Pourquoi une Partie II au cours de SI4

Applications Réparties ;=)

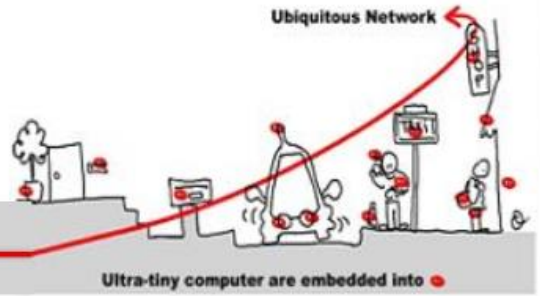
http://www-sop.inria.fr/members/Francoise.Baude/AppRep

contact: baude@unice.fr

POSITIONNEMENT

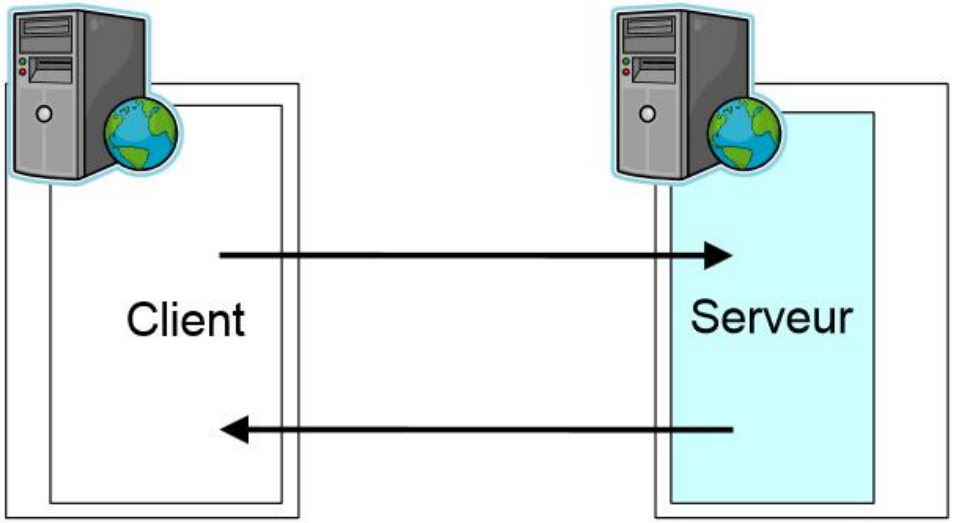
- Partie I : Service Oriented Computing, avec une approche dite faiblement couplée par WebServices
- Partie II: Présente des principes et solutions antérieures à la technologie des Web Services, mais, permet de « solidifier » l'apprentissage
  - Concepts fondamentaux qu'on retrouve dans de nombreux Middleware (Intergiciels)
    - Exemples précis dans ce cours: Java RMI, CORBA, JMS
  - Un middleware pour faire quoi ? Pour supporter l'exécution d'applications réparties sur plusieurs nœuds, le cas échéant parallèles (pour plus de performance à l'exécution)
    - Idée d'un projet sans serveur unique (à la J2EE)

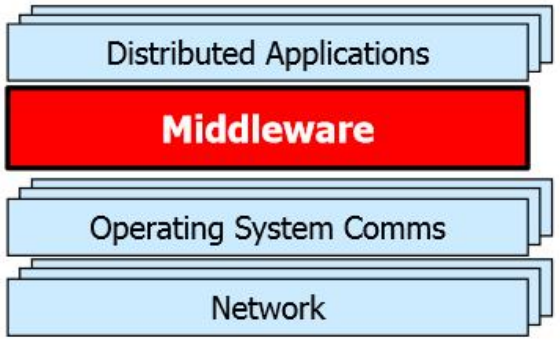
Evolution des intergiciels



- *"The intersection of the stuff that network engineers don't want to do with the stuff that applications developers don't want to do."*

[Kenneth J. Klingenstein ('99)]





Exemple d'un des objectifs clé de ce cours complet

- Voir transparent suivant !!
- Etre capable de maitriser le concept, et sa réalisation dans différentes technologies
- Exemple en

.....
.....
.....

