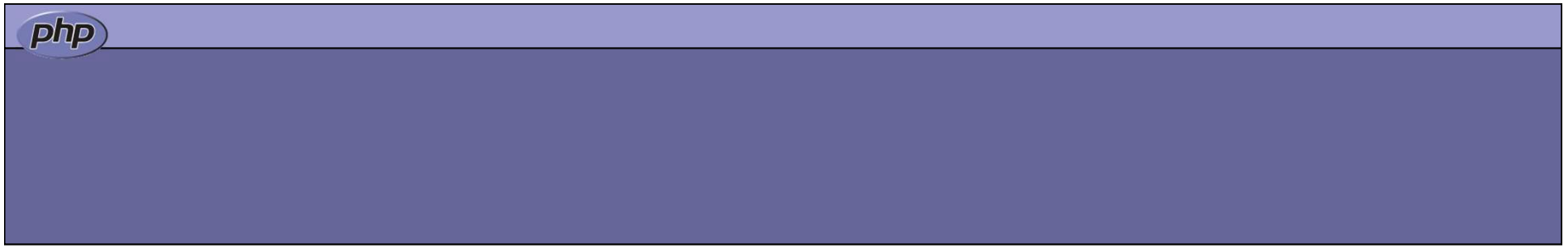


Design Pattern MVC

En PHP

MVC

- Le Modèle-Vue-Contrôleur organise l'interface Homme-machine d'une application logicielle en
  - un modèle (objet métier, modèle de données)
  - une vue (présentation, interface utilisateur)
  - un contrôleur (logique de contrôle, gestion des événements, traitement)

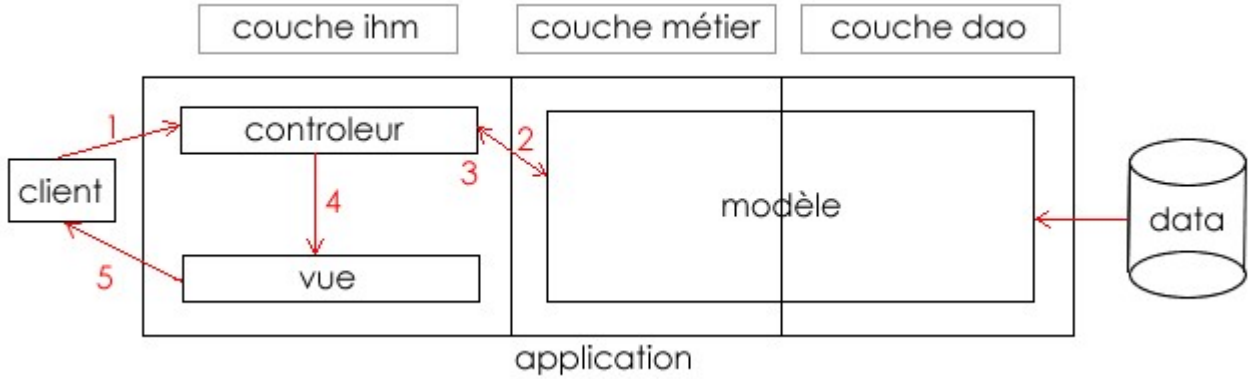
MVC - objectifs

- séparation entre
  - les données
  - la présentation
  - les traitements
- Meilleur réutilisabilité du code
  - Catalyse les temps de développement
  - Facilite la maintenance

MVC - web

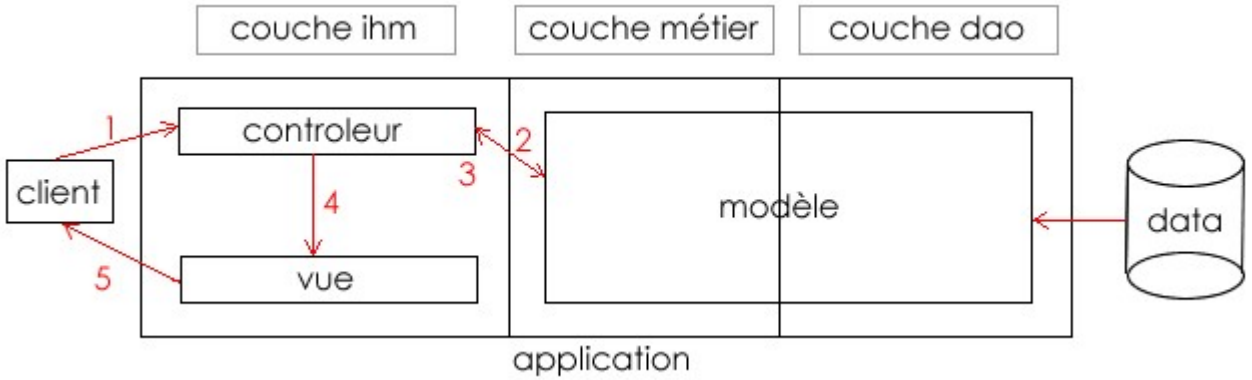
- La logique applicative est constituée des scripts réalisant les demandes de l'utilisateur (contrôleur), des classes métiers et des classes d'accès aux données (modèle).
- L'interface utilisateur sera gérée la plupart du temps par un navigateur web, elle peut éventuellement être un autre site web communiquant via des webservices.
  - Dans tous les cas tout va se passer via HTTP.
- La source de données sera une base de données dans toute la suite mais pourrait parfaitement être un serveur LDAP, un webservice, des fichiers texte ou encore un fichier XML.

