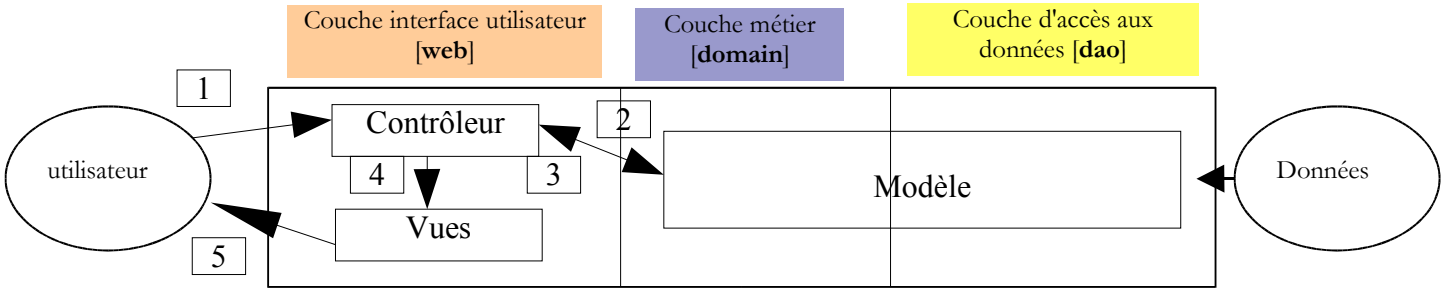
Méthodologie de développement MVC d'une application WEB/PHP

serge.tahe@istia.univ-angers.fr, révision mars 2005

1 Utiliser l'architecture MVC dans les applications web/php

Le modèle MVC (Modèle-Vue-Contrôleur) cherche à séparer nettement les couches présentation, traitement et accès aux données. Une application web respectant ce modèle sera architecturée de la façon suivante :



Le traitement d'une demande d'un client se déroule selon les étapes suivantes :

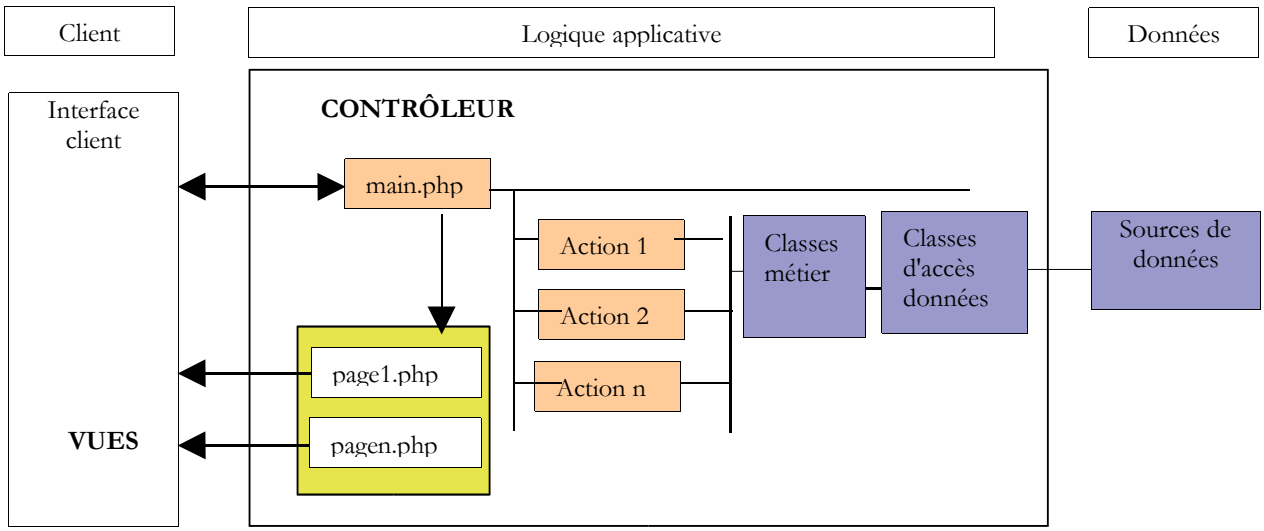
1. le client fait une demande au contrôleur. Ce contrôleur voit passer toutes les demandes des clients. C'est la porte d'entrée de l'application. C'est le C de MVC.
2. le contrôleur traite cette demande. Pour ce faire, il peut avoir besoin de l'aide de la couche métier, ce qu'on appelle le modèle M dans la structure MVC.
3. le contrôleur reçoit une réponse de la couche métier. La demande du client a été traitée. Celle-ci peut appeler plusieurs réponses possibles. Un exemple classique est
   - une page d'erreurs si la demande n'a pu être traitée correctement
   - une page de confirmation sinon
4. le contrôleur choisit la réponse (= vue) à envoyer au client. Celle-ci est le plus souvent une page contenant des éléments dynamiques. Le contrôleur fournit ceux-ci à la vue.
5. la vue est envoyée au client. C'est le V de MVC.

Une telle architecture est souvent appelée architecture 3-tier ou à 3 niveaux.

L'**interface utilisateur** est souvent un navigateur web mais cela pourrait être également une application autonome qui via le réseau enverrait des requêtes HTTP au service web et mettrait en forme les résultats que celui-ci lui envoie. La **logique applicative** est constituée des scripts traitant les demandes de l'utilisateur, des classes métier et d'accès aux données. La **source de données** est souvent une base de données mais cela peut être aussi de simples fichiers plats, un annuaire LDAP, un service web distant,... Le développeur a intérêt à maintenir une grande indépendance entre ces trois entités afin que si l'une d'elles change, les deux autres n'aient pas à changer ou peu.

L'architecture MVC est bien adaptée à des applications web écrites avec des langages orientés objet. Le langage PHP (4.x) n'est pas orienté objet. On peut néanmoins faire un effort de structuration du code et de l'architecture de l'application afin de se rapprocher du modèle MVC :

- On mettra la logique métier de l'application dans des modules séparés des modules chargés de contrôler le dialogue demande-réponse. L'architecture MVC devient la suivante :



M=modèle les classes métier, les classes d'accès aux données et la base de données
V=vues les pages PHP
C=contrôleur le script PHP de traitement des requêtes clientes, les scripts PHP [Action] de traitement des actions.

Dans le bloc [Logique Applicative], on pourra distinguer
- le programme principal ou contrôleur [**main.php**], qui est la porte d'entrée de l'application.
- le bloc [Actions], ensemble de scripts PHP chargés d'exécuter les actions demandées par l'utilisateur.
- le bloc [Classes métier] qui regroupe les modules PHP nécessaires à la logique de l'application. Ils sont indépendants du client. Par exemple, la fonction permettant de calculer un impôt à partir de certaines informations qui lui sont fournies en paramètres, n'a pas à se soucier de la façon dont ont été acquises celles-ci.
- le bloc [Classes d'accès aux données] qui regroupe les modules PHP qui obtiennent les données nécessaires au contrôleur, souvent des données persistantes (BD, fichiers, ...)
- les générateurs [pagex.php] des vues envoyées comme réponse au client.

Dans les cas les plus simples, la logique applicative est souvent réduite à deux modules :
- le module [contrôle] assurant le dialogue client-serveur : traitement de la requête, génération des diverses réponses
- le module [métier] qui reçoit du module [contrôle] des données à traiter et lui fournit en retour des résultats. Ce module [métier] gère alors lui-même l'accès aux données persistantes.

2 Une démarche de développement MVC en web/php

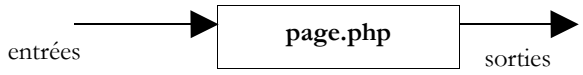
Nous proposons ici une démarche pour le développement d'applications web/php respectant l'architecture MVC. Elle n'est là que pour ouvrir des pistes. Le lecteur l'adaptera à ses goûts et besoins.

- on commencera par définir toutes les vues de l'application. Celles-ci sont les pages web présentées à l'utilisateur. On se placera du point de vue de celui-ci pour dessiner les vues. On distingue trois types de vues :
  - le **formulaire de saisie** qui vise à obtenir des informations de l'utilisateur. Celui-ci dispose en général d'un bouton pour envoyer au serveur les informations saisies.
  - la **page de réponse** qui ne sert qu'à donner de l'information à l'utilisateur. Celle-ci dispose souvent d'un ou de plusieurs liens permettant à l'utilisateur de poursuivre l'application avec une autre page.

la **page mixte** : le contrôleur a envoyé au client une page contenant des informations qu'il a générées. Cette même page va servir au client pour fournir au contrôleur de nouvelles informations provenant de l'utilisateur.

- chaque vue donnera naissance à une page PHP. Pour chacune de celles-ci :
  - on dessinera l'aspect de la page
  - on déterminera quelles sont les parties dynamiques de celle-ci :
    - les informations à destination de l'utilisateur qui devront être fournies par le contrôleur en paramètres à la vue PHP. Une solution simple est la suivante :
      - le contrôleur met dans un dictionnaire **$dReponse** les informations qu'il veut fournir à une vue V
      - le contrôleur fait afficher la vue V. Si celle-ci correspond au fichier source **V.php**, cet affichage est obtenu simplement par l'instruction **include V.php**.
      - l'inclusion précédente est une inclusion de code dans le contrôleur. Le dictionnaire **$dReponse** renseigné par celui-ci est accessible directement par le code de **V.php**.
    - les données de saisie qui devront être transmises au programme principal pour traitement. Celles-ci devront faire partie d'un formulaire HTML (**balise <form>**).

- on pourra schématiser les E/S de chaque vue



  - les entrées sont les données que devra fournir le contrôleur à la page PHP
  - les sorties sont les données que devra fournir la page PHP au contrôleur de l'aplication. Elles font partie d'un formulaire HTML et le contrôleur les récupèrera par une opération du type **$_GET["param"]** (méthode GET) ou **$_POST ["param"]** (méthode POST).

- fréquemment la page finale envoyée au client n'est pas une vue mais une composition de vues. Par exemple, la page envoyée à un utilisateur peut avoir la forme suivante :

  La zone 1 peut être un bandeau de titre, la zone 2 un bandeau de menu, la zone 3 une zone de contenu. En PHP cette composition peut être obtenue par le code HTML/PHP suivant :

```
<table>
  <tr>
    <td><?php include zone1.php ?></td>
  </tr>
  <tr>
    <td><?php include zone2.php ?></td>
    <td><?php include zone3.php ?></td>
  </tr>
</table>
```

  On peut rendre ce code dynamique en écrivant :

```
<table>
  <tr>
    <td><?php include $dReponse['urlZone1'] ?></td>
  </tr>
  <tr>
    <td><?php include $dReponse['urlZone2'] ?></td>
    <td><?php include $dReponse['urlZone3'] ?></td>
  </tr>
</table>
```

  Cette composition de vues peut être le format unique de la réponse faite à l'utilisateur. Dans ce cas, chaque réponse au client devra fixer les trois URL à charger dans les trois zones avant de faire afficher la page réponse. On peut généraliser cet exemple en imaginant qu'il y ait plusieurs modèles possibles pour la page de réponse. La réponse au client devra donc :
  - fixer le modèle à utiliser
  - fixer les éléments (vues élémentaires) à inclure dans celui-ci
  - demander l'affichage du modèle

- On écrira le code PHP/HTML de chaque modèle de réponse. Son code est généralement simple. Celui de l'exemple ci-desssus pourrait être :

```
<?php
  // initialisations pour les tests sans contrôleur
...
?>
<html>
  <head>
    <title><?php echo $dReponse['titre'] ?></title>
    <link type="text/css" href="<?php echo $dReponse['style']['url'] ?>" rel="stylesheet" />
  </head>
  <body>
    <table>
      <tr>
        <td><?php include $dReponse['urlZone1'] ?></td>
      </tr>
      <tr>
```

```
      <td><?php include $dReponse['urlZone2'] ?></td>
      <td><?php include $dReponse['urlZone3'] ?></td>
    </tr>
   </table>
  <body>
</html>
```

On utilisera chaque fois que c'est possible une feuille de style afin de pouvoir changer le "look" de la réponse sans avoir à modifier le code PHP/HTML.