Comment est-ce dans l'intergiciel JavaRMI

Cycle de Vie WSOA : une représentation explicite du Service

RMI CORBA utilisent ce même cycle de vie

CORBA explicite le service via du IDL

RMI: représentation implicite (partage des 2 côtés des .class interfaces)

- Etape 1 : Déploiement du service Web
  - Dépendant de la plate-forme

Préparer .class, dont ceux interfaces, et Lancer JVM(s) coté serveur

- Etape 2 : Enregistrement du service Web
  - WSDL : description du service
  - Cf. WS-* www.w3c.org

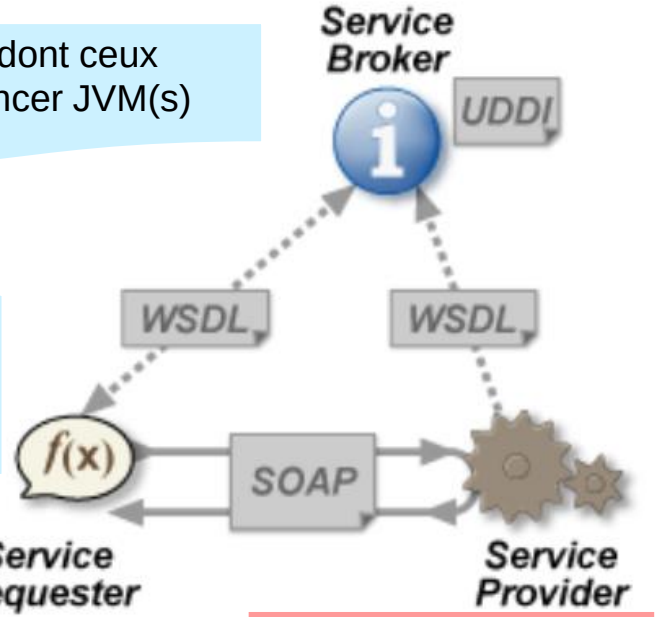
Basé sur bytecode proxy: Enregistrer nom/proxy ds RMIRegistry

- Etape 3 : Découverte du service Web
  - Référentiels : DISCO (local), UDDI (global)

Chercher proxy par nom dans RMIRegistry

- Etape 4 : Invocation du service Web par le client
  - WS-SOAP (Cf. WS-* www.w3c.org)

