequence>
               <xsd:element name="nom" type="xsd:string" />
               <xsd:element name="rue" type="xsd:string" />
               <xsd:element name="ville" type="xsd:string" />
               <xsd:element name="departement" type="xsd:string" />
               <xsd:element name="code_postal" type="xsd:decimal" />
          </xsd:sequence>
           <xsd:attribute name= "pays" type="xsd:NMTOKEN" use="fixed" value="FR"/>
</xsd:complexType>
</xsd:schema>
```

LES COMPOSANTS PRIMAIRES

- Un schéma XML est construit par assemblage de différents composants (13 sortes de composants rassemblés en différentes catégories).
  - **Composants de définition de types**
    - Définition de types simples (Simple type).
    - Définition de types complexes (Complex type).
  - **Composants de déclaration**
    - Déclaration d'éléments.
    - Déclaration d'attributs.

LESTYPES

- Les schémas sont basés sur la notion de type
  - Chaque élément et chaque attribut possède un type.
  - Approche objet: les **types de base** et types défini **par dérivation.**
- Deux grandes catégories de types :
  1. **Types simples** (simpleType) :
     - Chaînes de caractères, valeurs numériques, etc.
     - Un élément d'un type simple ne peut ni contenir d'autres éléments ni avoir des attributs.
     - Les attributs ont des types simples.
  2. **Types complexes** (complexType) :
     - tout le reste, en particulier les éléments contenant d'autres éléments et/ou des attributs.

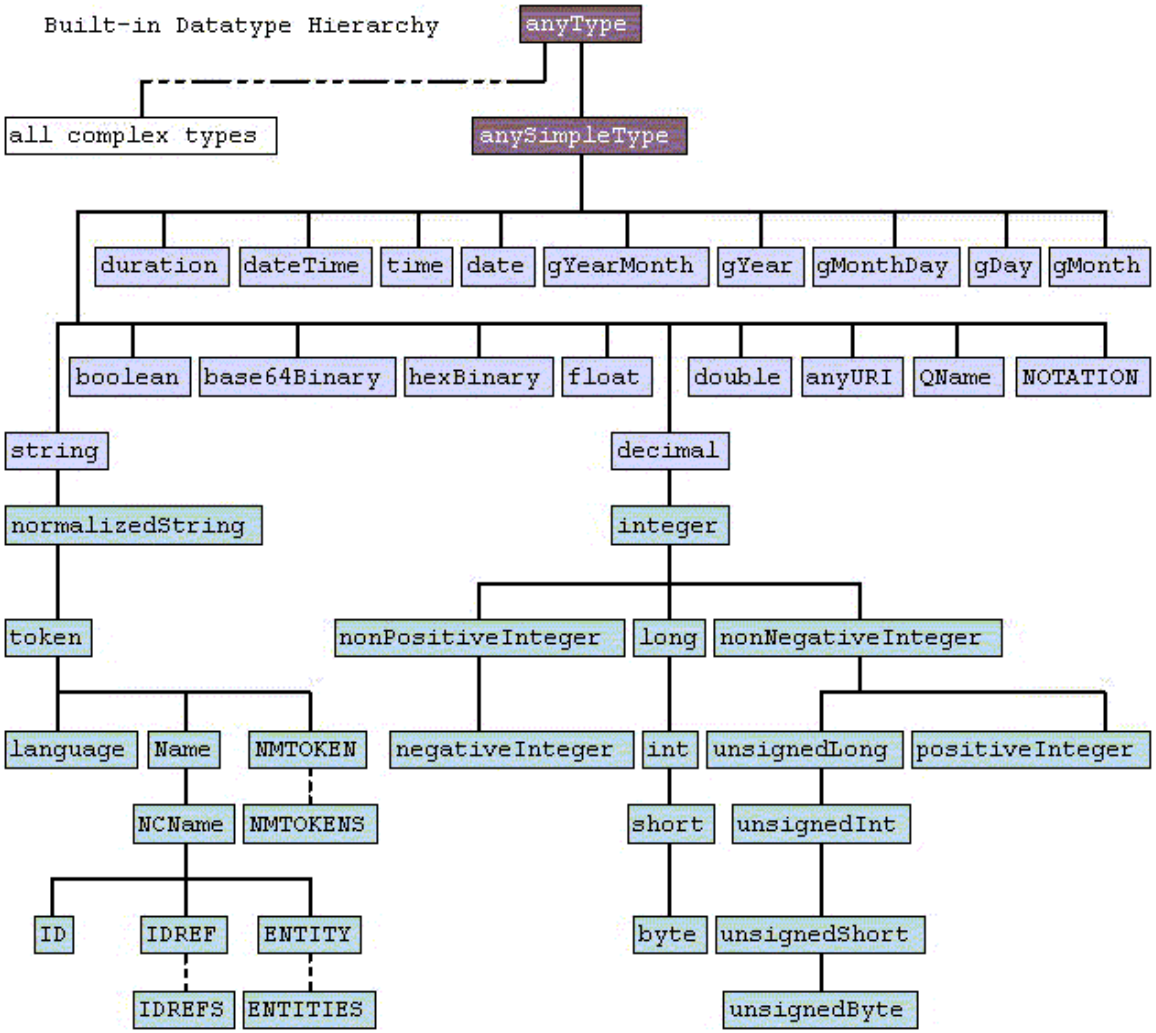
LESTYPES SIMPLES PRÉDÉFINIS

- Types simples **prédéfinis** au sens de la norme XML Schémas **'datatypes'**: string,integer,boolean …
- Exemple :

```
<xsd:element name="code_postal " type="xsd:integer"/>
```

QUELQUES TYPES PRÉDÉFINIS

Type	Forme lexicale
String	Bonjour
boolean	{true,false, 1, 0}
float	2345E3
double	23.456789E3
decimal	808.1
dateTime	1999-05-31T13:20:00-05:00.
binary	0100
uriReference	http://www.cnam.fr
….	



https://www.w3.org/TR/xmlschema-2/#dt-anySimpleType

RÉFLEXION

- Proposez un document XML valide par rapport à ce schéma:

```
<xs:element name="login" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="isAdmin" type="xs:boolean"/>
```

DÉRIVATION DESTYPES SIMPLES

- On distingue 4 types de dérivations des types simples
  - Dérivation par restriction
  - Dérivation par union
  - Dérivation par liste
  - Dérivation par extension

DÉRIVATION PAR RESTRICTION

- La dérivation par restriction restreint l'ensemble des valeurs d'un type préexistant.
- La restriction est définie par des contraintes de facettes du type de base: valeur min, valeur max ...
- Exemple:

```
<xsd:simpleType name= "ChiffresOctaux">
      <xsd:restriction base="xsd:integer">
            <xsd:minInclusive value="0" />
            <xsd:maxInclusive value= 7" />
      </xsd:restriction>
</xsd:simpleType>
```

DÉRIVATION PAR RESTRICTION (2)

Les contraintes de facettes sont:

- **length** : la longueur d'une donnée.
- **minLenght**: la longueur minimum.
- **maxLenght**: la longueur maximum.
- **pattern**: défini par une expression régulière.
- **enumeration**: un ensemble discret de valeurs.
- **maxInclusive**: une valeur max comprise.
- **maxExclusive**: une valeur max exclue.
- **minInclusive**: une valeur min comprise.
- **minExclusive**: une valeur min exclue.
- **totalDigits**: le nombre total de chiffres.
- **fractionDigits**: le nombre de chiffres dans la partie fractionnaire.

EXEMPLE POUR UNE ÉNUMÉRATION

```
<xsd:simpleType name= "Mois">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value= "Janvier"/>
        <xsd:enumeration value="Février"/>
        <xsd:enumeration value="Mars"/>
        <!-- ...-->
    </xsd:restriction>
</xsd:simpleType>
```

DÉRIVATION PAR UNION

- Pour créer un nouveau type on effectue l'union ensembliste de toutes les valeurs possibles de différents types existants.
- Exemple:

```
<xsd:simpleType name="TransportFormatCaracteres">
    <xsd:union memberTypes="base64Binary hexBinary"/>
</xsd:simpleType>
```

DÉRIVATION PAR LISTE

- Une liste permet de définir un nouveau type de sorte qu'une valeur du nouveau type est une liste de valeurs du type préexistant (valeurs séparées par espace).
- Exemple:

```
<simpleType name= "DebitsPossibles">
      <list itemType='nonNegativeInteger'/>
</simpleType>
<xsd:element name="debitsmodemV90"   type= " DebitsPossibles"/>
```

```
<debitsmodemV90>
            33600 56000
<debitsmodemV90>
```

DÉCLARATION DESATTRIBUTS

- Un attribut est déclaré par la balise **attribute**
- Un attribut est une valeur, nommée et typée, associée à un élément.
- Le type d'un attribut défini en XML schéma est obligatoirement simple.

```
<xsd:complexType name="TypeRapport">
        <xsd:attribute name= "Date_creation" type="xsd:date"/>
        ……
        </xsd:complexType>
<xsd:element name="Rapport" type="TypeRapport"/>
```

DÉCLARATION DESATTRIBUTS (2)

- L'élément attribute de XML Schema peut avoir deux attributs optionnels : **use** et **value**.
- On peut ainsi définir des contraintes de présence et de valeur.
- Selon ces deux attributs, la valeur peut:
  - être obligatoire ou non
  - être définie ou non par défaut.
- Exemple:

```
<xsd:attribute name= "Date_peremption" type="xsd:date"
                use="default" value= "2005-12-31"/>
```

DÉCLARATION DESATTRIBUTS (3)

- Valeurs possibles pour **use**
  - **use = required** :L'attribut doit apparaître et prendre la valeur fixée si elle est définie.
  - **use= prohibited** :L'attribut ne doit pas apparaître.
  - **use = optional** :L'attribut peut apparaître et prendre une valeur quelconque.
  - **use= default** :Si l'attribut à une valeur définie il la prend sinon il prend la valeur par défaut.
  - **use= fixed** :La valeur de l'attribut est obligatoirement la valeur définie.
- Exemple:

```
<xsd:attribute name= "Date_creation" type="xsd:date" use="required"/>
```

TYPES COMPLEXES

- Déclarés au moyen de l'élément **<xsd:complexType name="...**
- Ils peuvent contenir d'autres éléments,des attributs.
- Exemple:

```
<xsd:complexType name= "TypePrix">
        <xsd:simpleContent>
                <xsd:extension base="DeuxDecimales">
                        <xsd:attribute name="Unite" type= "FrancEuro" />
                </xsd:extension>
        </xsd:simpleContent>
</xsd:complexType>

<xsd:element name="prix" type="TypePrix" />
```

TYPES COMPLEXES : SEQUENCE

- Un type **sequence** est défini par une suite de sous éléments qui doivent être présents dans l'ordre donné.
- Le nombre d'occurences de chaque sous-élément est défini par les attributs minOccurs et maxOccurs.
- Exemple:

```
<xsd:complexType name= "TypeCommande">
      <xsd:sequence>
            <xsd:element name= "Ad_livraison" type="Addresse"/>
            <xsd:element name= "Ad_facturation" type="Addresse"/>
            <xsd:element name= "texte" type="xsd:string" minOccurs="1" />
            <xsd:element name="items" type="Items" maxOccurs= "30" />
      </xsd:sequence>
</xsd:complexType>
```

RÉFLEXION

- Proposer un XML valide par rapport à ce schéma

