Architecture Logicielle

Séquence 3: Architecture des Applications Mobiles

INSA BADJI

Basé sur le cours de Dr. Lilia SFAXI

UVS
UNIVERSITE VIRTUELLE DU SENEGAL
foo nekk foofu la

Introduction

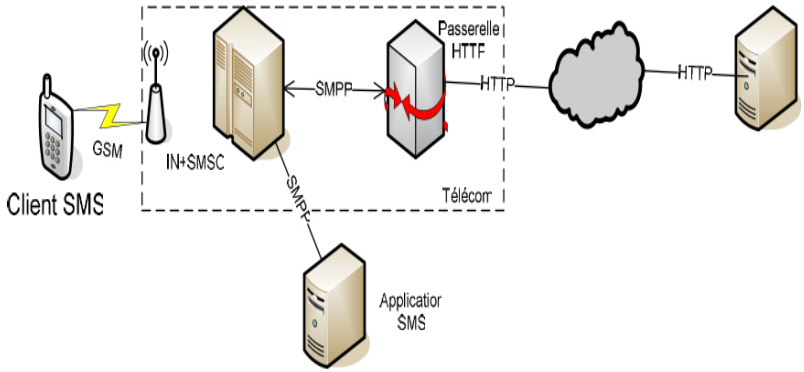
- La conception d'une architecture d'application mobile est la première étape et la plus cruciale. Il joue un rôle très important dans la croissance du marché d'une application.
- Nous définissons l’architecture logicielle comme un ensemble de constituants (ou composants) qui forment un système informatique. Bien qu’ils puissent fonctionner manière autonome, ces éléments ne sont pas indépendants, mais ils interagissent pour travailler ensemble.
- Un style architectural est un modèle d’organisation récurrent dont la compréhension est partagée dans un domaine. Il définit ainsi une abstraction des caractéristiques communes à un ensemble d’architectures c’est à dire une famille d’applications.
- En considérant les limitations de performance qui sont inhérents à la structure matérielle des appareils mobiles, il est clair qu’ils sont incapables d’effectuer certains traitements voire d’afficher certains contenus volumineux.

Introduction

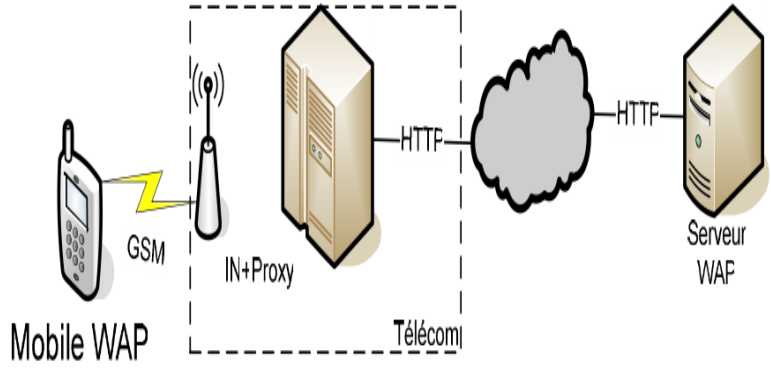
- D’où l’idée de répartition des charges de travail entre différentes entités communicantes en veillant non pas à équilibrer les charges, mais en affectant à chaque entité participante le volume de traitement conforme à sa configuration.
- Cette stratégie induit donc le modèle architectural de base client/serveur avec la possibilité de structurer la partie serveur en plusieurs couches selon la logique du métier.
- Le style architectural en couches est donc approprié pour les applications mobiles. Ce style présente l’organisation hiérarchique du système où chaque couche fournit des services aux couches adjacentes supérieures de même qu’il en consomme ceux des couches adjacentes inférieures.
- Selon les caractéristiques de l’appareil mobile dépendra la nature du client laquelle influencera la complexité infrastructurelle et logicielle du serveur.

