



ECOLE NATIONALE POLYTECHNIQUE D'ORAN – MAURICE AUDIN

Département Génie des Systèmes Informatiques

Filières IMSI & RT : 4^{ème} année ingénieurs

Systèmes Orientés Services

M. SABRI

ram.sabri@gmail.com

État des lieux des SI

La généralisation de l'outil informatique dans les directions métiers fait que les utilisateurs attendent toujours plus du SI :

- soit au niveau d'**agilité**,
- soit au niveau de **l'intégration des acteurs externes de leur SI**.

- Il devient donc vital de **trouver des moyens de mieux gérer et homogénéiser le SI**.
- Les DSI doivent **trouver un moyen d'organiser et de maîtriser cette hétérogénéité**.
- Et rationaliser la mise en commun d'informations transverses à l'entreprise - **un référentiel**.

État des lieux des SI

1.

La diversité des applications d'un même SI pose un grand **problème d'intégration d'applications**.

2.

La **communication entre les applications** et les différents SI devient de plus en plus difficile à cause des **technologies hétérogènes** utilisées qui amènent les utilisateurs à travailler dans un environnement incohérent, mal adapté et incompatible.

3.

Le **manque d'interopérabilité** est alors relevé comme principal problème de ces systèmes.

État des lieux des SI - Phénomène vertical

Processus rigides

- Processus complexes
- Processus non transférables

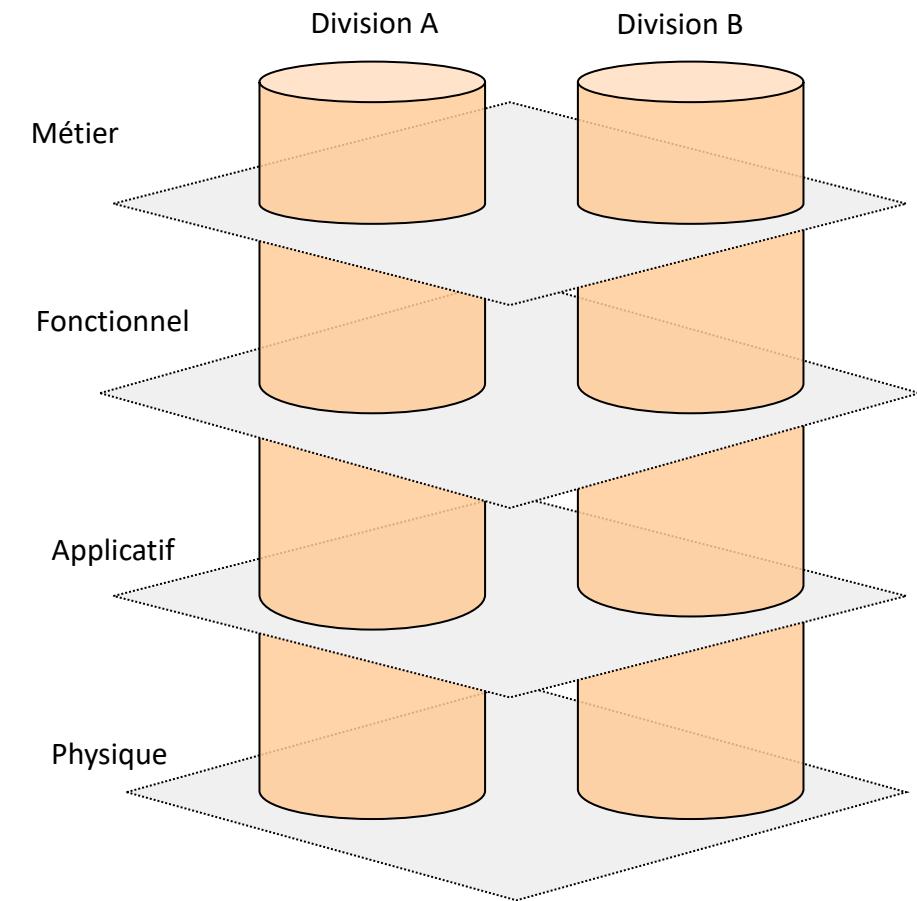
+

Composants peu réutilisables

- Hétérogénéité technologique

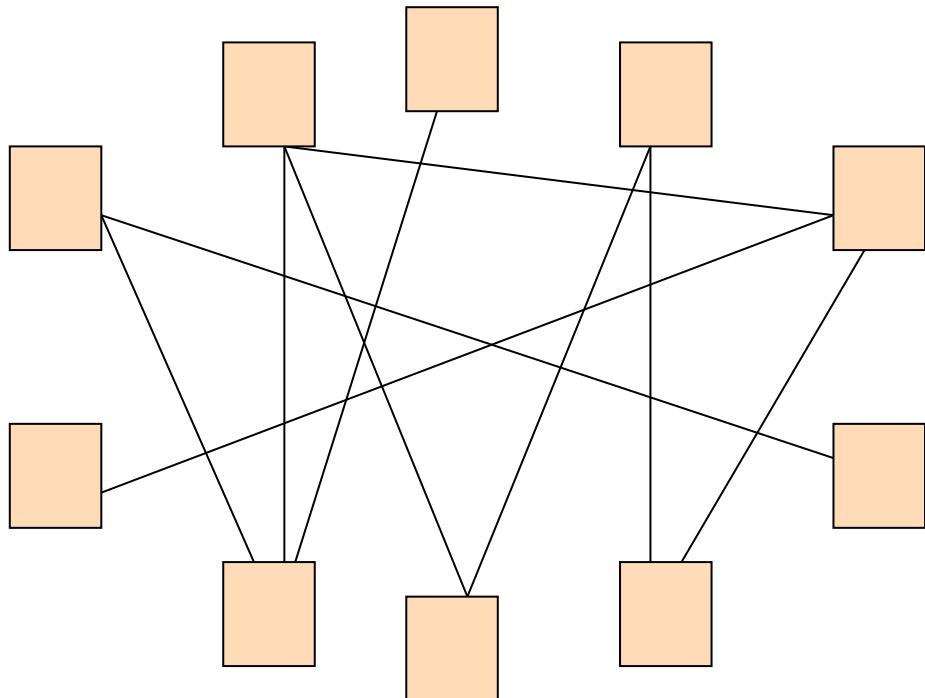
=

Problématiques des silos applicatifs



Le problème d'hétérogénéité technologique et plus que les applications sont mal équipés pour communiquer avec les autres blocs on parle de fonctionnement en silos

État des lieux des SI - Phénomène vertical



Redondance

- Données
- Traitements

Parc applicatif rigide

- Interdépendance élevée
- Difficulté d'évolution

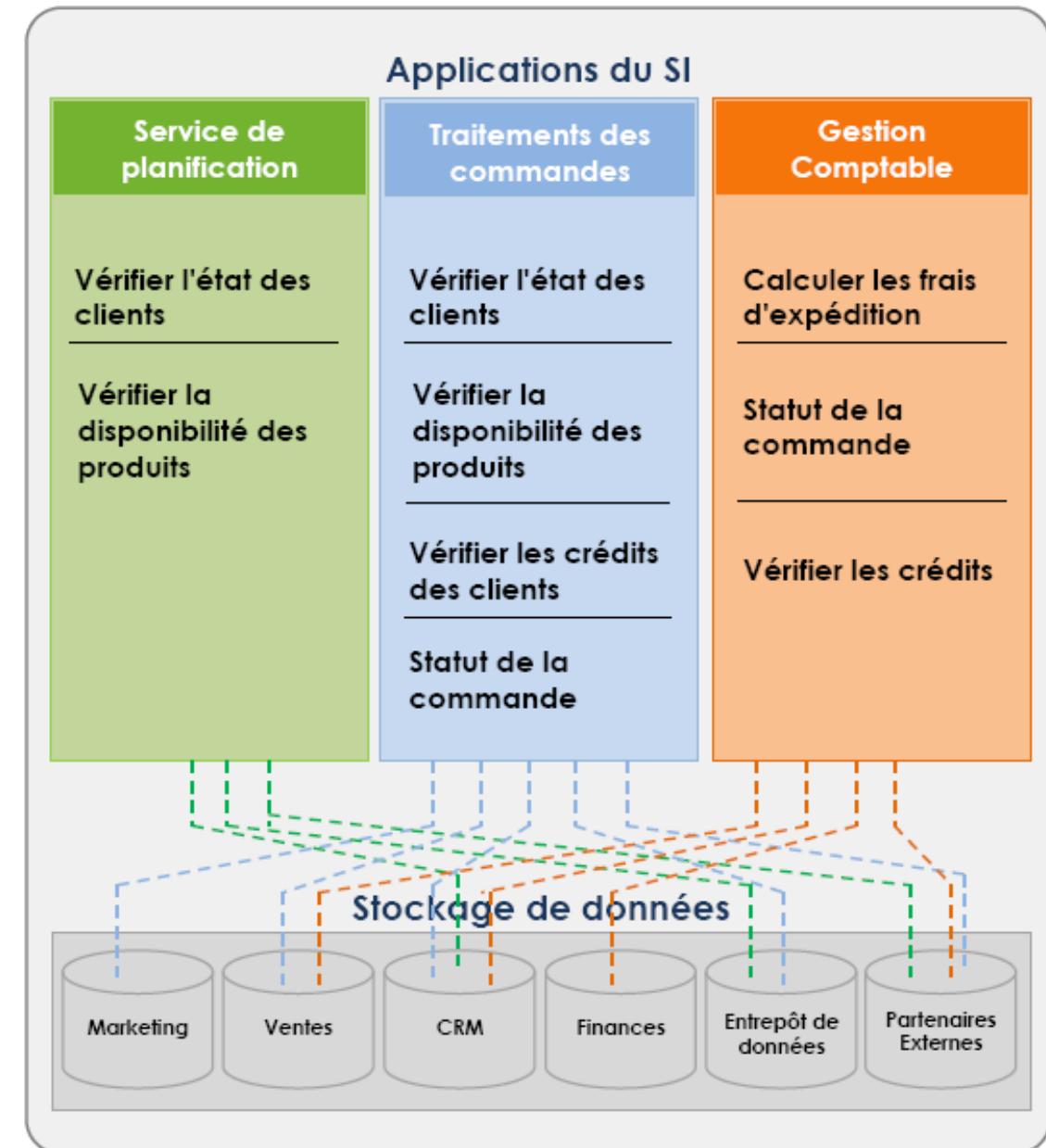
« Syndrome du plat de spaghetti »

Cartographie Applicative

la cartographie permet de représenter le système d'information d'une organisation ainsi que ses connexions avec l'extérieur.

Cette représentation peut être plus ou moins détaillée et inclure, par exemple, les biens matériels, logiciels, les réseaux de connexion, mais aussi les informations, activités et processus qui reposent sur ces biens.

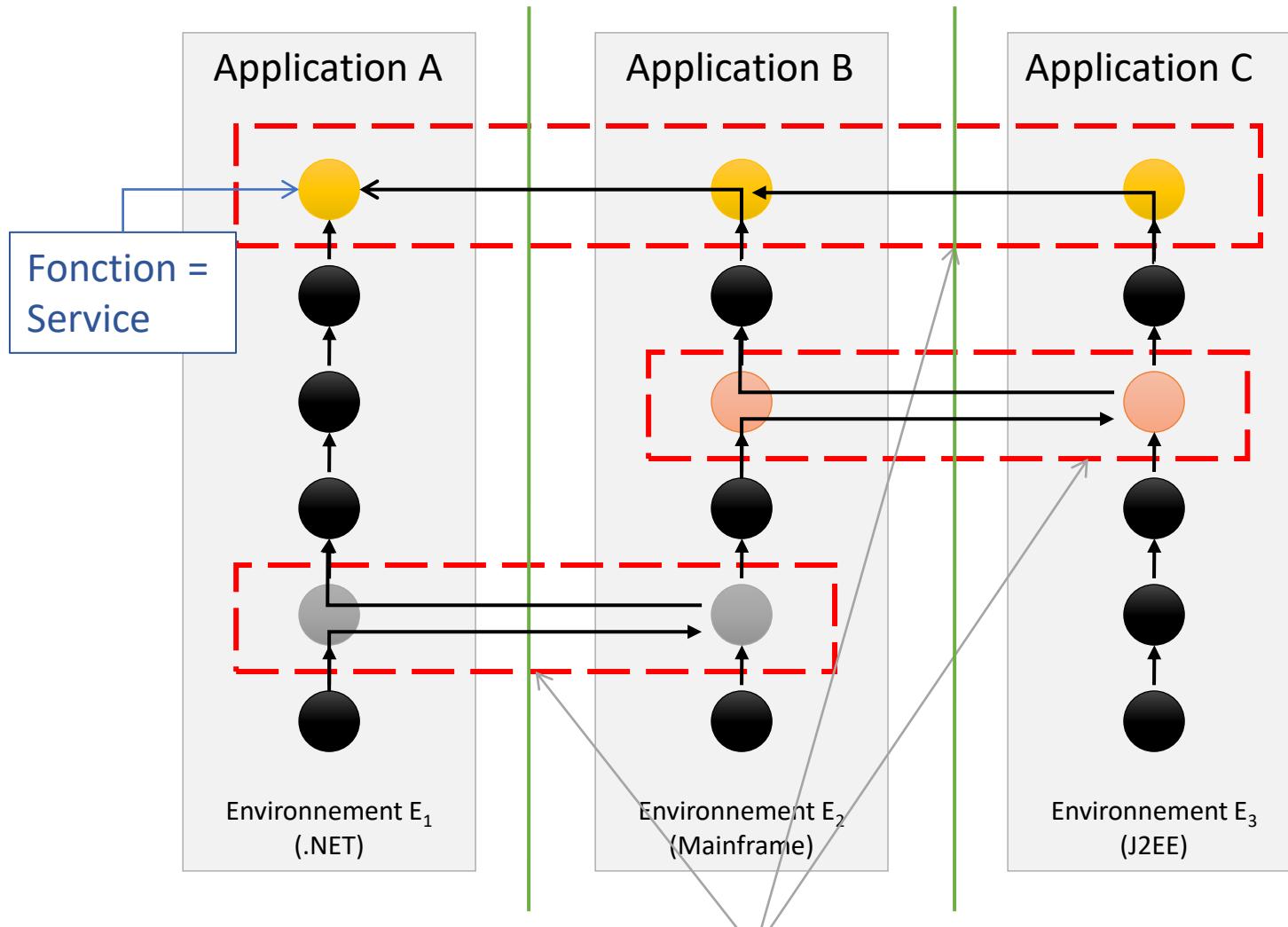
Travail à réaliser individuellement par l'étudiant pendant le stage du printemps : Etablir la cartographie applicative de l'organisme d'accueil du stage



Exposés

1. Le modèle MVC + Une implémentation avec un langage (Exemple Java JSF, Spring, ...)
2. Le modèle-vue-vue modèle (MVVM) (Microsoft WPF),
3. Frontend et Back end (Node JS et Angular)
4. Serveurs d'application (SA) – Etude d'un SA commercial ou gratuit (JBoss, Tomcat, WebSphere, Weblogic, ...)
5. Remote Procedure Call (RPC)
6. Remote Method Invocation (RMI) (Solution JAVA)
7. Solution CORBA
8. Socket
9. Message Exchange Patterns (MEPs)
10. Middleware Orienté Message (MOM)
11. Java Message Service (JMS)
12. Loose Coupling (couplage faible) for large distributed systems
13. Enterprise Application Integration (EAI)
14. Enterprise Service Bus (ESB)
15. HyperText Transfer Protocol (HTTP)
16. Multipurpose Internet Mail Extension (MIME)
17. Java API for XML Web Services JAX-WS

La cible



Une Architecture Orientée Service (SOA) est une Architecture technico-fonctionnelle dans laquelle les fonctions réutilisables du SI sont modélisées et exposées via des standards pour contribuer à la réalisation des Processus Métier.

Définition

L'architecture orientée services offre :

- la **possibilité de coupler des composants dans plusieurs configurations** dans la structure d'une plateforme,
- et de **les réutiliser** pour différentes constructions ce qui augmente considérablement la **flexibilité** du SI;

se sont les principaux avantages du SOA.

L'**interopérabilité** et la **cohérence** sont obtenues lorsque vous obtenez un système qui **répond aux besoins d'évolution des SI**.

