Architecture Logicielle

Introduction

INSA BADJI

UVS
UNIVERSITE VIRTUELLE DU SENEGAL
foo nekk foofu la

Introduction

- Qu’est-ce qu’une architecture logicielle ?
- L’architecture d’un logiciel est la structure des structures (modules) d’un système. Elle est composée
  - des composants logiciels
  - des propriétés externes visibles de ces composants
  - Les relations entre ces composants

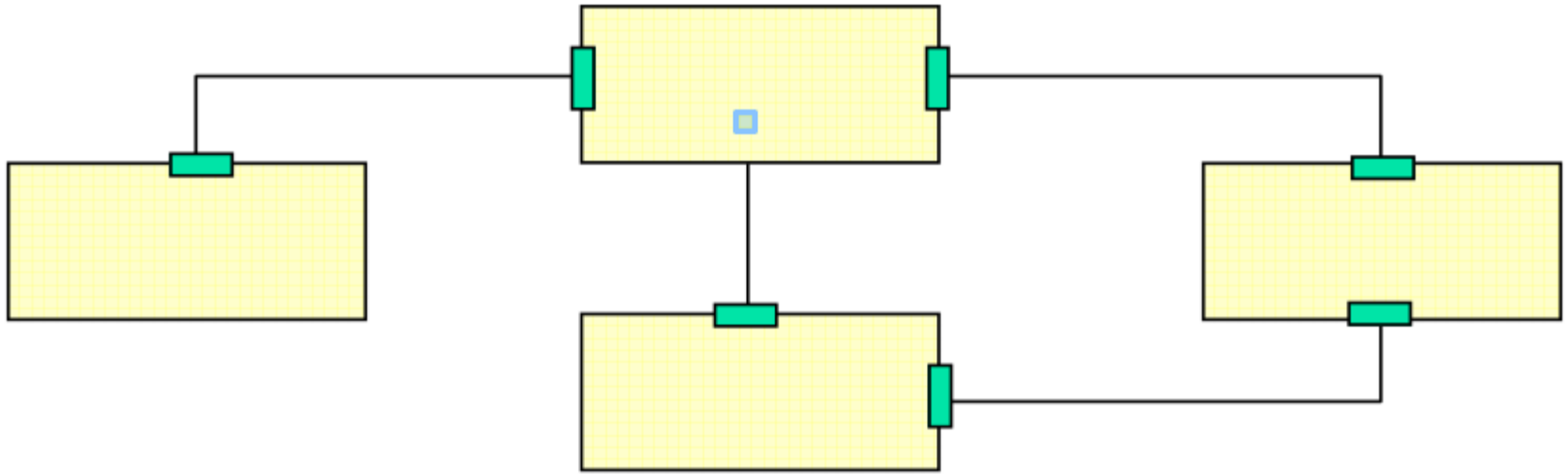
[Bass, Clements, and Kazman (1998)]

Introduction

- On peut dire qu'une architecture logicielle consiste à:
  - Décrire l'organisation générale d'un système et sa décomposition en sous-systèmes ou composants
  - Décrire les interactions et le flot de contrôle entre les sous-systèmes
  - Décrire également les composants utilisés pour implanter les fonctionnalités des sous-systèmes
  - Les propriétés de ces composants
  - Leur contenu (e.g., classes, autres composants)
  - Les machines ou dispositifs matériels sur lesquels ces modules seront déployés

Définition

- Une architecture logicielle est une représentation abstraite d'un système exprimée essentiellement à l’aide de composants logiciels en interaction via des connecteurs



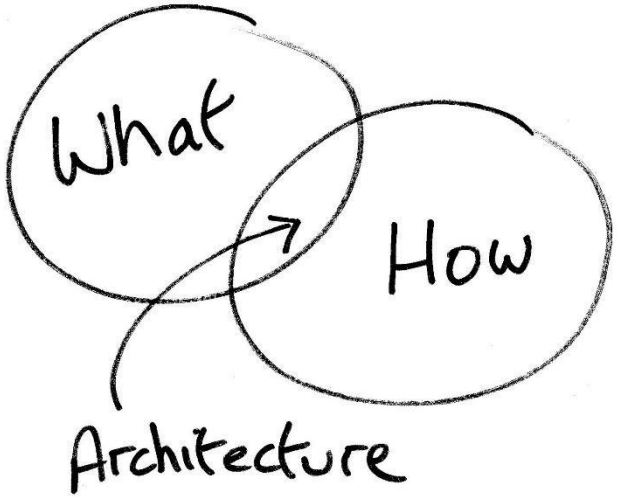
Tâches de la phase de conception

- Concevoir l'**Architecture**
  - « Qui » fera « quoi » et « où » ?