Utiliser proxy et protocole JRMP pour invoquer méthode

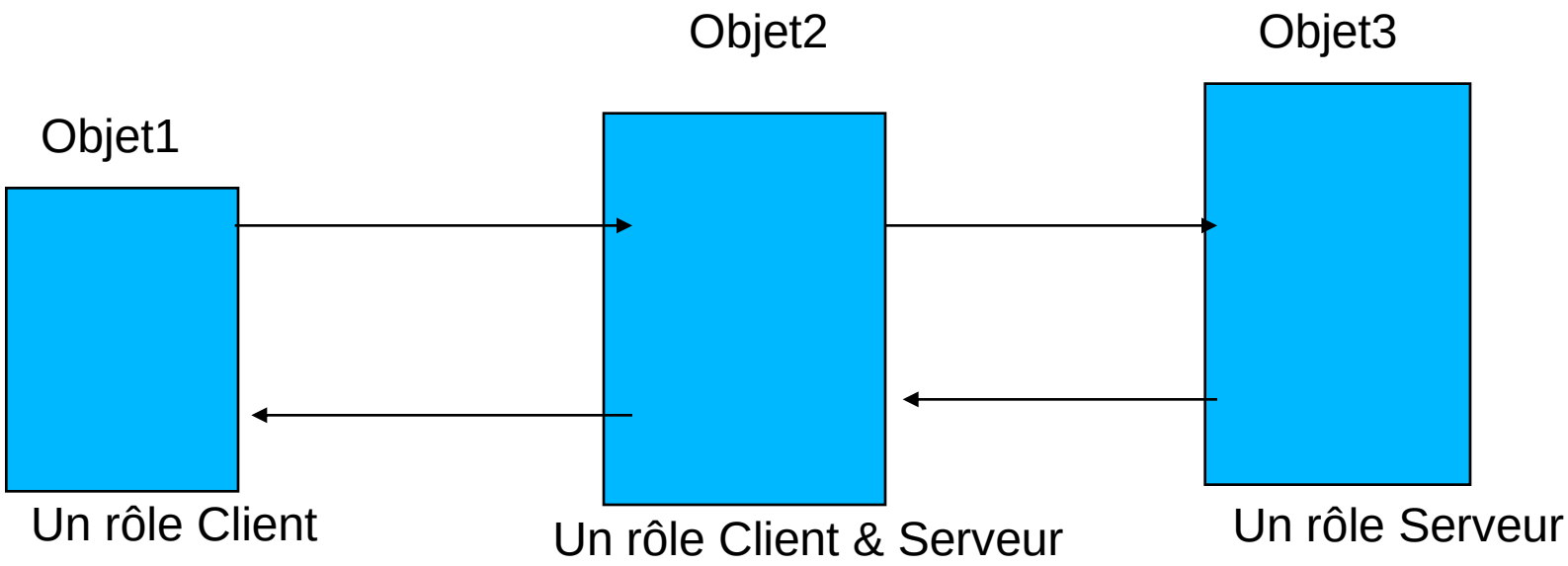


RMI suppose client connait déjà .class d'interfaces

Partie II : Contenu & Articulation globale

- Les approches purement Objet pour le Client / Serveur et les applications réparties (distribuées) en général
  - Un serveur/service peut être client d'un autre
  - Sans avoir forcément un point central qui coordonne tout
    - Approche pair-à-pair
- Les approches à échange de messages pour les applications réparties (distribuées) en général
  - Interaction totalement asychrone (désynchronisée)
  - Modèle publication/souscription (approche événementielle)
- Les plateformes à objets répartis
  - Sont fondées sur une extension naturelle du concept d'objet
    - En rendant un objet accessible depuis un autre objet, à distance
    - En respectant le protocole et support offert par la plateforme
      - RMI: plateforme Java étendue pour s'exécuter en distribué
      - DCOM: plateforme Microsoft COM étendue pour s'éxécuter en distribué
      - ORB "Object Request Broker": plateforme = intergiciel de type bus (transport des messages entre objets, avec transformations des messages pour interopérabilité)

Programmation à Objets Distribués: pourquoi ?



•La logique d'enchainement des méthodes offertes par les Objets (=services) peut être répartie sur les différents objets

- •Socle à une algo/prog intrinsèquement répartie ou parallèle
  - •Maitriser voire profiter de +eurs nœuds d'exécution en même temps

≠

•En prog .SOA, l'orchestration est le plus souvent guidée depuis un point central (code/moteur BPEL) permettant d'utiliser des WS tiers indépendants

- •Granularité plus grosse, services indépendants (stateless le + souvent)
- •Objectif est l'intégration faiblement couplée de services déjà existants

Comparaison WS / OD point de vue « plateforme »

- Plateforme (intergiciel = pour connecter des logiciels) toujours nécessaire !
  - Dans le cas WS: pourquoi ne pas tout simplement profiter de la présence de la plateforme web HTTP ?
    - A necessité d'établir un protocole d'interaction à distance entre clients et services offerts: SOAP et WSDL pour exposition,REST
  - Dans le cas O.D.: les objets sont naturellement faits pour interagir (Appli OO = un ensemble d'objets bien délimités qui s'envoient des messages / requête-réponse avec paramètres dont des paramètres=objets)
    - Partant de chaque solution OO: extension de la plateforme pour permettre transport des requêtes-réponses entre différents hôtes
      - Au dessus des protocoles standards de l'Internet (TCP)
      - Exposition des objets en services: naturel, +/- léger/intégré
      - Et services additionnels NF +/- intégrés: annuaires, GC D.