```
<xs:complexType name="user">
        <xs:sequence>
                <xs:element name="firstname" type="xs:string"/>
                <xs:element name="lastname" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="login"/>
</xs:complexType>

<xs:element name="teacher" type="user"/>
<xs:element name="student" type="user"/>
```

TYPES COMPLEXES : CHOICE

- Un seul des éléments listés doit être présent.
- Le nombre d'occurrences possible est déterminé par les attributs minOccurs et maxOccurs de l'élément.
- Exemple:

```
<xsd:complexType name= "type_temps">
      <xsd:choice >
            <xsd:element name= "Noire" type="Note" minOccurs="1" maxOccurs="1" />
            <xsd:element name= "Croche" type="Note" minOccurs="2" maxOccurs="2" />
      </xsd:choice>
</xsd:complexType>
```

TYPES COMPLEXES : ALL

- Les éléments listés doivent être tous présents au plus une fois.
- Il peuvent apparaître dans n'importe quel ordre.
- Exemple:

```
<xsd:complexType name= "Commande">
      <xsd:all>
            <xsd:element name= "Ad_livraison" type="Addresse"/>
            <xsd:element name= "Ad_facturation" type="Addresse"/>
            <xsd:element name= "texte" type="xsd:string" minOccurs="0" />
            <xsd:element name="items" type="Items" maxOccurs= "30" />
      </xsd:all>
</xsd:complexType>
```

DÉRIVATION PAR EXTENSION

- Dériver un nouveau type par **extension** consiste à ajouter à un type existant des sous-éléments ou des attributs.
- On obtient inévitablement un **type complexe**. Exemple:

```
<xsd:complexType name= "mesure">
        <xsd:simpleContent>
            <xsd:extension base="xsd:Decimal">
                <xsd:attribute name="unite" type="xsd:NMTOKEN"/>
            </xsd:extension>
        </xsd:simpleContent>
</xsd:complexType>
<xsd:element name= "temperature" type= "mesure"/>
```

DÉCLARATION DES ÉLÉMENTS

- Un élément XML est déclaré par la balise **element** de XML schéma qui a de nombreux attributs.
- Les deux principaux attributs sont:
  - **name** : Le nom de l'élément (de la balise associée).
  - **type** : Le type qui peut être simple ou complexe.
- Exemple:

```
<xsd:element name="code_postal" type="xsd:decimal"/>
```

ASSOCIER UN XSD À UN XML

```
<?xml version="1.0" encoding="UTF-8"?>

<racine
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="chemin_vers_fichier.xsd">

</racine>
```

LES SERVICES WEB ÉTENDUES

DÉFIS DANS UNE ARCHITECTURE ORIENTÉE SERVICES → DES STANDARDS

Invoquer le service

- Comment permettre au client d'invoquer un service distant mise à côté la technologie de la manière la plus transparente

WSDL

Echange de données

- Selon quel format et quelle structure de données les données sont échangées?

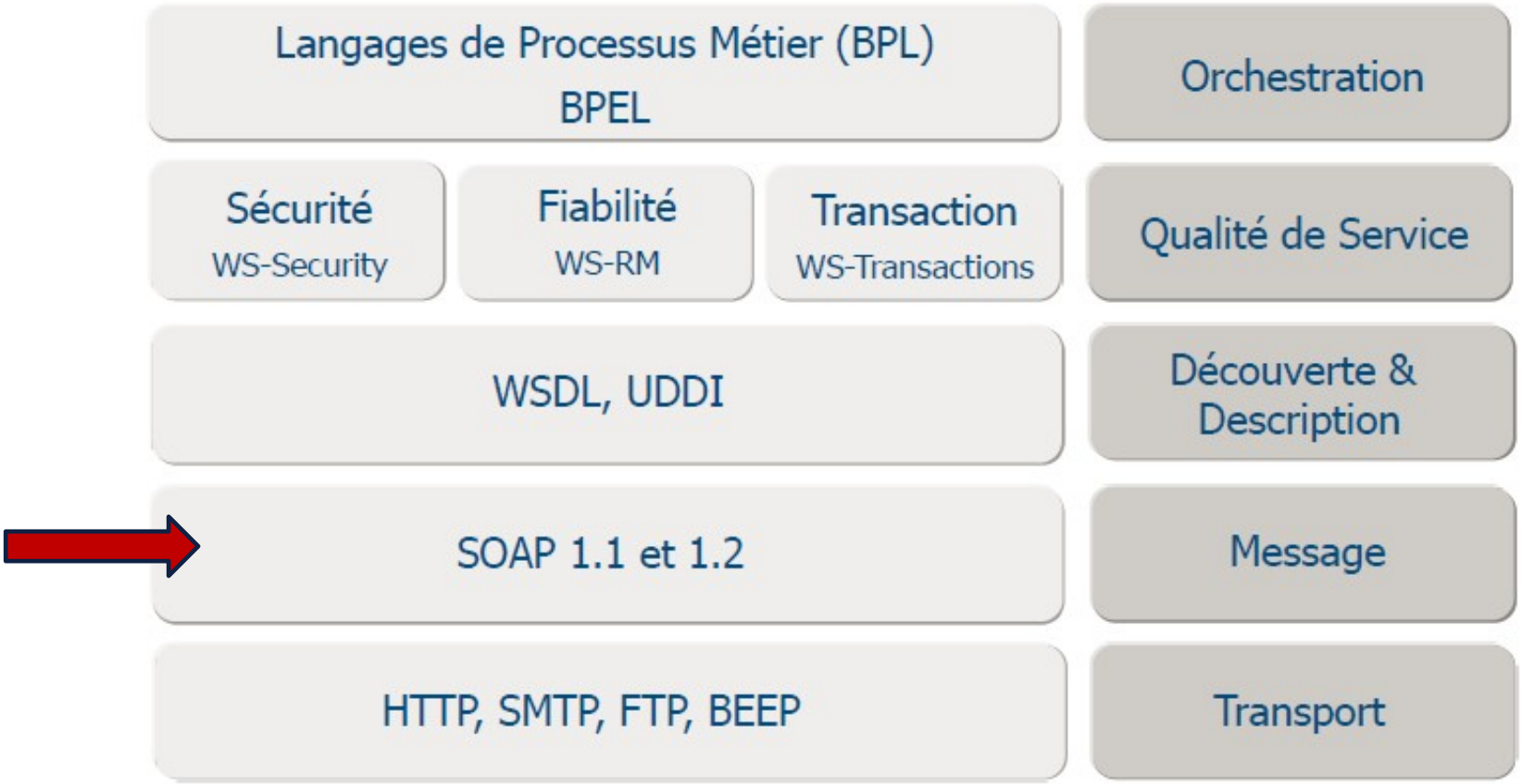
SOAP

Localiser le service

- Comment localiser le service adéquat parmi une large collection de services disponibles

UDDI

A PROPOS DU PROTOCOLE SOAP



PRÉSENTATION

- **SOAP** (Simple Object Access Protocol) est devenu le protocole standard pour décrire les messages en XML pour les services web.
- SOAP Peut être utilisé sur différents protocoles de transports.
  - Principalement **HTTP** et **SMTP**.
- Il Permet **l'interopérabilité** entre différents systèmes d'exploitation et différentes plate-formes (J2EE,.NET,…).

PRÉSENTATION

- Un message **SOAP** est un document XML composé d'une enveloppe qui contient une entête et le corps du message.

EXEMPLE D'UN MESSAGE SOAP

Exemple: un message SOAP appelant une méthode «echoHello », qui prend en argument une chaîne de Caractères à afficher.

La partie « **Header** » contient des informations non liées à la méthode (exemple: ID de la transaction) ou des données de sécurité.
Elle est facultative.

Le « **Body** » du message contient toutes les informations destinées au récepteur (les paramètres par exemple) sinon l'élément « **Fault** » en cas d'erreur

```
<Envelope>
    <Header>
        <Transaction>2</Transaction>
    </Header>
    <Body>
        <echoHello>
            <arg0>Hello!</arg0>
        </echoHello>
    </Body>
</Envelope>
```

EXEMPLE D'UN MESSAGE SOAP (2)

- Lorsque le serveur répond à la méthode **echoHello**, il ajoute **Response** à la suite du tag < echoHello >
- Généralement, il rajoute **Response** à la suite du tag contenant le nom de la requête.

```
<Envelope>
        <Header>
            <Transaction>2</Transaction>
        </Header>
        <Body>
            <echoHelloResponse>
                    <return>Hello!</return>
            </ echoHelloResponse >
        </Body>
</Envelope>
```

- En cas d'erreur, le « Body » contient l'élément Fault.

SOAP ET LES ESPACES DE NOM

- Les messages précédents présentaient SOAP sans l'utilisation des espaces de noms, obligatoires d'après les spécifications du protocole.
  - xsi correspond au namespace des types de données connus ;
  - xsd correspond au namespace du schema du document ;
  - SOAP-ENV correspond au namespace de l'enveloppe;
  - SOAP-ENC correspond au namespace de l'encodage des données;
  - SOAP-RPC correspond à la représentation rpc.

LESTYPES DANS SOAP

- SOAP permet l'interopérabilité entre différentes plate-formes. Il est donc important d'avoir des règles de codage des types de données, afin que ces dernières puissent être encodées/décodées sans difficultés.
- On distingue deux types de données:
  - Les données de types simples (entier, flottant, chaîne de caractère, énumération) ;
  - Les types composés : structures ou tableaux.
- Il faut donc définir ces types en utilisant des Schéma XML

TYPES SIMPLES : EXEMPLES

- Exemple1

```
<xsd:element name="age" type="xsd:int"\>
<xsd:element name="taille" type="xsd:float"\>

<age>23</age>
<taille>1.87</taille>
```

- Exemple2

```
<xsd:simpleType name="TypeCouleur">
        <xsd:restriction base="xsd:string">
                <enumeration value="Rouge">
                <enumeration value="Bleu">
        <xsd:restriction>
</xsd:simpleType>
<xsd:element name="couleur" type="TypeCouleur"\>

<couleur>Bleu</couleur>
```

TYPES COMPLEXES: EXEMPLE 1

```
<xsd:complexType name="TypePersonne">
    <xsd:sequence>
        <xsd:element name="Nom" type="xsd:string"/>
        <xsd:element name="Prenom" type="xsd:string"/>
        <xsd:element name="Age" type="xsd:float"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="Personne" type="TypePersonne">
```

