# Secteur Tertiaire Informatique

## Filière étude - développement

*Activité Développer des composants d'interface*

*Rapport Composants*

Romain Heller
Afpa 2014

# Table des matières

# 1 Présentation Air de Java

## 1.1 Analyse du cahier des charges

Une association de groupes folkloriques souhaite mettre en place une application permettant de gérer les groupes qui lui sont affiliés ainsi que les rencontres culturelles où les groupes peuvent participer.

Vous trouverez le Cahier des charges en Annexe cpp2-1.

## 1.2 Analyse de l'existant

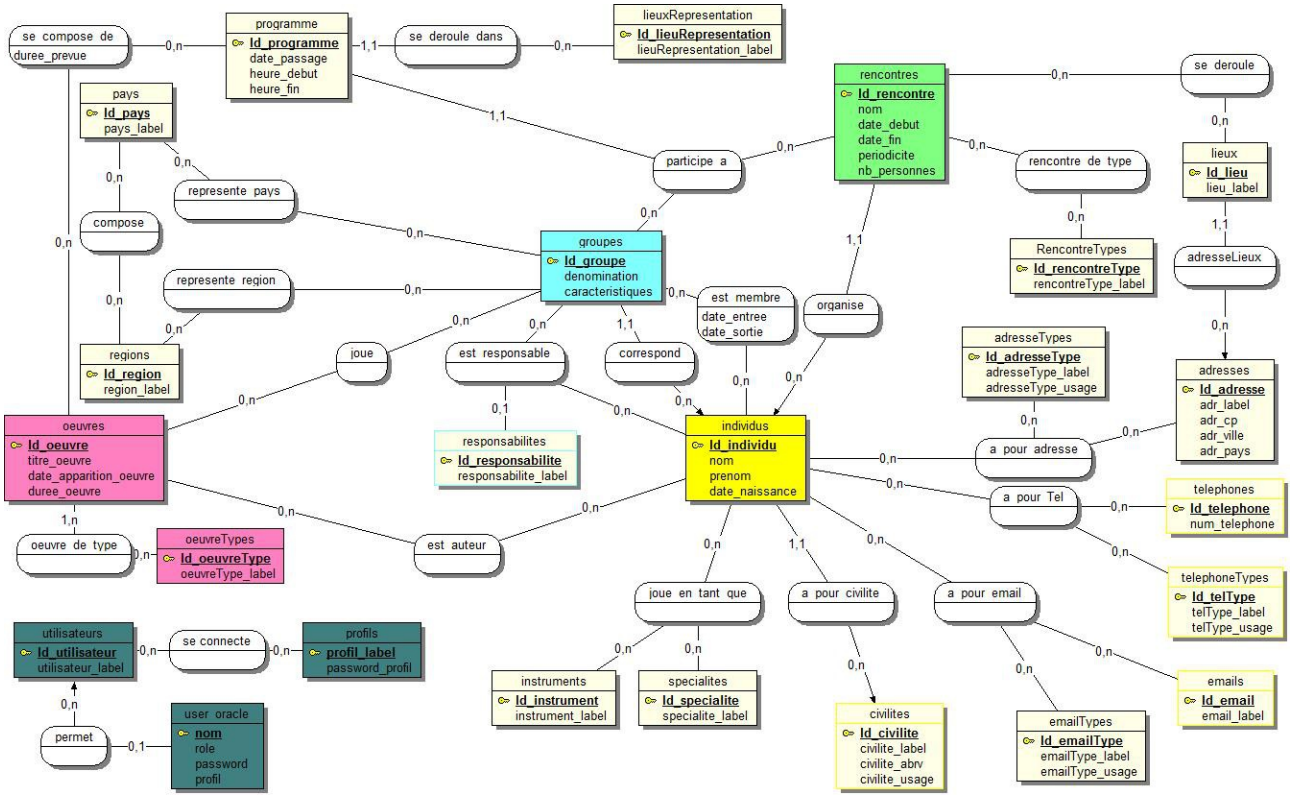
Une base de donnée a été développée et déployée sur un serveur Oracle XE version 11r2 pour Windows, et on désire garder Oracle comme serveur.

Deux applications seront développées :

- Un client lourd pour permettre à un gestionnaire de manipuler les données de la base.
- Un site web pour diffuser les données au grand public, et permettre aux membres de mettre à jour leurs propres données.

## 1.3 Analyse du domaine

Une Analyse a déjà été faite lors de la conception de la Base de Données.