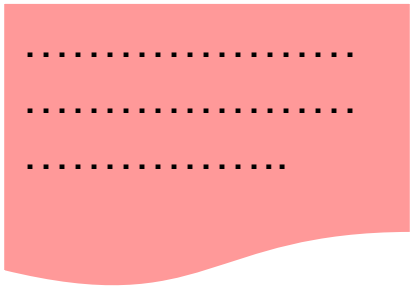
Technos Client Serveur: Comparaison

	RMI	RPC	DCOM	CORBA	WS SOAP
Qui	SUN	SUN/OSF	MicroSoft	OMG	W3C
Plate-formes	Multi	Multi	Win32	Multi	Multi
Langages de Programmation	Java	C, C++, ...	C++, VB, VJ, OPascal, ...	Multi	Multi
Langages de Définition de Service	Java	RPCGEN	ODL	IDL	XML
Réseau	TCP, HTTP, IIOP customisable	TCP, UDP	IP/IPX	GIOP, IIOP, Pluggable Transport Layer	RPC,HTTP SNMP
Firewall	Tunneling HTTP			HTTP Tunneling CORBA Firewall	HTTP
Nommage	RMI, JNDI,JINI	IP+Port	IP+Nom	COS Naming COS Trader	IP+Port, URL
Transaction	Non	Non	MTS	OTS, XA	Extension applicative dans le header
Extra	Chargement dynamique des classes			Services Communs Services Sectoriels	

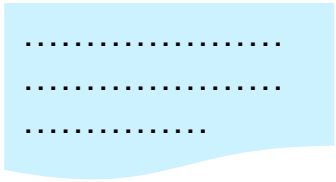
Jeu de Transparents de Synthèse

- A LIRE à la FIN du MODULE Applications Réparties
- Permet de bien situer PartieI vis-à-vis de PartieII
- Légende:

Les solutions étudiées offrent même avantages, propriétés que WebServices, et l'analogie est forte

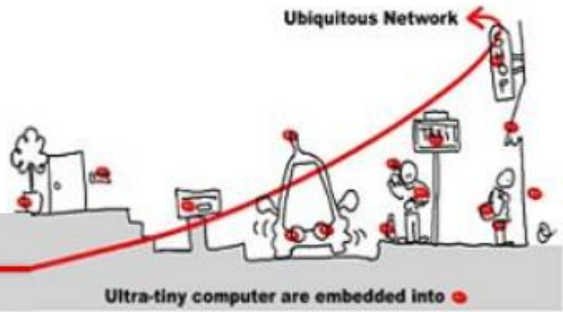


Les solutions étudiées présentent des inconvénients par rapport à la techno. WebServices



Element factuel

Caractéristiques des intergiciels orientés Service

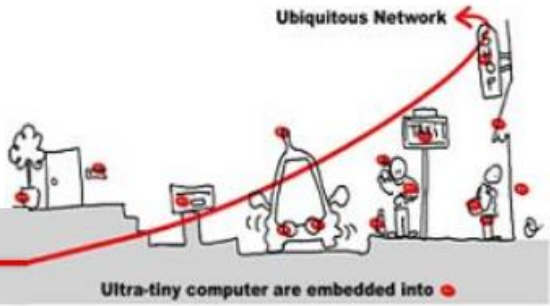


- Caractéristiques
  - Architecture de type Client / Serveur
    - Le serveur rend des services à un client
  - Réutilisable
    - Par plusieurs clients (simultanément ou pas)
  - Indépendamment de
    - la plate-forme (UNIX, Windows, …)
    - du langage pour l'implémentation (VB, C#, Java, …)
    - la plate-forme de développement sous-jacente (.NET, J2EE, Axis…)

Tout ça était déjà vrai dans le cas de CORBA, où IDL ≈ WSDL

Mais intergiciel de CORBA devait être installé sur chaque nœud alors que WS repose sur technos du web ±déjà là, pour le transport notamment

Quels objectifs pour les Services Web ?



