Développement Web
Le modèle MVC

Jean-Michel Richer
jean-michel.richer@univ-angers.fr
http://www.info.univ-angers.fr/pub/richer

FACULTÉ
DES SCIENCES
Unité de formation
et de recherche

24 janvier 2011

Objectif

Objectif du cours

Se familiariser avec l'architecture MVC

- comment est organisée l'architecture MVC
- comment peut on l'appliquer pour le Web

Plan

Le modèle MVC

Le modèle MVC

L'architecture MVC

Du modèle 1 au MVC

une plus grande maîtrise du développement web requiert le passage du modèle 1 au modèle MVC

Le modèle 1

Tous les traitements sont réalisés dans la même page

L'architecture MVC

Definition (Model View Controller)

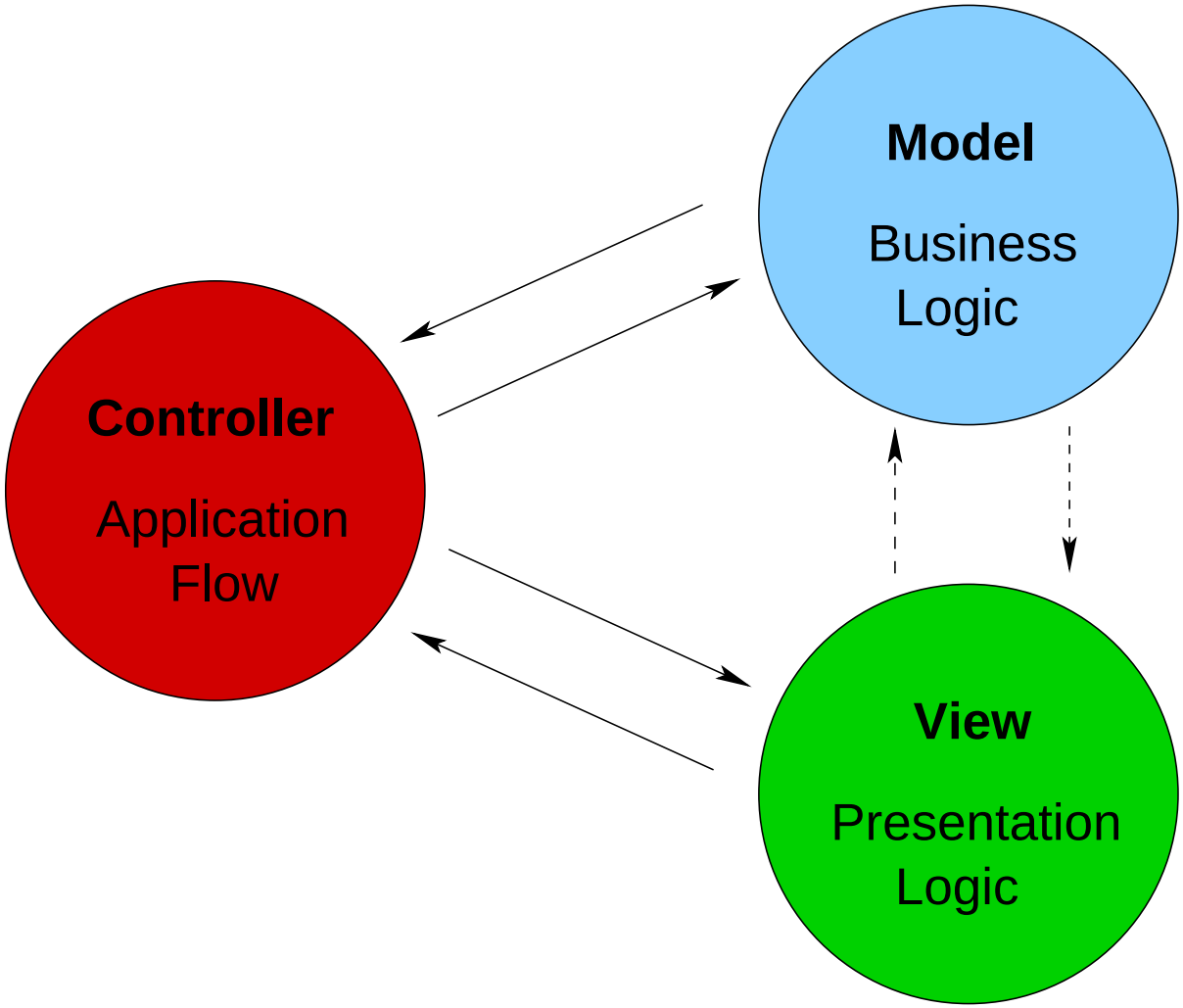
1. design pattern (orienté objet)
2. élaboré par Trygve Reenskaug en 1979 au Xerox PARC
3. dédié initialement au langage Smalltalk
4. formalisé par Steve Burbeck
5. repose sur une séparation des concepts (couches)
6. différentes interprétations et implantations

