Architectures Logicielles

Dr. Ghazala HCINI

Université de Gabès

2025-2026

Plan

CONCEPT DES ARCHITECTURES LOGICIELLES

CONCEPTION UML D'UNE ARCHITECTURE LOGICIELLE

STYLES ET PATRONS ARCHITECTURAUX

ARCHITECTURE J2EE

Plan

CONCEPT DES ARCHITECTURES LOGICIELLES

CONCEPTION UML D'UNE ARCHITECTURE LOGICIELLE

STYLES ET PATRONS ARCHITECTURAUX

ARCHITECTURE J2EE

Concepts de base

Génie logiciel

- Le Génie Logiciel (souvent appelé Software Engineering) est une discipline d'ingénierie industrielle. Son objectif est d'étudier et de codifier les méthodes de travail, les outils et les bonnes pratiques que les ingénieurs appliquent pour la conception et le développement de systèmes logiciels.
- Le Génie Logiciel se concentre sur l'établissement de procédures systématiques pour garantir que les systèmes complexes répondent aux attentes du client. L'objectif principal est de produire des logiciels qui possèdent des qualités essentielles telles qu'une haute fiabilité, des performances optimales, et un coût de maintenance réduit, tout en respectant les contraintes de budget et de délai de développement.

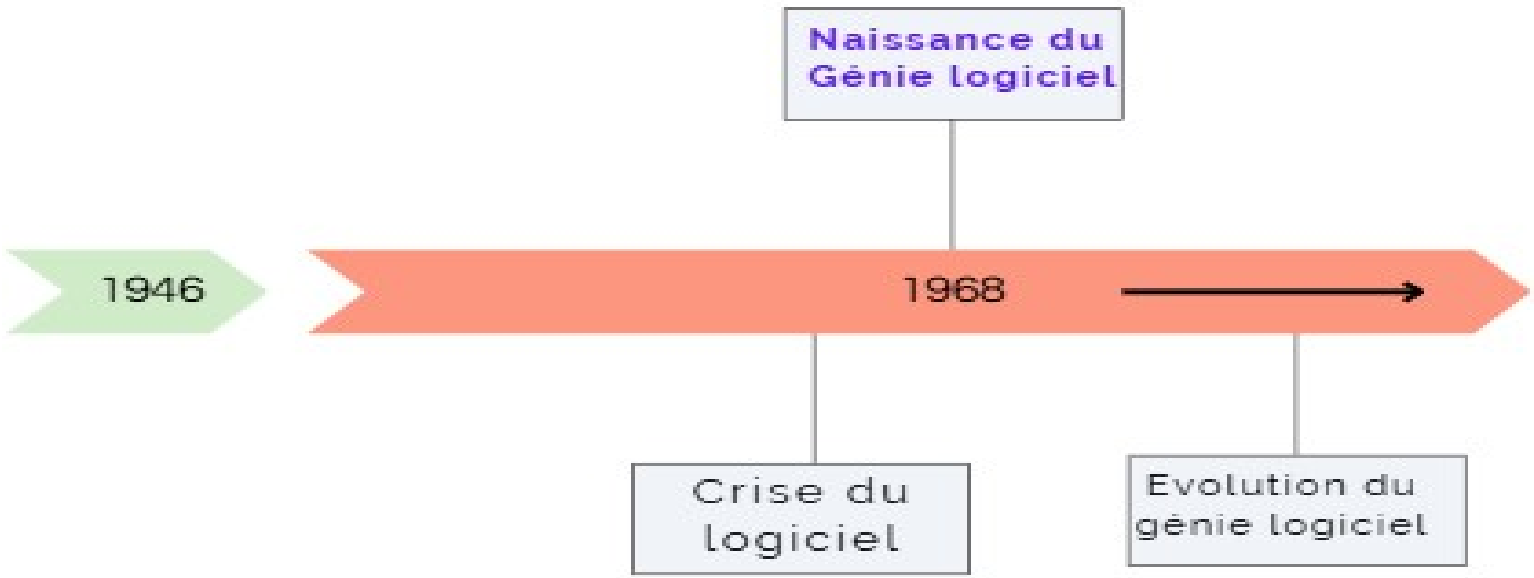
Concepts de base

Génie logiciel

Le champ du Génie Logiciel est vaste, couvrant de manière exhaustive les outils, techniques et méthodes utilisés par les équipes. Il organise l'intégralité des activités de développement et de maintenance, depuis la modélisation initiale jusqu'à la production des artefacts finaux et la gestion de l'ensemble du projet.

Un petit historique

Commençons par exposer brièvement l'origine et l'évolution du domaine de génie logiciel.



Un petit historique

De la crise du logiciel au GL

Problématique

- À la fin des années 60, avec l'avènement des machines de $3^{\text{ème}}$ génération est apparue la crise du logiciel.
- Les mécanismes utilisés pour les petits systèmes voyaient leur limite sur ces grosses applications.
- Les systèmes étaient peu fiable, difficiles à maintenir et avec de faible performance.

De la crise du logiciel au GL

Besoins

- De nouvelles techniques et méthodes pour concevoir et maîtriser ces nouveau logiciels devenaient indispensable.
- Nécessité de passer d'une démarche artisanale à une discipline d'ingénieur.

Un petit historique

- Pour sortir de la crise, en **1968** sous le parrainage de l'OTAN (Organisation du Traité de l'Atlantique Nord), un groupe de chercheurs et de praticiens crée le génie logiciel (anglais, software engineering)
- Le génie logiciel est défini comme un domaine qui couvre les méthodes, la modélisation, les techniques, les outils, les activités, les biens livrables et la gestion de projets relatifs au développement et à la maintenance du logiciel.
- En s'appuyant sur une approche méthodique, l'équipe de développement accroit sa productivité et réalise des logiciels de qualité. De plus, il devient plus facile aux gestionnaires de « prédire » l'échéancier puisque les étapes sont connues.

Un petit historique

Depuis 1968, on assiste à une prolifération et à une évolution continue :

- des méthodes d'analyse et des notations pour spécifier les fonctionnalités d'un logiciel ;
- des techniques de conception et des notations pour exprimer la solution ;
- des techniques et des langages de programmation ;
- des procédures de validation et de vérification pour assurer la qualité du logiciel ;
- des procédures de maintenance.

Génie logiciel

Malgré les avancées, le domaine de la recherche en Génie Logiciel est demeuré hautement pertinent, conduisant à l'émergence de nouvelles approches fondamentales dans les années 90, notamment :

- Le développement des Systèmes Distribués,
- L'essor des Modèles de Composants (visant la réutilisation),
- L'établissement d'une norme de modélisation unique à travers UML (Unified Modeling Language),
- L'Architecture Logicielle.

Nouveaux thèmes de recherche

De nombreux thèmes de recherche se sont alors développés et ont contribué à l'élaboration de ce nouveaux domaine :

- La spécification et analyse des besoins
- La conception et la modélisation
- Les langages de programmation et les environnements de développement
- La validation et le test
- L'ingénierie inverse et la maintenance
- La gestion de projets

Architecture Logicielle

L'architecture logicielle est un domaine du GL qui a reçu une attention particulière ces dernières années. C'est au cours de la décennie 1970–80 que les grands principes architecturaux furent élaborés.
La conception d'une bonne architecture logicielle peut amener à un produit :

- Qui répond aux besoins des clients
- Qui peut être modifié facilement pour rajouter de nouvelles fonctionnalités.

Architecture Logicielle

L'architecture logicielle consiste à :

- Décrire l'organisation générale d'un système et sa décomposition en sous-systèmes ou composants
- Déterminer les interfaces entre les sous-systèmes
- Décrire les interactions et le flot de contrôle entre les sous-systèmes
- Décrire également les composants utilisés pour implanter les fonctionnalités des sous-systèmes.

Pourquoi développer une architecture logicielle ?

- Pour permettre à tous de mieux comprendre le système
- Pour permettre aux développeurs de travailler sur des parties individuelles du système en isolation
- Pour préparer les extensions du système.

Utilité d'une architecture logicielle

- **Compréhension** : facilite la compréhension des grands systèmes complexes en donnant une vue de haut-niveau de leur structure et de leurs contraintes. Les motivations des choix de conception sont ainsi mis en évidence
- **Réutilisation** : favorise l'identification des éléments réutilisables, parties de conception
- **Évolution** : met en évidence les points où un système peut être modifié et étendu.
- **Analyse** : offre une base pour l'analyse plus approfondie de la conception du logiciel, analyse de la cohérence, test de conformité, analyse des dépendances,
- **Gestion** : la gestion générale.

Caractéristiques architecturales

Performance

Représente la performance par rapport à la quantité de ressources utilisées dans des conditions données.

Question :

Citez des exemples de “performance ”

- **Comportement temporel** : temps de réponse, débit
- **Utilisation des ressources** : quantité et types de ressources utilisées
- **Capacité** : limites maximales

Caractéristiques architecturales

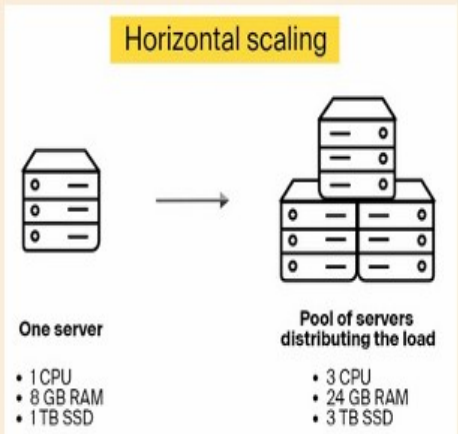
Scalabilité

La scalabilité consiste à gérer un grand nombre d'utilisateur sans avoir une dégradation sérieuse des performance.

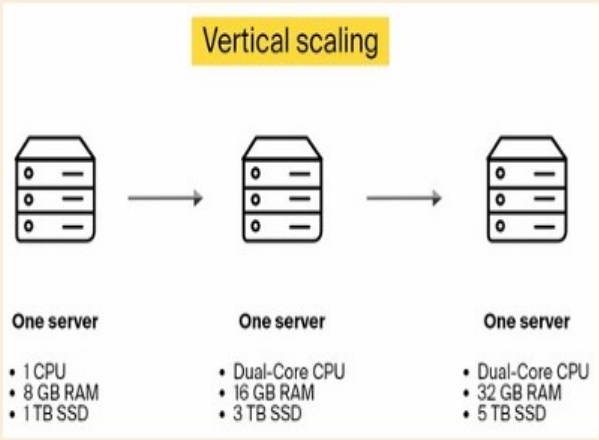
Question :

Comment pourrait-on faire ?

Ajouter nœuds supplémentaires (scale out)



Soit en renforçant le matériel (scale up)



Elasticité

L'élasticité est le degré auquel un système est capable de s'adapter aux demandes en approvisionnant et désapprovisionnant des ressources de manière automatique, de telle façon à ce que les ressources fournies soient conformes à la demande du système.
L'élasticité permet de saisir les aspects essentiels de l'adaptation, à savoir :

- La vitesse : correspond au temps nécessaire pour scale up.
- La précision : est défini comme l'écart absolu entre la quantité actuelle de ressources allouées et la demande réelle de ressources.

Caractéristiques architecturales

Interopérabilité

Est la capacité d'un système d'entreprise (ou de tout système informatique général) à utiliser les informations et les fonctionnalités d'un autre système.

- L'interopérabilité est la clé de la construction de systèmes d'entreprise avec des services mixtes.

Caractéristiques architecturales

Couplage faible, Forte cohésion

Concevoir des composants autonomes, indépendants et dotés d'un objectif unique et bien défini.

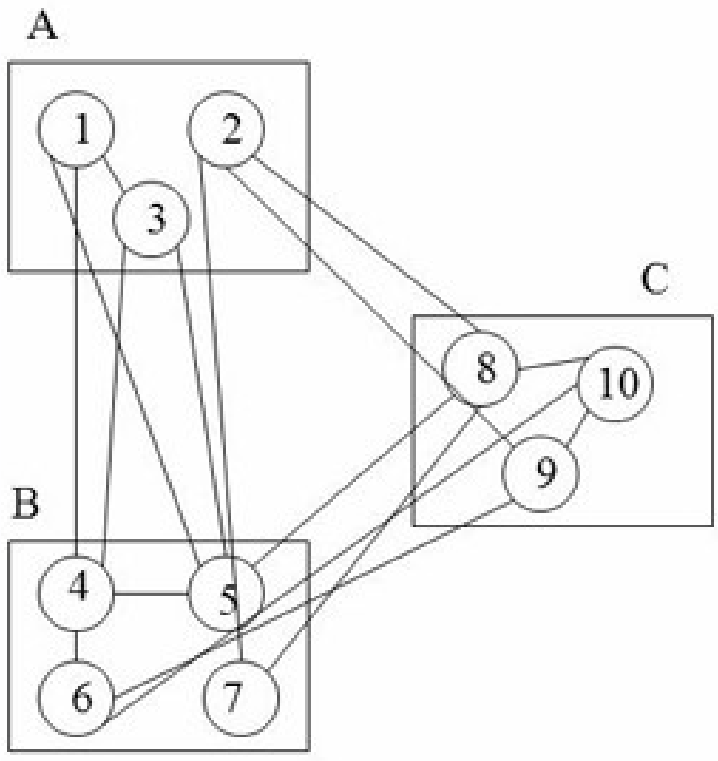
Question :

Que cela peut-il signifier (couplage et cohesion) ?

- Couplage faible : Les modules doivent être aussi indépendants que possible des autres modules, de sorte que les modifications apportées au module n'aient pas d'impact important sur les autres modules.
- Forte cohésion : La cohésion fait référence à la manière dont les éléments d'un module s'intègrent les uns aux autres. Les codes apparentés doivent être proches les uns des autres afin d'assurer une grande cohésion.

