# Architecture Distribuée

Cours n°5

A.Saval

# Objectifs du cours “Architectures distribuées”

- Compréhension des motivations
- Compréhension de la logique de conception d'une architecture distribuée
- Maîtrise des principaux modèles
- Aperçu des problèmes posés
- Aperçu de quelques frameworks existants

# Aperçu du cours

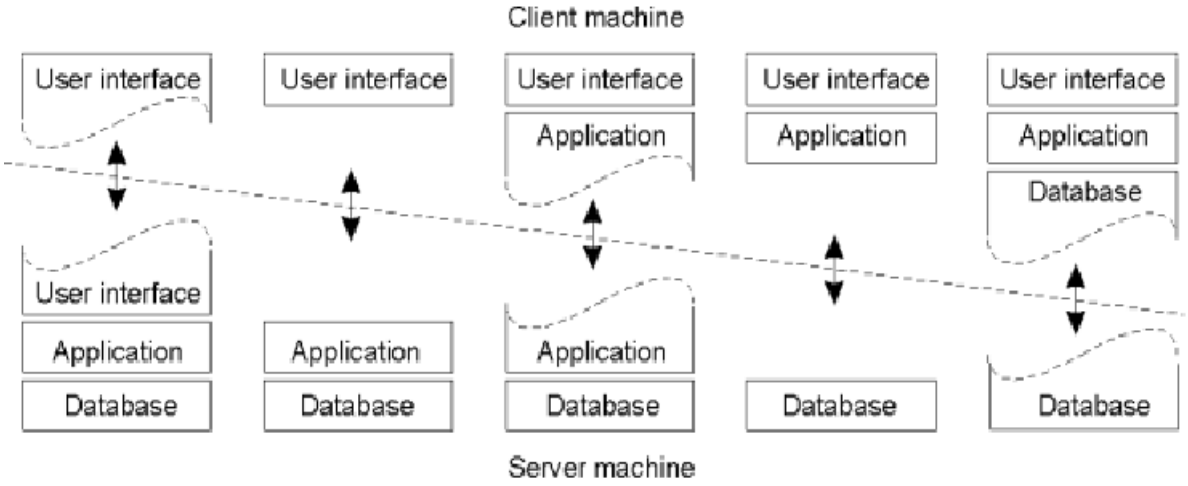
- Introduction
- Problème de conception d'architecture
- Architecture logique & matérielle
- Système distribué
- Modèles d'architecture
  - Client/serveur
  - 3-tiers
  - N-tiers
  - Virtualisation

# ARCHITECTURE N-TIERS

# Limites du 3-tiers

**Rappel sur l'achitecture 3-tiers :**

Client <--> Interface utilisateur <--> Application <--> Données




Limites :

- charge difficile à répartir entre client et serveur
- souvent concentré sur le serveur
- capacité d'extension limitée par le choix de la séparation C/S

# Architecture N-tiers

N-tiers ≠ N couches

N-tiers = séparation de la couche application

Distribution des traitements au niveau de la couche application :

- Séparation des couches applicatives
- Composants métiers réutilisables
- Maintenance facilité
- Possibilité d'extension par composants pour mieux adapter le système à la charge

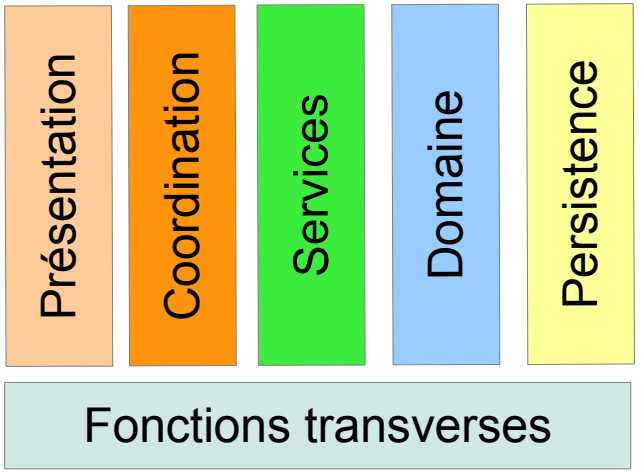
# “Nouvelles” notions

- Séparation en “composants” métiers
  - Correspondent aux fonctions de la couche applicative
  - Spécialisés et autonomes

- Indépendance des composants
  - Composant offrant un “service” clair et identifié
  - Possibilité de rendre le service générique (ré-utilisable)
  - Possibilité de distribuer (physiquement) les services
  - Facilite l'intégration d'applications tiers existantes par le masquage de l'implémentation

# Modèle en 5 couches

- Structuration des applications en décomposition logique sur 5 couches :
  - Présentation
  - Coordination
  - Services
  - Domaine
  - Persistance




- Proche du MVC (Modèle Vue Contrôleur)
- Chaque couche a ses propres responsabilités et utilise la couche située en dessous d'elle.
- Structuration est alors guidée par les contraintes exprimées et existantes : adapation de l'architecture aux contraintes d'environement spécifique à chaque projet.

# Couche présentation (1)

- Assure l'affichage de l'interface graphique utilisateur et/ou les Interfaces Homme-Machine (fenêtres, pages, composants graphiques...). Cette couche intègre principalement :
  - la gestion du domaine visuel
  - l'interaction avec les utilisateurs
  - l'interception des événements utilisateurs et l'appel à la couche de coordination
  - la gestion du multicanal (web, voix, mobile, fax)
  - les services de portail (agrégation d'IHM, bouquets de services)
  - les services d'export/impression (impressions PDF, gestion de templates...)

# Couche présentation (2)

- Trois grandes catégories d'IHM :
  - **Client léger**

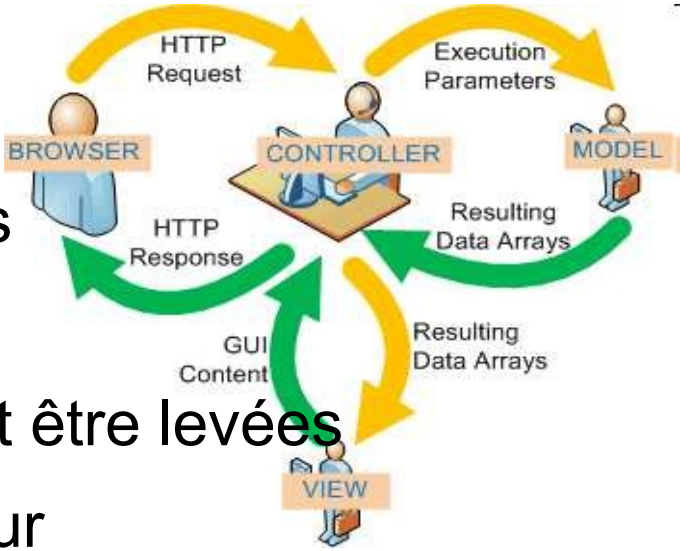
    ex : Gmail, Google Docs, Doodle...
  - **Client lourd**

    ex : Application Iphone/Androïd...
  - **Client riche (Rich/Smart Client)**

    ex:  Adobe Flex, Microsoft Silverlight, Google Web Toolkit...

