

A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the text 'Groupe 4'.

Groupe 4

ARCHITECTURE LOGICIELLE ET MATERIELLE DU SYSTEME

Chargé du cours : Dr. APEKE K. Séna

Several thin, curved lines in dark blue and light grey originate from the bottom left corner and sweep upwards and to the right.

Les Membres du groupe :

BANAVAÏ Roumanatou,
DIFEZI Tchakoli Chissou,
KATOH Komlavi David,
LOUKA Essossolim.

SOMMAIRE

INTRODUCTION	2
I. DEFINITIONS.....	3
II. LES MODELES D'ARCHITECTURE.....	3
III. L'organisation en couche d'un système informatique	16
IV. Sous-système informatique	20
V. Les paquetages et leurs relations.....	21
VI. Diagramme de composants : organisation du code en modules, dépendances.....	24
VII. Diagramme de déploiement : déploiement physique du système (machines, réseaux, etc.).....	28
CONCLUSION	32
BIBLIOGRAPHIE.....	33

INTRODUCTION

Les systèmes informatiques occupent une place clé dans le monde contemporain pour l'organisation et la mise en œuvre des processus productifs et d'autres natures dans les entreprises. Par exemple, les applications informatiques, pour l'échange d'informations permettent de surmonter les difficultés rencontrées à cause de la distance.

Un système informatique est donc un système automatisé de stockage, de traitement et de récupération de données qui tire parti des outils informatiques pour effectuer une série complexe de processus et d'opérations.

En effet, chaque système informatique est constitué de 03 composantes à savoir : le matériel, les logiciels et l'humain. Nous aurons donc à vous présenter dans ce développement, ce que c'est que l'architecture matérielle et logiciel d'un système informatique et en enfin faire l'illustration à partir de certains diagrammes UML.

I. DEFINITIONS

L'architecture matérielle d'un système informatique est l'ensemble des caractéristiques générales, la conception, le choix et l'organisation des différents dispositifs électroniques des appareils informatiques (ordinateurs, serveurs, consoles de jeux, ...).

L'architecture logicielle décrit les différentes applications, leurs indépendances et les règles associés. Pour un projet web, elle comprend en général : L'infrastructure de base : serveur HTTP, pare-feu, serveur messagerie, moteur de recherche, connecteur BD, BD, SE ...

II. LES MODELES D'ARCHITECTURE

Un modèle d'architecture est un patron décrivant une architecture logicielle permettant de résoudre un problème particulier. Il définit un ensemble de composants et de connecteurs ainsi que leurs règles de configurations. Il existe en particulier six grands modèles à savoir :

1. Architecture n-niveaux

En génie logiciel, l'architecture à plusieurs niveaux est une architecture client-serveur dans laquelle la présentation, le traitement des applications et la gestion des données sont des processus logiquement distincts. Par exemple, une application qui utilise un middleware pour traiter les demandes de données entre un utilisateur et une base de données utilise une architecture à plusieurs niveaux. L'utilisation la plus répandue de l'architecture à plusieurs niveaux fait référence à l'architecture à trois niveaux.

L'architecture client-serveur permet le développement et la modification de différentes interfaces utilisateurs pour la même logique applicative. Cependant il existe également quelques inconvénients majeurs : si tous les clients demandent simultanément des données au serveur, celui-ci peut être surchargé ; si le serveur tombe en panne pour une raison quelconque, aucun utilisateur ne peut utiliser le système.

✚ L'architecture à trois niveaux

C'est une architecture qui est organisée en 3 couches à savoir :

- La couche interface (présentation) : elle est composée d'objets d'interfaces (les boundary objects) pour interagir avec l'utilisateur (fenêtres, formulaires, pages web, etc.)
- La couche logique applicative(traitement) : elle Comporte tous les objets de contrôle et d'entités nécessaire pour faire les traitements, la vérification des règles et les notifications requises par l'application.
- La couche de stockage (accès aux données) : elle réalise le stockage, la récupération et la recherche des objets persistants.

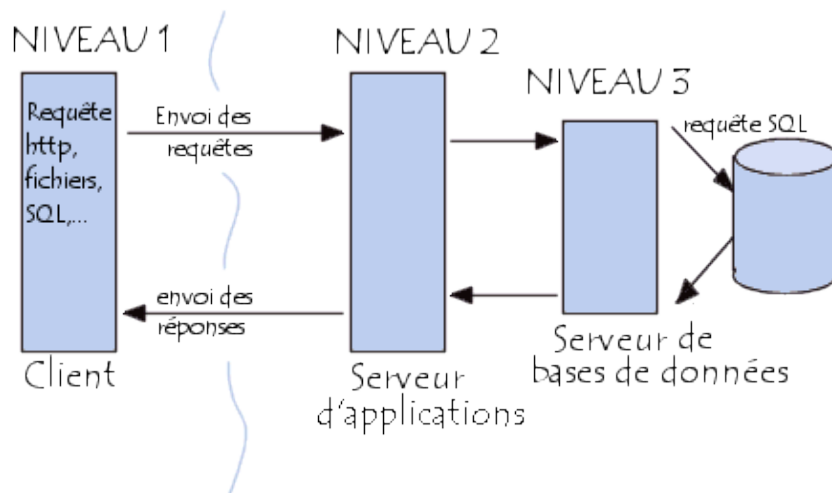


Figure 1: architecture client-serveur

✚ L'architecture à 2 niveaux

Exemple : un système d'informations utilisant une base de données centrale. Les clients reçoivent les données de l'utilisateur, initient les transactions ; le serveur reçoit et exécute les transactions, assure l'intégrité des données.

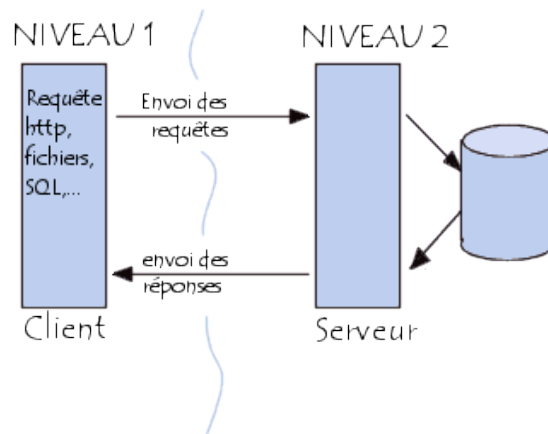


Figure 2: architecture à 2 niveaux

✚ L'architecture à 4 niveaux

Dans l'architecture à 3 niveaux, chaque serveur (niveaux 2 et 3) effectue une tâche (un service) spécialisée. Un serveur peut donc utiliser les services d'un ou plusieurs autres serveurs afin de fournir son propre service.

L'architecture à 3 niveaux permet de supporter un grand nombre de formats de présentation différents (propres à chaque client), tout en réutilisant certains des objets de présentation entre les clients. Réduction des redondances et elle est bien adaptée pour les applications Web devant supporter plusieurs types de clients.

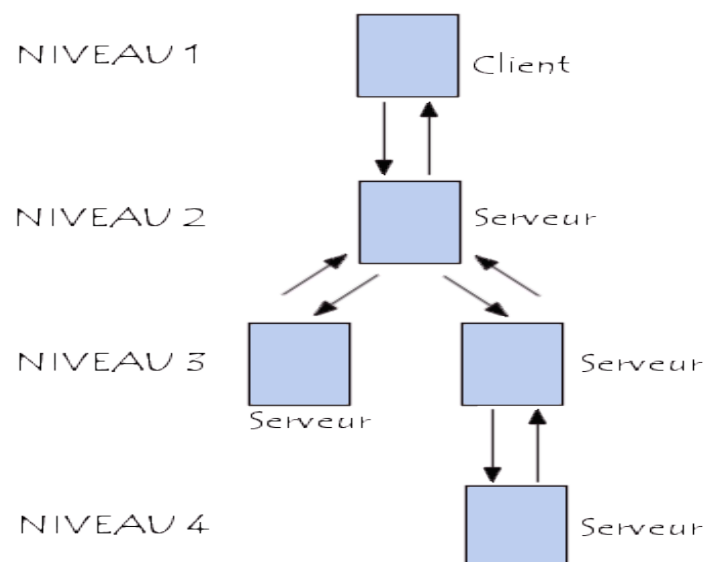


Figure 3: architecture à 4 niveaux

2. L'architecture pilotée par les événements

C'est une architecture qui produit, détecte et agit selon les événements du système pertinents pour les utilisateurs. Si aucun événement ne se produit, rien ne se passe dans le système. Cette architecture définit ces événements indispensables comme des déclencheurs. Lorsque ces déclencheurs s'activent, ils provoquent des comportements spécifiques.