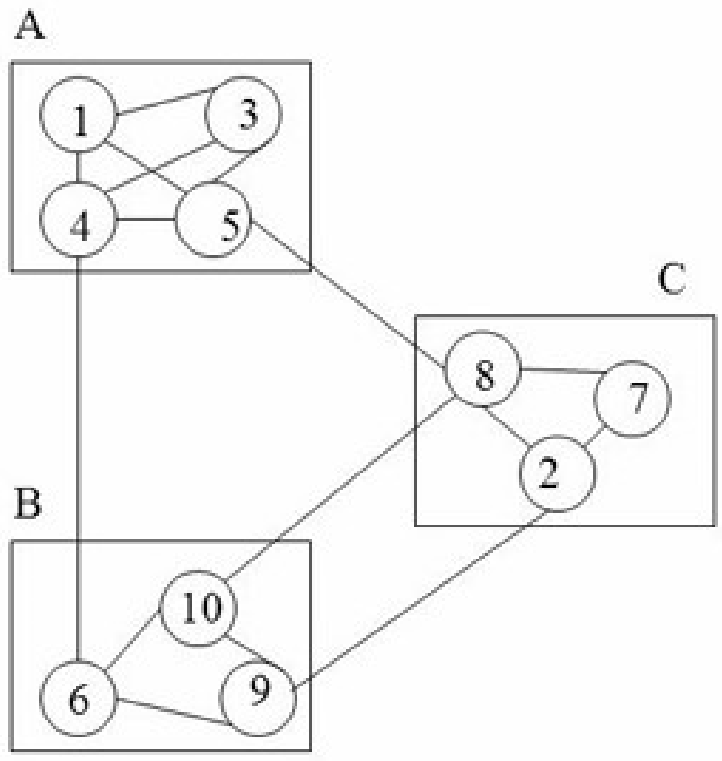
Caractéristiques architecturales

Couplage faible, Forte cohésion



Mauvaise modularisation

Faible cohésion, couplage élevé

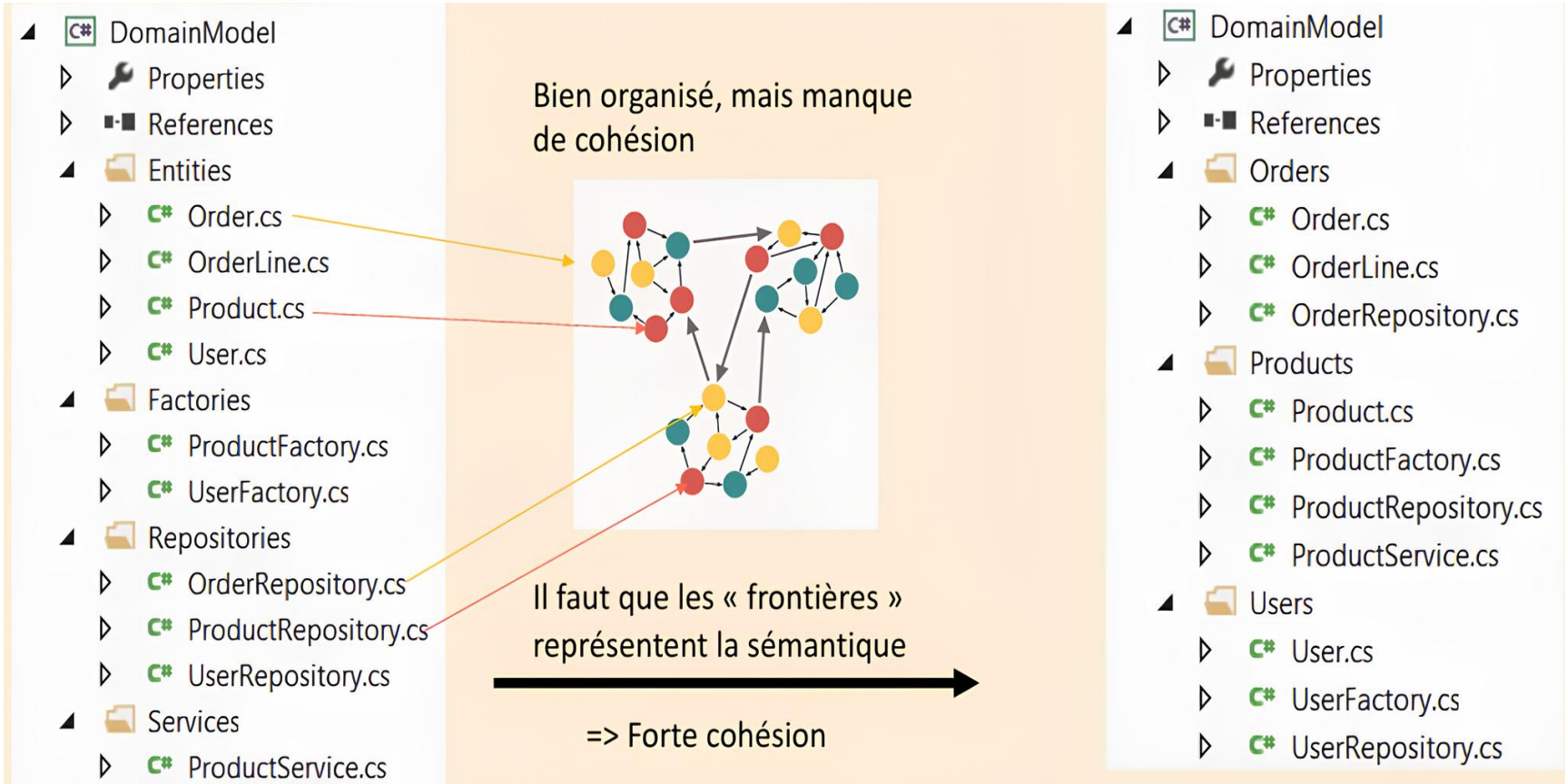


Bonne modularisation

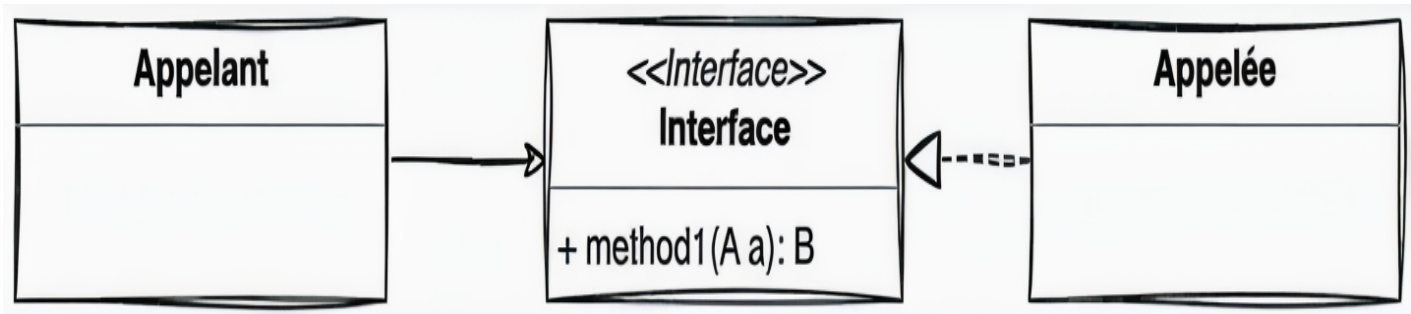
Haute cohésion, faible couplage

Caractéristiques architecturales

Couplage faible, Forte cohésion (package)

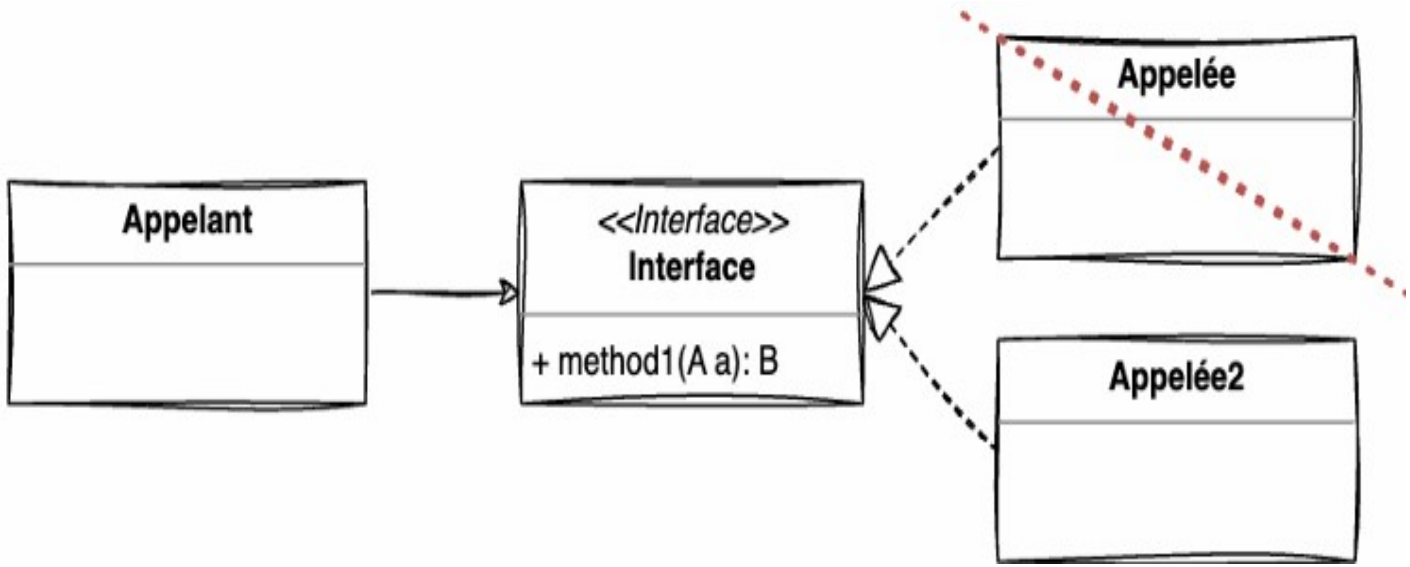


Interface



- Elle définit les méthodes (i.g. service) qui vont pouvoir être appelées
- Elle définit les informations que doit fournir l'appelant (i.g paramètre de la méthode)
- Elle définit les informations que l'appelant va obtenir (i.g type de retour).

Changer l'implémentation



Note : Cycle **Build** → **Test** → **Deploy**

Changer l'implémentation

Interface

- L'appelant utilise une interface qui définit une méthode commune.
- L'interface agit comme un contrat que toutes les classes concrètes doivent respecter.
- Les classes concrètes (Appelée, Appelée2) implémentent cette interface en fournissant des comportements spécifiques.
- L'appelant ne dépend que de l'interface, pas des classes concrètes, ce qui permet de découpler la logique métier des détails d'implémentation.

Avantages

- La modification ou le remplacement d'une implémentation peut être faite sans impacter le code de l'appelant, ce qui facilite la maintenance et l'évolution continue du logiciel.
- Ce modèle est parfaitement adapté au cycle **Build** $\rightarrow$ **Test** $\rightarrow$ **Deploy**.

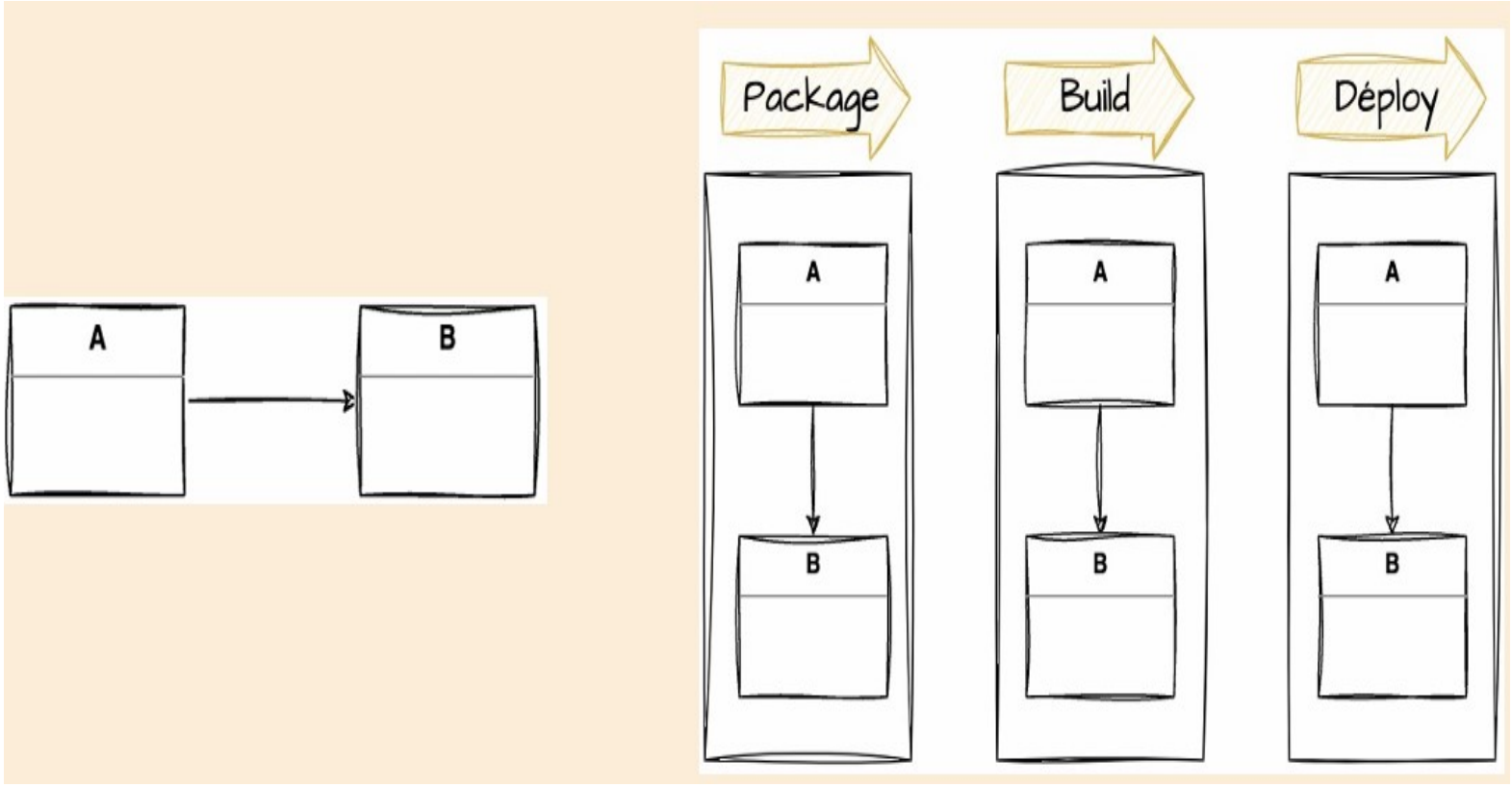
Concepts fondamentaux

Inversion de dépendance

Question :

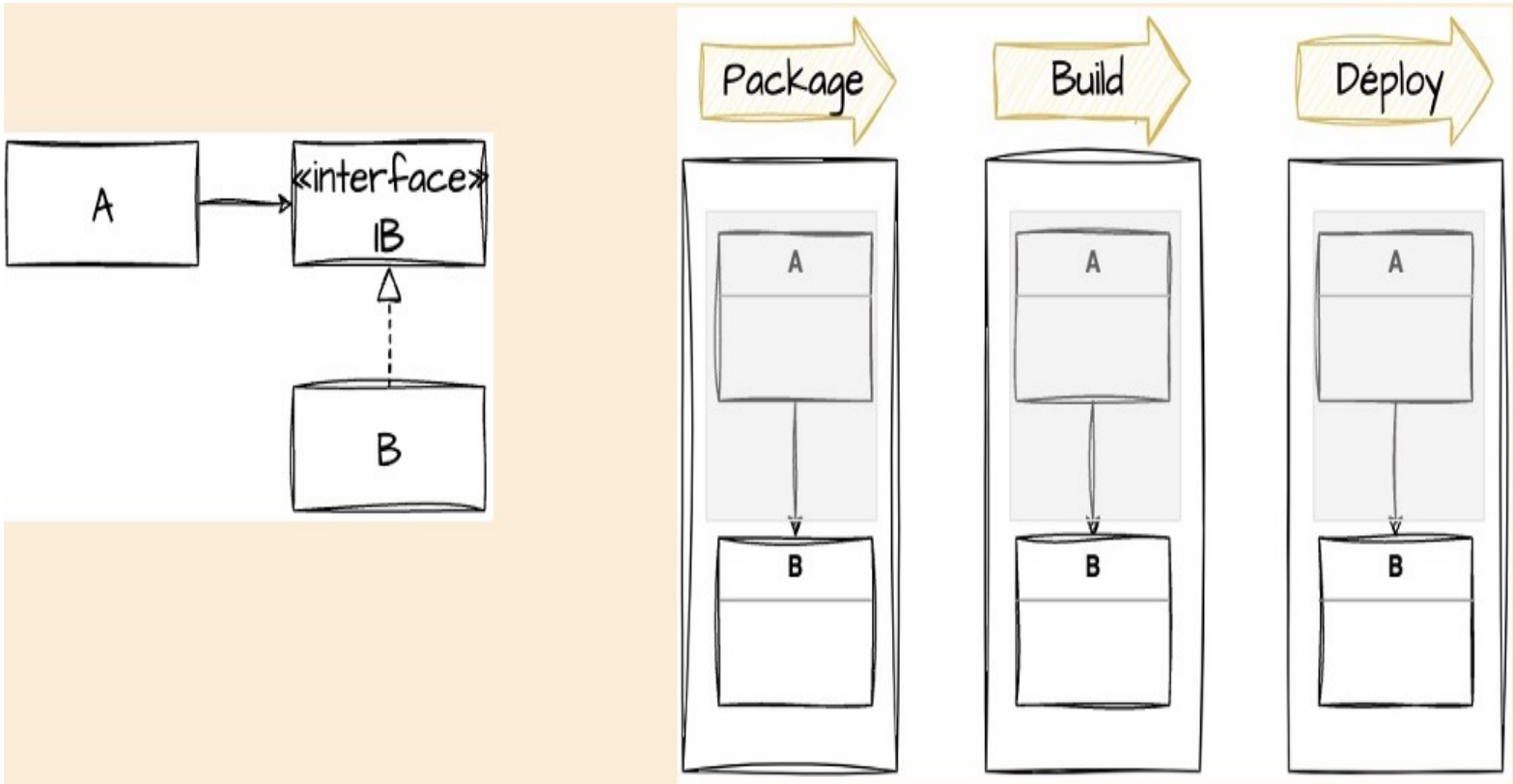
Que pouvez me dire de ce principe ?

Sans Inversion de dépendance



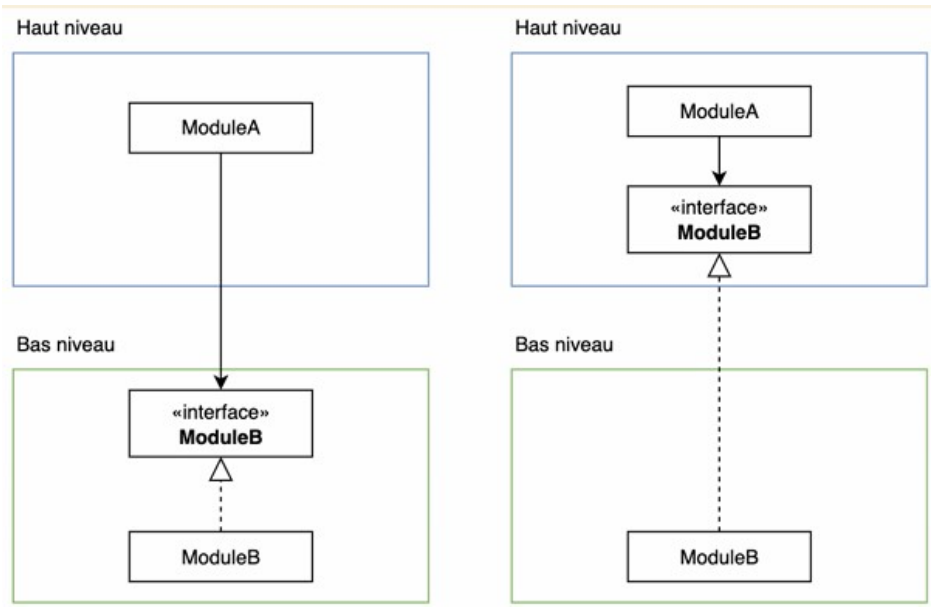
Inversion de dépendance

Avec Inversion de dépendance



- A ne dépend plus de B mais de son interface IB
- Donc si B est modifiée alors seulement B sera packagée, build et déployée chez le client.