Définition

La SOA est avant tout une **démarche de conception** contribuant au besoin d'urbanisation du SI, sans pour autant être l'apanage d'une technologie.

Pour les équipes métier :

la SOA permet d'être plus réactif et rapide dans l'innovation de modèles et des processus pour créer des produits à moindre coût en se dotant d'un avantage concurrentiel et en optimisant la collaboration interne et externe à l'entreprise.

Pour les équipes IT :

la SOA a pour but de créer une réelle interopérabilité entre les différents silos applicatifs du SI et de faciliter l'ouverture du SI aux partenaires de l'entreprise.

Démarche du cours

Patrons d'architecture pour les applications

- Architecture en couches
- Modèle n-tiers
- Composants
- Infrastructures logicielles

Flux

- Typologie
- Qualification des flux

Architectures d'échange

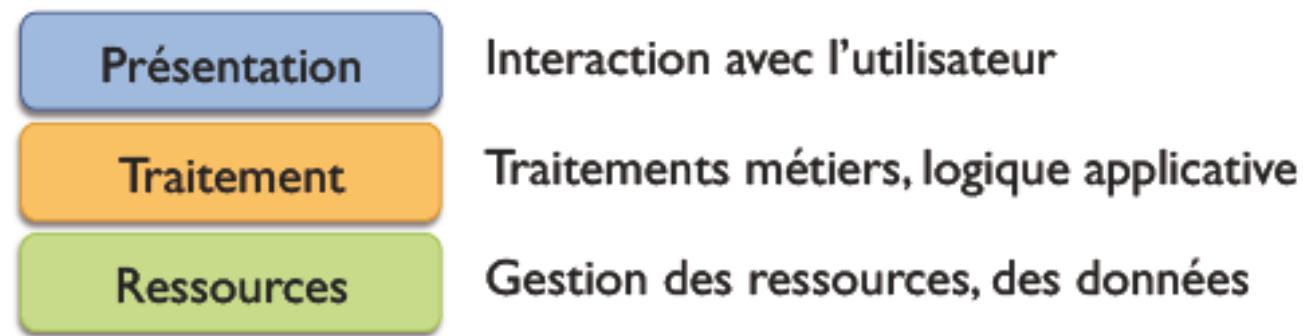
- Typologie des architectures
- Solutions logicielles

Les Services Web

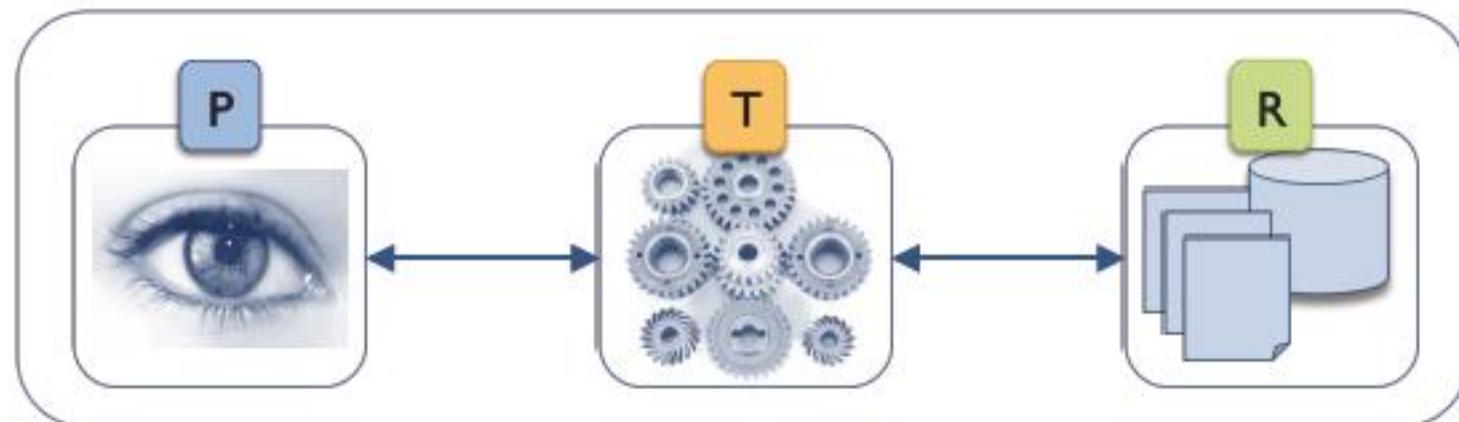
Les micro-services

Couches applicatives (rappel...)

3 types de responsabilités = 3 couches principales

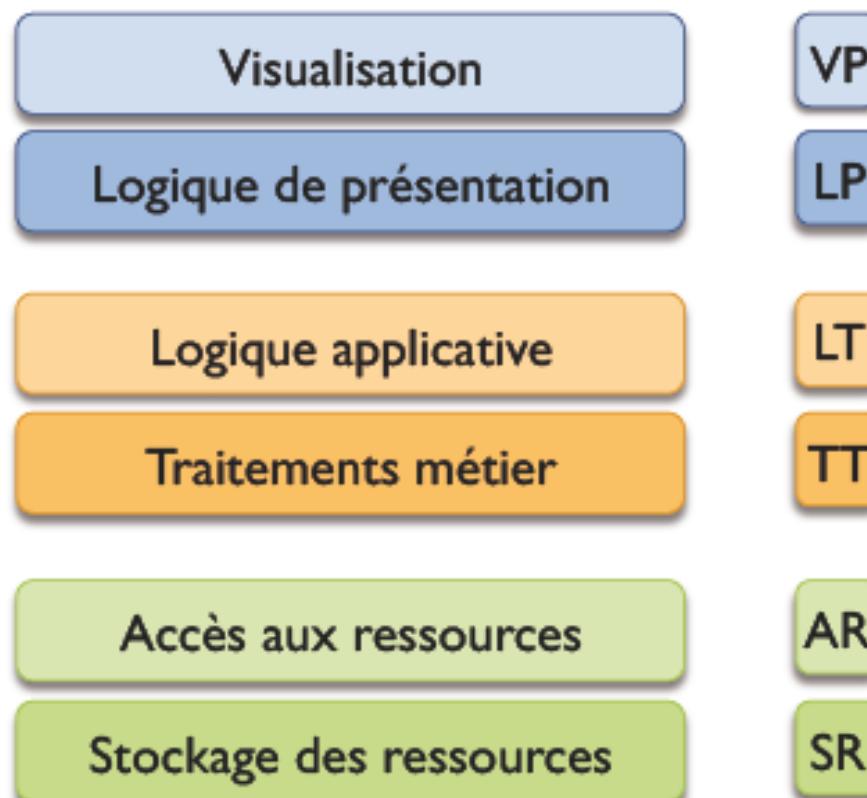


Principe de conception = séparation des responsabilités



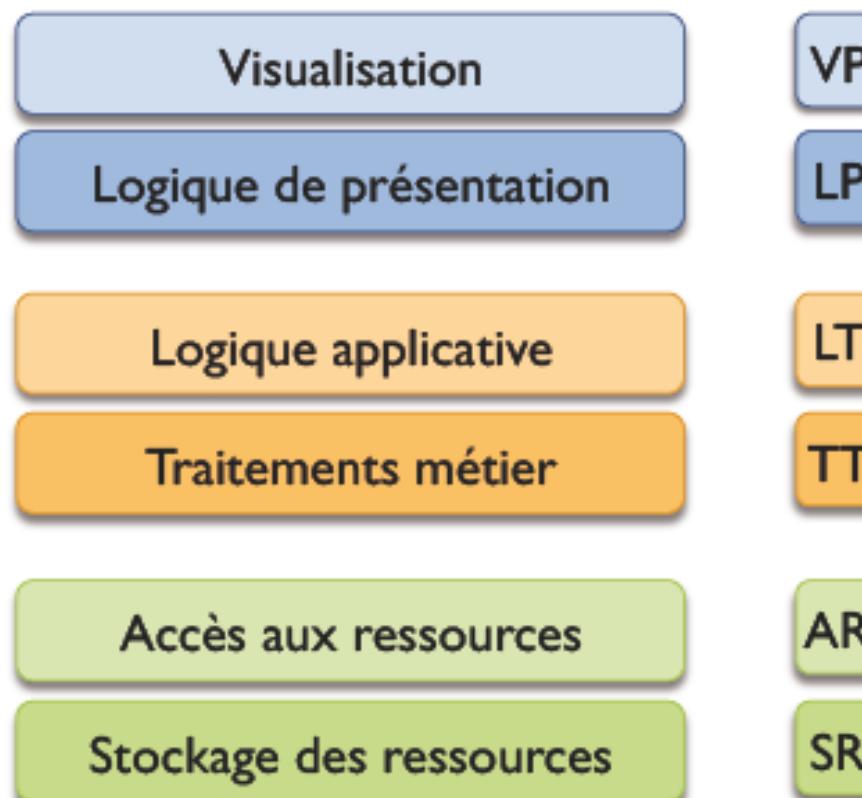
Couches applicatives détaillées

Vue plus fine des responsabilités



Couches applicatives détaillées

Vue plus fine des responsabilités

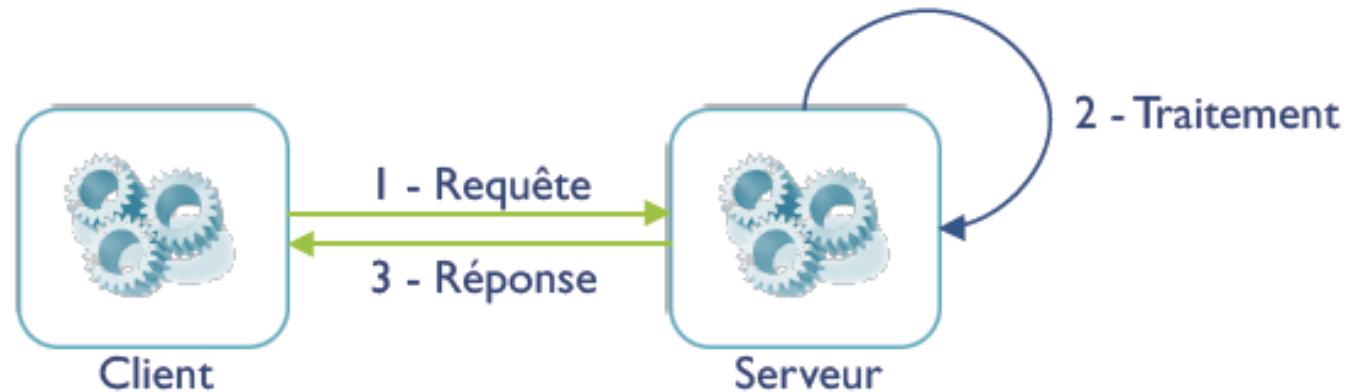


Modèle Client-Serveur (rappel...)

Modèle Client-Serveur

=
2 programmes

+
1 protocole



Programme « Serveur » =
Offre un service a des clients

Programme « Client » =
Utilise un service fourni par un serveur

Protocole =
Moyen de communication

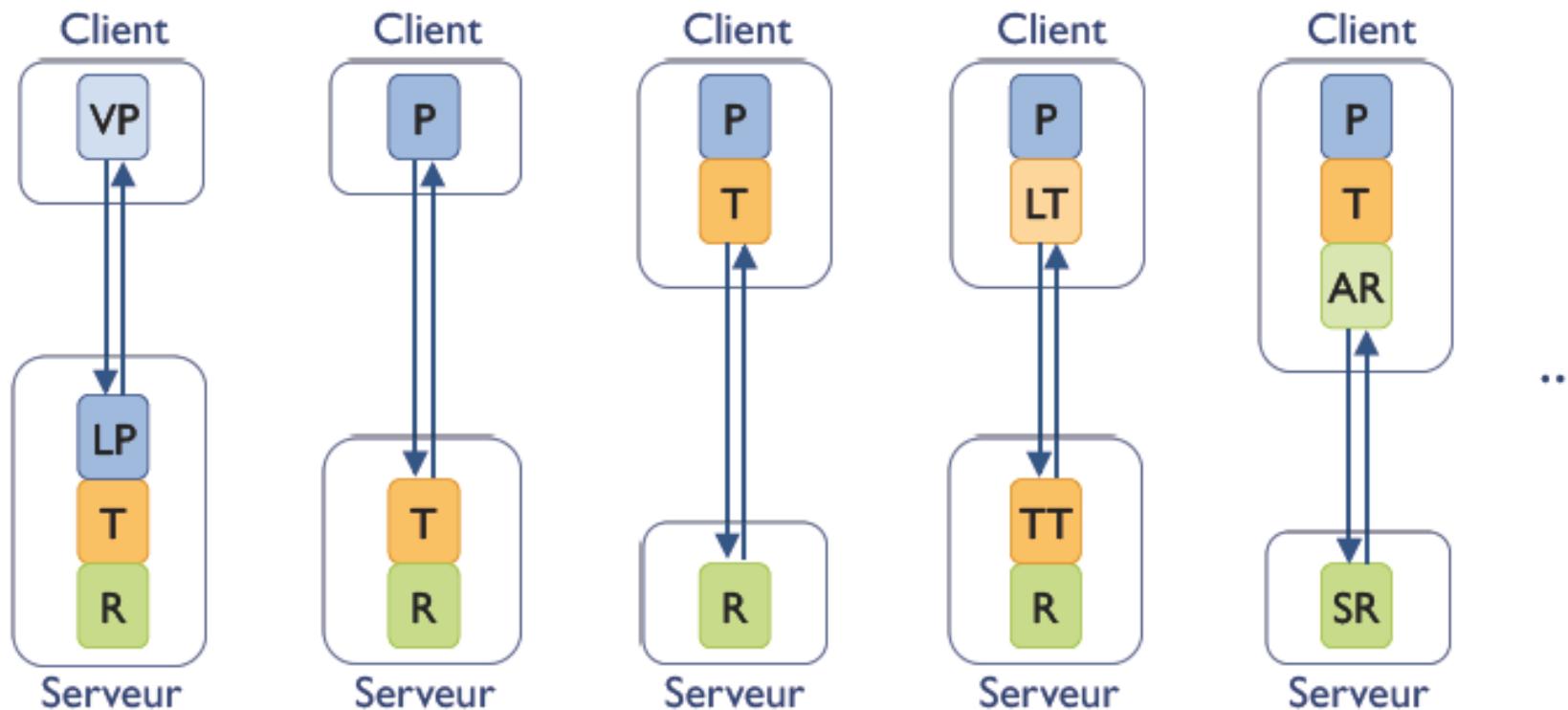
Indépendant de la notion de « machine »

- Client et serveur sur la même machine
- Client et serveur sur des machines différentes

Modèle Client-Serveur - Où placer les couches applicatives ?