Exemples Architectures mobile

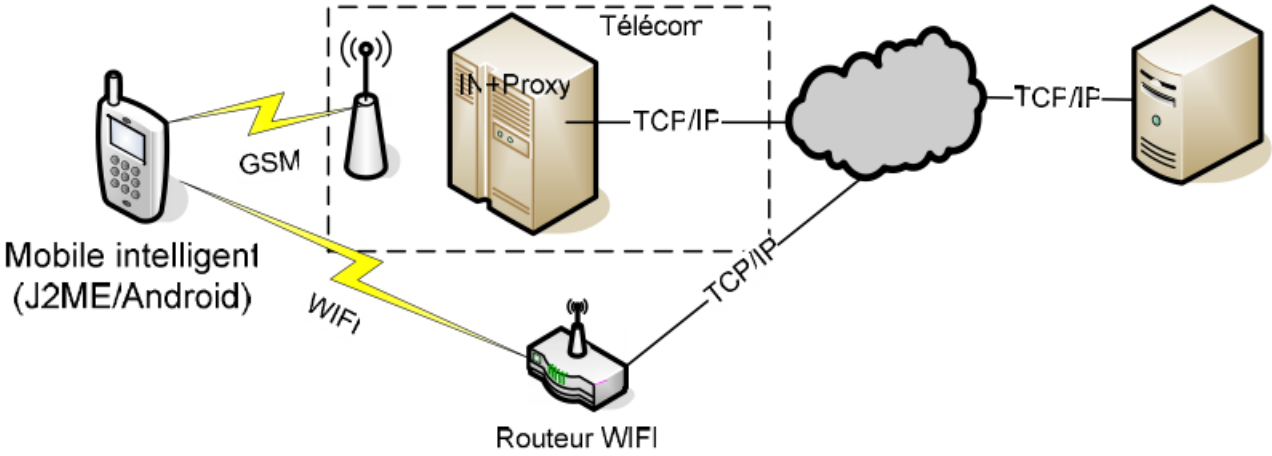
SMS CLIENT



WAP CLIENT



Client smart intelligent



Architectures des Applications Mobiles

- Usuellement: Application multi-couches
  - Interface Utilisateur
  - Couche métier
  - Couche données
- Choix possible entre:
  - Client léger web-based
    - Couches métier et client probablement localisées sur le serveur
  - Client lourd et riche
    - Couches métier et client probablement sur l'appareil

Principes de Conception

- Les principes clefs permettent de concevoir une application mobile qui correspond aux « meilleures pratiques », minimise les coûts et besoins de maintenance, et favorise l'utilisabilité et extensibilité:
  - Séparation des préoccupations
  - Principe de responsabilité unique
  - « Don't Repeat Yourself » (DRY)
  - Ne pas commencer par une conception évoluée
  - Privilégier la composition sur l'héritage

Séparation des Préoccupations

- « Separation of Concerns »
  - Diviser votre application en parties distinctes dont les fonctionnalités se chevauchent le moins possible
  - Avantages
    - Optimisation d'un module ou fonctionnalité indépendamment des autres
    - Si un module est défaillant, il n'impactera pas les autres
    - Rend l'application plus facile à comprendre et concevoir
    - Facilite la gestion des systèmes complexes indépendants

Principe de Responsabilité Unique

- « Single Responsibility Principle »
  - Chaque composant ou module doit être responsable uniquement d'une fonctionnalité ou caractéristique
  - Avantages
    - Rend les composants plus cohésifs (unis, rassemblés)
    - Rend l'optimisation des composants plus facile si une caractéristique ou fonctionnalité a changé

Pas de Répétition

- « Don’t Repeat Yourself (DRY) »
  - Il ne faut pas dupliquer une fonctionnalité dans une application sur plusieurs modules
  - Avantages
    - Faciliter l’implémentation des changements
    - Augmenter la clarté
    - Renforcer la consistance du code

Ne pas Commencer par une Conception Évoluée

- «Avoid doing a big design upfront »
  - Si vos besoins ne sont pas clairs, ou s'il y'a des risques d'évolution de la conception dans le temps, éviter de fournir trop d'effort pour la conception au début
  - Faites en sorte que votre conception progresse avec votre application

