Architecture logicielle « Fondamentaux »

Mohamed DERKAOUI

(last update : 2023-11-03)

Votre formateur

- Ingénieur en informatique (Développement logiciel)
- Consultant Formateur chez Dawan depuis 2007
- Entreprise Architect :
  - TOGAF 9 Certified (update 01/2023)
  - ArchiMate 3 Practitioner (update 04/2023)

The Open Group Open Certified

TOGAF® 9

This is to certify that

Mohamed DERKAOUI

has successfully met the requirements of the program:

The Open Group® Certification for People:
TOGAF® 9 Certified

Date certified: 16 January 2023
Certification Number: 166939

Steve Nunn, President and CEO, The Open Group

The Open O logo, Open O and Check Certification logo, The Open Group, and TOGAF are registered trademarks of The Open Group. The Certification Logo may only be used on or in connection with individuals that have been certified under this program and only with the Certification Level achieved by the individual. The certification register may be viewed at: http://togaf-cert.opengroup.org/certified-individuals.

The Open Group Open Certified

ArchiMate® 3 Practitioner

This is to certify that

Mohamed DERKAOUI

has successfully met the requirements of the ArchiMate Certification for People program at the ArchiMate 3 Practitioner level.

Date certified: 15 April 2023
Certification Number: 9878

Steve Nunn, President and CEO, The Open Group

The Open Group Certification Mark is a trademark and The Open Group and ArchiMate are registered trademarks of The Open Group. The Certification Mark Logo may only be used in connection with individuals that have been certified under this program and only with the Certification Level achieved by the individual. The certification register may be viewed at https://archimate-cert.opengroup.org/certified-individuals

Objectifs

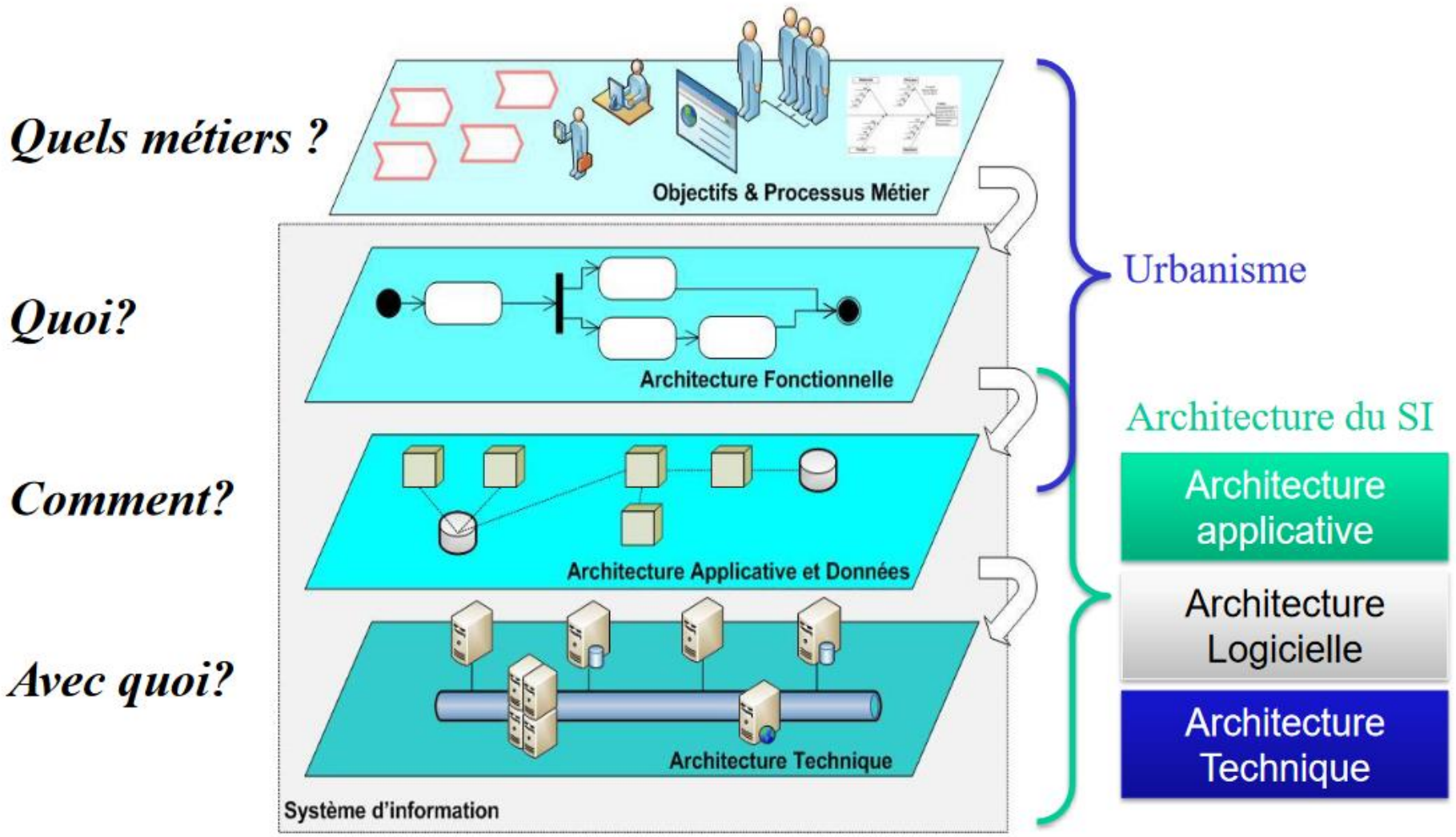
- Comprendre les architectures logicielles
- Découvrir les composants techniques
- Être capable de choisir une architecture répondant à vos critères

Plan

- Comprendre l'architecture logicielle
- Styles d'architecture
- Attributs de qualité
- Technologies

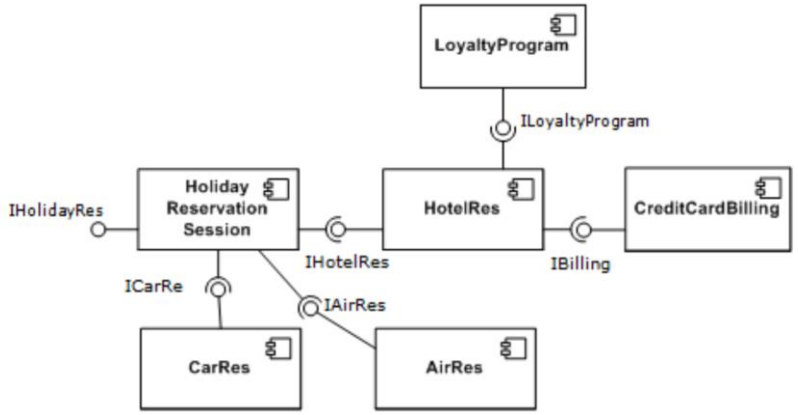
Qu'est-ce que l'architecture logicielle ?

Démarche d'architecture du SI
(structuration en vues)



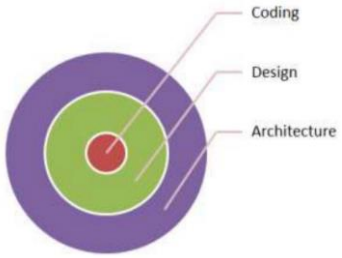
Architecture logicielle

- **Description symbolique et schématique des différents composants d'un ou plusieurs systèmes informatiques, leur interrelations et leurs relations.**
- Les composants sont des spécifications d'unités fonctionnelles, développées ou acquises.
- L'architecture logicielle implique plusieurs choix dont les **technologies**, les **produits** et les **serveurs** à utiliser.
- Exemples de composants : application client léger, micro-service, MoM, ESB,...



Les mythes

- Architecture = Design
- Architecture = infrastructure
- Architecture = technologie fétiche de l'architecte
- Architecture = œuvre d'un seul architecte
- Architecture = 1 schéma
- Architecture système avant architecture logicielle
- Pas de mesures ou de validations dans l'architecture



Quel est l'intérêt de définir une architecture ?

Architecture logicielle

- Partager le même niveau de connaissance
- Faciliter la compréhension du système
- Définir les points d'extensions possibles
- Prendre en compte l'état de l'art
- Évaluer la ré-utilisabilité de composants existants ou de ceux qui seront développés

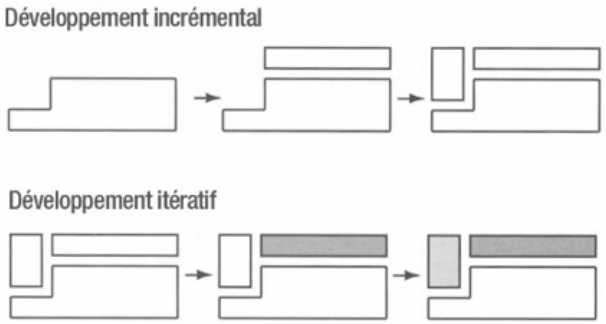
Quand la définir ?

Comment gérer le projet ?

Il existe 2 types d'approches :

- **Approche prédictive :** prévoir des phases séquentielles avec un engagement sur un planning précis de réalisation du projet.

- **Approche agile :** construire un processus itératif et incrémental qui consiste à découper le projet en plusieurs étapes qu'on appelle « itérations ».



Quels sont les inconvénients de l'approche prédictive ?

- **Périmètre fixé au début et de manière exhaustive**

=> pas applicable à tous les contextes

- **Délai et budget estimés en fonction du périmètre**

=> mauvaises estimations et non maîtrise, avenants au contrat

- **Approche séquentielle**

=> pas de retour arrière possible ou difficile

- **Approche prédictive**

=> changements difficiles à prendre en compte

- **« Effet tunnel »**

=> difficulté pour le client de connaître l'avancée du projet, pas de transparence

- **Produit livré à la fin du cycle**

=> valeur métier délivrée à la fin

Définition de l'architecture

- **Méthodes prédictives :**

1 seul cycle en V

Définition de l'architecture à l'étape "Conception globale"

- **Méthodes agiles :**

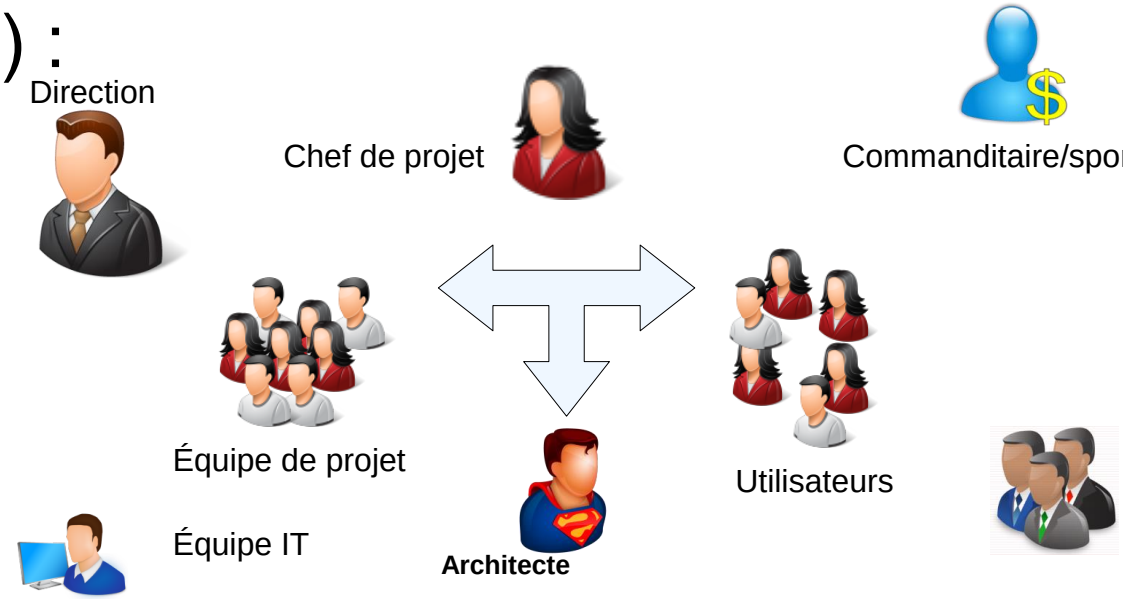
1 cycle en V par sprint (itération)

Définition de l'architecture au Sprint 0 puis au début de chaque sprint (Sprint Planning) et faire une revue à chaque review/retrospective.

Quel est le rôle de l'architecte ?

L'architecture au centre de l'organisation

- Maîtrise d'ouvrage (MOA) :
- Force de proposition
- Décideur
- Facilitateur
- Assure l'adhésion



- Maîtrise d'œuvre (MOE) :
- Référent technique
- Garant de la bonne mise en œuvre de l'architecture
- Peut faire partie de l'équipe projet

Risques

Architecte non reconnu par les acteurs du projet

- Syndrome de la Tour d'ivoire :

Isolement, choix imposés plutôt que consensus

- Titre honorifique

- Pas d'expérience de développement :

Choix non pertinents dans le contexte

- Spécialiste de la documentation :

Big Design Up Front : architecture parfaite avant tout développement

Comment représenter l'architecture d'un projet ?