Selon David Garlan, l'architecture logicielle est le pont qui relie les exigences et l'implémentation.

- Concevoir les interfaces utilisateurs du logiciel
- Concevoir les bases de données
- Concevoir les contrôles du logiciel
  - Mise en œuvre de tous les aspects liés au contrôle, à la correction, à la sécurité, à la tolérance aux fautes, à la protection des données, etc.
- Concevoir le réseau
  - Communication entre les processus distribués

Architecture logicielle



- Décrit
  - L'organisation générale du logiciel
  - Les modules et leurs interfaces
  - L'agencement du logiciel des sous-systèmes, des modules et des composants
    - Leurs propriétés
    - Leur composition («contient »)
    - Leurs collaborations («utilise»)
- Dépend des besoins fonctionnels et non fonctionnels du logiciel

Les besoins fonctionnels répondent aux points précis du cahier des charges, et sont donc requis par le client.
Ils ne sont pas négociables en général, c'est le "besoin primaire" du client. Les besoins non-fonctionnels sont soit des besoins optionnels, soit des besoins/contraintes liés à l'implémentation (contraintes de langage ou de plate-forme, par exemple) et à l'interopérabilité générale (ne pas bouffer toutes les ressources de la machine par exemple).
Ils peuvent être fixés par le client (fonctions optionnelles), ou par le développeur (contraintes d'implémentation).

Utilité des architectures

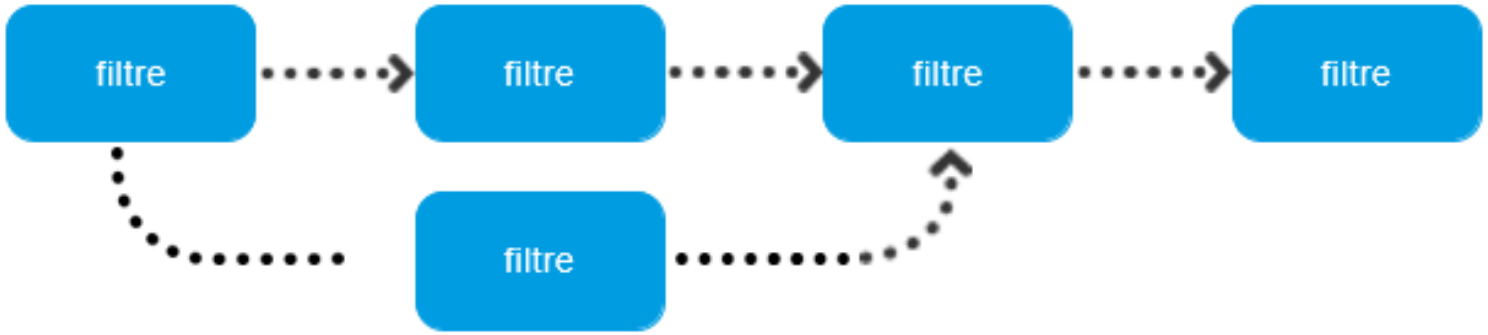
- Utilité d'une architecture logicielle [Garlan 2000]: au moins 06 raisons
  - **Compréhension** : facilite la compréhension des grands systèmes complexes en donnant une vue de haut-niveau de leur structure et de leurs contraintes. Les motivation des choix de conception sont ainsi mis en évidence
  - **Réutilisation** : favorise l'identification des éléments réutilisables, parties de conception, composants, caractéristiques, fonctions ou données communes
  - **Construction** : fournit un plan de haut-niveau du développement et de l'intégration des modules en mettant en évidence les composants, les interactions et les dépendances
  - **Évolution** : met en évidence les points où un système peut être modifié et étendu. La séparation composant/connecteur facilite une implémentation du type « plug-and-play »