    → tend à être remplacé par le HTML5 + javascript

# Couche coordination




- La couche de Coordination gère :
  - le contrôle de la cinématique des écrans
  - l'invocation des appels de services
  - les erreurs et les exceptions qui peuvent être levées
  - les sessions / espace de travail utilisateur
  - les habilitations et les droits d'accès

- Equivalent au contrôleur dans le motif de conception MVC (Modèle-Vue-Contrôleur)

# Couche Service

- **Traitements** effectués par l'application = implémentation de la logique des cas d'utilisation **(use-case fonctionnels)**.

- Cette couche doit :
  - implémenter la logique métier
  - gérer la sécurité applicative (login, rôles, droits d'accès...)
  - gérer les transactions étendues (processus, compensation)
  - gérer l'intégrité transactionnelle (transactions locales et distribuées)
  - gérer les appels aux objets métiers de la couche Domaine

- Elle gère les services métiers qui enchaînent des règles métiers (processus métier) et des appels à la couche Domaine

# Couche Domaine

- **Modèle « métier »**. Elle intègre principalement:
  - la gestion des règles métiers « élémentaires » (sans état, sans processus)
  - la fourniture des moyens d'accès à l'information (SGBDR, Mainframe...)
  - le respect des propriétés transactionnelles de la couche persistance

- Recense les **objets métiers** manipulées par l’application

# Couche Persistence

- Intègre principalement :
  - la persistance complète : données structurées ou non structurées, gérées entre autres via un SGBDR, annuaire LDAP, transaction CICS, ...
  - Services de stockage des données, moteurs relationnels, bases objets, bases XML...
  - la création, la modification, la suppression d'occurrences des objets métiers

- Offre les fonctionnalités de base qui permettent :
  - de **créer**, **rechercher**, **modifier** et **supprimer** (**CRUD**) des composants objets métiers dans le respect des propriétés transactionnelles classiques

# Couches transverses

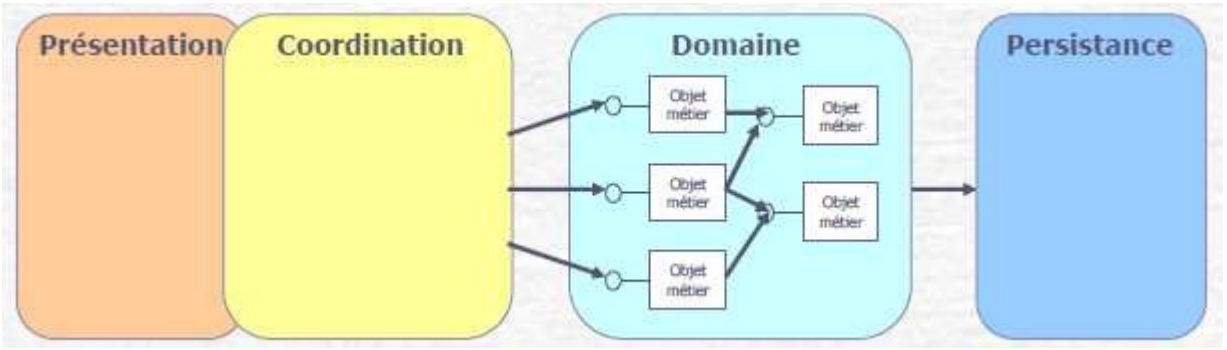
- Couche **Sécurité**
  - Services de sécurité : SSO, authentification, gestion des habilitations, intégrité, non-répudiation... La sécurité n'est pas une couche isolée, mais transverse aux autres couches:
    - authentification des utilisateurs et contrôle des habilitations au niveau des services IHM,
    - sécurisation des traitements (authentification, habilitations grosse maille et habilitations fines...),
    - sécurisation des échanges, sécurisation des données...

# Couches transverses

- Couche **Services Techniques** (Core Services)
  - Indépendamment des fonctionnalités des applications et de leur découpage en couches logicielles, on retrouve des composants et services de base communs (Core Services) et transverses à l'ensemble des couches :
    - gestion des traces
    - statistiques et logs
    - gestion des erreurs
    - gestion des propriétés de configuration
    - gestion des fichiers de messages (internationalisation, messages d'erreurs)
    - monitoring...

# Architecture Orientée Objets

- Liens forts entre la couche **Coordination** et les objets métiers de la couche **Domaine**.
- Le **code client traite directement avec le modèle objet** : dépendance forte au modèle et nombre d'appels important entre les deux couches.
- **Multiplication des appels entre couches**.