Comment la représenter ?

- Un ensemble de « **vues architecturales** »
- Chaque vue est destinée à un acteur du projet et comportera un ensemble de schémas et de tableaux de données.
- **Formats** :

- Document textuel (favoriser l'utilisation de schémas)
- **UML** : modélisation des données et traitements
- AADL : utilisé pour les systèmes distribués avec de fortes contraintes (sécurité, temps réel etc.)
- SysML : pendant d'UML, description de systèmes

Quel documents produire ?

Quelle documentation ?

- **Objectifs de la documentation :**

- Permettre d'évaluer et de valider les choix lors de revues
- Servir de référence aux équipes de développement
- Permettre au client de pouvoir communiquer sur l'architecture mise en place

- **Artefacts à produire :**

- DAT : Dossier d'architecture technique (SAD)
- (optionnel) : Guide de design des composants

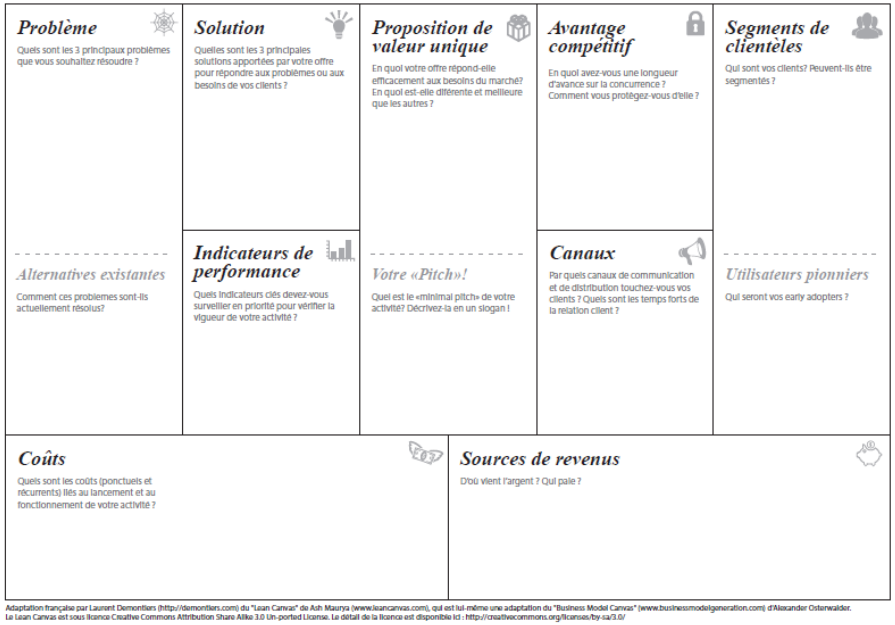
Quel est le contenu du DAT ?

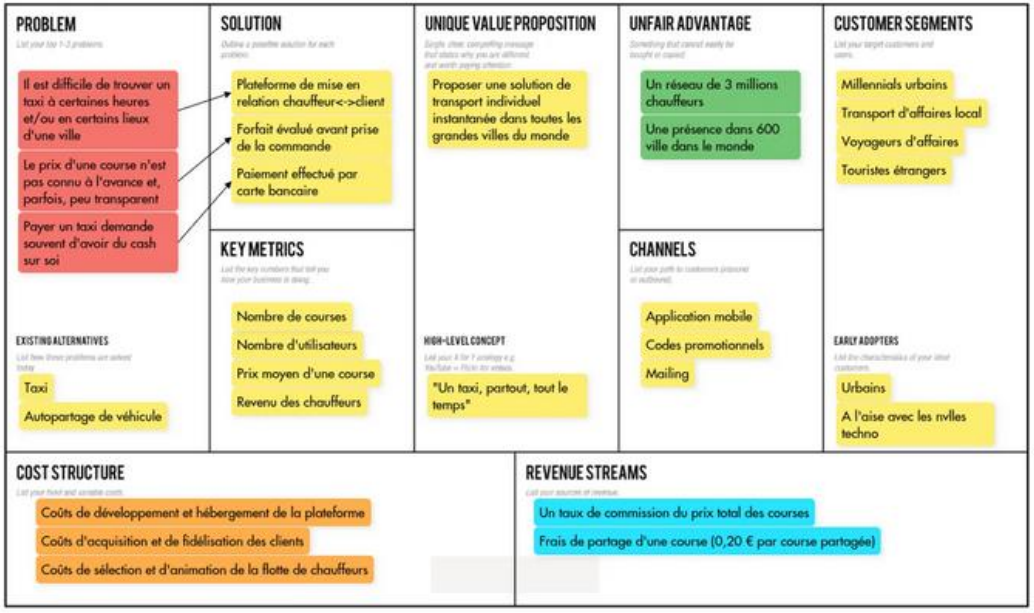
- Vision du produit
- Facteurs déterminants de l'architecture
- Vues architecturales : fonctionnelles, techniques, motivation du choix des styles d'architecture ainsi que le détail des composants et leur déploiement.
- Flux échangés / protocoles utilisés
- Mécanismes de sécurité
- Mécanisme de logs et les traces à conserver
- Stratégie de sauvegarde et de restauration
- Métriques de qualité / stratégies de mesure

Comment représenter la vision ?

Vision du produit

- Construite généralement par le Product Owner / Manager
- Synthétise la problématique et la solution
- Forme non imposée. Recommandation : Lean Canvas





Quels critères prendre en compte pour la conception de l'architecture ?

Facteurs déterminants (ou contraintes)

- Environnement cible
- Typologie de l'application
- Expérience utilisateur
- Nombre d'utilisateurs
- Disponibilité
- Performance
- Volumétrie des données
- Durée de vie
- Sécurité
- Mode d'utilisation
- Interaction avec SI existant
- Interaction avec SI externes
- Compétence des équipes
- Budget/ROI
- Etc, ...

Quelles sont les différentes vues architecturales ?

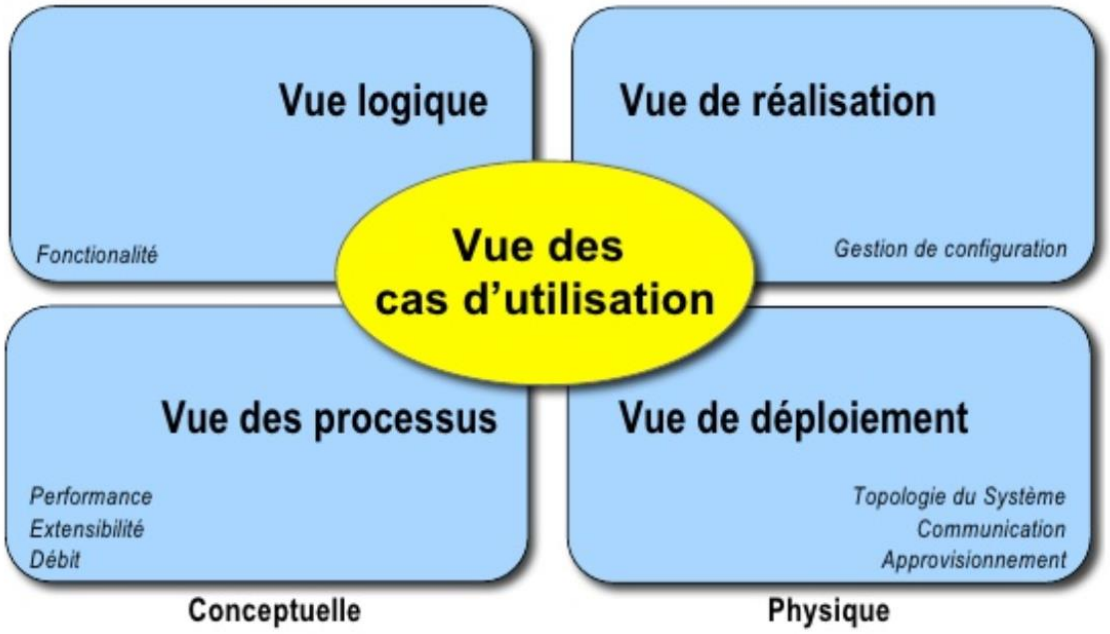
Objectif

- Représenter de manière graphique l'architecture retenue dans toute sa complexité
- Si vue unique (un seul schéma des composants) : peu lisible, ambiguë, incomplète
- On favorise la construction de vues multiples :

1 vue = 1 acteur
Plus de lisibilité
Cohérence unitaire
Cohérence de l'ensemble

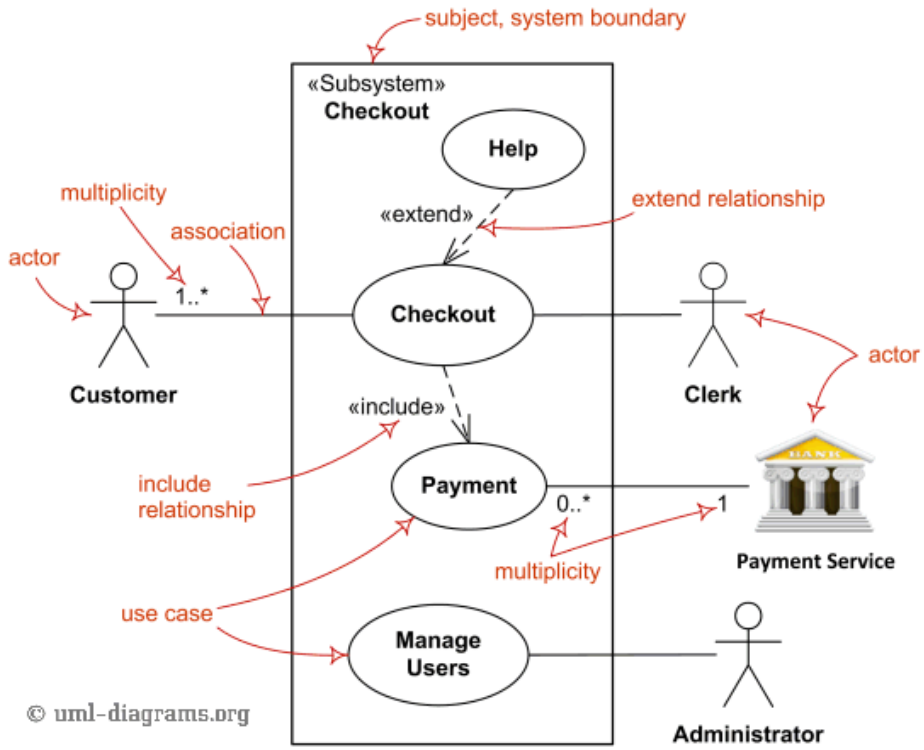
Vues architecturales

- Le **modèle de Kruchten dit modèle des 4 + 1 vues** est celui adopté dans l'Unified Process.
- Le modèle d'analyse, baptisé vue des cas d'utilisation, constitue le lien et motive la création de tous les diagrammes d'architecture.



Vue des cas d'utilisation (scénarios)

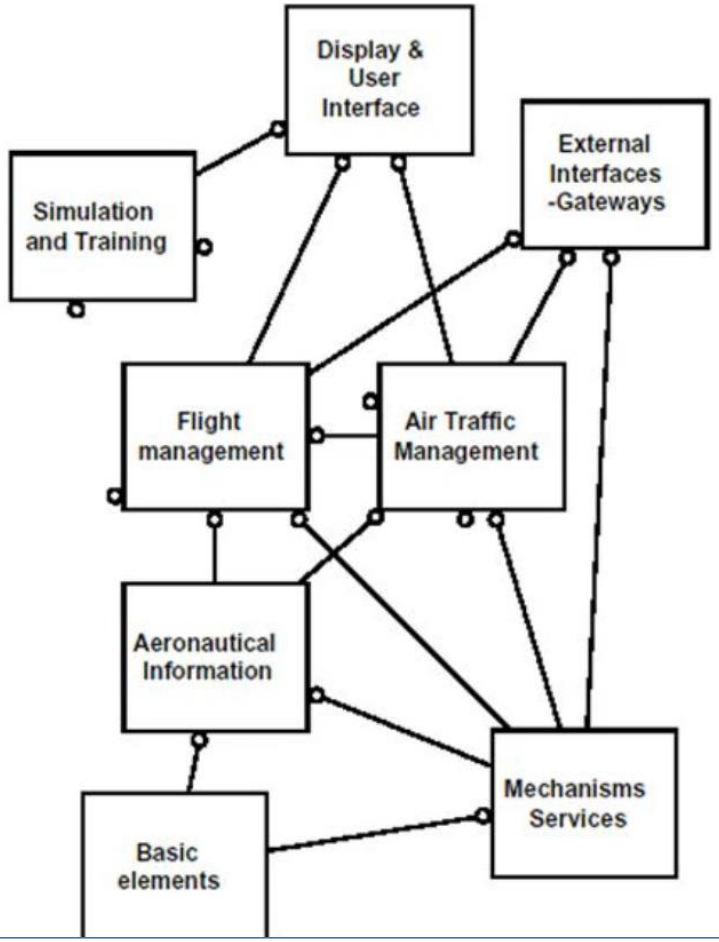
- Un cas d'utilisation est défini représente une séquence d'interaction des utilisateurs (acteurs) avec le système.
- L'intérêt des cas d'utilisation est de **piloter l'analyse par les exigences des utilisateurs**
- Doublon avec les SFD ?
  Mettre un lien si besoin
  ou synthétiser



Vue logique (structure)

- Permet d'identifier les différents éléments et mécanismes du système à réaliser (vue fonctionnelle).