Utilité des architectures

- Utilité d'une architecture logicielle [Garlan 2000]:
  - **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** : contribue à la gestion générale du projet en permettant aux différentes personnes impliquées de voir comment les différents morceaux du casse-tête seront agencés. L'identification des dépendance entre composants permet d'identifier où les délais peuvent survenir et leur impact sur la planification générale

Types d’architectures: Architecture en pipeline

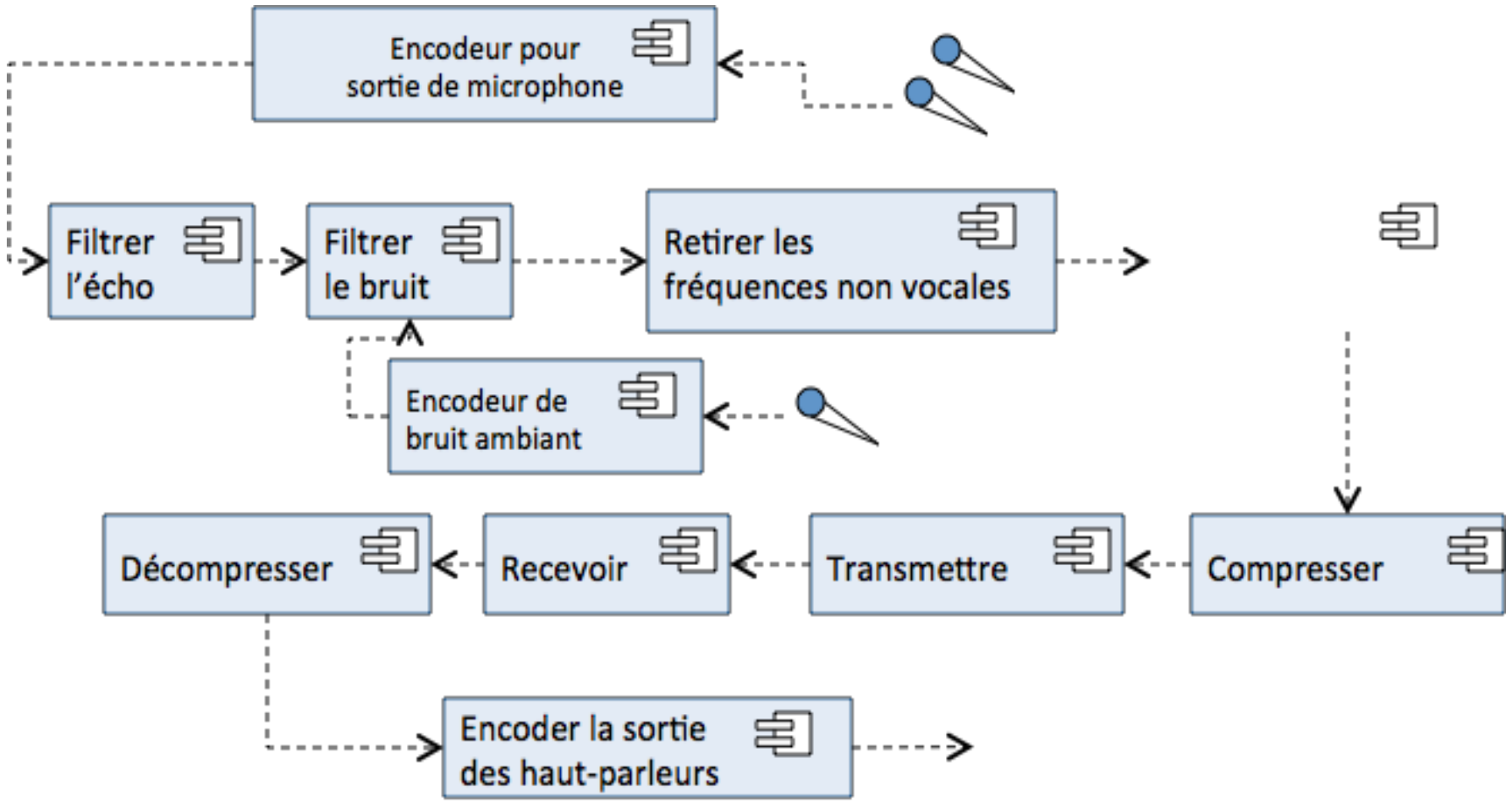
- Un **Pipeline** (ou chaîne de traitement)