MVC en image



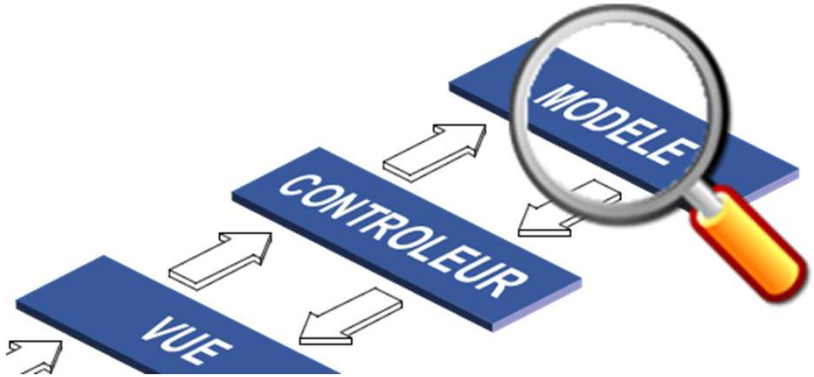
•couche ihm: c'est l'interface utilisateur encore appelé interface homme machine
•couche métier : c'est le coeur de l'application où réside les objets traités par l'application
•couche dao : couche d'accès aux données (data access object). Cette couche permet une indépendance de la logique métier et du stockage des données associées

MVC en action



1. le client fait une demande au contrôleur. Ce contrôleur voit passer toutes les demandes des clients
2. le contrôleur doit traiter la demande. Pour ce faire, il peut avoir besoin de la couche métier, cette dernière peut éventuellement accéder aux données (via la couche dao)
3. le contrôleur effectue les traitements nécessaires sur / avec les objets renvoyés par la couche métier
4. le contrôleur sélectionne et nourrit la (les) vue(s) pour présenter les résultats du traitement qui vient d'être effectuée
5. la vue est enfin envoyée au client par le controleur

Le modèle



- décrit et contient les données manipulées par l'application, ainsi que des traitements sur ces données
- les résultats renvoyés par le modèle sont dénués de toute présentation
- le modèle contient toute la logique métier de l'application

Le modèle - DAL

- Data Access Layer
  - Rend la gestion des données indépendante du stockage (BDD, XML, LDAP, etc ..)
    - Dans le contexte PHP/BDD cette couche permet une indépendance du SGBD utilisé (PDO, PEAR::MDB2, Creole)
    - Doit permettre les actions CRUD (Create Read Update Delete). La DAL va donc mettre à disposition des méthodes réalisant ces opérations. En Php c'est l'objet PDO

Le modèle - DAO

- Data Access Object
  - Le Dao a pour but de transformer les données contenues dans une bases de données en objets et inversement
- Correspondance bijective (SGBD / paradigme objet)
  - une table (appelée aussi relation) à une liste d'objets
  - une ligne d'une table (appelée aussi tuple) à un objet
  - un champs de base de données à un attribut d'objet
  - une valeur d'un champs à une valeur d'attribut d'un objet
- Si un objet est modifié, sa ligne associée dans la table l'est aussi