- On écrira le code PHP/HTML de chaque vue élémentaire, élément de la page envoyée en réponse au client. Il aura le plus souvent la forme suivante :

```
<?php
  // éventuellement qqs initialisations notamment en phase de débogage
  ...
?>

<balise>
...
    // on cherchera ici à minimiser le code php
</balise>
```

On notera qu'une vue élémentaire s'intègre dans un modèle qui les assemble pour en faire une page HTML. Le code HTML de la vue élémentaire vient s'insérer dans le code du modèle présenté à l'étape précédente. Le plus souvent, le code du modèle intègre déjà les balises <html>, <head>, <body>. Il est donc peu fréquent de trouver ces balises dans une vue élémentaire.

- On peut procéder aux tests des différents modèles de réponse et vues élémentaires

  - chaque modèle de réponse est testé. Si un modèle s'appelle **modele1.php**, on demandera avec un navigateur l'URL **http://localhost/chemin/modele1.php** Le modèle attend des valeurs du contrôleur. Ici on l'appelle directement et non via le contrôleur. Le modèle ne recevra pas les paramètres attendus. Afin que les tests soient néanmoins possibles on initialisera soi-même, avec des constantes, les paramètres attendus dans la page PHP du modèle.
  - chaque modèle est testé ainsi que toutes les vues élémentaires. C'est aussi le moment d'élaborer les premiers éléments des feuilles de style utilisées.

- On écrit ensuite la logique applicative de l'application :
  - Le contrôleur ou programme principal gère en général plusieurs actions. Il faut que dans les requêtes qui lui parviennent l'action à accomplir soit définie. Cela peut se faire au moyen d'un paramètre de la requête que nous appellerons ici **action** :
    - si la requête provient d'un formulaire (<form>), ce paramètre peut être un paramètre caché du formulaire :

```
<form ... action="/C/main.php" method="post"  ...>
    <input type="hidden" name="action" value="uneAction">
    ...
</form>
```

    - si la requête provient d'un lien, on peut paramétrer celui-ci :

```
<a href="/C/main.php?action=uneAction">lien</a>
```

    Le contrôleur peut commencer par lire la valeur de ce paramètre puis déléguer le traitement de la requête à un module chargé de traiter ce type de requête. Nous nous plaçons ici dans le cas ou tout est contrôlé par un unique script appelé **main.php**. Si l'application doit traiter des actions **action1**, **action2**, ..., **actionx** on peut créer au sein du contrôleur une fonction par action. S'il y a beaucoup d'actions, on peut arriver à un contrôleur "dinosaure". On peut aussi créer des scripts **action1.php**, **action2.php**, ...,**actionx.php** chargés de traiter chacune des actions. Le contrôleur devant traiter l'acton **actionx** se contentera de charger le code du script correspondant par une instruction du genre **include "actionx.php"**. L'avantage de cette méthode est qu'on travaille en-dehors du code du contrôleur. Chaque personne de l'équipe de développement peut ainsi travailler sur le script de traitement d'une action **actionx** de façon relativement indépendante. L'inclusion du code du script **actionx.php** dans le code du contrôleur au moment de l'exécution a également l'avantage d'alléger le code chargé en mémoire. Seul le code de traitement de l'action en cours est chargé. Cette inclusion de code fait que des variables du contrôleur peuvent entrer en collision avec celles du script de l'action. Nous verrons que nous pouvons faire en sorte de limiter les variables du contrôleur à quelques variables bien définies qu'il faudra alors éviter d'utiliser dans les scripts.

  - On cherchera systématiquement à isoler le code métier ou le code d'accès aux données persistantes dans des modules distincts. Le contrôleur est une sorte de chef d'équipe qui reçoit des demandes de ses clients (clients web) et qui les fait exécuter par les personnes les plus appropriées (les modules métier). Lors de l'écriture du contrôleur, on déterminera l'interface des modules métier à écrire. Cela si ces modules métier sont à construire. S'ils existent déjà, alors le contrôleur s'adaptera à l'interface de ces modules existants.

- On écrira le squelette des modules métier nécessaires au contrôleur. Par exemple, si celui-ci utilise un module **getCodes** rendant un tableau de chaînes de caractères, on peut se contenter dans un premier temps d'écrire :

```
function getCodes(){
  return array("code1","code2","code3");
}
```

- On peut alors passer aux tests du contrôleur et des scripts PHP associés :

  - le contrôleur, les scripts d'actions, les modèles, les vues, les ressources nécessaires à l'application (images,...) sont placés dans le dossier **DC** associé au contexte **C** de l'application.
  - ceci fait, l'application est testée et les premières erreurs corrigées. Si **main.php** est le contrôleur et **C** le contexte de l'application, on demandera l'URL **http://localhost/C/main.php**. A la fin de cette phase, l'architecture de l'application est opérationnelle. Cette phase de test peut être délicate sachant qu'on n'a peu d'outils de débogage si on n'utilise pas des environnements de développement évolués et en général payants. On pourra s'aider d'instructions **echo "message"** qui écrivent dans le flux HTML envoyé au client et qui apparaissent donc dans la page web affichée par le navigateur.

- On écrit enfin les classes métier dont a besoin le contrôleur. On a là en général le développement classique d'une classe PHP, développement le plus souvent indépendant de toute application web. Elle sera tout d'abord testée en dehors de cet environnement, avec une application console par exemple. Lorsqu'une classe métier a été écrite, on l'intégre dans l'architecture de déploiement de l'application web et on teste sa correcte intégration. On procèdera ainsi pour chaque classe métier.

3 Un contrôleur générique

3.1 Introduction

Dans la méthode précédente, il était entendu que nous avions à écrire le contrôleur appelé **main.php**. Avec un peu d'expérience, on réalise que ce contrôleur fait souvent les mêmes choses et il est alors tentant d'écrire un contrôleur générique utilisable dans la plupart des applications web. Le code de ce contrôleur pourrait être le suivant :

```
<?php
 // contrôleur générique

  // lecture config
  include 'config.php';

  // inclusion de bibliothèques
  for($i=0;$i<count($dConfig['includes']);$i++){
    include($dConfig['includes'][$i]);
  }//for

  // on démarre ou reprend la session
  session_start();
  $dSession=$_SESSION["session"];
  if($dSession) $dSession=unserialize($dSession);

  // on récupère l'action à entreprendre
  $sAction=$_GET['action'] ? strtolower($_GET['action']) : 'init';
  $sAction=strtolower($_SERVER['REQUEST_METHOD']).":$sAction";

 // l'enchaînement des actions est-il normal ?
  if( ! enchainementOK($dConfig,$dSession,$sAction)){
    // enchaînement anormal
    $sAction='enchainementinvalide';
  }//if

 // traitement de l'action
  $scriptAction=$dConfig['actions'][$sAction] ?
    $dConfig['actions'][$sAction]['url'] :
    $dConfig['actions']['actioninvalide']['url'];
  include $scriptAction;

  // envoi de la réponse(vue) au client
  $sEtat=$dSession['etat']['principal'];
  $scriptVue=$dConfig['etats'][$sEtat]['vue'];
  include $scriptVue;

  // fin du script - on ne devrait pas arriver là sauf bogue
  trace ("Erreur de configuration.");
  trace("Action=[$sAction]");
  trace("scriptAction=[$scriptAction]");
  trace("Etat=[$sEtat]");
  trace("scriptVue=[$scriptVue]");
```

```
  trace ("Vérifiez que les script existent et que le script [$scriptVue] se termine par l'appel à
finSession.");
  exit(0);

  // ------------------------------------------------------------------
  function finSession(&$dConfig,&$dReponse,&$dSession){
    // $dConfig : dictionnaire de configuration
    // $dSession : dictionnaire contenant les infos de session
  // $dReponse : le dictionnaire des arguments de la page de réponse

    // enregistrement de la session
    if(isset($dSession)){
      // on met les paramètres de la requête dans la session
      $dSession['requete']=strtolower($_SERVER['REQUEST_METHOD'])=='get' ? $_GET :
        strtolower($_SERVER['REQUEST_METHOD'])=='post' ? $_POST : array();
     $_SESSION['session']=serialize($dSession);
      session_write_close();
    }else{
     // pas de session
      session_destroy();
    }

  // on présente la réponse
  include $dConfig['vuesReponse'][$dReponse['vuereponse']]['url'];

    // fin du script
    exit(0);
  }//finsession

  //--------------------------------------------------------------------
 function enchainementOK(&$dConfig,&$dSession,$sAction){
    // vérifie si l'action courante est autorisée vis à vis de l'état précédent
    $etat=$dSession['etat']['principal'];
    if(! isset($etat)) $etat='sansetat';

    // vérification action
    $actionsautorisees=$dConfig['etats'][$etat]['actionsautorisees'];
    $autorise= ! isset($actionsautorisees) || in_array($sAction,$actionsautorisees);
  return $autorise;
  }

  //--------------------------------------------------------------------
  function dump($dInfos){
    // affiche un dictionnaire d'informations
    while(list($clé,$valeur)=each($dInfos)){
      echo "[$clé,$valeur]<br>\n";
    }//while
  }//suivi

  //--------------------------------------------------------------------
  function trace($msg){
    echo $msg."<br>\n";
  }//suivi

?>
```

3.2 Le fichier de configuration de l'application

L'application est configurée dans un script portant obligatoirement le nom **config.php**. Les paramètres de l'application sont mis dans un dictionnaire appelé $**dConfig** utilisé aussi bien par le contrôleur, que les scripts d'actions, les modèles et vues élémentaires.