Exemple : Avez-vous déjà voulu acheter quelque chose sur un site pour vous rendre compte que l'article en question dépassait largement votre budget ? Heureusement, certains sites web proposent de vous envoyer une alerte lorsque le prix d'un article baisse en dessous d'une certaine valeur. Il suffit d'ajouter une alerte sur le site web : « Si l'article atteint le prix X, veuillez m'en informer. » C'est comme si on disait au site : « Si l'événement X se produit, alors faites ceci. »

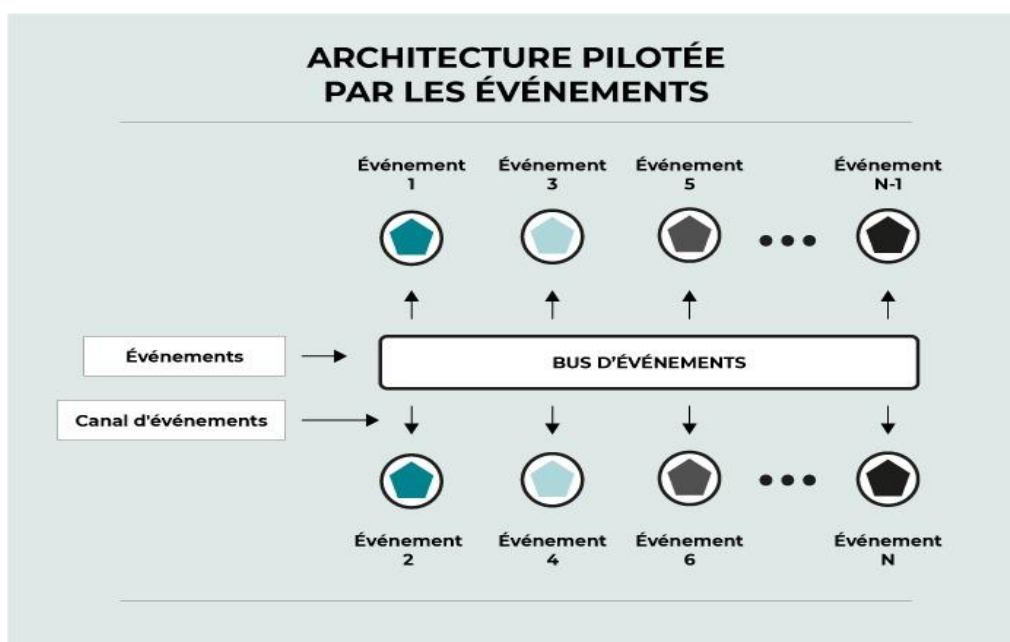


Figure 4: architecture pilotée par les événements

🚦 **Le bus d'événements (Event Bus) :** Il s'agit d'une ligne de communication qui relie tous les utilisateurs aux événements. Cette ligne n'est pas un câble physique mais fait référence à une connexion logique, similaire à celle qui existe entre votre ordinateur portable et un site web que vous visitez. Les événements sont transportés par ce bus d'événements abstrait et atteignent

tous les utilisateurs du système. Le bus n'existe pas réellement, mais il sert à décrire l'échange de données entre de nombreux composants ou utilisateurs.

- ✚ **Le canal d'événements (Event Channel) :** Un canal d'événements est une étiquette pour le type d'événements auxquels les utilisateurs s'abonnent. Par exemple, le canal « ventes » publiera tous les événements concernant les ventes à tous les utilisateurs abonnés. Sur Twitter, un hashtag comme #awesomejapan est un canal qui rassemble tous les tweets publiés par d'autres utilisateurs et contenant ce même hashtag. Si un utilisateur est abonné à un canal donné, il recevra tous les messages qui s'y rapportent.
- ✚ **Le traitement des événements (Event Processing) :** Toutes les actions entreprises après un événement donné sont exécutées dans le module de traitement des événements. Ce module agit selon un événement donné pour servir l'utilisateur. Par exemple : « Si je reçois ce type de message, faites ceci », ou de façon plus concrète : « Si je reçois un message de nouvel achat, alors facturez le client ».

L'utilisation d'une architecture pilotée par les événements présente plusieurs **avantages** :

- ✚ Si des utilisateurs ont besoin d'écouter (de s'abonner à) différents messages sur un sujet spécifique, l'architecture pilotée par les événements est la bonne approche. Cela se produit lorsque le système doit réagir à des événements sans comportement prédictif : par exemple, si je m'abonne à la chaîne « Informations africaines », je n'ai aucun contrôle sur les actualités que je recevrai ni sur leur quantité. Le système n'est pas prédictif ; il réagit aux événements générés par d'autres utilisateurs.
- ✚ Si ces utilisateurs sont en dehors de l'organisation, les événements peuvent être transportés dans un bus d'événements publics et atteindre tous les utilisateurs immédiatement.
- ✚ De nombreux événements peuvent être traités en même temps, voire des millions.

Il existe également quelques **inconvénients** majeurs :

- ✚ Le bus d'événements peut être surchargé.
- ✚ Si le bus d'événements tombe en panne, tout le système tombe en panne.
- ✚ Il n'y a pas de contrôle du flux d'événements : de nombreux événements peuvent se produire en même temps, créant un véritable chaos pour les utilisateurs.

3. L'architecture orientée services

L'architecture orientée services (ou SOA, Service-Oriented Architecture) est un modèle de conception qui rend des composants logiciels réutilisables, grâce à des interfaces de services qui utilisent un langage commun pour communiquer via un réseau.

Un service est une unité autonome de fonctionnalité logicielle, ou d'un ensemble de fonctionnalités, conçue pour réaliser une tâche précise comme récupérer des informations ou exécuter une opération. Il contient les intégrations de code et de données nécessaires pour exécuter une fonction métier distincte et complète. Vous pouvez y accéder à distance, et interagir avec lui ou le mettre à jour de manière indépendante.

En d'autres termes, l'architecture SOA permet à des composants logiciels déployés et gérés séparément de communiquer et de fonctionner ensemble sous la forme d'applications logicielles communes à différents systèmes.

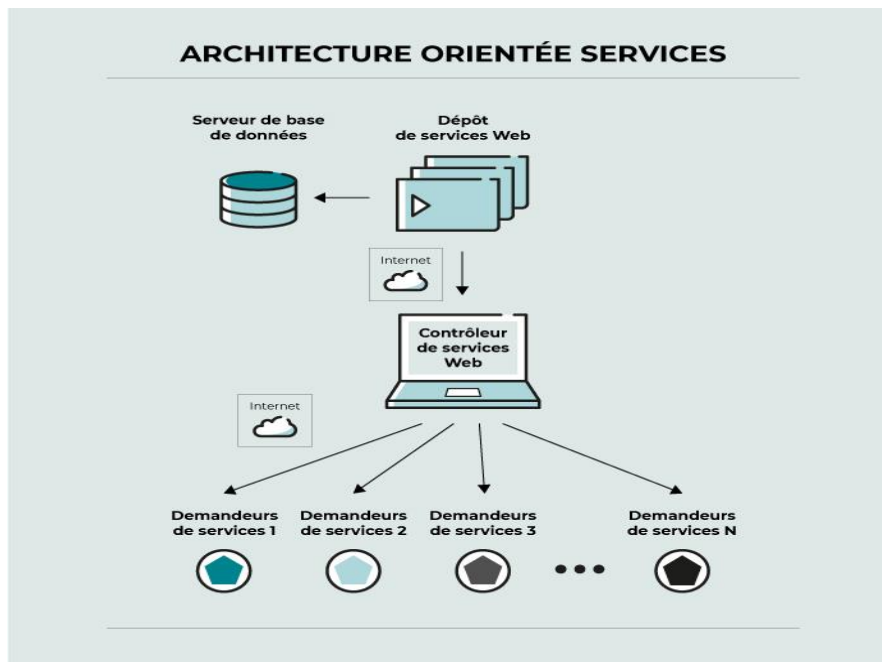


Figure 5: Architecture orientée service

- ✚ **Le dépôt de services web (Web service repository) :** Il s'agit d'une bibliothèque de services web conçue pour répondre à des demandes d'informations externes. L'information fournie est généralement un petit élément, comme un numéro, un mot, quelques variables, etc. Par exemple, un numéro de vol, un numéro de suivi de colis, le statut d'une commande (une lettre), etc. Cette bibliothèque est généralement documentée de manière très détaillée, car des applications externes font appel aux fonctions qu'elle contient.
- ✚ **Le contrôleur de services web (Web service controller) :** Ce module communique les informations contenues dans le dépôt de services web aux demandeurs de services. Lorsqu'un demandeur de service externe appelle une certaine fonction du dépôt de services web, le contrôleur de services web interprète la demande et recherche la fonction dans le dépôt de services web. Il exécute ensuite cette fonction et renvoie une valeur au demandeur.