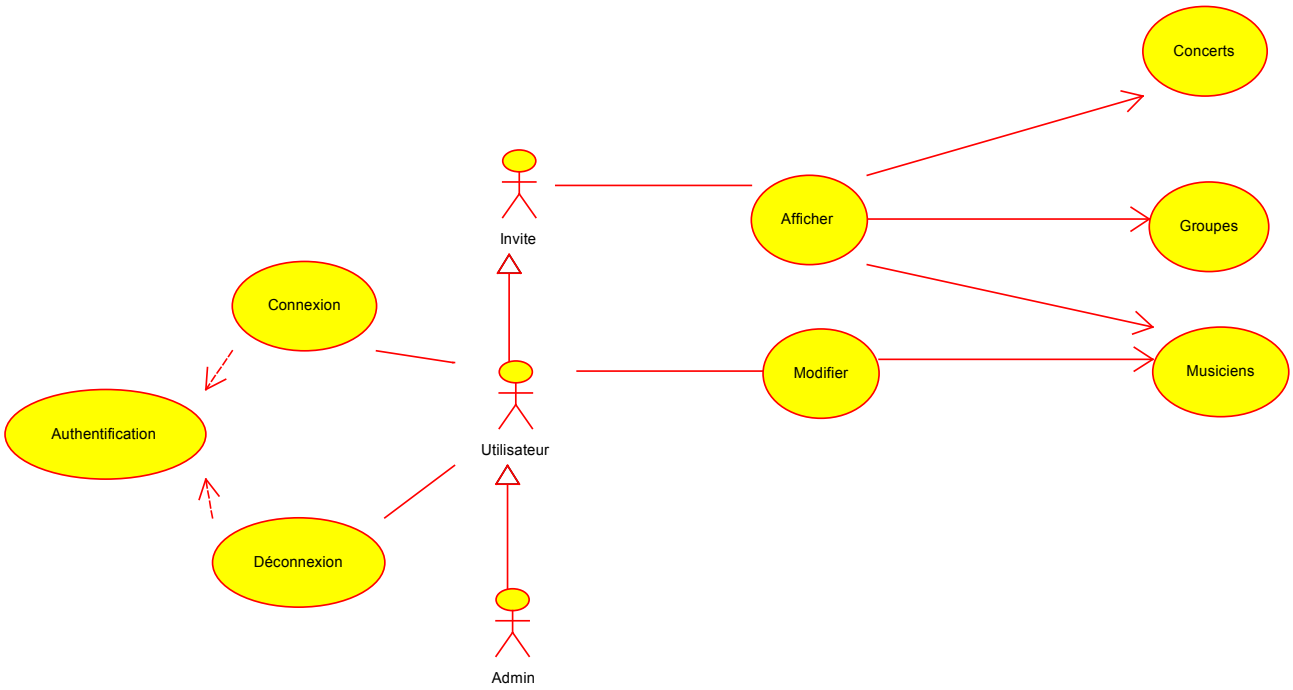


# 2 Modélisation du domaine

L'application web est un site déployé en PHP qui présentera les données de la base Oracle.

Il respectera l'architecture MVC et utilisera une couche de persistance pour permettre d'ajouter et de modifier les données de la Base.

## 2.1 Cas d'utilisations




## 2.2 Maquettage des vues

Les pages Web sont présentés selon une maquette générale définissant :

- un en-tête contenant le logo et le titre de l'association
- un pied de page avec les dispositions légales
- un menu de navigation permettant l'accès aux différentes rubriques
- un module de connexion pour l'accès au site.


## 2.3 Le plan de navigation

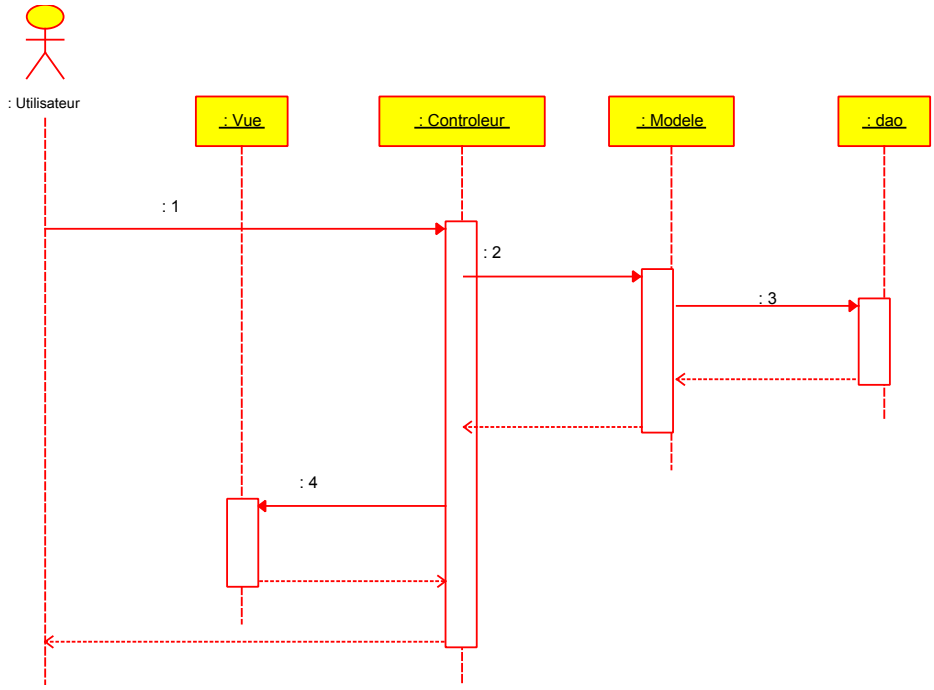
Le Plan de Navigation a été simplifié à l'extrême :

- le menu Groupes affiche une liste des Groupes
- le menu Musiciens affiche une liste des Musiciens (Individus jouant d'un Instrument)
- le menu Concerts affiche une liste des Rencontres à venir.
- La partie membre permet à un utilisateur de modifier son profil :
  - Si il est Responsable de Groupe, il pourra modifier ce groupe.
  - Si il est Organisateur, il pourra modifier ses rencontres.

## 2.4 Diagramme de Séquence Système

L'application utilise une architecture Modèle-Vue-Contrôleur orienté Web :

1. L'Utilisateur (ici le Navigateur Web) envoie une requête au Contrôleur.
2. Le Contrôleur demande les ressources au Modèle.
3. Le Modèle va chercher les données nécessaires via la Couche de Persistance.
4. Le Contrôleur met à jour la vue et la renvoie a l'Utilisateur.




# 3 Conception technique

## 3.1 Choix technologiques

La Base est déjà construite et déployée sur un serveur Oracle.

J'ai choisis de développer le site web en PHP Objet.

Le serveur Web est un Apache avec le module php activé.

Les vues ont été écrites en html5 et communiquent avec le serveur grâce à Jquery.