Privilégier la Composition à l'Héritage

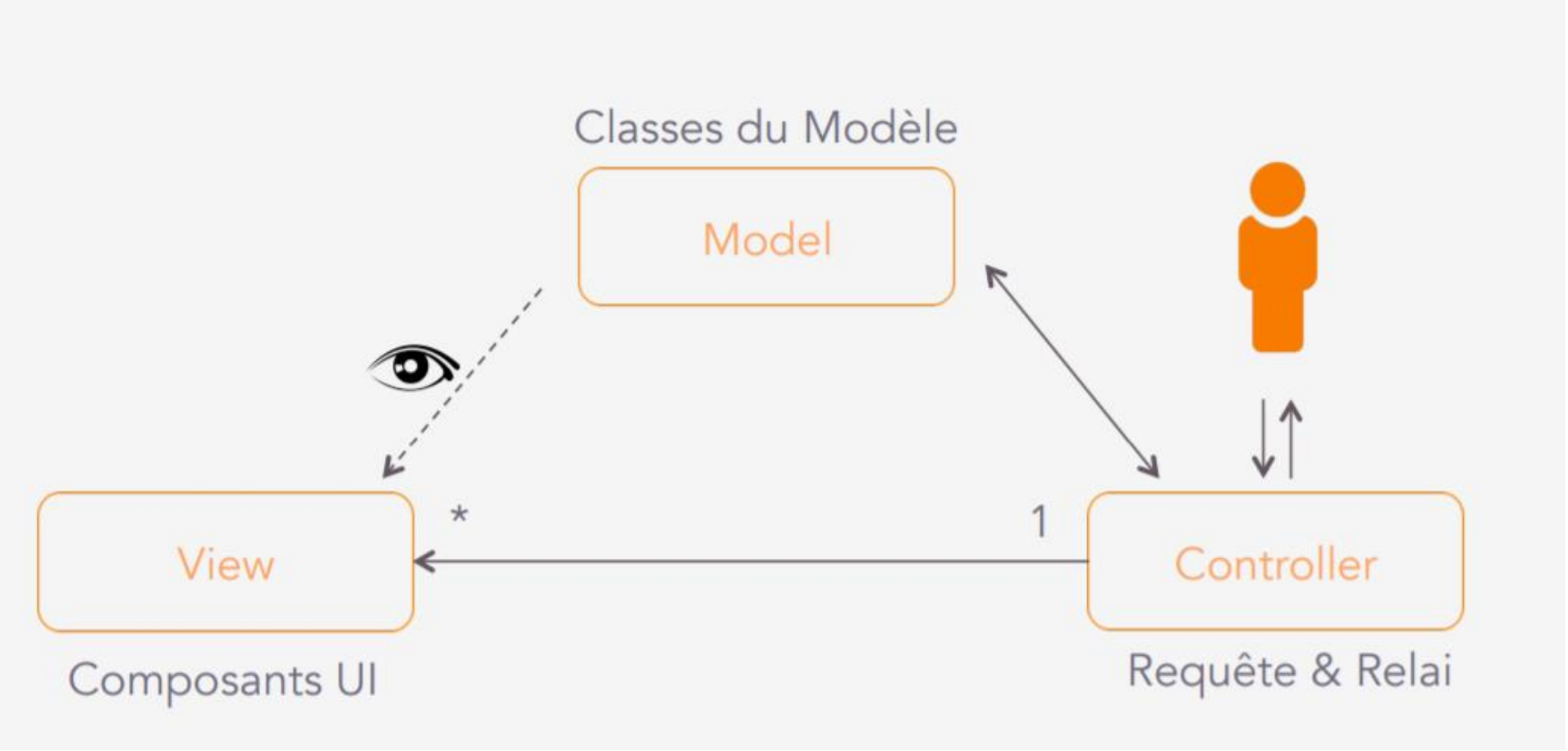
- «Composition over Inheritance »
  - Utiliser si possible la composition à la place de l'héritage quand vous voudrez réutiliser une fonctionnalité
  - Avantages
    - L'héritage augmente la dépendance entre les classes parentes et filles
    - Limitation de la réutilisation des classes filles
    - Réduire la hiérarchie de l'héritage, qui peut s'avérer pénible

- Grouper les types de composants différents dans des couches logiques
- Faire en sorte que les design patterns d'une couche soient consistants
  - Ne pas utiliser de design patterns incompatibles dans une même couche
- Ne pas mixer plusieurs types de composants dans la même couche logique
- Déterminer le type de couches que vous voulez renforcer
- Utiliser l'abstraction pour implémenter le couplage faible entre les couches
- Éviter de surcharger les fonctionnalités d'un composant
- Comprendre comment est-ce que les composants communiquent les uns avec les autres
- Garder des formats de données consistantes dans une couche
- Séparer le plus possible le code non fonctionnel (sécurité, logging…) du code métier
- Être consistant dans les conventions de nommage
- Établir des standards pour la gestion des exception

Les Patrons de Conception MV*

- « Les patrons de conception sont des solutions réutilisables pour des problèmes récurrents »
  - Réutilisabilité
    - Le modèle n'est pas couplé avec sa représentation, et peut donc être facilement réutilisé dans d'autres projets
  - Testabilité
    - Tester les couche indépendamment les unes des autres les rend plus faciles à gérer et à corriger
  - Maintenabilité
    - Il est plus facile de modifier une partie de l'application sans impacter les autres modules
  - Compréhensibilité
    - Le code est plus compréhensible et plus lisible

Model-View-Controller



Architecture des applications

- La conception de l'architecture des applications est un processus qui doit être exécuté dans un flux défini. Le flux comprend essentiellement trois couches différentes. À savoir:
- Couche de présentation
  - Cette couche comprend des composants d'interface utilisateur et des composants de processus d'interface utilisateur (vues et contrôleurs).
  - Étant à ce niveau, l'équipe doit définir la façon dont l'application mobile se présentera devant les utilisateurs finaux.
  - Sur cette couche, la décision des entités et de leur emplacement est principalement focalisée.
  - Cependant, simultanément, l'équipe décide également d'autres aspects comme le thème, la taille de la police, etc.