- ✚ **Le serveur de base de données (Database Server) :** Ce serveur contient les tables, les index et les données gérés par l'application. Les recherches et les opérations d'insertion/suppression/mise à jour sont exécutées ici.
- ✚ **Les demandeurs de services (Service Requester) :** Il s'agit d'applications externes qui demandent des services au dépôt de services web par l'intermédiaire d'Internet, comme une organisation demandant des informations sur les vols à une compagnie aérienne, ou une autre entreprise demandant à un transporteur la localisation d'un colis à un moment donné.

Avantages de l'architecture orientée services :

- ✚ Ce modèle permet de collaborer avec des acteurs externes sans les laisser accéder à nos systèmes. Dans l'exemple ci-dessus, Visa ne veut pas ouvrir sa base de données à des tiers pour des raisons de sécurité, mais elle souhaite néanmoins que les clients puissent obtenir les informations dont ils ont besoin.
- ✚ Il permet également de partager avec le monde entier, de manière ordonnée et contrôlée, la sélection des fonctions les plus populaires du site. Visa dit : « Si quelqu'un veut connaître l'état de la carte X, appelez cette fonction, donnez la valeur de X et je répondrai par oui ou par non. » La fonction est ouverte au monde en tant que service dans un environnement très strict et contrôlé ; les clients ne peuvent pas accéder à ce qu'ils veulent.

Inconvénients de l'architecture orientée services :

- ✚ Les services web peuvent représenter une faiblesse au niveau du site pour les pirates qui veulent engorger le système. Certaines formes d'attaques sont des « dénis de service ». Elles consistent à demander le même service web des millions de fois par seconde, jusqu'à ce que le serveur tombe en panne. Il existe aujourd'hui une technologie permettant de résoudre ce problème, mais c'est toujours un problème à prendre en compte dans les architectures de services web.

- ✚ Le propriétaire du service web aide d'autres sites, mais reçoit une petite rémunération pour ce faire.

4. L'architecture modulaire

L'architecture modulaire consiste à l'assemblage de plusieurs éléments (modules) et permet d'avoir d'autres fonctionnalités en ajoutant d'autres modules.

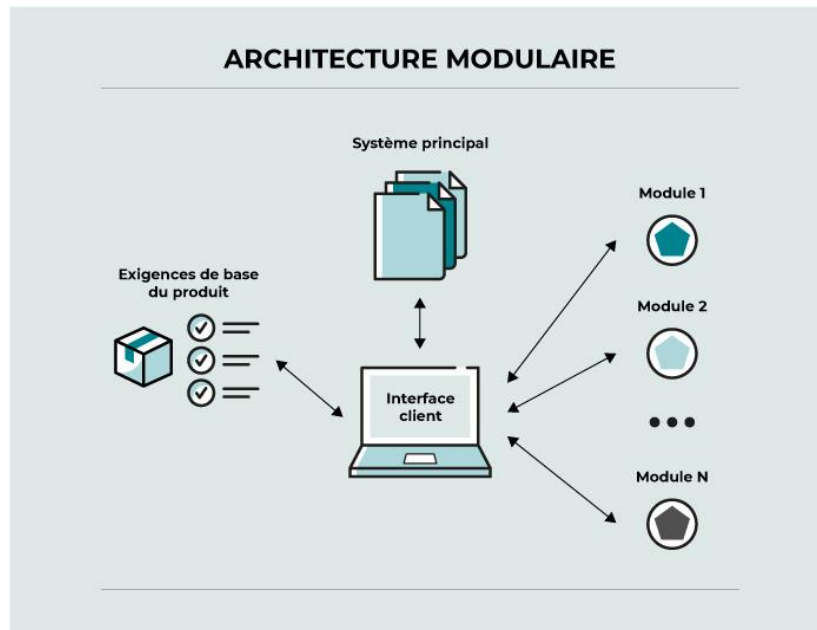


Figure 6: Architecture modulaire

- ✚ **Les exigences de base du produit (Baseline product requirements) :** Il s'agit de l'ensemble des exigences minimales qui définissent l'application, déterminées au début du processus de développement lorsqu'un ensemble initial de fonctionnalités a été inclus dans le produit.
- ✚ **Le système principal (Main System) :** Il s'agit de l'application à laquelle on connecte les modules. Le système principal doit fournir un moyen d'intégrer les modules et, par conséquent, il modifiera légèrement le produit de base original pour assurer la compatibilité.
- ✚ **L'interface client (Customer Interface) :** Il s'agit de la partie qui interagit avec le client, par exemple, un navigateur web (Chrome, Mozilla, etc.).

- ✚ **Les modules (Plug-in)** : Il s'agit de modules complémentaires qui complètent les exigences minimales de l'application et lui confèrent des fonctionnalités supplémentaires.

Les avantages d'une architecture modulaire:

- ✚ L'architecture modulaire est le meilleur moyen d'ajouter une fonctionnalité à un système qui n'a pas été conçu initialement pour cela.
- ✚ Cette architecture supprime les limites de quantité de fonctionnalités qu'une application peut offrir. Nous pouvons ajouter une infinité de modules (le navigateur Chrome dispose de centaines de modules appelés extensions).

Les inconvénients d'une architecture modulaire:

- ✚ Les modules peuvent être une source de virus et d'attaques venant d'acteurs externes.
- ✚ La présence de nombreux modules dans une application peut affecter ses performances.

5. L'architecture en couche

L'architecture en couches (aussi appelée architecture multi-tiers) est une pratique d'architecture logicielle qui propose de concevoir le système comme une superposition de strates, chaque strate étant définie par une responsabilité spécifique.

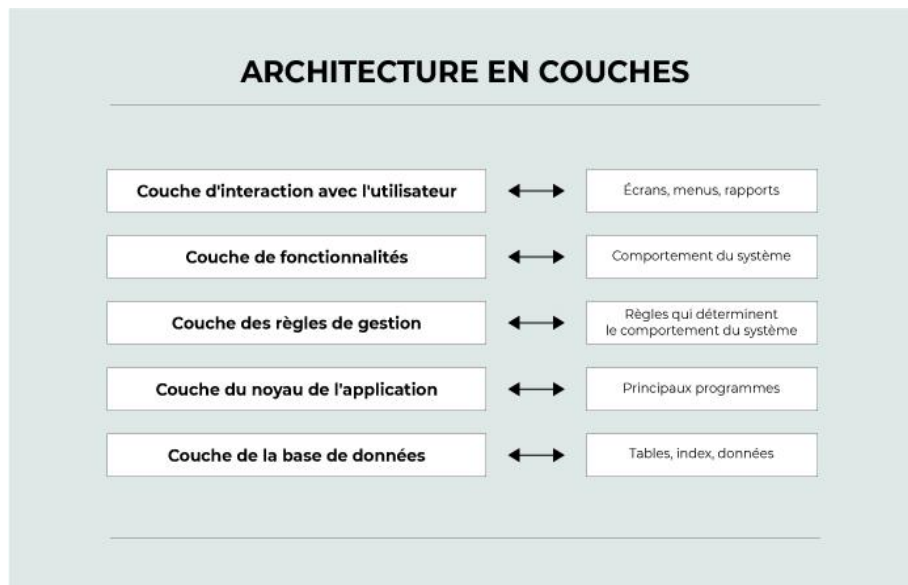


Figure 7: Architecture en couches

- ✚ **La couche d'interaction avec l'utilisateur (User interaction layer) :** C'est la couche qui interagit avec les utilisateurs par le biais d'écrans, de formulaires, de menus, de rapports, etc. C'est la couche la plus visible de l'application. Elle définit l'aspect de l'application.
- ✚ **La couche de fonctionnalités (Functionality Layer) :** Il s'agit de la couche qui présente les fonctions, les méthodes et les procédures du système selon la couche des règles de gestion. Elle détermine comment les menus déroulants fonctionnent, comment les boutons fonctionnent et comment le système navigue entre les écrans.
- ✚ **La couche des règles de gestion (Business rules layer) :** Cette couche contient des règles qui déterminent le comportement de l'ensemble de l'application, par exemple : « Si une facture est imprimée, il faut envoyer un courriel au client, sélectionner tous les articles vendus et diminuer leur stock dans le module de gestion des stocks. »
- ✚ **La couche du noyau de l'application (Application core layer) :** Cette couche contient les principaux programmes, les définitions du code et les fonctions de base de l'application. Les programmeurs travaillent la plupart du temps sur cette couche.