Inversion de dépendance



À droite :

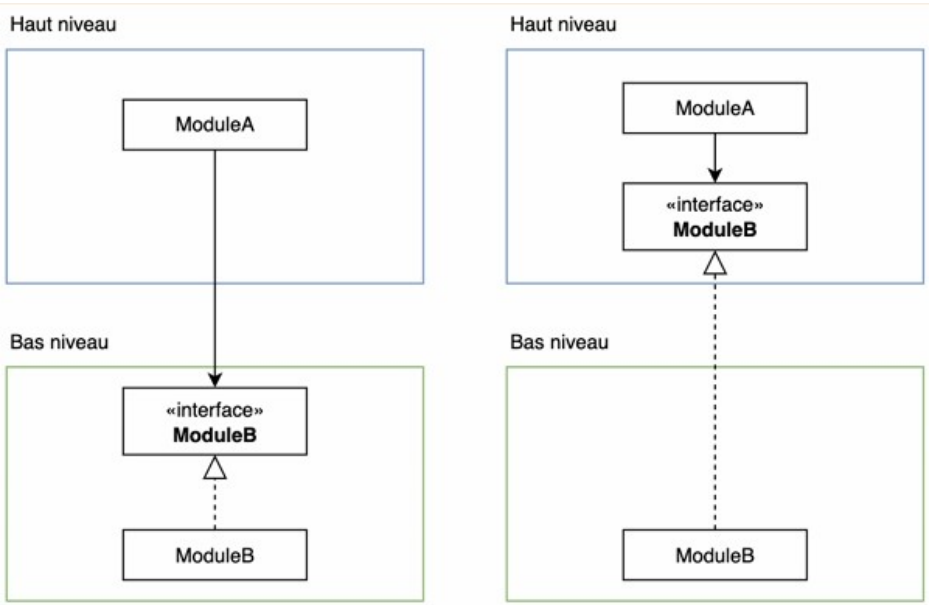
- C'est le module de bas niveau qui définit comment on interagit avec lui
- $\Rightarrow$ c'est le module de bas niveau qui dirige

À gauche :

- On change « la phrase »
- Le module de haut niveau dit les fonctionnalités requises par chaque implémentation
- $\Rightarrow$ c'est le module de haut niveau qui dirige

Inversion de dépendance

Amélioration ; MAIS . . .

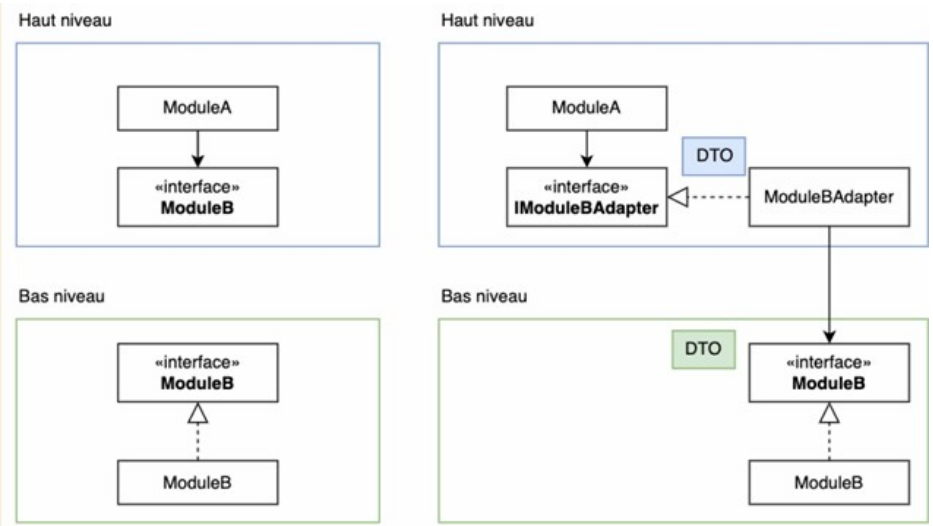


Avec cette conception :

- Il est impossible de réutiliser le module de bas niveau dans un autre contexte
- En effet, c'est le module de haut niveau qui dirige et le module de bas niveau subit l'interface.

Inversion de dépendance

Amélioration ; DONC . . .



On rajoute une interface :

- Les deux modules peuvent maintenant être réutilisés dans n'importe quel contexte

Faire le lien :

- Pour faire le lien entre les deux modules on utilise le patron adaptateur et la notion de DTO.

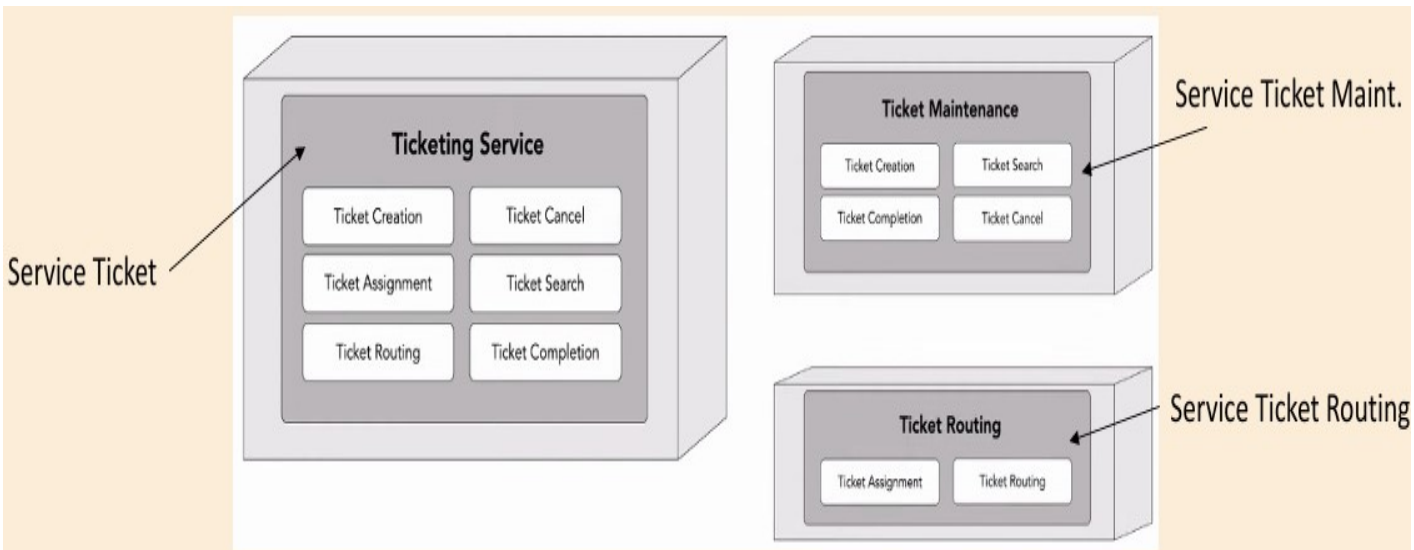
- L'adaptateur implémente l'interface du module de haut niveau
- L'adaptateur utilise l'interface du module de bas niveau
- L'adaptateur permet aussi de convertir les structuteurs de données entre les deux modules.

Service

Définition Service

Un service est une unité déployable qui accomplit une activité métier ou d'infrastructure.

Granularité d'un service ?



On fait quoi ? Un gros service ou plusieurs petits ?

La réponse . . . ça dépend !

Architecture microservices

- Une fonctionnalité métier
- Plusieurs centaines de services
- Forte interconnexion.

Architecture service-based

- Plusieurs fonctionnalités métiers
- De 3 à 12 services
- Eviter les interconnexions.

Un Module

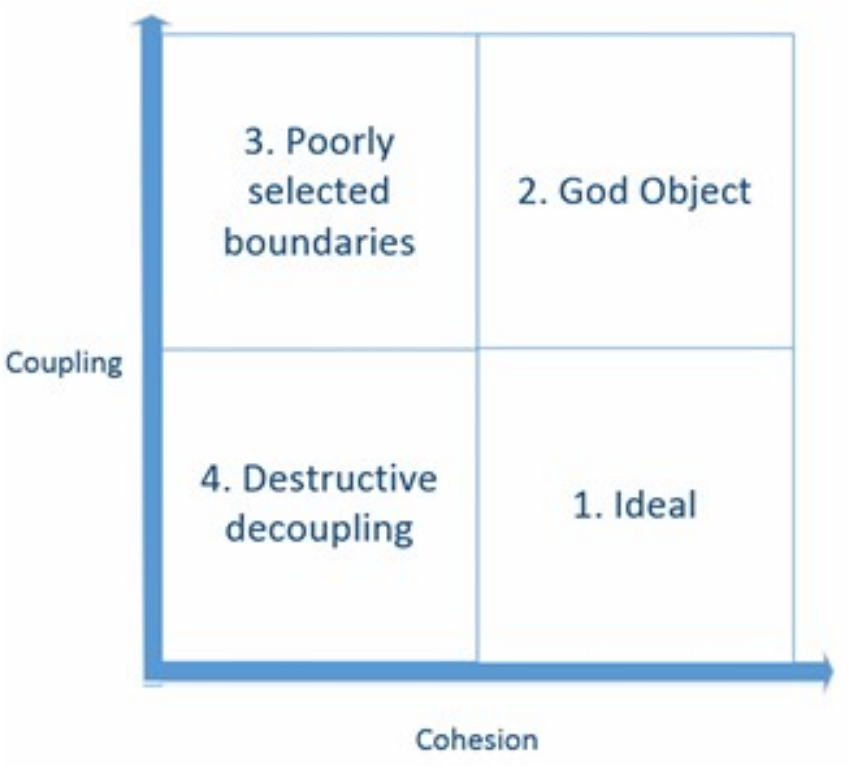
Définition module

Un module logiciel est une unité logicielle déployable, « manageable », réutilisable, composable et stateless qui fournit une interface concise au client.

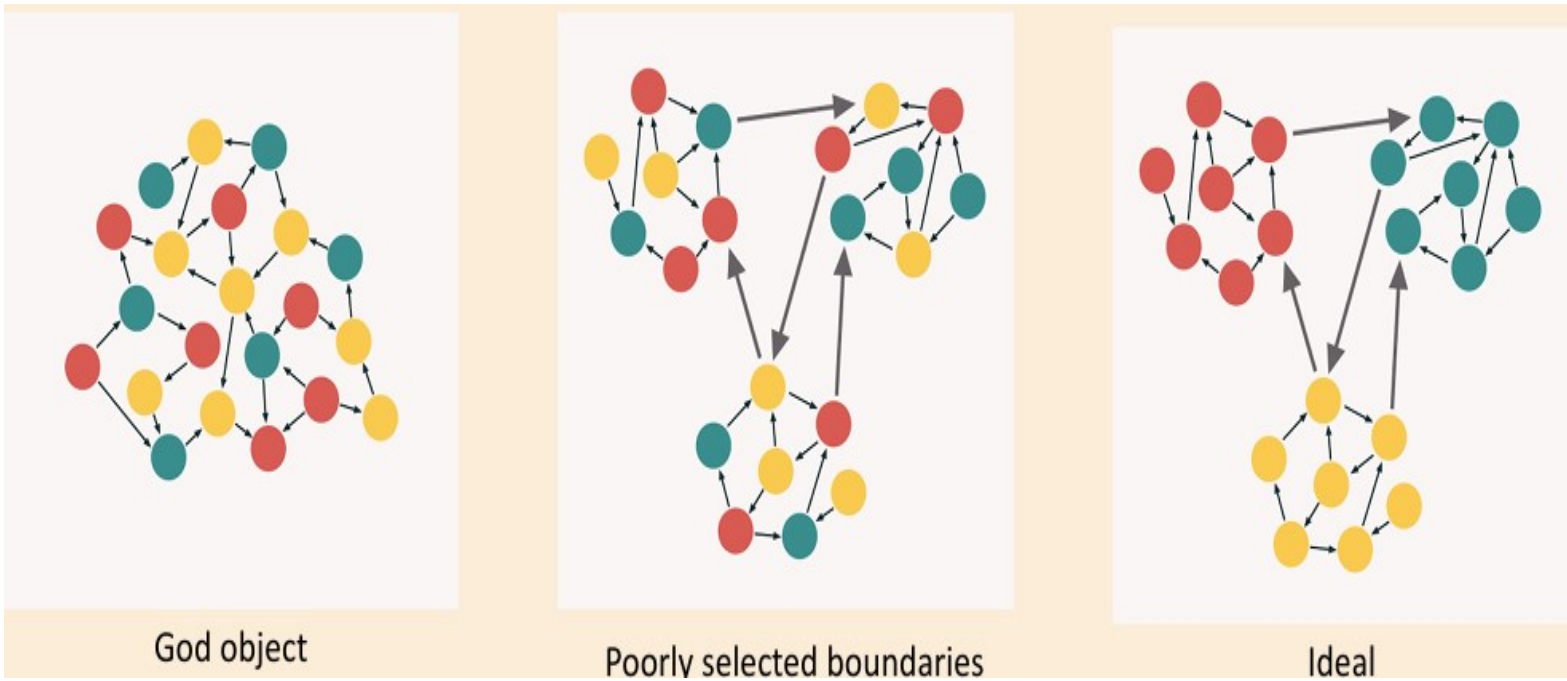
A noter que la modularité participe à la réussite d'un logiciel dans le temps et est un sujet très complexe.

L'objectif est

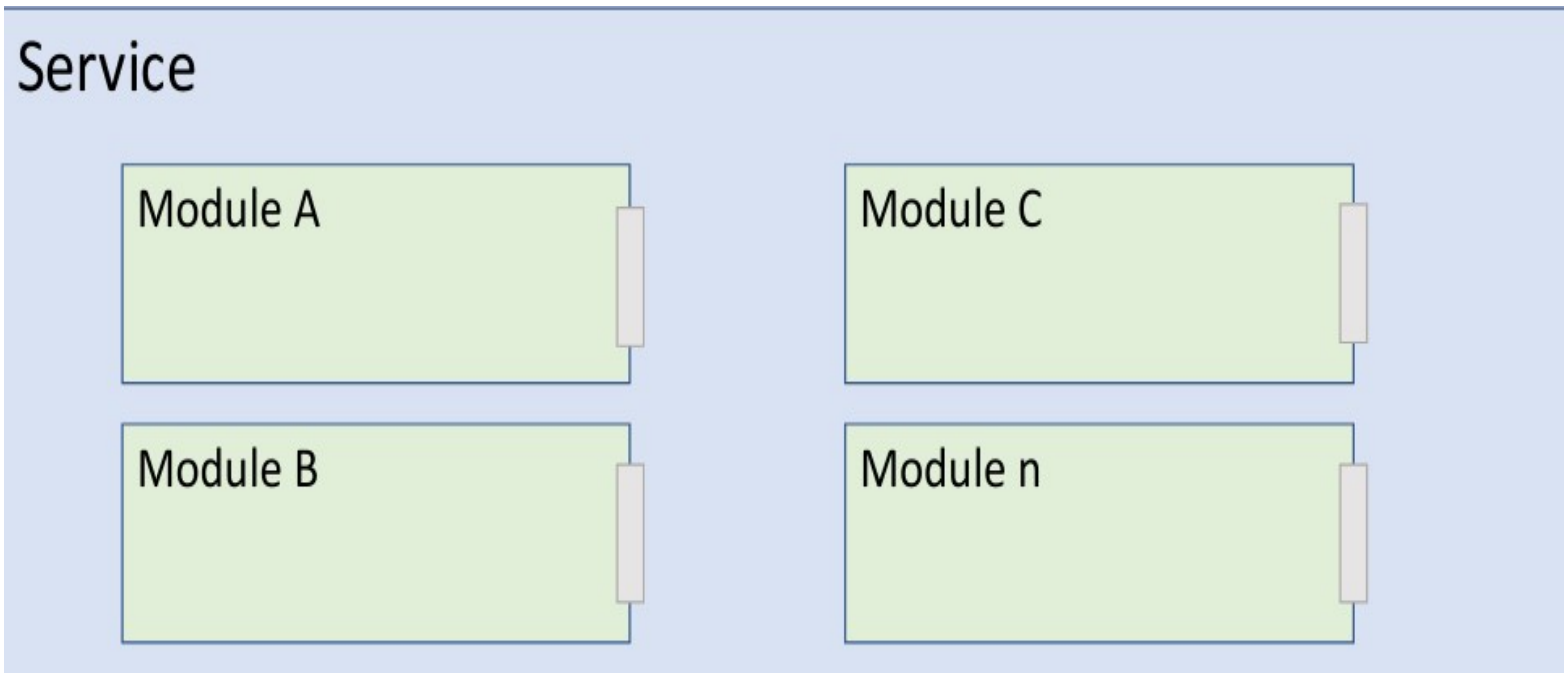
Respecter le principe « Low coupling, High cohésion ».