- La vue logique est représentée, principalement, par des diagrammes statiques d'objets enrichis de descriptions dynamiques : structure composite, d'activités, ...



Vue des processus (comportement)

- **Précise les threads et les processus qui forment les mécanismes de concurrence et de synchronisation du système**.
- On met l'accent sur les classes actives qui représentent les threads et les processus.
- Elle montre : La décomposition du système en terme de processus (tâches), les interactions entre les processus (leur communication) et la synchronisation et la communication des activités parallèles (threads).

Vue des processus (comportement)

Le découpage de l'application en threads apparaît lorsqu'on établit :

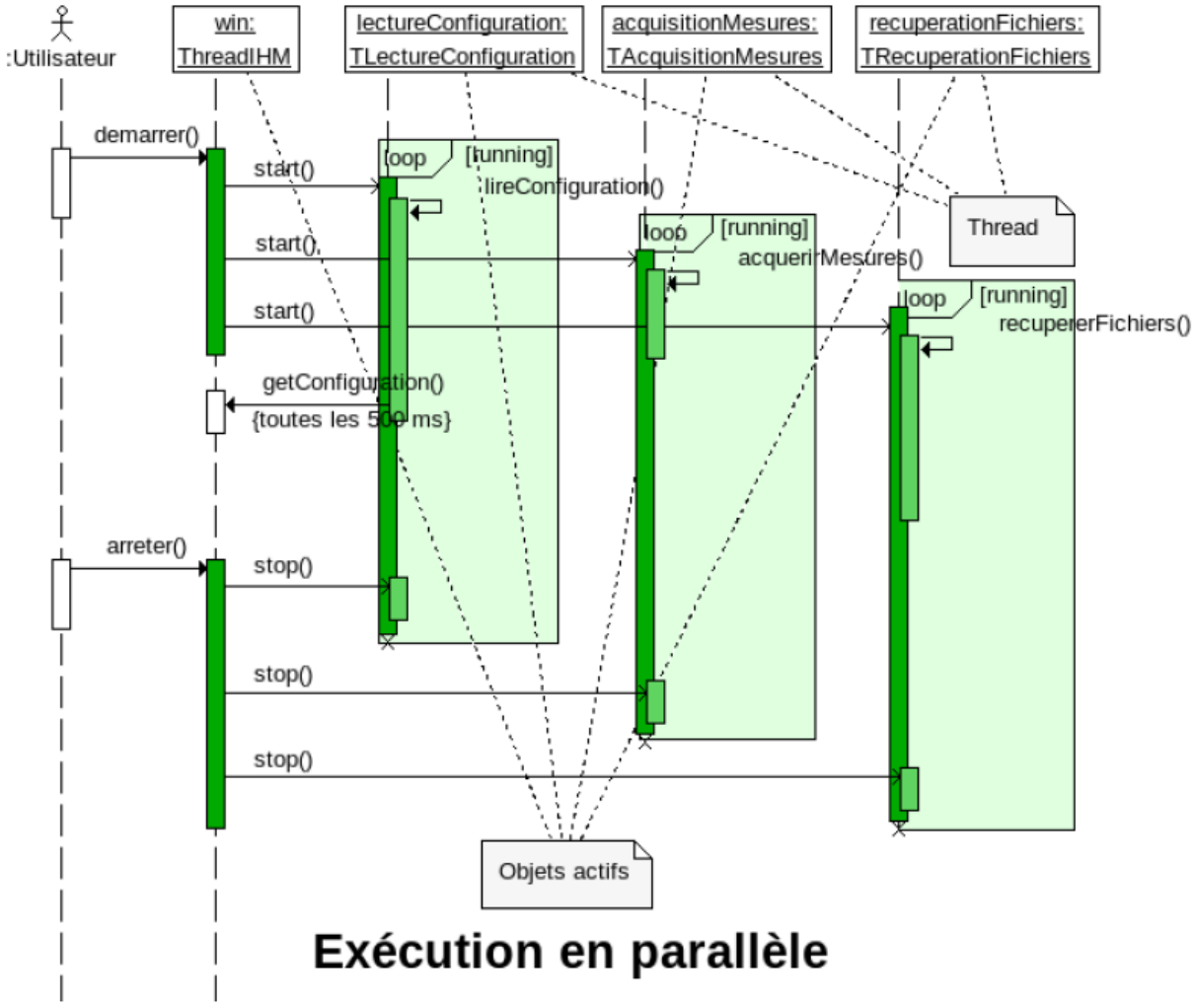
- Les diagrammes d'activités : activités concurrentes et/ou
- Les diagrammes d'états-transitions : états concurrents et/ou
- Les diagrammes de séquences : exécutions concurrentes de plusieurs objets

Ces diagrammes distinguent ce qui se produit de manière séquentielle de ce qui se produit de manière parallèle (donc besoin de processus ou thread).

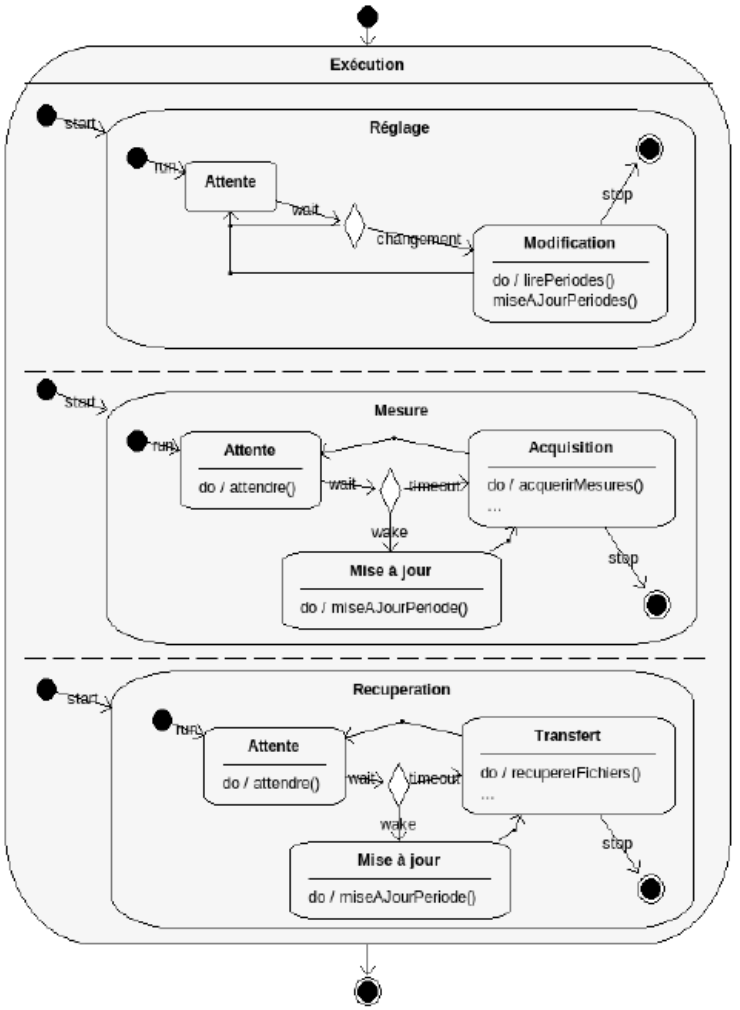
Le besoin est toujours la parallèlisation :

- le système réalise plusieurs activités en même temps et/ou
- un objet prend plusieurs états en même temps et/ou
- une utilisation du système réalise plusieurs exécutions en même temps

Vue des processus (comportement)

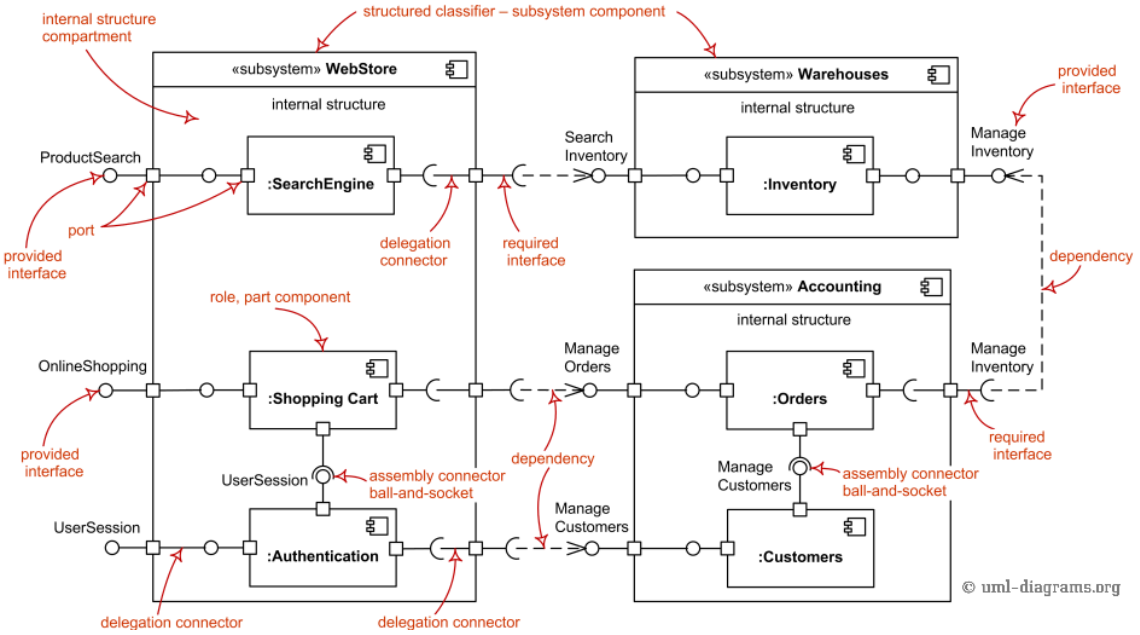


Exécution en parallèle



Vue de réalisation (implémentation)

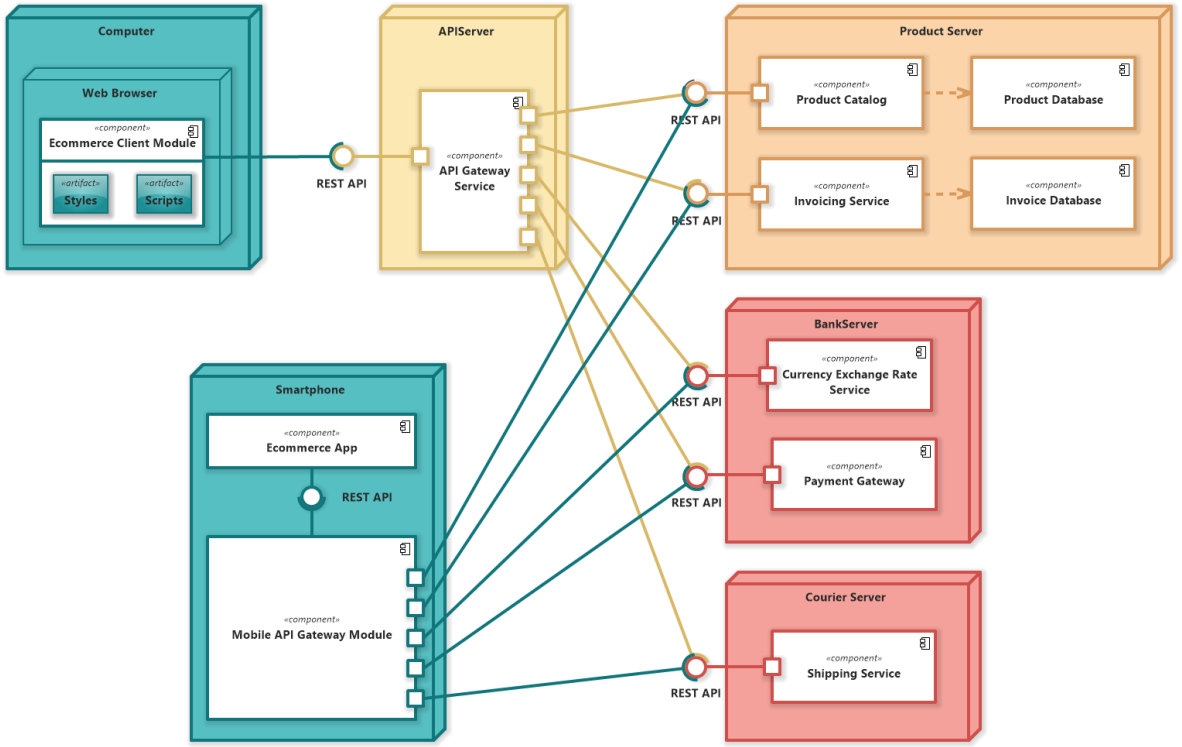
- Permet de visualiser l'organisation des composants dans l'environnement de développement. Cette vue permet également de gérer la configuration (auteurs, versions...).
- Seuls les **diagrammes de composants** sont utilisés dans cette vue.