Patron de conception

Definition (Design Pattern - Wikipedia)

- un **patron de conception** est un concept destiné à résoudre les problèmes récurrents suivant le paradigme objet
- décrivent des solutions standard pour répondre à des problèmes d'architecture et de conception des logiciels
- on encourage les développeurs à les appliquer même si c'est parfois contraignant

MVC schématique



La couche *modèle*

Definition (Model)

- représente le *fond* (sujet d'étude)
- s'intéresse à la représentation des données de la **couche métier** (*business logic*), i.e. données spécifiques à l'application

La couche *vue*

Definition (View)

- concerne la *forme* (représentation)
- elle interagit avec le modèle
- la vue concerne principalement la représentation des données du modèle à l'écran (ou sur tout autre périphérique de sortie)
- il peut donc exister plusieurs vues

La couche *contrôleur*

Definition (Controller)

Le contrôleur gère les interactions avec l'utilisateur :

- s'occupe de la réécriture des URL
- détermine quels traitements doivent être réalisés

Avantages

Avantages de l'architecture MVC

- adaptée aux applications graphiques (non web)
- séparation des tâches :
  - diminution de la complexité lors de la conception
  - répartition suivant les développeurs
  - maintenance et modification facilitées

Inconvénients

Inconvénients de l'architecture MVC

- moins bien adaptée aux applications web
- séparation des tâches :
  - augmentation de la complexité lors de l'implantation
  - éventuel cloisonement des développeurs

Problèmes liés au MVC pour le web

Application Web

- prendre en compte la persistence des données
- prendre en compte la distribution des objets (cas des applications distribuées)
- la couche *contrôleur* doit-il prendre en compte les traitements ?

Persistence

Definition (Persistence)

la couche persistence traite de l'échange de l'information avec les bases de données.

Persistence et MVC

- certains considèrent qu'elle fait partie du modèle
- il est préférable de l'extraire du modèle afin de réaliser de l'ORM (*Object Relational Mapping*) (ex. Hibernate pour Java)

Distribution

Definition (Distributivité)

certaines applications web sont dites distribuées, i.e. les classes du modèle ne sont pas toutes stockées sur la même machine.