Illustration



Modules et Services



Conclusion

- Ces concepts sont utilisés partout, tout le temps
- Nous avons vu
  - Les interfaces : définissent comment on appel un service (paramètres) et qu'est-ce qu'on obtient d'un service (type de retour)
  - L'inversion de dépendance : éviter de tout package $\rightarrow$ **Build** $\rightarrow$ **Test** $\rightarrow$ **Deploy**
    - (Utilise la notion d'interface).
  - Qu'est-ce qu'un service : granularité différente suivant l'architecture
    - (Utilise la notion d'inversion de dépendance).

Plan

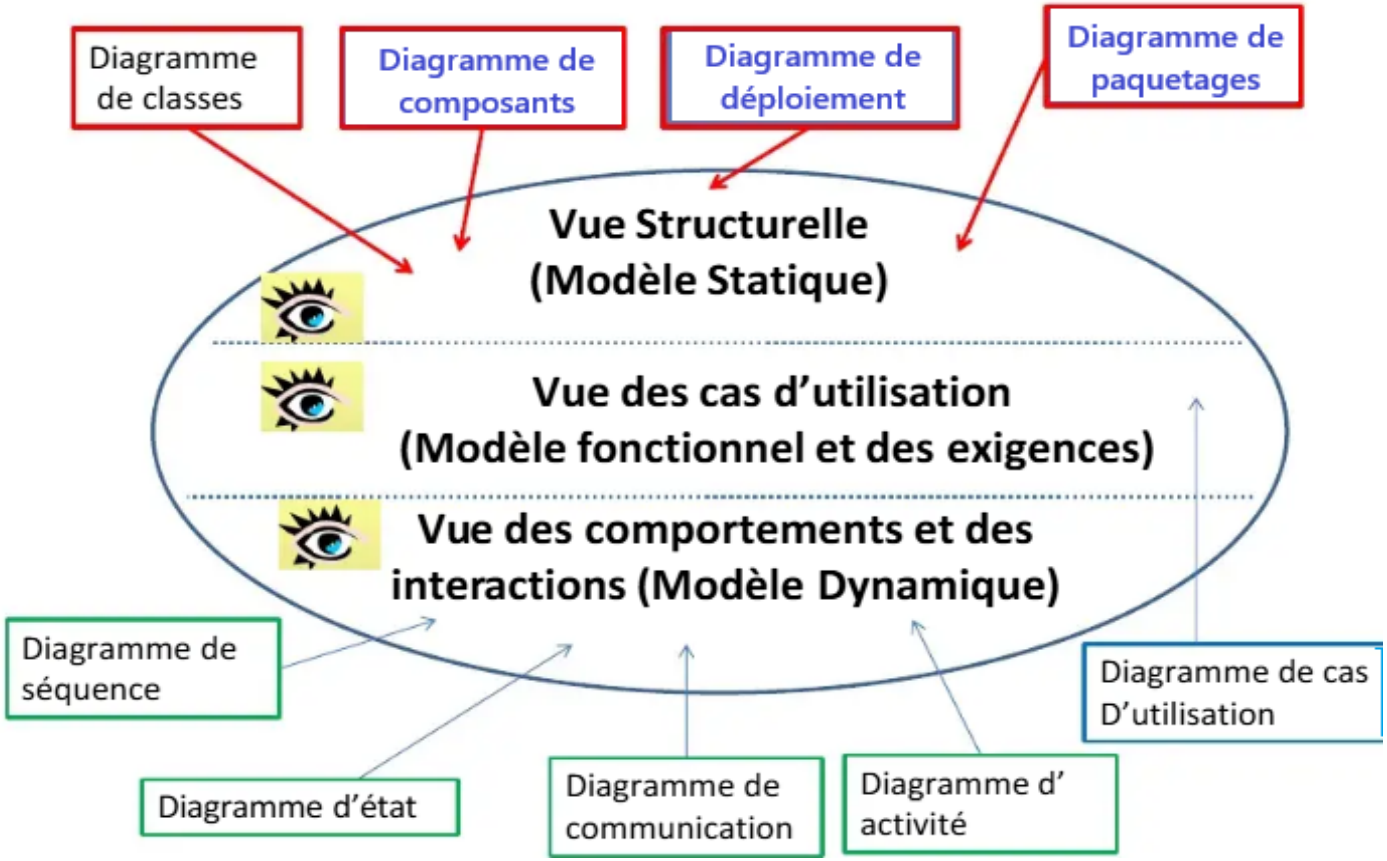
CONCEPT DES ARCHITECTURES LOGICIELLES

CONCEPTION UML D'UNE ARCHITECTURE LOGICIELLE

STYLES ET PATRONS ARCHITECTURAUX

ARCHITECTURE J2EE

Les diagrammes UML



Diagrammes Structurels ou Diagrammes statiques (1/)

- Diagramme de classes : il représente les classes intervenant dans le système
- Diagramme d'objets : il sert à représenter les instances de classes (objets) utilisées dans le système
- Diagramme de composants : il permet de montrer les composants du système d'un point de vue physique, tels qu'ils sont mis en œuvre (fichiers, bibliothèques, bases de données...)

Diagrammes Structurels ou Diagrammes statiques (2/)

- Diagramme de déploiement : il sert à représenter les éléments matériels (ordinateurs, périphériques, réseaux, systèmes de stockage...) et la manière dont les composants du système sont répartis sur ces éléments matériels et interagissent avec eux
- Diagramme des paquetages : permet de représenter la hiérarchie des paquetages du projet, leur organisation et leurs interdépendances, simplifie les diagrammes (donc plus simple à comprendre)
- Diagramme de structures composites : permet de décrire la structure interne d'un objet complexe lors de son exécution (c'est à dire, décrire l'exécution du programme), dont ses points d'interaction avec le reste du système.

Diagrammes Comportementaux ou Diagrammes dynamiques

- Diagramme des cas d'utilisation : il décrit les possibilités d'interaction entre le système et les acteurs, c'est-à-dire toutes les fonctionnalités que doit fournir le système
- Diagramme états-transitions : il montre la manière dont l'état du système (ou de sous-parties) est modifié en fonction des événements du système
- Diagramme d'activité : variante du diagramme d'états-transitions, il permet de représenter le déclenchement d'événements en fonction des états du système et de modéliser des comportements parallélisables (multithreads ou multi-processus)

Diagrammes Comportementaux ou Diagrammes dynamiques

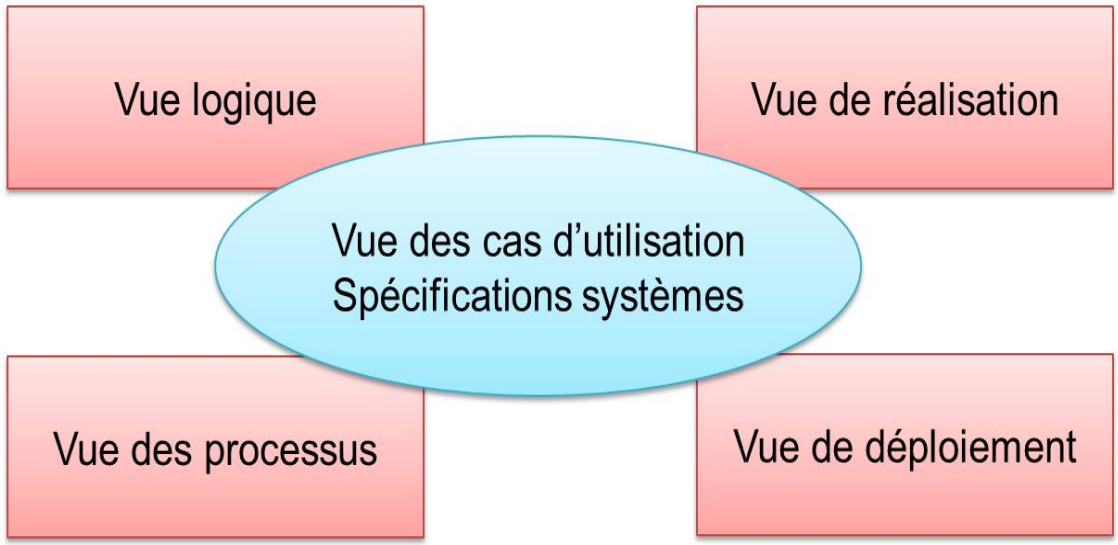
- Diagramme d'interactions
  - Diagramme de séquence : la représentation séquentielle du déroulement des traitements et des interactions entre les éléments du système et/ou des acteurs.
  - Diagramme de communication : la représentation simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets
  - Diagramme global d'interaction : variante du diagramme d'activité où les nœuds sont des interactions, permet d'associer les notations du diagramme de séquence à celle du diagramme d'activité, ce qui permet de décrire une méthode complexe
  - Diagramme de temps : la représentation des interactions où l'aspect temporel est mis en valeur ; il permet de modéliser les contraintes d'interaction entre plusieurs objets, comme le changement d'état en réponse à un événement extérieur

Le Rôle Crucial de la Documentation

Une documentation complète améliore

- la communication,
- la compréhension du système,
- facilite le transfert de connaissances aux nouveaux membres,
- Elle doit inclure le contexte, les objectifs architecturaux, les vues, et les exigences non fonctionnelles.

Le modèle « 4+1 » vues, dit de Kruchten



Le modèle de Kruchten dit modèle des 4 + 1 vues est celui adopté dans le Processus unifié. Ici encore, le modèle d'analyse, mentionné vue des cas d'utilisation, constitue le lien et motive la création de tous les diagrammes d'architecture.

Vue Logique

- Rôle : Décrit les classes clés, les objets, et les relations entre eux dans le système
- Objectif : Représenter l'espace problème (domaine métier) et l'espace solution (conception orientée objet). Elle permet de s'assurer que le système répond aux exigences fonctionnelles.
- Diagrammes UML : Le Diagramme de Classes et le Diagramme d'Objets sont les outils principaux pour exprimer cette vue.

La Vue des Processus (Systèmes Multitâches)

- Description : Décrit les interactions entre les processus, threads ou tâches. Elle exprime la synchronisation et l'allocation des objets
- Objectif : Vérifier le respect des contraintes de performance, d'efficacité et de fiabilité des systèmes multitâches
- Diagrammes UML : Exclusivement dynamiques (séquence, communication, états-transitions).

La Vue de Réalisation (Développement)

- Description : Permet de visualiser l'organisation des composants (code source, bibliothèques dynamiques et statiques) dans l'environnement de développement. Elle est utile pour la gestion de la configuration (auteurs, versions)
- Diagrammes UML : Uniquement les **diagrammes de composants**

La Vue de Déploiement (Environnement d'Exécution)

- Description : Représente le système dans son environnement d'exécution physique. Elle gère les contraintes géographiques, de bande passante, temps de réponse et tolérance aux pannes/fautes
- Diagrammes UML : Diagrammes de composants et diagrammes de déploiement.

Critères de documentation efficaces

- Utiliser un langage clair et simple pour être accessible à toutes les parties prenantes
- Les représentations visuelles (diagrammes UML, organigrammes) sont souvent plus efficaces que le texte pour transmettre des idées complexes.

Outils de Modélisation UML et de Documentation

- Les outils de création de diagrammes UML (comme Visio ou Lucidchart) facilitent la création de diagrammes,
- Pour la documentation structurée, des plateformes comme Atlassian Confluence peuvent être utilisées.

Approches de modélisation spécialisée

- Utilisation de langages de modélisation comme ArchiMate ou le C4 Model pour décrire l'architecture logicielle,
- Les plateformes no-code comme AppMaster permettent de concevoir visuellement l'architecture, réduisant le besoin d'une documentation écrite approfondie.

Plan

CONCEPT DES ARCHITECTURES LOGICIELLES

CONCEPTION UML D'UNE ARCHITECTURE LOGICIELLE

STYLES ET PATRONS ARCHITECTURAUX

ARCHITECTURE J2EE

Introduction

- Un style architectural est un modèle définissant comment sera le système.
- Comme les systèmes ont des points communs, ces systèmes auront des architectures qui se ressemblent. Le regroupement de ces architectures est appelé «style architectural».
- Un style architectural définit quels sont les composants, les connecteurs et les contraintes définissant l'architecture d'un système.

Définition

Définition

Un style architectural est un patron décrivant une architecture logicielle permettant de résoudre un problème particulier.

Un style architectural définit :

- Un ensemble de composants et de connecteurs, ainsi que leurs types
- Les règles de configuration des composants et connecteurs (topologie)
- Une spécification du comportement du patron
- Des exemples de systèmes construits selon ce patron.

Style architectural

Un style architectural correspond à un modèle validé dans la pratique et affiné par l'expertise de différents développeurs.

- **Compréhensibilité**, **maintenance**, **évolution**, **réutilisation**, **performance**, **documentation**, etc.
- Un style architectural permet de caractériser une famille de systèmes qui ont les mêmes propriétés structurelles et sémantiques.
- Un style architectural décrit une classe générique d'architecture telles que : client-serveur, pipe-filtre et architecture multi-couches.

Objectifs

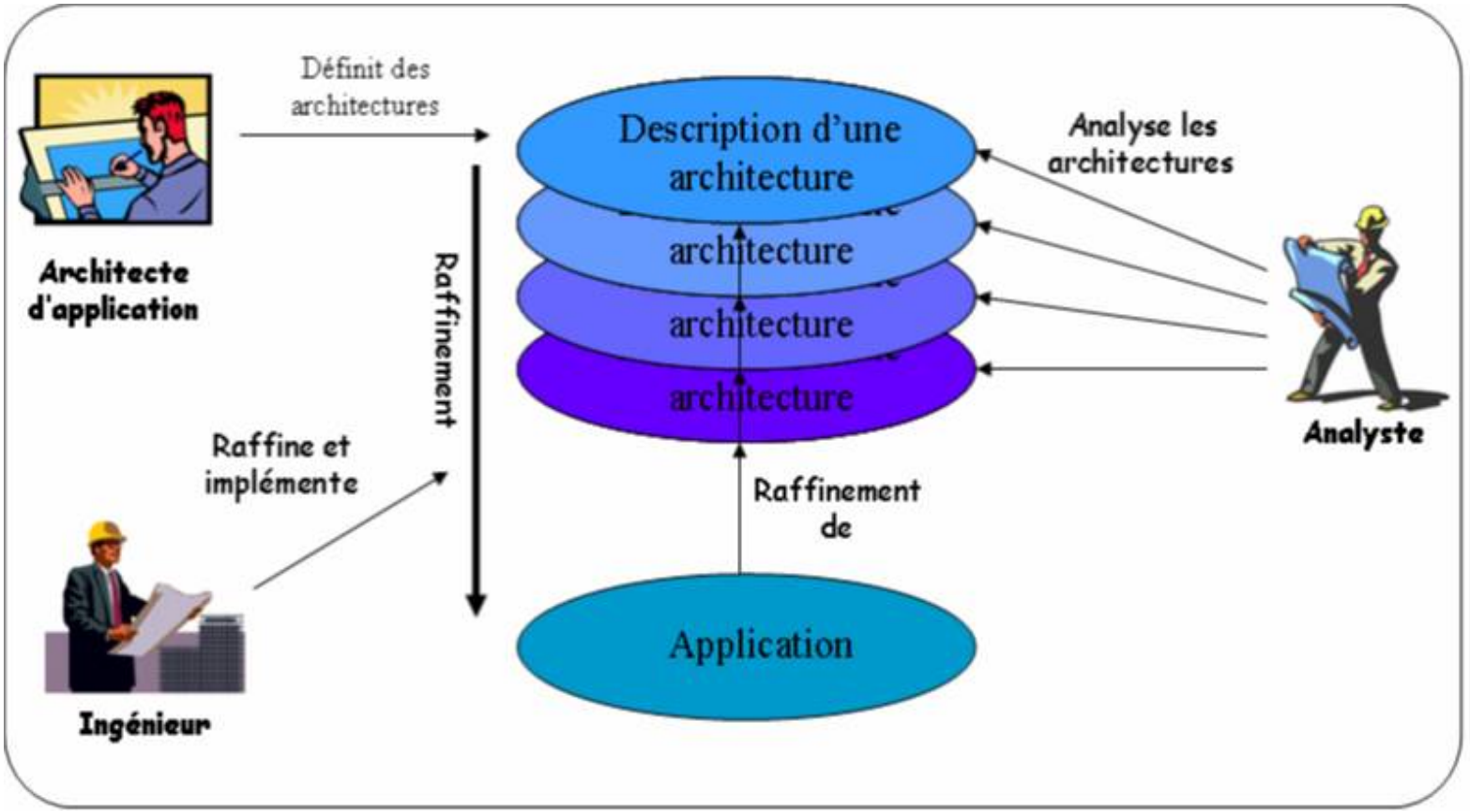
- Faciliter la conception et la réutilisation des logiciels en capitalisant sur l'expérience acquise dans la création de systèmes.
- Définir uniquement les contraintes essentielles (structure, comportement, ressources) sans imposer le niveau de détail d'une architecture spécifique.
- Fournir une vision globale du système avant sa mise en œuvre.
- Permettre au développeur de tirer profit de l'expérience accumulée par les concepteurs confrontés à des problèmes similaires.