## 3.2 Outils et plate-formes

La modélisation UML a été réalisée avec Umbrello, suite a des crashes répétés de MagicDraw.
Le format de fichier utilisé par Umbrello est du XMI, ce qui permet aussi de l'utiliser facilement avec d'autres logiciels.

Le codage a été effectué avec NetBeans, pour découvrir un autre IDE que Eclipse.

## 3.3 Choix Frameworks

Il existe plusieurs frameworks php répondant aux spécifications du projet, et permettant en particulier de gérer le MVC et le Dao.

Les plus connus sont sûrement CakePHP, CodeIgniter, Open Web Framework, Postnuke, Symfony, Yii Framework, Zend Framework

Le principal attrait pour ces solutions clés en main, est de ne pas avoir à réinventer la roue.

Mais pour pouvoir choisir le produit le mieux adapté à la situation , il faut commencer par faire le tour des fonctionnalités de chaque produit pour pouvoir les comparer et choisir lequel utiliser, et ensuite, il faut apprendre à le maîtriser : structure interne, syntaxe et grammaires internes, etc.
De plus, il faut savoir ce qui est fait derrière, j'ai donc choisis de partir de zéro, et de tout construire pas à pas.

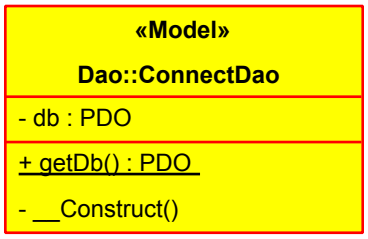
# 4 Conception générale

## 4.1 Patterns utilisés

Plusieurs patrons ont été utilisés, certains lors de la phase de conception, d'autres ont été découverts rétrospectivement, après avoir passé du temps à vouloir résoudre des problèmes d'articulations entre composants.

### 4.1.1 Singleton (Création)

L'accès à la *Base de données* se fait en suivant le patron *Singleton* pour limiter le nombre de connexions ouvertes simultanément.




La classe *ConnectDao* permet de récupérer un connecteur à la Base de Donnée.
Ce connecteur sera unique, et (re)créé qu'en cas de nécessité.

Exemple en Php (extraits) :

```
class ConnectDao {
    /** @var \PDO */
    private $db = null;

    private function __construct() { // Pattern Singleton
    }
    private function __clone() { // Pattern Singleton
    }
    protected function getDb() {
        if ($this->db == null) {
            try {
                $this->db = new \PDO('dsn', 'username', 'password');
            } catch (Exception $ex) {
                throw new Exception('DB connection error: ' . $ex->getMessage());
            }
        }
        return $this->db;
    }
}
```

Un appel à la base devra donc se faire de manière statique.

Exemple depuis AccessDao.php :

```
$this->db = ConnectDao::getDb()
```

### 4.1.2 Facade (Structure)

Tout les appels à l'application passent par un point d'entrée unique : la page index.php de la racine du site sert de contrôleur frontal.

Cela permet d'isoler toutes les fonctionnalités nécessaires au navigateur pour afficher le site, sans qu'il ait a se préoccuper de savoir où aller chercher les informations.

L'utilisation d'un connecteur unique à la Base de données (*ConnectDao* utilise un *Singleton* de la classe *PDO*) peut aussi être vu comme une *Façade* au sous-système *Base de Données*

Les contrôleurs gérant la Connexion au site, ou les entités Groupes, Individus, etc. fournissent diverses fonctionnalités, dont certaines sont inutiles, voire interdites au Navigateur.
Le Contrôleur Frontal utilise les implémentations des sous-contôleurs pour présenter certaines de ces fonctionnalités, présentant une version simplifiée du système.
Le Client (Navigateur), utilise les ressources présentées par la Façade, et n'est donc pas dépendant des sous-systèmes.

### 4.1.3 Patron de Méthode (Comportement)

Les classes abstraites permettent de définir un squelette et un contrat pour chaque classe étendant la classe mère. Elles permettent également de factoriser du code qui, définit dans la classe mère, n'aura pas besoin d'être définit dans les classes filles.

Exemple avec les Classes entité qui héritent de AModel :

```
14: abstract class AModel {
27:     /**
28:      * Get a XML representation of the object.
29:      * @return \DOMNode
30:      */
31:     final public function __toXmlNode() {
32:         /** @var \DOMDocument */
33:         $doc = \DOMDocument::loadXML('<node>old content</node>');
34:         /** @var \DOMNode */
35:         $node = $doc->getElementsByTagName('node')->item(0);
36:         $node->appendChild(new \DOMText($this->__toXmlString()));
37:         return $doc;
38:     }
39:
40:     abstract public function __toXmlString();
41:
42:     abstract public function __toString();
43: }
```

Les classes filles héritent des méthodes `__toXml()` et `__toXmlNode()`, et doivent redéfinir les méthodes `__toXmlString()` et `__toString()` :