3.3 Les bibliothèques à inclure dans le contrôleur

Les bibliothèques à inclure dans le code du contrôleur sont placées dans le tableau **$dConfig['includes']**. Le contrôleur les inclut avec la séquence de code suivante :

```
  // lecture config
  include "config.php";

  // inclusion de bibliothèques
  for($i=0;$i<count($dConfig['includes']);$i++){
    include($dConfig['includes'][$i]);
  }//for
```

3.4 La gestion des sessions

Le contrôleur générique gère automatiquement une session. Il sauvegarde et récupère le contenu d'une session via le dictionnaire **$dSession**. Ce dictionnaire peut contenir des objets qu'il faut sérialiser pour pouvoir les récupérer ensuite correctement. La clé associée à ce dictionnaire est **'session'**. Aussi récupérer une session se fait avec le code suivant :

```
// on démarre ou reprend la session
session_start();
$dSession=$_SESSION["session"];
if($dSession) $dSession=unserialize($dSession);
```

Si une action souhaite mémoriser des informations dans la session, elle ajoutera des clés et des valeurs dans le dictionnaire **$dSession**. Toutes les actions partageant la même session, il y a un risque de conflit de clés de session si l'application est développée indépendamment par plusieurs personnes. C'est une difficulté. Il faut développer un référentiel listant les clés de session, référentiel partagé par tous. Nous verrons que chaque action se termine par l'appel à la fonction **finSession** suivante :

```
// ------------------------------------------------------------------
function finSession(&$dConfig,&$dReponse,&$dSession){
  // $dConfig : dictionnaire de configuration
  // $dSession : dictionnaire contenant les infos de session
// $dReponse : le dictionnaire des arguments de la page de réponse

  // enregistrement de la session
  if(isset($dSession)){
    // on met les paramètres de la requête dans la session
    $dSession['requete']=strtolower($_SERVER['REQUEST_METHOD'])=='get' ? $_GET :
      strtolower($_SERVER['REQUEST_METHOD'])=='post' ? $_POST : array();
   $_SESSION['session']=serialize($dSession);
    session_write_close();
  }else{
   // pas de session
    session_destroy();
  }

// on présente la réponse
include $dConfig['vuesReponse'][$dReponse['vuereponse']]['url'];

  // fin du script
  exit(0);
}//finsession
```

Une action peut décider de ne pas poursuivre une session. Il lui suffit pour cela de ne pas passer de valeur au paramètre **$dSession** de la fonction **finSession**, auquel cas la session est supprimée (session_destroy). Si le dictionnaire **$dSession** existe, il est sauvegardé dans la session et celle-ci est ensuite enregistrée (session_write_close). L'action en cours peut donc mémoriser des éléments dans la session en ajoutant des éléments au dictionnaire **$dSession**. On notera que le contrôleur mémorise automatiquement les paramètres de la requête courante dans la session. Cela permettra de les récupérer si besoin est pour traiter la requête suivante.

3.5 L'exécution des actions

Le contrôleur attend des requêtes ayant un paramètre **action=XX**. Si ce paramètre n'existe pas dans la requête et que celle-ci est un GET, l'action prend la valeur **'get:init'**. C'est le cas de la toute première requête faite au contrôleur et qui est de la forme **http://machine:port/chemin/main.php**.

```
// on récupère l'action à entreprendre
$sAction=$_GET['action'] ? strtolower($_GET['action']) : 'init';
$sAction=strtolower($_SERVER['REQUEST_METHOD']).":$sAction";
```

Par configuration, à chaque action est associé un script chargé de traiter cette action. Par exemple :

```
// configuration des actions de l'application
$dConfig['actions']['get:init']=array('url'=>'a-init.php');
$dConfig['actions']['post:calculerimpot']=array('url'=>'a-calculimpot.php');
$dConfig['actions']['get:retourformulaire']=array('url'=>'a-retourformulaire.php');
$dConfig['actions']['post:effacerformulaire']=array('url'=>'a-init.php');
$dConfig['actions']['enchainementinvalide']=array('url'=>'a-enchainementinvalide.php');
$dConfig['actions']['actioninvalide']=array('url'=>'a-actioninvalide.php');
```

Deux actions sont prédéfinies :

`enchainementinvalide`	cas où l'action en cours ne peut suivre l'action précédente
`actioninvalide`	cas où l'action demandée n'existe pas dans le dictionnaire des actions

Les actions propres à l'application sont notées sous la forme **méthode:action** où méthode est la méthode **get** ou **post** de la requête et **action** l'action demandée, ici : *init, calculerimpot, retourformulaire, effacerformulaire*. On remarquera que l'action est récupérée, quelque soit la méthode GET ou POST d'envoi des paramètres, par la $_GET['action'] :

```
  // on récupère l'action à entreprendre
  $sAction=$_GET['action'] ? strtolower($_GET['action']) : 'init';
  $sAction=strtolower($_SERVER['REQUEST_METHOD']).":$sAction";
```

En effet, même si un formulaire est posté, on peut toujours écrire :

```
<form method='post' action='main.php?action=calculerimpot'>
..
</form>
```

Les éléments du formulaire seront postés (method='post'). Néanmoins l'url demandée sera **main.php?action=calculerimpot**. Les paramètres de cette URL seront récupérés dans le dictionnaire **$_GET** alors que les autres éléments du formulaire seront eux récupérés dans le dictionnaire **$_POST**.

Muni du dictionnaire des actions, le contrôleur exécute l'action demandée de la façon suivante :

```
// traitement de l'action
  $scriptAction=$dConfig['actions'][$sAction] ?
    $dConfig['actions'][$sAction]['url'] :
    $dConfig['actions']['actioninvalide']['url'];
  include $scriptAction;
```

Si l'action demandée n'est pas dans le dictionnaire des actions, c'est le script correspondant à une action invalide qui sera exécuté. Une fois le script de l'action chargé au sein du contrôleur, il s'exécute. On remarquera qu'il a accès aux variables du contrôleur ($dConfig, $dSession) ainsi qu'aux dictionnaires super-globaux de PHP ($_GET, $_POST, $_SERVER, $_ENV, $_SESSION). On trouvera dans le script, de la logique applicative et des appels à des classes métier. Dans tous les cas, l'action devra

1. renseigner le dictionnaire **$dSession** si des éléments doivent être sauvegardés dans la session courante
2. indiquer dans **$dReponse['vuereponse']** le nom du modèle de réponse à afficher
3. se terminer par l'appel à **finSession($dConfig, $dReponse, $dSession)**. Si la session doit être détruite, l'action se terminera simplement par l'appel à **finSession($dConfig, $dReponse)**.

Par souci de cohérence, l'action pourra mettre dans le dictionnaire **$dReponse** toutes les informations dont ont besoin les vues. Mais il n'y a pas d'obligation. Seule la valeur **$dReponse['vuereponse']** est indispensable. On remarquera que tout script d'action se termine par l'appel à la fonction **finSession** qui elle-même se termine par une opération **exit**. On ne revient donc pas d'un script d'action.

3.6 L'enchaînement des actions

On peut voir une application web comme un automate à états finis. Les différents états de l'application sont associés aux vues présentées à l'utilisateur. Celui-ci par le biais d'un lien ou d'un bouton va passer à une autre vue. L'application web a changé d'état. Nous avons vu qu'une action était initiée par une requête du genre **http://machine:port/chemin/main.php?action=XX**. Cette URL doit provenir d'un lien contenu dans la vue présentée à l'utilisateur. On veut en effet éviter qu'un utilisateur tape directement l'URL **http://machine:port/chemin/main.php?action=XX** court-circuitant ainsi le cheminement que l'application a prévu pour lui. Ceci est vrai également si le client est un programme.

Un enchaînement sera correct si l'URL demandée est une URL qui peut être demandée à partir de la dernière vue présentée à l'utilisateur. La liste de celles-ci est simple à déterminer. Elle est constituée

- des URL contenues dans la vue soit sous forme de liens sois sous forme de cibles d'actions de type [submit]
- des URL qu'un utilisateur est autorisé à taper directement dans son navigateur lorsque la vue lui est présentée.

La liste des états de l'applications ne se confond pas nécessairement avec celle des vues. Considérons par exemple une vue élémentaire **erreurs.php** suivante :

```
Les erreurs suivantes se sont produites :
<ul>
 <?php
  for($i=0;$i<count($dReponse["erreurs"]);$i++){
    echo "<li class='erreur'>".$dReponse["erreurs"][$i]."</li>\n";
  }//for
 ?>
</ul>
<div class="info"><?php echo $dReponse["info"] ?></div>
<a href="<?php echo $dReponse["href"] ?>"><?php echo $dReponse["lien"] ?></a>
```

Cette vue élémentaire s'intègrera dans une composition de vues élémentaires qui formera la réponse. Sur cette vue, il y a un lien qui peut être positionné dynamiquement. La vue **erreurs.php** peut alors être affichée avec **n** liens différents selon les circonstances. Cela donnera naissance à **n** états différents pour l'application. Dans l'état n° **i**, la vue **erreurs.php** sera affichée avec le lien **lieni**. Dans cet état, seule l'utilisation du lien **lieni** est acceptable.

La liste des états d'une application et celle des actions possibles dans chaque état seront consignées dans le dictionnaire **$dConfig ['etats']** :

```
  // configuration des états de l'application
  $dConfig['etats']['e-formulaire']=array(
    'actionsautorisees'=>array('post:calculerimpot','get:init','post:effacerformulaire'),
     'vue'=>'e-formulaire.php');
  $dConfig['etats']['e-erreurs']=array(
    'actionsautorisees'=>array('get:retourformulaire','get:init'),
    'vue'=>'e-erreurs.php');
  $dConfig['etats']['sansetat']=array('actionsautorisees'=>array('get:init'));
```

L'application ci-dessus a deux états nommés : **e-formulaire** et **e-erreurs**. On rajoute un état appelé **sansetat** qui correspond au démarrage initial de l'application alors qu'elle n'avait pas d'état. Dans un état **E**, la liste des actions autorisées se trouve dans le tableau **$dConfig['etats'][E]['actionsautorisees']**. On y précise la méthode (get/post) autorisée pour l'action et le nom de celle-ci. Dans l'exemple ci-dessus, il y a quatre actions possibles : *get:init*, *post:alculerimpot*, *get:retourformulaire* et *post:effacerformulaire*.

Muni du dictionnaire **$dConfig['etats']**, le contrôleur peut dire si l'action **$sAction** en cours est autorisée ou non dans l'état actuel de l'application. Celui est construit par chaque action et stocké dans la session dans **$dSession['etat']**. Le code du contrôleur pour vérifier si l'action courante est autorisée ou non est le suivant :

```
.....
 // l'enchaînement des actions est-il normal ?
  if( ! enchainementOK($dConfig,$dSession,$sAction)){
    // enchaînement anormal
    $sAction='enchainementinvalide';
  }//if

 // traitement de l'action
  $scriptAction=$dConfig['actions'][$sAction] ?
    $dConfig['actions'][$sAction]['url'] :
    $dConfig['actions']['actioninvalide']['url'];
  include $scriptAction;
..........
  //---------------------------------------------------------------------
 function enchainementOK(&$dConfig,&$dSession,$sAction){
    // vérifie si l'action courante est autorisée vis à vis de l'état précédent
    $etat=$dSession['etat']['principal'];
    if(! isset($etat)) $etat='sansetat';

    // vérification action
    $actionsautorisees=$dConfig['etats'][$etat]['actionsautorisees'];
    $autorise= ! isset($actionsautorisees) || in_array($sAction,$actionsautorisees);
  return $autorise;
  }
```

La logique est la suivante : une action **$sAction** est autorisée si elle se trouve dans la liste **$dConfig['etats'][$etat] ['actionsautorisees']** ou si cette liste est inexistante autorisant alors toute action. **$etat** est l'état de l'application à la fin du précédent cycle demande client/réponse serveur. Cet état a été stocké dans la session et est retrouvé là. Si on découvre que l'action demandée est illégale, on fait exécuter le script **$dConfig['actions']['enchainementinvalide']['url']**. Ce script se chargera de transmettre une réponse adéquate au client.