```
<Personne>
    <Nom>Durand</Nom>
    <Prenom>Michel</Prenom>
    <Age>34.7</Age>
</Personne>
```

SQUELETTE D'UN MESSAGE SOAP

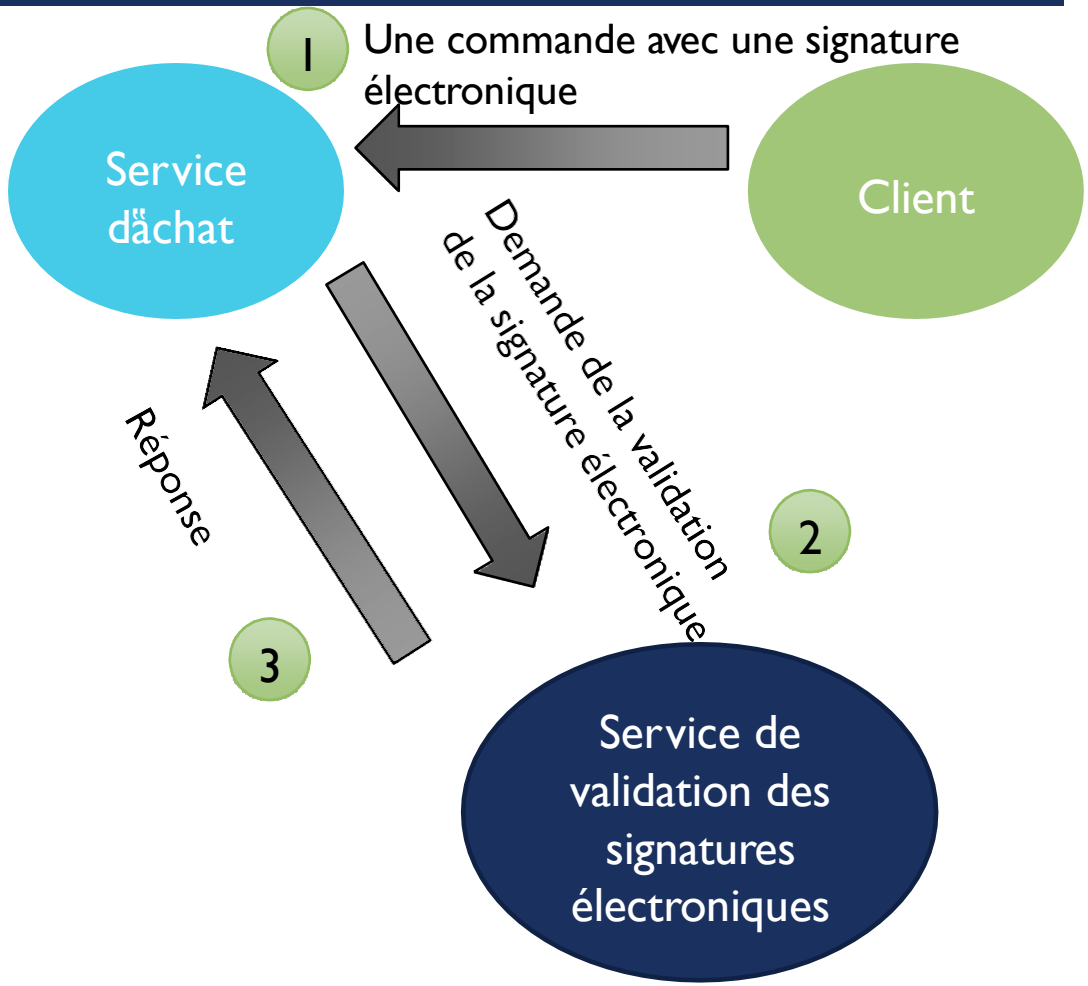
```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    <soapenv:Header>
      ...
    </soapenv:Header>
    <soapenv:Body>
      ...
        <soapenv:Fault>
        ...
        </soapenv:Fault>
    </soapenv:Body>
</SOAP-ENV:Envelope>
```

SOAP HEADER

- **SOAP Header** : Mécanisme d'extension du protocole SOAP
  - La balise Header est optionnelle
  - Si la balise Header est présente, elle doit être **le premier fils** de la balise Envelope
  - La balise Header contient des **entrées**
    - Une **entrée** est n'importe quelle balise incluse dans un namespace.
  - Les entrées contenues dans la balise Header sont **non applicatives**.

EXEMPLE D'USAGE DU HEADER

- Le Soap Header est conçue pour la participation des nœuds intermédiaires lors de l'invocation des services web.
- La réception du message SOAP est finalement pour le nœud « **ultimate-receiver** »



SOAP BODY

- **SOAP Body** : Le Body contient le message à échanger
  - La balise Body est obligatoire
  - La balise Body doit être le premier fils de la balise Envelope (ou le deuxième si existe une balise Header)
  - La balise Body contient des entrées qui sont des **données applicatives.**
  - Une entrée est n'importe quelle balise incluse optionnellement dans un namespace
  - Une entrée peut être une Fault.

SOAP FAULT

- **SOAP Fault** : Balise permettant de signaler des cas d'erreur. **Fault** contient les balises suivantes:
  - **Faultcode** (Obligatoire): un code permettant d'identifier le type d'erreur.
  - On distingue 4 groupes de code d'erreur :
    - **Client** : Le message n'a pas été correctement formé ou il manque certaines informations
    - **Server** : Serveur non accessible ou erreur de décodage du message
    - **MustUnderstand** : L'élément de l'entête n'a pas été compris
    - **VersionMismatch** : Le namespace donné ne permet pas de valider le message
  - **Faultstring** (Obligatoire): une explication en langage naturel.
  - **Faultactor**: une information identifiant l'initiateur de l'erreur.
  - **Detail**: Définition précise de l'erreur.

SOAP FAULT: EXEMPLE

```
<s:Envelope
xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
    <s:Body>
        <s:Fault>
            <faultcode xmlns="">s:Client</faultcode>
            <faultstring xml:lang="fr-FR" xmlns="">
                Une opération invalide s'est produite.
            </faultstring>
        </s:Fault>
    </s:Body>
</s:Envelope>
```

SOAP : TESTER UN SERVICEWEB EXISTANT

- Installer SOAPUI (https://www.soapui.org/downloads/soapui.html).
- Aller sur le site : http://xstandard.com/en/documentation/xstandard-oem/web-services/spell-checker/
- Choisir le service « **Spell Checker** »
  - Que fait ce service ?
  - Combien d'opérations sont gérés par le service web.
  - Tester le service web en utilisant SOAP UI.
  - Interpréter les messages échangés

DÉFIS DANS UNE ARCHITECTURE ORIENTÉE SERVICES → DES STANDARDS

Invoquer le service

- Comment permettre au client d'invoquer un service distant mise à côté la technologie de la manière la plus transparente

WSDL

Echange de données

- Selon quel format et quelle structure de données les données sont échangées?

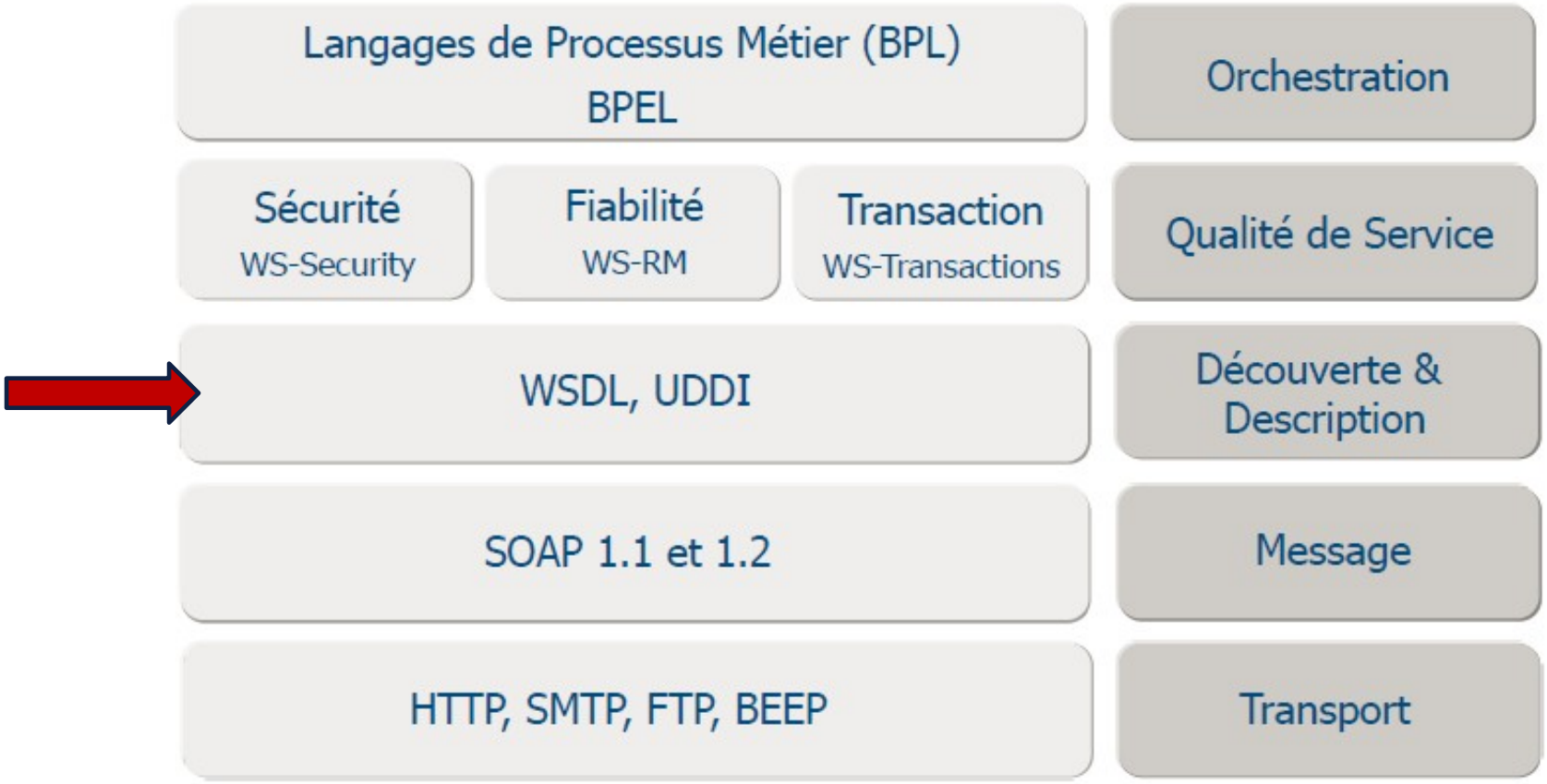
SOAP

Localiser le service

- Comment localiser le service adéquat parmi une large collection de services disponibles

UDDI

A PROPOS DU PROTOCOLE SOAP



PRÉSENTATION

- Spécification (09/2000)
  - Ariba, IBM, Microsoft
  - W3C v1.1 (25/03/2001)
- Objectifs
  - Décrire les services comme un ensemble d'opérations et de messages abstraits relié (bind) à des protocoles réseaux
- Grammaire XML (schema XML)
  - Modulaire (importer d'autres documents WSDL et XSD)
- Séparation entre la **partie abstraite** et **concrète** (implémentation)

PRÉSENTATION (2)

- **WSDL** (Web Service Description Langage), est un langage de description de services web en XML.
- Il décrit :
  - Les informations sur les fonctions publiques du service web ;
  - Les types de données utilisés durant l'échange de messages ;
  - Les différents protocoles aux travers desquels le service est accessible ; et comment y accéder ;
  - Une adresse permettant de localiser le service web.

STRUCTURE D'UNE DESCRIPTIONWSDL