Distribution et MVC

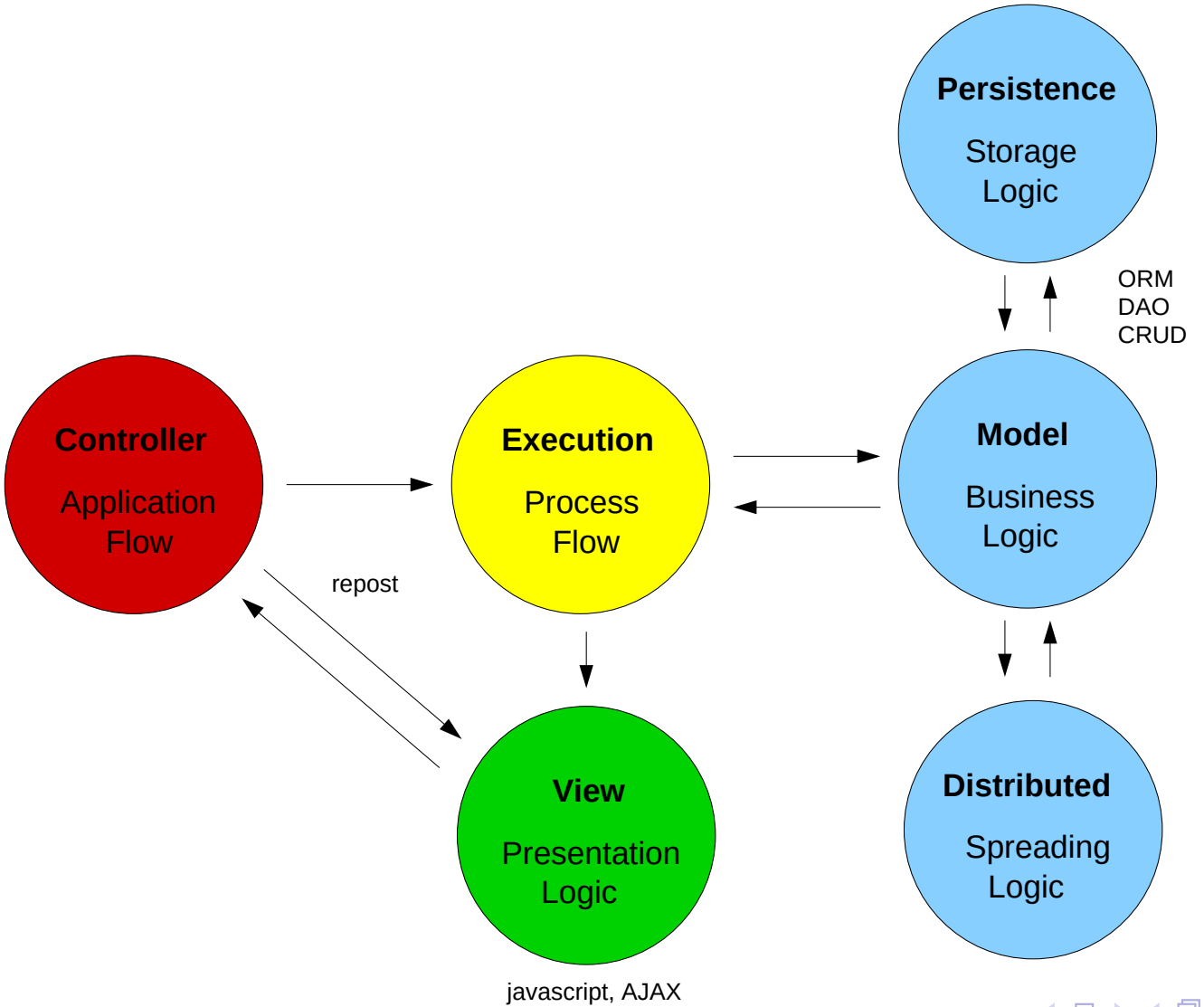
il est pnécessaire de prendre ce facteur en compte lors de la conception et l'implantation

Contrôleur et traitements

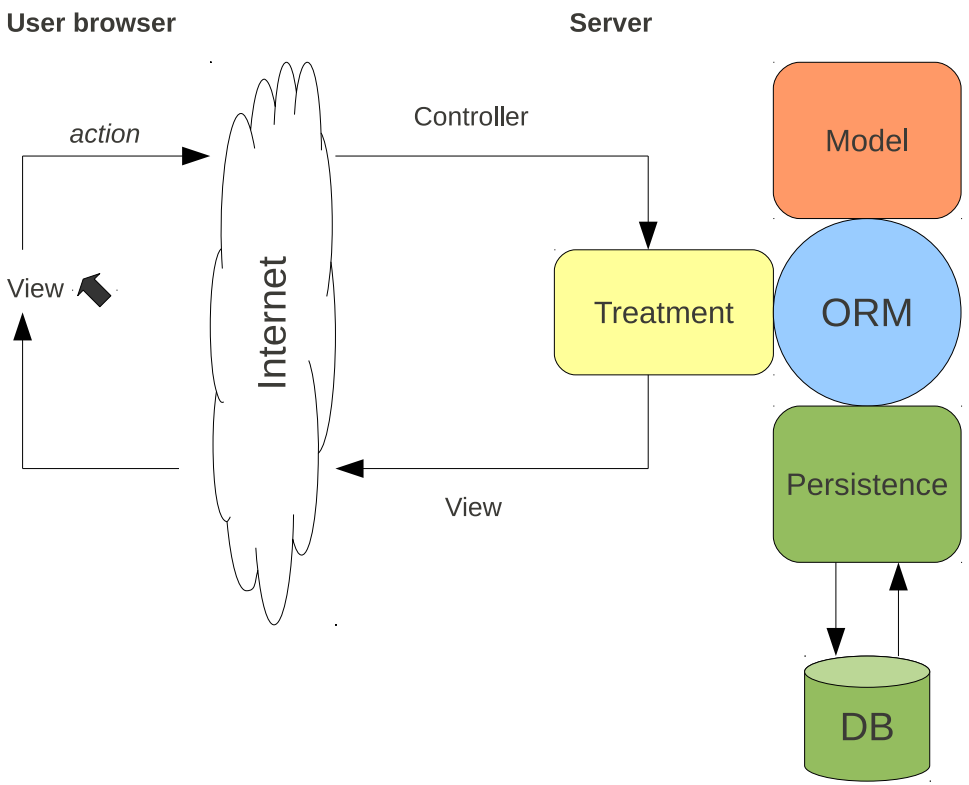
Le contrôleur doit-il réaliser les traitements ?