- Remplacer les outils (RPC, DCOM, CORBA, RMI) par une approche entièrement ouverte et interopérable, basée sur la généralisation des serveurs Web avec scripts CGI.

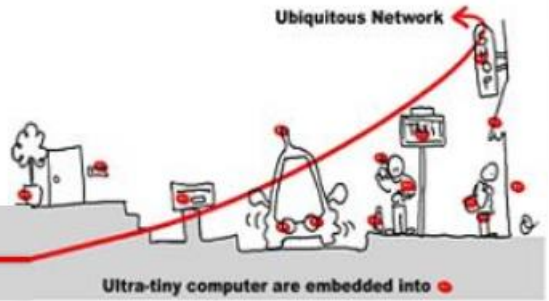
RMI, CORBA requièrent leur propre technologie serveur pour rendre les services accessibles

- Faire interagir des composants hétérogènes, distants, et indépendants avec un protocole standard (ex. SOAP).

- Passer les politiques de sécurité grâce en grande partie à une couche session basée sur HTTP (port 80).

RMI comme CORBA requièrent des ports a priori non ouverts

Services Web et Interopérabilité



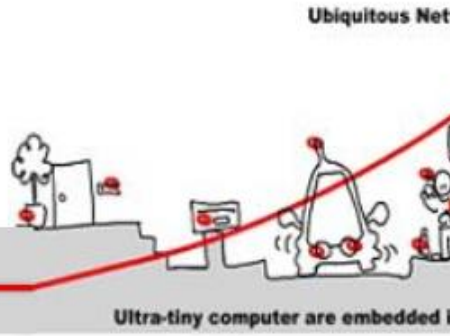
- Plateform independent : A l'instar de Corba (précurseur historique), les Services Web gèrent l'interopérabilité au niveau du protocole d'échange.

- Plateform dependent: d'autres approches gèrent l'interopérabilité par le portage de la plate-forme d'exécution (ex. OSGi, RMI sur Java et historiquement COM/DCOM)

RMI suppose versions compatibles de JDK pour les JVMs…

Java néanmoins est bien répandu, et masque bien hétérogénéité sous-jacente => environnement virtuel uniforme pour le pgmmeur

Pour quoi faire ?



- Les Services Web permettent d'interconnecter :
  - Différentes entreprises
  - Différentes applications
  - Différents clients
  - Différents matériels

JavaRMI et CORBA peuvent aussi être interconnectés pour finalement offrir ces différentes interconnections

- Utilisé dans différents cadres:
  - B2B (Business To Business)
  - EAI (Enterprise Application Integration)
  - …

JavaRMI supporte assez bien des applis parallèles distribuées où un serveur est aussi un client d'un autre

Cycle de Vie WSOA : une représentation explicite du Service

RMI CORBA utilisent ce même cycle de vie

CORBA explicite service via du IDL

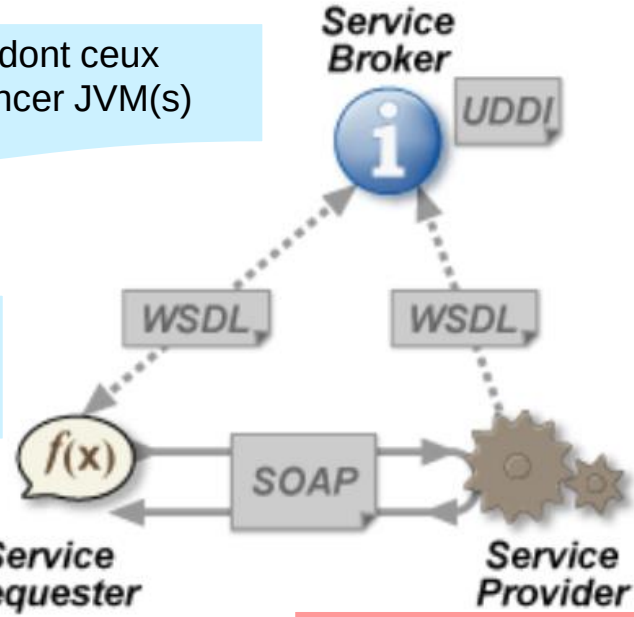
RMI: représentation implicite (partage des 2 côtés des .class interfaces)