Vue de déploiement (physique)

- Représente le système dans son environnement d'exécution. Elle traite des contraintes géographiques, des contraintes de bandes passantes, des performances du système ainsi que de la tolérance aux fautes et aux pannes.

- Les diagrammes de cette vue sont les diagrammes de composants et les **diagrammes de déploiement**.



Quelle est la démarche pour développer un modèle architectural ?

Développer un modèle architectural

- Faire une esquisse de l'architecture

Cas d'utilisation => décomposition en sous-systèmes
Déterminer les principaux composants requis

- Sélectionner un ou plusieurs styles d'architecture

- Raffiner l'architecture :

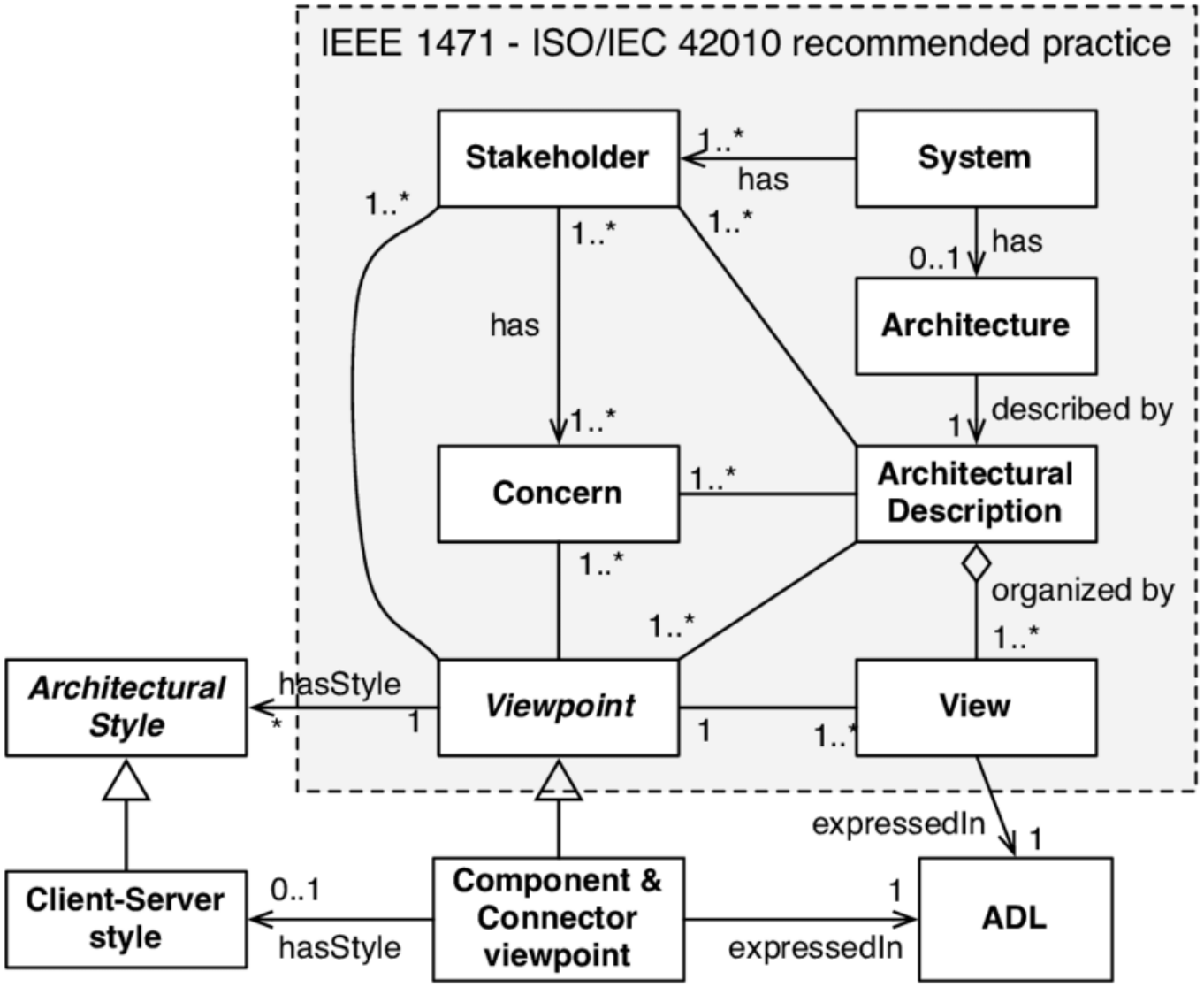
- Identifier les principales interactions entre les composants et les interfaces requises
- Décider comment chaque donnée et chaque fonctionnalité sera distribuée parmi les différents composants
- Déterminer si on peut réutiliser un cadriciel existant (réutilisation) ou si on peut en construire un
- Considérer chacun des cas d'utilisation et ajuster l'architecture pour qu'il soit réalisable
- Détailler l'architecture et la faire évoluer

ISO/IEC/IEEE 42010
Ingénierie des systèmes et logiciels

- La description d'architecture est une norme internationale pour les descriptions d'architecture de systèmes et de logiciels.
- Le standard *ISO 42010* n'impose aucun formalisme pour la représentation des modèles : c'est à l'architecte de sélectionner le format adéquat de ses modèles

http://www.iso-architecture.org/ieee-1471/cm/

ISO/IEC/IEEE 42010



Quels sont les principaux composants à notre disposition ?

Panorama de composants

- Application Client lourd (composant Desktop)
- Application Client léger (composant web)
- Bibliothèque
- Web Service
- Micro-service
- Web Socket
- Middleware Orienté Messages
- Entreprise Services Bus
- API Gateway
- Base de données
- Répartiteur de charges
- Ressource cloud
- etc...

Composant Desktop

- Centralise l'IHM et un potentiel traitement métier
- **Contraintes à prendre en compte :**

- Environnement cible (OS, machine)
- Portabilité
- Déploiement et mises à jour
- Sécurité

- Langages :

- Java (~~Swing~~/**JavaFx**)
- C#/VB.NET (~~Windows Forms~~ / **WPF**)
- C++ (~~MFC~~ / **Qt**)
- Python (~~tk~~, PyQt)
- ...

Composant web

- Application client léger déployé sur un serveur d'applications

- **Contraintes à prendre en compte :**

- Navigateur (compatibilité, fonctionnalités, sécurité)
- Portabilité ?
- Déploiement
- Gestion des sessions
- Maintenance
- Sécurité

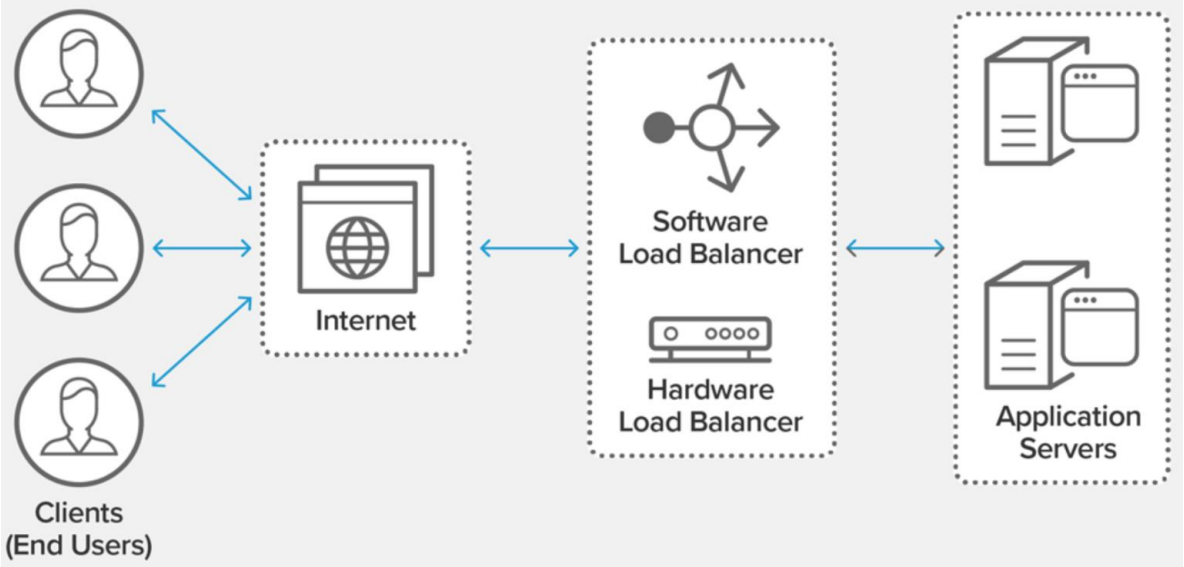
- Technologies :

Java EE, .NET, PHP, Python, JavaScript...

Load Balancer

- Répartiteur de charges sur plusieurs serveurs
- Types : Matériel vs Logiciel
- **Considérations de conception :**

- affinité de session (sticky-session)
- reprise après incident



Bibliothèque

- Ensemble d'entités et fonctions métiers packagées dans une archive (dll/so, jar,…) ou un répertoire

- **Contraintes à prendre en compte :**

- Compatibilité de versions / dépendances
- Chargement (statique/dynamique)
- Interopérabilité entre langages

- Technologies : tout langage permettant de créer une librairie

Application mobile

- Application déployée sur chaque terminal mobile

- **Contraintes à prendre en compte :**

- Les mêmes contraintes que pour un composant desktop
- Fonctionnalités supportées par le device mobile (NFC, Bluetooth,….)
- Type d'usage (connecté/non connecté)
- Mode de développement : natif, web mobile, hybride

- **Technologies** :

- Natif : Java, Kotlin (Android), Swift (iOS),…
- Web Mobile : framework responsive design ou pure CSS
- Hybride :
  Apache Cordova, Xamarin.Forms, MAUI, …
  PWA (Progressive Web Application) : Flutter, Ionic,...

Web Service

- Application web interopérable (sans ihm) déployée sur un serveur web qui expose un ensemble de méthodes
- Types de services : SOAP vs REST
- **Contraintes à prendre en compte :**

- Les mêmes contraintes que pour une application web
- Granularité : micro-services ?
- Mécanisme d'authentification et de gestion des utilisateurs : token JWT, OAuth2,…
- Multiplication des flux (attention au trafic réseau)
- Indépendance sur les données

- Technologies : Java EE, .NET, PHP, Python, Node.js...