En phase de développement, on pourra ne pas remplir le dictionnaire **$dConfig['etats']**. Dans ce cas, tout état autorise toute action. On pourra mettre au point le dictionnaire lorsque l'application aura été totalement déboguée. Il protègera l'application d'actions non autorisées.

3.7 L'envoi de la réponse au client

Le script [**$scriptAction**] de traitement de l'action courante place dans **$dSession['etat']['principal']**, l'état dans lequel doit être placée l'application. Cet état est l'une des clés du dictionnaire des états **$dConfig['etats']**. Dans l'exemple suivant :

```
  // configuration des états de l'application
  $dConfig['etats']['e-formulaire']=array(
    'actionsautorisees'=>array('post:calculerimpot','get:init','post:effacerformulaire'),
     'vue'=>'e-formulaire.php');
  $dConfig['etats']['e-erreurs']=array(
    'actionsautorisees'=>array('get:retourformulaire','get:init'),
    'vue'=>'e-erreurs.php');
  $dConfig['etats']['sansetat']=array('actionsautorisees'=>array('get:init'));
```

On a deux états appelés [e-formulaire] et [e-erreurs]. A un état correspond un générateur de réponse et un seul. Ce générateur doit générer la réponse/vue à envoyer au client. Dans l'exemple ci-dessus, si un script d'action [**$scriptAction**] indique que l'application

doit être mise dans l'état [e-formulaire], alors le contrôleur demandera l'exécution du script [e-formulaire.php]. Ceci est réalisé par la séquence de code suivante dans le contrôleur :

```
// envoi de la réponse(vue) au client
$sEtat=$dSession['etat']['principal'];
$scriptVue=$dConfig['etats'][$sEtat]['vue'];
include $scriptVue;
```

Le générateur de vue [$scriptVue] va s'exécuter et préparer la vue à envoyer. Chaque générateur de vue se termine par un appel à la procédure [**finSession**] du contrôleur générique. Celle-ci a pour objectif :

- d'envoyer à l'utilisateur la vue préparée par le générateur [$scriptVue]
- d'enregistrer certaines informations dans la session
- de terminer l'exécution du contrôleur (exit)

Nous avons dit qu'une réponse pouvait avoir différents modèles de page. Ceux-ci sont placés par configuration dans **$dConfig ['vuesReponse']**. Dans une application à deux modèles, on pourrait ainsi avoir :

```
$dConfig['vuesReponse']['modele1']=array('url'=>'m-modele1.php');
$dConfig['vuesReponse']['modele2']=array('url'=>'m-modele2.php');
```

Le générateur de vue précise dans **$dReponse['vuereponse']** le modèle de la réponse désirée. Par exemple :

```
$dReponse['vueReponse']='modele1';
```

La réponse est envoyée au client par le contrôleur générique avec l'instruction :

```
// on présente la réponse
include $dConfig['vuesReponse'][$dReponse['vuereponse']]['url'];
```

Dans notre exemple, cela revient à écrire :

```
include 'm-modele1.php';
```

Le script [m-modele1.php] s'exécute. C'est une page PHP de présentation, sans logique de programmation mais avec des éléments dynamiques. Une fois cette page envoyée au client, le contrôleur s'arrête (exit). Le cycle demande-réponse du client-serveur est terminé.

3.8 Débogage

Le contrôleur propose deux fonction de débogage :

- la fonction **trace** permet d'afficher un message dans le flux html
- la fonction **dump** permet d'afficher le contenu d'un dictionnaire dans ce même flux

Tout script d'action pourra utiliser ces deux fonctions. En effet, le code du script d'action étant inclus (include) dans le code du contrôleur, les fonctions **trace** et **dump** seront visibles des scripts.

3.9 Conclusion

Le contrôleur générique vise à permettre au développeur de se concentrer sur les actions et les vues de son application. Il assure pour lui :

- la gestion de la session (restauration, sauvegarde)
- la vérification de la validité des actions demandées
- l'exécution du script associé à l'action
- l'envoi au client de la réponse adaptée à l'état résultat de l'exécution de l'action

4 Une application d'illustration

Nous nous proposons d'illustrer la méthode précédente avec un exemple de calcul d'impôts.

4.1 Le problème

on désire écrire une application permettant à un utilisateur de calculer son impôt sur le web. On se place dans le cas simplifié d'un contribuable n'ayant que son seul salaire à déclarer (chiffres 2004 pour revenus 2003) :

- on calcule le nombre de parts du salarié **nbParts=nbEnfants/2 +1** s'il n'est pas marié, **nbEnfants/2+2** s'il est marié, où *nbEnfants* est son nombre d'enfants.
- s'il a au moins trois enfants, il a une demi part de plus
- on calcule son revenu imposable **R=0.72*S** où S est son salaire annuel
- on calcule son coefficient familial **QF=R/nbParts**
- on calcule son impôt **I**. Considérons le tableau suivant :

4262	0	0
8382	0.0683	291.09
14753	0.1914	1322.92
23888	0.2826	2668.39
38868	0.3738	4846.98
47932	0.4262	6883.66
0	0.4809	9505.54

Chaque ligne a 3 champs. Pour calculer l'impôt I, on recherche la première ligne où *QF<=champ1*. Par exemple, si QF=5000 on trouvera la ligne

8382 0.0683 291.09

L'impôt I est alors égal à **0.0683*R - 291.09*nbParts**. Si QF est tel que la relation *QF<=champ1* n'est jamais vérifiée, alors ce sont les coefficients de la dernière ligne qui sont utilisés. Ici :

0 0.4809 9505.54

ce qui donne l'impôt I=0.4809*R - 9505.54*nbParts.

4.2 La base de données

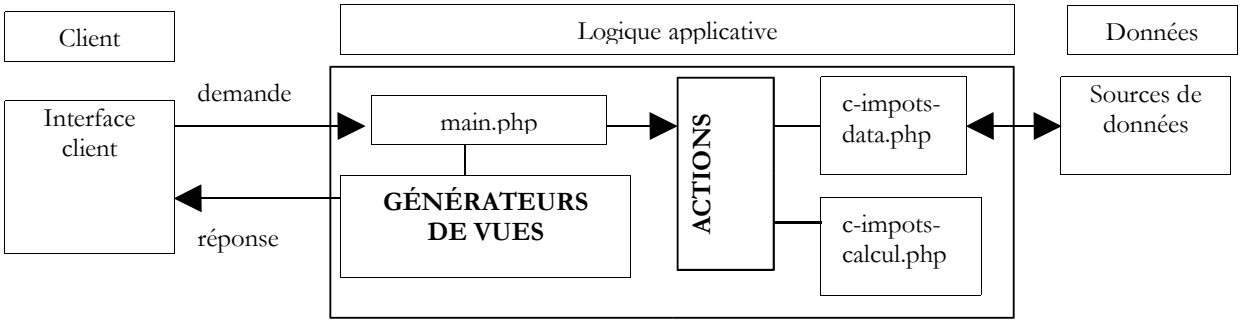
Les données précédentes sont enregistrées dans une base MySQL appelée **dbimpots.** L'utilisateur **seldbimpots** de mot de passe **mdpseldbimpots** a un accès en lecture seule au contenu de la base. Celle-ci a une unique table appelée **impots** dont la structure et le contenu sont les suivants :

Field	Type
limites	double
coeffr	double
coeffn	double

limites	coeffr	coeffn
4262	0	0
8382	0.0683	291.09
14753	0.1914	1322.92
23888	0.2826	2268.39
38868	0.3738	4846.98
47932	0.4262	6883.66
0	0.4809	9505.54

4.3 L'architecture MVC de l'application

L'application aura l'architecture MVC suivante :



- le contrôleur **main.php** sera le contrôleur générique exposé précédemment
- la demande du client est envoyée au contrôleur sous la forme d'une requête de la forme **main.php?action=xx**. La valeur du paramètre action détermine le script du bloc **ACTIONS** à exécuter. Le script d'action exécuté rend au contrôleur une variable indiquant l'état dans lequel doit être placée l'application web. Muni de cet état, le contrôleur va activer l'un des générateurs de vues pour envoyer la réponse au client.
- **c-impots-data.php** est la classe chargée de fournir au contrôleur les données dont il a besoin
- **c-impots-calcul.php** est la classe métier permettant le calcul de l'impôt

4.4 La classe d'accès aux données

La classe d'accès aux données est construite pour cacher à l'application web la provenance des données. Dans son interface, on trouve une méthode **getData** qui délivre les trois tableaux de données nécessaires au calcul de l'impôt. Dans notre exemple, les données sont prises dans une base de données MySQL. Afin de rendre la classe indépendante du type réel du SGBD, nous utiliserons la bibliothèque **pear::DB** décrite en annexe. Le code [**c-impots-data.php**] de la classe est le suivant :

```
<?php

  // bibliothèques
  require_once 'DB.php';

  class impots_data{
    // classe d'accès à la source de données DBIMPOTS

    // attributs
     var $sDSN;          // la chaîne de connexion
    var $sDatabase;     // le nom de la base
     var $oDB;           // connexion à la base
     var $aErreurs;      // liste d'erreurs
     var $oRésultats;    // résultat d'une requête
    var $connecté;      // booléen qui indique si on est connecté ou non à la base
    var $sQuery;        // la dernière requête exécutée

     // constructeur
     function impots_data($dDSN){

       // $dDSN : dictionnaire définissant la liaison à établir
        // $dDSN['sgbd'] : le type du SGBD auquel il faut se connecter
        // $dDSN['host'] : le nom de la machine hôte qui l'héberge
        // $dDSN['database'] : le nom de la base à laquelle il faut se connecter
        // $dDSN['user'] : un utilisateur de la bse
        // $dDSN['mdp'] : son mot de passe

        // crée dans $oDB une connexion à la base définie par $dDSN sous l'identité de $dDSN['user']
        // si la connexion réussit
         // met dans $sDSN la chaîne de connexion à la base
         // met dans $sDataBase le nom de la base à laquelle on se connecte
          // met $connecté à vrai
        // si la connexion échoue
         // met les msg d'erreurs adéquats dans la liste $aErreurs
          // ferme la connexion si besoin est
          // met $connecté à faux

       // raz liste des erreurs
      $this->aErreurs=array();

       // on crée une connexion à la base $sDSN
                $this->sDSN=$dDSN["sgbd"].":".$dDSN["user"].":".$dDSN["mdp"]."@".$dDSN["host"]."/".$dDSN
["database"];
       $this->sDatabase=$dDSN["database"];
       $this->connect();

       // connecté ?
       if( ! $this->connecté) return;
```

```
      // la connexion a réussi
      $this->connecté=TRUE;
    }//constructeur

    // ------------------------------------------------------------------
    function connect(){
      // (re)connexion à la base
      // raz liste des erreurs
      $this->aErreurs=array();

      // on crée une connexion à la base $sDSN
      $this->oDB=DB::connect($this->sDSN,true);

      // erreur ?
      if(DB::iserror($this->oDB)){
        // on note l'erreur
        $this->aErreurs[]="Echec de la connexion à la base [".$this->sDatabase."] : [".$this->oDB->getMessage()."]";
        // la connexion a échoué
        $this->connecté=FALSE;
        // fin
        return;
      }

      // on est connecté
      $this->connecté=TRUE;
    }//connect

    // ------------------------------------------------------------------
    function disconnect(){
      // si on est connecté, on ferme la connexion à la base $sDSN
      if($this->connecté){
        $this->oDB->disconnect();
        // on est déconnecté
        $this->connecté=FALSE;
      }//if
    }//disconnect

    // ------------------------------------------------------------------
    function execute($sQuery){
      // $sQuery : requête à exécuter

      // on mémorise la requête
      $this->sQuery=$sQuery;

      // est-on connecté ?
      if(! $this->connecté){
        // on note l'erreur
        $this->aErreurs[]="Pas de connexion existante à la base [$this->sDatabase]";
        // fin
        return;
      }//if

      // exécution de la requête
      $this->oRésultats=$this->oDB->query($sQuery);

      // erreur ?
      if(DB::iserror($this->oRésultats)){
        // on note l'erreur
        $this->aErreurs[]="Echec de la requête [$sQuery] : [".$this->oRésultats->getMessage()."]";
        // retour
        return;
      }//if
    }//execute

    // ------------------------------------------------------------------
    function getData(){
      // on récupère les 3 séries de données limites, coeffr, coeffn
      $this->execute('select limites, coeffR, coeffN from impots');
      // des erreurs ?
      if(count($this->aErreurs)!=0) return array();
      // on parcourt le résultat du select
      while ($ligne = $this->oRésultats->fetchRow(DB_FETCHMODE_ASSOC)) {
        $limites[]=$ligne['limites'];
        $coeffr[]=$ligne['coeffR'];
        $coeffn[]=$ligne['coeffN'];
      }//while
      return array($limites,$coeffr,$coeffn);
    }//getDataImpots

  }//classe
?>
```