Distribution des couches

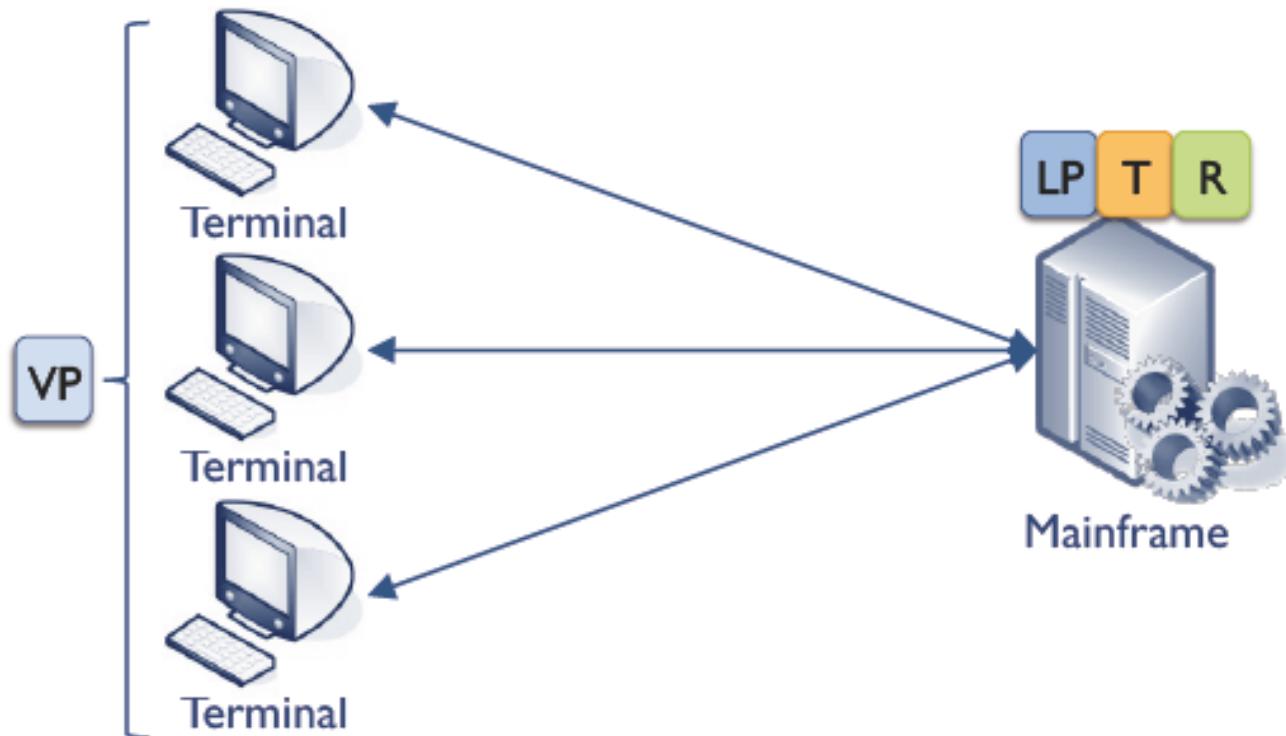
Possibilités multiples = typologies multiples



Architecture 1-tiers (mainframe)

- L'application est sur un serveur (éventuellement distant)
- Le client est une application « légère » de visualisation (client « passif »)

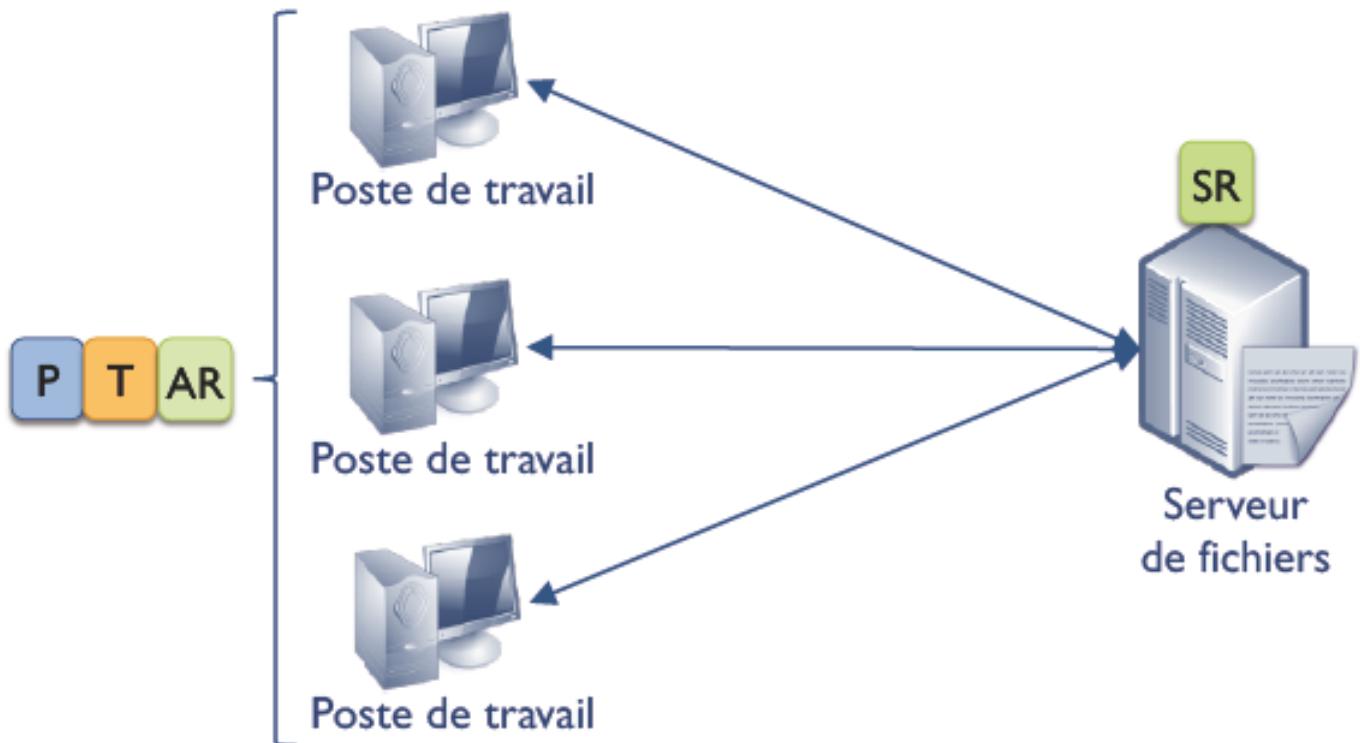
Exemple :



Architecture 1-tiers (Client Autonome)

- L'application est sur le client
- Les données sont sur un serveur (éventuellement distant)

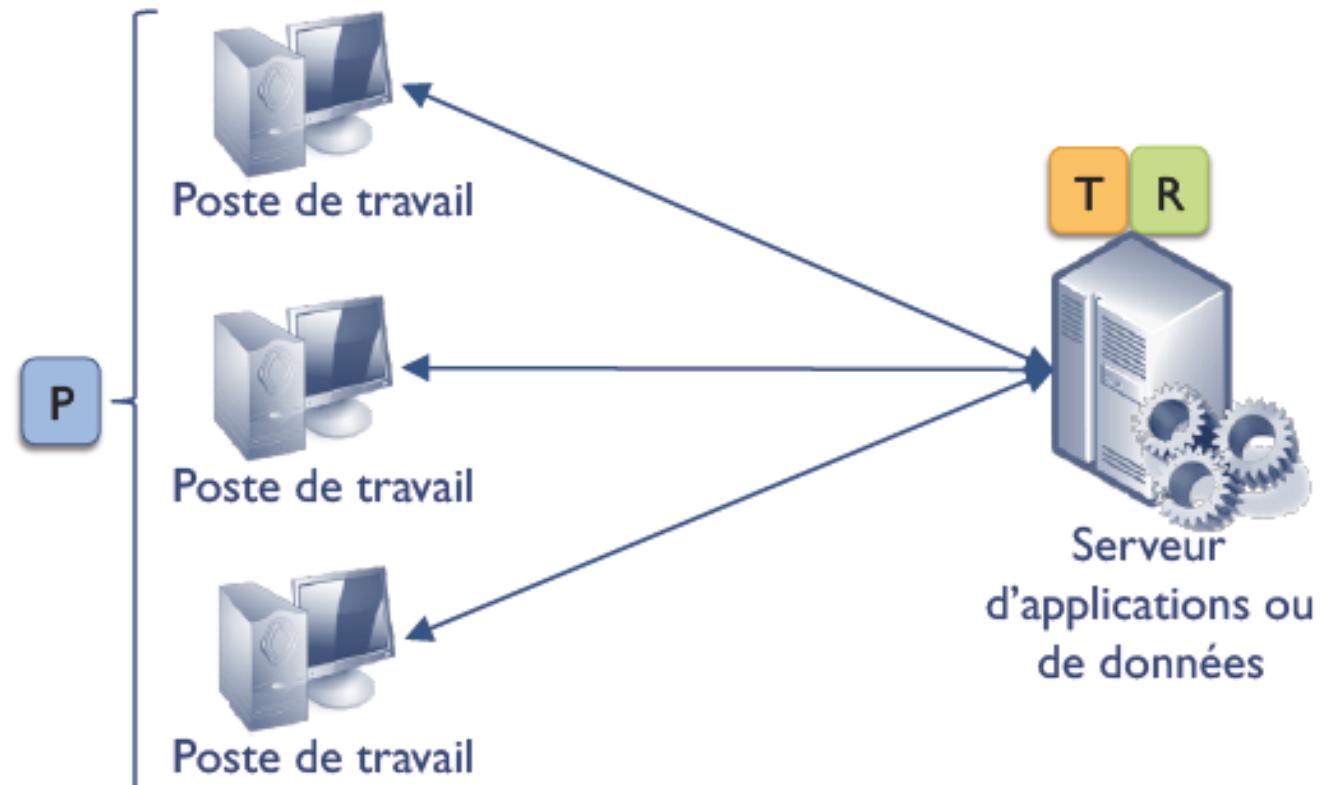
Exemple :



Architecture 2-tiers (Client Lourd)

- Le cœur de l'application est sur un serveur (éventuellement distant)
- La couche présentation (IHM) de l'application est sur le client

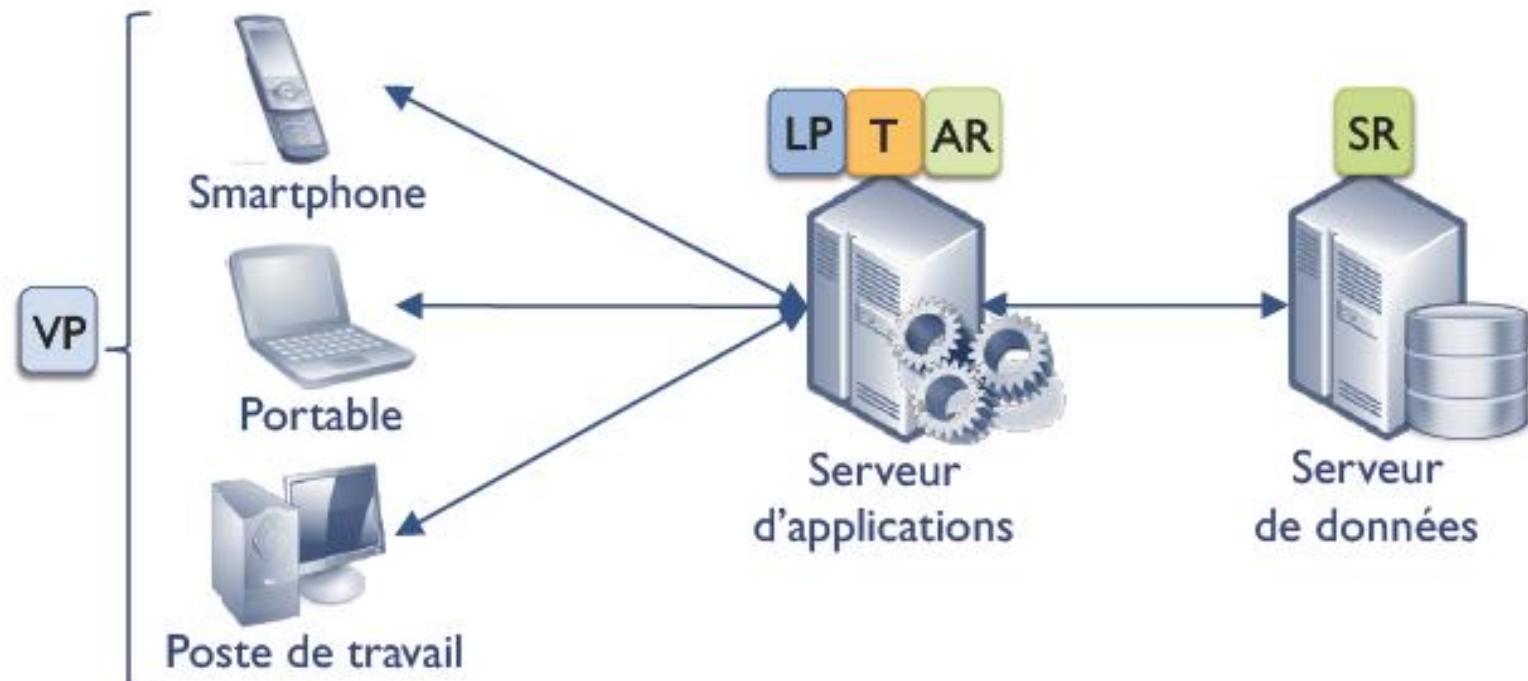
Exemple :



Architecture 3-tiers (Client Léger)

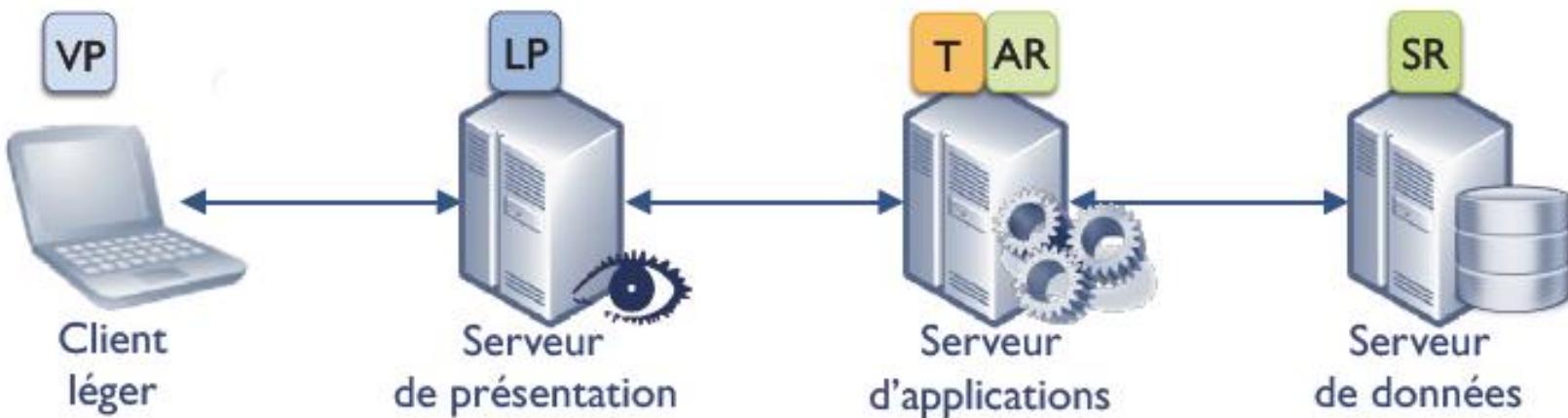
- Le cœur de l'application est sur un serveur
- Les données sont sur un autre serveur
- Le client est une application « légère » de visualisation (ex : navigateur web)

Exemple :



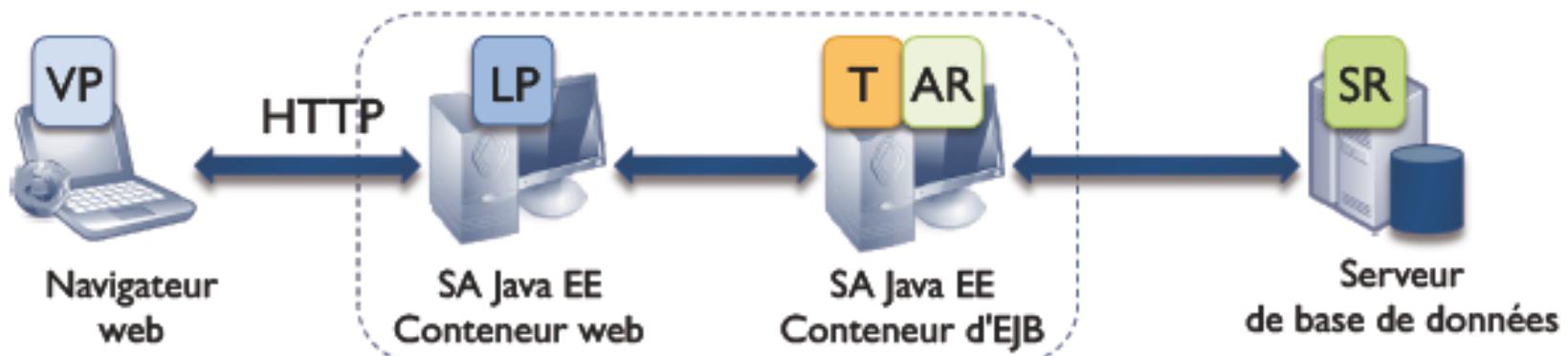
Architecture n-tiers

- Généralisation des modèles précédents (remarque : un serveur peut être un client pour un autre serveur !)
- Distribution des responsabilités en 4 ou + tiers
- Architecture type :