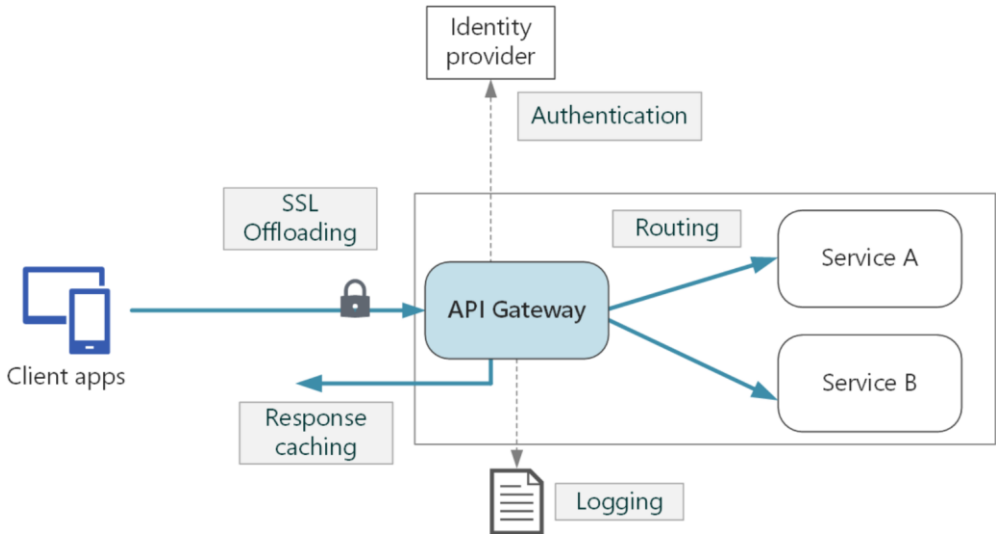
API Gateway

- Passerelle API : point d'entrée unique pour les APIs
- Usage : sécurité, monitoring, routage, caching, limitation d'usage,...
- **Contraintes d'architecture :**

- fonctionnalités offertes
- performances

- Produits :

- Apigee API Management
- Gravitee
- Apache APISIX
- SoftwareAG API Gateway
- Kong
- Nginx (serveur web configuré en passerelle)



Web Socket

- Protocole d'échange **full duplex** qui **maintient une connexion TCP unique entre deux nœuds**

- Usage : messagerie, flux live, système collaboratif, système de notification en temps réel

- **Contraintes à prendre en compte :**

- les mêmes contraintes que pour un service web
- Charge générée par le maintien des connexions

- A sécuriser avec du SSL/TLS :

wss://
(SSL over ws://)

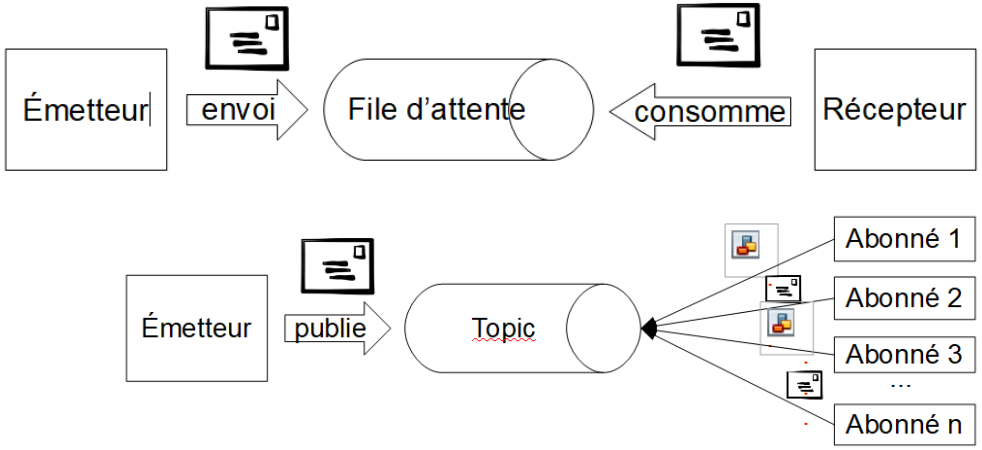
Middleware Orienté Messages

- Système de **communication asynchrone** à base de messages avec l'utilisation d'une **file d'attente**

- Usage : découpler consommateurs et émetteurs

- **Modes** :

Point à point (queue)
Abonnement (topic)



- **Contraintes :**

- Type de file d'attente
- Gestion de la persistance
- Choix du protocole : AMQP, MQTT, HTTP, STOMP, JMS
- Gestion de l'ordre des messages

- Technologies : ActiveMQ, RabbitMQ, Kafka, WebSphereMQ, ZeroMQ (bibliothèque de messagerie)....

Entreprise Service Bus

- Bus de messages constitué :

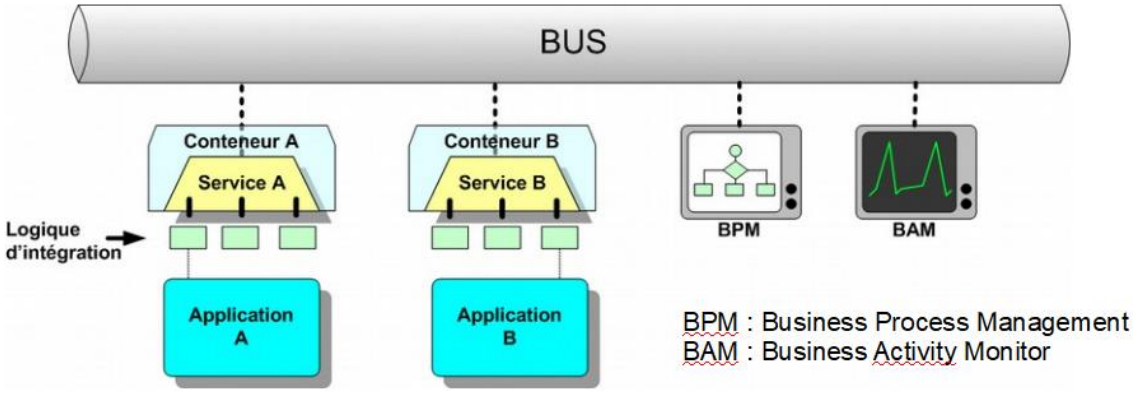
- d'un MoM (middleware orienté messages)
- de services pour router et transformer les messages

- Intérêt : découpler le client du service à appeler et du format exigé par le service.

- **Usage : applications multi-protocoles, fort besoin d'agilité**

- Technologies/Produits :

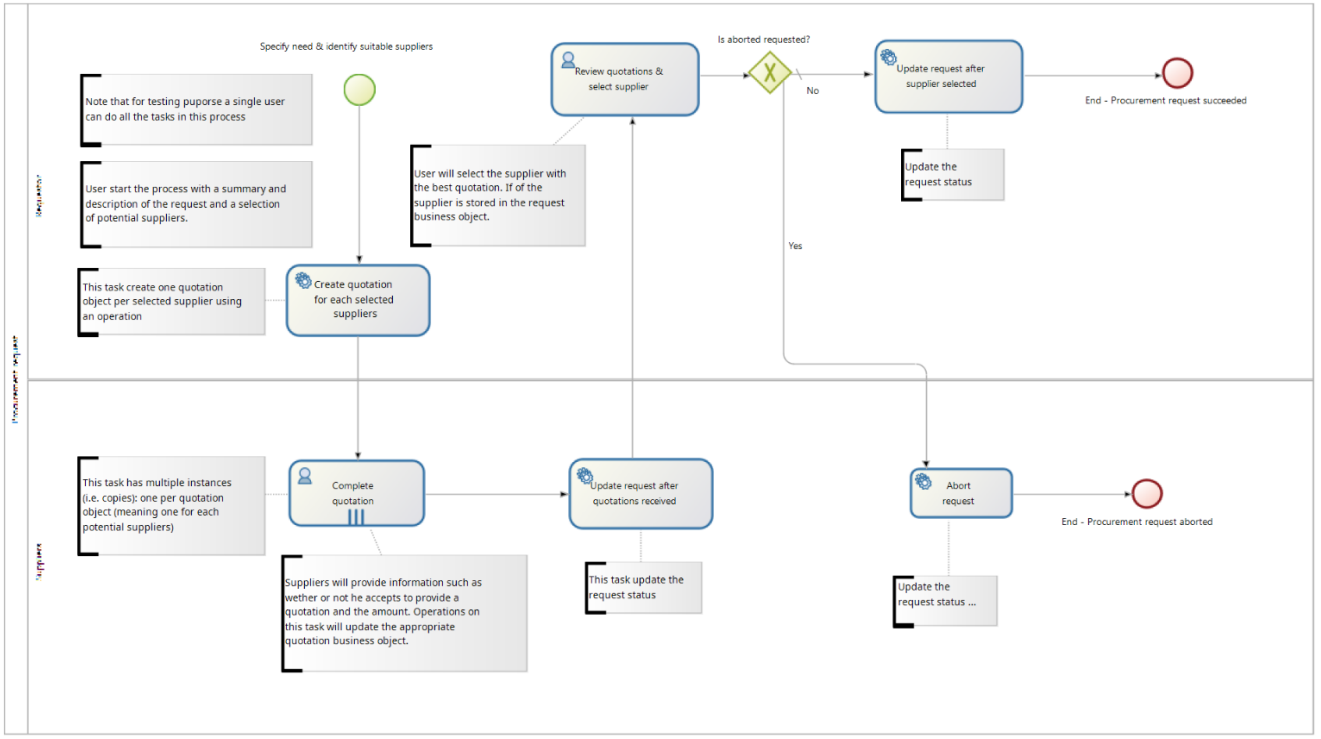
SoftwareAG ESB,
Fuse ESB,
WSO2 ESB



.....

Plateforme d'automatisation de processus métier

- Processus métier modélisé en BPMN et embarqué dans un outil BPM (Business Process Model)
- Exemple : Bonita, Camunda



Composant de stockage

- Serveur de fichiers
- Base de données :

- Relationnelle (SQL)
- NoSQL : 4 types (clé/valeur, colonne, graphe, document)

- NoSQL : structures de données
- SQL vs NoSQL
- **Contraintes à prendre en compte :**

- volume des données
- structure de stockage

Ressource cloud

- Ressource en libre service et adaptable à la demande
Mutualisable ; Paiement à l'usage
- Modèles de service : cloud public vs privé vs communautaire
- **Considérations de conception** :
- typologie de la ressouce,
- packaging de l'application

Autres composants

- Batch : traitement en lots
- Cache distribué
- Borne tactile
- Toute autre ressource matérielle
- etc...

Quels sont les différents styles d'architecture ?

Notion de style

A l'instar de l'architecture traditionnelle, l'architecture logicielle peut se caractériser par des styles.

En fonction du résultat souhaité, un système informatique pourra utiliser plusieurs styles.

Catalogue

6 styles :

- Architecture en appels et retours
- **Architecture en couches**
- ~~Architecture centrée sur les données~~
- Architecture en flot de données
- **Architecture orientée objets**
- Architecture orientée agents

Appels et retours

- Basée sur le raffinement graduel proposé par Niklaus Wirth (**décomposition fonctionnelle**).
- Principe : décomposer la fonctionnalité initiale en sous fonctionnalités jusqu'à obtenir des problèmes simples.
- Avantage : diminution de la complexité.
Inconvénient : diminution de la performance
- Exemples d'architecture :
  - Architectures basées sur les composants distribués
  - Architectures orientées services

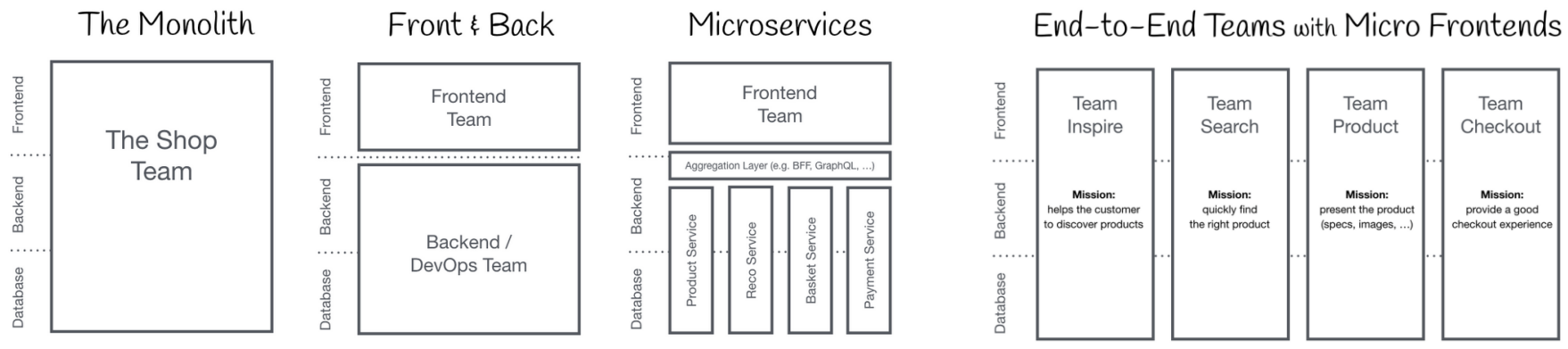
Couches

Le système est décomposée en couches, chaque couche ayant une responsabilité spécifique.

- Ré-utilisabilité des couches inférieures par les couches supérieures
- Décomposition selon 2 axes :

- Technique (persistance, métier, vue, ...)

- Fonctionnel

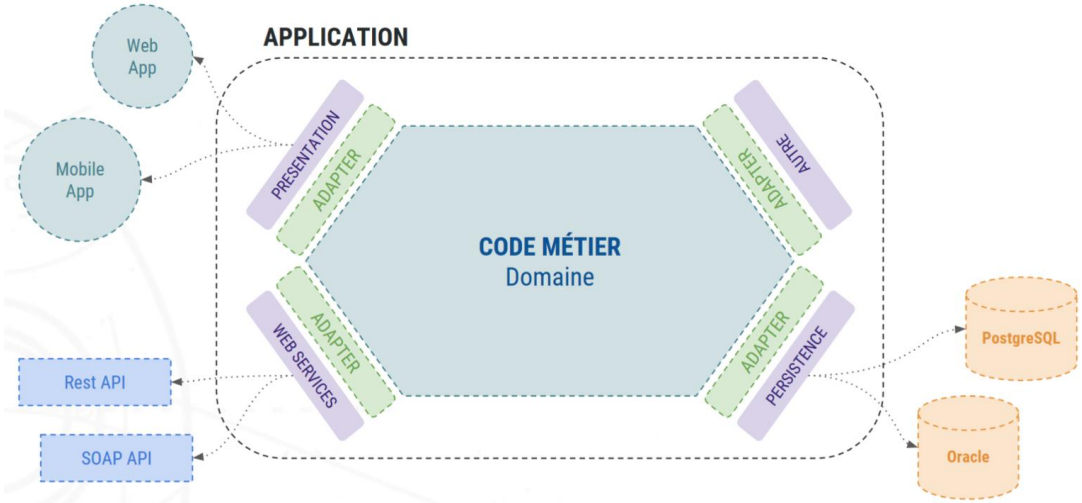


Couches
Architecture hexagonale

- Structurer le code en centralisant le code métier et en développant des couches d'abstraction.