Objectifs

L'utilisation des styles architecturaux comporte des intérêts précis :

- Favoriser la réutilisation dès la phase de conception du système.
- Faciliter la standardisation des architectures, améliorant ainsi la compréhension de l'organisation d'un système.
- Permettre des analyses adaptées au style architectural choisi.
- Rester indépendant des technologies : un même style peut être implémenté avec différentes plateformes (ex. serveurs Linux et clients Windows).
- Offrir la possibilité de combiner plusieurs styles au sein d'un même système.

Processus de Développement centré Architecture



Processus de Développement centré Architecture

Les acteurs du processus sont :

- l'architecte d'application,
- l'ingénieur,
- Ainsi qu'éventuellement l'analyste.

L'architecte a pour rôle de définir l'architecture qui servira de base au développement de l'application.

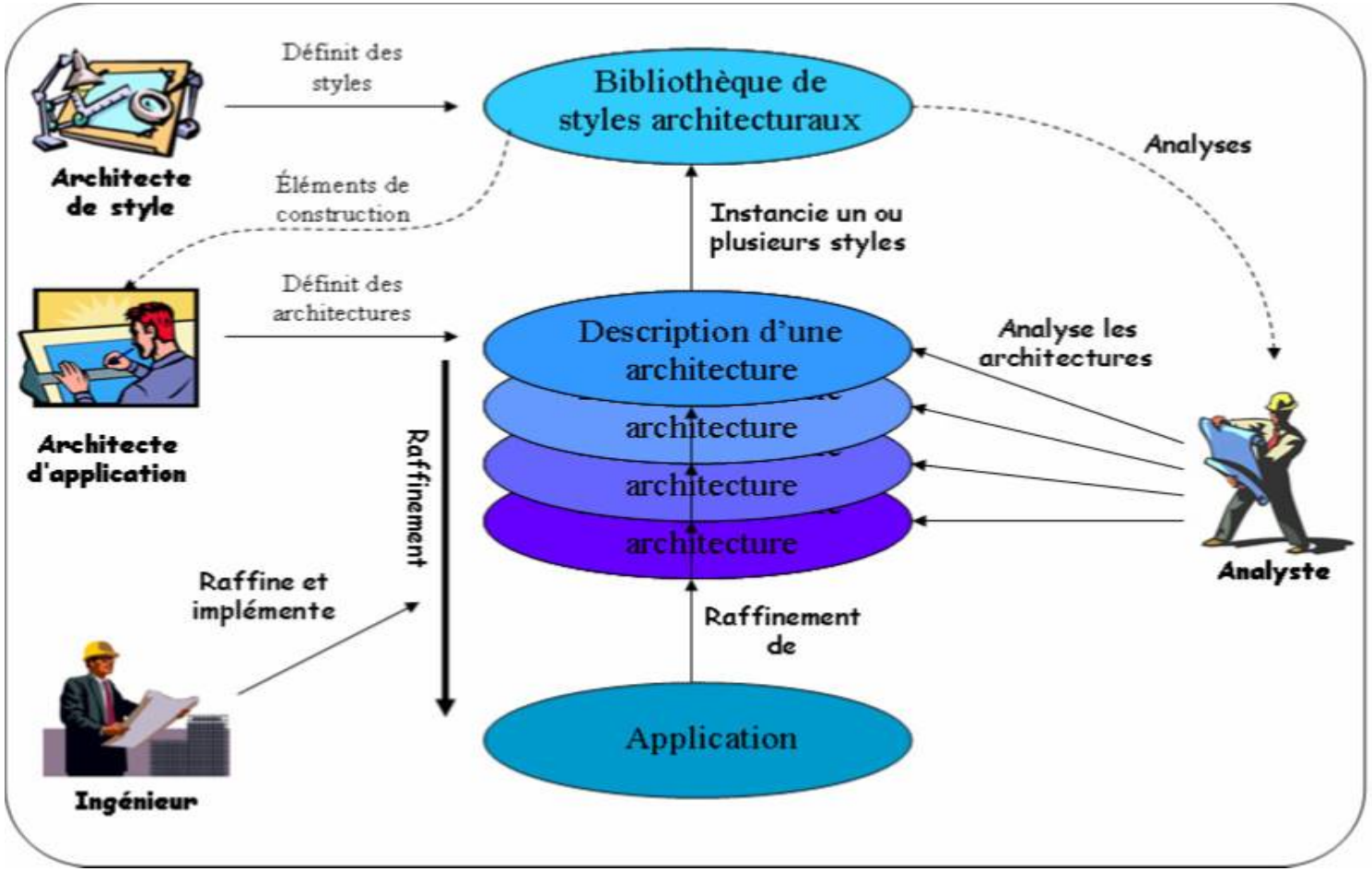
Processus de Développement centré Architecture

- L'ingénieur affine l'architecture en implémentant progressivement les composants et leurs interactions, tout en respectant la structure et les propriétés initiales.
- À chaque raffinement, l'analyste vérifie la conformité avec l'architecture supérieure afin d'assurer que l'application finale respecte les propriétés définies avec le client et les utilisateurs.

Processus de Développement centré Architecture

- Le nombre de raffinements varie selon l'application et dépend du niveau de précision requis avant le passage au code.
- Cette étape de raffinement est nécessaire pour les grosses applications qui ne peuvent pas être construites en une seule passe, mais on peut très facilement s'en passer pour des applications de plus petites tailles. Dans ce cas, l'application pourrait directement être implémentée à partir de la première architecture si celle-ci est assez précise.

Processus de développement architectural orienté style



Processus de développement architectural orienté style

Étapes du processus

- La formalisation du style architectural ou des styles architecturaux. Au fur et à mesure des définitions de styles, une bibliothèque peut être définie.
- L'architecte d'application instancie alors le ou les styles qui correspondent à ses besoins spécifiques et les utilise comme éléments de construction pour définir une architecture de base.
- La suite du développement est identique au processus précédemment étudié : l'ingénieur raffine petit à petit l'architecture de base en architectures de plus en plus concrètes jusqu'à l'obtention de l'application finale.

Processus de développement architectural orienté style

Remarque

- Ce processus de développement permet à l'analyste d'appliquer, aux différentes architectures, des analyses propres aux styles utilisés.
- Les analyses ne se limitent plus aux analyses génériques applicables à toutes les architectures.
- Elles peuvent également être spécifiques à un domaine particulier et ne s'appliquer qu'aux architectures définies avec les styles correspondants.

Styles standards

- Il existe plusieurs standards de styles architecturaux
- Les styles standards les plus connus sont «pipe/filtre», «client/serveur»
  «3 tiers/N-tiers», «architecture multi-couches», «SOA», « Shared data »

Styles standards Pipe/Filtre (1/)

- Le style pipeline Pipe/Filtre permet à l'information d'être traitée par plusieurs composants d'une manière séquentielle
- Le filtre est un composant qui traite l'information
- La pipe est un canal par lequel transite l'information.

Styles standards Pipe/Filtre (2/)

- Ce style convient bien aux systèmes de traitement et de transformation de données

  Composant = filtre ; Connecteur = canal

- **Le filtre** :
  - Traite indépendamment et asynchrone
  - Reçoit ses données d'un ou plusieurs canaux d'entrée, effectue la transformation/traitement des données et envoie les données de sortie produites sur un ou plusieurs canaux de sorties
  - Fonctionnent en concurrence : chacun traite les données au fur et mesure qu'il les reçoit.

Styles standards Pipe/Filtre (2/)

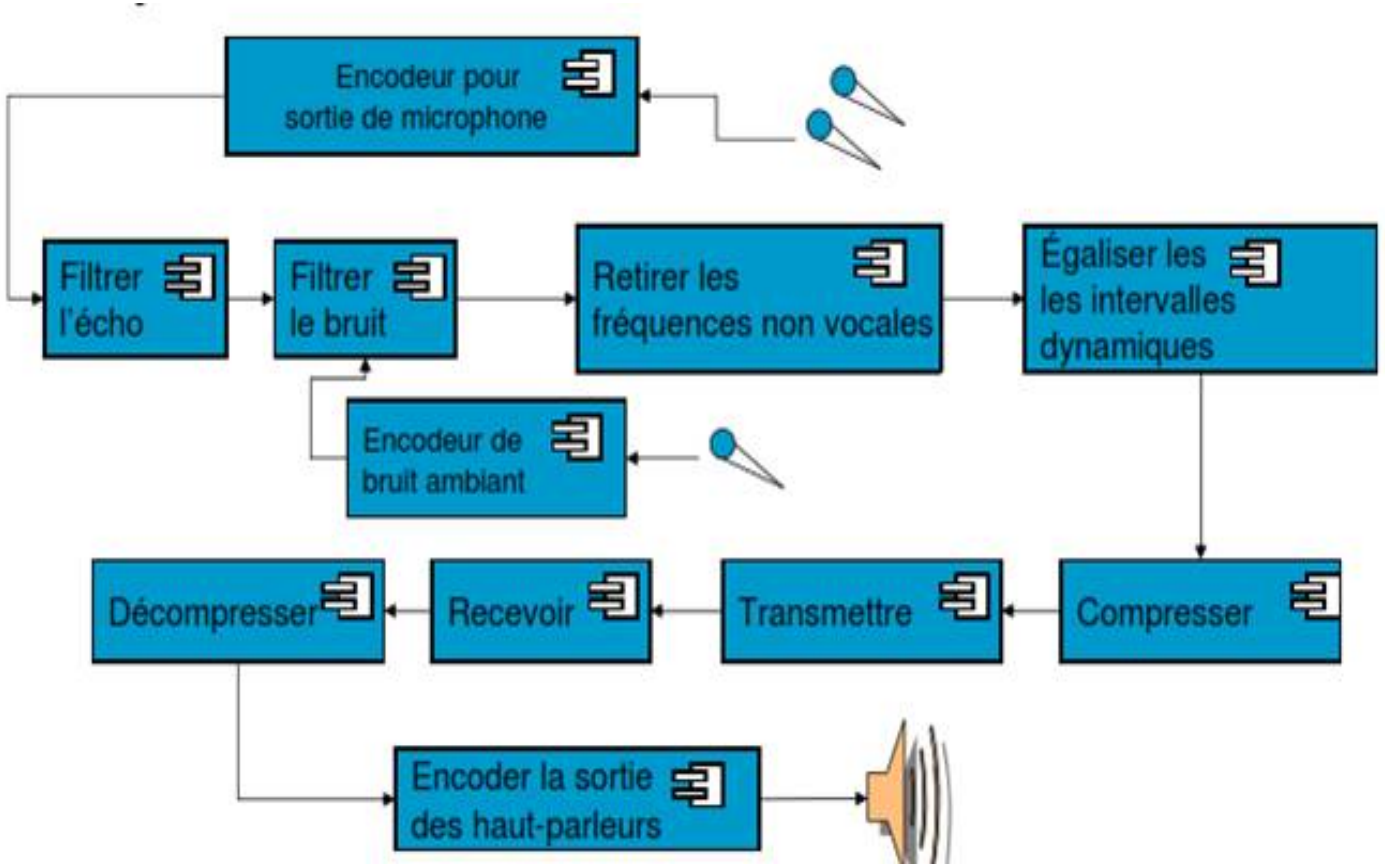
- **Le canal** :
  - Unidirectionnel au travers duquel circulent un flot de données (stream).
  - Synchronisation et utilisation d'un tampon parfois nécessaire pour assurer la bon fonctionnement entre filtre producteur et filtre consommateur.

Exemples

- Application de traitement de texte, de traitement de signaux. Compilateur (analyse lexicale, syntaxique, sémantique).

Styles standards Pipe/Filtre (3/)

Système de traitement du son :



Styles standards Pipe/Filtre (4/)

- Avantages du style Pipe/Filtre
  - Bon pour traitement en lot (batch)
  - Filtres faciles à réutiliser
  - Facilite la maintenance
  - La dépendance entre les filtres est faible
  - Analyse facilitée : performance, synchronisation, goulot
  - Se prête bien à la décomposition fonctionnelle d'un système.
- Inconvénients du style Pipe/Filtre
  - Ne convient pas aux applications à haute interactivité
  - Les performances dépendent des pipes.

Styles standards Pipe/Filtre (5/)

D'un point de vue conception

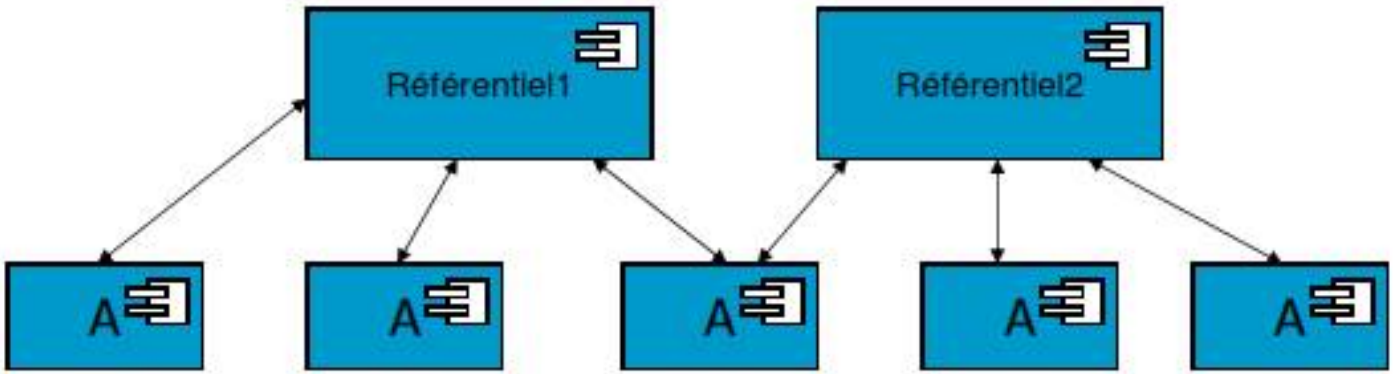
- Diviser pour régner : respect de cette stratégie car les filtres peuvent être conçus séparément
- Cohésion : les filtres sont un type de cohésion fonctionnelle
- Couplage : les filtres n'ont qu'une entrée et une sortie en général
- Abstraction : les filtres cachent généralement bien leurs détails internes
- Réutilisabilité : les filtres peuvent très souvent être réutilisés dans d'autres contextes
- Réutilisation : il est souvent possible d'utiliser des filtres déjà existants pour les insérer dans le pipeline.

Style standard shared data

Une Architecture avec référentiel de données (shared data) est utilisée dans le cas où des données sont partagées et fréquemment échangées entre les composants.

- Deux types de composants : référentiel de données et accesseur de données.
  - Les référentiels constituent le medium de communication entre les accesseurs.
- Connecteur : relie un accesseur à un référentiel
  - Rôle de communication, mais aussi possiblement de coordination, de conversion et de facilitation

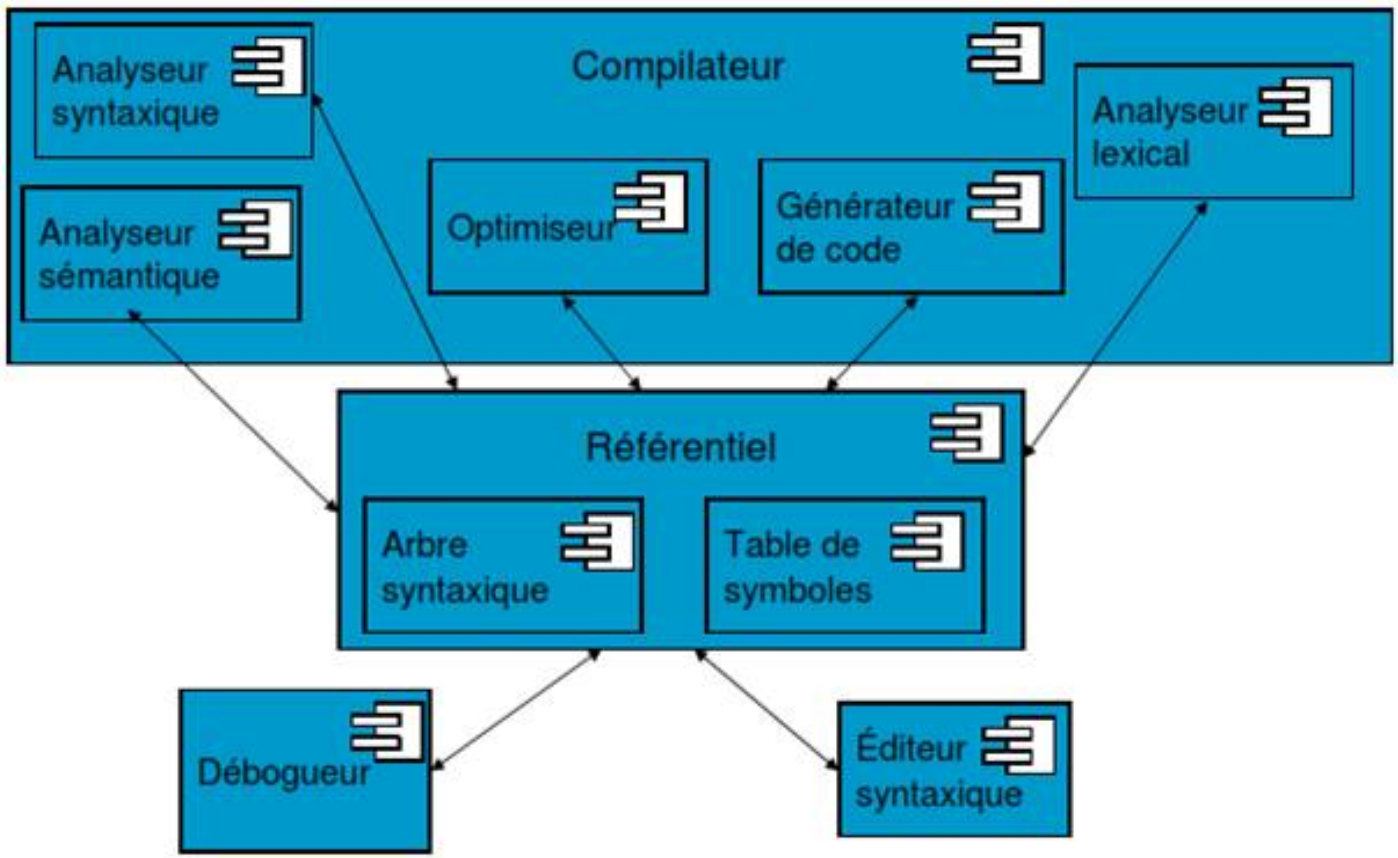
Style standard shared data



Exemples : Application de bases de données, systèmes Web, systèmes centrés sur les données (e.g. système bancaire, système de facturation ,etc.), environnement de programmation

Style standard shared data

Environnement de programmation



Style standard shared data

- Avantages
  - Avantageux pour les applications impliquant des tâches complexes sur les données, nombreuses et changeant souvent. Une fois le référentiel bien défini, de nouveaux services peuvent facilement être ajoutés.
- Inconvénients
  - Le référentiel peut facilement constituer un goulot d'étranglement, tant du point de vue de sa performance que du changement.