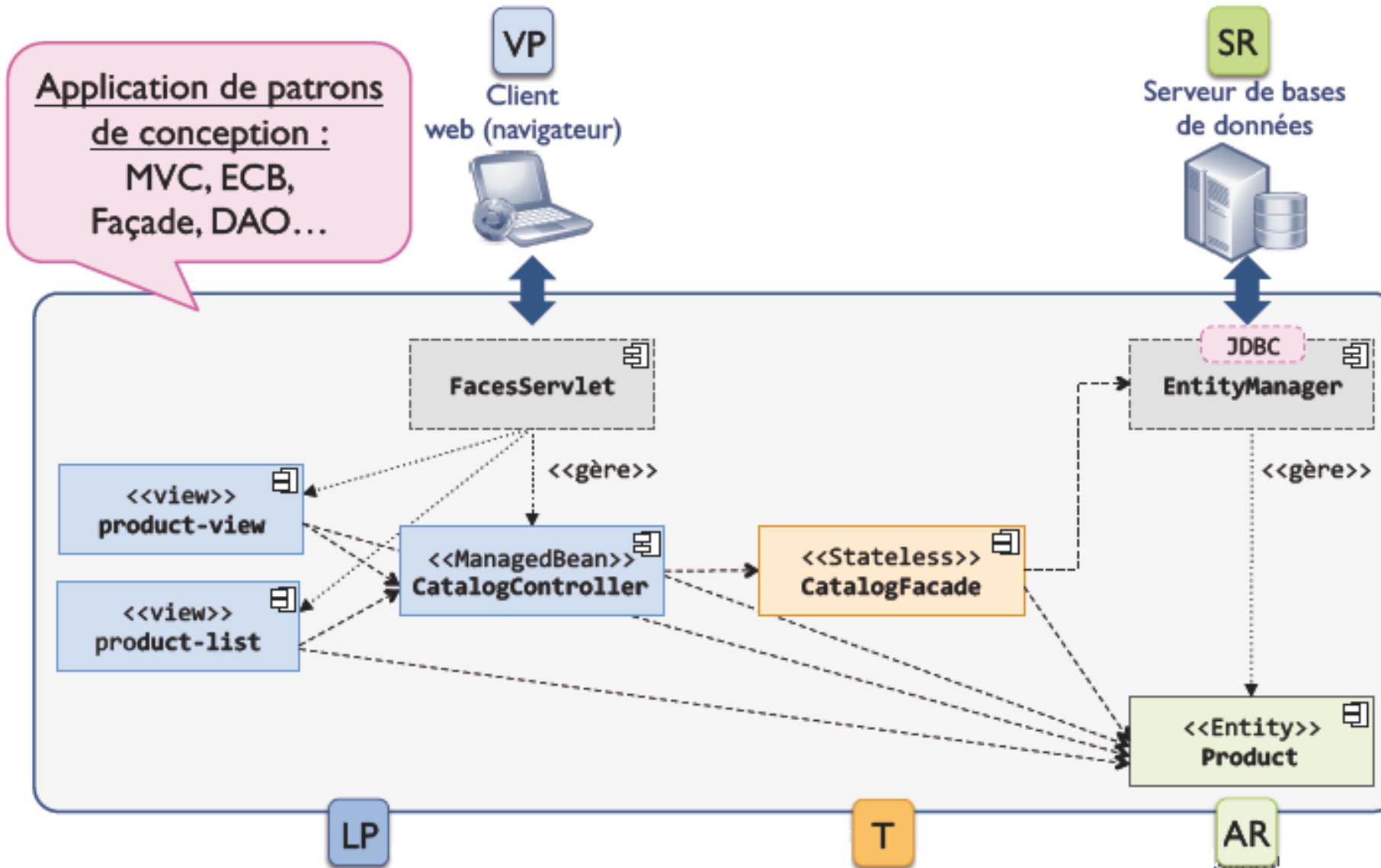


Exemple avec Java EE

- Application de gestion de catalogue de produits
- Architecture 3 ou 4 tiers :
 - Client léger
 - Serveur de présentation (pouvant être fusionné avec le serveur de traitement dans le cas de Java EE)
 - Serveur d'application
 - Serveur de données



Gestion de catalogue avec Java EE



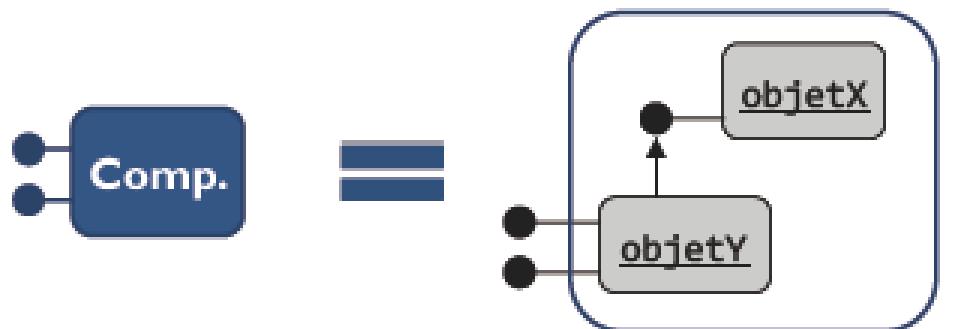
Composants

Composant = Unité logique de traitement

- Assemblage d'objets interdépendants
- Rend un service (fonction)
- Vue boîte noire

Propriétés

- **Identification** : nom unique, référencé dans un annuaire
- **Indépendance** : utilisable tout seul
- **Réutilisation** : utilisable dans différents contextes
- **Intégration** : combinable avec d'autres composants



Technologies d'implémentation multiples

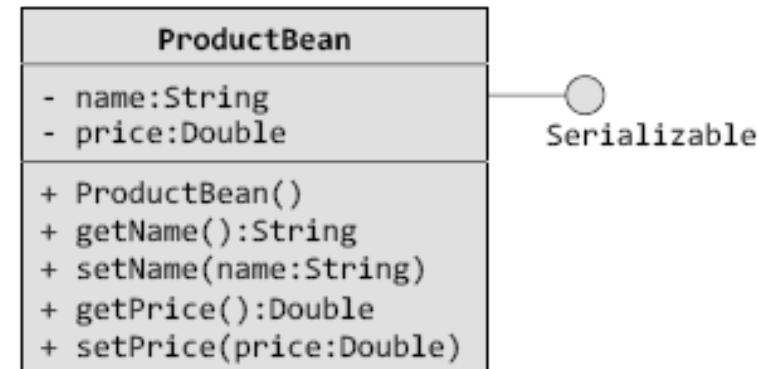
Exemple avec Java : JavaBeans

- Composant implémenté par une classe Java
 - ≈ Plain Old Java Object (POJO)
- Conventions à respecter
 - Sérialisation
 - Constructeur par défaut
 - Propriétés privées avec accesseurs (encapsulation et introspection)
 - Public <returntype> *getPropertynname()*
 - Public void *setPropertynname(parameter)*
 - Méthodes d'interception d'événements
 - Utilisation d'écouteurs et génération d'événements
 - Ex : PropertyChangeListener



Exemple avec Java : JavaBeans

```
public class ProductBean implements Serializable {  
    private static final long serialVersionUID = 1L;  
  
    private String name;  
    private Double price;  
  
    public ProductBean() {  
        this.name = "";  
        this.price = 0.0;  
    }  
  
    public String getName() {  
        return this.name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public Double getPrice() {  
        return this.employed;  
    }  
    public void setPrice(Double price) {  
        this.price = price;  
    }  
}
```

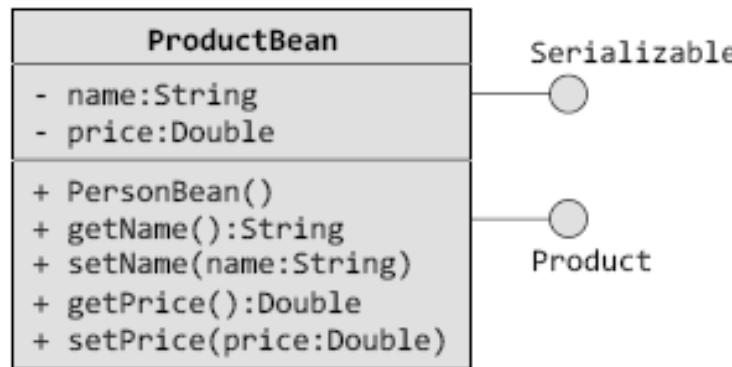


OU

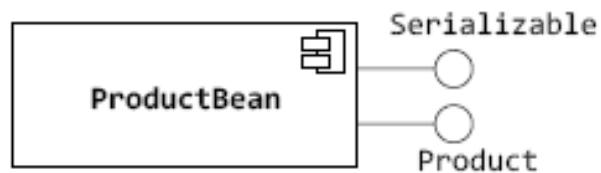


Interfaces d'un JavaBeans

```
public interface Product {  
    public String getName();  
    public void setName(String name);  
    public Double getPrice();  
    public void setPrice(Double price);  
}
```



OU



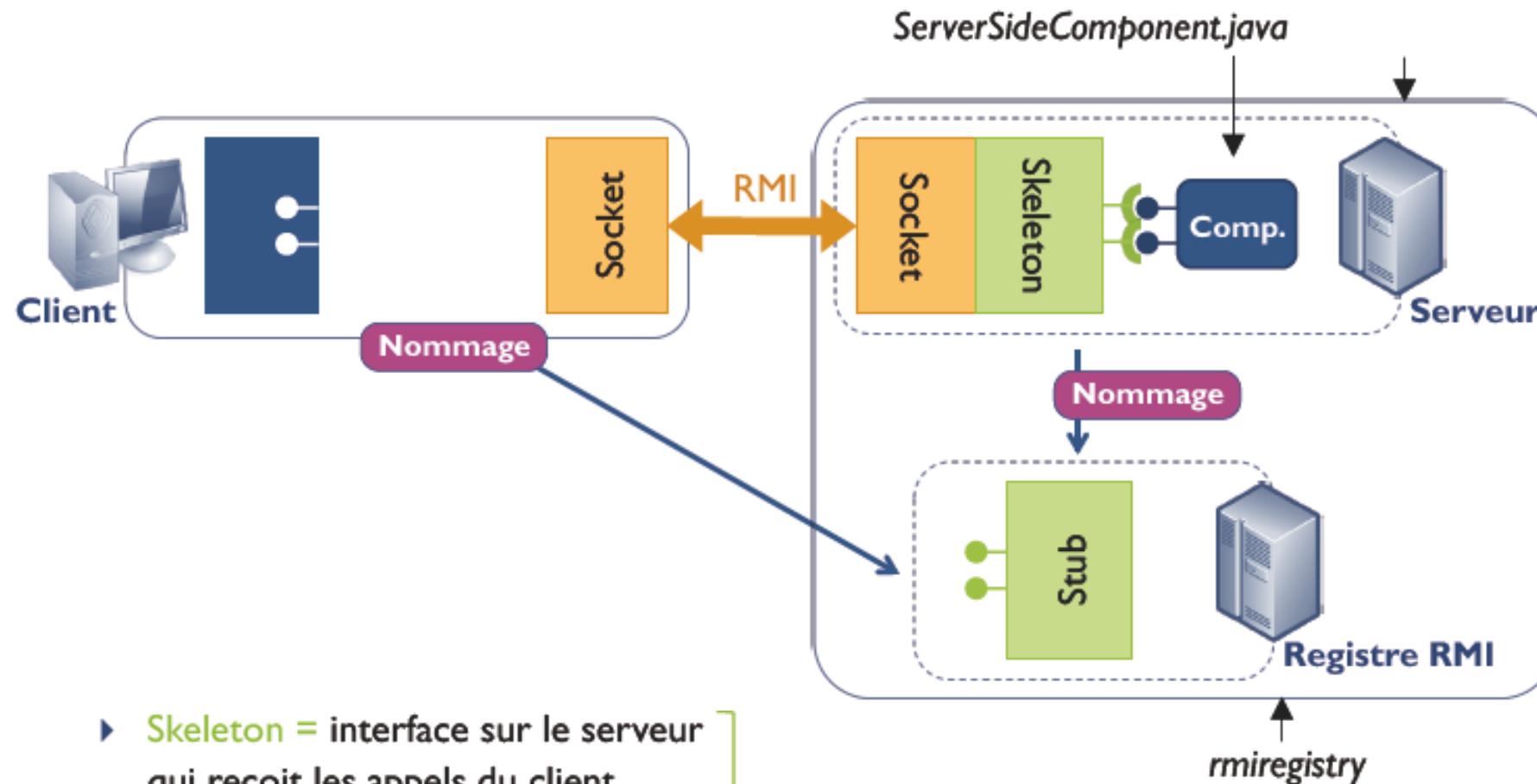
```
public class ProductBean implements Product,  
                                Serializable {  
  
    private String name;  
    private Double price;  
  
    public ProductBean() {  
        this.name = "";  
        this.price = 0.0;  
    }  
  
    public String getName() {  
        return this.name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public Double getPrice() {  
        return this.employed;  
    }  
    public void setPrice(Double price) {  
        this.price = price;  
    }  
}
```

Composants distribués

Un Client veut utiliser un composant qui se trouve sur un Serveur (distant)



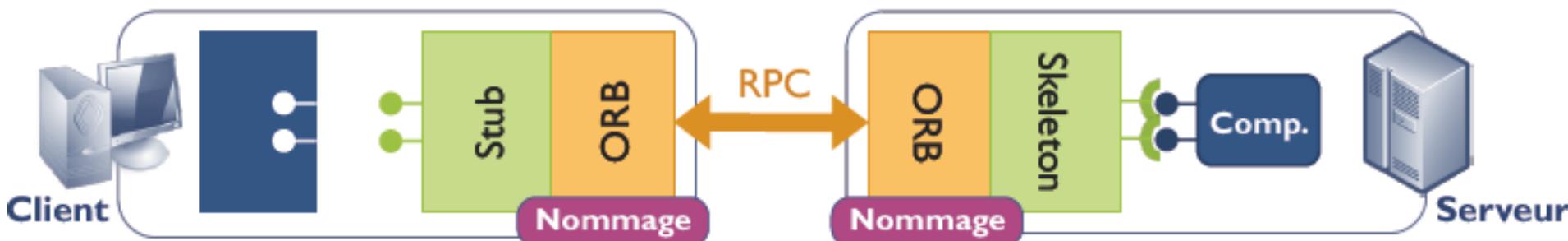
Solution Java RMI (Remote Method Invocation)



- ▶ **Skeleton** = interface sur le serveur qui reçoit les appels du client
 - ▶ **Stub** = interface sur le client qui envoie les appels au serveur
- dérivés du composant

Solution CORBA

Un Client veut utiliser un composant qui se trouve sur un Serveur (distant)



- ▶ **Object Request Broker (ORB)** = « bus logiciel » qui permet au client de rechercher le composant sur le serveur et de communiquer avec lui
 - ▶ **Skeleton** = interface sur le serveur qui reçoit les appels du client
 - ▶ **Stub** = interface sur le client qui envoie les appels au serveur
 - ▶ **Remote Procedure Call (RPC)** = appel de procédure distant
-] dérivés du composant