Types d’architectures: Architecture en pipeline

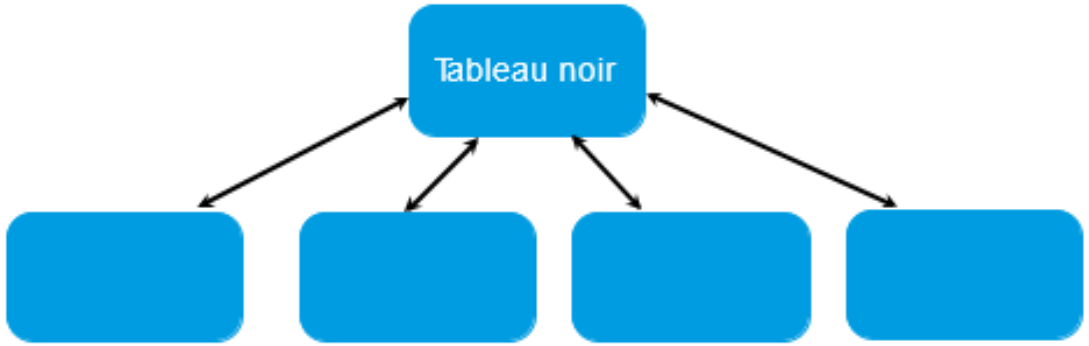
- Sous-systèmes organisés en pipeline de filtres indépendants
  - La sortie d’un filtre correspond à l’entrée d’un autre
- Communication locale (voisins de gauche et de droite)
  - un filtre peut souvent commencer à opérer avant même d’avoir lu tout le flux d’entrée
- Utile pour les traitements en plusieurs étapes
- Exécution concurrente de filtres possible, synchronisation des flux parfois nécessaire

Architecture en pipeline: exemple



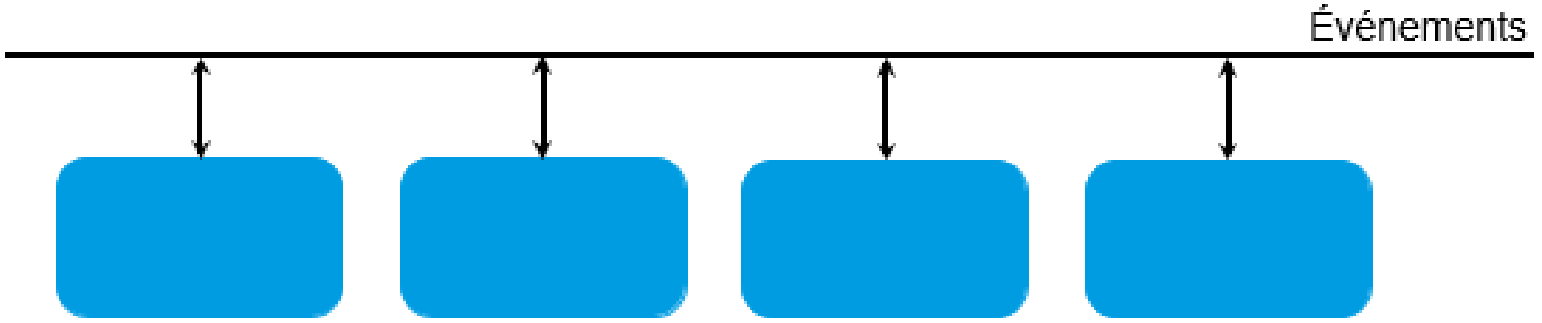
Types d’architectures: Architecture tableau noir