```
<definitions>
    <types>définition des types</types>
    <message>définition des messages</message>
    <portType>définition des interfaces </portType>
    <binding>définition des bindings</binding>
    <services>définition de endpoint</service>
</definitions>
```

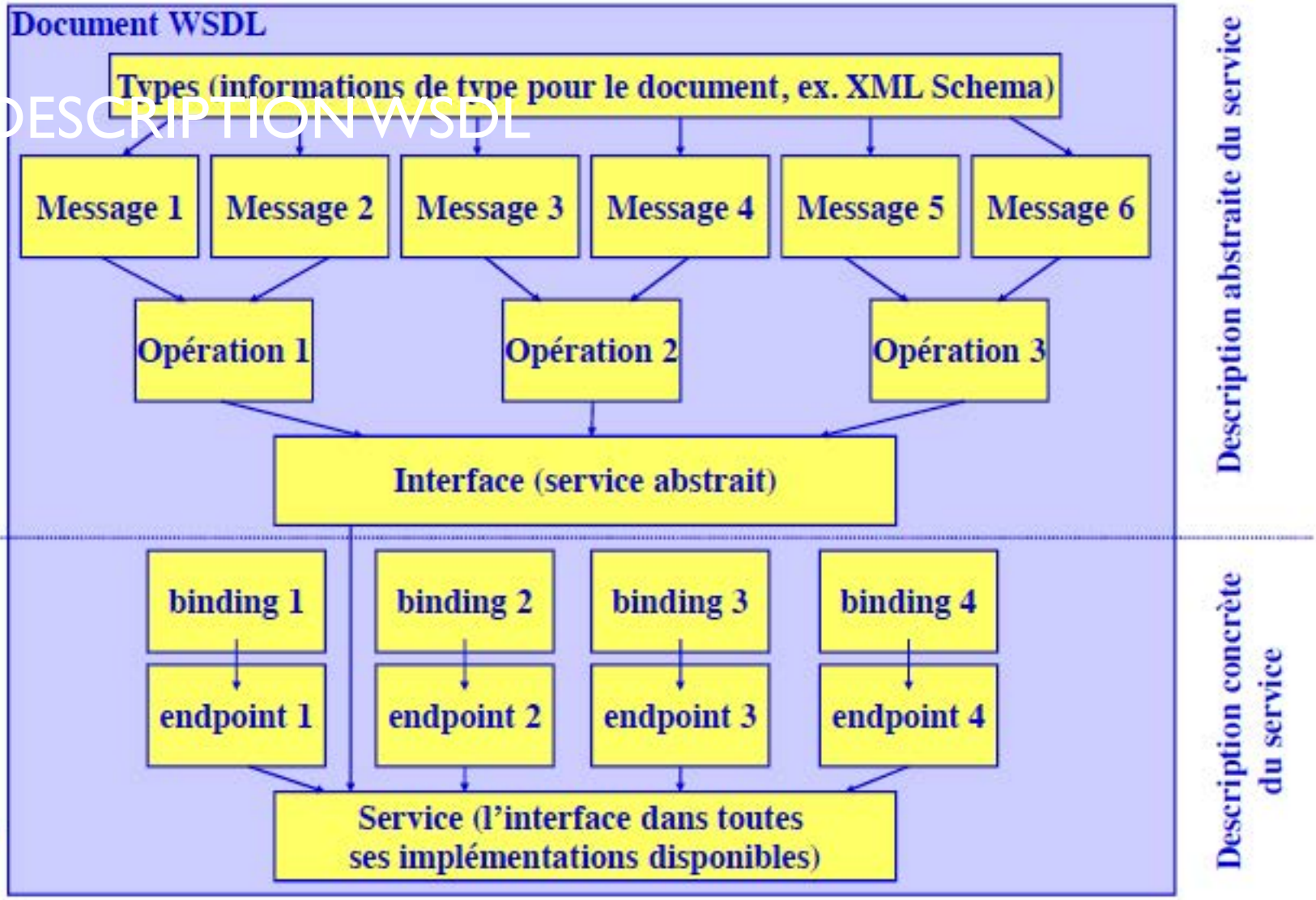
ÉLÉMENTS D'UN DOCUMENT WSDL (2)

- <types>
  - Contient les définitions de types en utilisant un système de typage (comme XSD).
- <message>
  - Décrit les noms et types d'un ensemble de champs à transmettre
    - Paramètres d'une invocation, valeur du retour, …
- <portType>
  - Décrit un ensemble d'opérations.
  - Chaque opération a zéro ou un message en entrée, zéro ou plusieurs messages de sortie ou de fautes

ÉLÉMENTS D'UN DOCUMENT WSDL (3)

- <binding>
  - Spécifie une liaison d'un <porttype> à un protocole concret (HTTP 1.1, SMTP, MIME, …).
  - Un <porttype> peut avoir plusieurs liaisons !
- <port>
  - Spécifie un point d'entrée (endpoint) comme la combinaison d'un <binding> et d'une adresse réseau.
- <service>
  - Une collection de points d'entrée (endpoint) relatifs.

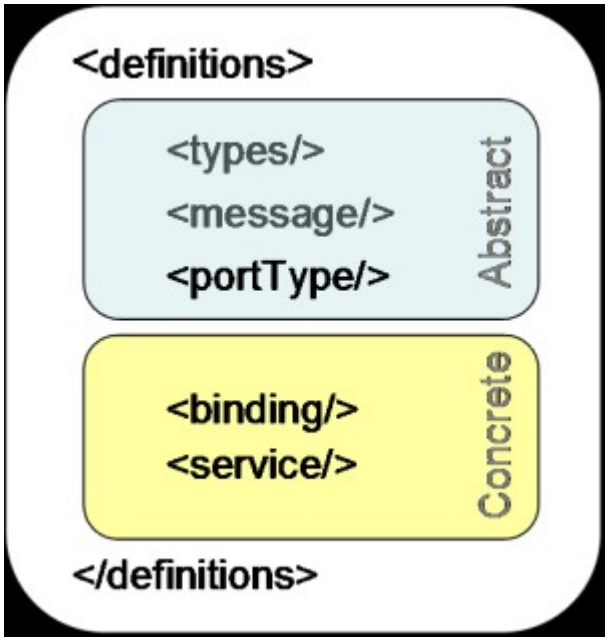
DESCRIPTION WSDL



Source: G. Alonso

STRUCTURE D'UN DOCUMENT WSDL

- Séparation entre la partie **abstraite** et **concrète**
- La partie **abstraite** : décrit les messages et les opérations. Cette partie est composée des éléments :
  - <types>
  - <message>
  - <portType>
- La partie **concrète** : décrit le protocole et le type d'encodage à utiliser. Cette partie est composée des éléments :
  - <binding>
  - <service>



PARTIE ABSTRAITE : ÉLÉMENT TYPES

Contient les définition de types utilisant un système de typage (comme XSD).

```
<types>
  <xsd:schema targetNamespace="urn:xml-soap-address-demo"
                    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <xsd:complexType name="phone">
      <xsd:element name="areaCode" type="xsd:int"/>
      <xsd:element name="exchange" type="xsd:string"/>
      <xsd:element name="number" type="xsd:string"/>
    </xsd:complexType>
    <xsd:complexType name="address">
      <xsd:element name="streetNum" type="xsd:int"/>
      <xsd:element name="streetName" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
      <xsd:element name="state" type="xsd:string"/>
      <xsd:element name="zip" type="xsd:int"/>
      <xsd:element name="phoneNumber" type="typens:phone"/>
    </xsd:complexType>
  </xsd:schema>
</types>
```

PARTIE ABSTRAITE : ÉLÉMENT **MESSAGE**

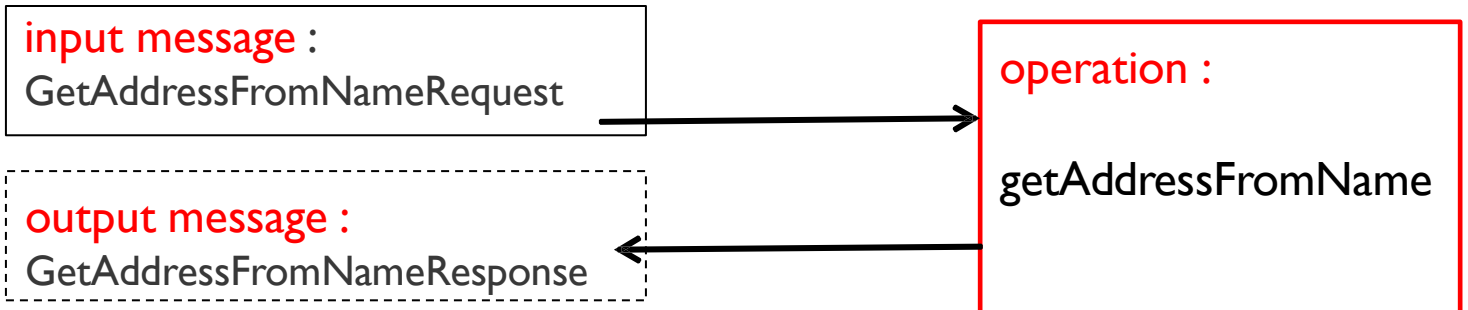
- Décrit un message unique : message **requête** ou **réponse**
- Contient une ou plusieurs balises **<part>** pour définir les **paramètres** d'entrée et de sortie d'une **opération**

```
<message name="AddEntryRequest">
   <part name="name" type="xsd:string"/>
   <part name="address" type="typens:address"/>
</message>

<message name="GetAddressFromNameRequest">
   <part name="name" type="xsd:string"/>
</message>

<message name="GetAddressFromNameResponse">
   <part name="address" type="typens:address"/>
</message>
```

PARTIE ABSTRAITE : ÉLÉMENT OPERATIONS

- <operations> : Une opération est comparable à une méthode en Java
  - Est identifiée par un nom
  - Contient ou non une ou plusieurs entrée(s)
  - Contient ou non une ou sortie
- Exemple

```
<operation name="getAddressFromName">
  <input message="GetAddressFromNameRequest"/>
  <output message="GetAddressFromNameResponse"/>
</operation>
```



PARTIE ABSTRAITE : ÉLÉMENT **PORTTYPE**

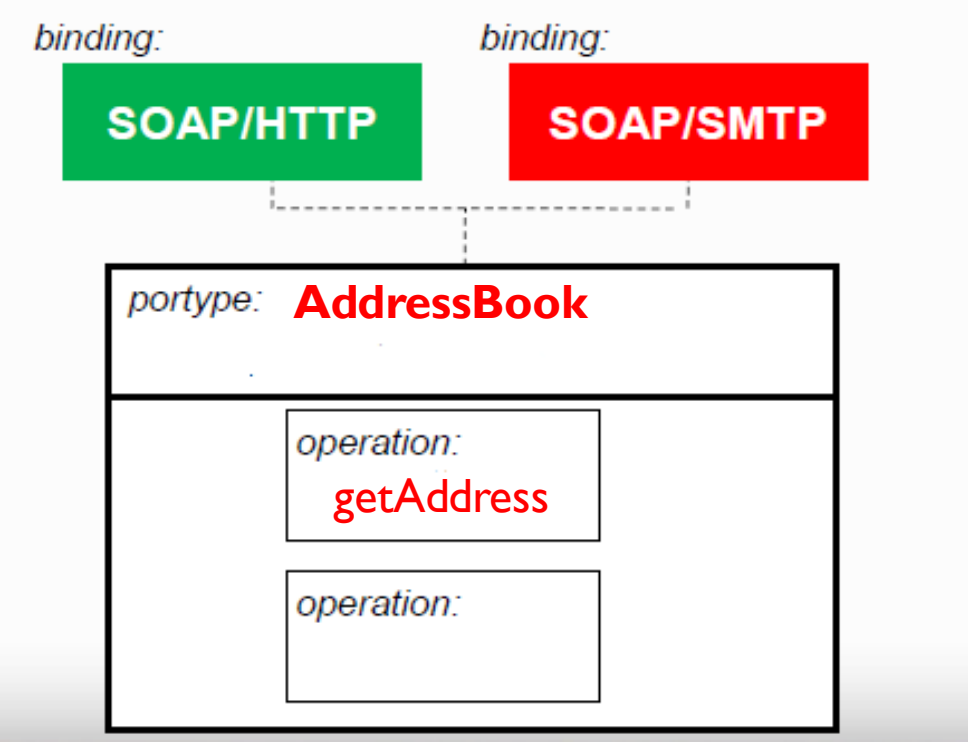
- Décrit l'ensemble des **opérations** fournies. On distingue Plusieurs types d'opérations:
  - **One-way**
    - Le service reçoit un message (<input>).
  - **Request-response**
    - Le service reçoit un message (<input>) et retourne un message corrélé (<output>) ou des éventuels messages de faute (<fault>).
  - **Solicit-response**
    - Le service envoie un message (<output>) et reçoit un message corrélé (<input>) ou un ou plusieurs messages de faute (<fault>).
  - **Notification**
    - Le service envoie un message de notification (<output>)
- Paramètres
  - Les champs des messages constituent les paramètres (in, out, inout) des opérations

ÉLÉMENT **PORTTYPE** :EXEMPLE