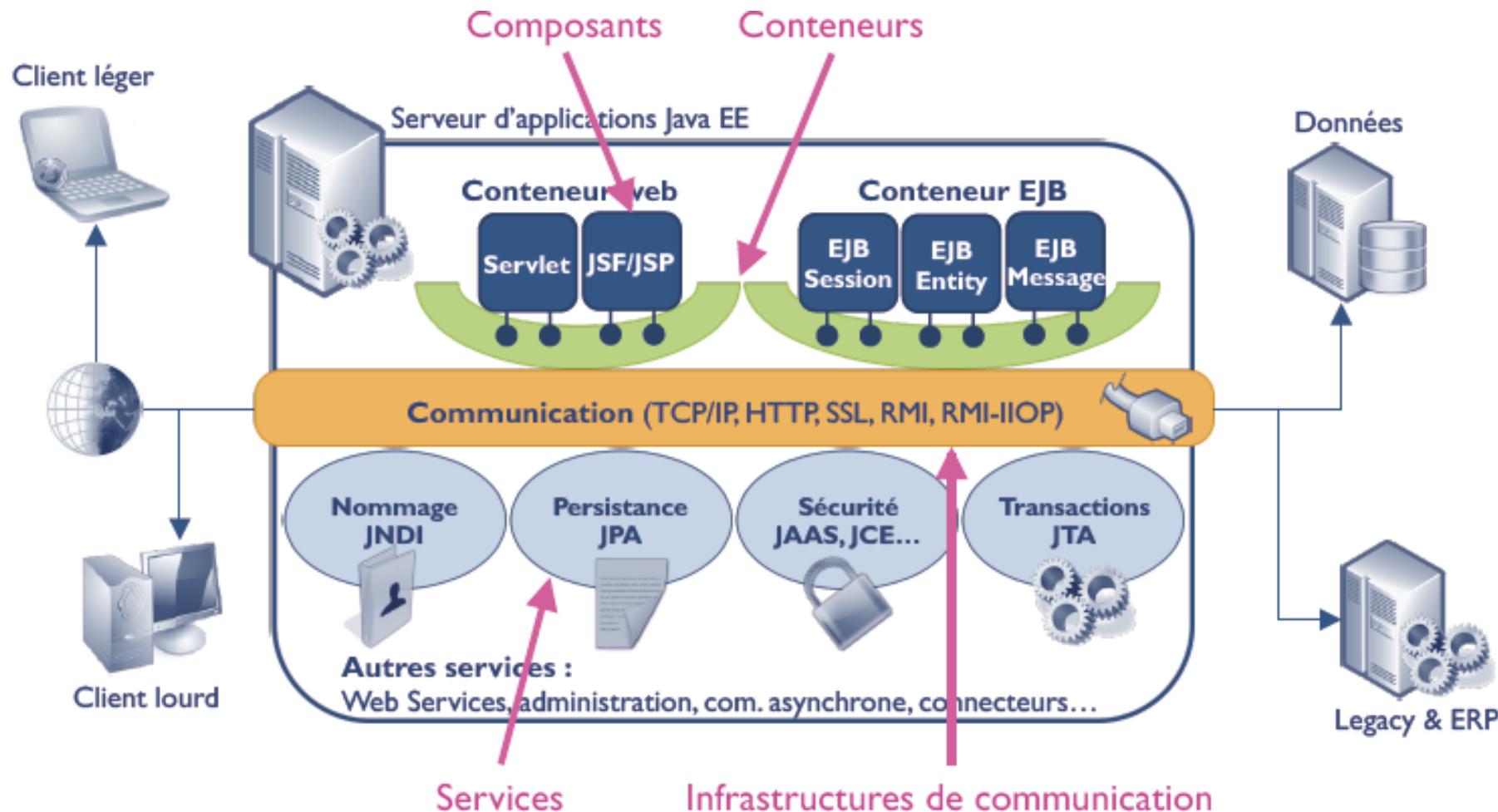
Problématiques

- Applications n-tiers à base de composants = **composants distribués avec responsabilités distribuées**
- Problématiques :
 - **Complexité**
 - Conception des composants et des applications
 - Développement des composants et des applications
 - Gestion des **aspects transverses** : sécurité, disponibilité, communication, persistance, transactions...
 - **Interopérabilité** des composants
 - **Administration** des composants et des applications
 - **Sécurisation** de bout en bout des applications
 - ...

Serveurs d'application & frameworks de développement

- Serveur d'Applications (SA) = conteneur et fournisseur de services pour des composants et des applications
 - Gestion du cycle de vie des applications et des composants
 - Administration des applications et des composants
 - Allocation de ressources
 - Processeur, mémoire, réseau, composants logiciels externes...
 - Support pour les aspects transverses
 - Sécurité, gestion des transaction, accès réseau...
 - Support pour l'interopérabilité
- Frameworks de développement =
 - Cadres pour la conception et le développement de composants (déployés sur SA) et d'applications à base de composants

Exemple : SA + framework Java EE



Sommaire

① Applications d'entreprise

- Etat des lieux des SI
- Modèle de référence

② Patrons d'architecture pour les applications

- Architecture en couches
- Modèle n-tiers
- Composants
- Infrastructures logicielles

③ Flux

- Typologie
- Qualification des flux

④ Architectures d'échange

- Typologie des architectures
- Solutions logicielles

Echanges d'informations ?

- Flux = données qui passent d'un point A à un point B
 - D'une application à une autre,
 - D'un module applicatif à un autre
 - D'un utilisateur à un autre
 - D'une base de données à une autre
 - D'une entreprise à une autre
 - ...
- Exemples de flux
 - Transfert de fichier
 - Partage de fichier
 - Appel de procédure distant
 - Requête sur une base de données
 - ...

Périmètre

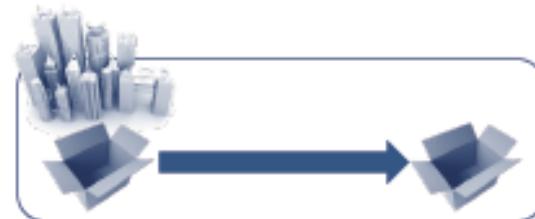
- ▶ Flux **privé** = intra-application
(entre composants)



- ▶ Flux **public** = inter-applications
 - ▶ Bonne gestion des flux publics = flexibilité !



- ▶ Flux **AtoA** = flux inter-applications sur un périmètre **intra-entreprise**



- ▶ Flux **BtoB** = flux inter-applications sur un périmètre **inter-entreprises**
 - ▶ Exemple : envoi d'une commande de pièce à un fournisseur



Granularité / Fréquence

« Evénementiels »

- ▶ Flux **unitaire** = données transmises une à une



- ▶ Flux **au fil de l'eau** = données transmises dès qu'elles sont disponibles



Exemple : transmission des commandes à la plate-forme logistique au fur et à mesure de leur validation

« Batch »

- ▶ Flux de **masse** = données regroupées en lots



- ▶ Flux **cadencés** = données transmises à des moments prédéterminés



Exemple : transmission chaque soir des données concernant l'ensemble des ventes de la journée pour stockage dans l'entrepôt de données

Exemples de flux

- Souvent pour les **flux unitaires au fil de l'eau** (« événementiels ») :
 - Appels distants entre composants (CORBA, RMI...)
 - Transferts de fichiers
 - Partage de base de données
 - **Electronic Data Interchange (EDI)** = norme définissant le(s) protocole(s) + le format d'échange de données pour le B2B
 - **Web Services**
- Souvent pour les **flux de masse cadencés** (« batch ») :
 - Transferts de fichiers
 - Partage de base de données
 - **Batch** = script qui ordonne et cadence un déplacement de données en volume
 - Solution « historique » et sans doute encore l'une des plus utilisées
 - **Extract-Transform-Load (ETL)** = progiciel de batch « à grande échelle » permettant de connecter des entrepôts de données

Modalité

- Flux **synchrone** = **bloquant pour l'émetteur et le récepteur**
 - Suppose la disponibilité de l'émetteur et du récepteur au même moment
 - *Exemple : appel de méthode RMI*
- Flux **asynchrone** = **non bloquant** (émission / réception différées)
 - Suppose l'existence d'une zone de stockage intermédiaire
 - *Exemple : emails*
- Flux **requête-réponse** = l'émetteur et le récepteur se connaissent
 - Contact direct
 - *Exemple : récupération d'une donnée dans un référentiel*
- Flux **publication-abonnement** = les récepteurs s'abonnent aux flux sans connaître les émetteurs
 - Contact indirect
 - *Exemple : abonnement aux mises à jour d'un référentiel de données*

Sommaire

① Applications d'entreprise

- Etat des lieux des SI
- Modèle de référence

② Patrons d'architecture pour les applications

- Architecture en couches
- Modèle n-tiers
- Composants
- Infrastructures logicielles

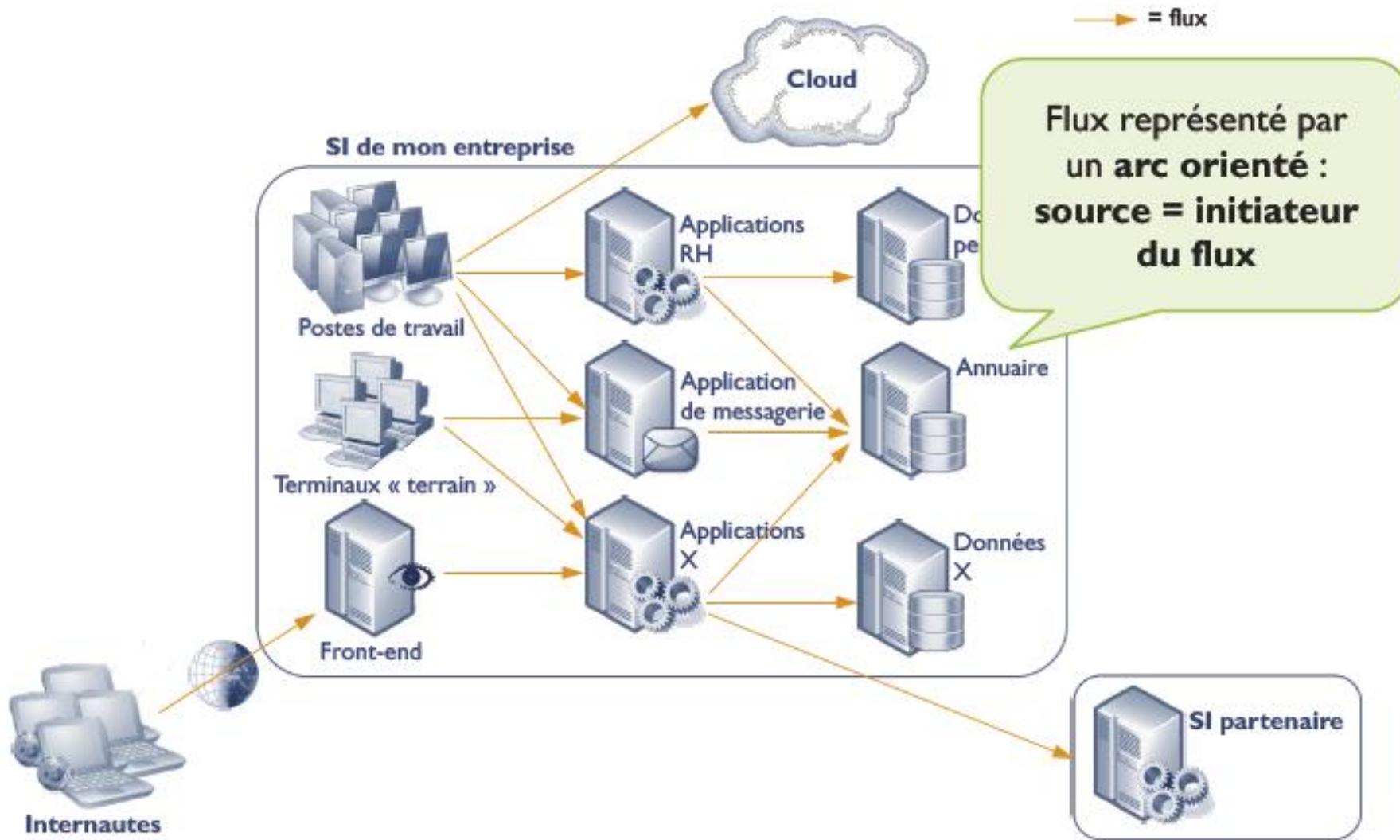
③ Flux

- Typologie
- Qualification des flux

④ Architectures d'échange

- Typologie des architectures
- Solutions logicielles

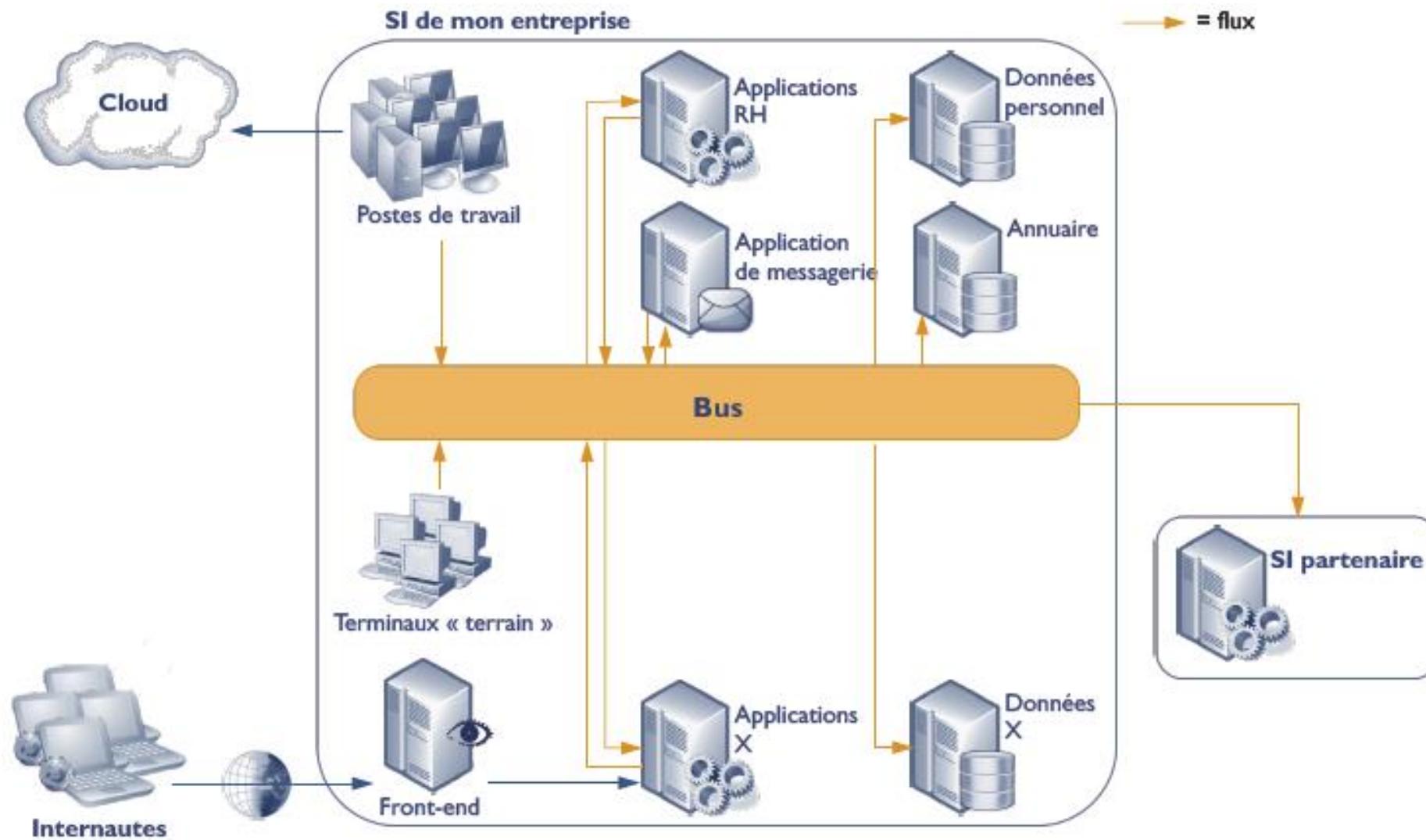
Architecture point-à-point



Analyse des solutions point-à-point

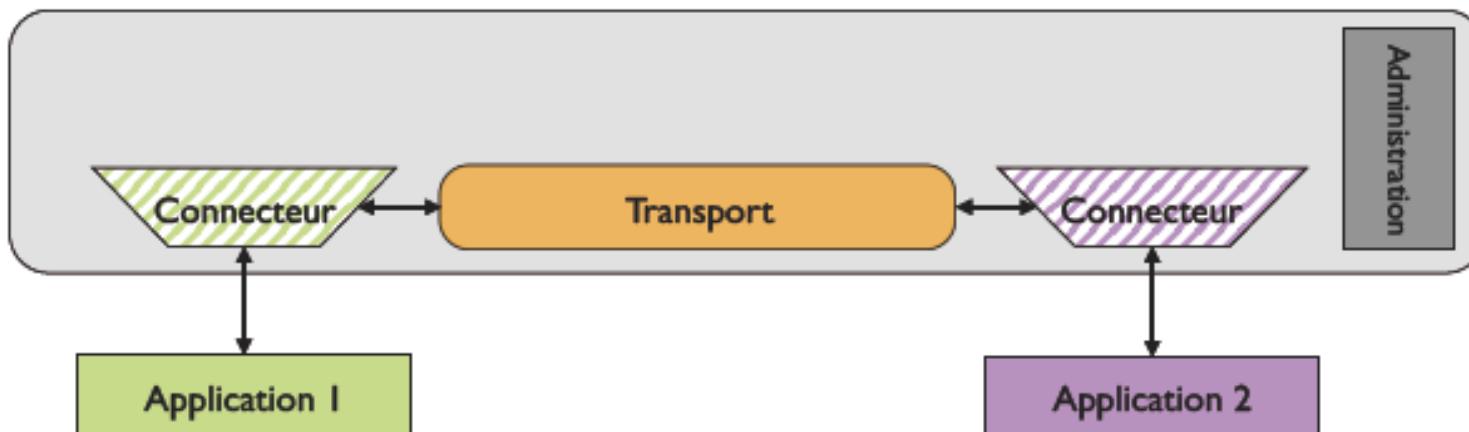
- Architecture « **intuitive** »
- Simplicité de mise en œuvre dans le cas où le nombre d'applications à intégrer est faible
- **Efficacité** des échanges directs
- **Problème de passage à l'échelle**
 - Si N applications, $N(N-1)/2$ liens...
 - Effet « plat de spaghetti »
- **Evolutivité très réduite**
 - Intégration d'une nouvelle application = ajout de nombreux nouveaux liens
 - **Couplage fort entre les applications**
 - fort impact des évolutions (notamment interfaces)
- **Exploitation et administration complexes**
 - Manque de visibilité sur les échanges