Exemple avec la classe Groupe qui représente l'Entité *Groupes* de la *Base de Données* :

```
 9: namespace Air2Java\Model;
10: require_once __DIR__ . '/AModel.php';
11:
12: /**
13:  * Model class representing one Groupe item.
15:  * @category Model
16:  * @package Groupes
17:  */
18: final class Groupe extends AModel {
19:
20:     /** @var int */
21:     private $_id;
22:
23:     /** @var string */
24:     private $_denomination;
25:
26:     /** @var string */
27:     private $_caracteristiques;
28:
29:     /** @var Individu */
30:     private $_responsable;
31:
32:     /** @var \DateTime */
33:     private $_createdOn;
34:
176:     /**
177:      * Get a XML representation of the object.
178:      * @return string
179:      */
180:     public function __toXmlString() {
181:         return "<groupe><id>" . $this->getId() . "</id>"
182:                 . "<nom>" . $this->getNom() . "</nom>"
183:                 . "<description>" . $this->getDescription() .
"</description>"
184:                 . "<responsable>" . $this->getResponsable() .
"</responsable>"
185:                 . "</groupe>";
186:     }
187:
188:     /**
189:      * Get a string representation of the object.
190:      * @return string
```

```
191:       */
192:      public function __toString() {
193:          return "Groupe{" . $this->getId() . "," . $this->getNom() . "," .
$this->getDescription() . "," . $this->getResponsable() . "}";
194:      }
195:
196:      /**
197:       * Get a Json representation of the object.
198:       * @return string
199:       */
200:      public function __toJson() {
201:          return "Groupe{" . $this->getId() . "," . $this->getNom() . "," .
$this->getDescription() . "," . $this->getResponsable()->__toJson() . "}";
202:      }
203:
204: }
```

### 4.1.4 Inversion de Contrôle (Comportement)

Les Interfaces Java et Php permettent une *Injection de Dépendances*.

### 4.1.5 Modèle-Vue-Contrôleur (Comportement / Structure)

Le patron Modèle-Vue-Contrôleur (MVC) est une combinaison des patrons Observateur, Stratégie et Composite.

### 4.1.6 Dao

La Couche de Persistance est implémentée par un Objet d'Accès aux Données (Data Access Object), qui fait le lien entre la partie Métier (Classes Modèle du MVC), et la *Base de Donnée*.
Les classes Modèles instancient un objet AccessDao qui fournit un CRUD sur la Base.

A chaque entité est associé une classe Dao particulière, qui s'occupe de ses spécificités.

Des Framework comme *Open Web Framework* ou *PHP Dao* permettent de ne pas avoir à produire toute la couche de persistance.

## 4.2 Les patrons non mis en œuvres

### 4.2.1 Post/Redirect/Get (PRG)

Pour éviter (entre autres) la re-soumission des formulaires.

## 4.3 Spécifications de test

Les tests fonctionnels ont été réalisés au fur et à mesure du développement , le temps étant trop court pour concevoir des tests avec PhpUnit, Nette ou Atoum.

Ce serait une des principales choses à mettre en œuvre dans une prochaine itération de projet.

## 4.4 Documentation ApiGen

La documentation complète de l'application web est disponible dans le répertoire *Doc*.

Elle a été générée grâce à ApiGen en utilisant les marqueurs écrits dans les commentaires du code.

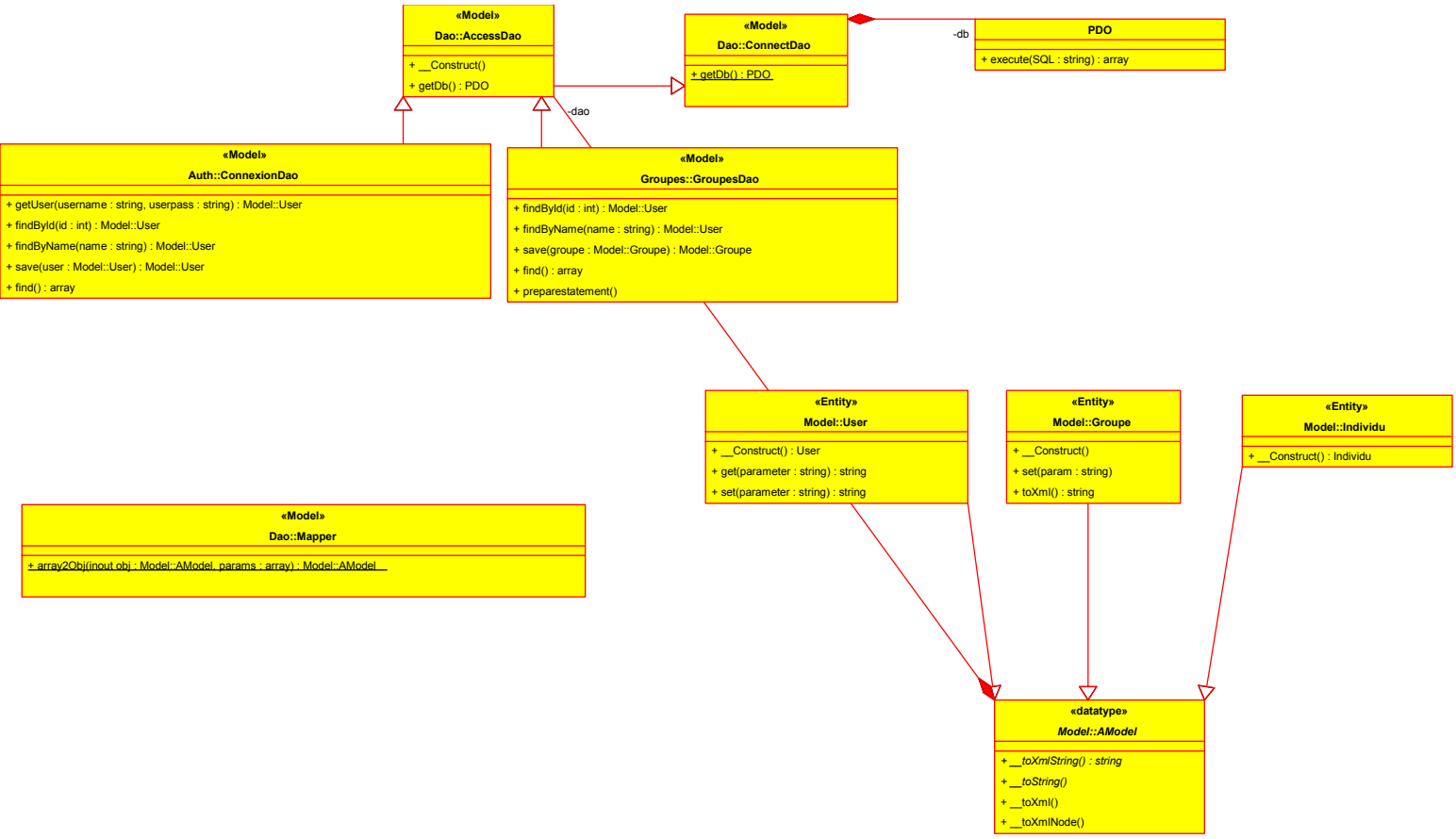
Exemple ce commentaires

```
65:     /**Authenticate the User
66:      *
67:      * @param \Air2Java\Model\User $user User to connect in.
68:      * @return boolean true if user status is defined..
69:      * @deprecated : une vraie identification est faite par Connexion_M !
70:      */
71:     private function connect($user) {
72:         require_once __DIR__ . '/dao/UserDao.php';
73:         $dao = new \Air2Java\Auth\UserDao();
74:         $this->user->setStatus($dao->getUser($user->getNom(), $user-
>getPassword()));
75:         //return ($user->getNom() == $user->getPassword());
76:         return $this->user->getStatus();
77:     }
```