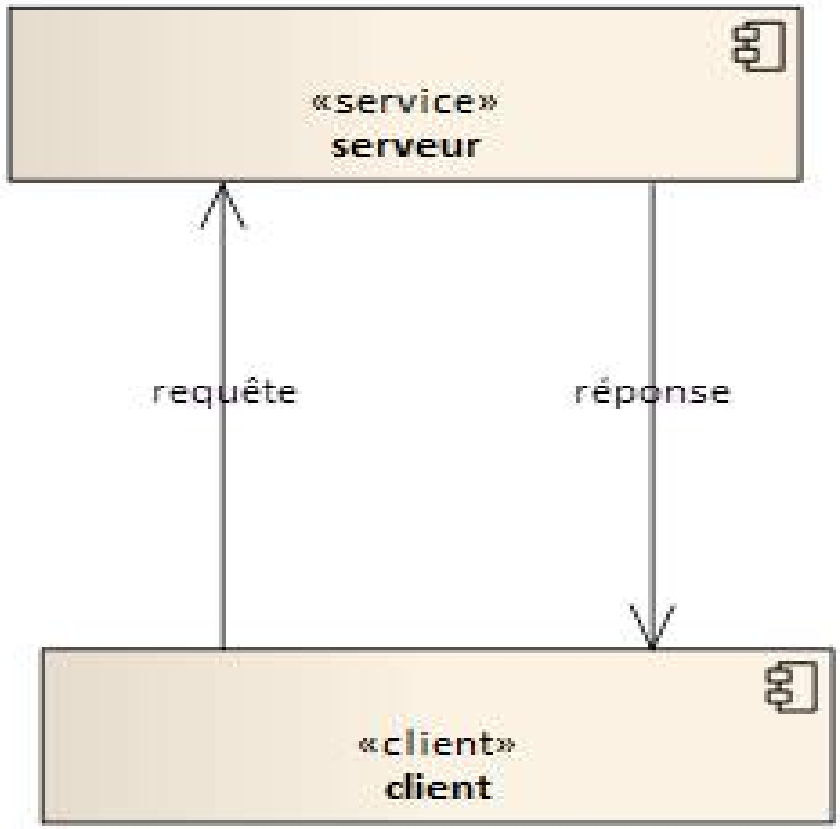
Style standard Client/Serveur

- Le style **Client/Serveur** permet de composer un système par deux composants principaux se trouvant généralement dans des machines séparées : le client et le serveur
- Le **Client** : processus demandant l'exécution d'une opération à un autre processus par envoi de message (Requête) contenant le descriptif de l'opération à exécuter et attendant la réponse de cette opération par un message en retour.
- Le **Serveur** : processus accomplissant une opération sur demande d'un client, et lui transmettant le résultat (Réponse).

Style standard Client/Serveur

- La Requête : message transmis par un client à un serveur décrivant l'opération à exécuter pour le compte du client.
- La Réponse : message transmis par un serveur à un client suite à l'exécution d'une opération, contenant le résultat de l'opération.
- L'interface utilisateur se trouve au niveau du client.

Style standard Client/Serveur



Style standard Client/Serveur

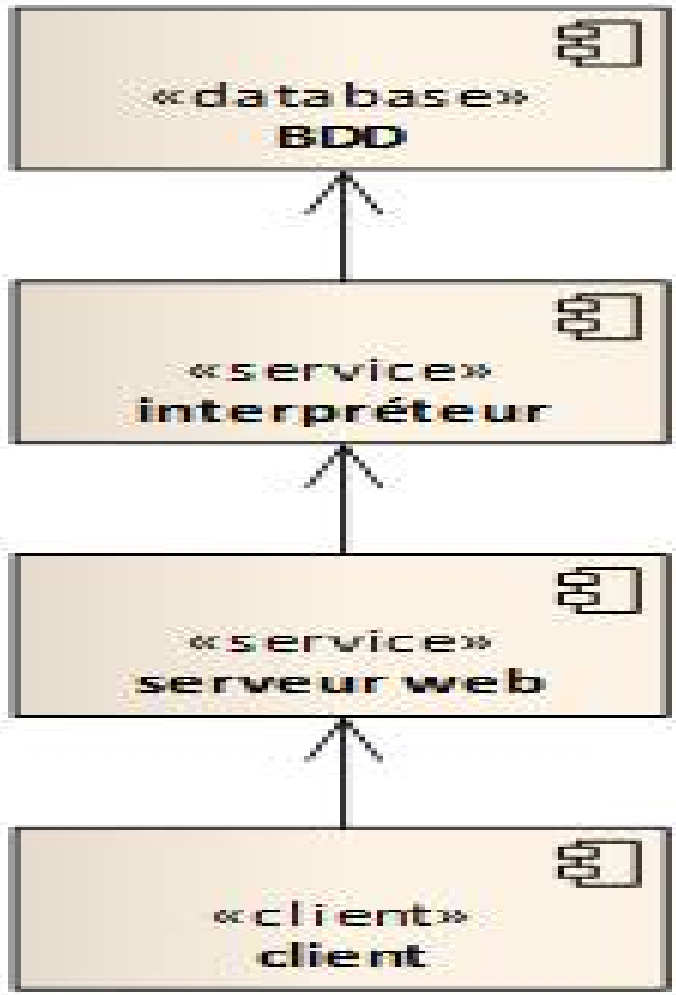
- Exemples
  - Serveur web ($\text{IIS}_{\text{internet information service}}$ / Apache), Client web (FireFox / Chrome / Internet Explorer)
  - Serveur FTP (ftpd) / Client FTP (FileZilla)
- Avantages
  - Séparation des tâches
  - Simple à utiliser
- Inconvénient
  - Souvent insuffisant pour des cas complexes.

Style standard N-Tiers

- Le style N-Tiers fragmente le système en plusieurs niveaux
- Chaque niveau dépend uniquement du niveau qui est au dessus
- Exemple : applications web modernes.

Style standard N-Tiers

Exemple d'une architecture 4 tiers



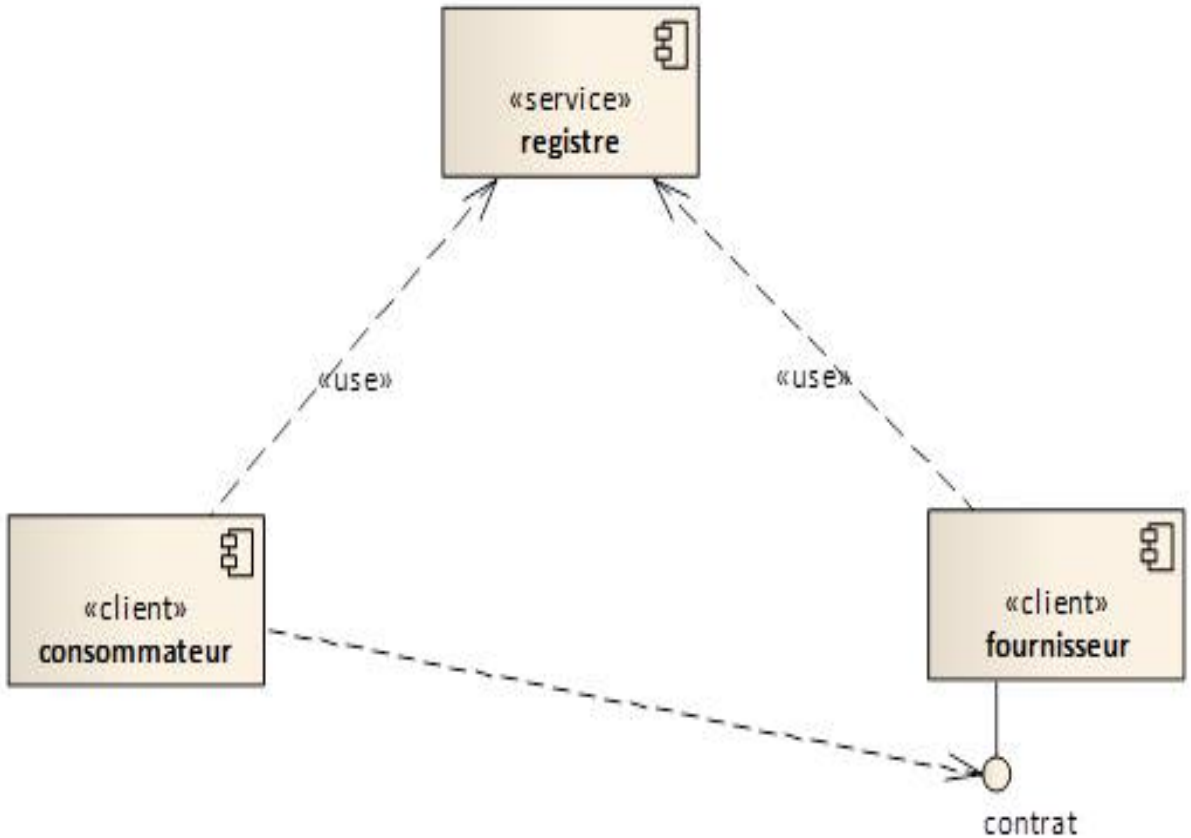
Style standard N-Tiers

- Avantages
  - Séparation poussée des tâches
  - Haute flexibilité
- Inconvénient
  - Nécessite des ressources matérielles importantes.

Style standard SOA

- Caractéristiques des services :
  - Les services sont autonomes
  - Les services sont composables : créer un service à partir d'autres services
  - Les services sont réutilisables
  - Les services permettent leur découverte.
- Composants de SOA :
  - Une architecture SOA est basée sur un consommateur de service, un fournisseur de service et un registre de services (Service Broker)
  - Le consommateur utilise le service
  - Le fournisseur assure le service
  - Le registre fait le lien entre le fournisseur et le consommateur.

Style standard SOA



Style standard SOA

- Avantages :
  - Indépendance et facilité de découverte de services
  - Permettent l'utilisation des applications depuis n'importe quel équipement (PC, mobile, etc. . .)
- Exemples :
  - Google Search Engine (web + application android + application iOs - iPhone OS-)
  - Youtube (web + application android + application WP7 -Windows Phone mobile client operating system- + application blackberry + application iOs)
  - Facebook (web + applications mobiles).

Modèle Architectural

Comment développer un modèle Architectural ?

- Commencer par faire une esquisse de l'architecture
  - En se basant sur les principaux requis des cas d'utilisation ; décomposition en sous-systèmes
  - Déterminer les principaux composants requis
  - Sélectionner un style architectural.
- Raffiner l'architecture
  - Identifier les principales interactions entre les composants et les interfaces requises
  - Décider comment chaque donnée et chaque fonctionnalité sera distribuée parmi les différents composants.
- Considérer chacun des cas d'utilisation et ajuster l'architecture pour qu'elle soit réalisable
- Détailler l'architecture et la faire évoluer.

Décrire l'architecture avec UML

- Tous les diagrammes UML peuvent être utiles pour décrire les différents aspects du modèle architectural
- Trois des diagrammes UML sont particulièrement utile pour décrire une architecture logicielle :
  - Diagramme de packages
  - Diagramme de composants
  - Diagramme de déploiement.

Plan

CONCEPT DES ARCHITECTURES LOGICIELLES

CONCEPTION UML D'UNE ARCHITECTURE LOGICIELLE

STYLES ET PATRONS ARCHITECTURAUX

ARCHITECTURE J2EE

Avant-propos

Java est reconnu actuellement comme l'un des meilleurs langages de programmation objet pour la réalisation de projets Internet complexes avec son API spécifique, Java EE.

Java EE

Java Enterprise Edition est un framework :

- Riche (**Java SE (Standard Edition)** + nombreuses **API**),
- Ouvert
- Dédié au développement, au déploiement et à l'exécution d'applications Internet modernes (nécessaires aux entreprises).

Les API Java : Fondements et Avantages

Une API (Application Programming Interface) est un ensemble de fonctions, de classes, de méthodes et de règles qui permettent à des programmes ou des applications de communiquer entre eux. En d'autres termes, une API définit comment les différents composants logiciels interagissent et s'échangent des informations.
Dans le contexte de Java :

- **API Java SE** : Comprend les bibliothèques de base pour le développement général en Java, telles que les classes pour les collections, les entrées/sorties, la gestion des threads, et plus encore.
- **API Java EE** : Spécialise Java pour les applications d'entreprise, avec des outils comme les servlets, l'accès aux bases de données, la gestion des transactions, et les services web.

Un thread

Un thread fait référence à une séquence d'instructions pouvant être exécutées indépendamment au sein d'un programme. Les threads permettent l'exécution simultanée et le multitâche en une seule application. Les fils partagent le même espace mémoire et les mêmes ressources que le processus auquel ils appartiennent, ce qui permet une communication et un partage de données efficaces.

- Par exemple, dans une application graphique, un thread peut gérer l'interface utilisateur tandis qu'un autre effectue des calculs en arrière-plan.

Serveur

Serveur : un ordinateur disposant d'un certain nombre de ressources qu'il met à disposition d'autres ordinateurs (clients) via le réseau.

- Serveur web
  - Programme s'exécutant sur une machine reliée à internet,
  - Protocole HTTP : répond aux requêtes des clients (navigateur web) et les traite,
  - Retourne des pages HTML au Client
- Serveur d'application

Si l'on s'en tient à une définition stricte, un serveur Web est un sous-ensemble commun d'un serveur d'applications.

Présentation de Java EE

Java EE est une plate-forme fortement orientée serveur pour le développement et l'exécution d'applications distribuées. Elle est composée de deux parties essentielles :

- Un ensemble de spécifications pour une infrastructure dans laquelle s'exécute les composants écrits en Java : un tel environnement se nomme serveur d'application.
- Un ensemble d'APIs qui peuvent être obtenues et utilisées séparément. Pour être utilisées, certaines nécessitent une implémentation de la part d'un fournisseur tiers.

Objectifs

- Concevoir une infrastructure d'application distribuée n'est pas une mince affaire,
- Faciliter la vie des développeurs en permettant l'encapsulation de la complexité inhérente aux environnements distribués

Pourquoi choisir Java EE

Les principaux avantages d'utiliser Java EE (et donc Java) sont :

- La portabilité
- L'indépendance
- La sécurité
- La multitude de librairies.

Pourquoi choisir Java EE

- L'architecture Java EE est intéressante car tous les éléments fondamentaux sont déjà en place. Pas besoin de concevoir une architecture, des librairies et des outils spécialement adaptés
- La plate-forme Java EE est basée sur des spécifications, ce qui signifie que les projets sont portables sur n'importe quel serveur d'applications conforme (Tomcat, JBoss, WebSphere...) à ces spécifications. Cette implémentation est gratuite et permet de bénéficier de la totalité de l'API sans investissement. La plate-forme Java EE est la plus riche des platesformes Java et offre un environnement standard de développement et d'exécution d'applications d'entreprise multi-tiers.