- Tableau noir : médium de communication
- Communication étendue à tous les partenaires
- Un des sous-système est désigné comme le tableau noir
- On ne peut recevoir et transmettre des informations que via le tableau
- Utile lorsqu’on ne connaît pas de solution déterministe



Architecture par événements

- Les sous-systèmes répondent aux événements
- Appropriée pour les logiciels dont les composants doivent interagir avec l'environnement
- Les sous-systèmes s'abonnent pour recevoir les annonces de certains types d'événements

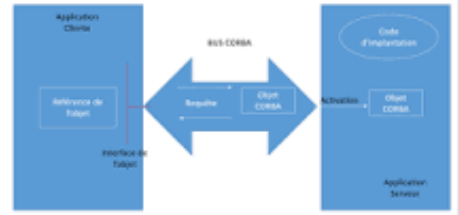


Architecture 2-tiers

- Ce type d'application permet d'utiliser toute la puissance des ordinateurs présents sur le réseau et permet de fournir à l'utilisateur une interface riche, tout en garantissant la cohérence des données qui restent gérées de façon centralisée.
- La gestion des données est prise en charge par un SGBD centralisé sur un serveur dédié. On interroge ce serveur à travers un langage de requête, le plus courant étant SQL.
- Le dialogue entre le client et le serveur se résume donc à l'envoi de requêtes et aux données en réponse.

Architecture 2-tiers

En architecture informatique, un **middleware** (anglicisme) ou intergiciel est un logiciel tiers qui crée un réseau d'échange d'informations entre différentes applications informatiques.



Middleware — Wikipédia
https://fr.wikipedia.org/wiki/Middleware

- On distingue deux parties:
  - Le **client**
  - Le **serveur** qui se contente de répondre aux requêtes du client.
- Le client provoque l'établissement d'une conversation afin d'obtenir des données ou un résultat de la part du serveur.
- Cet échange de messages transite à travers le réseau reliant les deux machines. Il met en œuvre des mécanismes complexes qui sont en général pris en charge par un intergiciel appelé **middleware**.

Architecture 2-tiers

- Le **middleware** est l'ensemble des couches réseau et services logiciels qui permettent le dialogue entre différents composants d'une application répartie. Ce dialogue s'établie sur des protocoles applicatifs communs, défini par l'API du middleware.
- Le **middleware** se contente d'unifier, pour les applications mises en jeu, l'accès et la manipulation de l'ensemble des services disponibles sur le réseau, afin de rendre l'utilisation de ces derniers presque transparente.

Architecture 2-tiers: les limites

- L'architecture client-serveur de première génération présente les inconvénients suivants :
  - on ne peut pas soulager la charge du client qui supporte déjà **tous les traitements applicatifs** ;
  - le poste client est fortement sollicité, il devient de plus en plus complexe et nécessite des mises à jour régulière, ce qui est contraignant ;
  - le client et le serveur sont assez bruyants, ce qui s'adapte mal à des bandes passantes étroites ;
  - ce qui cantonne ce type d'application à des réseaux locaux ;
  - il est difficile de modifier l'architecture initiale, les applications supportent donc mal les fortes montées en charge ;
  - la relation forte et étroite entre le programme du client et l'organisation de la partie serveur complique les évolutions de cette dernière ;
  - ce type d'architecture est grandement rigidifié par les coûts et la complexité de maintenance.

Biblio et Webographie

- Len Bass, Paul Clements, and Rick Kazman(2003).Software architecture in practice,2nd edition,Addison-Wesley.
- David Garlan. 2000. Software architecture: a roadmap. In Proceedings of the Conference on The Future of Software Engineering (ICSE '00). ACM, New York, NY, USA, 91-101. DOI=http://dx.doi.org/10.1145/336512.336537
- Cours de Yann-Gaël Guéhéneuc au Département de génie informatique et logiciel de l'École Polytechnique de Montréal http://www.yann-gael.gueheneuc.net/Work/Teaching/
- https://apiumhub.com/tech-blog-barcelona/benefits-of-software-architecture/
- Https://www.supinfo.com/articles/single/5676-qu-est-ce-que-architecture-microservices
- https://techbeacon.com/top-5-software-architecture-patterns-how-make-right-choice