# 5 Conception détaillée

## 5.1 Persistance : DAO

La couche de persistance suit le patron DAO, et est organisé en plusieurs classes :




## 5.1.1 ConnectDAO

Assure la connexion à la base de données Oracle grâce à un connecteur PDO, librairie fournie en standard par PHP5.

Pour instancier l'objet PDO, il utilise la classe *Config* qui fournit les paramètres de connexion enregistrés dans le fichier config.ini

ConnectDAO utilise un patron Singleton pour limiter le nombre de connexions concurrentes.

```
class ConnectDao {
    /** @var \PDO */
    private $db = null;

    /**
     * Private Constructor for Singleton Pattern
     */
    private function __construct() { // Pattern Singleton
    }
    private function __clone() { // Pattern Singleton
    }

    /**
     *  Close db connection and liberate the ressource.
     */
    public function __destruct() {
        $this->db = null;
```

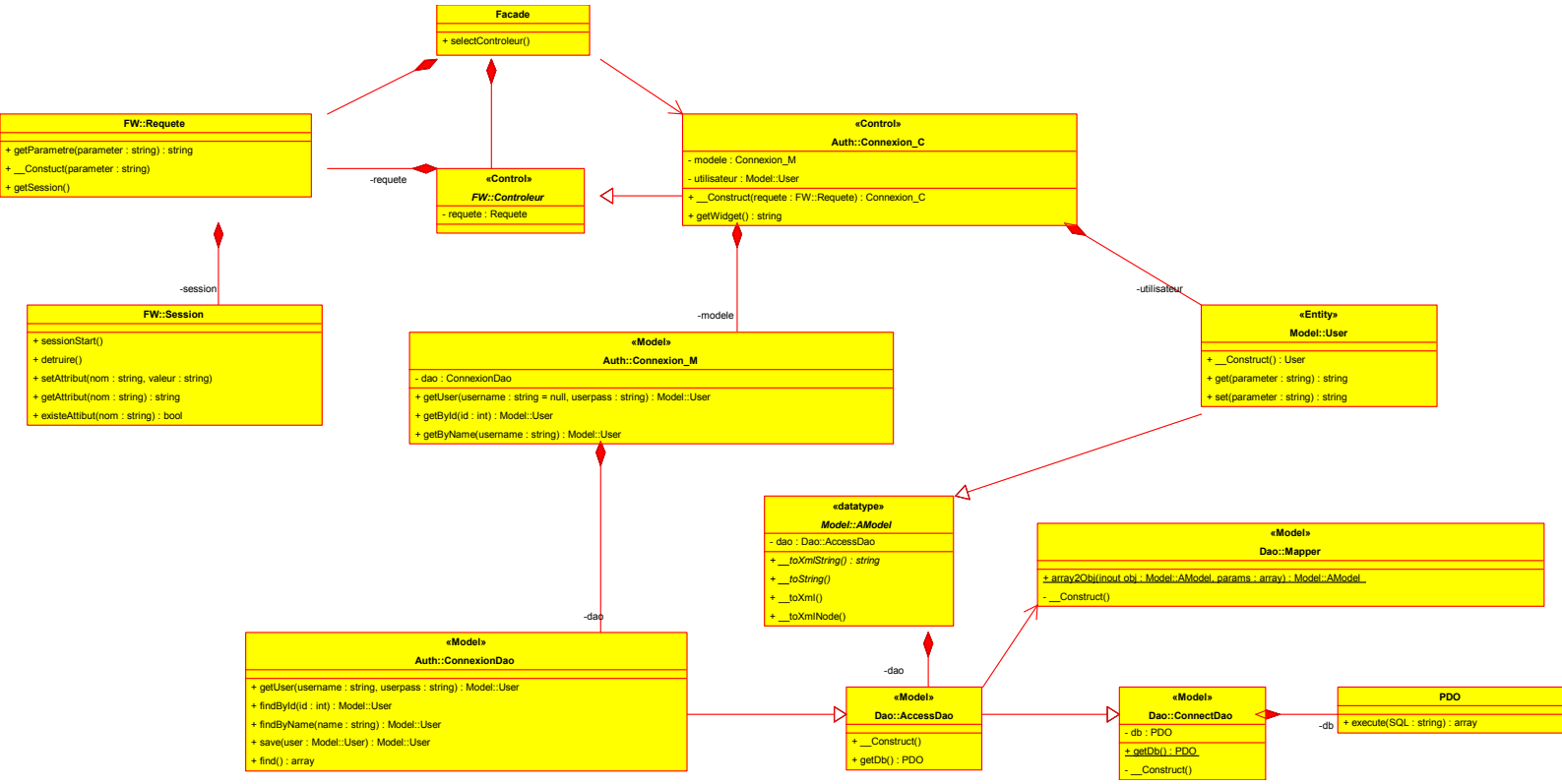
```
    }

    /**
     * (PHP 5 &gt;= 5.1.0, PECL pdo &gt;= 0.1.0)
     * @link \PDO::__construct()
     * @return \PDO instance representing a connection to a database
     * @throws Exception if new PDO() fails.
     * @author Rom1 <air2Java@localhost>
     */
    protected function getDb() {
        if ($this->db !== null) {
            return $this->db;
        }

        require_once __DIR__ . '/../config/Config.php';

        $config = \Air2Java\Config::getConfig("db");
        try {
            $this->db = new \PDO($config['dsn'], $config['username'], $config['password']);
        } catch (Exception $ex) {
            throw new Exception('DB connection error: ' . $ex->getMessage());
        }
        return $this->db;
    }
}
```