- ✚ **La couche de la base de données (Database layer) :** Cette couche contient les tables, les index et les données gérées par l'application. Les recherches et les opérations d'insertion/suppression/mise à jour sont exécutées ici.

Les avantages de l'architecture en couches:

- ✚ Les couches sont autonomes : les changements effectués sur une couche n'affectent pas les autres. C'est pratique car nous pouvons augmenter les fonctionnalités d'une couche, par exemple faire en sorte qu'une application qui ne fonctionne que sur les PC fonctionne sur les téléphones et les tablettes, sans avoir à réécrire toute l'application.
- ✚ Les couches permettent une meilleure personnalisation du système.

Les Inconvénients de l'architecture en couches :

- ✚ Les couches rendent une application plus difficile à maintenir. Chaque changement nécessite une analyse.
- ✚ Les couches peuvent affecter les performances des applications car elles créent une surcharge dans l'exécution : chaque couche des niveaux supérieurs doit se connecter à celles des niveaux inférieurs à chaque opération du système.

6. L'architecture centrée sur les données

L'architecture centrée sur les données est le processus qui permet de standardiser la façon dont les entreprises collectent, stockent, transforment, distribuent et utilisent les données. Le but est de fournir les données pertinentes aux personnes qui en ont besoin au moment opportun et de les aider à les interpréter.

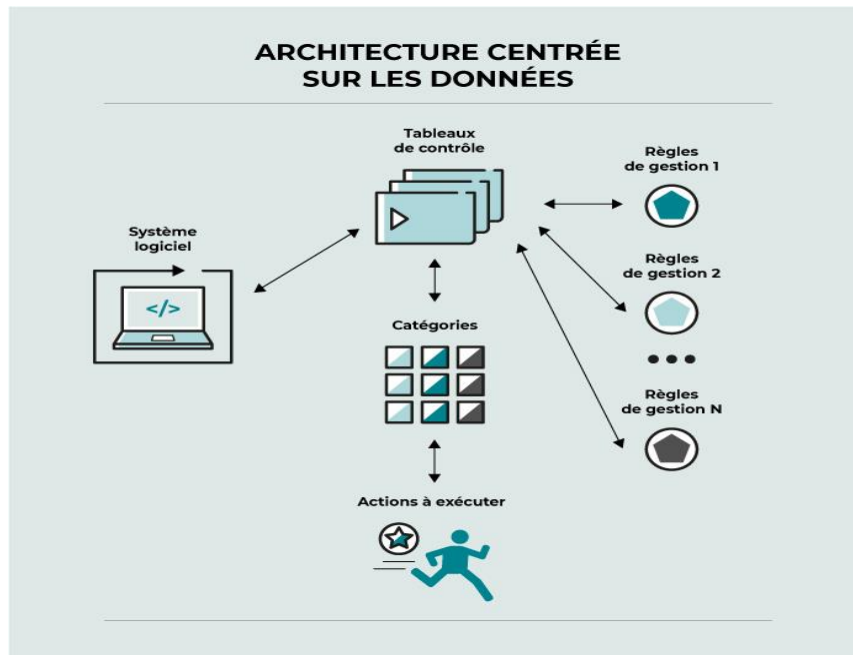


Figure 8: L'architecture centrée sur les données

- ✚ **Le système logiciel (Software System)** : Le système développé en utilisant le modèle d'architecture centré sur les données.
- ✚ **Les tableaux de contrôle (Control Tables)** : Un ensemble de tableaux qui définissent les actions que le système doit exécuter pour chaque catégorie.
- ✚ **Les catégories (Categories)** : Un ensemble de catégories d'un élément dans le système (types de clients, types de produits, types d'articles, types d'employés) utilisé pour exécuter des actions.
- ✚ **Les actions à exécuter (Action Items)** : Un ensemble d'actions à exécuter pour une certaine catégorie.
- ✚ **Les règles de gestion (Business Rules)** : Les règles de gestion qui déterminent les actions à exécuter en fonction des catégories.

Les avantages d'une architecture centrée sur les données:

- ✚ Si des catégories changent ou sont ajoutées, il n'est pas nécessaire de modifier le code.
- ✚ Les règles de gestion sont beaucoup plus faciles à gérer dans un tableau que dans un ensemble de programmes.

Les Inconvénients d'une architecture centrée sur les données:

- ✚ Ce schéma peut poser des problèmes de performance car chaque fois que le code s'exécute, il doit rechercher des données dans une table.
- ✚ Si la table de contrôle est endommagée, l'ensemble du système ne fonctionne plus.

III. L'organisation en couche d'un système informatique

1. Couche d'abstraction

La notion de couche d'abstraction (*abstraction layer*) nous permet de décrire les systèmes informatiques comme s'il s'agissait d'empilement de couches qui se superposent en apportant à chaque niveau supplémentaire de nouvelles fonctions de plus en plus élaborées et reposant sur les fonction plus élémentaires assurées par les couches sous-jacentes.

Ces notions de couches d'abstraction ou de structure en couches seront réutilisées plusieurs fois tantôt pour décrire les langages de programmation, tantôt pour décrire l'architecture des machines, pour la structure des logiciels, les communications des réseaux, etc.

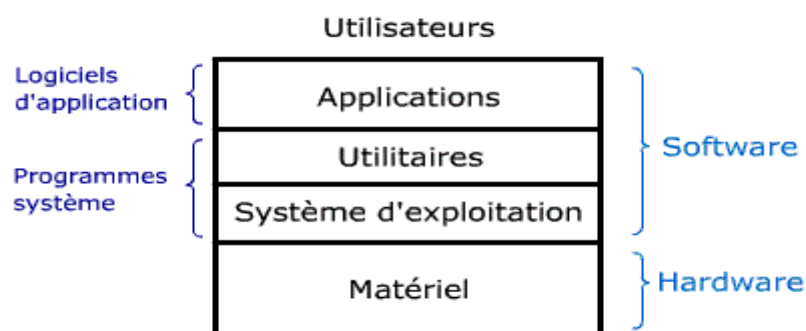


Figure 9: Représentation de la couche d'abstraction

Matériel : Circuits électroniques / circuits logiques. On dira que c'est au niveau du matériel que se trouve la couche d'abstraction la plus basse.

Système d'exploitation : Élément le plus déterminant d'un système informatique.

Applications : Traitement de texte, gestionnaire de bases de données, tableurs etc. Compilateurs, debugger.

Utilitaires : Services de base aux utilisateurs. Ex. interface graphique, interpréteur de commandes, gestionnaires divers qui tournent en tâches de fond, imprimant, cherchent le courrier etc.

Utilisateurs : C'est à eux que le système informatique est destiné. Les utilisateurs interagissent avec la couche de plus haut niveau

2. Structure en couche d'un logiciel

Chaque couche est construite sur la couche précédente. Elle est une sorte de machine virtuelle qui permet de faire abstraction des détails qui composent les couches sous-jacentes.

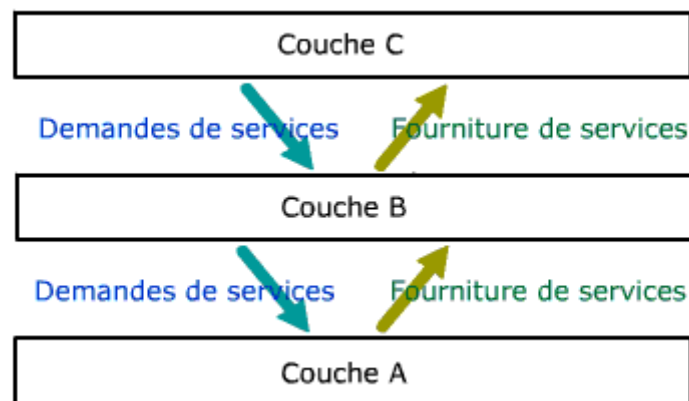


Figure 10: Structure en couche d'un logiciel

- ✚ Chaque couche offre des services à la couche qui lui est supérieure et est cliente de la couche sous-jacente.
- ✚ Elle ne communique avec ces deux couches adjacentes qu'au travers d'interfaces bien définies.
- ✚ Chaque couche est seule responsable de son fonctionnement interne. Les éventuelles modifications de ce fonctionnement ne doivent pas influencer les autres couches.