```
<portType name="AddressBook">

   <!– One way operation -->
   <operation name="addEntry">
     <input message="AddEntryRequest"/>
   </operation>

   <!– Request-Response operation -->
   <operation name="getAddressFromName">
      <input message="GetAddressFromNameRequest"/>
      <output message="GetAddressFromNameResponse"/>
   </operation>

</portType>
```

PARTIE CONCRÈTE: ELÉMENT <BINDING>

- Spécifie une liaison d'un <porttype> à un protocole concret (SOAP, HTTP).
- Un <porttype> peut avoir plusieurs liaisons



EXEMPLE: BINDING

Exemple1: Binding SOAP et HTTP

```
<!-- binding declns -->
<wsdl:binding name="AddressBookBinding1" type="AddressBook">
 <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
 <wsdl:operation name="getAddress">
 ……..
</wsdl:operation>
</wsdl:binding>
```

Exemple2: Binding SOAP et SMTP

```
<wsdl:binding name="AddressBookBinding2" type="AddressBook">
 <soap:binding style="document" transport="http://stockquote.com/smtp"/>
 <wsdl:operation name="getAddress">
  ……..
 </wsdl:operation>
</wsdl:binding>
```

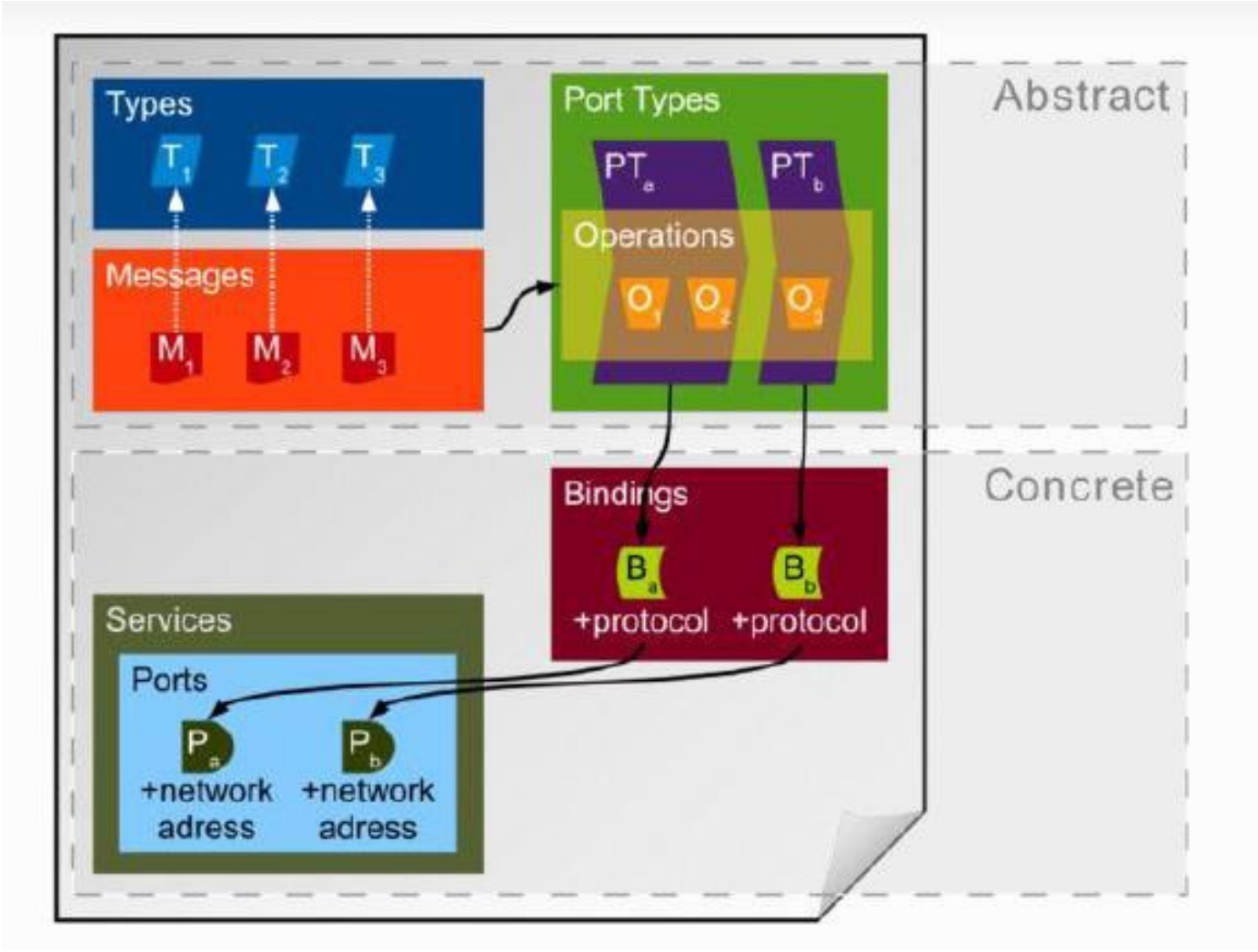
PARTIE CONCRÈTE: ÉLÉMENT **SERVICE**

- Une **collection** de points d'entrée (**endpoint**) relatifs
- <**port**> : définit un point de terminaison (adresse internet + liaison).