Architecture bus



Exemple de solution de type bus : le MOM

- **Middleware Orienté Message (MOM)** = bus logiciel de transport qui permet à des applications de recevoir des messages émis par d'autres
 - **Connectivité** : supporte différents protocoles de communication
 - **Transport** :
 - Garantit l'acheminement (intégrité, gestion des erreurs)
 - Gère différentes modalités : synchrone/asynchrone, publication/abonnement...
 - Gère les transactions
 - Existe aujourd'hui sur la plupart des serveurs d'applications



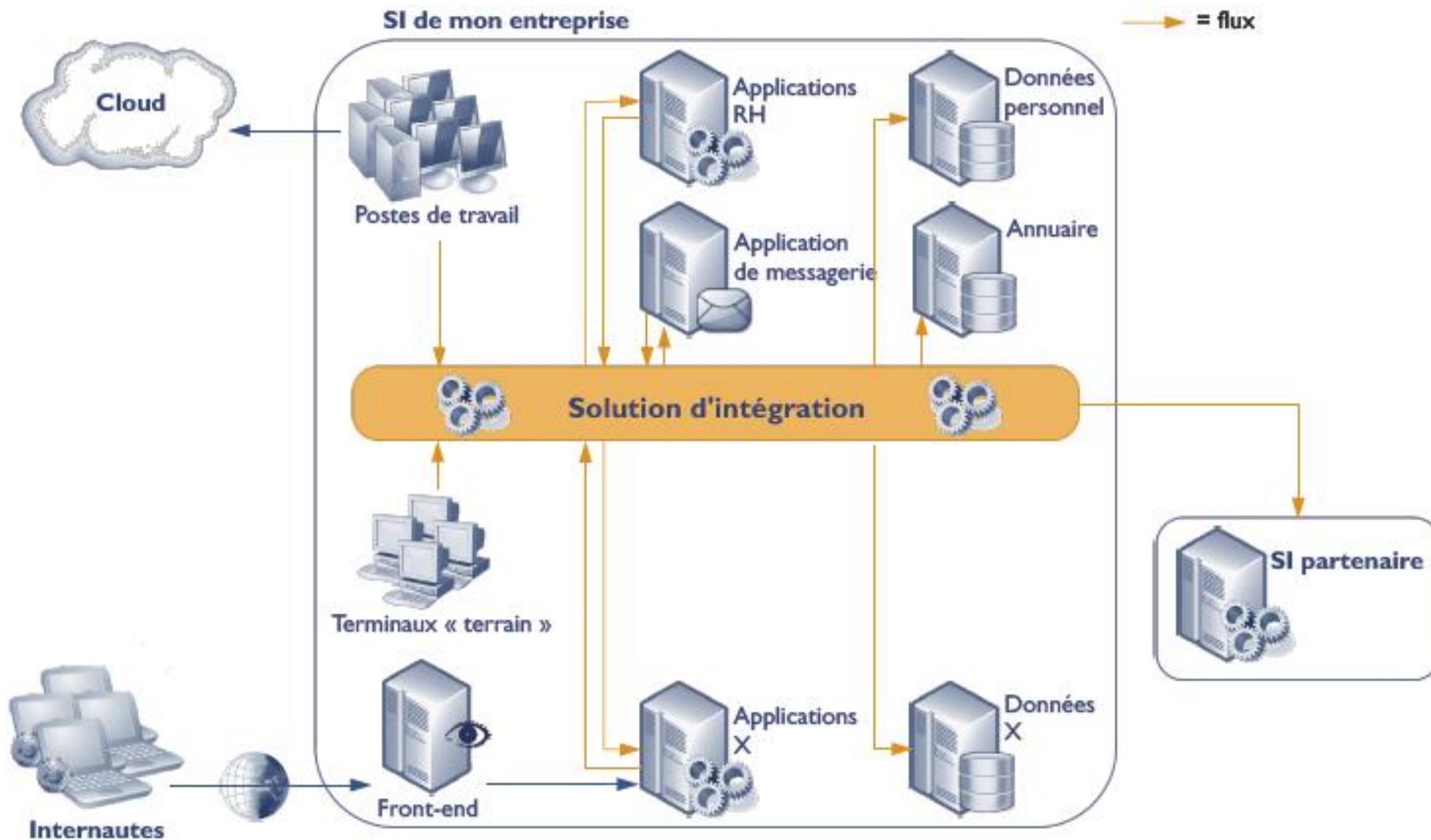
Exemple avec Java EE : JMS et EJB Message

- JMS (Java Message Service) = interface Java standard pour les MOM
 - Files d'attentes (queues) *pour le mode requête / réponse*
 - Sujets (topics) *pour le mode publication / abonnement*
- EJB Message = composant invoqué par messages
 - Traite les messages postés dans une file / sujet
 - Poste des messages réponse dans la file / sujet

Analyse des solutions de type bus

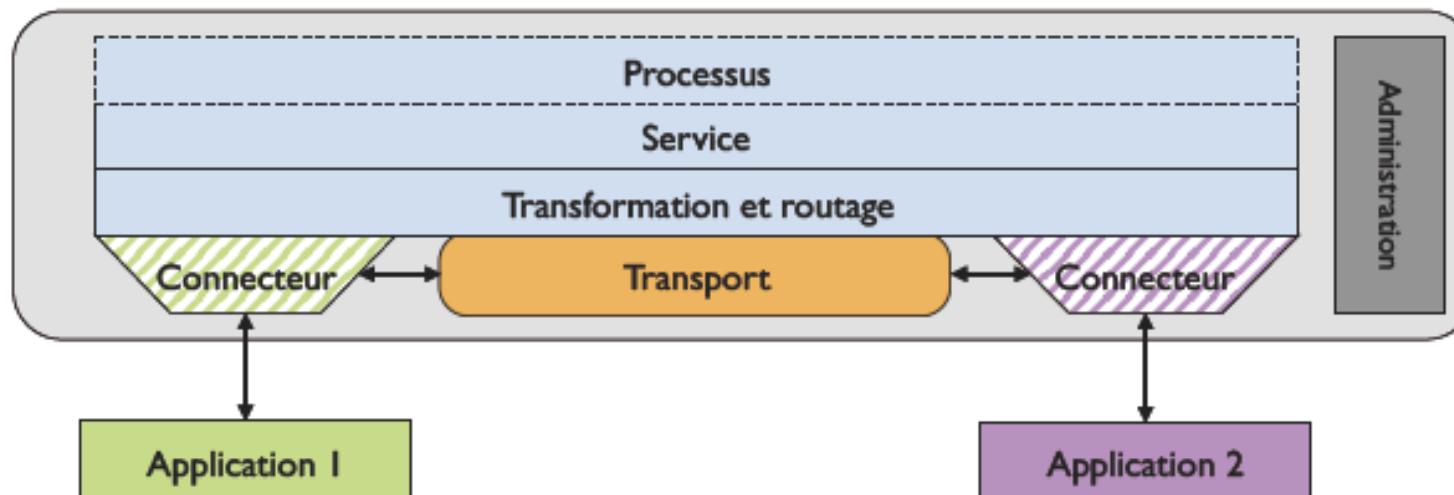
- Passage à l'échelle facilité
 - Si N applications, au plus N liens bidirectionnels
- Meilleure évolutivité
 - Intégration d'une nouvelle application = un seul connecteur
- Couplage faible entre les Applications
- Services de transport
 - Acheminement garanti des données (reprise sur erreur, gestion des doublons)
 - Intégrité des données
- Adhérence forte entre les applications et le bus
- Couplage fort entre les formats des données
 - fort impact des évolutions du format d'échange
- Plate-forme centralisée
 - hautement critique
 - goulot d'étranglement

Architecture intégrée



Exemple de solution intégrée : l'EAI

- Enterprise Application Integration (EAI) = progiciel d'intégration inter-applicative
- Connectivité & Transport
- Transformation : gère l'hétérogénéité des formats et des valeurs
- Routage : adresse intelligemment les données aux différents destinataires
- Service : encapsule la logique d'intégration
- + Eventuellement, processus : orchestre les services d'intégration



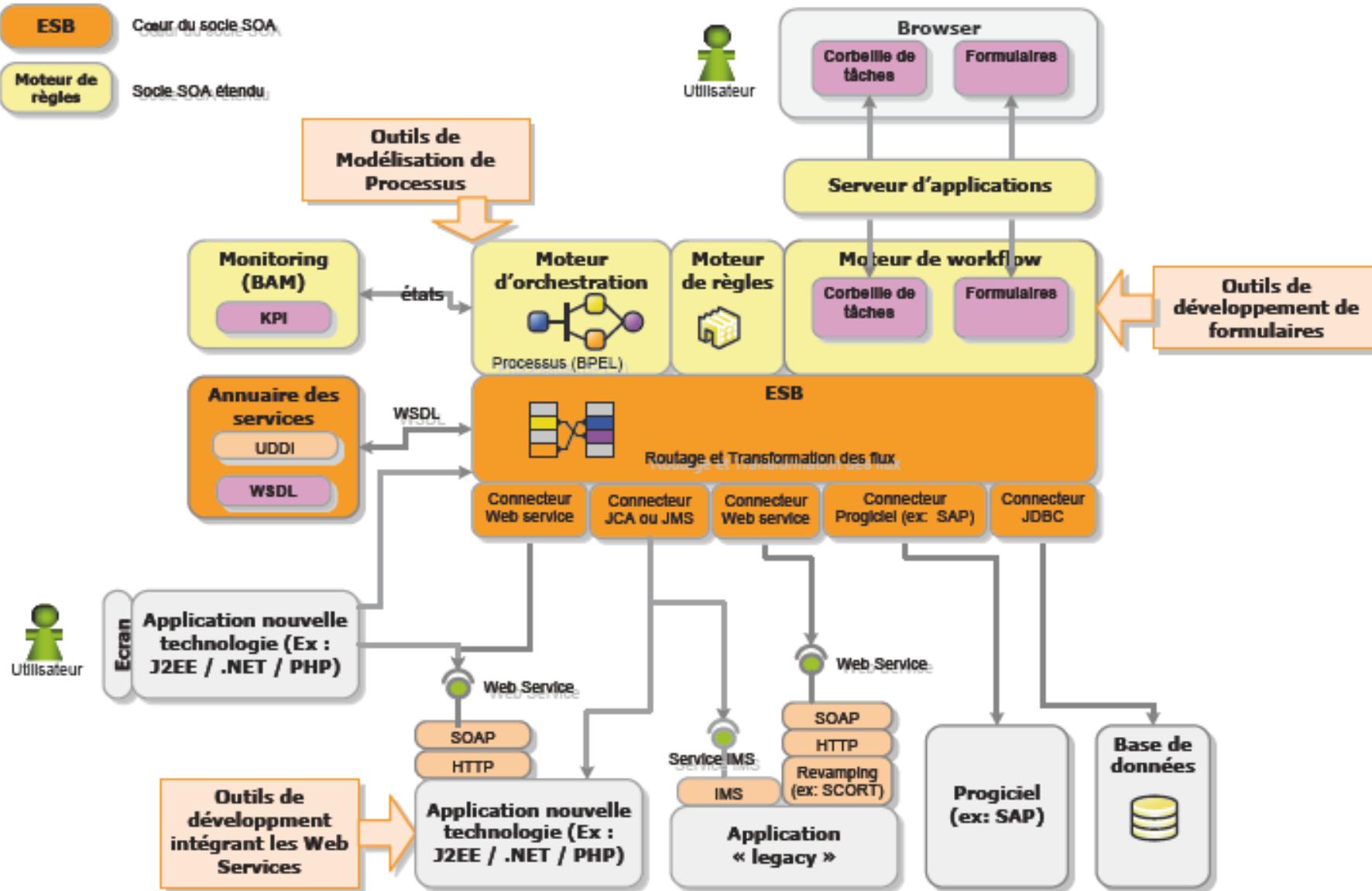
Analyse des solutions EAI

- Relations entre processus métiers et échanges inter-applicatifs plus lisibles
 - *Urbanisation fonctionnelle*
 - + *Urbanisation technique*
- Services applicatifs riches
- Coûts d'administration moins importants
- Coûts de développement réduits
- Important travail **d'urbanisation** et /ou de réorganisation fonctionnelle **indispensable**
 - *Sinon « plat de spaghetti » dans l'outil...*
- Experts indispensables (et malheureusement très rares)
- Projets transverses par essence
 - Difficulté à établir les responsabilités
 - Problématiques organisationnelles (nombreux acteurs, besoin de processus)
- **Technologies d'interconnexion propriétaires**

Autre solution intégrée : ESB

- Enterprise Service Bus (ESB) ≈ EAI basé sur les standards
 - Connectivité & Transport
 - Transformation : gère l'hétérogénéité des formats et des valeurs
 - Routage : adresse intelligemment les données aux différents destinataires
 - Service : encapsule la logique d'intégration
 - Processus : orchestre les services d'intégration
 - Monitoring : supervise le déroulement des processus
- L'ESB est considéré comme le socle technique de l'approche SOA

Enterprise Service Bus (ESB)



Architecture Orientée Services Concepts

ECOLE NATIONALE POLYTECHNIQUE D'ORAN
Département Mathématiques et Informatique
Filière IMSI : 4^{ème} année ingénieur

M. SABRI
2017-2018

Introduction

L'architecture orientée services offre la possibilité de coupler des composants dans plusieurs configurations dans la structure d'une plateforme et de les réutiliser pour différentes constructions ce qui augmente considérablement la flexibilité du système d'informations; se sont les principaux avantages du SOA.

L'interopérabilité et la cohérence sont obtenues lorsque vous obtenez un système qui répond aux besoins d'évolution des systèmes d'informations.

Définitions

Une Architecture Orientée Service (SOA) est une Architecture technico-fonctionnelle dans laquelle les fonctions réutilisables du SI sont modélisées et exposées via des standards pour contribuer à la réalisation des Processus Métier .

- La SOA est avant tout une démarche de conception contribuant au besoin d'urbanisation du SI, sans pour autant être l'apanage d'une technologie.
- Pour les équipes métier, la SOA permet d'être plus réactif et rapide dans l'innovation de modèles et des processus pour créer des produits à moindre coût en se dotant d'un avantage concurrentiel et en optimisant la collaboration interne et externe à l'entreprise.
- Pour les équipes IT, la SOA a pour but de créer une réelle interopérabilité entre les différents silos applicatifs du SI et de faciliter l'ouverture du SI aux partenaires de l'entreprise.