- Etape 1 : Déploiement du service Web
  - Dépendant de la plate-forme

Préparer .class, dont ceux interfaces, et Lancer JVM(s) coté serveur

- Etape 2 : Enregistrement du service Web
  - WSDL : description du service
  - Cf. WS-* www.w3c.org

Basé sur bytecode proxy: Enregistrer nom ds RMIRegistry



- Etape 3 : Découverte du service Web
  - Référentiels : DISCO (local), UDDI (global)

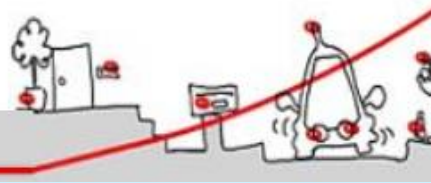
Chercher proxy par nom ds RMIRegistry

RMI suppose clie connait déjà .clas d'interfaces

- Etape 4 : Invocation du service Web par le client
  - WS-SOAP (Cf. WS-* www.w3c.org)

Utiliser proxy et protocole JRMP pour invoquer méthode

Modèle de message



SOAP (ancien *acronyme* de Simple Object Access Protocol) est un protocole de RPC orienté objet bâti

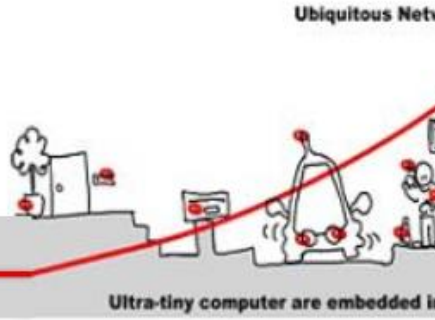
- SOAP permet une communication par message
  - d'un expéditeur vers un récepteur
- Structure d'un message
  - Envelop (enveloppe)
    - Elément racine
    - Namespace : SOAP-ENV http://schemas.xmlsoap.org/soap/envelope/
  - Header (entête)
    - Elément optionnel
    - Contient des entrées non applicatives (transactions, session, ...)
  - Body (corps)
    - Contient les entrées du message
      - Nom d'une procédure, valeur des paramètres, valeur de retour

En RMI: c'est l'équivalent de la mécanique de JRMP

En CORBA: c'est l'équiva de IIOP

SOAP et autres= **Mécanique** et **sémantique** d'une invocation d' méthode distante « RPC »

Portée et Limitations de SOAP



- SOAP est simple et extensible…
  - Format XML over HTTP
  - Multi-langages
  - Multi-plateformes

RMI par prin
d'une platefo
virtualisante
connait tous
objets distan
JRMP offre a
support d'op
DGC

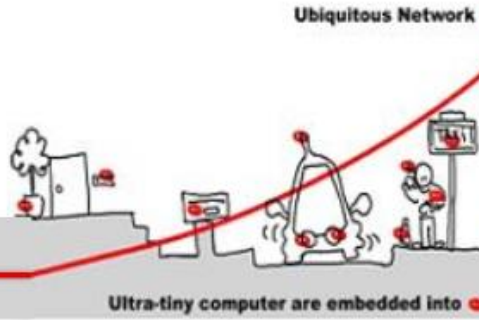
- … mais il ne couvre pas les fonctions suivantes :
  - Distributed garbage collection
  - Objects-by-reference (qui requière en autre une distributed garbage collection)
  - Activation (qui requière objects-by-reference)

RMI et COR
offrent Activa

SOAP décrit **la manière** dont les applications doivent communiquer entre elles, certains considèrer
le couplage reste fort entre le serveur et ses clients. Une modification de l'API implique ainsi une év
côté client, contrairement à une architecture orientée ressources telle que REST.