```
<?xml version="1.0" ?>
<definitions name="urn:AddressFetcher"
             targetNamespace="urn:AddressFetcher2"
             xmlns:typens="urn:xml-soap-address-demo"
             xmlns:xsd="http://www.w3.org/1999/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">
 ...
 <!-- service decln -->
 <service name="AddressBookService">
  <port name="AddressBook" binding="AddressBookSOAPBinding">
   <soap:address location="http://www.mycomp.com/soap/servlet/rpcrouter"/>
  </port>
 </service>
</definitions>
```

WSDL : RÉCAPITULATIF

- **Operation** : une action particulière supportée par le service web décrit. En faisant l'analogie avec Java, on peut comparer une Operation à une (méthode d'une) Interface ;
- **Message** : définition des types de données utilisées lors de l'invocation (et de la réponse) d'une opération ;
- **PortType** : un ensemble (1..n) d'opérations ;
- **Binding** : lien entre un PortType et un protocole d'accès ;
- **Port** : définit un point d'accès (c-à-d une URL par exemple) pour un binding ;
- **Service** : contient une collection de ports.


Types
T1
T2
T3
Messages
M1
M2
M3
Port Types
PTa
PTb
Operations
O1
O2
O3
Abstract
Concrete
Bindings
Ba
Bb
+protocol
+protocol
Services
Ports
Pa
Pb
+network adress
+network adress

LES SERVICES WEB « REST »

LA COUCHE DETRANSPORT : HTTP

- Le protocole HTTP (HyperTextTransfert Protocol) est l'un des protocoles les plus utilisés sur Internet.
- La plupart des clients web (IE, Firefox,..) l'utilisent pour communiquer avec un serveur.
- Il définit le format des requêtes qu'un client peut envoyer ainsi que les résultats qu'il peut attendre.
- Chaque requête contient une URL qui contient l'identifiant d'un objet demandé par le client (ex.: pages HTML, images, …).

LA COUCHE DETRANSPORT :HTTP (2)

- Exemple : un navigateur web souhaite obtenir la page par défaut du site www.google.fr

LA COUCHE DETRANSPORT :HTTP (3)

- Quand un client envoie une requête,il l'envoie/
  - une méthode (GET,POST ou HEAD),
  - suivie d'une URI (Uniform Ressource Identifier) qui identifie la ressource demandée.
  - Après cette URI se trouve la version du protocole HTTP (1.0 ou 1.1) ;
- Dans les lignes suivantes se trouvent les entêtes qui précisent par exemple quels sont les documents acceptés par client,de quel type de client il s'agit,…
- Après les entêtes se trouve le corps de la requête,rempli seulement lorsque la méthode POST est utilisée.

Les méthodes	
GET	Utilisé pour demander un document : `GET index.html` Le corps d'une telle requête est toujours vide. Permet également de passer des paramètres au serveur, en les collant à l'URL : `GET index.jsp?userLogin=toto&userPasswd=kkju` Comme résultat, GET renvoie d'abord les entêtes, puis le contenu du document
HEAD	Comme GET, mais aucune information ne se trouve dans le corps du résultat. Notamment utilisé pour voir si un document a été mis à jour.
POST	Permet au client d'envoyer des données dans le corps de la requête. Utile pour envoyer des formulaires, des documents, poster des messages dans les newsgroups... **Cette méthode est celle qui convient le mieux à SOAP**
PUT	Permet d'ajouter une ressource.
DELETE	Permet de supprimer une ressource.

Utilisées dans les architectures REST

LA COUCHE DETRANSPORT :HTTP (4)

- La réponse du serveur contient le statut de la réponse, les entêtes puis le corps de la réponse (par exemple le contenu d'un document HTML ;
- Différents statuts existent, les principaux sont :
  - 200 (ok),
  - 400 (mauvaise requête),
  - 403 (client non autorisé),
  - 404 (document inexistant),
  - 500 (erreur d'exécution sur le serveur).

HTTP : RÉCAPITULATIF

Une requête HTTP

- Commande
- Entêtes
- [LigneVide]
- Corps

Une réponse HTTP

- Status
- Entêtes des réponses
- [Ligne vide]
- Coprs de la réponse

Codes de retour HTTP: https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

INTRODUCTION

- Acronyme de « Representational State Transfert », défini dans la thèse de Roy Fielding en 2000 [1].
- Contrairement à SOAP et à HTTP, REST n'est pas un protocole ou un format, mais un style d'architecture inspiré de l'architecture du web fortement basé sur le protocole HTTP