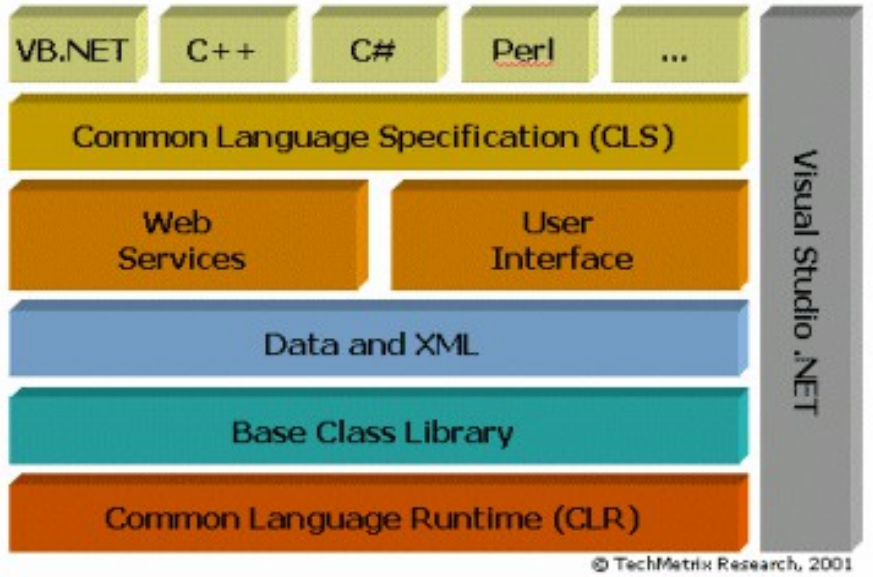


# Implémentation de l'OOA

2 mondes dominent le marché industriel

# Plate-forme Microsoft .NET

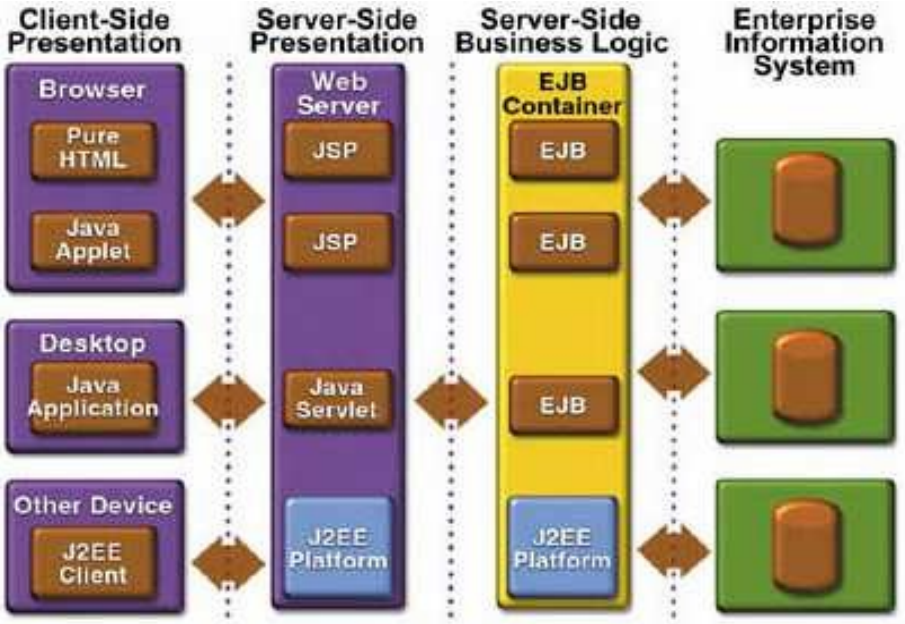
- .NET est une stratégie de produits Microsoft
  - Composé de 3 parties :
  - CLR (Common Language Runtime)
  - BCL (Base Class Library)
  - ASP.NET




- CLS (Common Language Specification)
- CTS (Common Type System)
- MSIL (Microsoft Intermediate Language)

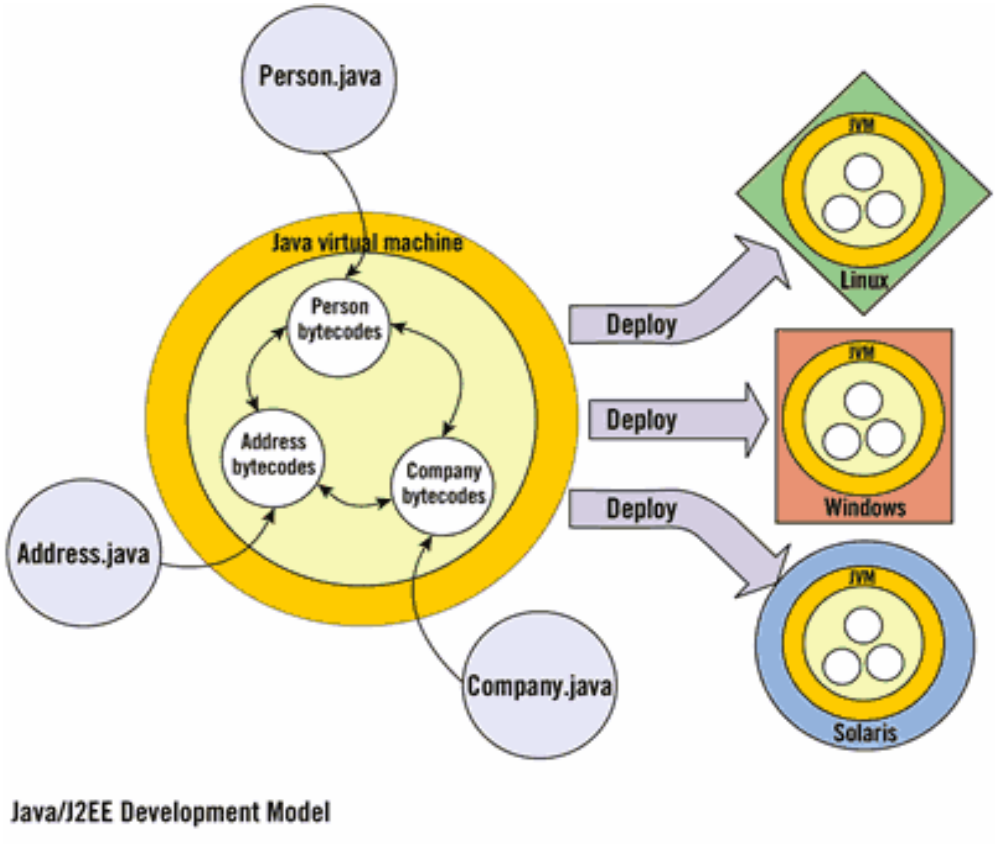
# Plate-forme J2EE

- J2EE est un standard industriel (contrairement à .net c'est une spécification)
- Une application J2EE assemble des composants
  - composants clients : applications clients, applets
  - composants web : servlet et JSP
  - composants business : EJB