API JEE - Partie 1

	API JEE	version de l'API dans J2EE/Java EE					
		1.2	1.3	1.4	5	6	7
1	Entreprise Java Bean (EJB)	1.1	2.0	2.1	3.0	3.1	3.2
2	Remote Method Invocation (RMI) et RMI-IIOP	1.0					
3	Java Naming and Directory Interface (JNDI)	1.2	1.2	1.2.1			
4	**Java Database Connectivity (JDBC)**	2.0	2.0	3.0			
5	Java Transaction API (JTA) Java Transaction Service (JTS)	1.0	1.0	1.0	1.1	1.1	1.2
6	Java Messaging service (JMS)	1.0	1.0	1.1	1.1	1.1	2.0
7	**Servlets**	2.2	2.3	2.4	2.5	3.0	3.1
8	**Java Server Pages (JSP)**	1.1	1.2	2.0	2.1	2.2	2.2
9	**Java Server Faces (JSF)**				1.2	2.0	2.2
10	**Expression Language (EL)**					2.2	3.0
11	**Java Server Pages Standard Tag Libray (JSTL)**				1.2	1.2	1.2
12	JavaMail	1.1	1.2	1.3	1.4	1.4	1.4
13	J2EE Connector Architecture (JCA)		1.0	1.5	1.5	1.6	1.6
14	Java API for XML Parsing (JAXP)		1.1	1.2			
15	Java Authentication and Authorization Service (JAAS)		1.0				
16	JavaBeans Activation Framework		1.0.2	1.0.2			
17	Java API for XML-based RPC (JAXP-RPC)			1.1	1.1	1.1	1.1
18	SOAP with Attachments API for Java (SAAJ)			1.2	1.3		
19	Java API for XML Registries (JAXR)			1.0	1.0	1.0	1.0

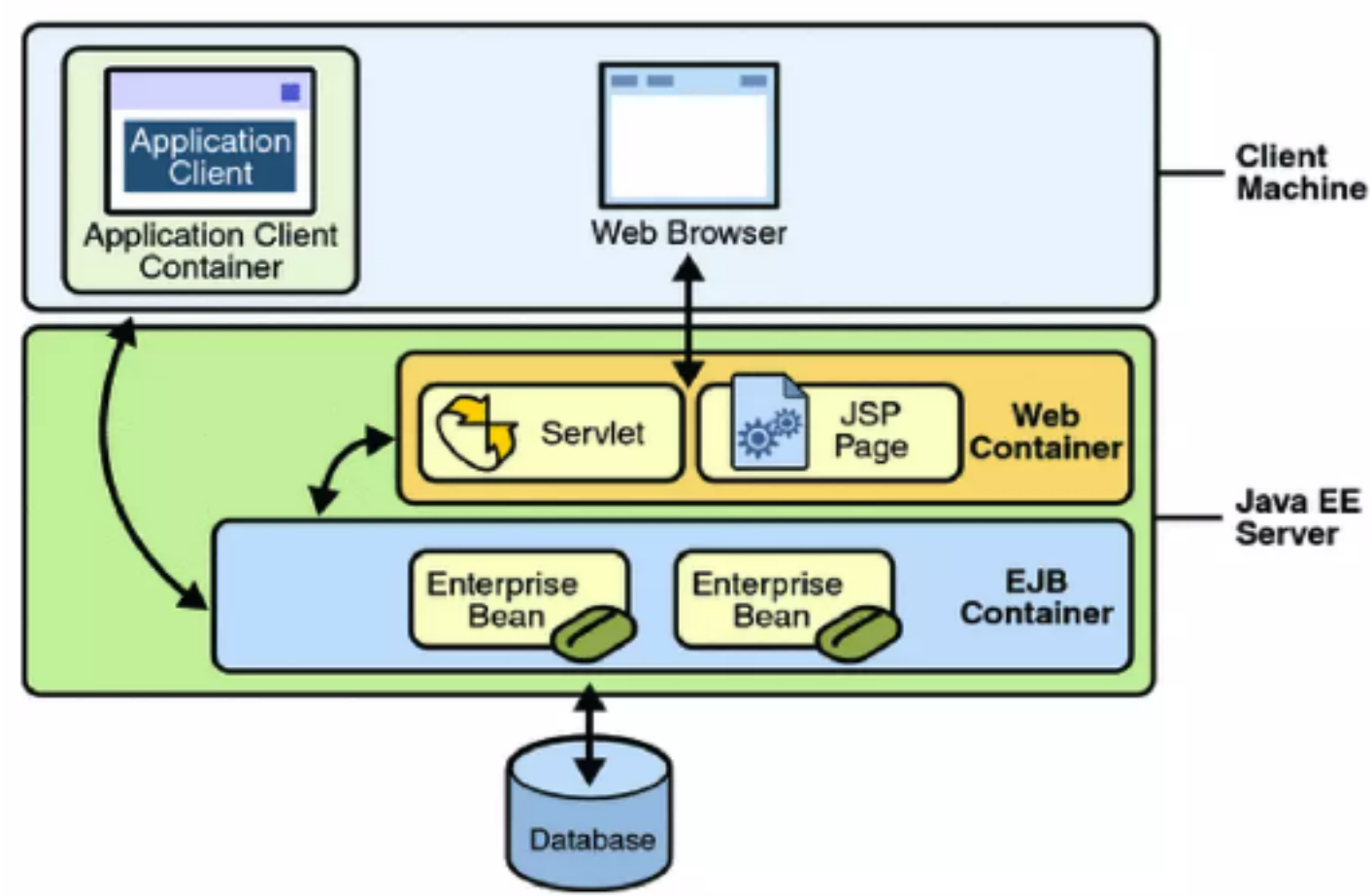
API JEE - Partie 2

	API JEE	version de l'API dans J2EE/Java EE					
		1.2	1.3	1.4	5	6	7
20	Java Management Extensions (JMX)			1.2			
21	Java Authorization Service Provider Contract for Containers (JACC)			1.0	1.1	1.4	1.4
22	Java API for XML-Based Web Services (JAX-WS)				2.0	2.2	2.2
23	Java Architecture for XML Binding (JAXB)				2.0	2.2	
24	Streaming API for XML (StAX)				1.0		
25	**Java Persistence API (JPA)**				1.0	2.0	2.1
26	Java API for RESTful Web Services (JAX-RS)					1.1	2.0
27	Web Services				1.2	1.3	1.3
28	Web Services Metadata for the Java Platform					2.1	2.1
29	Java APIs for XML Messaging (JAXM)					1.3	
30	Context and Dependency Injection (CDI)					1.0	1.1
31	**Dependancy Injection (DI)**					1.0	
32	Bean Validation					1.0	1.1
33	Managed beans					1.0	1.0
34	Interceptors					1.1	1.1
35	**Common Annotations**					1.1	1.1
36	Java API for WebSocket						1.0
37	Java API for JSON						1.0
38	Concurrency Utilities for Java EE						1.0
39	Batch Applications for the Java Platform						1.0

L'environnement d'execution des applications J2EE (1/2)

- Pour exécuter ces composants de natures différentes, Java EE définit des conteneurs pour chacun d'eux.
- Les conteneurs permettent aux applications d'accéder aux ressources et aux services en utilisant les API définies.
- Les appels aux composants se font par des clients en passant par les conteneurs.
- Les clients n'accèdent pas directement aux composants, mais sollicitent le conteneur pour les utiliser.

L'environnement d'execution des applications J2EE (2/2)



Les conteneurs

Les conteneurs assurent la gestion du cycle de vie des composants qui s'exécutent en eux.
Il existe plusieurs conteneurs définis par JEE

- Conteneur Web : pour exécuter des servlets et des JSP.
- Conteneur d'EJB : pour exécuter les EJB.
- Conteneur client : pour exécuter des applications standalone sur les postes qui utilisent des composants JEE.

Les serveurs d'application peuvent fournir :

- Un conteneur web uniquement
- Ou un conteneur d'EJB uniquement
- Ou les deux.

Deploiement d'une application (1/2)

Pour deployer une application dans un conteneur, il faut lui fournir deux elements :

- **L'application** comprend tous les composants nécessaires à son exécution, tels que les classes compilées, les bibliothèques, les fichiers de configuration, les ressources (comme les images, les fichiers CSS/JS, etc.).
- Ces composants sont généralement regroupés dans une archive spécifique, selon le type d'application et le conteneur utilisé. Par exemple :
  - Pour les applications Java : le format d'archive est souvent un JAR (Java ARchive) ou un WAR (Web Application Archive) pour les applications web.
  - Pour les conteneurs Docker : l'application est souvent packagée sous la forme d'une image Docker.

Deploiement d'une application (2/2)

Pour deployer une application dans un conteneur, il faut lui fournir deux elements :

- **Fichier Descripteur de Déploiement**
  - Ce fichier est utilisé pour configurer le conteneur et spécifier des options d'exécution pour l'application.
  - Pour les applications Java EE, ce fichier est généralement nommé *web.xml* et contient des informations sur les servlets, les mappings, les paramètres d'initialisation, etc.
  - Dans le contexte de conteneurs Docker, le descripteur de déploiement peut être un fichier Dockerfile ou un fichier de configuration *docker-compose.yml* pour orchestrer plusieurs conteneurs.

Serveur JEE

Un site web est un ensemble constitué de pages web (elles-mêmes faites de fichiers HTML, CSS, JavaScript, etc.).
On distingue deux types de sites :

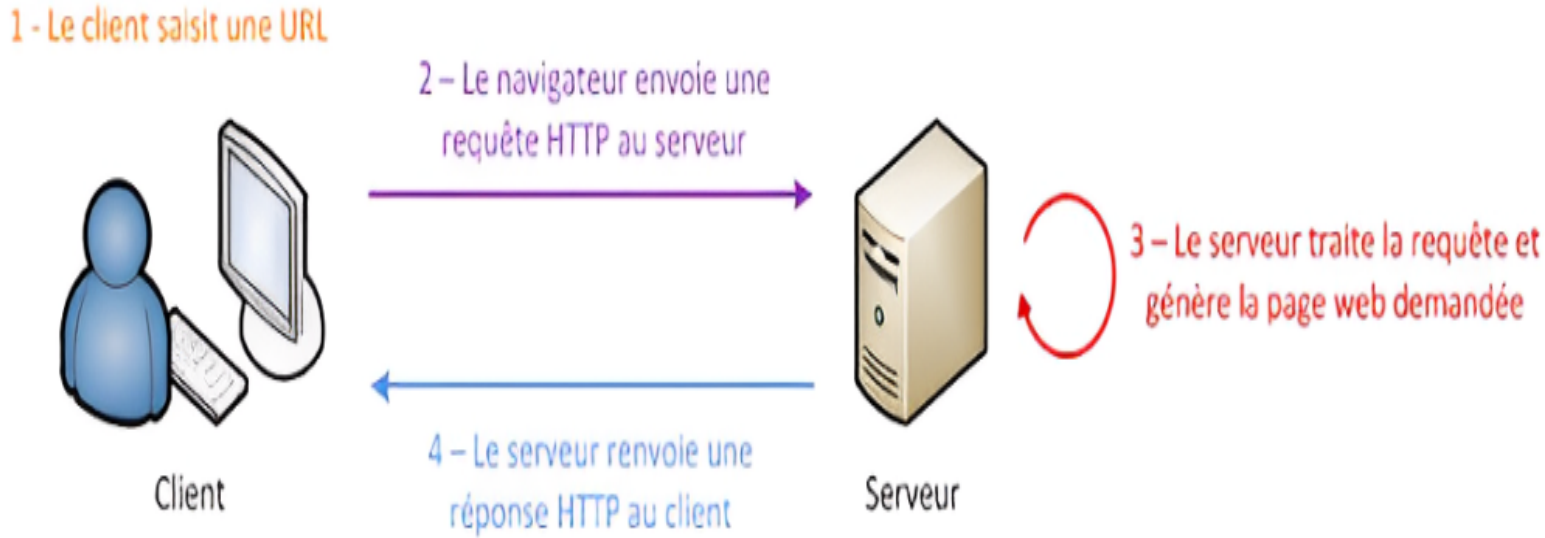
- Les sites internet statiques : ce sont des sites dont le contenu est "fixe". Il n'est modifiable que par le propriétaire du site. Ils sont réalisés à l'aide des technologies HTML, CSS et JavaScript uniquement.
- Les sites internet dynamiques : ce sont des sites dont le contenu est "dynamique". En plus des langages précédemment cités, ils font intervenir d'autres technologies (Java EE, PHP, etc.).

Serveur JEE

La communication **client/serveur** qui s'effectue entre le client et le serveur est régie par des règles du protocole HTTP :

1. L'utilisateur saisit une URL dans la barre d'adresses de son navigateur ;
2. Le navigateur envoie alors une requête HTTP au serveur pour lui demander la page correspondante ;
3. Le serveur reçoit cette requête, l'interprète et génère ensuite une page web qu'il renvoie au client par le biais d'une réponse HTTP ;
4. Le navigateur reçoit cette réponse, qui contient la page web finale, et l'affiche alors à l'utilisateur.

Serveur JEE


1 - Le client saisit une URL
2 – Le navigateur envoie une requête HTTP au serveur
3 – Le serveur traite la requête et génère la page web demandée
4 – Le serveur renvoie une réponse HTTP au client
Client
Serveur

Serveur JEE

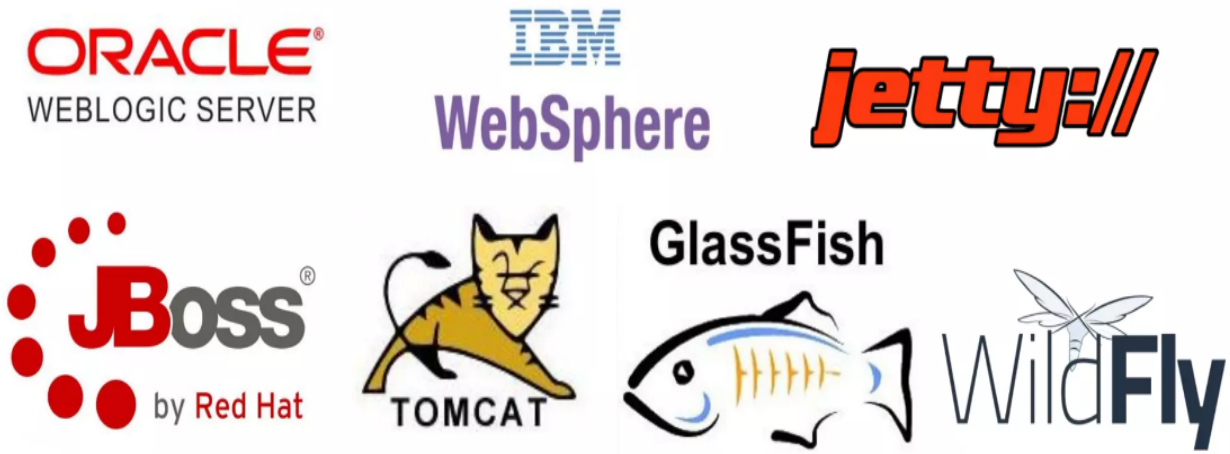
- Le client ne comprend que les langages de présentation tels que : HTML, CSS et JavaScript (ces langages sont interprétés directement par le navigateur sur la machine client).
- Le serveur génère dynamiquement des pages à envoyer au client. Il dispose de technologies (JEE, PHP, .NET, Django, etc.) capables de :
  - Analyser les données reçues via HTTP
  - Transformer les données
  - Enregistrer les données dans une base de données
  - Intégrer les données dans le design des pages
  - etc.

Serveur JEE

- Serveur HTTP : c'est un serveur qui gère exclusivement des requêtes HTTP.
- Un serveur web JEE est constitué d'un serveur HTTP (web) et d'un conteneur web.
- Un serveur d'applications JEE est constitué d'un serveur HTTP, d'un conteneur web et d'un conteneur EJB.

Serveur JEE

- Les solutions propriétaires et payantes : WebLogic (Oracle) et Websphere (IBM) sont les références dans le domaine, utilisées dans les banques et la finance, elles sont à la fois robustes, finement paramétrables et très coûteuses.
- Les solutions libres et gratuites : Apache Tomcat, JBoss, GlassFish, JOnAS sont les principaux représentants.