Exemple :

Le système de fichier

Les développeurs d'applications disposent de primitives pour ouvrir, lire, écrire et refermer un fichier comme on le ferait avec un document quelconque, un livre par exemple. La programmation de fichier est donc possible sans devoir connaître le mécanisme interne des disques, des disquettes ou d'autres lecteurs amovibles. Le système de fichiers est donc pour le développeur une couche logicielle dont il ne doit connaître que l'interface d'application pour pouvoir l'utiliser en pouvant faire abstraction des détails propres à la mise en œuvre des fichiers.

3. La couche de langages et machines

Les langages informatiques sont à considérer à plusieurs niveaux allant du plus bas, le plus proche des composants électronique au plus proche de l'utilisateur, plus indépendant du matériel.

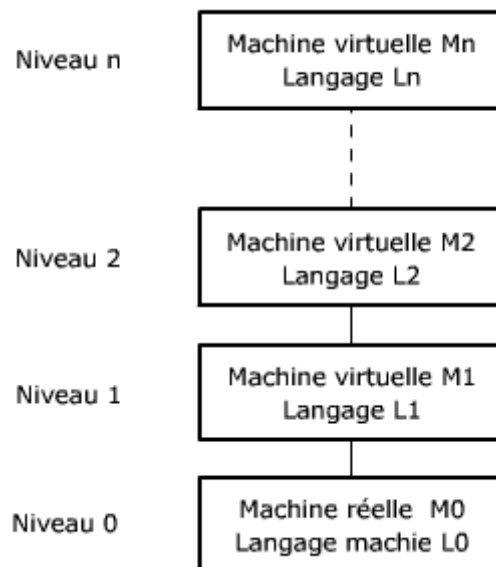


Figure 11: La couche de langages et machines

- ✚ Soit L_0 , le langage machine dont les instructions sont exécutées "en dur" par les circuits électroniques de la machine M_0 .
- ✚ L_0 est parfaitement adapté aux circuits électroniques mais pour nous, humains, son code est fastidieux !
- ✚ Il faut donc un langage L_1 plus proche de l'utilisateur.
- ✚ C'est comme si une machine hypothétique M_1 exécutait directement les instructions L_1 .

- ✚ Le langage L1 n'est pas très différent de L0. On a dès lors besoin d'un langage L2 plus proche de l'utilisateur et moins dépendant de la machine
- ✚ Chaque langage s'appuie sur son prédécesseur et devient un peu plus pratique que le précédent.

Exécution d'un programme écrit en L1

Comment fonctionne la machine virtuelle M1 ?

Compilation

Chaque instruction L1 est remplacée par une suite d'instructions L0.

La machine M0 exécute ensuite le programme en langage L0.

Interprétation

Un programme écrit en L0 examine les instructions L1 et exécute directement des séquences d'instructions qui correspondent aux tâches demandées.

4. Machines multicouches

Nous nous référons ici aussi à l'approche en six couches que Andrew Tanenbaum développe dans son ouvrage « Architecture de l'ordinateur ». Maintenant que nous savons comment interagissent les langages de niveaux successivement de plus en plus élaborés, voyons quels sont réellement ces langages dans nos machines informatiques.

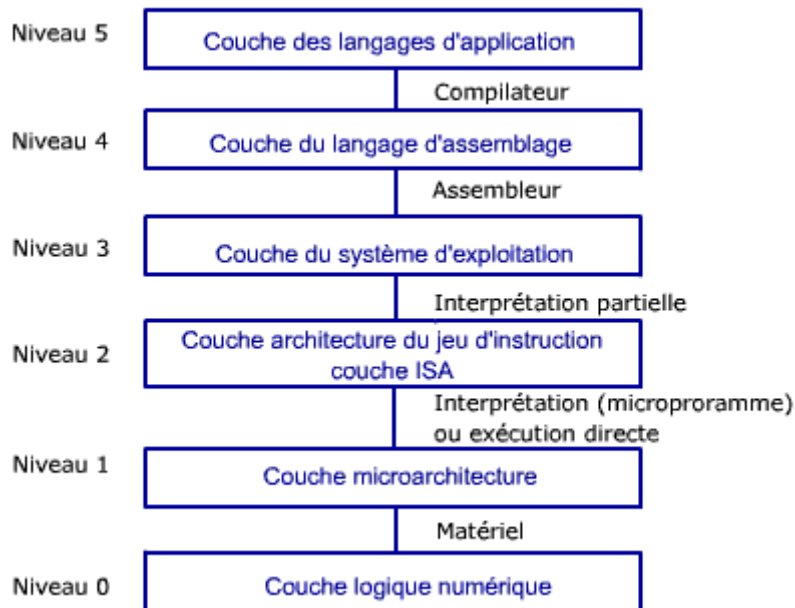


Figure 12: Machines multicouches

IV. Sous-système informatique

Dans les modèles UML, les sous-systèmes sont un type de composant stéréotypé représentant des unités comportementales indépendantes dans un système. Les sous-systèmes sont utilisés dans les diagrammes de classes, de composants et de cas d'utilisation pour représenter des composants de grande taille dans le système que vous modélisez.

Vous pouvez modéliser un système entier sous forme de hiérarchie de sous-systèmes. Vous pouvez également définir le comportement que chaque sous-système représente en spécifiant les interfaces avec les sous-systèmes, ainsi que les opérations qui prennent en charge ces interfaces.

Dans les diagrammes, les compartiments affichent des informations sur les attributs, les opérations, les interfaces fournies, les interfaces requises, les réalisations et la structure interne du sous-système.

Généralement, un sous-système porte un nom qui décrit son contenu et son rôle dans le système.

Comme le montre la figure suivante, un sous-système est représenté par un rectangle qui contient le nom du sous-système. Il contient également le mot clé «Subsystem» et l'icône du sous-système.

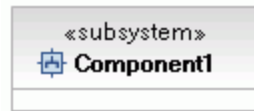


Figure 13: Représentation d'un sous-système

V. Les paquetages et leurs relations

En UML, on peut regrouper des éléments en utilisant des paquetages. Les paquetages UML peuvent servir à organiser pratiquement n'importe quel élément UML: des classes, des cas d'utilisation, des interfaces, des diagrammes, ... et même des paquetages imbriqués et à fournir un espace de noms pour ces éléments. Les éléments contenus dans un paquetage sont généralement de même nature et de même niveau sémantique.

Généralement, il existe un seul paquetage racine qui détient la totalité du modèle d'un système. Les diagrammes de paquetages font partie de la vue de développement, qui s'intéresse à l'organisation des parties du système en modules et paquetages.

1. Notion de paquetage

Lorsque nous sommes en présence d'un système de grande taille, il peut être intéressant de le décomposer en plusieurs parties (appelées paquetage). Un paquetage est donc un regroupement de différents éléments d'un système (regroupement de classes, diagrammes, fonctions, interfaces...). Cela permet de clarifier le modèle en l'organisant. Il est représenté par un dossier avec son nom à l'intérieur :

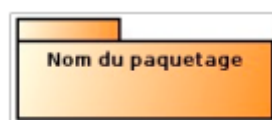


Figure 14: Représentation d'un paquetage

Il est possible de représenter les éléments du système appartenant au paquetage :

à l'intérieur de celui-ci :



ou à l'extérieur :

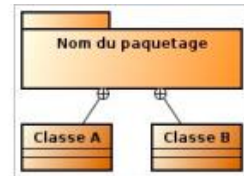


Figure 15: Un paquetage contenant deux classes

Pour faire appel à un élément d'un paquetage, nous indiquons le nom du paquetage (espace de nommage) suivi de deux fois deux points (::) puis du nom de l'élément.