Un appel à la base devra donc se faire de manière statique (ex. depuis AccessDao.php :

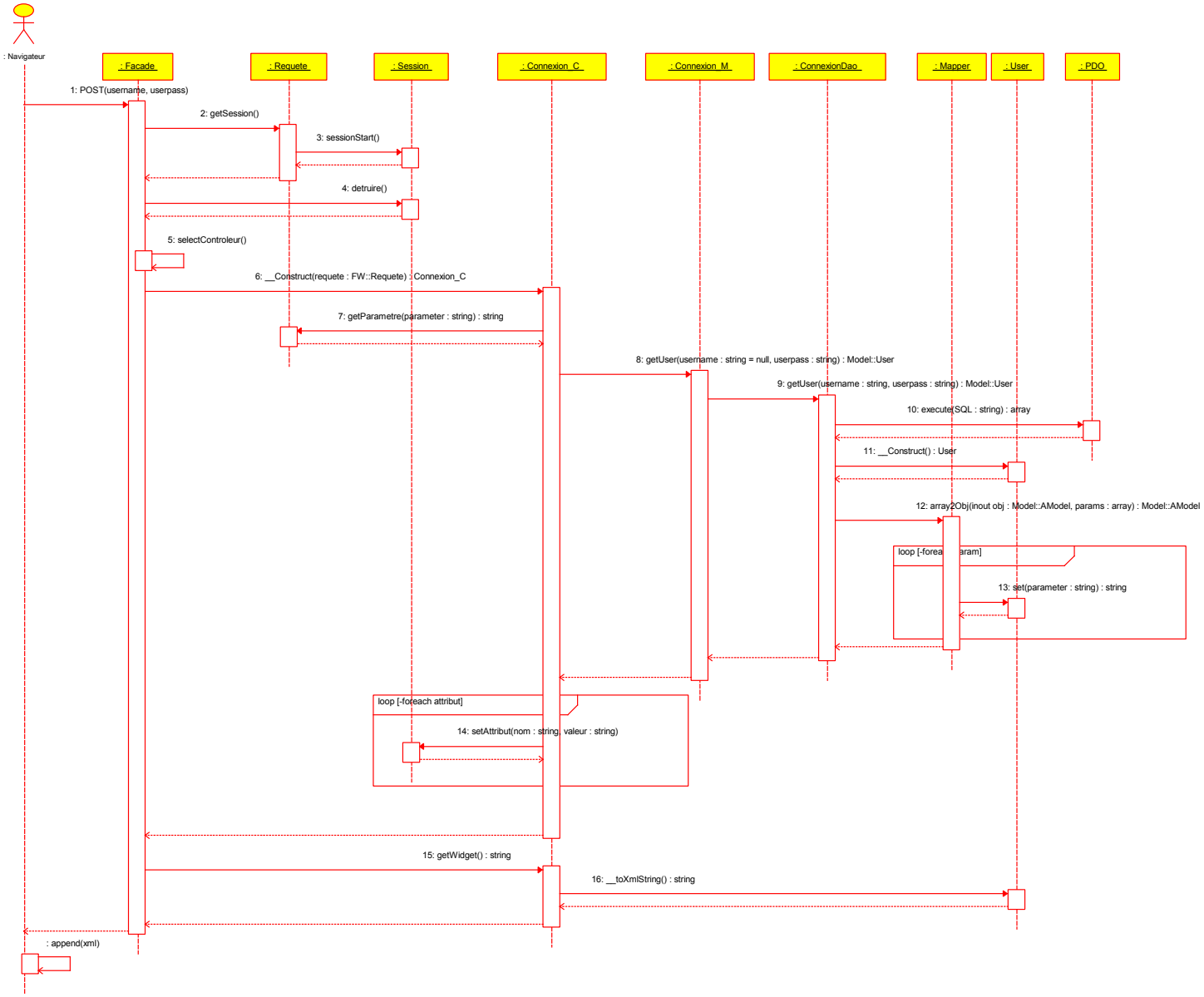
```
$this->db = ConnectDao::getDb()
```

# 5.2 Cas d'Utilisation : Authentification

## 5.2.1 Diagramme des Classes Participantes

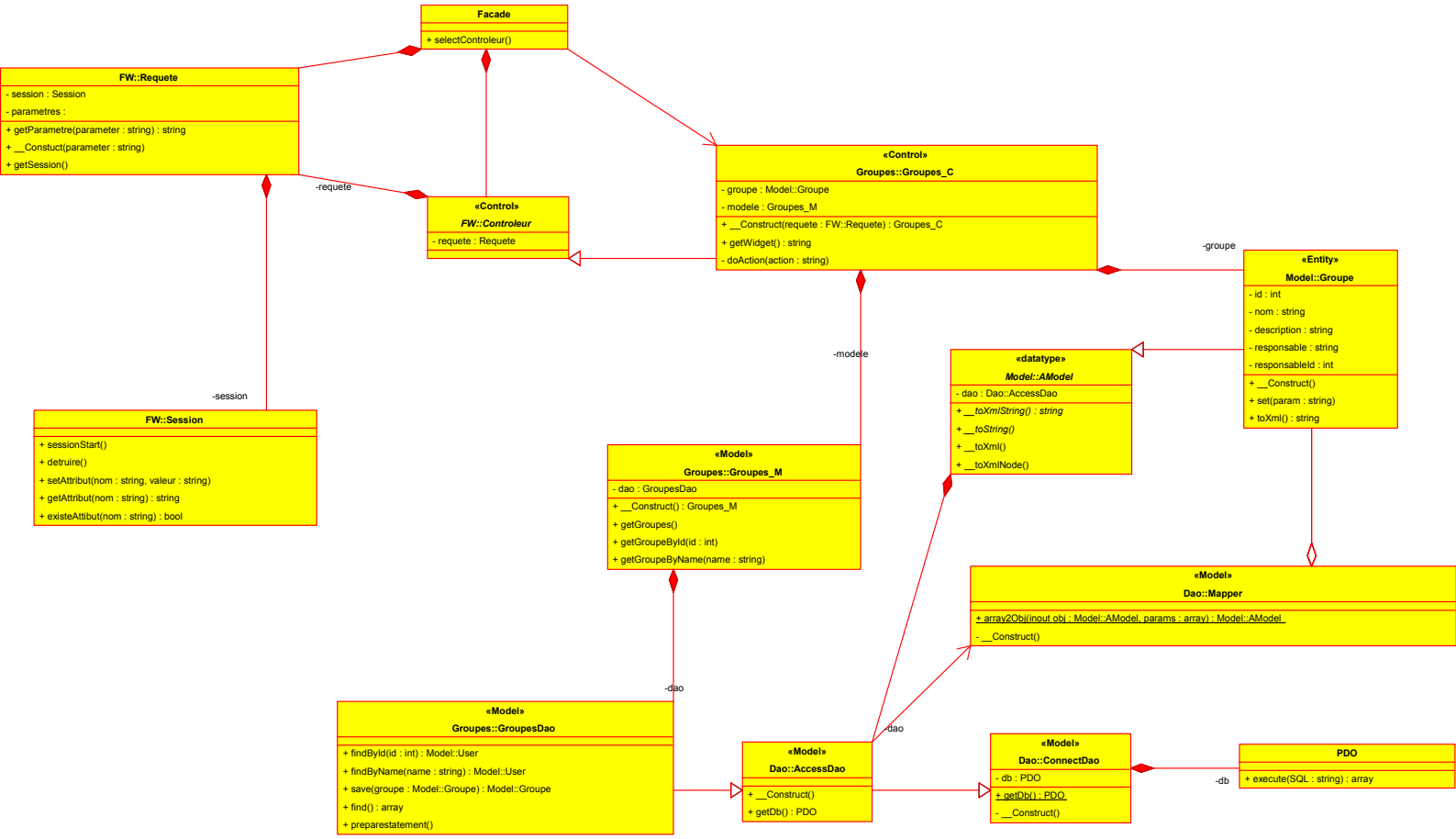



## 5.2.2 Diagramme de Séquence

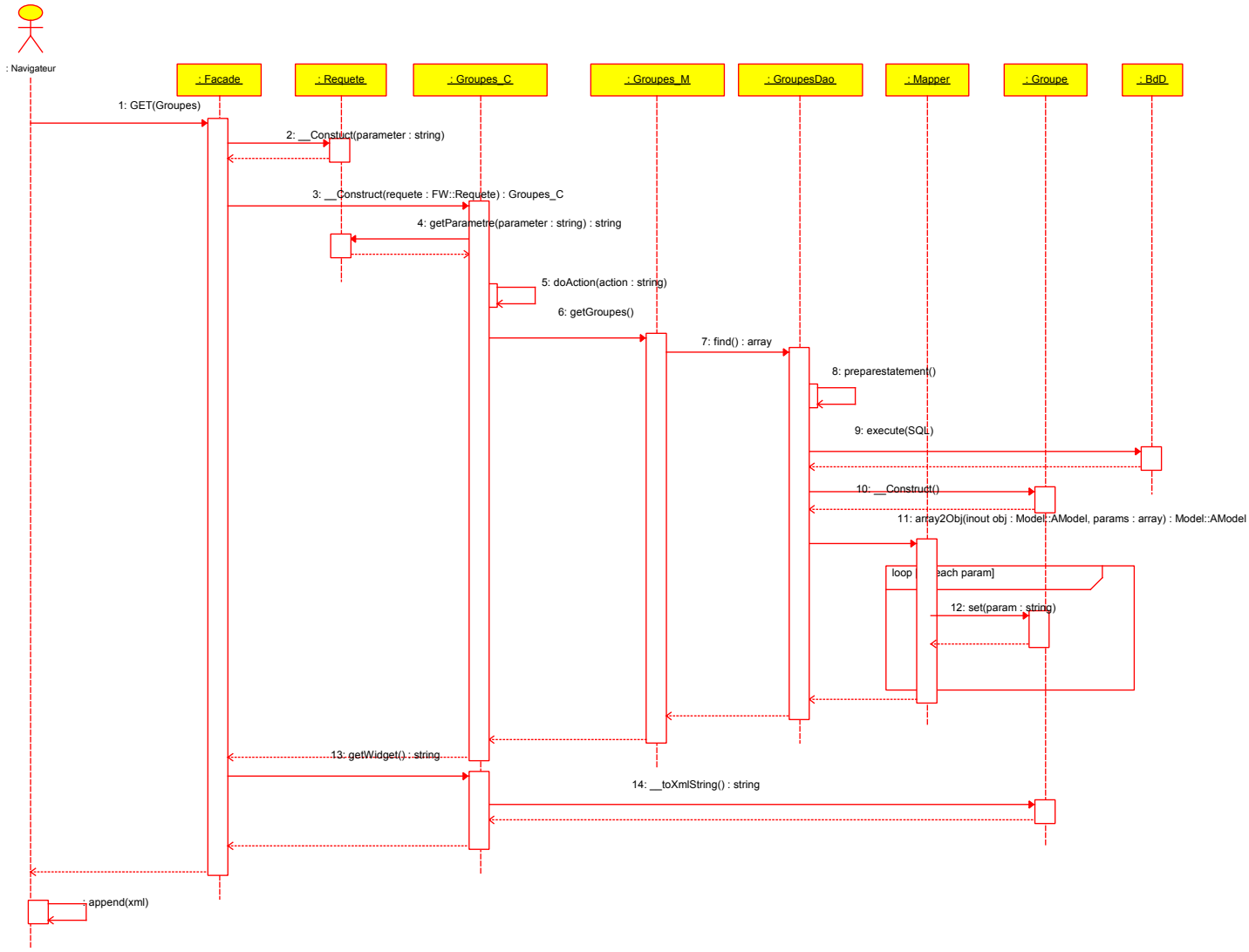

: Navigateur
: Facade
: Requete
: Session
: Connexion_C
: Connexion_M
: ConnexionDao
: Mapper
: User
: PDO
1: POST(username, userpass)
2: getSession()
3: sessionStart()
4: detruire()
5: selectControleur()
6: __Construct(requete : FW::Requete) : Connexion_C
7: getParametre(parameter : string) : string
8: getUser(username : string = null, userpass : string) : Model::User
9: getUser(username : string, userpass : string) : Model::User
10: execute(SQL : string) : array
11: __Construct() : User
12: array2Obj(inout obj : Model::AModel, params : array) : Model::AModel
loop [-forea aram]
13: set(parameter : string) : string
loop [-foreach attribut]
14: setAttribut(nom : string, valeur : string)
15: getWidget() : string
16: __toXmlString() : string
: append(xml)

# 5.3 Cas d'Utilisation : Groupes

## 5.3.1 Diagramme de Classes Participantes



### 5.3.2 Diagramme de Séquence




# 5.4 IHM

L'Interface Homme-Machine se fait au travers d'une page web.