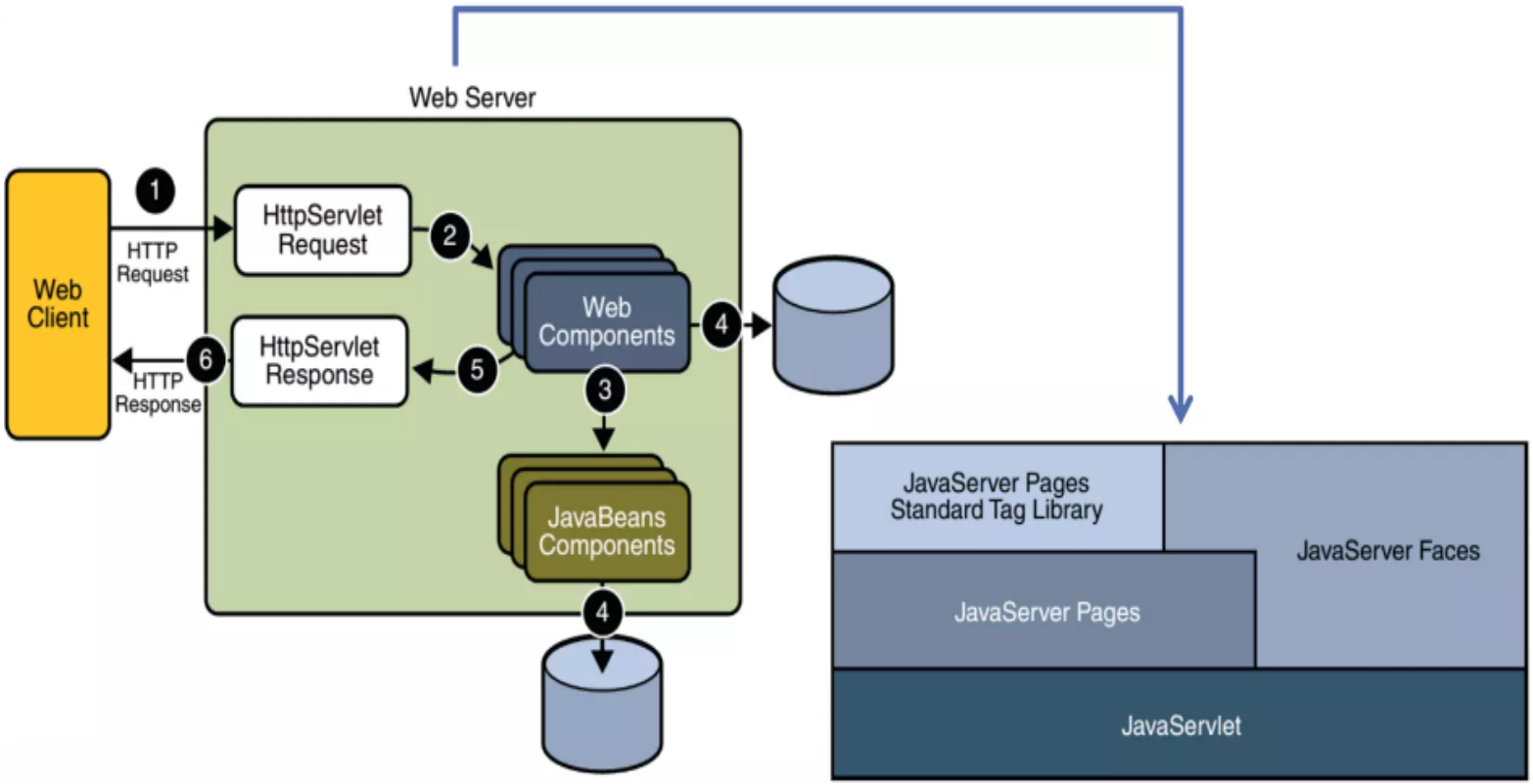
Développement d'application web JEE

- Environnement de Développement Integré (IDE)
  - Eclipse IDE for Java EE developers
  - autres : Netbeans, etc
- Serveur web
  - Apache Tomcat
  - autres : Wildfly, GlassFish, etc
- Serveur BD
  - Serveur MySql avec Workbench
  - autres : Postgres, SQLServer, Oracle, etc.

Application web JEE (1/2)

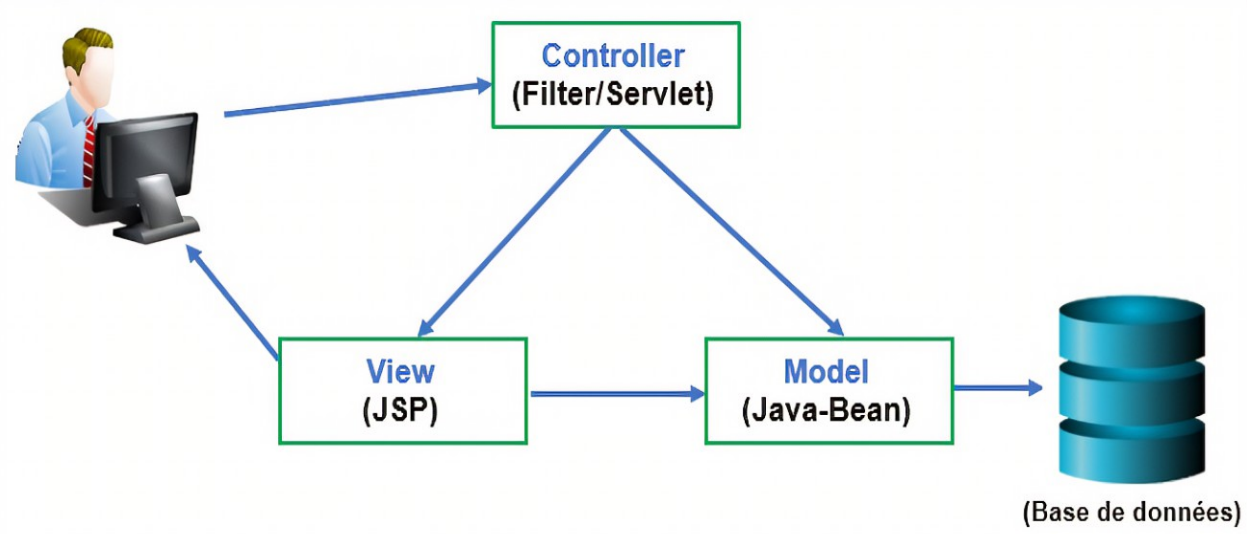
- Les Servlets et les JSPs sont la base de JEE.
- Les frameworks **Modèle-Vue-Contrôleur(MVC)** les plus populaires aujourd'hui sont à la base des Servlets et JSPs :
  - Spring
  - JSF
  - Struts

Application web JEE (2/2)



Le modèle MVC (Modèle-Vue-Contrôleur) (1/3)

Le modèle MVC (Model-View-Controller) est un modèle d'architecture utilisé pour structurer les applications. En Java EE, ce modèle est souvent utilisé pour créer des applications web en relation avec les concepts Servlets, JSP (Java Server Pages), et Java Beans.



Le modèle MVC (2/3)

Le modèle MVC divise une application en trois composants principaux :

- **Modèle** : Représente la logique métier et les données de l'application. Il ne contient pas de code lié à l'affichage ou aux interactions avec l'utilisateur. En JEE, un Java Bean ou des classes métiers sont souvent utilisés pour représenter ce modèle.
- **Vue** : La partie qui s'occupe de l'interface utilisateur, c'est-à-dire ce que l'utilisateur voit. Dans le contexte de JEE, les JSP sont souvent utilisées pour créer cette partie de la vue. Elles génèrent du code HTML dynamique en fonction des données envoyées par le contrôleur.

Le modèle MVC (3/3)

- **Contrôleur** : Comme un intermédiaire entre le modèle et la vue. Il traite les requêtes des utilisateurs, interagit avec le modèle pour récupérer ou manipuler les données, et envoie ensuite les données à la vue pour l'affichage. En JEE, les Servlets jouent le rôle de contrôleur.

Application web JEE

- Une application web est composée de servlets, JSP, html, et d'autres ressources.
- La portabilité de telles applications impose :
  - Une structure commune
  - Une description standard de leur déploiment
- Toutes les ressources d'une application web sont réunies dans un meme répertoire dont la structure est strictement définie
- La desciption du déploiment est contenue dans un fichier XML de nom web.xml.

Application web JEE

- L'ensemble des fichiers peut être réuni dans un fichier d'archive (.war) sous forme compressée.
- L'ensemble de ces informations est indépendant du serveur et simplifie le portage et le déploiement d'une application web.
- Structure générale de déploiement d'une application web :
  - Ressources publiques : HTML, CSS, etc.
  - Ressources privées : WEB-INF
  - Classes : fichiers compilés (Servlets, ...)
  - web.xml (fichier de déploiement)
  - lib : fichiers JAR

Application web JEE (1/2)

- Le fichier web.xml est appelé un fichier de déploiement.
- Il contient notamment des informations établissant une ou plusieurs correspondances entre un répertoire spécifié par le client et le véritable répertoire concerné sur le serveur.

Application web JEE (2/2)

web.xml

```
<?xml version= 1.0  encoding= UTF-8 ?>
<web-app version= 3.0  xmlns=
http://java.sun.com/xml/ns/javaee
xmlns:xsi= http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation= http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd >
 <servlet>
  <servlet-name>nomServlet</servlet-name>
  <servlet-class>cheminServlet</servlet-class>
 </servlet>
 <servlet-mapping>
  <servlet-name>nomServlet</servlet-name>
  <url-pattern>/urlServlet</url-pattern>
 </servlet-mapping>
</web-app>
```

Qu'est ce qu'une Servlet ?

- Une **Servlet** est un composant Web du coté serveur Web :
  - Qui permet d'étendre les possibilités d'un serveur Web.
    - Possibilité de générer du contenu **dynamique** en réponse à des requêtes clients.
  - C'est une **classe Java** exécutée sur un serveur multi-threadé (la servlet est partagée par plusieurs threads)
    - Elle est compilée sous forme de byte-code,
    - Elle est exécutée par une machine virtuelle Java (JVM) du conteneur Web.
  - Elle est mise en œuvre (créée) est gérée(cycle de vie) par un conteneur Web (Exemples : Tomcat, Caucho Resin, Enhydra).

Servlet

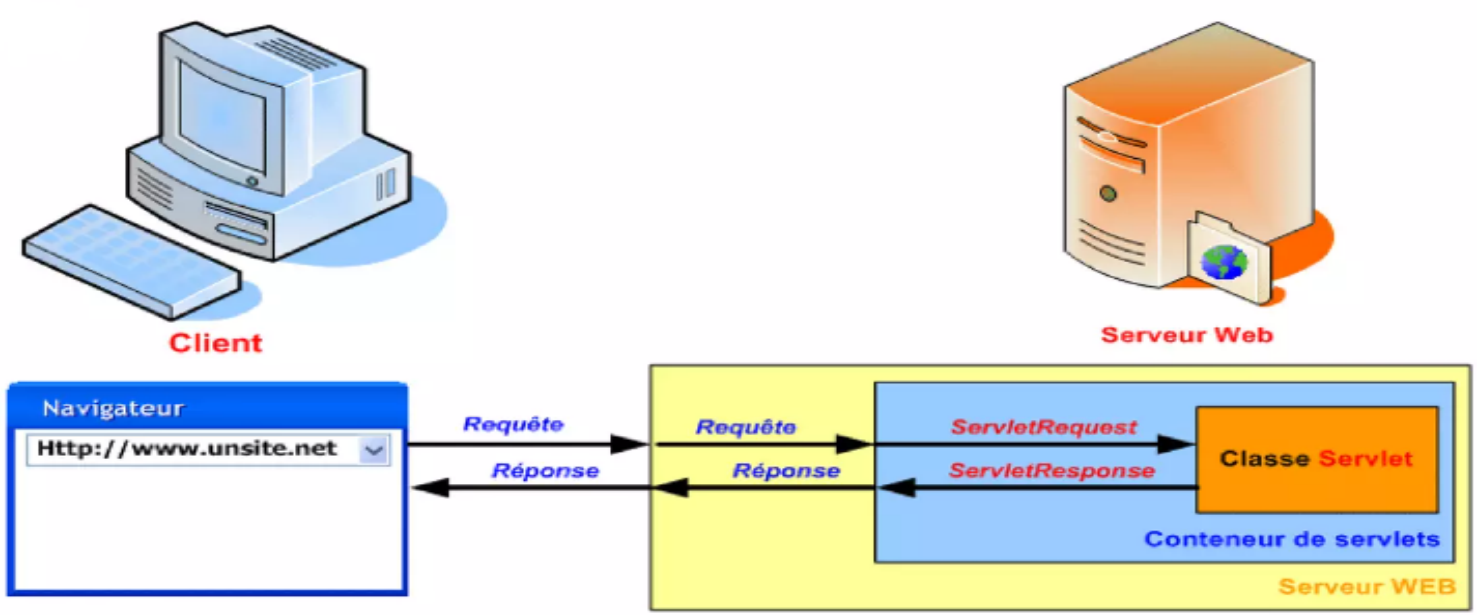
Les servlets utilisent l'API Java Servlet (package javax.servlet).
Grâce aux servlets, vous pouvez :

- Collecter les entrées des utilisateurs via des formulaires de pages web,
- Présenter des enregistrements à partir d'une base de données ou d'une autre source,
- Et créer dynamiquement des pages web.

Conteneur de servlets

Pour exécuter une application basée sur des servlets, il faut disposer d'un conteneur de servlets. Son rôle est de :

- Gérer les requêtes clients,
- Transmettre les requêtes à une servlet,
- Envoyer les réponses au client.



Servlet avec vue ?

Le modèle MVC nous conseille de placer tout ce qui touche à l'affichage final (texte, mise en forme, etc.) dans une couche à part : la vue. La technologie utilisée pour réaliser une vue est la page JSP.

Nature d'une JSP ?

Quoi ?

- Les pages JSP sont une des technologies de la plate-forme Java EE les plus puissantes, simples à utiliser et à mettre en place
- Elles se présentent sous la forme d'un simple fichier au format texte, contenant des balises respectant une syntaxe à part entière.
- Le langage JSP combine à la fois les technologies HTML, XML, servlet et JavaBeans, pour le moment retenez simplement que c'est un objet Java) en une seule solution permettant aux développeurs de créer des vues dynamiques

Nature d'une JSP ?

Pourquoi ?

- La technologie servlet est trop difficile d'accès et ne convient pas à la génération du code de présentation. Il est nécessaire de disposer d'une technologie qui joue le rôle de simplification de l'API servlet : les pages JSP sont en quelque sorte une abstraction "haut-niveau" de la technologie servlet.
- Le modèle MVC recommande une séparation nette entre le code de contrôle et la présentation. Il est théoriquement envisageable d'utiliser certaines servlets pour effectuer le contrôle, et d'autres pour effectuer l'affichage, mais nous rejoignons alors le point précédent : la servlet n'est pas adaptée à la prise en charge de l'affichage.

Nature d'une JSP ?

Pourquoi ?.....suite

- Le modèle MVC recommande une séparation nette entre le code métier et la présentation : dans le modèle on doit trouver le code Java responsable de la génération des éléments dynamiques, et dans la vue on doit simplement trouver l'interface utilisateur ! Ceci afin notamment de permettre aux développeurs et designers de travailler facilement sur la vue, sans avoir à y faire intervenir directement du code Java.

C'est quoi ?

- Le langage JSP est un langage de scripts composé à la fois :
  - de balises HTML
  - et d'instructions provenant du langage de programmation Java.
- Pour que le serveur puisse différencier le code HTML du code JSP, il est nécessaire d'entourer les instructions JSP par des éléments de script spécifiques, appelés étiquettes ou encore balises JSP.
- Ecrire un script JSP demande donc d'acquérir une bonne connaissance du code HTML et d'apprendre à programmer en Java.

Cycle de vie d'une JSP

En théorie :

Quand une JSP est demandée pour la première fois, ou quand l'application web démarre, le conteneur de servlets va vérifier, traduire puis compiler la page JSP en une classe héritant de HttpServlet, et l'utiliser durant l'existence de l'application.

Cela signifie-t-il qu'une JSP est littéralement transformée en servlet par le serveur ?

C'est exactement ce qui se passe. Lors de la demande d'une JSP, le moteur de servlets va exécuter la classe JSP auparavant traduite et compilée et envoyer la sortie générée (typiquement, une page HTML/CSS/JS) depuis le serveur vers le client à travers le réseau, sortie qui sera alors affichée dans son navigateur !

Cycle de vie d'une JSP

Pourquoi ?
La technologie JSP consiste en une véritable abstraction de la technologie servlet : cela signifie concrètement que les JSP permettent au développeur de faire du Java sans avoir à écrire de code Java ! Bien que cela paraisse magique, rassurez-vous il n'y a pas de miracles : vous pouvez voir le code JSP écrit par le développeur comme une succession de raccourcis en tous genres, qui dans les coulisses appellent en réalité des portions de code Java toutes prêtes !

Cycle de vie d'une JSP

Que se passe-t-il si le contenu d'une page JSP est modifié ? Que devient la servlet auto-générée correspondante ?

Le conteneur vérifie si la JSP a déjà été traduite et compilée en une servlet :

- Si non, il vérifie la syntaxe de la page, la traduit en une servlet (du code Java) et la compile en une classe exécutable prête à l'emploi.
- si oui, il vérifie que l'âge de la JSP et de la servlet sont identiques : si non, cela signifie que la JSP est plus récente que la servlet et donc qu'il y a eu modification, le conteneur effectue alors à nouveau les tâches de vérification, traduction et compilation.

Il charge ensuite la classe générée, en crée une instance et l'exécute pour traiter la requête.

Cycle de vie d'une JSP

Il faut retenir que le processus initial de vérification/ traduction/ compilation n'est pas effectué à chaque appel ! La servlet générée et compilée étant sauvegardée, les appels suivants à la JSP sont beaucoup plus rapides : le conteneur se contente d'exécuter directement l'instance de la servlet stockée en mémoire.