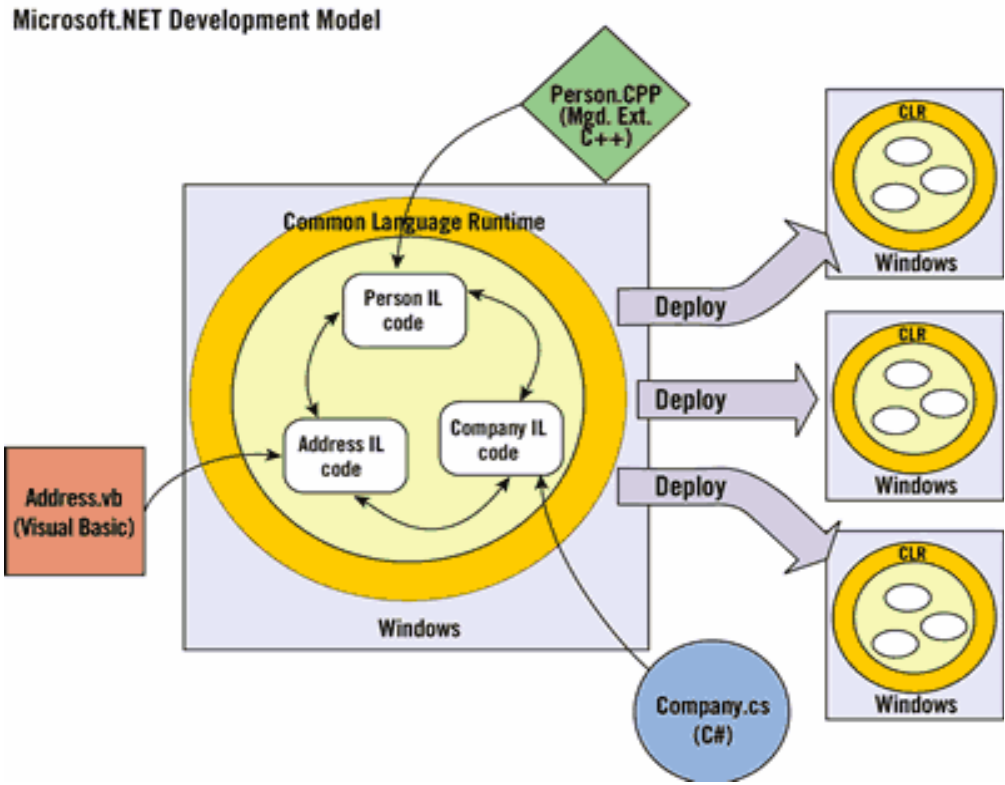
# Comparaison du modèle de développement de J2EE et .NET

**Un langage**
**Plusieurs plate-formes**




Java/J2EE Development Model

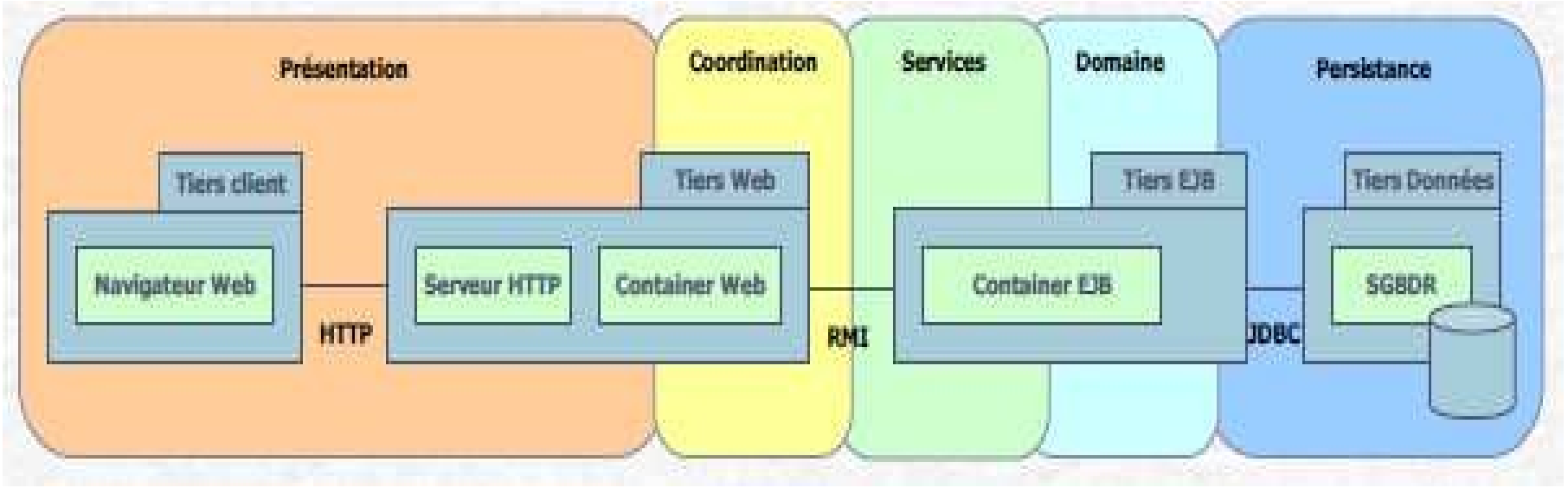
**Plusieurs langages**
**Une plate-forme**



# Implémentation

- Adaptation des 5 couches

  Ex : modèle architecture J2EE



# Architecture Orientée Services (1)

- Les axes majeurs de la SOA sont :
  - **La réutilisation** et la composition : partage de modules entre applications
  - **La pérennité** : implique le support des technologies existantes et à venir
  - **L'évolutivité** : la majeure partie des applications sont amenées à évoluer dans le temps afin de pouvoir répondre aux nouveaux besoins fonctionnels
  - **L'ouverture** et l'interopérabilité : partager des modules applicatifs entre plates-formes et environnements
  - **La distribution** : pouvoir utiliser ces modules à distance et les centraliser au sein de l'entreprise par exemple
  - **La performance** : induit par un pasage à l'échelle simplifié

# Architecture Orientée Services (2)

- Toute application/composant du système d'information est un **fournisseur de services**