### 5.4.1 HTML

Les vues ont tout d'abord été écrites en HTML5, mais des problèmes se sont posés avec l'API DOM de php, qui considère que les nouvelles balises ne sont pas conformes a son standard html, et renvoie des erreurs lors de l'utilisation du parseur html.
N'ayant pas le temps de chercher plus avant les possibilités offertes par d'autres bibliothèques, j'ai évité d'utiliser les balises qui posaient problèmes.

### 5.4.2 CSS

La mise en page est définie dans 2 CSS :

- style.css définit le style des éléments.
- layout.css définit le placement des composants.

### 5.4.3 Modules

Le site étant conçu de façon modulaire, il est assez facile de rajouter des fonctionnalités.

Parmi celles envisagées, on peut imaginer :

- des flux RSS pour annoncer les rencontres sur des agrégateurs extérieurs
- un Web-Service qui fournisse des annuaires de groupes et de musiciens
- Une géolocalisation des membres, groupes, et œuvres
- des extraits musicaux et vidéos ainsi que les fiches de scènes des groupes
- une encyclopédie des musiques et danses folkloriques
- une boutique en ligne avec les CD des groupes
- ...

# 6 Conclusions

Pour quelqu'un qui avait finit, la mort dans l'âme, par tirer un trait sur son métier, redécouvrir l'Informatique m'a donner l'envie de revenir Informaticien, et de me donner les moyens de faire du bel ouvrage.