Un programme de test [**t-impots-data.php**] pourrait être le suivant :

```
<?php
```

```
  // bibliothèque
  require_once "c-impots-data.php";
  require_once "DB.php";

  // test de la classe impots-data
  ini_set('track_errors','on');
  ini_set('display_errors','on');

  // configuration base dbimpots
  $dDSN=array(
    "sgbd"=>"mysql",
    "user"=>"seldbimpots",
    "mdp"=>"mdpseldbimpots",
    "host"=>"localhost",
    "database"=>"dbimpots"
  );

  // ouverture de la session
  $oImpots=new impots_data($dDSN);
  // erreurs ?
  if(checkErreurs($oImpots)){
    exit(0);
  }
  // suivi
  echo "Connecté à la base...\n";

  // récupération des données limites, coeffr, coeffn
  list($limites,$coeffr,$coeffn)=$oImpots->getData();
  // erreurs ?
  if( ! checkErreurs($oImpots)){
    // contenu
    echo "données : \n";
    for($i=0;$i<count($limites);$i++){
      echo "[$limites[$i],$coeffr[$i],$coeffn[$i]]\n";
    }//for
  }//if

  // on se déconnecte
  $oImpots->disconnect();
  // suivi
  echo "Déconnecté de la base...\n";
  // fin
  exit(0);

  // ----------------------------------
  function checkErreurs(&$oImpots){
    // des erreurs ?
    if(count($oImpots->aErreurs)!=0){
      // affichage
      for($i=0;$i<count($oImpots->aErreurs);$i++){
        echo $oImpots->aErreurs[$i]."\n";
      }//for
      // des erreurs
      return true;
    }//if
    // pas d'erreurs
    return false;
  }//checkErreurs

?>
```

L'exécution de ce programme de test donne les résultats suivants :

```
Connecté à la base...
données :
[4262,0,0]
[8382,0.0683,291.09]
[14753,0.1914,1322.92]
[23888,0.2826,2668.39]
[38868,0.3738,4846.98]
[47932,0.4262,6883.66]
[0,0.4809,9505.54]
Déconnecté de la base...
```

4.5 La classe de calcul de l'impôt

Cette classe permet de calculer l'impôt d'un contribuable. On fournit à son constructeur les données permettant ce calcul. Il se charge ensuite de calculer l'impôt correspondant. Le code [**c-impots-calcul.php**] de la classe est le suivant :

```
<?php
```

```
  class impots_calcul{
    // classe de calcul de l'impôt

    // constructeur
    function impots_calcul(&$perso,&$data){
      // $perso : dictionnaire avec les clés suivantes
      // enfants(e) : nbre d'enfants
      // salaire(e) : salaire annuel
      // marié(e) : booléen indiquant si le contribuable est marié ou non
      // impot(s) : impôt à payer calculé par ce constructeur
      // $data : dictionnaire avec les clés suivantes
      // limites : tableau des limites de tranches
      // coeffr : tableau des coefficients du revenu
      // coeffn tableau des coefficients du nombre de parts
      // les 3 tableaux ont le même nbre d'éléments

      // calcul du nombre de parts
      if($perso['marié'])
        $nbParts=$perso['enfants']/2+2;
        else $nbParts=$perso['enfants']/2+1;
      if ($perso['enfants']>=3) $nbParts+=0.5;

      // revenu imposable
      $revenu=0.72*$perso['salaire'];

      // quotient familial
      $QF=$revenu/$nbParts;

      // recherche de la tranche d'impots correspondant à QF
      $nbTranches=count($data['limites']);
      $i=0;
      while($i<$nbTranches-2 && $QF>$data['limites'][$i]) $i++;

      // l'impôt
      $perso['impot']=floor($data['coeffr'][$i]*$revenu-$data['coeffn'][$i]*$nbParts);
    }//constructeur
  }//classe
?>
```

Un programme de test [**t-impots-calcul.php**] pourrait être le suivant :

```
<?php

  // bibliothèque
  require_once "c-impots-data.php";
  require_once "c-impots-calcul.php";

  // configuration base dbimpots
  $dDSN=array(
    "sgbd"=>"mysql",
    "user"=>"seldbimpots",
    "mdp"=>"mdpseldbimpots",
    "host"=>"localhost",
    "database"=>"dbimpots"
  );

  // ouverture de la session
  $oImpots=new impots_data($dDSN);
  // erreurs ?
  if(checkErreurs($oImpots)){
    exit(0);
  }
  // suivi
  echo "Connecté à la base...\n";
  // récupération des données limites, coeffr, coeffn
  list($limites,$coeffr,$coeffn)=$oImpots->getData();
  // erreurs ?
  if(checkErreurs($oImpots)){
    exit(0);
  }
  // on se déconnecte
  $oImpots->disconnect();
  // suivi
  echo "Déconnecté de la base...\n";

  // calcul d'un impôt
  $dData=array('limites'=>&$limites,'coeffr'=>&$coeffr,'coeffn'=>&$coeffn);
  $dPerso=array('enfants'=>2,'salaire'=>60000,'marié'=>true,'impot'=>0);
  new impots_calcul($dPerso,$dData);
  dump($dPerso);
  $dPerso=array('enfants'=>2,'salaire'=>60000,'marié'=>false,'impot'=>0);
  new impots_calcul($dPerso,$dData);
  dump($dPerso);
  $dPerso=array('enfants'=>3,'salaire'=>60000,'marié'=>false,'impot'=>0);
  new impots_calcul($dPerso,$dData);
```

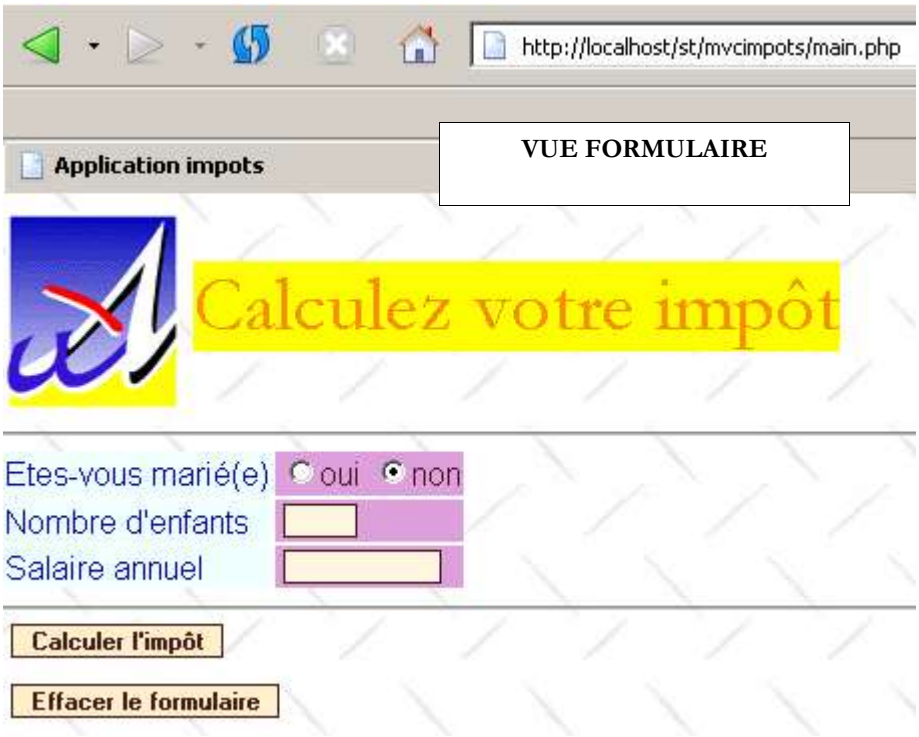
```
  dump($dPerso);
  $dPerso=array('enfants'=>3,'salaire'=>60000,'marié'=>true,'impot'=>0);
  new impots_calcul($dPerso,$dData);
  dump($dPerso);

  // fin
  exit(0);

  // ---------------------------------
  function checkErreurs(&$oImpots){
    // des erreurs ?
    if(count($oImpots->aErreurs)!=0){
      // affichage
      for($i=0;$i<count($oImpots->aErreurs);$i++){
        echo $oImpots->aErreurs[$i]."\n";
      }//for
      // des erreurs
      return true;
    }//if
    // pas d'erreurs
    return false;
  }//checkErreurs

?>
```

L'exécution de ce programme de tests donne les résultats suivants :

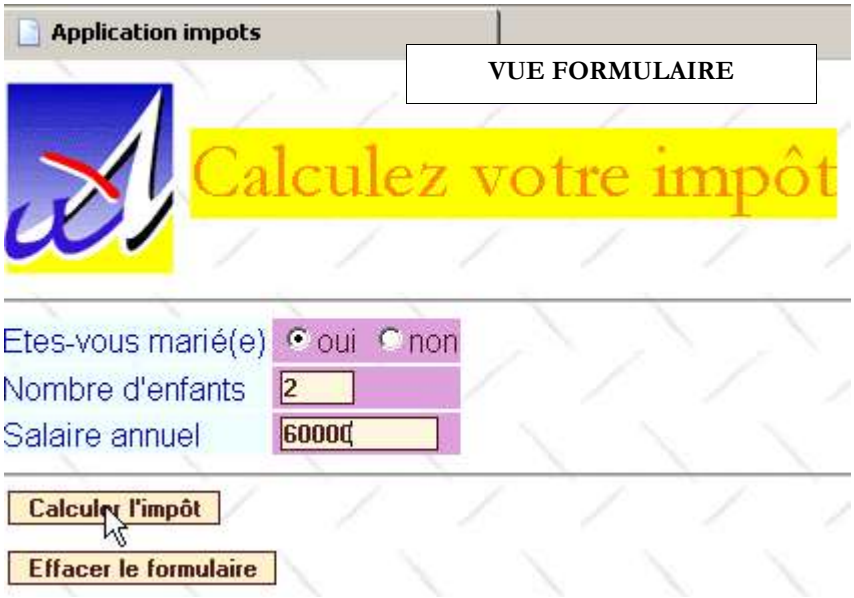
```
Connecté à la base...
Déconnecté de la base...
[enfants,2] [salaire,60000] [marié,1] [impot,4299]
[enfants,2] [salaire,60000] [marié,0] [impot,6871]
[enfants,3] [salaire,60000] [marié,0] [impot,4299]
[enfants,3] [salaire,60000] [marié,1] [impot,2976]
```