Définitions

L'architecture SOA est définie comme un style architectural qui permet de construire des solutions d'entreprises basées sur les *services*.

SOA a introduit une nouvelle philosophie pour le développement des applications distribuées, où les **services** peuvent être **publiés**, **découverts**, **composés**, **réutilisés**, et **invoqués** en utilisant l'interface, indépendamment de la technologie utilisée pour implémenter chaque service.

SOA permet de **décomposer** les fonctionnalités d'un système ou d'une application **en un ensemble de services**.

Rassembler les services pour créer les applications de l'entreprise (dites **applications composites**).

Définitions

L'avantage le plus important de la SOA est qu'elle permet de séparer l'implémentation du service de son interface.

C'est cette caractéristique qui assure le haut degré d'interopérabilité visé par cette architecture.

Bénéfices attendus de la SOA

Un objectif : maîtriser et optimiser les coûts d'intégration

Deux types de bénéfices :

- Des bénéfices intrinsèques à la mise en œuvre d'une SOA
- Des bénéfices indirects de la mise en œuvre d'une SOA à grande échelle
 - Il s'agit des opportunités exploitables dans le cadre de la mise en œuvre de la SOA

Bénéfices attendus de la SOA

Des bénéfices intrinsèques

- Favoriser la **mutualisation** des fonctions du SI
 - Réutilisation des composants métier existants
 - Création de services métier réutilisables
- Contribuer à **maitriser les coûts d'intégration et de maintenance applicative**
 - Réutilisation des services → mutualisation des coûts de maintenance
 - De plus, les facilités offertes par les plateformes d'intégration SOA (ESB) doivent permettre de réduire les délais d'intégration
- Améliorer la **réactivité** et la **qualité** des développements
 - Accélérer le processus de développement de nouvelles applications
 - Fiabiliser les applications offertes aux différents métiers (la réutilisation permettant de mieux éprouver les systèmes existants)
- Favoriser le recentrage de la conception des applications autour des **processus métiers**

Bénéfices attendus de la SOA

Des bénéfices indirects

- Favoriser une plus grande **standardisation** du SI
 - Permet une meilleure capacité d'ouverture du SI, capacité d'intégration d'environnements hétérogènes
 - Et par là favoriser la productivité des filières de développement
- Favoriser la mise en place de mesure de **qualité de service** rendu par le SI
 - La SOA s'accompagne de la définition de « contrats de service » que sont capables de supporter les nouvelles infrastructures (ESB, Annuaire de service, etc.)
- Permettre au SI de **s'ouvrir vers ses principaux partenaires** (filiale & SI externes)
 - En proposant une infrastructure et des services spécifiques exposés à l'extérieur
- Améliorer la **disponibilité des informations**
 - En amenant le SI depuis une architecture Batch avec de traitements nocturnes lourds vers une architecture en mode de traitement au fil de l'eau / asynchrone plus souple

Les risques liés à la mise en œuvre d'une SOA

Des préoccupations liées aux modèles organisationnelle et méthodologique actuels des DSIs :

- La gestion d'un lourd changement au niveau des collaborateurs ou des processus (en particulier de développement)
- Un manque de support/compréhension de la part des métiers
 - Mutualiser de manière efficace exige de modular un certain nombre de fonctionnalités spécifiques. Les MOA doivent le comprendre et l'accepter.
 - Une sensibilisation des métiers aux enjeux de la SOA est nécessaire
- L'adoption d'une démarche « services » a un impact certain sur la gestion des projets
 - Les nouveaux projets applicatifs éligibles à une approche « services » doivent être identifiés au plus tôt dans le cycle de vie des projets.

Les risques liés à la mise en œuvre d'une SOA

Des préoccupations liées aux modes de financement :

- Des **investissements lourds** sont nécessaires pour la mise en place de nouveaux composants logiciels (annuaires de services, bus de services, etc.)
- Les **modèles de financement** actuels des DSI (en mode projet) peuvent être un frein au déploiement de la SOA

Les craintes récurrentes

Des craintes liées aux impacts sur les architectures applicatives et techniques

- La mutualisation des ressources peut entraîner des **difficultés pour identifier les applicatifs impactés** par la panne d'un composant.
- Le modèle d'architecture distribué de la SOA rend plus difficile le **suivi d'un traitement de bout en bout**.
- Une dégradation possible des **performances** par l'ajout d'une couche logique de services supplémentaire.
- Des risques de sécurité notamment dans le **contrôle d'accès** aux services.

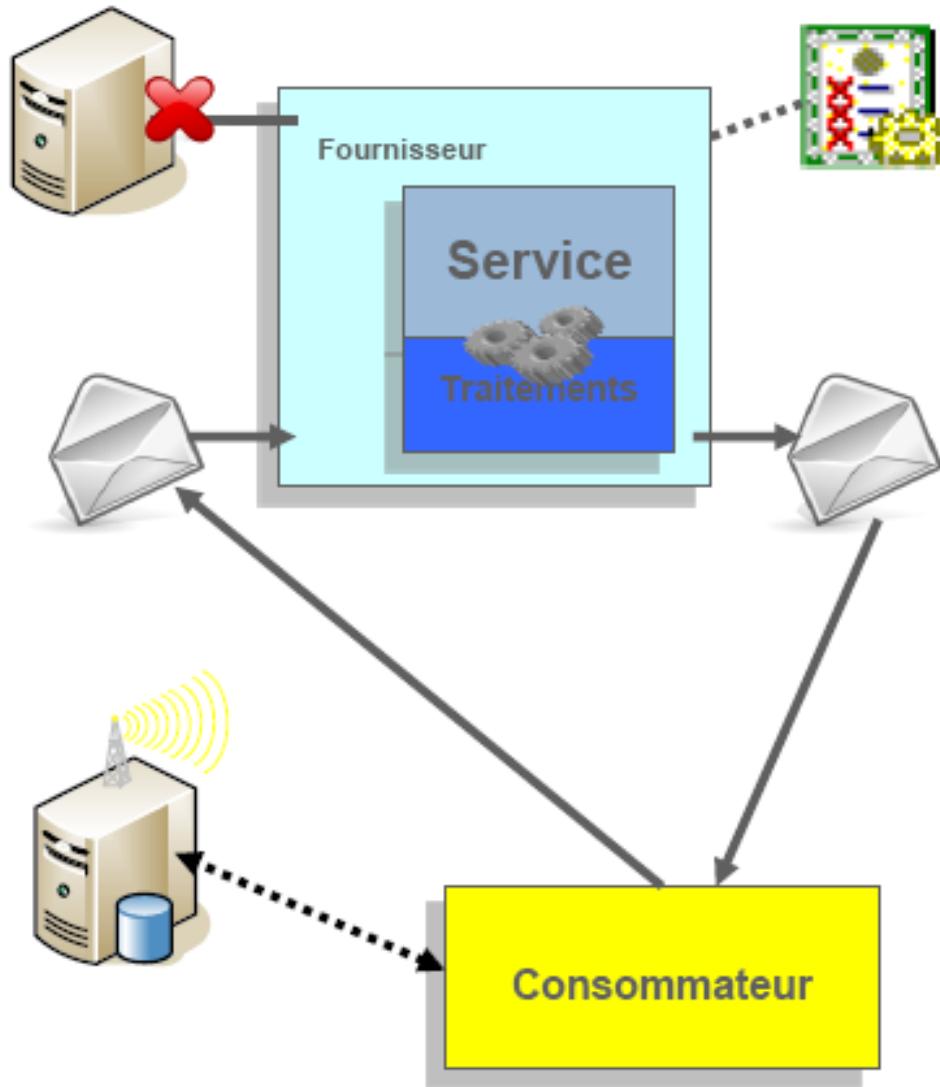
Définitions du Service

- Le service est le composant clef d'une architecture SOA.
- C'est une entité de traitement qui représente une fonction ou fonctionnalité bien définie qui est divisée en opérations.
- Les services interagissent et communiquent entre eux.
- Idéalement chaque service doit être indépendant des autres afin de garantir sa réutilisabilité et son interopérabilité

Définitions du Service

- A chaque service doit correspondre un contrat d'utilisation (contrat de service) qui permet à ses utilisateurs de comprendre son usage fonctionnel et technique.
- De plus, les données échangées en entrée/sorties des services doivent être décrite par un langage commun.
- Le service se doit d'avoir un propriétaire dûment identifié
- La pertinence de création d'un service doit être évaluée au cas par cas

Service vu du SI



➤ Un Service

- Effectue un ensemble de traitements qui répondent à un besoin donné
- Est exposé via une interface qui décrit un message en entrée et un autre en sortie
- Correspond à un niveau logique de traitement et pas à un niveau physique d'implémentation
- Garanti la stabilité de l'action qu'il effectue (contrat de service)

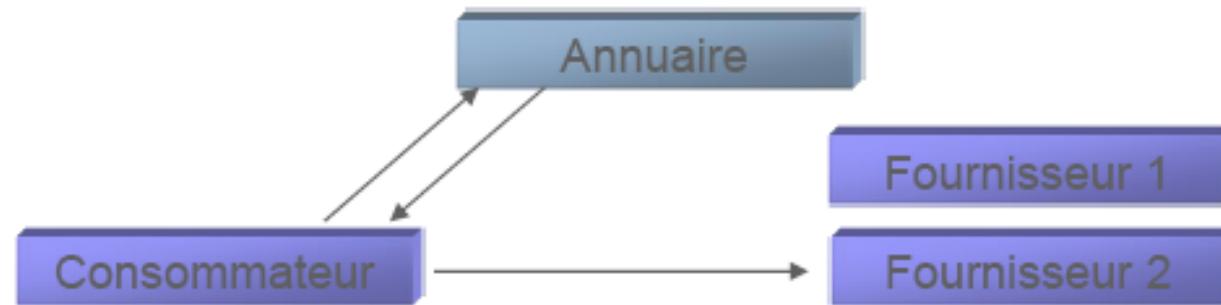
➤ Dans la SOA, la notion de service est associée à :

- Un découplage tant logique que physique entre le consommateur et le fournisseur
- Une hiérarchisation des services
- L'existence d'un engagement entre le fournisseur et le consommateur (le contrat de service)

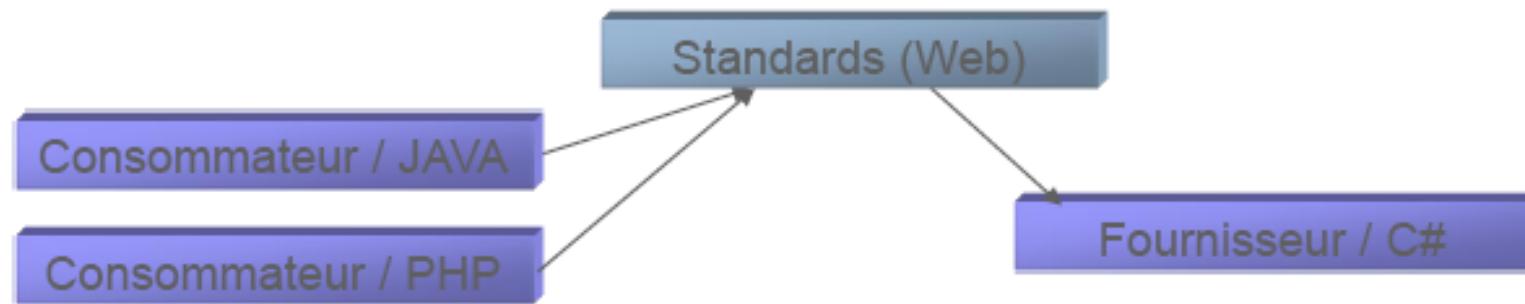
Une relation consommateur / fournisseur

➤ Découplage entre le fournisseur et le consommateur :

- Pas d'adressage direct



- Découplage technologique



La gestion du cycle de vie des services



- Identifier et concevoir les services
 - Méthodologie et Framework d'identification des services
 - Elaboration des contrats de service
 - Développer les services
 - Assurer l'homogénéité dans l'implémentation des services
 - Développer en vu de la réutilisation
 - Intégrer et déployer les services
 - Définir les règles de déploiement (sécurité, orchestration)
 - Exploiter et Superviser les services
 - Gestion des changements et du versioning
 - Gérer la qualité de services (SLA)
 - Pilotage des KPI (métier)
-
- ```
graph TD; subgraph SLC [Service Life Cycle]; A[Etude opportunité] --> B[Conception]; B --> C[Réalisation]; C --> D[Recette Déploiement]; D --> E[Exploitation]; end; subgraph Architecture [Architecture]; Eng[Engagement des Services]; SD[Services Design & Développement]; end; subgraph Development [Développement]; Comp[Composants]; IS[Integration des Services & Integrations]; end; subgraph Assembly [Assemblage & Déploiement]; CS[Contrats de services, Routage, Sécurité]; end; subgraph Management [Management]; MS[Management des Services]; Sup[Supervision]; end; C --> Eng; Eng --> SD; SD --> Comp; Comp --> IS; IS --> CS; CS --> Sup; Sup --> C; style SLC fill:#0070C0,color:#fff,stroke:#005090,stroke-width:2px; style Architecture fill:#D9E1F2,color:#005090,stroke:#005090,stroke-width:1px; style Development fill:#D9E1F2,color:#005090,stroke:#005090,stroke-width:1px; style Assembly fill:#D9E1F2,color:#005090,stroke:#005090,stroke-width:1px; style Management fill:#D9E1F2,color:#005090,stroke:#005090,stroke-width:1px; style C fill:#FFF,color:#005090,stroke:#005090,stroke-width:2px; style Eng fill:#FFF,color:#005090,stroke:#005090,stroke-width:1px; style SD fill:#FFF,color:#005090,stroke:#005090,stroke-width:1px; style Comp fill:#FFF,color:#005090,stroke:#005090,stroke-width:1px; style IS fill:#FFF,color:#005090,stroke:#005090,stroke-width:1px; style CS fill:#FFF,color:#005090,stroke:#005090,stroke-width:1px; style Sup fill:#FFF,color:#005090,stroke:#005090,stroke-width:1px; style C fill:#FFF,color:#005090,stroke:#005090,stroke-width:2px; style Eng fill:#FFF,color:#005090,stroke:#005090,stroke-width:1px; style SD fill:#FFF,color:#005090,stroke:#005090,stroke-width:1px; style Comp fill:#FFF,color:#005090,stroke:#005090,stroke-width:1px; style IS fill:#FFF,color:#005090,stroke:#005090,stroke-width:1px; style CS fill:#FFF,color:#005090,stroke:#005090,stroke-width:1px; style Sup fill:#FFF,color:#005090,stroke:#005090,stroke-width:1px;
```

La gouvernance du cycle de vie des services est un élément clé d'une démarche SOA