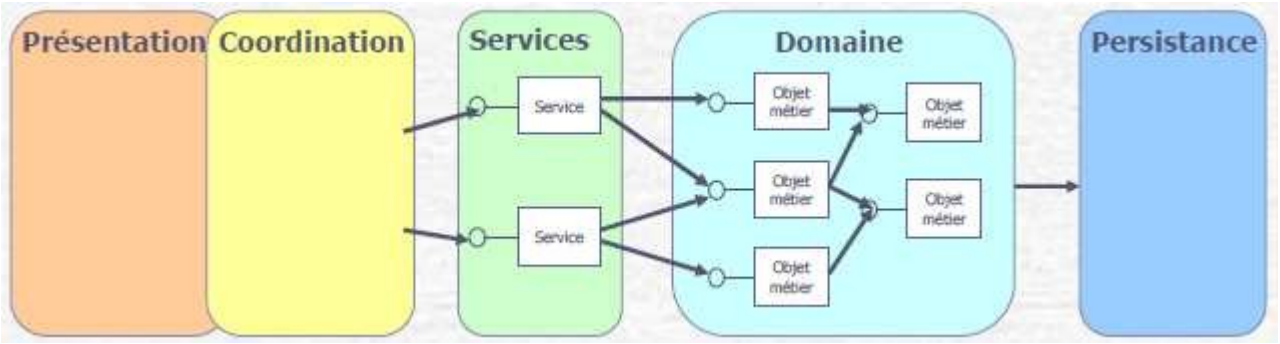
- Le principal enjeu de la SOA : **réutilisation des services**, ceux-ci doivent être pensés non seulement en fonction d'un projet immédiat, mais aussi sur le long terme pour servir à d'autres applications

- **Couplage faible** entre les services.

- Intégration facilitée de l'existant dans le système d'information de l'entreprise.

# Architecture Orientée Services (3)

- Niveau d'indirection supplémentaire est introduit dans la **couche Services**.
- La couche Coordination ne manipule plus directement les objets métiers, mais passe par des appels de services. Les **objets métiers se trouvent dans des bibliothèques de classes** directement chargées.
- Les **services** agissent comme des « **boites noires** » :
  - abstraction de la complexité du modèle objet
  - ensemble de fonctionnalités restreints
  - réduction des échanges entre les couches



# Architecture Orientée Services (4)

- Un service (local ou distant) est un module :
  - Pouvant être invoqué en mode synchrone ou asynchrone
  - Chargé d'une fonction particulière (= offrant un service)
  - Présentant une interface bien définie (= contrat de service)

# Architecture Orientée Services (5)

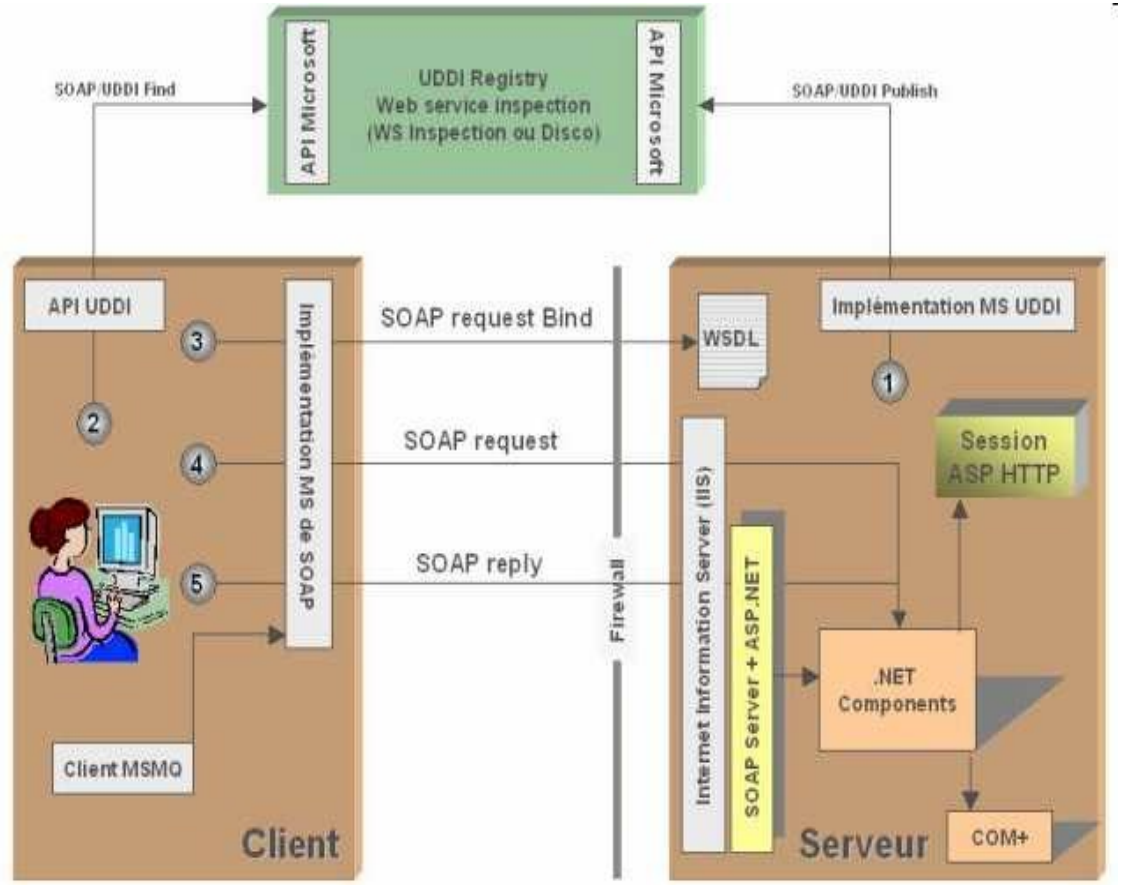
- Un service doit assurer un rôle bien défini et non redondant (les services doivent être factorisés)
- Un service doit pouvoir être utilisé par plusieurs types de consommateurs Internes ou Externes (clients, filiales, partenaires, ...)
- Un service doit pouvoir être utilisé par exemple pour un traitement asynchrone multi-valué ou pour un traitement synchrone mono-valué