4.6 Le fonctionnement de l'application

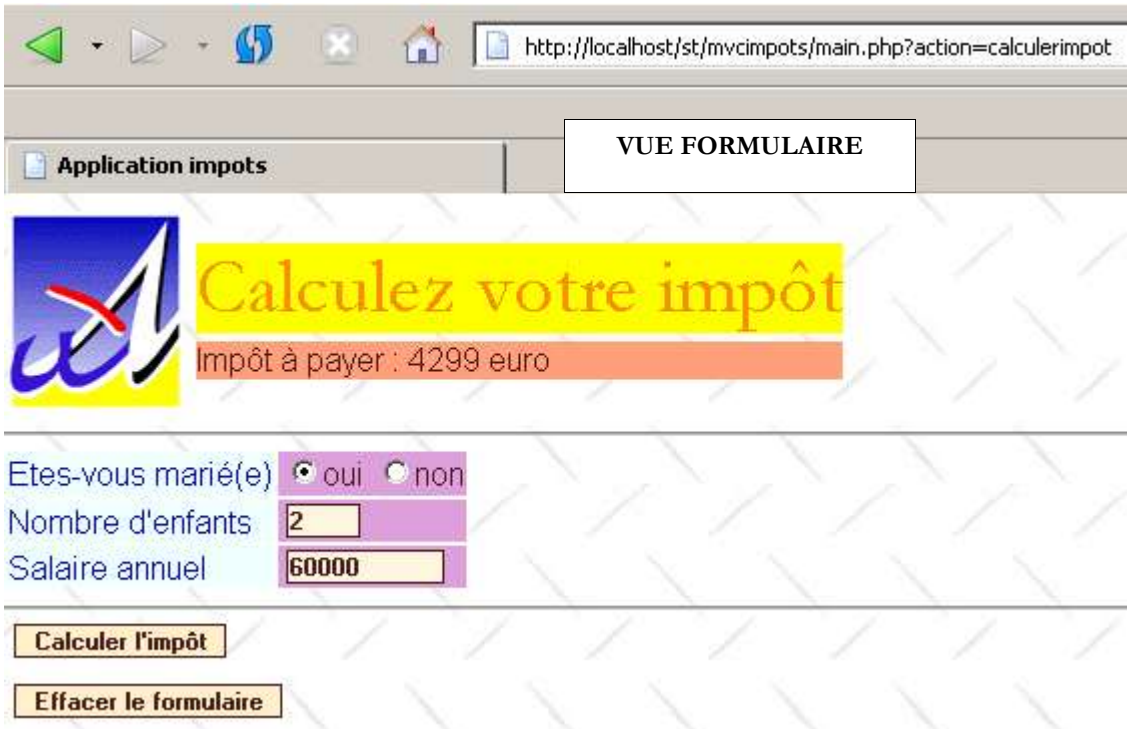
Lorsque l'application web de calcul de l'impôt est lancée, on obtient la vue **[v-formulaire]** suivante :



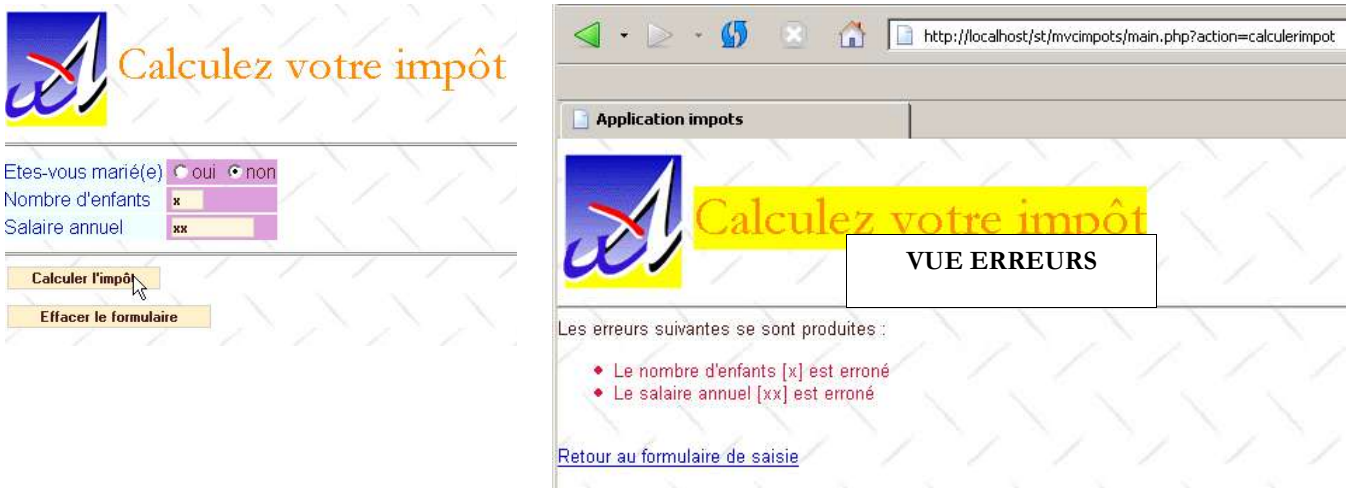
L'utilisateur remplit les champs et demande le calcul de l'impôt :



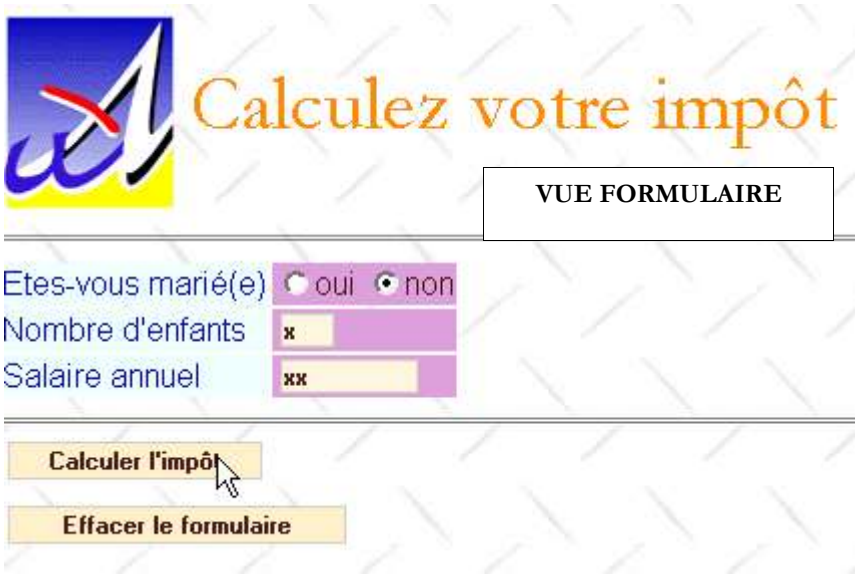
Il obtient la réponse suivante :



On remarquera que le formulaire est régénéré dans l'état où l'utilisateur l'a validé et qu'il affiche de plus le montant de l'impôt à payer. L'utilisateur peut faire des erreurs de saisie. Celles-ci lui sont signalées par une page d'erreurs qu'on appelera la vue **[v-erreurs]**.



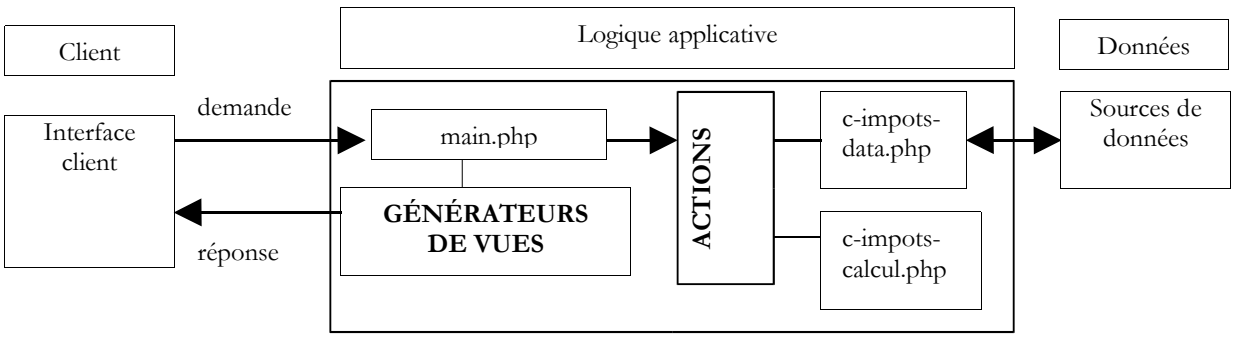
Le lien [Retour au formulaire de saisie] permet à l'utilisateur de retrouver le formulaire tel qu'il l'a validé.



Enfin le bouton [Effacer le formulaire] remet le formulaire dans son état initial, c.a.d. tel que l'utilisateur l'a reçu lors de la demande initiale. C'est un simple bouton HTML de type [Reset].

4.7 Retour sur l'architecture MVC de l'application

L'application a l'architecture MVC suivante :



Nous venons de décrire les deux classes **c-impots-data.php** et **c-impots-calcul.php**. Nous décrivons maintenant les autres éléments de l'architecture de notre application.