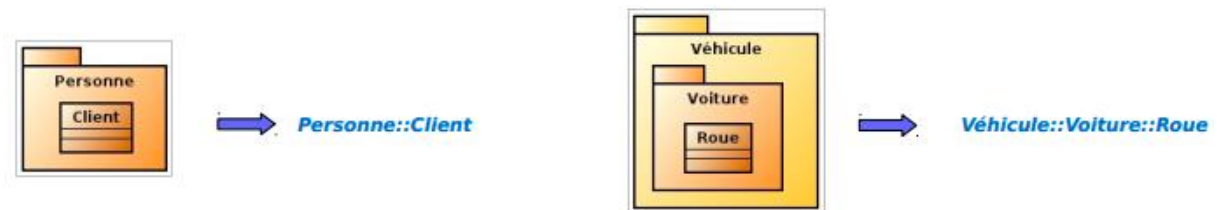


Figure 16:élément d'un paquetage

Les paquetages peuvent s'imbriquer (décomposition hiérarchique) mais pas se chevaucher. Un élément du système ne peut appartenir qu'à un et un seul paquetage. Chaque paquetage doit posséder un nom différent.



Rôle du diagramme de paquetages

Le diagramme de paquetages est un diagramme structurel (statique) d'UML qui représente les paquetages (ou espaces de noms) composant un système, ainsi que les relations qui lient ces différents paquetages.

2. Dépendance entre paquetages

a) Visibilité :

Chaque éléments d'un paquetage est soit :

-  **privé**, c'est-à-dire encapsulé dans le paquetage et invisible à l'extérieur de celui-ci. Un élément privé est désigné par un signe **-** devant lui.
-  **public**, c'est-à-dire visible et accessible e de l'extérieur du paquetage. Un élément public est désigné par un signe **+** devant lui.

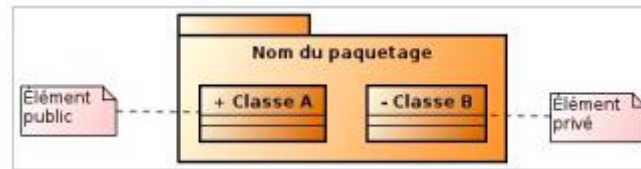


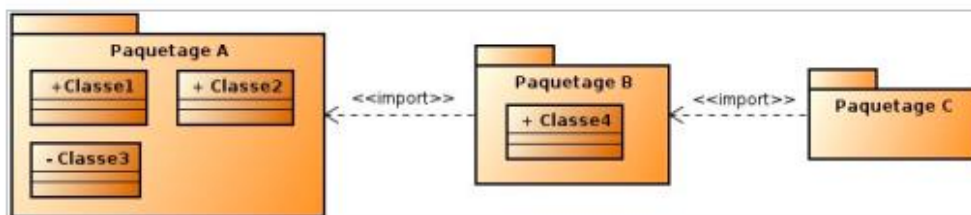
Figure 17:visibilité d'un paquetage

b) Dépendance de type « **import** »

Elle correspond à l'importation par un paquetage B de tous les éléments publics d'un paquetage A. Ces éléments :

- ✚ auront la visibilité « **public** » dans le paquetage B (et seraient donc aussi transmis à un paquetage C qui ferait une importation du paquetage B.
- ✚ seront accessibles au paquetage B sans avoir à utiliser explicitement le nom du paquetage A.

La dépendance de type « **import** » est représentée par une flèche pointillée muni du stéréotype **<<import>>**.

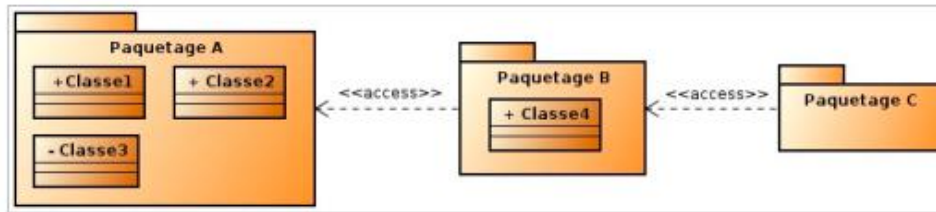


Le paquetage B importe **Classe1** et **Classe2** (pas **Classe3** qui a une visibilité de type **privée**). **Classe1** et **Classe2** ont une visibilité de type **public** dans paquetage B. Le paquetage C importe **Classe1**, **Classe2** et **Classe4**.

c) Dépendance de type « **access** »

Elle correspond à l'accès par un paquetage B de tous les éléments publics d'un paquetage A. Ces éléments auront la visibilité privée dans le paquetage B, ils ne peuvent donc pas être transmis à un paquetage C qui ferait une importation ou un accès au paquetage B (pas de transitivité).

La dépendance de type « **access** » est représentée par une flèche pointillée muni du stéréotype **<<access>>**.

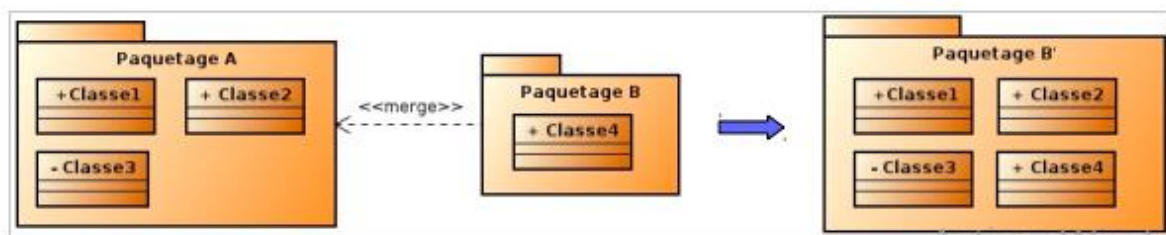


Le paquetage B a accès à **Classe1** et **Classe2** (pas à **Classe3** qui a une visibilité de type **privée**). **Classe1** et **Classe2** ont une visibilité de type **privé** dans paquetage B. Le paquetage C a accès à **Classe4** (pas à **Classe1** et **Classe2** qui ont une visibilité de type **privée** dans paquetage B).

d) Dépendances de type « merge »

Elle correspond à la fusion de 2 paquetages en un seul.

La dépendance de type « merge » est représentée par une flèche pointillée muni du stéréotype **<<merge>>**.



Le paquetage A est fusionné dans le paquetage B (le paquetage A n'est pas modifié alors que le paquetage B est écrasé pour accueillir la fusion des 2 paquetages).

Activer Window
Accédez aux paramé

VI. Diagramme de composants : organisation du code en modules, dépendances

1. Notion de composant (component) et d'interface

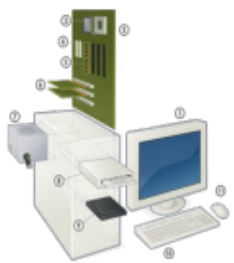
En UML, un **composant** est un élément logiciel remplaçable et réutilisable qui fournit ou reçoit un service bien précis. Il peut être vu comme une pièce détachée du logiciel. Les plu-gins, les drivers, les codecs, les bibliothèques sont des composants.

La notion de composant est proche de celle d'objet, dans le sens de la modularité et de réutilisation avec toutefois une granularité qui peut être différente. Le composant est à l'architecture du logiciel ce que l'objet est à l'architecture du code.

Les composants fournissent des services via des interfaces. Un composant peut être remplacé par n'importe quel autre composant compatible c'est-à-dire ayant les mêmes interfaces. Un composant peut évoluer indépendamment des applications ou des autres composants qui l'utilise à partir du moment où les interfaces sont respectées.

Il existe deux types d'interface :

- + Les interfaces requises : Ce sont des interfaces qui fournissent un service au composant et dont il a besoin pour fonctionner.
- + Les interfaces fournies : Ce sont des interfaces par lesquels le composant fourni lui-même un service.



Ex : parallèle avec les composants d'un ordinateur.

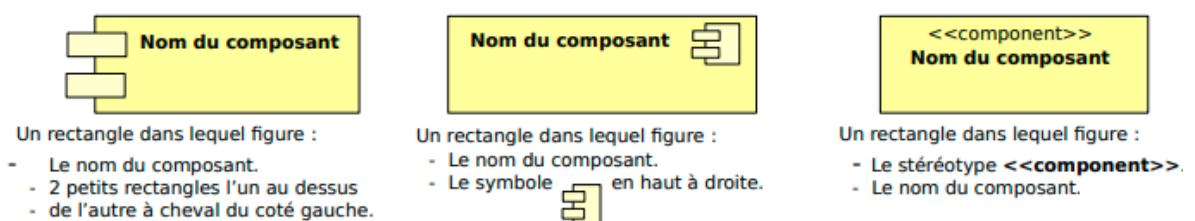
Un ordinateur est un ensemble de composants modulaires qui fournissent et reçoivent des services (carte mère, carte graphique, disque dur, clavier, écran...). Chacun de ces composants est remplaçable par un autre composant (pas forcément identique) à condition qu'il ait des interfaces compatibles (nous ne pouvons pas mettre un écran avec une connexion VGA à la place d'un écran avec une connexion HDMI).