**SOA ≠ Web service**

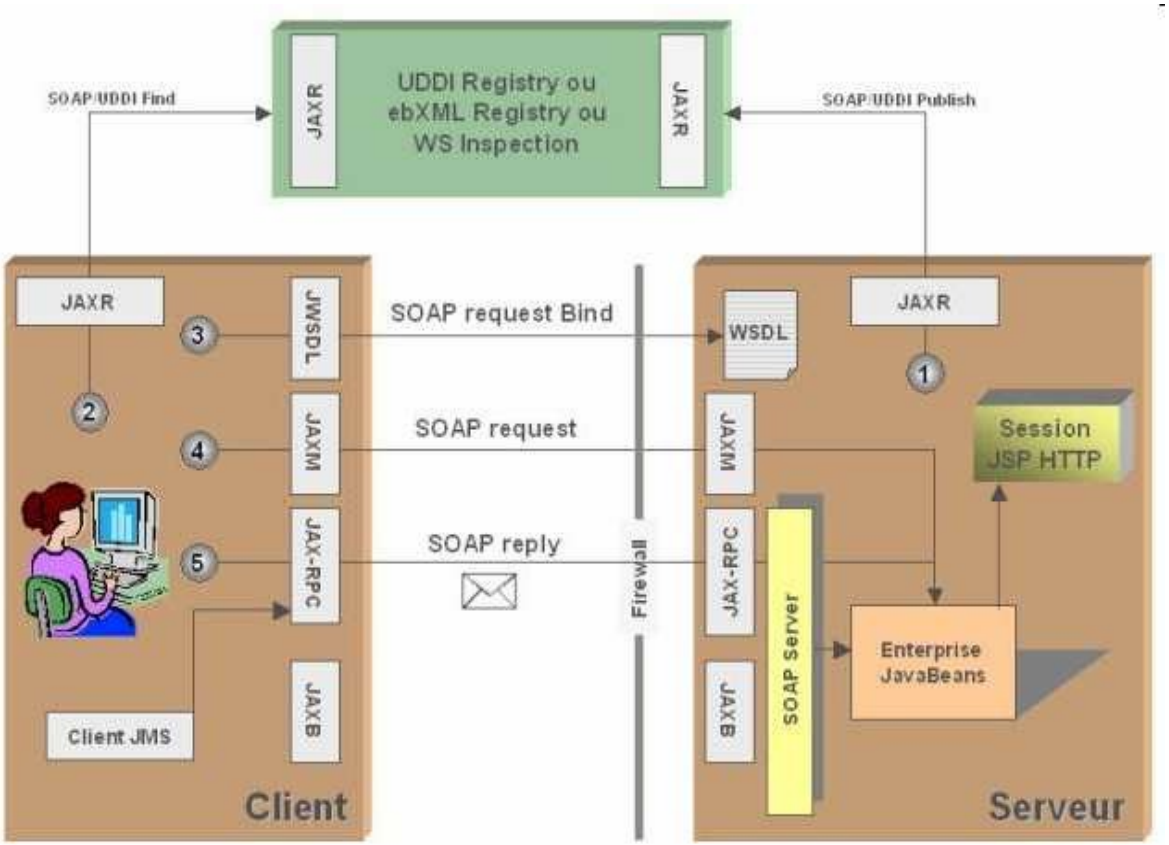
# Implémentation de la SOA

- Collection de fonctions packagées dans une même entité et publiées pour etre utilisée sur le reseau
  - Interface XML sur un ensemble de services
  - Evolution naturelle des systèmes distribués
  - Fondement des web services est l'utilisation de messages XML transportés sur des protocoles standard comme HTTP
  - « Old technologies wearing a new hat »
- Protocoles simples et standards permettant une utilisation universelle
    - UDDI, WSDL, REST, SOAP...
- Interface au-dessus des architectures n-tiers existantes (.NET, J2EE)

# Architecture Web Services .NET



# Architecture Web Services J2EE



# Problème : communication inter-service

La SOA ne présage pas du choix de la structure des communications

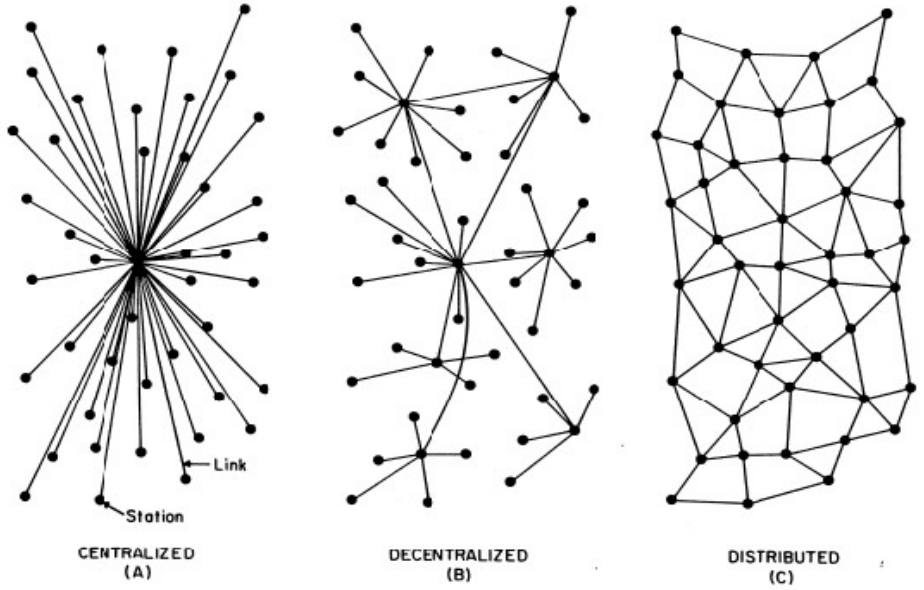
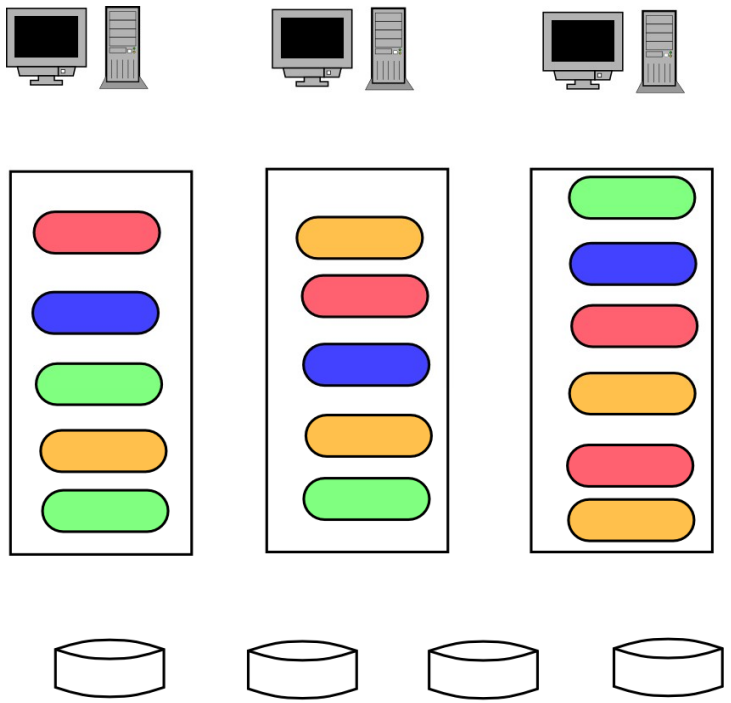