- c'est le cas pour les applications standard (non web)
- on peut ajouter une nouvelle partie à l'architecture MVC : Exécution qui se charge de réaliser les traitements
- l'objectif est de simplifier l'implantation
- nouvelle architecture : MPD-V-CE

MPD-V-CE schématique



L'architecture M(PD)-V-C(E) en action



Implantation

Implantation

Répertoires

Elaboration du système de fichiers

Un crée un répertoire par couche de l'architecture

- `model`
- `view`
- `controller`
- `persistence`

Implantation

Le modèle

Le modèle

Model

On crée une classe mère `CoreObject` dont hériteront tous les autres objets. Cette classe a pour but de réaliser les tâches de base :

- obtention de la valeur d'un attribut
- fixer la valeur d'un attribut
- transformation de format (String, JSON)

La classe CoreObject

prototype Object

```
1  class CoreObject {
2    // default empty constructor
3    public function __construct();
4    // return attributes names and values
5    private function getProperties();
6    // return value of a gicen attribute name
7    public function __get($attribtueName);
8    // transform into string
9    public function __toString();
10   // JSON representation
11   public function exportAsJSON();
12 }
```

Autres classes

Autres classes

Elles sont construites à la manière des JavaBean :

- constructeur sans argument
- on implante les getters et setters

Implantation

La Vue

La Vue

Architecture de la vue

la partie mise en forme est composée de plusieurs classes pour le rendu XHTML :

- `Document` qui est chargée de la structure de la page XHTML et de la gestion de l'authentification des utilisateurs
- `Forms` (et les classes dérivées) qui gèrent les formulaires

La classe Document

La classe Document

La page est décomposée en 4 parties :

- l'entête (*header*)
  - header logo
  - header menu : accessible à tous
- le menu contextuel : différent en fonction du niveau d'accès de l'utilisateur
- le sujet : relatif à la page
- le bas de page (*footer*)

La classe `Document`

La classe `Document`

Elle comporte deux attributs liés à la session pour gérer l'authentification des utilisateurs :

- `userId` (*int*) : identifiant de l'utilisateur (lorsqu'il est connecté)
- `userLevel` (*int*) : le niveau d'accès de l'utilisateur connecté

La classe Document

prototype Page

```
1  class Document {
2    private $userId;
3    private $userLevel;
4
5    // default constructor : calls htmlHeader
6    public function __construct($css="",$dojoRequire="",$meta="");
7    // start the <body> part of the page
8    public function begin($level=0);
9    // display the <head> part of the page
10   private function htmlHeader($css="",$dojoRequire="",$meta="");
11   // header section
12   protected function header();
13   // contextual menu
14   public function menu();
15   // end <body> and display footer
16   public function end() {
17   // begin a subject section
18   public function beginSection($title);
19   // end a subject section
20   public function endSection();
21 }
```

utilisation de la classe Page

la classe Page en action

```
1 session_start();
2 require_once('config.php');
3 require_once('view/document.php');
4 $document=new Document();
5 if (!$document->begin(0)) die();
6 $document->beginSection("Welcome!");
7 bla bla bla
8 $document->endSection();
9 $document->end();
```