- Nécessité d'écrire des couches d'abstraction pour :

- communiquer avec des Bdds
- rendre différentes vues tel que le rendu web et mobile
- accepter plusieurs types d'accès en exposant des web-services REST, GraphQL ou encore SOAP.



Couches
Architecture Hexagonale (2)

- **Avantages :**

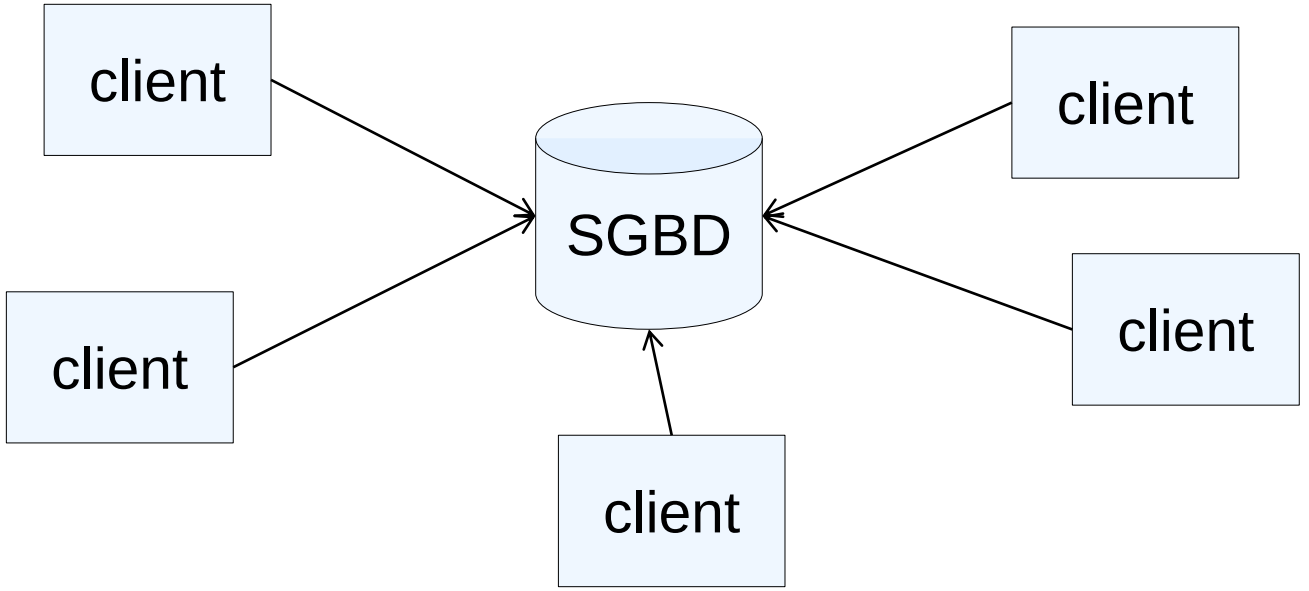
- Permet de garder le code métier complètement agnostique des différentes couches.
- Permet de se concentrer sur le code métier lorsqu'une fonctionnalité doit être développée, sans se soucier du reste
- Il suffit d'ajouter un adapter, en se basant sur les interfaces métiers pour pouvoir intégrer un nouveau service.

- **Inconvénients** :

- D'un point de vue architecture, cette implémentation peut apparaître comme complexe au premier abord.
- YAGNI (You Aren't Gonna Need It) : la mise en place de ce pattern d'architecture nécessite d'avoir le besoin de mettre en place plusieurs couches d'abstraction, à différents niveaux, afin que son choix soit perçu comme bénéfique.

Centrée sur les données

- Un serveur de bases de données est au centre de cette architecture. Il est responsable de la gestion des données.
- Des composants clients accèdent au serveur.



Centrée sur les données (2)

- Le serveur peut être :
  - Passif : il exécute les instructions envoyées par les clients
  - Actif : il notifie les clients en cas de modification des données.

Avantages : séparation entre les données et les traitements → un nouveau client peut facilement être intégré dans l'architecture.

Inconvénients : tous les clients dépendent des données, la structure des données doit être stable → faible évolutivité

Flot de données

Plusieurs composants logiciels/matériels sont reliés entre eux par des flux de données.

L'information circule dans le réseau de composants.

L'information est consommée et transformée par le composant qu'elle traverse.

- Réseau linéaire + transformation → architecture par lot (batch)
- Réseau complexe + transformation + synthèse → architecture de médiation

Orientée objets

Les composants du système sont des objets.

Ils intègrent des données et les opérations de traitement de ces données.

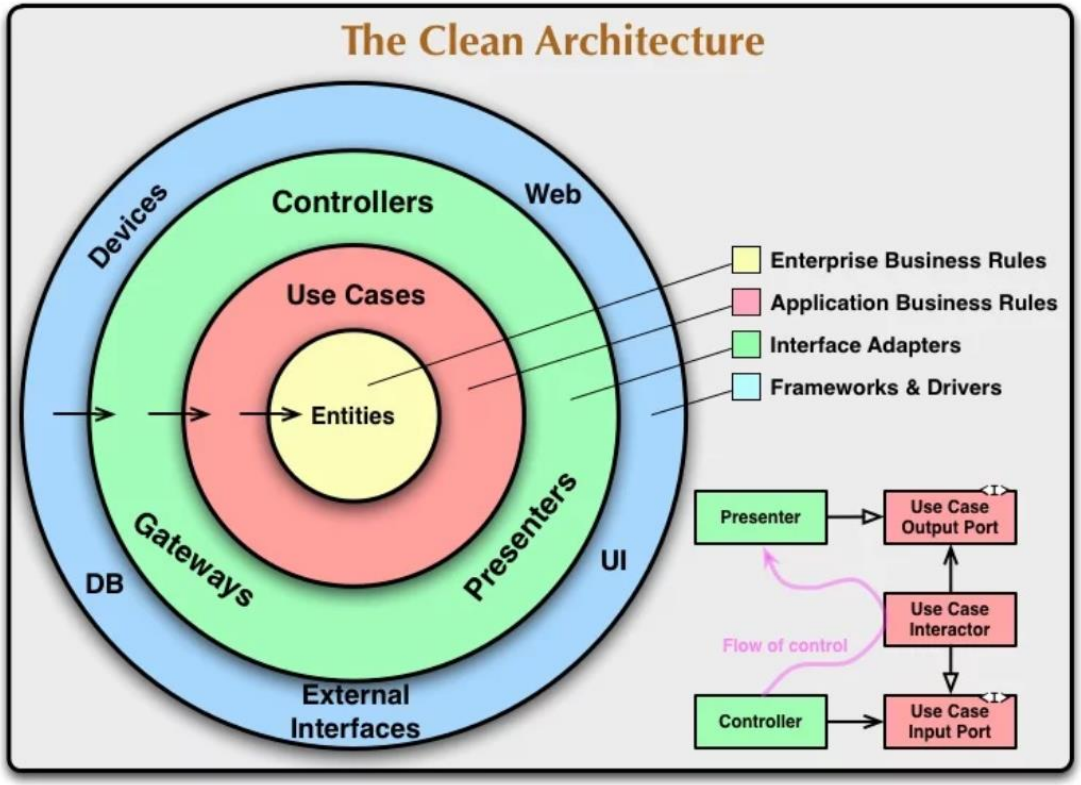
La communication et la coordination entre les objets sont réalisées par un mécanisme de passage de messages.

- Encapsulation
- Héritage
- Polymorphisme

→ peut être vu comme une extension de l'architecture appels et retours

Clean architecture

- La Clean Architecture vise à réduire les dépendances de votre logique métier avec les services que vous consommez (API, Base de données, Framework, Librairies tierces), pour maintenir une application stable au cours de ses évolutions, de ses tests mais également lors de changements ou mises à jour des ressources externes.



Orientée agents

- Un paradigme où l'objet, de composant passif, devient un composant projectif.
- (l'architecture objet n'est qu'une extension de l'architecture en appels et retours, le programme peut être écrit de manière à demeurer déterministe et prédictible).
- L'agent logiciel, utilise de manière relativement autonome, avec une capacité d'exécution propre, les autres agents pour réaliser ses objectifs : il établit des dialogues avec les autres agents, il négocie et échange de l'information, décide à chaque instant avec quels agents communiquer en fonction de ses besoins immédiats et des disponibilités des autres agents.

Comment gérer les traces de l'application ?

Où peut-on stocker les traces de l'application ?

- **Fichier (plus rapide) :**

- format libre
- prévoir une rotation de logs : sur taille et/ou date
- attention aux entrées/sorties => ressources++
- gestion des accès concurrents

- **Bdd relationnelle / NoSQL :**

- structure imposée ou pas
- recherche rapide

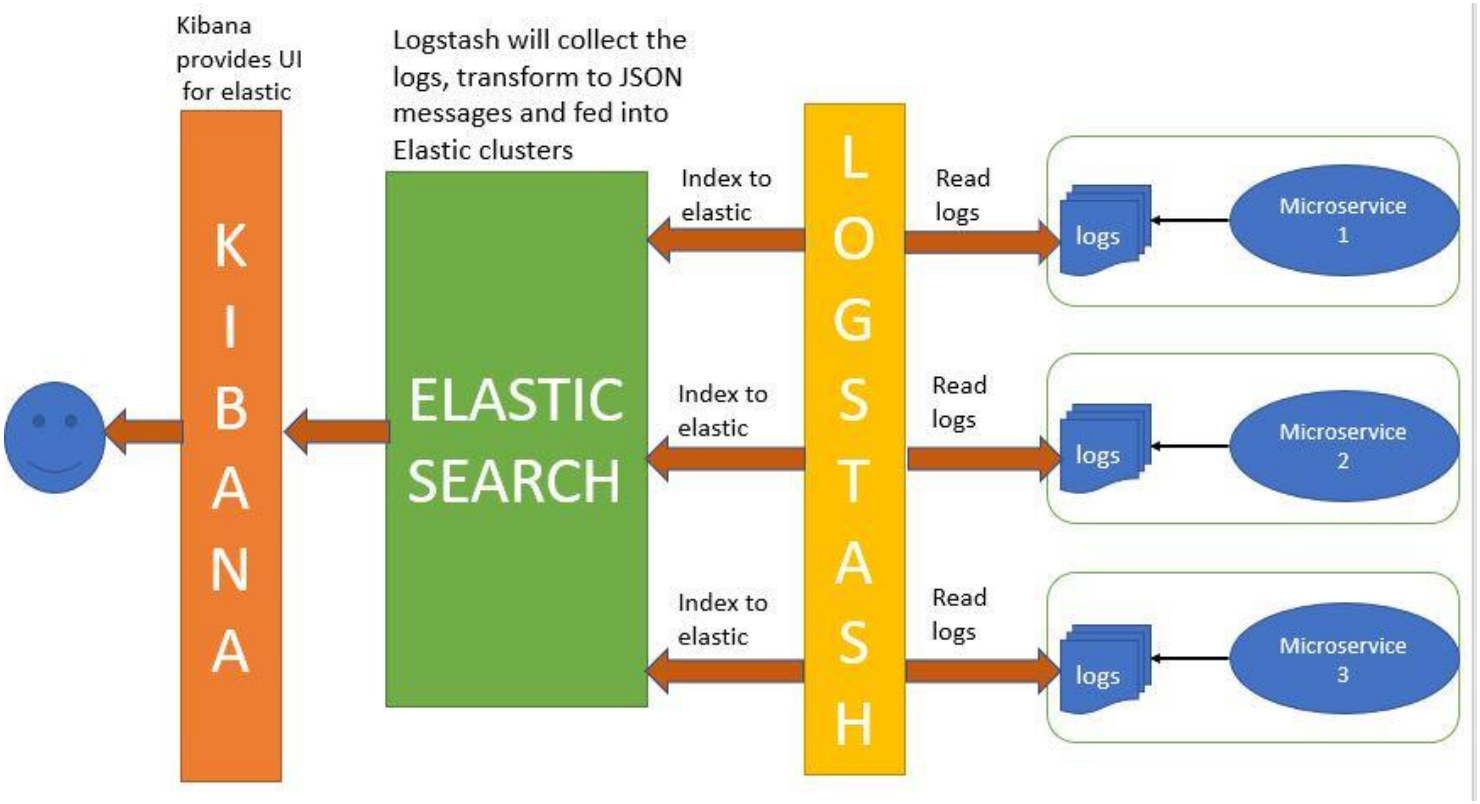
- **Solution externe :** ELK (Elasticsearch, Logstash, Kibana), Graylog, Splunk, …