Remettre dans le bon ordre des savoirs acquis sur le tas au fils d'exploration aléatoires m'a fait réaliser les liens qui existent entre les différents professions souvent cloisonnés de nos métiers, et m'a permis d'acquérir certaines clés qui me manquaient pour travailler proprement dans un monde où l'individualité a été un modèle récurent.

Cependant, si ce projet été à refaire, certains points seraient à revoir :

## 6.1 Graphisme

Je ne suis pas graphiste.

Un bon desinger permet d'avoir un bon rendu, et mets en valeur le travail obscur du développeur, dont l'utilisateur final ignore l'élégance et les subtilités...

Dans le cadre d'un projet professionnel, travailler avec des personnes compétentes dans leurs domaines permet d'être plus performant dans le sien !

## 6.2 Il manque les tests !

Le temps impartit était trop court pour réaliser une gestion de projet intégrant des test unitaires poussés.

Cependant, on ne peut pas se permettre de passer à côté, et l'utilisation de solutions telles que PhpUnit, Nette ou Atoum simplifient suffisamment les choses pour pouvoir au moins indiquer les

assertions que l'on revendique.

## 6.3 Utiliser un Framework

Cela m'aurai fait gagner beaucoup de temps, et m'aurai sûrement permis d'aboutir un projet acceptable...

Mais aurais-je bien compris tout de qu'il se passe dans ces boites noires ?