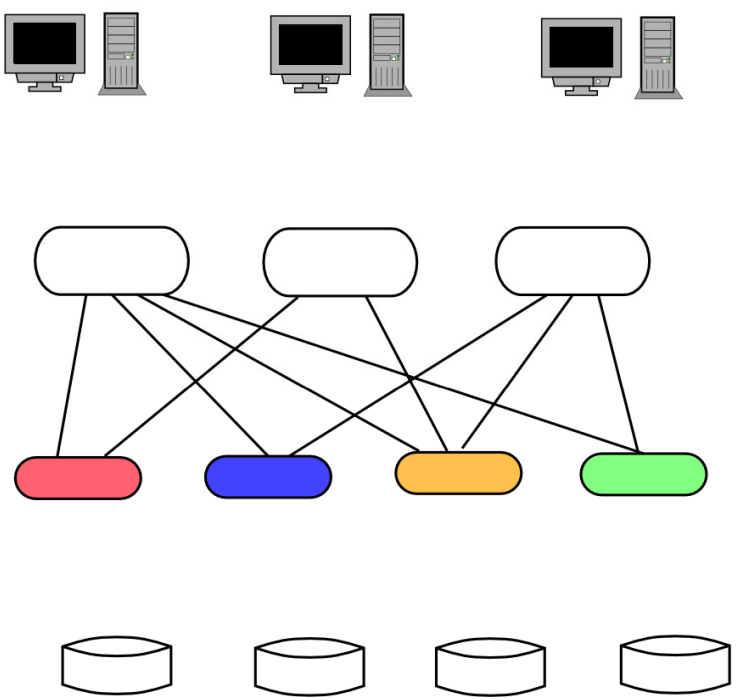

FIG. 1 – Centralized, Decentralized and Distributed Networks

Intégration de l'historique : multiplicité des protocoles à faire collaborer...

# Solution ?



(1) **Monolitic applications**

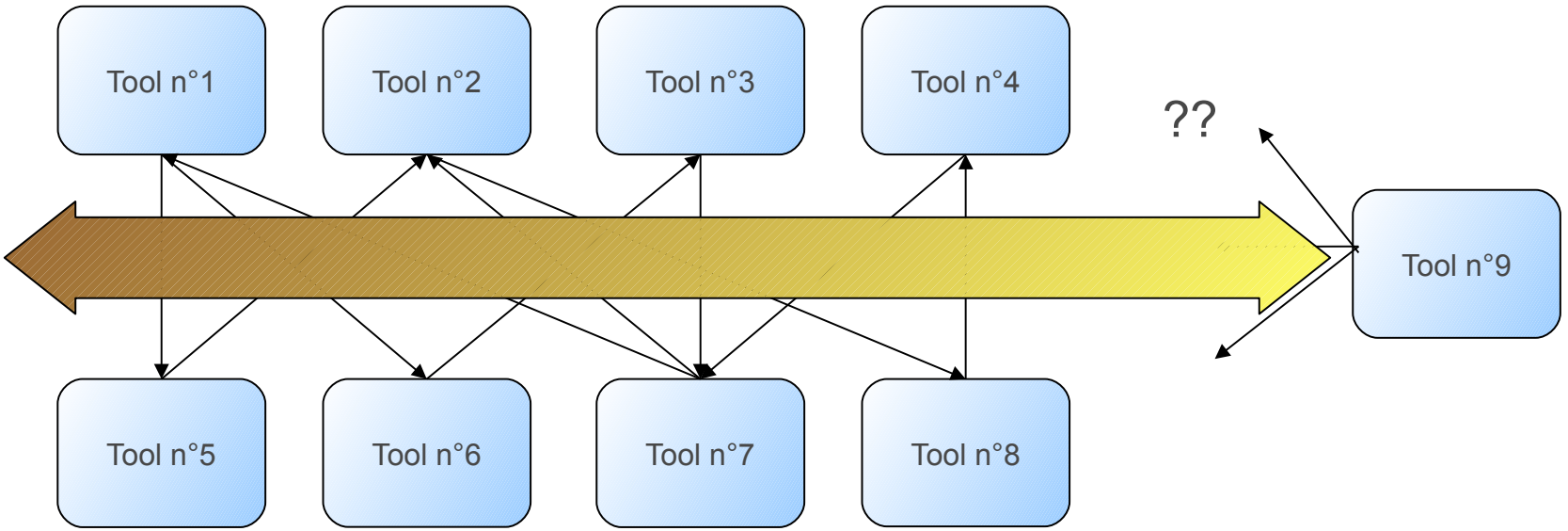


(2) **Tiered service architecture**

(3) ???

# Bus de service (1)

- Problème de communication inter-composants
  - Rationalisation des comunications
  - Abstraction des protocoles



# Bus de service (2)

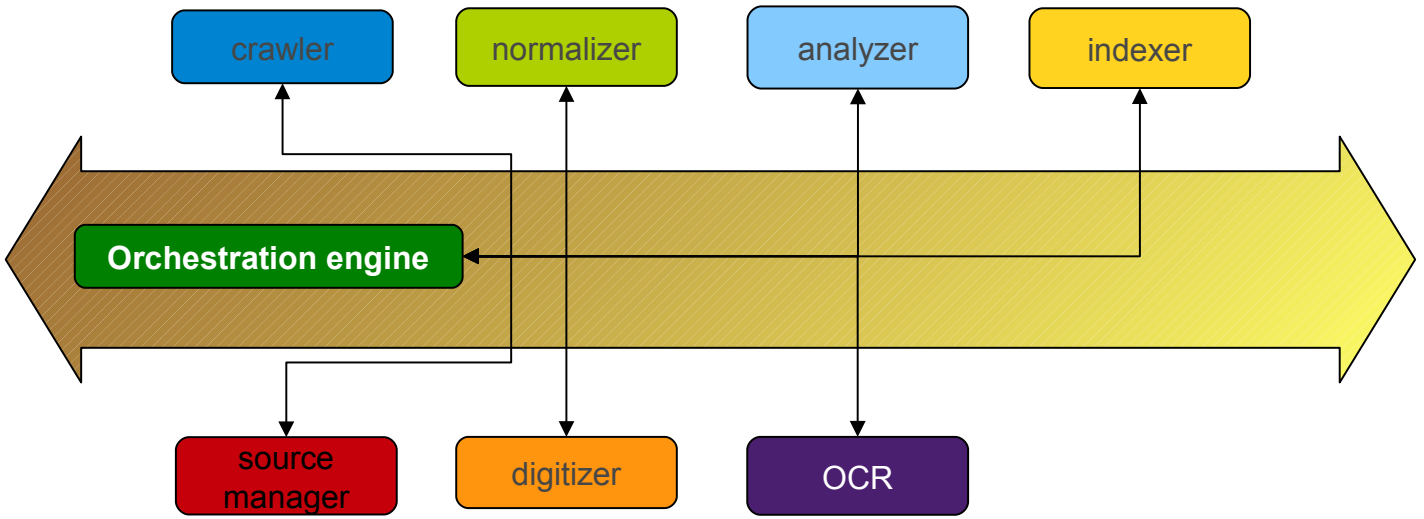
Conception des chaines de traitement à un niveau faisant abstraction des protocoles