- accessible au travers de multiples protocoles : HTTPs ou autre
- agrégation de logs depuis plusieurs applications
- indexation automatique des logs
- outils associées

Log dans un fichier puis agrégation via Logstash

Les microservices écrivent dans leur propre support de logs

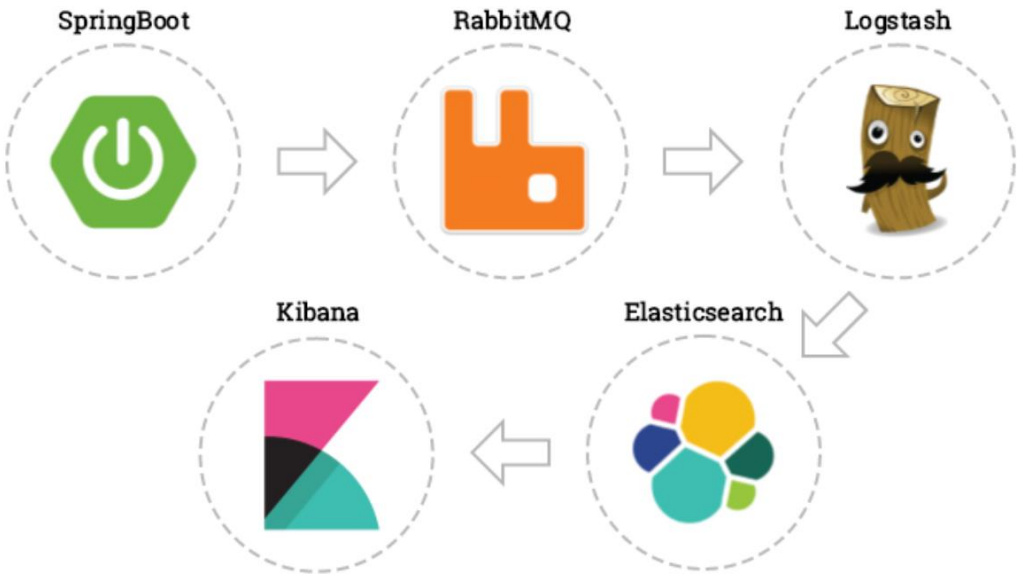
Logstash intervient pour collecter les différents fichiers et injecter la donnée dans Elasticsearch



Log dans un message broker puis consommation

Le log est mis dans un file d'attente du message broker :
- directement via AMQP
- ou passant par un logback appender – AMQP ou HTTP)

Une configuration de logstash permet de réceptionner les messages de la file d'attente et de les injecter dans Elasticsearch



Que doit-on tracer dans une application ?

- Les accès :
  - à l'application cliente
  - aux différentes APIs
- Les erreurs :
  - techniques
  - fonctionnelles : échecs d'authentification, règles de gestion,…
- Les actions des utilisateurs / le téléchargement des assets
- Historisation des modifications réalisées sur les données

Comment formaliser les logs dans le DAT

Trace	Composant(s) responsable(s)	Stratégie / Support / Flux
Accès	WebPortal pour l'accès à l'appli cliente API Gateway pour les appels de services	via un service de logs log de l'adresse ip de l'utilisateur log de trame HTTP complète Fichier textuel libre Rotation sur date/heure, maxi 50MB/fichier Rétention : 15j
Actions de l'utilisateur	Services métiers	Fichier textuel libre Rotation sur date/heure, maxi 50MB/fichier Rétention : 15j
Erreurs	WebPortal API Gateway Services métiers	Ecriture de la stack complète Fichier textuel libre Rotation sur date/heure, maxi 50MB/fichier Rétention : 15j
Historisation des modifications réalisées sur les données	Services métiers	Log des modifications sur les champs Table d'audit dans la bdd (Auteur, DateHeure, Entité, champs AncienneVal, NouvValeur)

Quels sont les attributs de qualité d'une architecture ?

Attentes

Qu'attend l'utilisateur ?

- Robustesse
- Performance
- Disponibilité
- Utilisabilité
- Sécurité

Qu'attend le développeur ?

- Modifiabilité
- Testabilité
- Réutilisabilité

Mesure de la qualité logicielle

- **Objectif :**

Garantir un niveau de qualité grâce à des mesures

- **Attributs de qualité :**

Norme ISO 25010

https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

- **Démarche :**

- choisir les attributs les plus prioritaires
- définir la stratégie et/ou le moyen de mesure
- définir le niveau à atteindre

Outils

- Outils de monitoring externes (systèmes)
- Prévoir des mécanismes internes à l'application
- Outils d'analyse de code : abstraction, instabilité,...

Documentation

- **Forme :** Tableau

- la métrique
- la stratégie de mesure
- le seuil (minimal à atteindre ou maximal à tolérer)
- les actions à mener pour la mise en place

- **Exemple :**

- Métrique : Disponibilité
- Stratégie : Ping ou heartbeat
- Seuil : 24/7
- Actions :
  * Monitoring des serveurs : outil xyz
  * Monitoring de l'application : prévoir un port d'écoute ou un log applicatif si heartbeat.

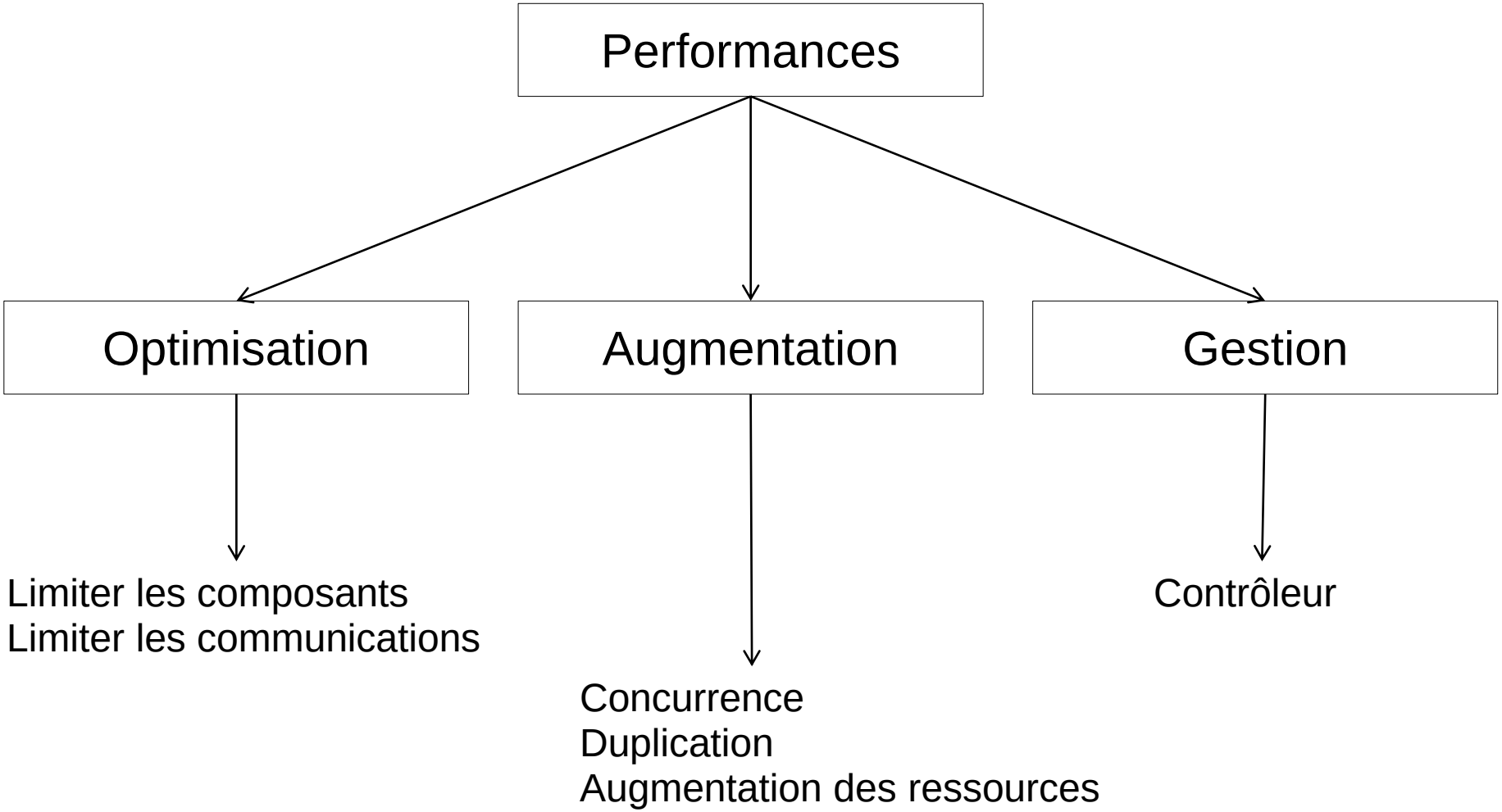
Qu'est-ce qu'une tactique architecturale ?

Notion de tactique

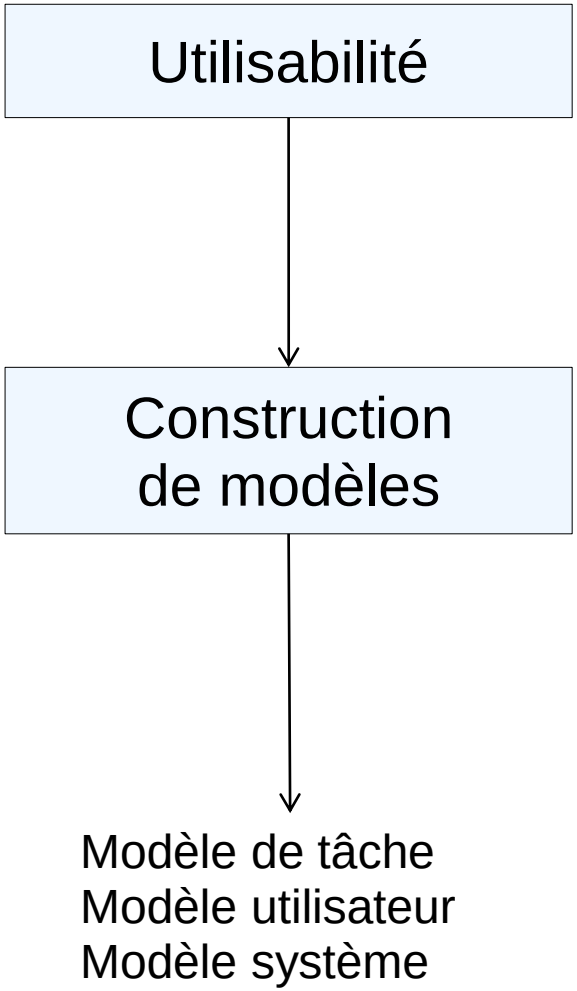
Stratégie locale pour structurer un ensemble de composants

- Suppose une organisation logique et une organisation dynamique
- Mise en place pour répondre à un critère de qualité (ISO 25010)
- Utilisation de scénarii, différents en fonction du style architectural
- Tactiques orientées utilisateurs et développeurs