4.8 Le contrôleur de l'application

Le contrôleur **main.php** de l'application est celui qui a été décrit dans la première partie de ce document. C'est un contrôleur générique indépendant de l'application.

```
<?php
  // contrôleur générique

  // lecture config
  include 'config.php';

  // inclusion de bibliothèques
  for($i=0;$i<count($dConfig['includes']);$i++){
    include($dConfig['includes'][$i]);
  }//for

  // on démarre ou reprend la session
  session_start();
  $dSession=$_SESSION["session"];
  if($dSession) $dSession=unserialize($dSession);

  // on récupère l'action à entreprendre
  $sAction=$_GET['action'] ? strtolower($_GET['action']) : 'init';
  $sAction=strtolower($_SERVER['REQUEST_METHOD']).":$sAction";

  // l'enchaînement des actions est-il normal ?
  if( ! enchainementOK($dConfig,$dSession,$sAction)){
    // enchaînement anormal
    $sAction='enchainementinvalide';
  }//if

  // traitement de l'action
  $scriptAction=$dConfig['actions'][$sAction] ?
    $dConfig['actions'][$sAction]['url'] :
    $dConfig['actions']['actioninvalide']['url'];
  include $scriptAction;


  // envoi de la réponse(vue) au client
  $sEtat=$dSession['etat']['principal'];
  $scriptVue=$dConfig['etats'][$sEtat]['vue'];
  include $scriptVue;

  // fin du script - on ne devrait pas arriver là sauf bogue
  trace ("Erreur de configuration.");
  trace("Action=[$sAction]");
  trace("scriptAction=[$scriptAction]");
  trace("Etat=[$sEtat]");
  trace("scriptVue=[$scriptVue]");
   trace ("Vérifiez que les script existent et que le script [$scriptVue] se termine par l'appel à
finSession.");
  exit(0);

  // ------------------------------------------------------------------
  function finSession(&$dConfig,&$dReponse,&$dSession){
    // $dConfig : dictionnaire de configuration
    // $dSession : dictionnaire contenant les infos de session
    // $dReponse : le dictionnaire des arguments de la page de réponse

    // enregistrement de la session
    if(isset($dSession)){
      // on met les paramètres de la requête dans la session
      $dSession['requete']=strtolower($_SERVER['REQUEST_METHOD'])=='get' ? $_GET :
        strtolower($_SERVER['REQUEST_METHOD'])=='post' ? $_POST : array();
     $_SESSION['session']=serialize($dSession);
      session_write_close();
    }else{
      // pas de session
      session_destroy();
    }

   // on présente la réponse
   include $dConfig['vuesReponse'][$dReponse['vuereponse']]['url'];

    // fin du script
    exit(0);
  }//finsession

  //-------------------------------------------------------------------
  function enchainementOK(&$dConfig,&$dSession,$sAction){
    // vérifie si l'action courante est autorisée vis à vis de l'état précédent
    $etat=$dSession['etat']['principal'];
    if(! isset($etat)) $etat='sansetat';
```

```
    // vérification action
    $actionsautorisees=$dConfig['etats'][$etat]['actionsautorisees'];
    $autorise= ! isset($actionsautorisees) || in_array($sAction,$actionsautorisees);
    return $autorise;
  }

  //---------------------------------------------------------------------
  function dump($dInfos){
    // affiche un dictionnaire d'informations
    while(list($clé,$valeur)=each($dInfos)){
      echo "[$clé,$valeur]<br>\n";
    }//while
  }//suivi

  //---------------------------------------------------------------------
  function trace($msg){
    echo $msg."<br>\n";
  }//suivi
?>
```

4.9 Les actions de l'application web

Il y a quatre actions :

1. **get:init** : c'est l'action qui est déclenchée lors de la demande initiale sans paramètres au contrôleur. Elle génère la vue [**v-formulaire**] vide.
2. **post:effacerformulaire** : action déclenchée par le bouton [Effacer le formulaire]. Elle génère la vue [**v-formulaire**] vide.
3. **post:calculerimpot** : action déclenchée par le bouton [Calculer l'impôt]. Elle génère soit la vue [**v-formulaire**] avec le montant de l'impôt à payer, soit la vue **[v-erreurs]**.
4. **get:retourformulaire** : action déclenchée par le lien [Retour au formulaire de saisie]. Elle génère la vue **[v-formulaire]** pré-remplie avec les données erronées.

Ces actions sont configurées de la façon suivante dans le fichier de configuration :

```
// configuration des actions de l'application
$dConfig['actions']['get:init']=array('url'=>'a-init.php');
$dConfig['actions']['post:calculerimpot']=array('url'=>'a-calculimpot.php');
$dConfig['actions']['get:retourformulaire']=array('url'=>'a-retourformulaire.php');
$dConfig['actions']['post:effacerformulaire']=array('url'=>'a-init.php');
$dConfig['actions']['enchainementinvalide']=array('url'=>'a-enchainementinvalide.php');
$dConfig['actions']['actioninvalide']=array('url'=>'a-actioninvalide.php');
```

A chaque action, est associé le script chargé de la traiter. Chaque action va emmener l'application web dans un état enregistré dans l'élément **$dSession['etat']['principal']**. Cet état est destiné à être sauvegardé dans la session et c'est pourquoi il est placé dans le dictionnaire $**dSession**. Par ailleurs, l'action enregistre dans le dictionnaire **$dReponse**, les informations utiles pour l'affichage de la vue liée au nouvel état dans lequel va se trouver l'application.

4.10 Les états de l'application web

Il y en a deux :

- **[e-formulaire]** : état où sont présentées les différentes variantes de la vue [**v-formulaire]**.
- **[e-erreurs]** : état où est présentée la vue **[v-erreurs]**.

Les actions autorisées dans ces états sont les suivantes :

```
// configuration des états de l'application
$dConfig['etats']['formulaire']=array(
  'actionsautorisees'=>array('post:calculerimpot','get:init','post:effacerformulaire'),
   'vue'=>'e-formulaire.php');
$dConfig['etats']['erreurs']=array(
  'actionsautorisees'=>array('get:retourformulaire','get:init'),
  'vue'=>'e-erreurs.php');
$dConfig['etats']['sansetat']=array('actionsautorisees'=>array('get:init'));
```

Dans un état, les actions autorisées correspondent aux URL cibles des liens ou boutons [submit] de la vue associée à l'état. De plus, l'action '**get:init**' est tout le temps autorisée. Cela permet à l'utilisateur de récupérer l'URL **main.php** dans la liste des URL de son navigateur et de la rejouer quelque soit l'état de l'application. C'est une sorte de réinitialisation 'à la main'. L'état **'sansetat'** n'existe qu'au démarrage de l'application.

A chaque état de l'application est associé un script chargé de générer la vue associée à l'état :

- état **[e-formulaire]** : script **e-formulaire.php**

- état **[e-erreurs]** : script **e-erreurs.php**

L'état **[e-formulaire]** va présenter la vue [**v-formulaire]** avec des variantes. En effet, la vue **[v-formulaire]** peut être présentée vide ou pré-remplie ou avec le montant de l'impôt. L'action qui amènera l'application dans l'état [**e-formulaire]** précise dans la variable **$dSession['etat']['principal']** l'état principal de l'application. Le contrôleur n'utilise que cette information. Dans notre application, l'action qui amène à l'état **[e-formulaire]** ajoutera dans **$dSession['etat']['secondaire']** une information complémentaire permettant au générateur de la réponse de savoir s'il doit générer un formulaire vide, pré-rempli, avec ou sans le montant de l'impôt. On aurait pu procéder différemment en estimant qu'il y avait là trois états différents et donc trois générateurs de vue à écrire.

4.11 Le fichier config.php de configuration de l'application web

```
<?php

  // configuration de php
  ini_set("register_globals","off");
  ini_set("display_errors","off");
  ini_set("expose_php","off");

  // liste des modules à inclure
  $dConfig['includes']=array('c-impots-data.php','c-impots-calcul.php');

  // contrôleur de l'application
  $dConfig['webapp']=array('titre'=>"Calculez votre impôt");

  // configuration des vues de l'aplication
  $dConfig['vuesReponse']['modele1']=array('url'=>'m-reponse.php');
  $dConfig['vuesReponse']['modele2']=array('url'=>'m-reponse2.php');
  $dConfig['vues']['formulaire']=array('url'=>'v-formulaire.php');
  $dConfig['vues']['erreurs']=array('url'=>'v-erreurs.php');
  $dConfig['vues']['formulaire2']=array('url'=>'v-formulaire2.php');
  $dConfig['vues']['erreurs2']=array('url'=>'v-erreurs2.php');
  $dConfig['vues']['bandeau']=array('url'=>'v-bandeau.php');
  $dConfig['vues']['menu']=array('url'=>'v-menu.php');
  $dConfig['style']['url']='style1.css';

  // configuration des actions de l'application
  $dConfig['actions']['get:init']=array('url'=>'a-init.php');
  $dConfig['actions']['post:calculerimpot']=array('url'=>'a-calculimpot.php');
  $dConfig['actions']['get:retourformulaire']=array('url'=>'a-retourformulaire.php');
  $dConfig['actions']['post:effacerformulaire']=array('url'=>'a-init.php');
  $dConfig['actions']['enchainementinvalide']=array('url'=>'a-enchainementinvalide.php');
  $dConfig['actions']['actioninvalide']=array('url'=>'a-actioninvalide.php');

  // configuration des états de l'application
  $dConfig['etats']['e-formulaire']=array(
    'actionsautorisees'=>array('post:calculerimpot','get:init','post:effacerformulaire'),
     'vue'=>'e-formulaire.php');
  $dConfig['etats']['e-erreurs']=array(
    'actionsautorisees'=>array('get:retourformulaire','get:init'),
    'vue'=>'e-erreurs.php');
  $dConfig['etats']['sansetat']=array('actionsautorisees'=>array('get:init'));

  // configuration modèle de l'application
  $dConfig["DSN"]=array(
    "sgbd"=>"mysql",
    "user"=>"seldbimpots",
    "mdp"=>"mdpseldbimpots",
    "host"=>"localhost",
    "database"=>"dbimpots"
  );
?>
```

4.12 Les actions de l'application web

4.12.1 Fonctionnement général des scripts d'action

- Un script d'action est appelé par le contrôleur selon le paramètre **action** que celui-ci a reçu du client.
- Après exécution, le script d'action doit indiquer au contrôleur l'état dans lequel placer l'application. Cet état doit être indiqué dans **$dSession['etat']['principal']**.
- Un script d'action peut vouloir placer des informations dans la session. Il le fait en plaçant celles-ci dans le dictionnaire **$dSession**, ce dictionnaire étant automatiquement sauvegardé dans la session par le contrôleur à la fin du cycle demande-réponse.

- Un script d'action peut avoir des informations à passer aux vues. Cet aspect est indépendant du contrôleur. Il s'agit de l'interface entre les actions et les vues, interface propre à chaque application. Dans l'exemple étudié ici, les actions fourniront des informations aux générateurs de vues via un dictionnaire appelé **$dReponse**.

4.12.2 L'action get:init

C'est l'action qui génère le formulaire vide. Le fichier de configuration montre qu'elle sera traitée par le script **a-init.php** :

```
  $dConfig['actions']['get:init']=array('url'=>'a-init.php');
```

Le code du script **a-init.php** est le suivant :

```
<?php
  // on affiche le formulaire de saisie
  $dSession['etat']=array('principal'=>'e-formulaire', 'secondaire'=>'init');
?>
```

Ce script se contente de fixer dans **$dSession['etat']['principal']** l'état dans lequel doit se trouver l'application, l'état **[e-formulaire]** et amène dans **$dSession['etat']['secondaire']** une précision sur cet état. Le fichier de configuration nous montre que le contrôleur exécutera le script **e-formulaire.php** pour générer la réponse au client.

```
  $dConfig['etats']['e-formulaire']=array(
    'actionsautorisees'=>array('post:calculerimpot','get:init','post:effacerformulaire'),
     'vue'=>'e-formulaire.php');
```

Le script **e-formulaire.php** génèrera la vue [**v-formulaire]** dans sa variante [init], c.a.d. le formulaire vide.