Example 1 : web information gathering

**define sources → crawl → normalize → analyze → index**

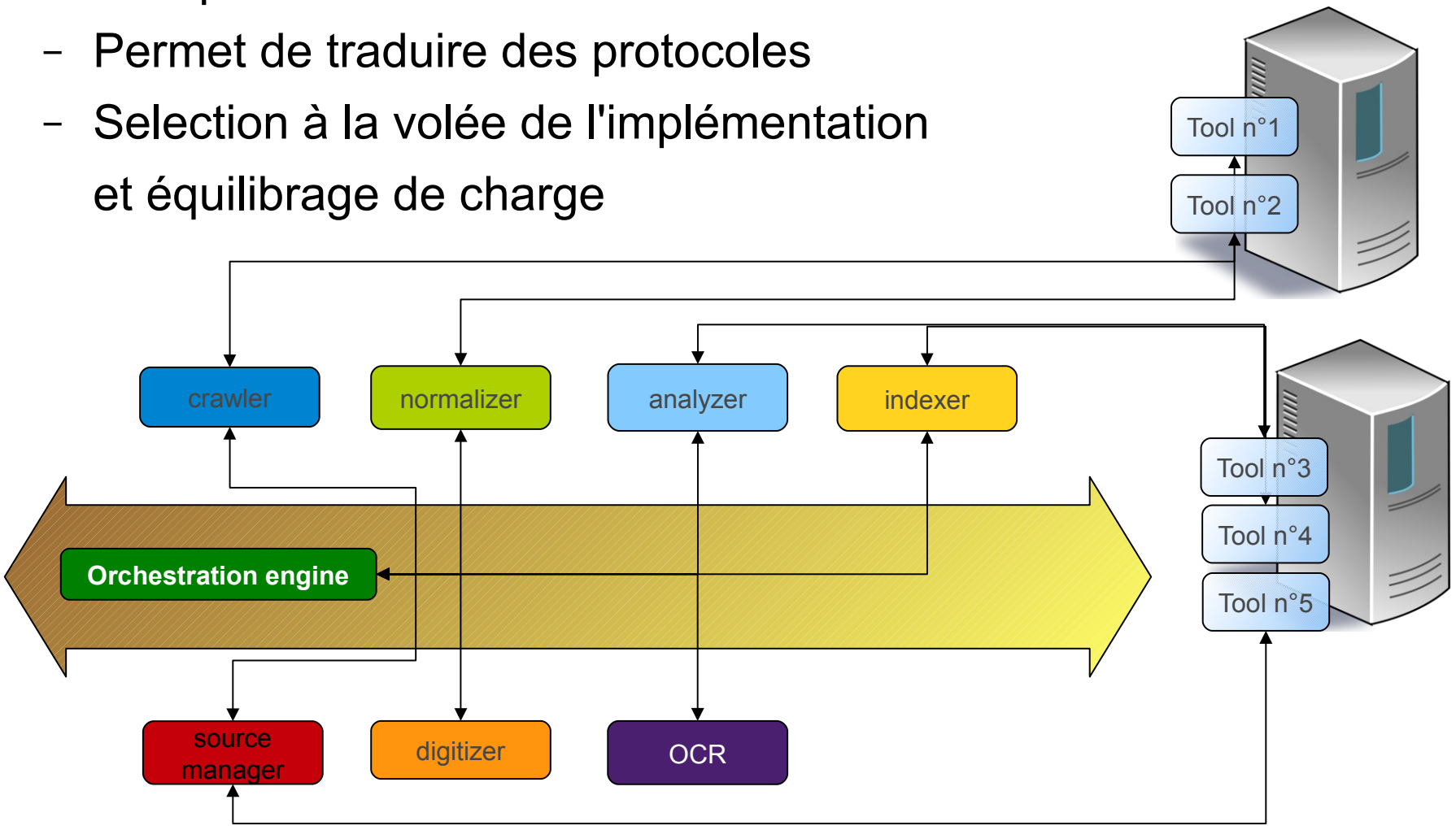
Example 2 : paper document analysis

**digitize document → apply OCR → analyze → index**



# Bus de service (3)

- Indépendance complète des composants !
  - Masque la localisation
  - Permet de traduire des protocoles
  - Selection à la volée de l'implémentation et équilibrage de charge



# Bus de service (4) : technologie

- Nombreux Bus open source
  - Apache, Petals, Camel...
- Différence au niveau de l'implémentation des standards et des composants internes
  - Connecteurs aux protocoles
  - Annuaire de services
  - Orchestrateur...

# Références


- Software Architecture: IEEE Standard 1471-2000
- P. Kruchten, Architectural Blueprints—The “4+1” View Model of Software Architecture, IEEE Software 12 (6), Nov. 1995, pp42-50
- Tanenbaum & van Steen, Distributed Systems, Principles and Paradigms, seconde édition
- Architecture of Distributed Systems, cours de Johan Lukkien, 2011
- Architectural Patterns Revisited – A Pattern Language, Paris Avgeriou & Uwe Zdun, 2005
- Software Architecture, Foundations, Theory, and Practice, R.N. Taylor, N. Medvidovic, E.M. Dashofy, Wiley & Sons, 2009
- Software Architecture in Practice, Second Edition, L. Bass, P. Clements, R. Kazman, SEI Series in Software Engineering, Addison-Wesley, 2003