La classe FormField

La classe FormField

- classe de base pour la construction des champs des formulaires
- dérivée en :
  - FormFieldText, FormFieldEmail, FormFieldPassword
  - FormFieldTextArea
  - FormFieldSelect, FormFieldRadio, FormFieldCheckbox

La classe FormField

prototype FormField

```php
define('FORM_FIELD_TEXT',1);
define('FORM_FIELD_PASSWORD',2);
define('FORM_FIELD_TEXTAREA',3);
define('FORM_FIELD_SELECT',4);
abstract class FormField {
  //name of field
  protected $name;
  //label that will be displayed
  protected $label;
  // type of field (see constants defined above)
  protected $type;
  // boolean required, if needs to be filled
  protected $required;
  // message to display under the field for information
  protected $message;
  // value filled by user
  protected $value;
  // constructor
  function __construct($name,$label,$type,$required,$message);
  /**
   * check if field is filled or value is chosen
   * @return null if ok, the field otherwise
   */
  function check();
}
```

La classe FormFieldText

La classe FormFieldText

- gère les champs de type `text`
- dérivée en :
  - `FormFieldPassword` pour les mots de passe
  - `FormFieldEmail` pour les emails

La classe FormFieldText

prototype FormFieldText

```
1  class FormFieldText extends FormField {
2    // size of text
3    protected $size;
4    // maxlength
5    protected $maxlength;
6    // constructor
7    function __construct($name,$label,$required,$message,$size,$maxlength) {
8      parent::__construct($name,$label,FORM_FIELD_TEXT,$required,$message);
9      $this->size=$size;
10     $this->maxlength=$maxlength;
11   }
12 }
```

La classe FormFieldSelect

La classe FormFieldSelect

- gère les champs de type `select`
- dérivée en :
  - `FormFieldPassword` pour les mots de passe
  - `FormFieldEmail` pour les emails

La classe FormFieldSelect

prototype FormFieldSelect

```
1  class FormFieldSelect extends FormField {
2    protected $options;
3    function __construct($name,$label,$required,$message,$options) {
4      parent::__construct($name,$label,FORM_FIELD_SELECT,$required,$message);
5      $this->options=$options;
6    }
7    function get_options() {
8      return $this->options;
9    }
10   function get_option_value($key) {
11     return $this->options[$key];
12   }
13   function check() {
14     if ($this->required==true) {
15       if (empty($this->value)) return $this;
16       if ($this->value==-32768) return $this;
17     }
18     return null;
19   }
20 }
```

Utilisation de la classe FormFieldSelect

utilisation de FormFieldSelect

```php
$options=array(1=>"rouge", 2=>"vert", 3=>"bleu");
$fields=new FormFieldSelect("couleur","",true,
  "couleur préférée",$options);
```

La classe FormFieldSet

La classe FormFieldSet

- classe abstraite qui regroupe plusieurs champs corrélés
- dérivée en :
  - FormFieldSetInputs : pour les champs à saisir
  - FormFieldSetButtons : pour les boutons

La classe FormFieldSet

prototype FormFieldSet

```
1  abstract class FormFieldSet {
2     // array of fields that compose the fieldset
3     protected $fields;
4     // constructor
5     function __construct() {
6        $this->fields=array();
7     }
8     // add new field
9     function add_field($field);
10    // generate part of the form
11    function generate();
12    // set fields values from given array
13    function set($array);
14    // set fields values from $_POST
15    function get_fields_from_post();
16    // check if all fields of the set are valid
17    // return null if true, or the field if not valid
18    function check();
19 }
```