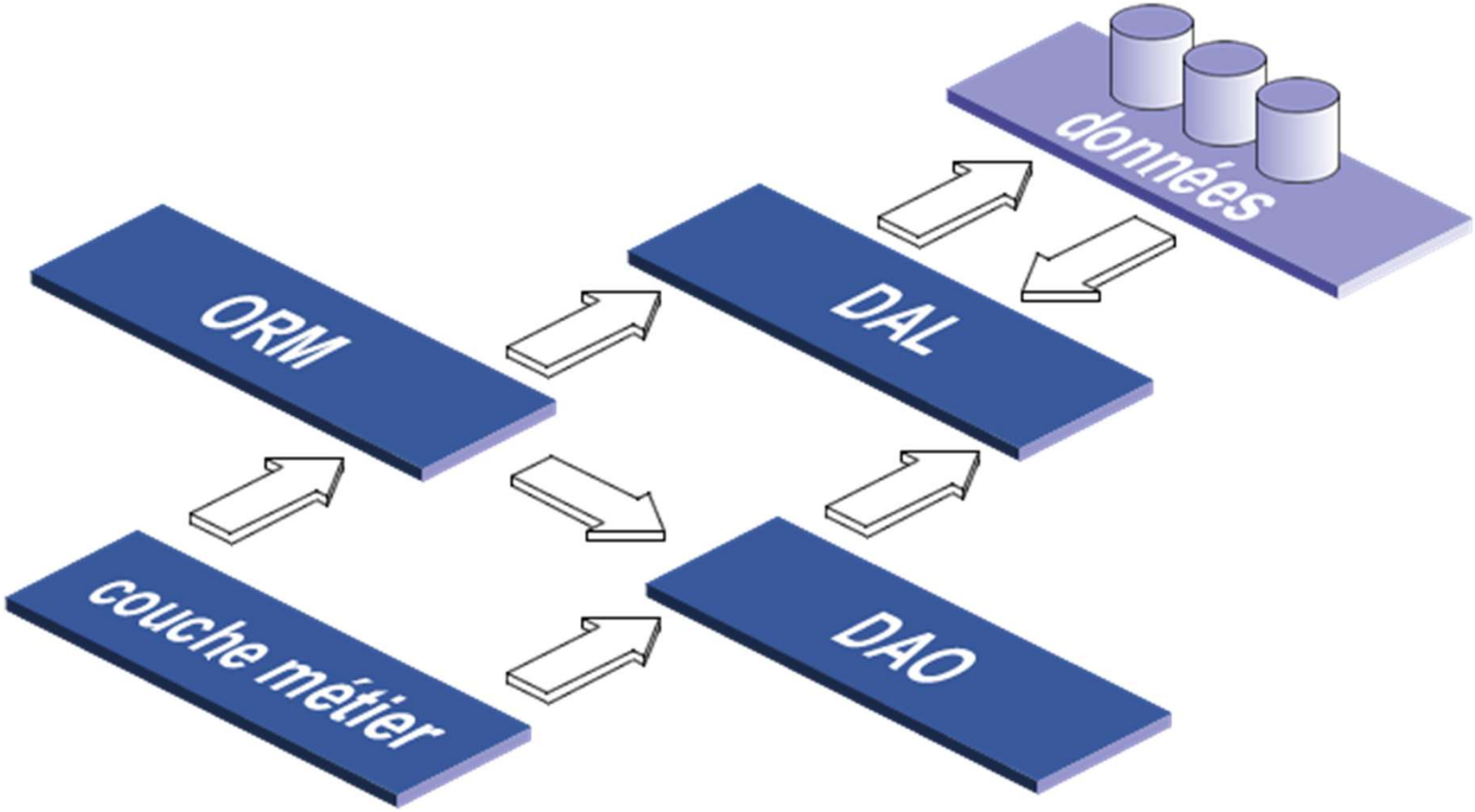
Le modèle - ORM

- Object-relation Mapping a pour but de transformer les relations entre les tables d'une base de données en relations entre objets et inversement
- Une table ou vue de la BDD correspond à une classe
  - Typiquement une clé étrangère est matérialisé par un objet, présenté comme un attribut de l'objet correspondant à la table contenant la clé étrangère

Le modèle - la couche métier

- Idéalement la couche métier utilise une ORM et un DAO, qui utilisent eux-mêmes la DAL
  - Dans ce cas là les méthodes CRUD (Create Read Update Delete) ainsi que les dépendances entre objets sont gérées par les couches supérieures
    - La couche métier ne contient que les traitements métiers (propre à l'objet)

Le modèle



Plus de détail sur : https://blog.mazenod.fr/design-pattern-mvc-zoom-sur-la-couche-modele-dal-dao-orm-crud.html

vue

- interface avec laquelle l'utilisateur interagit
- présentation des résultats renvoyés par la couche modèle, après le traitement du contrôleur
- recevoir toutes les actions de l'utilisateur (clic de souris, sélection d'une entrée, boutons, soumission de formulaire …)
  - événements envoyés au contrôleur
- La vue n'effectue aucun traitement

contrôleur

- Détermine l'action à réaliser (analyse de la requête HTTP
- gestion des événements de synchronisation entre modèle et vue
- Si une action nécessite un changement des données
  - demande la modification des données au modèle
  - avertit ensuite la vue que les données ont changé pour qu'elle se mette à jour
- Ne fait qu'appeler des méthodes
  - n'effectue aucun traitement directement
  - ne modifie aucune donnée directement

Les actions

- partie du code du contrôleur
- correspond à un évènement utilisateur précis
- associée à des vues précises
- Matérialisée par une ou plusieurs variable HTTP (passées en GET)

Contrôleur et actions

- Le contrôleur analyse la requête HTTP (et notamment les variables)
  - pour déterminer l'action à exécuter
- Le contrôleur doit déterminer les vues associées (les templates HTML, mais aussi les css et js à embarquer)
- Multitude d'action = contrôleur lourd
  - Modèle MVC2
    - Un front controler redirige vers des contrôleurs spécialisés (méthode dispatch)
    - Les contrôleurs spécialisés exécutent effectivement l'action

Séquence typique de traitement MVC2

1. la requête est analysée par le front controler
2. Le contrôleur spécialisé adéquat est appelé avec la requête en paramètre
3. Le contrôleur spécialisé demande au(x) modèle(s) approprié(s) d'effectuer les traitements
4. le contrôleur spécialisé sélectionne la (les) vue(s) adaptée(s)
5. le contrôleur spécialisé remplit la vue adaptée avec le résultats des traitements
6. le contrôleur spécialisé renvoie la vue adaptée.