- Il n'est pas dépendant uniquement du web et peut utiliser d'autre protocoles que HTTP.

[1] https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

« REST» : C'EST QUOI?

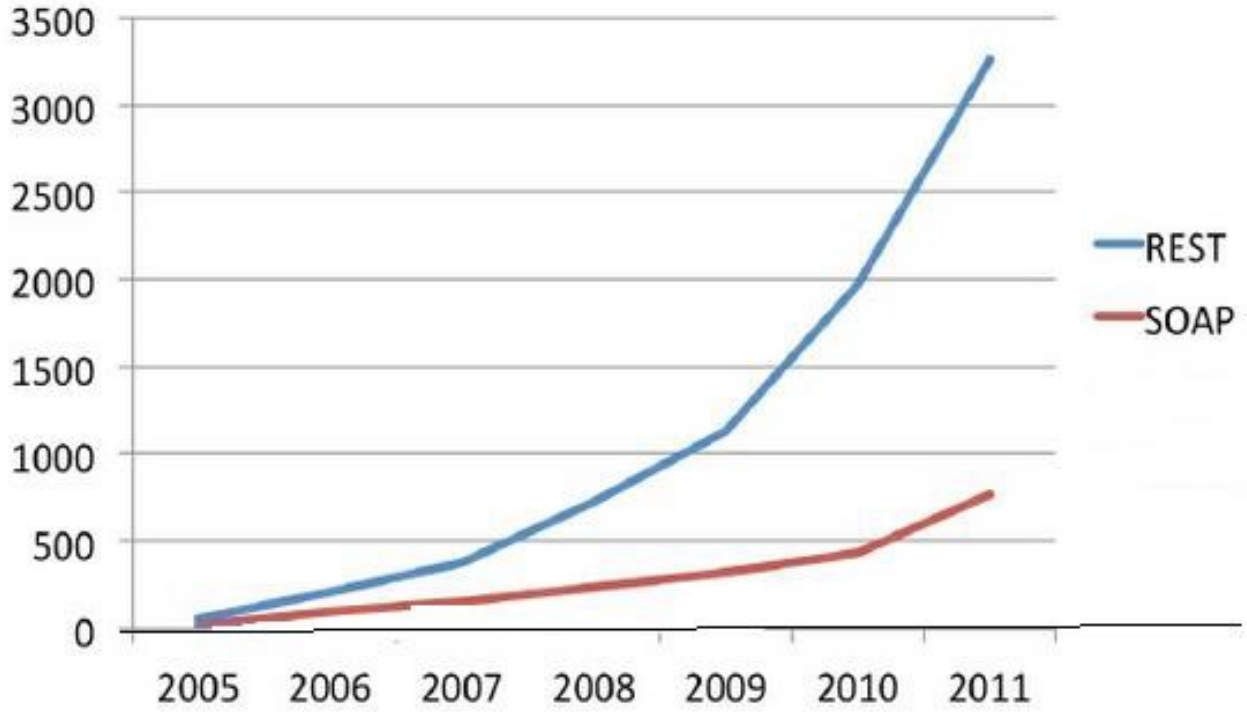
- Ce qu'il est :
  - Un système d'architecture
  - Une approche pour construire une application
- Ce qu'il n'est pas
  - Un protocole
  - Un format
  - Un standard

DES FOURNISSEURS DE SERVICESWEB « REST »

Où trouver des APIs: https://nordicapis.com/api-discovery-15-ways-to-find-apis/

STATISTIQUES

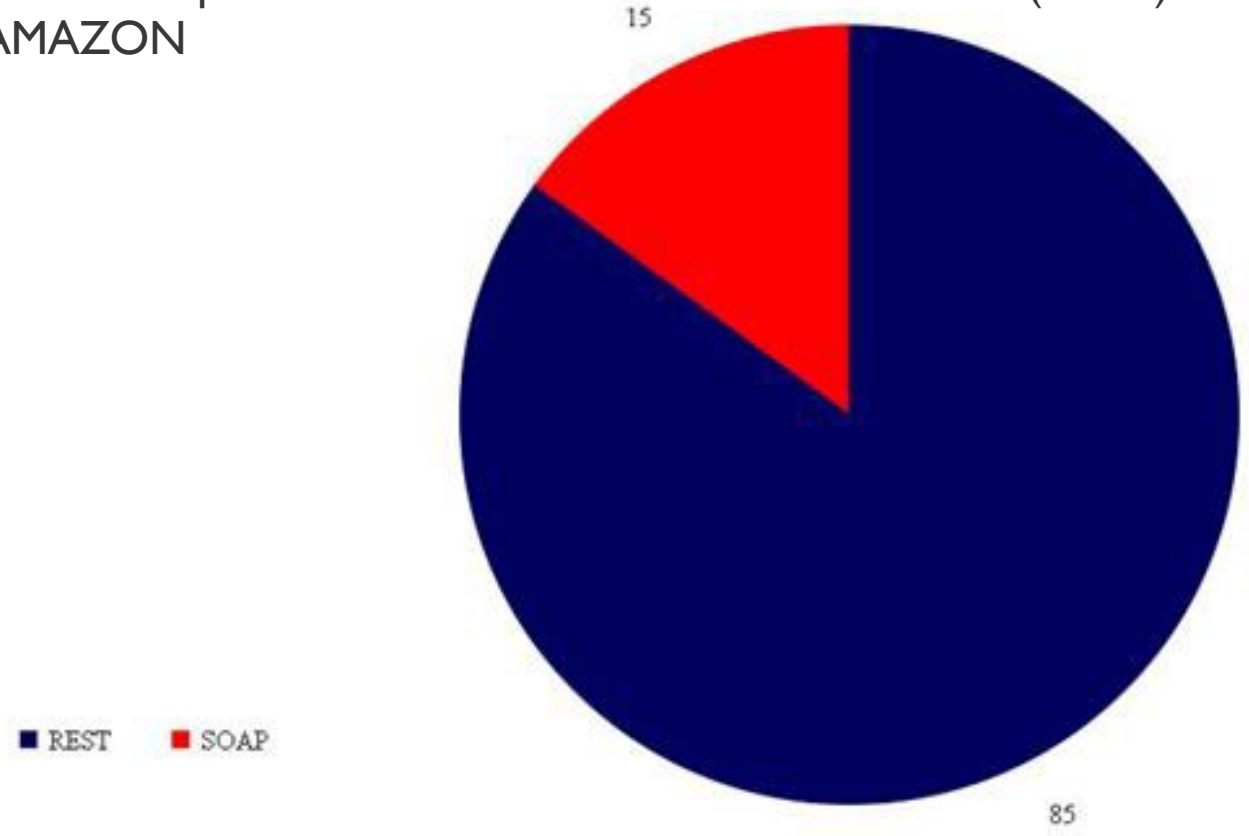
- Evolution de l'usage des services web étendus et des services REST.



Source: www.programmableweb.com

STATISTIQUES

- Les statistiques d'utilisation des services web étendus (SOAP) et REST chez AMAZON



CARACTÉRISTIQUES

- Les services REST sont sans états (**Stateless**)
  - Chaque requête envoyée au serveur doit contenir toutes les informations relatives à son état et est traitée indépendamment de toutes autres requêtes
  - Minimisation des ressources
- **Interface uniforme** basée sur les méthodes HTTP (GET, POST, PUT, DELETE)

LES REQUÊTES « REST »

- **Ressources**
  - Identifiée par une URI (Exemple:http://tek-up.tn/cursus/engineer)
- **Méthodes** (verbes) permettant de manipuler les ressources (identifiants)
  - Méthodes HTTP : GET, POST, PUT, DELETE
- **Représentation** :
  - Vue sur l'état de la ressource
  - Format d'échanges entre le client et le serveur (XML, JSON, text/plain,…)

LES REQUÊTES « REST » : RESSOURCE

- Une ressource est un objet identifiable sur le système
- Livre, Catégorie, Etudiant, Prêt
- Une ressource est identifiée par une URI : Une URI identifie uniquement une ressource sur le système

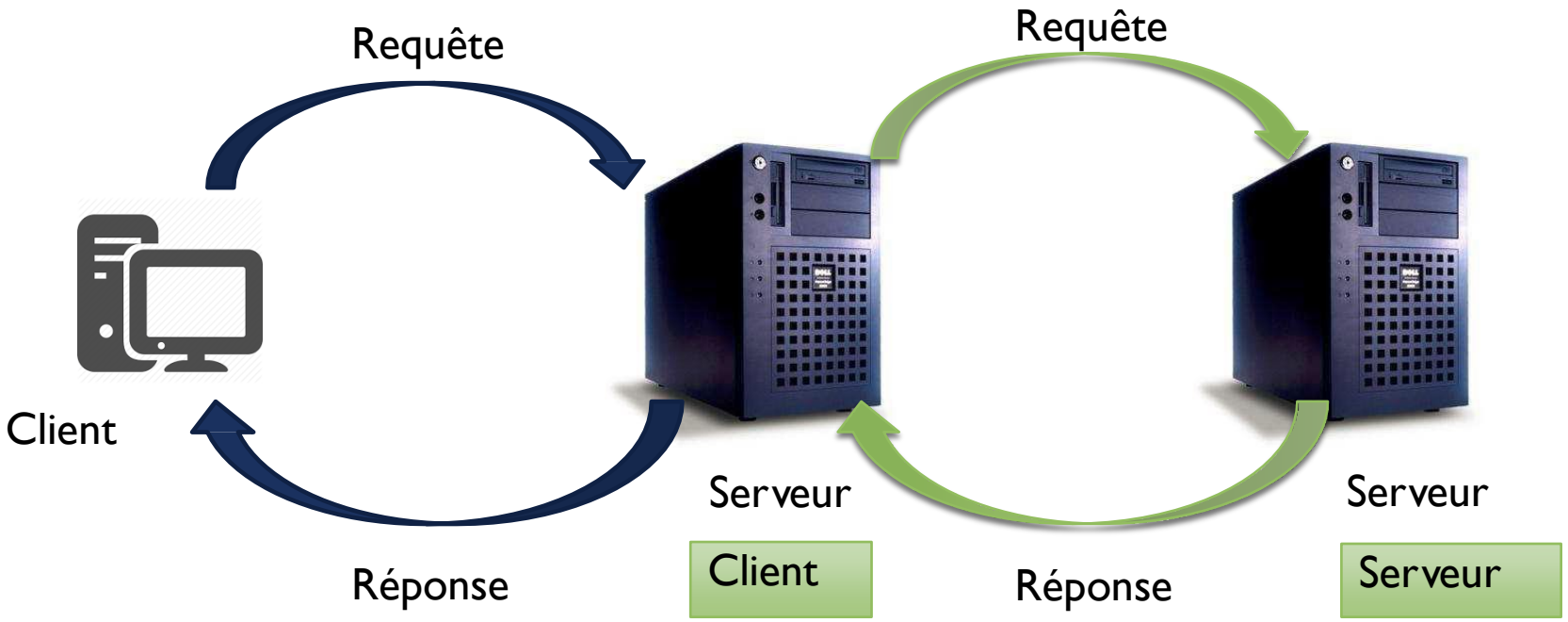
Exemple: http://books.tek-up.tn/bookstore/books/1

Identifie la clé primaire de la ressource dans la BD

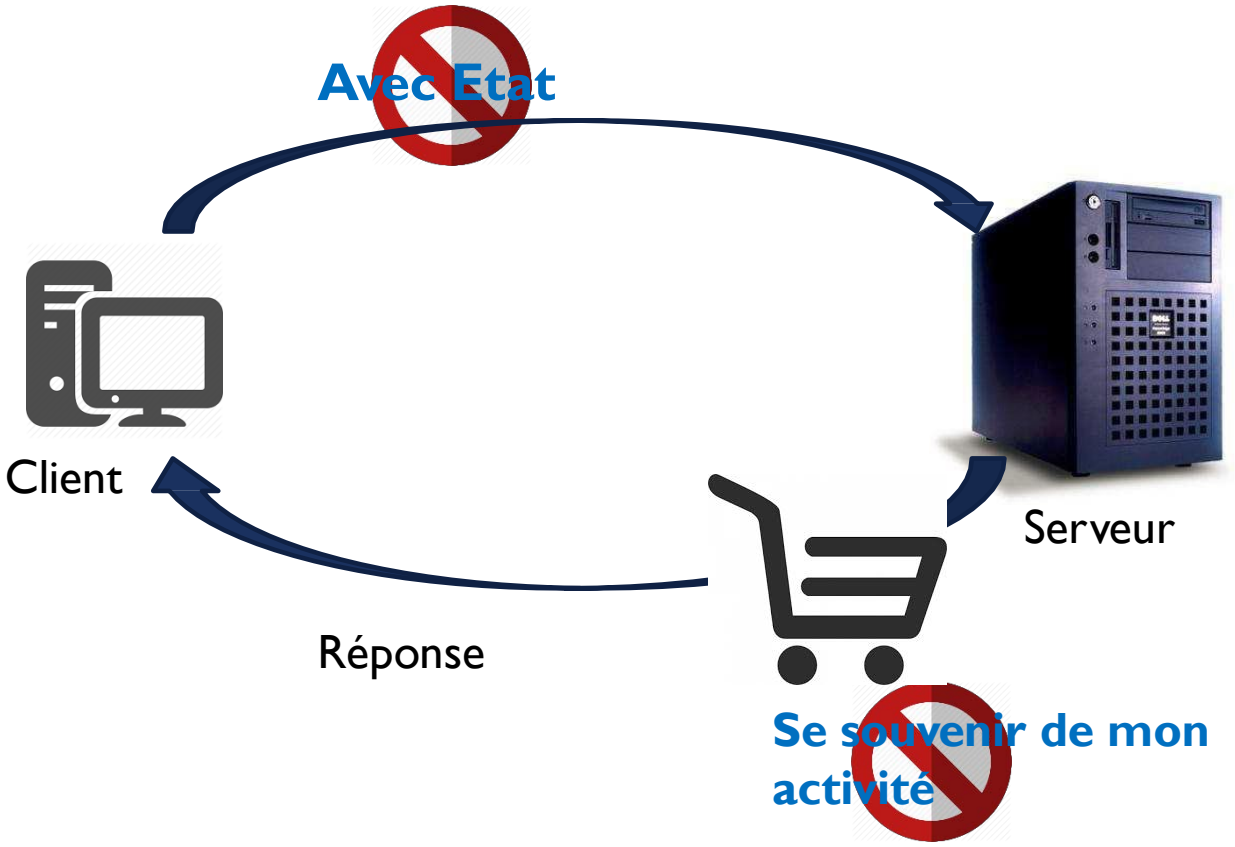
LES REQUÊTES « REST » : MÉTHODES (VERBE)

- Une ressource peut subir quatre opérations de bases **CRUD** correspondant aux quatre principaux types de requêtes HTTP (GET, PUT, POST, DELETE)
- REST s'appuie sur le protocole HTTP pour effectuer ces opérations sur les objets
  - CREATE → POST
  - RETRIEVE → GET
  - UPDATE → PUT
  - DELETE → DELETE

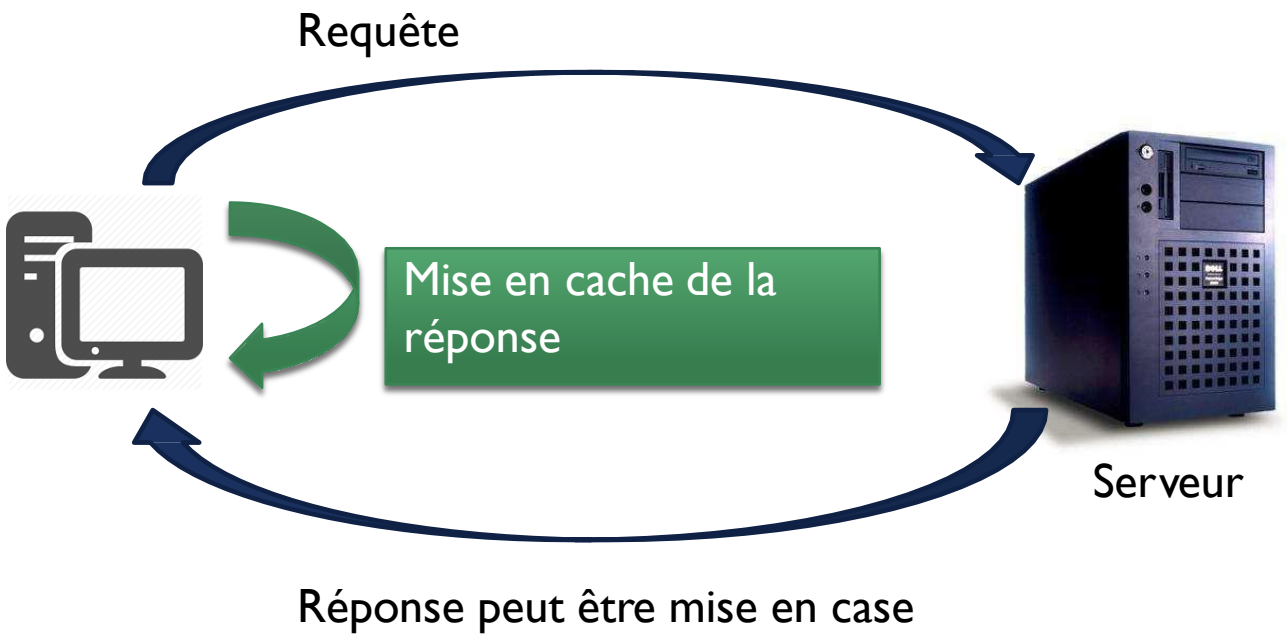
CARACTÉRISTIQUE REST:CLIENT / SERVEUR



CARACTÉRISTIQUE REST:SANS ÉTAT

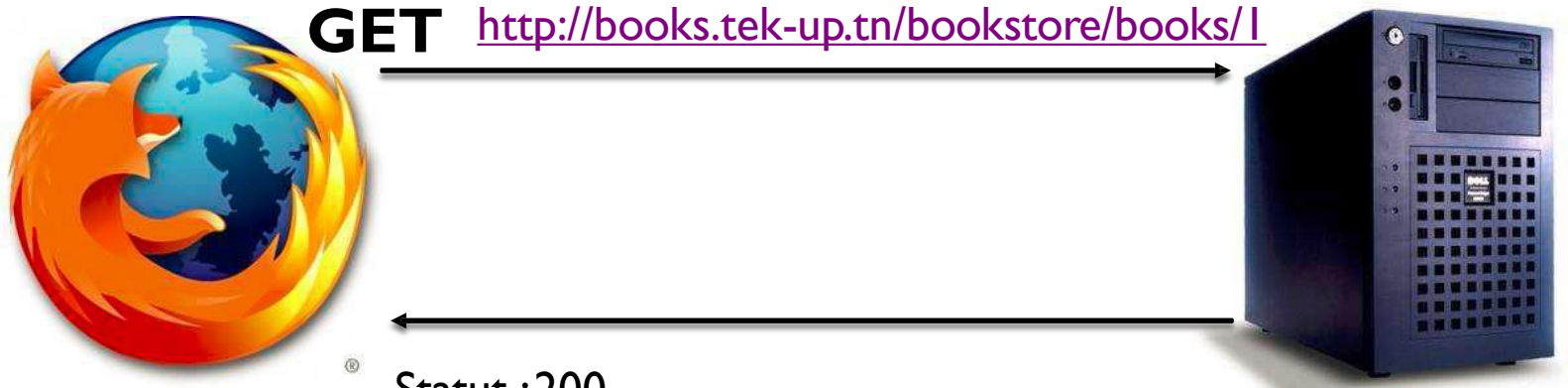


CARACTÉRISTIQUE REST:AVEC CACHE



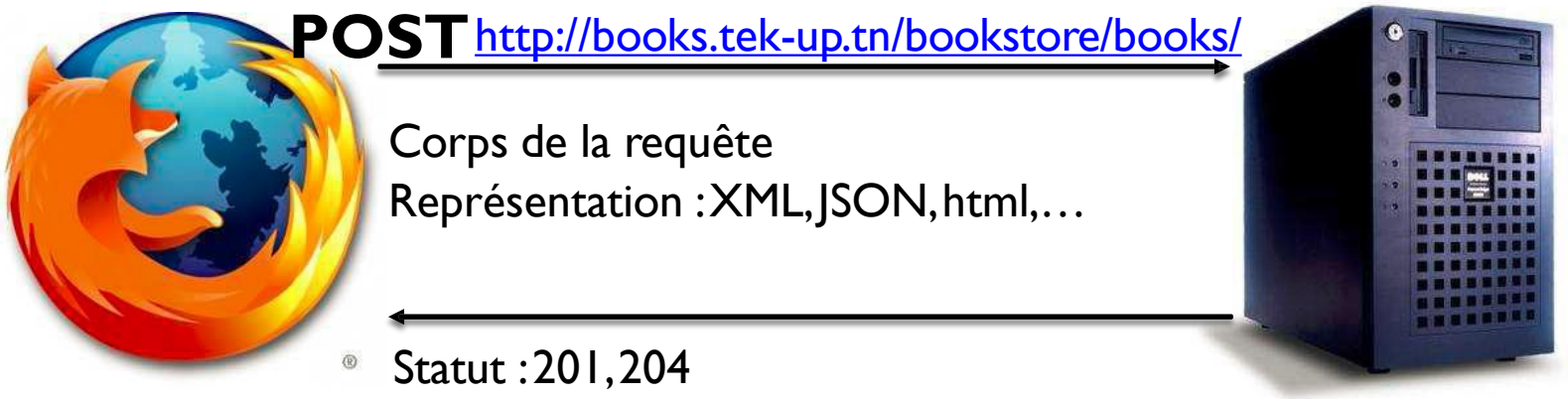
LES REQUÊTES « REST » : LA MÉTHODE « GET »

- La méthode GET renvoie une représentation de la ressource tel qu'elle est sur le système



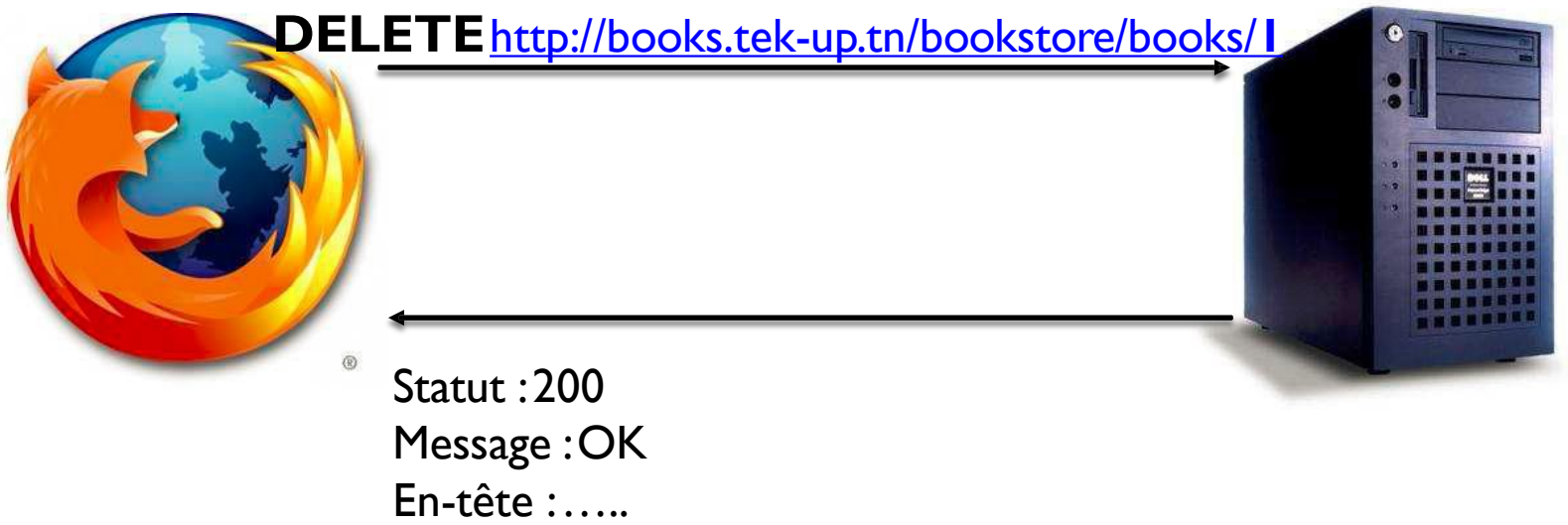
LES REQUÊTES « REST » : LA MÉTHODE « POST »

- La méthode POST crée une nouvelle ressource sur le système



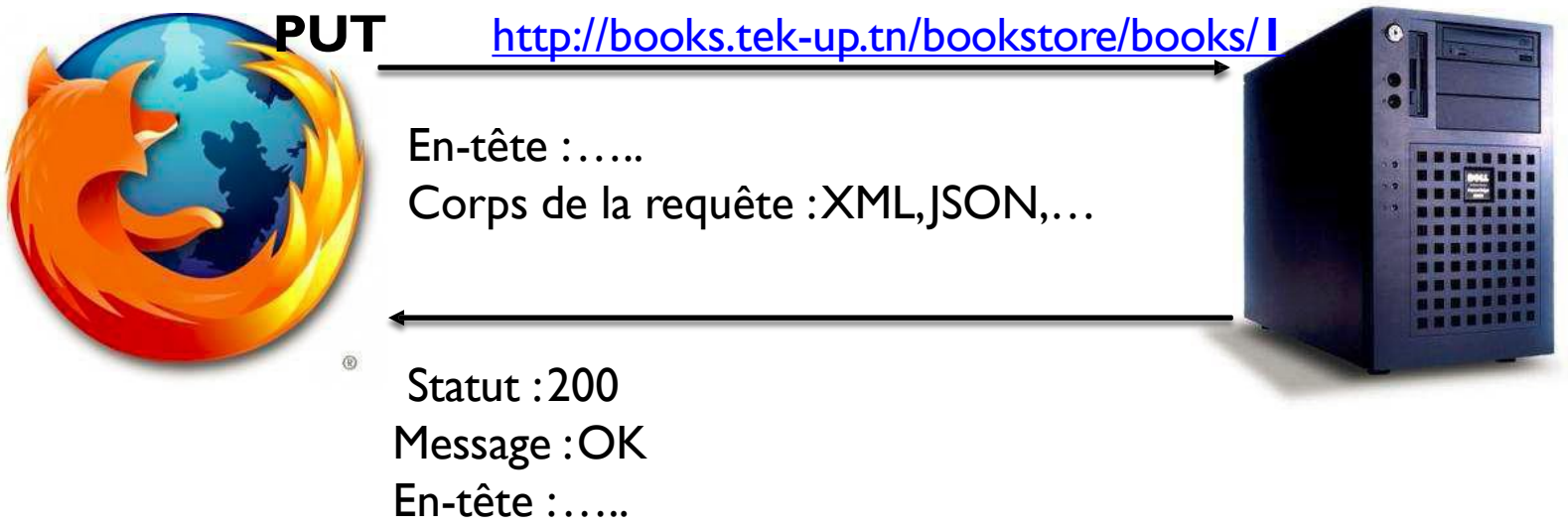
LES REQUÊTES « REST » : LA MÉTHODE « DELETE »

- Supprime la ressource identifiée par l'URI sur le serveur



LES REQUÊTES « REST » : LA MÉTHODE « POST »

- Mise à jour de la ressource sur le système



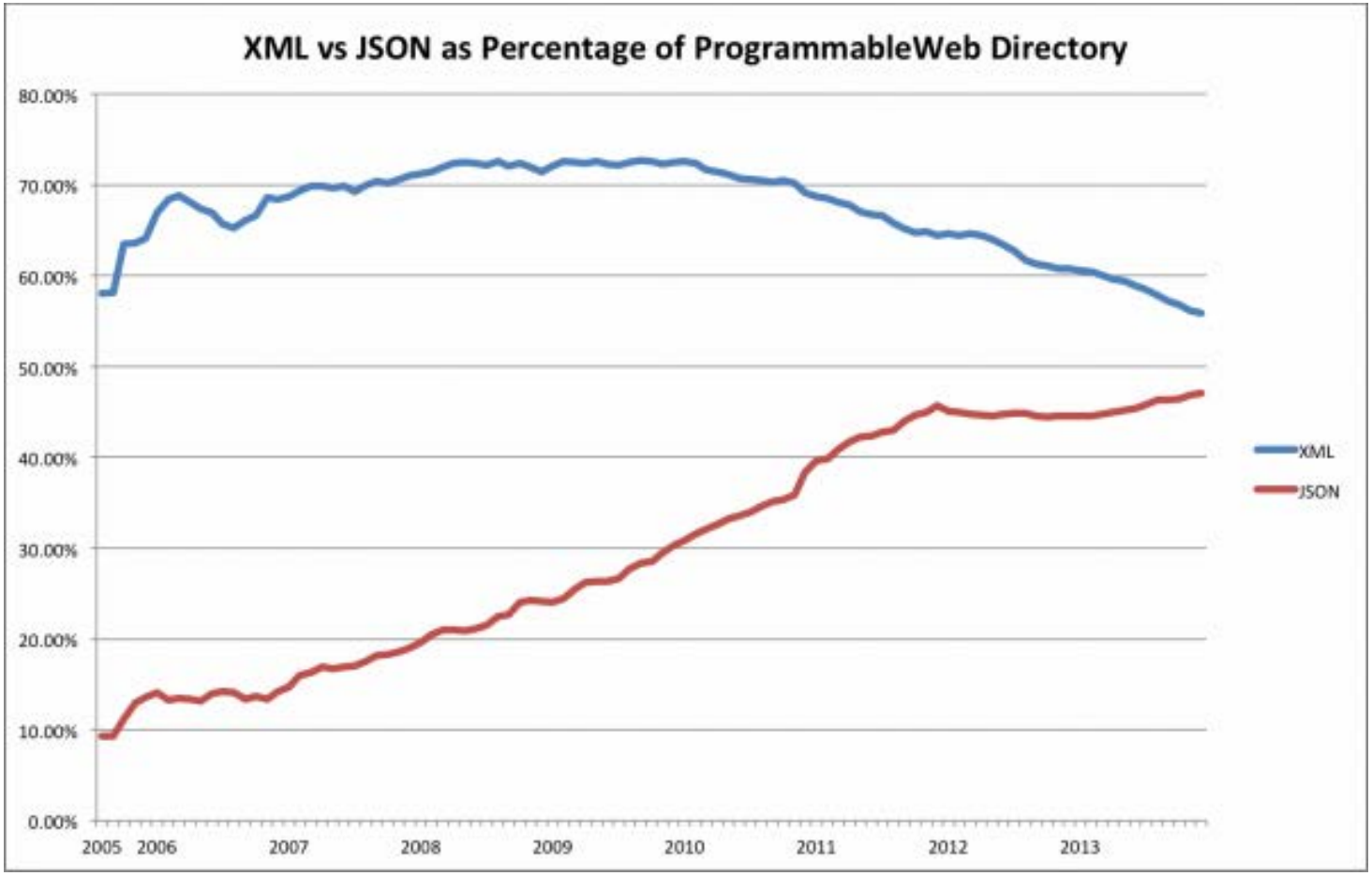
LES REQUÊTES « REST » : REPRÉSENTATION

- Une représentation désigne les données échangées entre le client et le serveur pour une ressource:
  - **HTTP GET** → Le serveur renvoie au client l'état de la ressource
  - **PUT, POST** → Le client envoie l'état d'une ressource au serveur
- Peut être sous différent format :
  - JSON
  - XML
  - XHTML
  - CSV
  - Text/plain …

JSON : RAPPEL

- **JSON** « JavaScript Object Notation » est un format d'échange de données, facile à lire par un humain et interpréter par une machine.
- JSON est basé sur le syntaxe JavaScript, il est complètement indépendant des langages de programmation mais utilise des conventions qui sont communes à toutes les langages de programmation (C, C++, Perl, Python, Java, C#, VB, JavaScript, ….)

JSON : DES STATISTIQUES



Source: www.programmableweb.com

UN OBJET JSON

- Il s'agit de la représentation de la structure Objet et qui s'agit d'une collection
  de clé → valeurs.
- Un objet JSON commence par un « { » et se termine par « } » et composé d'une liste non ordonnée de paire clefs/ valeurs. Une clef est suivie de « : » et les paires clef/ valeur sont séparés par « , ». Exemple:

```
{       "id": 51,
        "nom": "Mathematiques 1",
        "resume":"Resume of math ",
        "isbn": "123654",
        "categorie":
        {
                "id": 2, "nom": "Mathematiques", "description":
                "Description of mathematiques "
        },
        "quantite": 3,
}
```

UN «ARRAY » JSON

- Un « array » s'agit d'une collection ordonnée d'objets.
- Un « array » commençepar « [« et se terminant par «] »,les objets sont séparés l'un de l'autre par « ,».

```
[
      {     "id": 51,
            "nom": "Mathematiques 1",
            "resume": "Resume of math ",
            "isbn": "123654",
            "quantite": 3,
      },{   "id": 102,
            "nom": "Mathematiques 1|",
            "resume": “Linear Algebra",
            "isbn": "12365444455",
            "quantite": 2,
      }
]
```

EXEMPLE: TESTER UN SERVICEWEB

- « OpenWeather API » est un service Web nous permettant de récupérer des données sur les données de météo d'une ville (Température, vent, humidité…).
- Accéder à la documentation sur: https://openweathermap.org/current
- Créer une « APIKey »
- Construire l'URL qui permet de récupérer les prévisions de météo de « Tunis » pour les 5 jours prochains en degré Celsius et en format JSON.

SOAPVS REST: CARTE POSTALEVS COURRIER