La classe Form

La classe Form

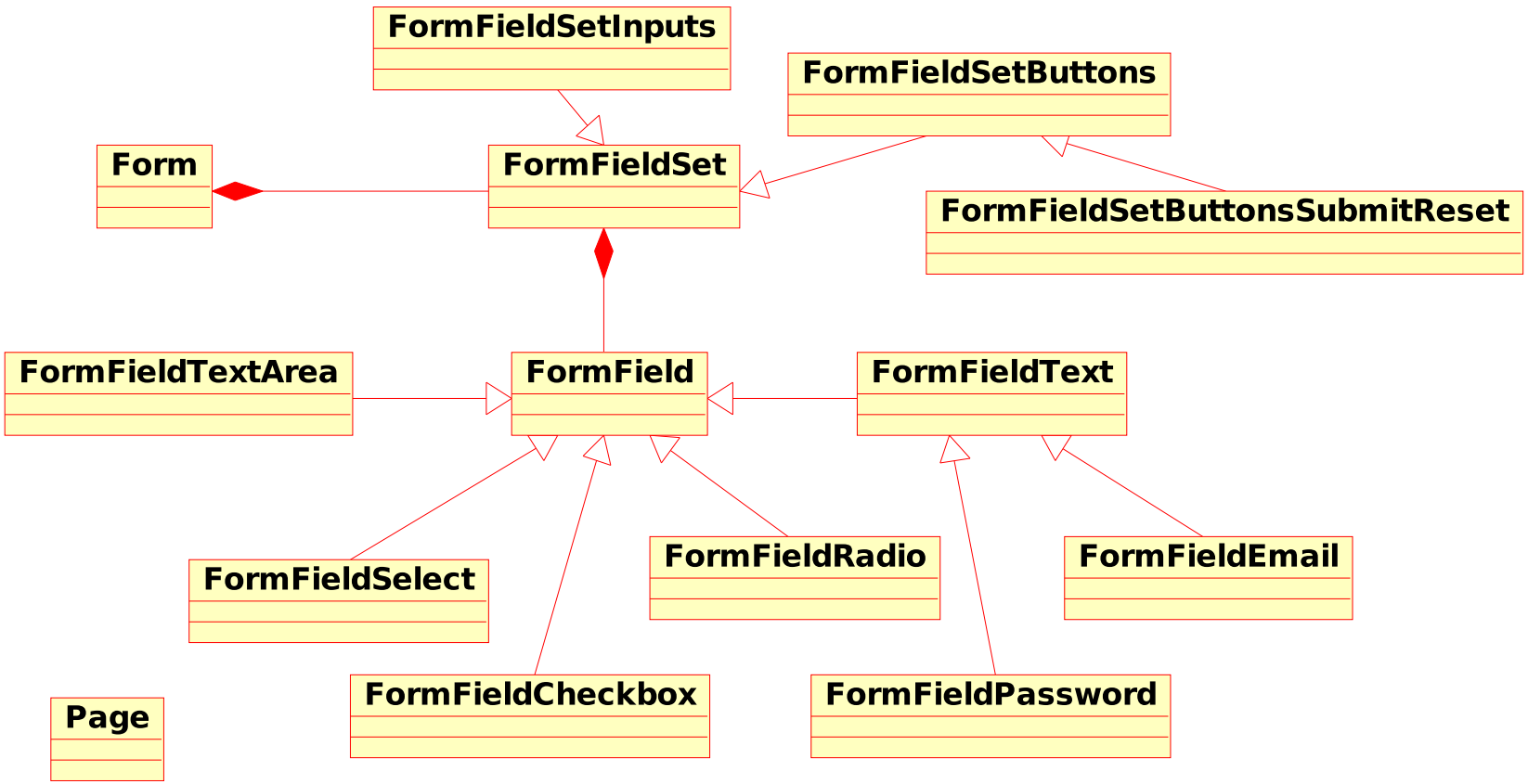
- classe chargée de gérer le formulaire, dont :
  - initialisation des champs
  - vérification que les champs *requis* sont saisis

La classe Form

prototype Form

```php
class Form {
   // name of Form
   protected $name;
   // action executed on submit
   protected $action;
   // method (POST or GET)
   protected $method;
   // array of fieldsets
   protected $fieldsets;
   // javascript
   protected $js;
   /** constructor
     * @param $name name of the form
     * @param $action script called on submit
     * @param $method post or get
     * @param $js javascript */
   function __construct($name,$action,$method="POST",$js="");
   // add a new fieldset
   function add_fieldset($fieldset);
   /** generate form
     * @param $highlight highlight required fields which are not set (true
     * or false) */
   function generate($highlight=false);
   /** check that all fields are set properly
     *return null if ok, or the first field that is not set properly */
   function check();
}
```