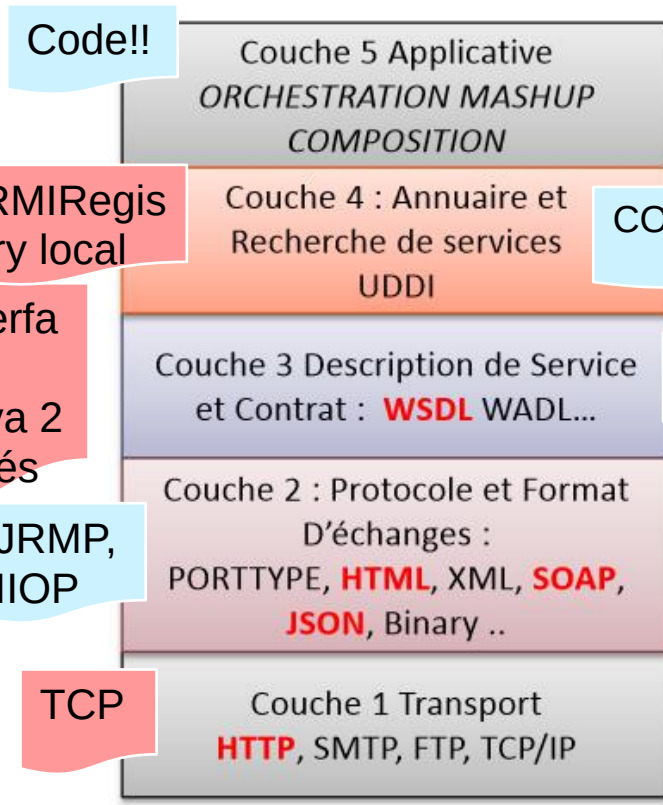
http://fr.wikipedia.org/wil

Pour mémoire : Pile Protocolaire des Services Logiciels issus du Web

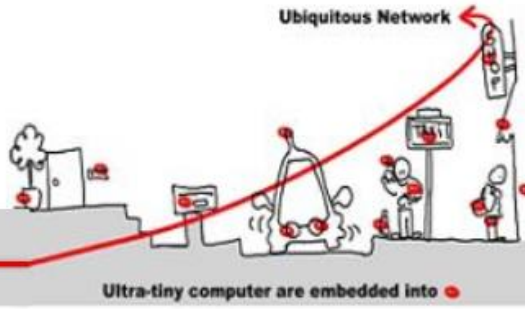


- Les principaux composants ou couches d'une pile de protocoles de services Web incluent :

- **Couche Transport**— assure la transmission des messages entre les applications
- **Couche Protocole et Format D'échanges** — encode et gére la séquence des messages échangés entre le service et son consommateur
- **Couche Description de Service et Contrat** — décrit le service fourni
- **Couche Annuaire et Recherche de Services**— centralise les services au moyen d'un registre commun



Élément <binding>



- Spécifie une liaison d'un <portType> à un protocole concret (SOAP1.1, HTTP1.1, MIME, …)

```
<binding name="AddressBookSOAPBinding" type="AddressBook">
 <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="addEntry">
  <soap:operation soapAction=""/>
  <input> <soap:body use="encoded" namespace="urn:AddressFetcher2"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/> </input>
  <output> <soap:body use="encoded" namespace="urn:AddressFetcher2"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/> </output>
 </operation>
 <operation name="getAddressFromName">
  <soap:operation soapAction=""/>
  <input> <soap:body use="encoded" namespace="urn:AddressFetcher2"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></input>
  <output> <soap:body use="encoded" namespace="urn:AddressFetcher2"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></output>
 </operation>
</binding>
```

Un seul binding possible