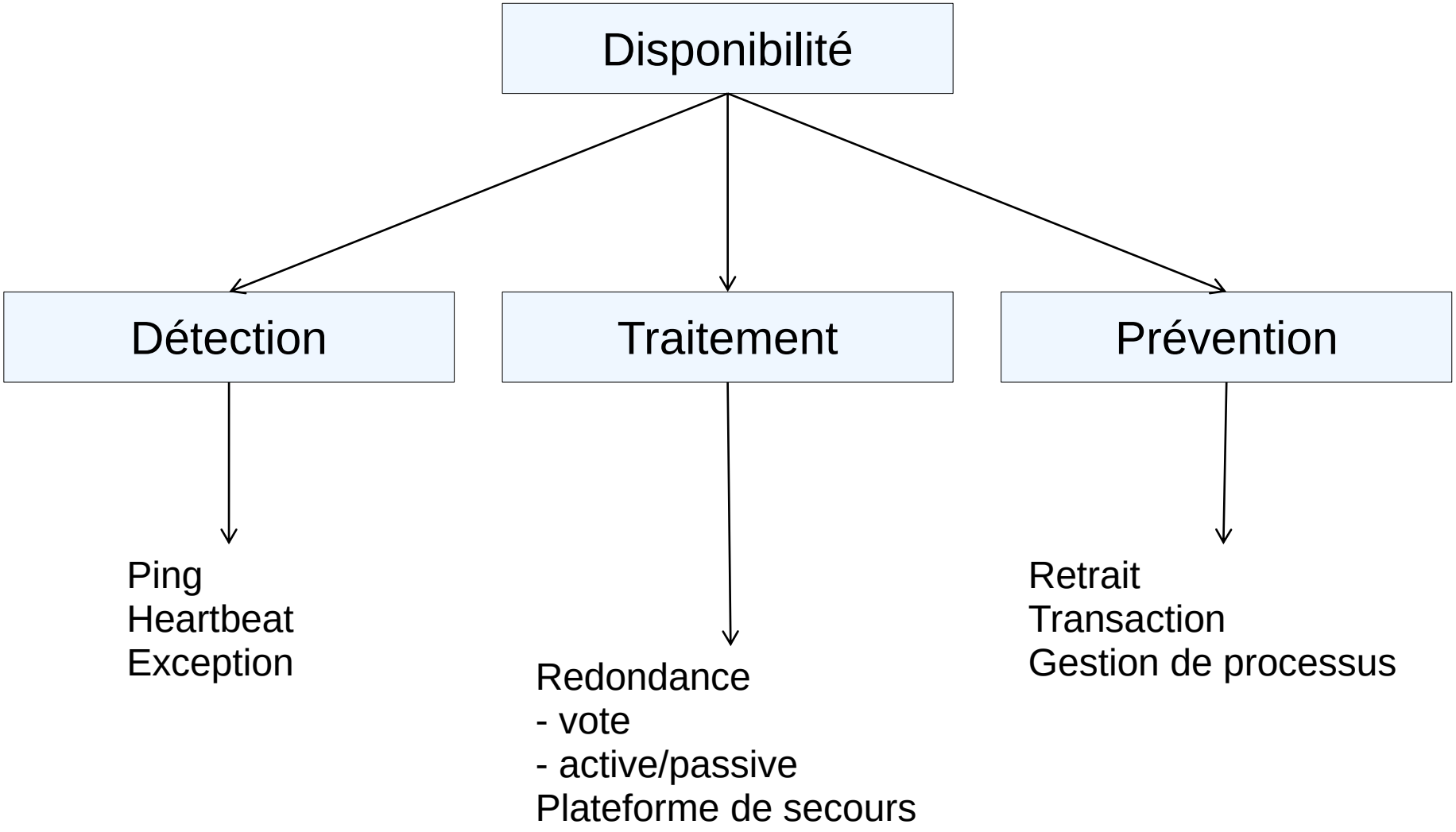
Tactique utilisateur : performances



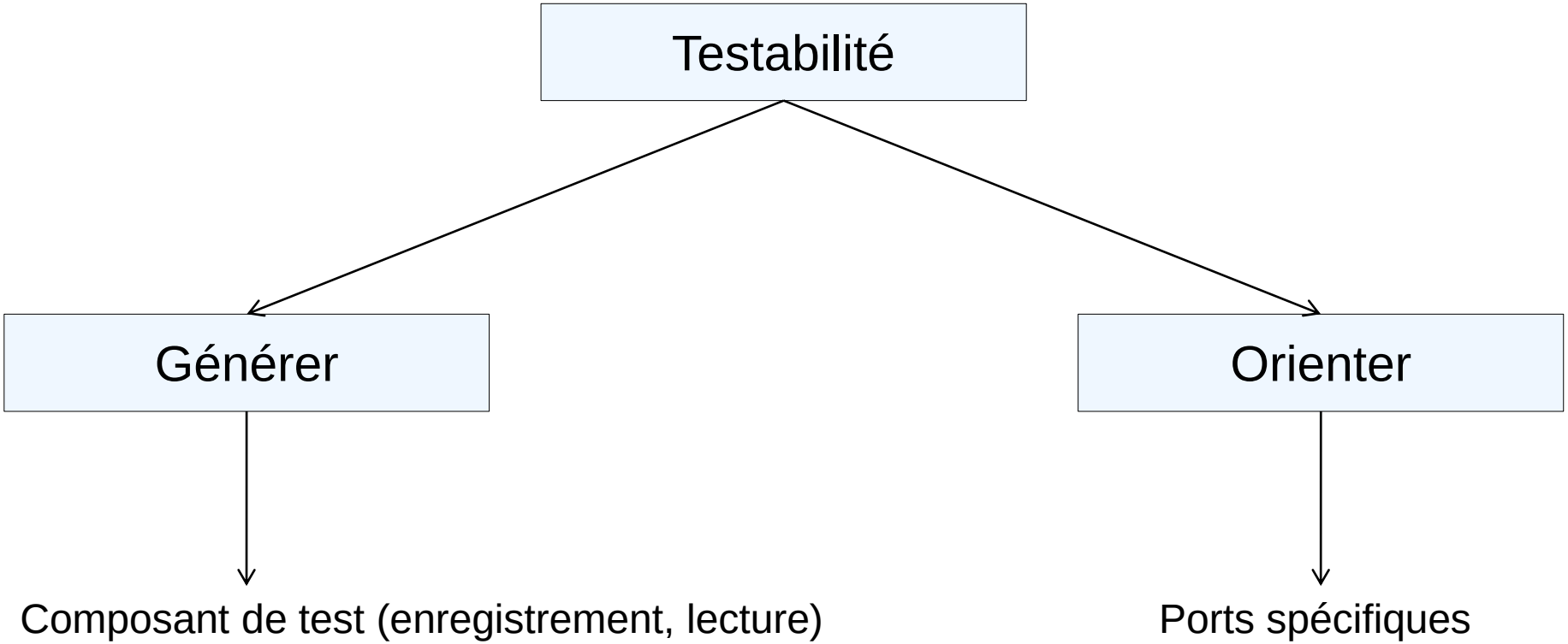
Tactique utilisateur : utilisabilité



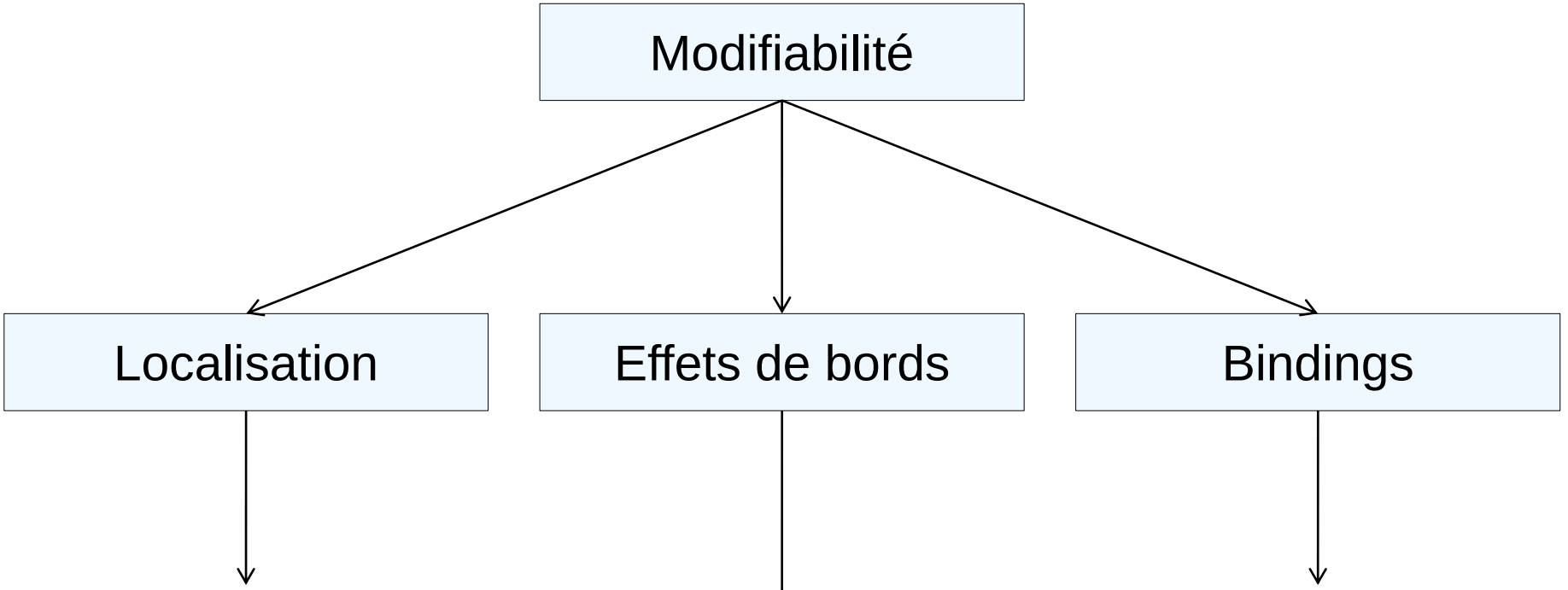
Tactique utilisateur : disponibilité



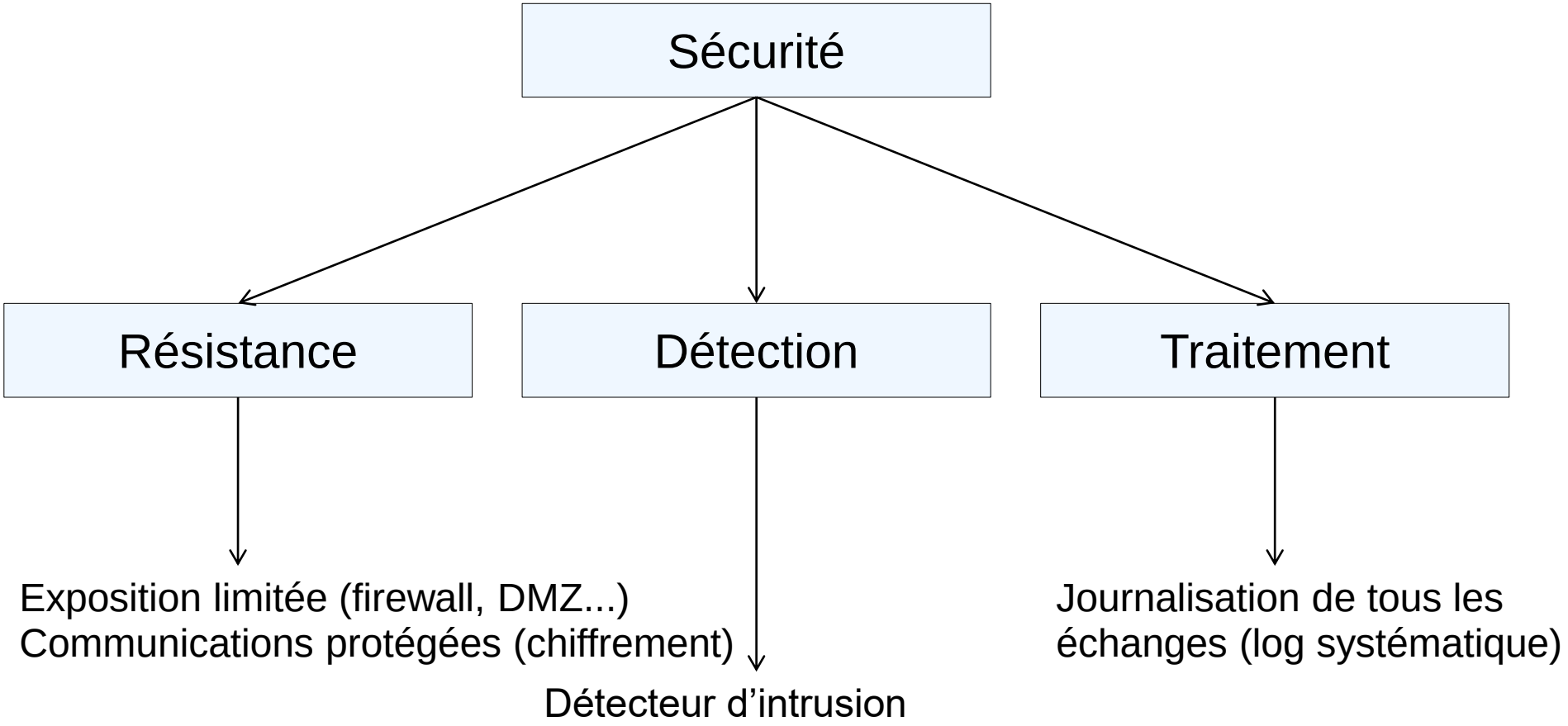
Tactique développeur : testabilité



Tactique utilisateur : modifiabilité



Tactique utilisateur : sécurité



Tactiques architecturales

Exemples non exhaustifs ...

A vous de trouver les vôtres en tenant compte :

- Du contexte
- Des propriétés non fonctionnelles à réaliser
- Des compromis que vous êtes prêts à faire

Questions ?

Plus d'informations sur http://www.dawan.fr

Contactez notre service commercial au
09.72.37.73.73 (prix d'un appel local)