La vue en UML


FormFieldSetInputs
FormFieldSetButtons
Form
FormFieldSet
FormFieldSetButtonsSubmitReset
FormFieldTextArea
FormField
FormFieldText
FormFieldSelect
FormFieldRadio
FormFieldEmail
FormFieldCheckbox
FormFieldPassword
Page

Implantation

Le Contrôleur

Le Contrôleur

Architecture du contrôleur

- `Controller` classe de base
- on crée un contrôleur par classe du modèle

un ou plusieurs contrôleurs ?

- on pourrait créer un seul contrôleur
- plusieurs : conception facilitée, réutilisabilité

La classe `Controller`

La classe `Controller`

classe abstraite comportant les attributs suivants :

- `action` (*string*) : action à exécuter
- `destination` (*string*) : page de destination après exécution du traitement
- `repost_data` (*array*) : données éventuelles à reposter

La classe Controller

prototype Controller

```php
abstract class Controller {
   // action to process
   protected $action;
   // php destination page
   protected $destination;
   // data to repost (if needed)
   protected $repost_data;
   // constructor
   public function __construct() {
      $this->action=""; $this->destination="index.php";
      $this->repost_data=array();
   }
   // process
   public function process() {
      $this->action=$_GET['control']; $this->execute();
      $url='http:///'.$_SERVER['HTTP_HOST'].'/pub/richer/bibliotheque/'.
         $this->destination;
      if (count($this->repost_data)==0) header('Location: '.$url);
      else $this->repost($url);
   }
   // execute algorithms
   protected function execute();
   // repost data to $this->destination
   protected function repost($url);

}
```

Utilisation de Controller

utilisation de Controller

```
<li><a href='controller/uneclasse_controller.php?action=create'>ajouter</a></li>

<form name='user_login' action='controller/uneclasse_controller.php?action=login'>
```

Implantation

La Persistence

La couche Persistance (Persistence layer)

Definition (Persistence)

chargée de gérer les objets persistants i.e. gère les interactions avec une base de données

Technologies associées

- ORM (Object Relational Mapping)
- DAO (Data Access Object)
- CRUD (Create Retrieve Update Delete)

L'ORM

Definition (ORM - Object Relational Mapping)

technique dédiée à la mise en relation entre les attributs de la classe avec les champs des tables de la base de données

Le CRUD

Definition (CRUD - Create Retrieve Update Delete)

Opérations de base à implanter pour gérer l'échange d'information entre objets et tables de la base de données

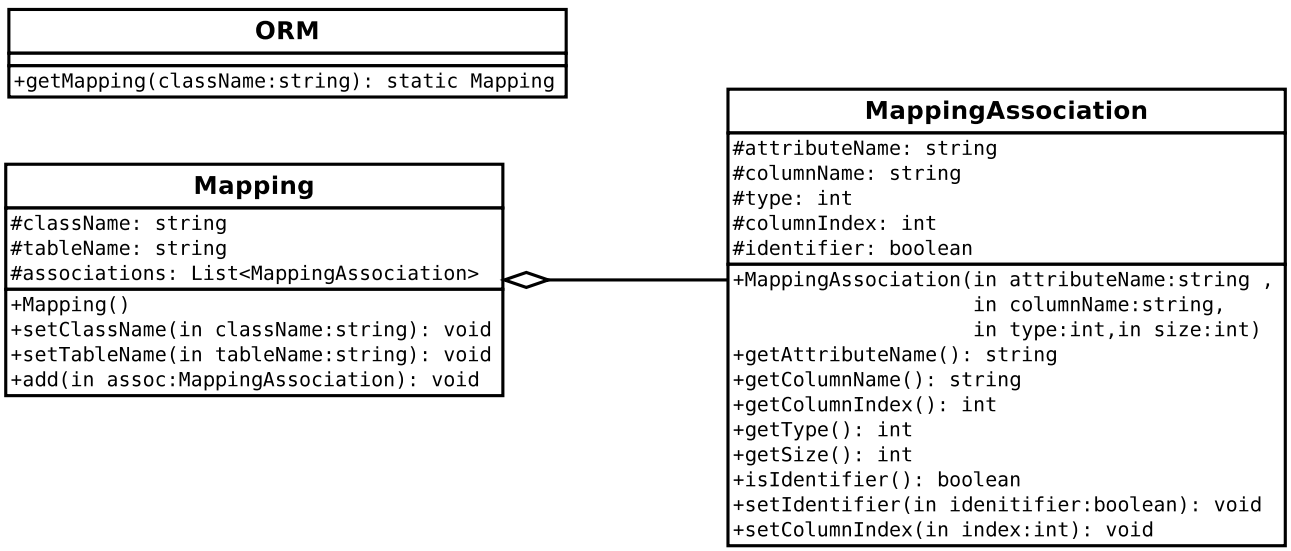
le CRUD peut être vu comme une interface