ATTENTION : ceci n'est qu'un parallèle pour faire comprendre la notion de composant. **En UML les composants ne sont pas des éléments matériels mais des éléments logiciels.** Éléments logiciels qui par contre seront installés sur des éléments matériels (ce que nous verrons lorsque nous aborderons le diagramme de déploiement).

2. Représentation graphique

+ Les composants :

Il existe plusieurs possibilités pour représenter un composant:



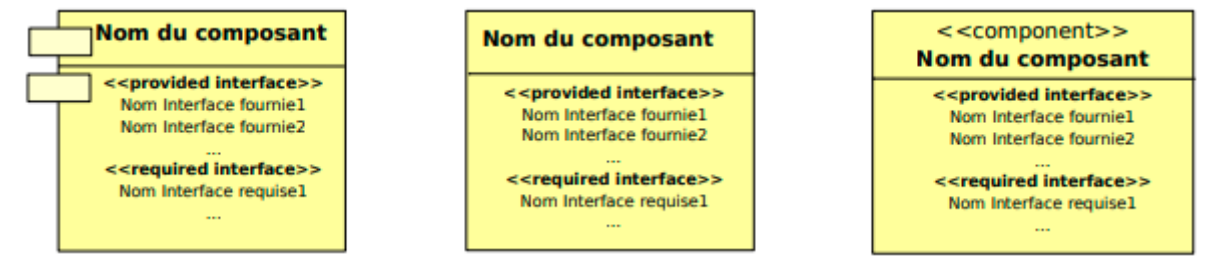
+ Les interfaces :

Là aussi, il existe plusieurs possibilités pour représenter les interfaces :

- Intégrées dans la représentation du composant :

Nous reprenons l'une des trois représentations du composant que nous venons juste de voir et nous ajoutons un compartiment dans lequel nous listons les

interfaces requises et fournies (grâce aux stéréotypes `<<required interface>>` et `<<provided interface>>`).



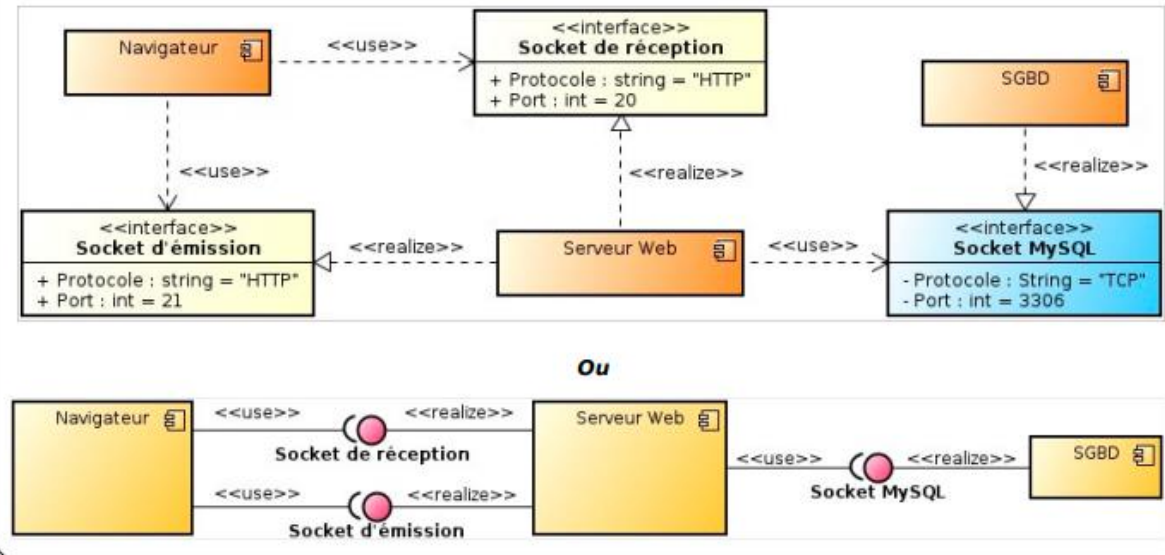
- Dans un classeur séparé du composant dans lequel sont listés les différents services :
- ✓ Les interfaces requises sont reliées au composant par une flèche en pointillées sur laquelle figure le stéréotype `<<use>>`.
- ✓ Les interfaces fournies sont reliées au composant par une flèche en pointillées sur laquelle figure le stéréotype `<<realize>>` (le bout de la flèche est un triangle vide).



- Avec des connecteurs d'assemblage :
- ✓ Les interfaces requises (représentées par un demi-cercle) et les interfaces fournies (représentées par un cercle) sont raccordées au composant par un trait.

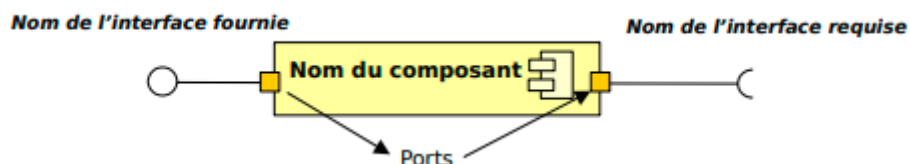


Ex : Transfert de données par Internet.



Les ports

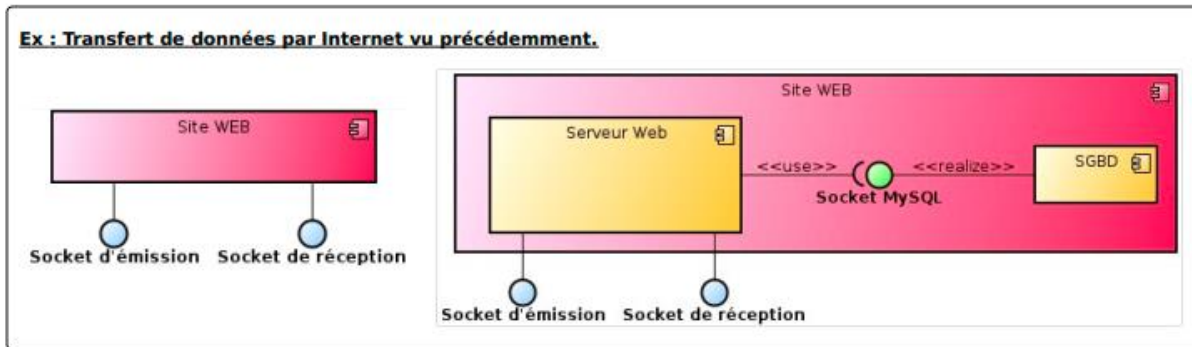
Le port est le point de connexion entre le composant et son environnement, il est la matérialisation de l'interface. Nous le représentons par un petit carré à la périphérie du composant.



Boite noire-boites blanche

Un composant peut être vu de 2 manières :

- ✓ Comme une boîte noire dont nous ne connaissons pas le contenu et auquel nous accédons via les interfaces qui sont la seule partie visible.
- ✓ Comme une boîte blanche en spécifiant les objets qui constituent le composant et en indiquant leurs relations.



3. Rôle du diagramme de composants

Le diagramme de composants fait parti des diagrammes structuraux (statiques) d'UML. Il permet de représenter les différents éléments logiciels (composants) du système et leurs dépendances (relations qui les lient). Comme nous venons de le voir lorsque nous définissons la notion de composant, ces dépendances peuvent être :

- + Des relations de compositions (boîte blanche)
- + Des relations d'assemblages et de connexion (via des interfaces)

Mais elles peuvent aussi être d'autre type tel que :

- + Des contraintes de compilation, des éditions de liens (les composant sont alors des fichiers de code source, des fichiers en binaire, des bibliothèques...). Dans ce cas, un stéréotype peut préciser la nature de la dépendance.

VII. Diagramme de déploiement : déploiement physique du système (machines, réseaux, etc.)

Tout d'abord, le diagramme de déploiement est une vue statique qui sert à représenter l'utilisation de l'infrastructure physique (machines, réseaux, ...) par le système et la manière dont les composants du système sont répartis ainsi que leurs relations entre eux.