# Architectures orientées services (SOA)

# Web Services

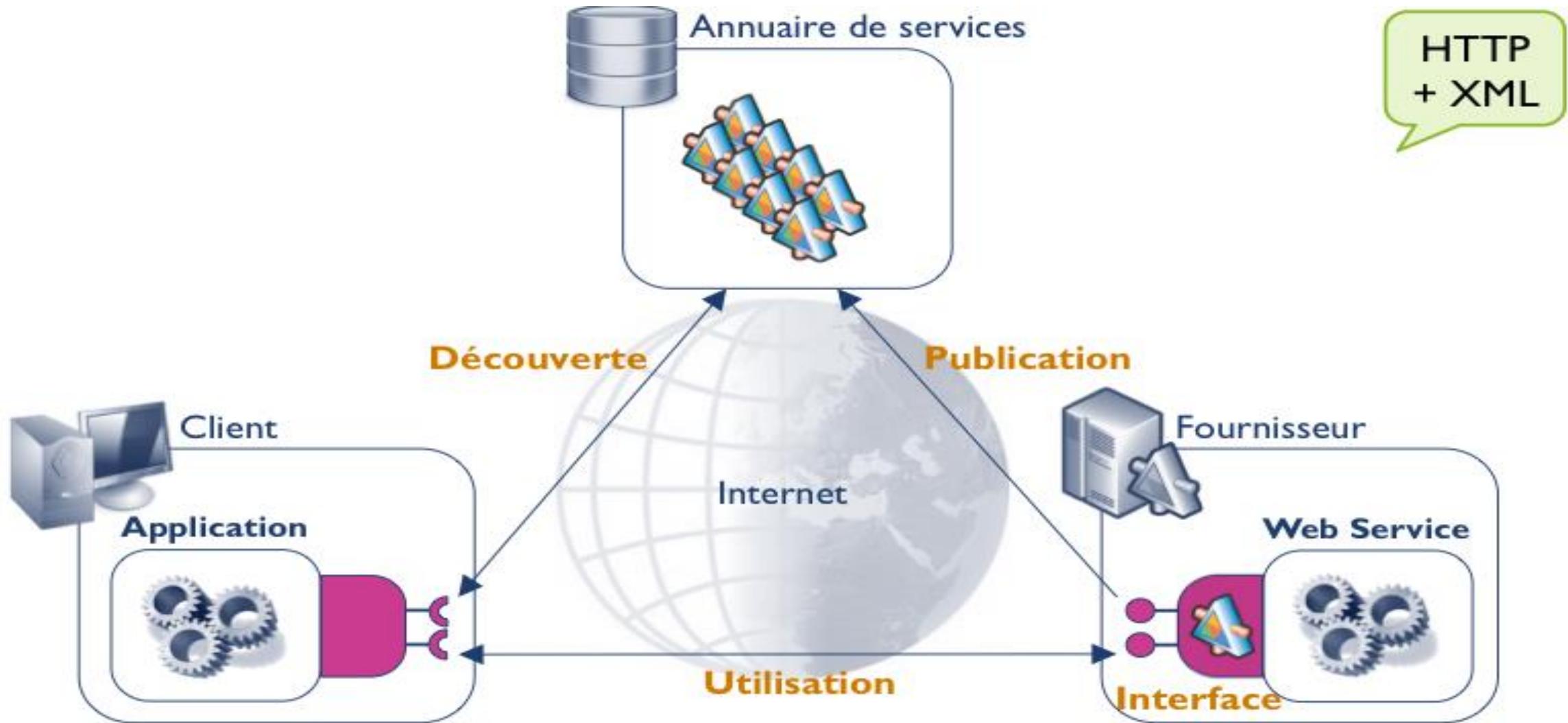
**ECOLE NATIONALE POLYTECHNIQUE D'ORAN**  
Département Mathématiques et Informatique  
Filière IMSI : 4<sup>ème</sup> année ingénieur

M. SABRI

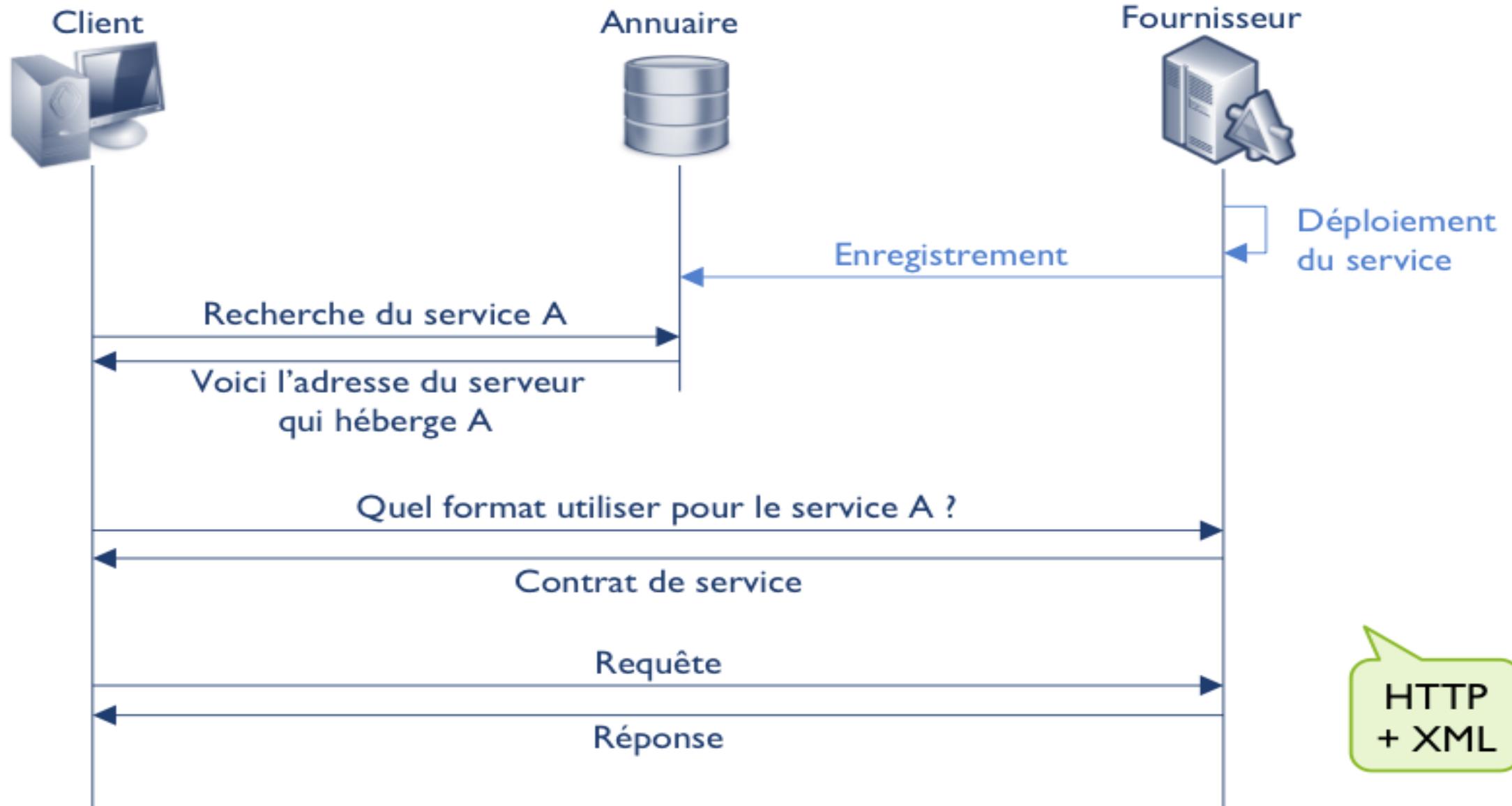
# Web Service = Service + Web ?

- Service = fonctionnalité mise à disposition et exécutée par un fournisseur lorsqu'elle est invoquée par un consommateur -> réutilisable + composable + indépendant
  - Interface :
    - Définit l'usage du service (syntaxe, sémantique, qualité) ↗ contrat
    - Masque l'implémentation du service pour un couplage consommateur/fournisseur faible
  - Format pivot : langage commun pour décrire et échanger les données
- Web Service = service mis à disposition sur Internet
  - Associé à une URL sur le web ↗ HTTP
  - Accessible via des protocoles internet standard
  - Accessible indépendamment des technologies d'implémentation
  - Auto-descriptif ↗ XML

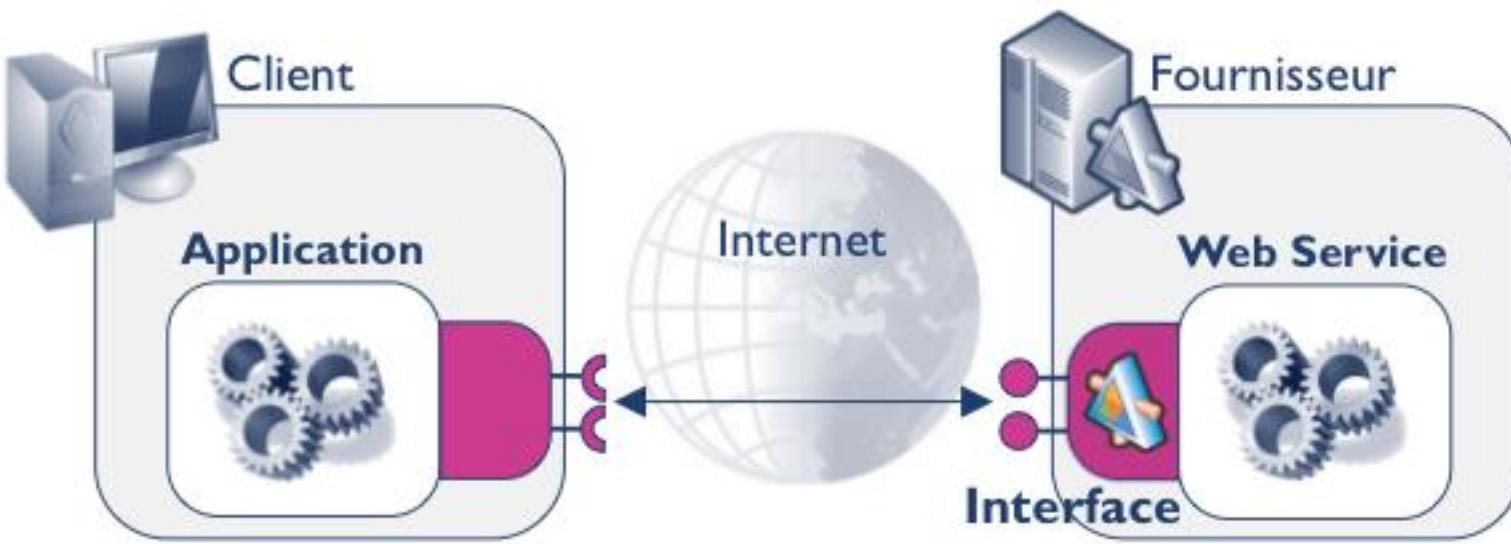
# Principe des Web Services



# Utilisation d'un Web Service



# Implémentation (hors annuaire)

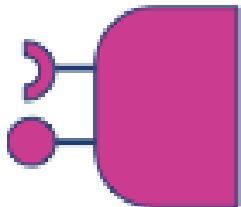


- Implémentation côté client et côté fournisseur :



Application « métier »

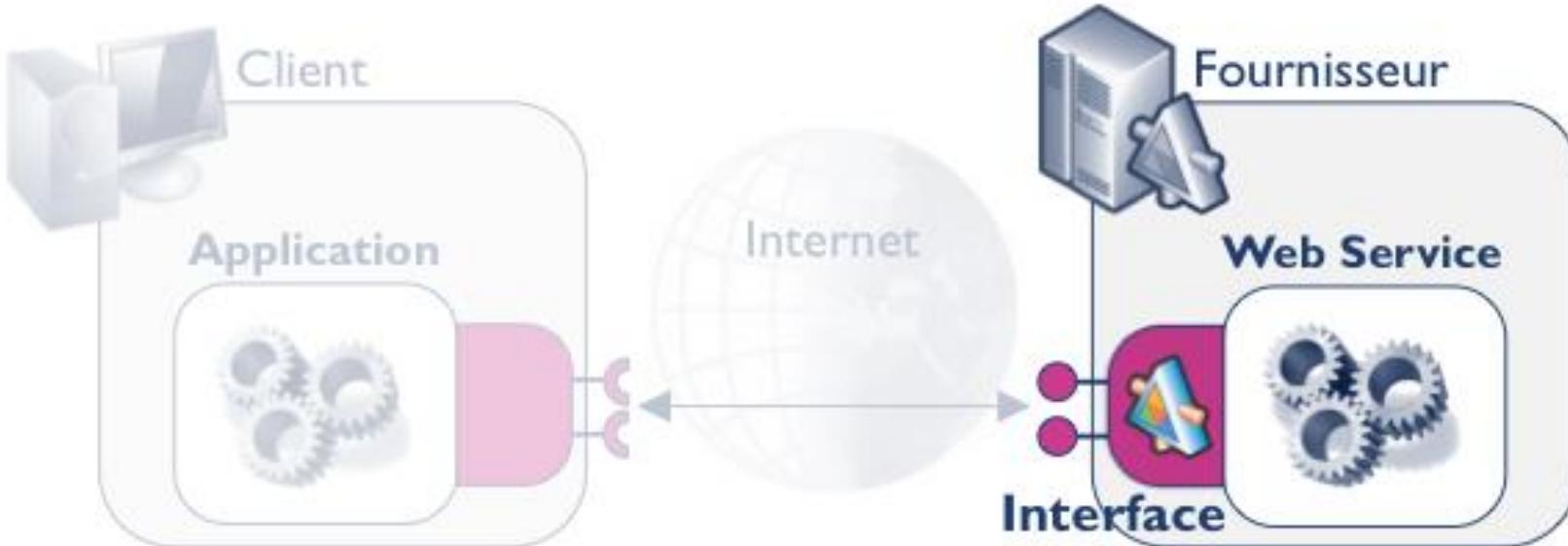
- ➥ toutes technologies possibles (Java, .NET, PHP...)



Traitements liés au **protocole**, basé sur **HTTP/XML**

- ➥ deux grandes familles : famille **WS-\*** et famille **RESTful**

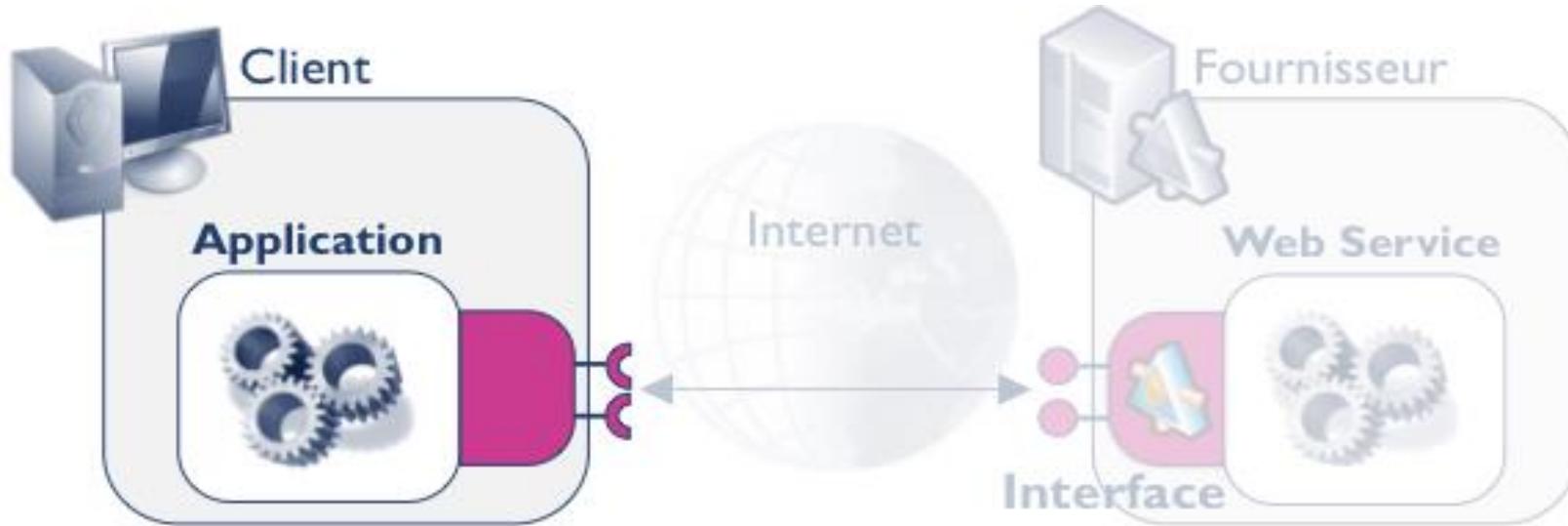
# Côté fournisseur



- Pour créer un Web Service :
  1. Définir le contrat du service
  2. Développer le service
  3. Développer la couche de traitement XML
  4. Déployer sur le service
  5. Publier dans l'annuaire

Suivant les technologies, certaines tâches sont automatisées...

# Côté client



➤ Pour créer une application cliente :

1. Rechercher le service dans l'annuaire
2. Récupérer le contrat du service
3. Créer un stub/proxy
4. Développer la couche de traitement XML
5. Utiliser le service et présenter les résultats (rendu)

Suivant les technologies, certaines tâches sont automatisées...

# Principales technologies (ws-\*)