Le DAO

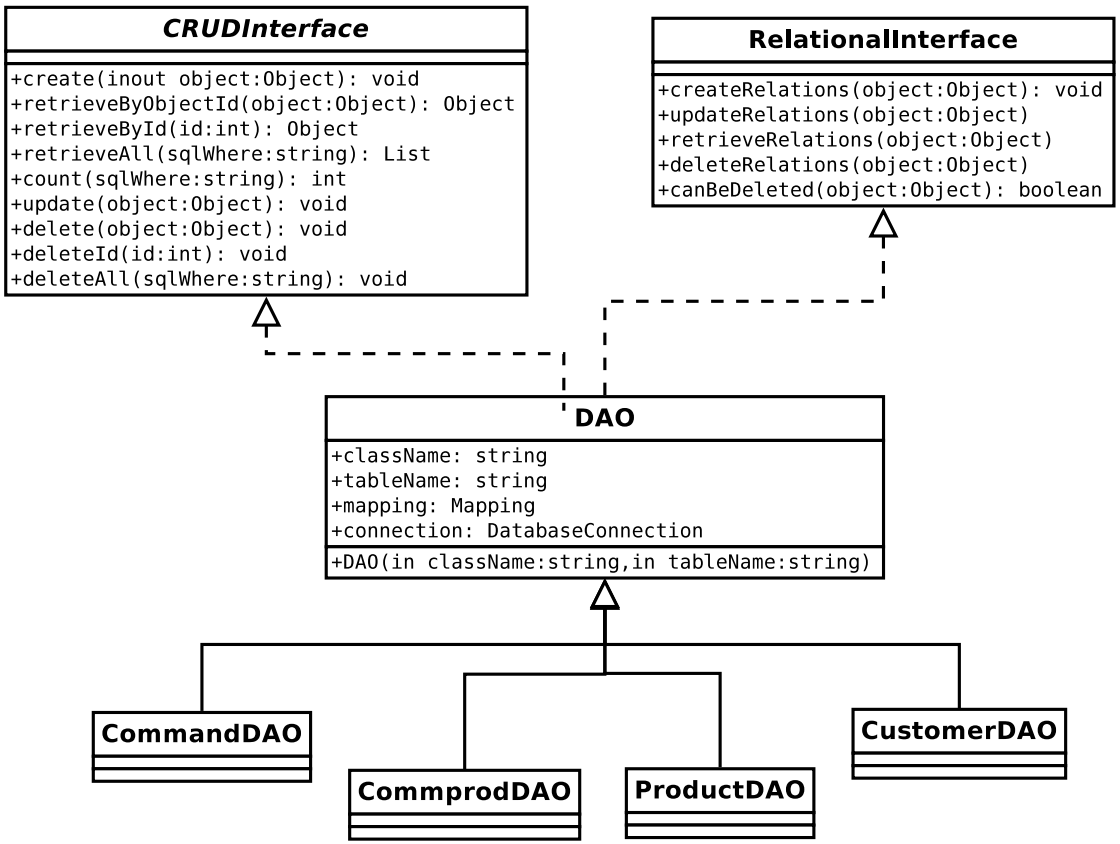
Definition (DAO - Data Access Object)

implante le CRUD en gérant l'accès à la base de données

La persistance en UML



La persistance en UML



Fin

Fin