1. Rôles du diagramme de déploiement

Comme mentionné ci-dessus, le diagramme de déploiement fait parti des diagrammes structuraux (statiques). Il permet donc :

- ✚ De faire la disposition physique des ressources qui constituent le système et montre la répartition des composants (éléments logiciels) sur ces matériels ;
- ✚ De connaître la nature des connexions de communication entre les différentes ressources matérielles.

2. Les éléments du diagramme de déploiement

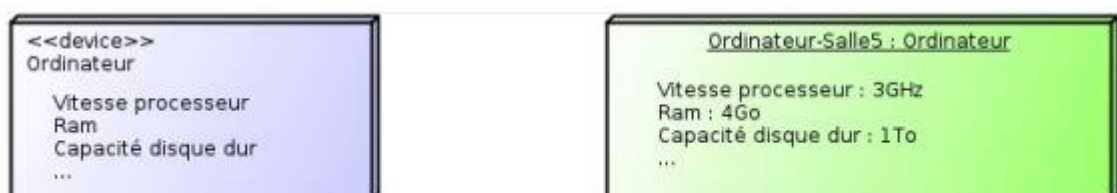
✚ Les nœuds :

Un nœud est une ressource matérielle du système. En général, cette ressource possède au minimum de la mémoire et parfois aussi des capacités de calcul (des ressources humaines ou des périphériques sont des ressources modélisées par les nœuds). Les ressources matérielles sont quelques fois représentées avec le stéréotypé `<<device>>` (exemple : ordinateurs de bureau, ...).

- Un nœud est représenté par un parallélépipède rectangle dans lequel figure son nom :

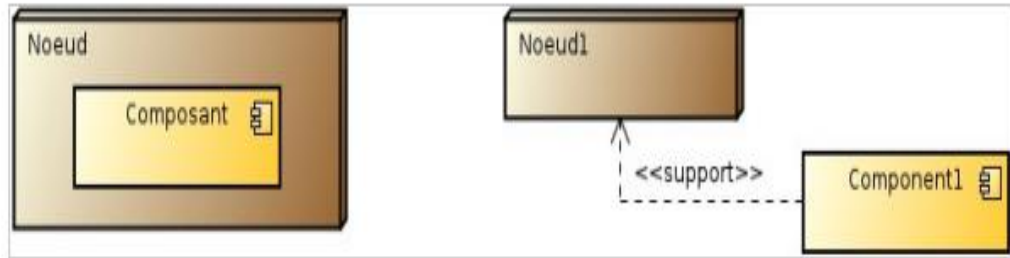


- Un nœud possède des attributs (quantité de mémoire, vitesse du processeur, marque, type, ...) que nous pouvons spécifier à l'intérieur du parallélépipède :



- Pour montrer qu'un composant (`<<Device>>`) est affecté sur un nœud, il faut :
 - Soit placer le composant dans le nœud.

- Soit en le reliant à l'aide d'une relation de dépendance (flèche en pointillées) stéréotypée <<support>> orienté du composant vers le nœud.



✚ Les chemins de communication

Les différents nœuds qui apparaissent dans le diagramme de déploiement sont connectés entre eux par des lignes qui symbolisent un support de communication → Ce sont les chemins de communications. Le chemin de communication est donc un lien qui permet de modéliser de façon simpliste la communication entre 2 nœuds (liaison Ethernet, USB, série...).

Il est possible de faire figurer sur ce lien :

- Les cardinalités ;
- Des contraintes entre accolades (pour indiquer par exemple qu'un accès est sécurisé) ;
- Le type de réseau et/ou son débit en l'indiquant comme un stéréotype...

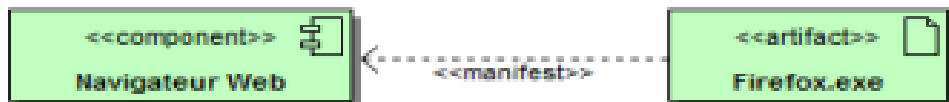


✚ Les artefacts :

L'artefact est le terme générique qui désigne n'importe quel élément produit du travail, c'est un élément concret et existant dans le monde réel (document, exécutable, fichier, base de donnée...). L'implémentation des modèles se fait sous

forme d'artefacts. On dit que l'artefact est la manifestation du modèle qu'il implémente.

Un artefact se représente par un rectangle contenant son nom et le stéréotype `<<artefact>>`. Un artefact qui est la manifestation d'un composant est relié à celui-ci par une relation de dépendance (flèche en pointillés) stéréotypé `<<manifest>>` orienté de l'artefact vers le composant.

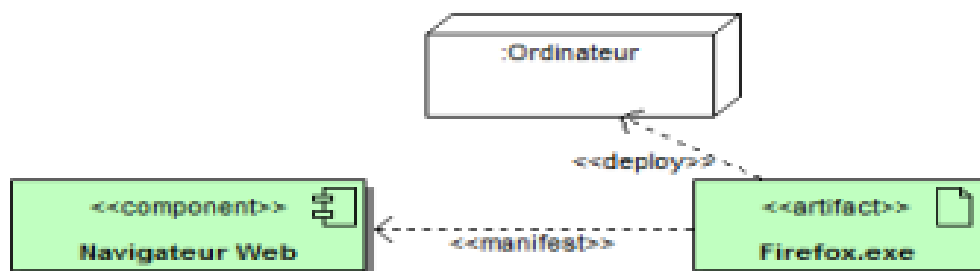


L'artefact est placé soit :

- À l'intérieur du nœud dans lequel il est déployé :



- À l'extérieur du nœud dans lequel il est déployé, mais relié à celui-ci par une relation de dépendance stéréotypé `<<deploy>>`.



CONCLUSION

Au terme de ce développement, nous devons retenir que tout système informatique est constitué de 02 grandes composantes : le matériel et le logiciel.

Le matériel n'est rien d'autre que l'ensemble des composants électroniques et informatiques (serveurs, ordinateurs, consoles de jeux, ...) tandis que le logiciel est tout simplement un ensemble de programmes et procédures nécessaires au fonctionnement d'un système informatique.

Il est donc important de connaître le type de matériels informatiques et le type de logiciel qu'il faudra utiliser pour modéliser un système informatique pour une bonne optimisation et un bon fonctionnement à long terme.

BIBLIOGRAPHIE

Ouvrages

- ✓ Pascal Roques UML 2.5 par la pratique 8è édition. Paris : Editions EYROLLES.
- ✓ Xavier B., Isabelle M. UML2 pour les développeurs – Cours avec exercices corrigés. Paris : Editions EYROLLES. ISBN 2-212-12029X.
- ✓ [Cazes 2003] Architecture des machines et des systèmes informatiques, Dunod [Tanenbaum 1988]
- ✓ Architecture de l'ordinateur, InterEditions
- ✓ [De Blasi 1990] Computer Architecture, Addison Wesley
- ✓ [Krakowiak 1982] Logiciel de base, Université de Grenoble

Sites web consultés

- ✓ <https://openclassrooms.com/fr/>
- ✓ <https://wikipedia.org/>
- ✓ <https://laurent-audibert.developpez.com/>

TABLE DES MATIERES

INTRODUCTION.....	2
I. DEFINITIONS	3
II. LES MODELES D'ARCHITECTURE.....	3
1. Architecture n-niveaux	3
2. L'architecture pilotée par les évènements.....	6
3. L'architecture orientée services	8
4. L'architecture modulaire	11
5. L'architecture en couche	12
6. L'architecture centrée sur les données	14
III. L'organisation en couche d'un système informatique	16
1. Couche d'abstraction.....	16
2. Structure en couche d'un logiciel.....	17
3. La couche de langages et machines	18
4. Machines multicouches.....	19
IV. Sous-système informatique.....	20
V. Les paquetages et leurs relations.....	21
1. Notion de paquetage	21
2. Dépendance entre paquetages	22
a) Visibilité :	22
b) Dépendance de type « import ».....	23
c) Dépendance de type « access »	23
d) Dépendances de type « merge ».....	24
VI. Diagramme de composants : organisation du code en modules, dépendances	24
1. Notion de composant (component) et d'interface	24
2. Représentation graphique.....	25
3. Rôle du diagramme de composants.....	28
VII. Diagramme de déploiement : déploiement physique du système (machines, réseaux, etc.)..	28
1. Rôles du diagramme de déploiement	28
2. Les éléments du diagramme de déploiement.....	29
CONCLUSION.....	32
BIBLIOGRAPHIE	33