Architecture des applications

- Couche métier
  - Comme son nom l'indique, la couche se concentre sur le front commercial. Dans un langage simple, il se concentre sur la façon dont les entreprises seront présentées devant les utilisateurs finaux.
  - Cela inclut les workflows, les composants métier et les entités sous le capot de deux sous-couches nommées couche de modèle de service et de domaine.
  - Alors que la couche de service se concentre sur la définition d'un ensemble commun de fonctions d'application qui seront disponibles pour les clients et les utilisateurs finaux, la couche de modèle de domaine représente l'expertise et les connaissances liées au domaine de problème spécifique.
- Couche de données
  - À cette troisième étape, les facteurs liés aux données sont gardés à l'esprit. Cela inclut les composants d'accès aux données, les assistants/utilitaires de données et les agents de service.

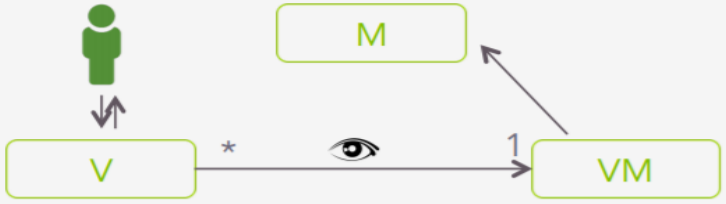
MVC : Couches

- Model
  - Données de l’application
  - Méthodes manipulant ces données
  - Stockage et extraction de la BD
- View
  - Représentation visuelle du modèle
  - Gère les interactions utilisateur
- Contrôleur
  - Accès aux données à partir du modèle
  - Affichage des données dans les vues
  - Intermédiaire entre plusieurs vues et modèles
  - Observe les changements du modèle et les transmet à la vue

MVC : Flux

- Les entrées utilisateur sont interceptées par le contrôleur
- Un contrôleur peut faire appel à plusieurs vues (erreur, succès…)
- Une vue n’a pas de visibilité sur son contrôleur
- Le contrôleur passe le modèle à la vue
- Le modèle a en général peu (ou pas) de méthodes (comportement)
- La vue fait référence au modèle, mais pas le contraire
- La vue « observe » les changements du modèle, et s’y adapte
- Le contrôleur fait référence au modèle, le remplit et le passe à la vue
- La vue n’a pas de visibilité sur le contrôleur, mais fait référence et s’attend à un modèle particulier

MVVM : Model – View – ViewModel



- L’ entrée utilisateur commence par la vue et peut provoquer l’exécution d’un comportement ViewModel
- La vue et le modèle ne communiquent jamais
- ViewModel est un modèle fortement typé de la vue, qui est une réflexion exacte ou une abstraction de cette vue
- ViewModel et Vue sont toujours synchronisés
- Le modèle n’a aucune idée que le VM ou la vue existent
- n VM n’a pas besoin de référencer la vue
- La vue est reliée à des propriétés du VM
- La vue n’a aucune idée sur l’existence du modèle
- Le VM et le modèle n’ont pas de référence ou visibilité sur la vue
- Le modèle n’a aucune idée de l’existence de la vue et du VM

MVP : Model – View – Presenter

- L’ entrée utilisateur commence par la vue, pas par le **Presenter**
- La vue invoque les commandes du **Presenter**, et le **Presenter** modifie la vue
- La vue et le modèle ne communiquent jamais
- Le **Presenter** est une couche d’abstraction de la vue
- Chaque vue a un **Presenter** qui lui est associé (1-to-1)
- Le **Presenter** a besoin d’une référence vers la vue
- La vue a également une référence vers son **Presenter**
- C’est le **Presenter** qui va charger la vue à partir du modèle, pas le modèle

Biblio et Webographie

- Structuration des Applications Mobiles : Expérimentation dans le cadre de la dématérialisation duprocessus d'inscription dans une Université David Garlan. 2000.
- Microsoft Patterns and Practices, « Mobile Application Architecture Guide: Application Architecture Pocket Guide », 2008
- Amir Jalilifard, « A belly dance in the holy lands of MVC, MVP and MVVMPatterns via Javascript », Code Project, 24 février 2015, url: http://www.codeproject.com/Articles/844789/A-Belly-Dance-in-HolyLands-of-MVC-MVP-and-MVVM-Pa
- Ketan Thakkar, « Difference between MVC vs. MVP vs. MVVM », 11 septembre 2014, url:http://blogs.k10world.com/technology/difference-between-mvc-vsmvp-vs-mvvm/