4.12.3 L'action post:calculerimpot

C'est l'action qui permet de calculer l'impôt à partir des données saisies dans le formulaire. Le fichier de configuration indique que c'est le script **a-calculimpot.php** qui va traiter cette action :

```
  $dConfig['actions']['post:calculerimpot']=array('url'=>'a-calculimpot.php');
```

Le code du script **a-calculimpot.php** est le suivant :

```
<?php
  // requête de demande de calcul d'impôt

  // on vérifie d'abord la validité des paramètres
  $sOptMarie=$_POST['optmarie'];
  if($sOptMarie!='oui' && $sOptMarie!='non'){
    $erreurs[]="L'état marital [$sOptMarie] est erroné";
  }
  $sEnfants=trim($_POST['txtenfants']);
  if(! preg_match('/^\d{1,3}$/',$sEnfants)){
    $erreurs[]="Le nombre d'enfants [$sEnfants] est erroné";
  }
  $sSalaire=trim($_POST['txtsalaire']);
  if(! preg_match('/^\d+$/',$sSalaire)){
    $erreurs[]="Le salaire annuel [$sSalaire] est erroné";
  }

  // s'il y a des erreurs, c'est fini
  if(count($erreurs)!=0){
    // préparation de la page d'erreurs
     $dReponse['erreurs']=&$erreurs;
     $dSession['etat']=array('principal'=>'e-erreurs','secondaire'=>'saisie');
    return;
  }//if

  // les données saisies sont correctes
  // on récupère les données nécessaires au calcul de l'impôt
  if(! $dSession['limites']){
    // les données ne sont pas dans la session
     // on les récupère dans la source de données
     list($erreurs,$limites,$coeffr,$coeffn)=getData($dConfig['DSN']);
     // s'il y a des erreurs, on affiche la page d'erreurs
     if(count($erreurs)!=0){
       // préparation de la page d'erreurs
        $dReponse['erreurs']=&$erreurs;
       $dSession['etat']=array('principal'=>'e-erreurs','secondaire'=>'database');
        return;
     }//if
     // pas d'erreurs - les données sont mises dans la session
     $dSession['limites']=&$limites;
     $dSession['coeffr']=&$coeffr;
     $dSession['coeffn']=&$coeffn;
  }//if
```

```
// ici on a les données nécessaire au calcul de l'impôt
// on calcule celui-ci
$dData=array('limites'=>&$dSession['limites'],
  'coeffr'=>&$dSession['coeffr'],
   'coeffn'=>&$dSession['coeffn']);
$dPerso=array('enfants'=>$sEnfants,'salaire'=>$sSalaire,'marié'=>($sOptMarie=='oui'),'impot'=>0);
new impots_calcul($dPerso,$dData);

// préparation de la page réponse
$dSession['etat']=array('principal'=>'e-formulaire','secondaire'=>'calculimpot');
$dReponse['impot']=$dPerso['impot'];
return;

//---------------------------------------------------------------------
function getData($dDSN){
  // connexion à la source de données définie par le dictionnaire $dDSN
  $oImpots=new impots_data($dDSN);
   if(count($oImpots->aErreurs)!=0) return array($oImpots->aErreurs);
   // récupération des données limites, coeffr, coeffn
  list($limites,$coeffr,$coeffn)=$oImpots->getData();
  // on se déconnecte
  $oImpots->disconnect();
   // on rend le résultat
   if(count($oImpots->aErreurs)!=0) return array($oImpots->aErreurs);
    else return array(array(),$limites,$coeffr,$coeffn);
}//getData
```

Le script fait ce qu'il a à faire : calculer l'impôt. Nous laissons au lecteur le soin de décrypter le code du traitement. Nous nous intéressons aux états qui peuvent survenir à la suite de cette action :

- les données saisies sont incorrectes ou l'accès aux données se passe mal : l'application est mise dans l'état **[e-erreurs]**. Le fichier de configuration montre que c'est le script **e-erreurs.php** qui sera chargé de générer la vue réponse :

```
  $dConfig['etats']['e-erreurs']=array(
    'actionsautorisees'=>array('get:retourformulaire','get:init'),
    'vue'=>'e-erreurs.php');
```

- dans tous les autres cas, l'application est placée dans l'état **[e-formulaire]** avec la variante **calculimpot** indiquée dans **$dSession['etat']['secondaire']**. Le fichier de configuration montre que c'est le script **e-formulaire.php** qui va générer la vue réponse. Il utilisera la valeur de **$dSession['etat']['secondaire']** pour générer un formulaire pré-rempli avec les valeurs saisies par l'utilisateur et de plus le montant de l'impôt.

4.12.4 L'action get:effacerformulaire

Elle est associée par configuration au script **a-init.php** déjà décrit.

```
  $dConfig['actions']['post:effacerformulaire']=array('url'=>'a-init.php');
```

4.12.5 L'action get:retourformulaire

Elle permet de revenir à l'état **[e-formulaire]** à partir de l'état **[e-erreurs]**. C'est le script [**a-retourformulaire.php]** qui traite cette action :

```
  $dConfig['actions']['get:retourformulaire']=array('url'=>'a-retourformulaire.php');
```

Le script **a-retourformulaire.php** est le suivant :

```
<?php
  // on affiche le formulaire de saisie
  $dSession['etat']=array('principal'=>'e-formulaire','secondaire'=>'retourformulaire');
?>
```

On demande simplement à ce que l'application soit placé dans l'état **[e-formulaire]** dans sa variante [**retourformulaire**]. Le fichier de configuration nous montre que le contrôleur exécutera le script **e-formulaire.php** pour générer la réponse au client.

```
  $dConfig['etats']['e-formulaire']=array(
    'actionsautorisees'=>array('post:calculerimpot','get:init','post:effacerformulaire'),
     'vue'=>'e-formulaire.php');
```

Le script **e-formulaire.php** génèrera la vue [**v-formulaire]** dans sa variante [**retourformulaire**], c.a.d. le formulaire pré-rempli avec les valeurs saisies par l'utilisateur mais sans le montant de l'impôt.

4.13 L'enchaînement d'actions invalide

Les actions valides à partir d'un état donné de l'application sont fixées par configuration :

```
  // configuration des états de l'application
  $dConfig['etats']['e-formulaire']=array(
    'actionsautorisees'=>array('post:calculerimpot','get:init','post:effacerformulaire'),
     'vue'=>'e-formulaire.php');
  $dConfig['etats']['e-erreurs']=array(
    'actionsautorisees'=>array('get:retourformulaire','get:init'),
    'vue'=>'e-erreurs.php');
  $dConfig['etats']['sansetat']=array('actionsautorisees'=>array('get:init'));
```

Nous avons déjà expliqué cette configuration. Si un enchaînement invalide d'actions est détecté, le script **[a-enchainementinvalide.php]** s'exécute :

```
  $dConfig['actions']['enchainementinvalide']=array('url'=>'a-enchainementinvalide.php');
```

Le code de ce script est le suivant :

```
<?php
  // enchaînement d'actions invalide
  $dReponse['erreurs']=array("Enchaînement d'actions invalide");
  $dSession['etat']=array('principal'=>'e-erreurs','secondaire'=>'enchainementinvalide');
?>
```

Il consiste à placer l'application dans l'état **[e-erreurs]**. Nous donnons dans **$dSession['etat']['secondaire']** une information qui sera exploitée par le générateur de la page d'erreurs. Comme nous l'avons déjà vu, ce générateur est **[e-erreurs.php]** :

```
  $dConfig['etats']['e-erreurs']=array(
    'actionsautorisees'=>array('get:retourformulaire','get:init'),
    'vue'=>'e-erreurs.php');
```

Nous verrons le code de ce générateur ultérieurement. La vue envoyée au client est la suivante :

4.14 Les vues de l'application

4.14.1 Affichage de la vue finale

Regardons comment le contrôleur envoie la réponse au client, une fois exécutée l'action demandée par celui-ci :

```
....
  // on démarre ou reprend la session
  session_start();
  $dSession=$_SESSION["session"];
  if($dSession) $dSession=unserialize($dSession);

  // on récupère l'action à entreprendre
  $sAction=$_GET['action'] ? strtolower($_GET['action']) : 'init';
```

```
  $sAction=strtolower($_SERVER['REQUEST_METHOD']).":$sAction";

  // l'enchaînement des actions est-il normal ?
  if( ! enchainementOK($dConfig,$dSession,$sAction)){
    // enchaînement anormal
    $sAction='enchainementinvalide';
  }//if

  // traitement de l'action
  $scriptAction=$dConfig['actions'][$sAction] ?
    $dConfig['actions'][$sAction]['url'] :
    $dConfig['actions']['actioninvalide']['url'];
  include $scriptAction;

   // envoi de la réponse(vue) au client
   $sEtat=$dSession['etat']['principal'];
   $scriptVue=$dConfig['etats'][$sEtat]['vue'];
   include $scriptVue;

.....

  // ------------------------------------------------------------------
  function finSession(&$dConfig,&$dReponse,&$dSession){
    // $dConfig : dictionnaire de configuration
    // $dSession : dictionnaire contenant les infos de session
    // $dReponse : le dictionnaire des arguments de la page de réponse

    // enregistrement de la session
...

    // envoi de la réponse au client
    include $dConfig['vuesReponse'][$dReponse['vuereponse']]['url'];

    // fin du script
    exit(0);
  }//finsession
```

Au retour d'un script d'action, le contrôleur récupère dans **$dSession['etat']['principal']** l'état dans lequel **il doit mettre l'application**. Cet état a été fixé par l'action qui vient d'être exécutée. Le contrôleur fait alors exécuter le générateur de vue associée à l'état. Il trouve le nom de celui-ci dans le fichier de configuration. Le rôle du générateur de vue est le suivant :

- il fixe dans **$dReponse['vuereponse']** le nom du modèle de réponse à utiliser. Cette information sera passée au contrôleur. Un modèle est une composition de vues élémentaires qui, rassemblées, forment la vue finale.
- il prépare les informations dynamiques à afficher dans la vue finale. Ce point est indépendant du contrôleur. Il s'agit de l'interface entre le générateur de vues et la vue finale. Elle est propre à chaque application.
- il se termine obligatoirement par l'appel à la fonction **finSession** du contrôleur. Cette fonction va
    - sauvegarder la session
    - envoyer la réponse

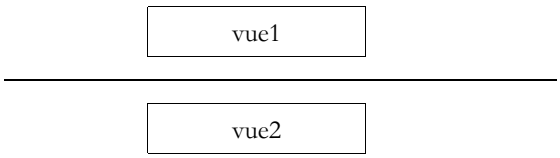
Le code de la fonction **finSession** est le suivant :

```
  // ------------------------------------------------------------------
  function finSession(&$dConfig,&$dReponse,&$dSession){
    // $dConfig : dictionnaire de configuration
    // $dSession : dictionnaire contenant les infos de session
    // $dReponse : le dictionnaire des arguments de la page de réponse

    // enregistrement de la session
...

    // envoi de la réponse au client
    include $dConfig['vuesReponse'][$dReponse['vuereponse']]['url'];

    // fin du script
    exit(0);
  }//finsession
```

La vue envoyée à l'utilisateur est définie par l'entité **$dReponse['vuereponse']** qui définit le modèle à utiliser pour la réponse finale.

4.14.2 Modèle de la réponse

L'application génèrera ses différentes réponses selon le modèle unique suivant :



Ce modèle est associé à la clé **modéle1** du dictionnaire **$dConfig['vuesreponse']** dans [**config.php**] :

```
  $dConfig['vuesReponse']['modele1']=array('url'=>'m-reponse.php');
```

Le script **m-reponse.php** est chargé de générer ce modèle :

```
<html>
  <head>
    <title>Application impots</title>
    <link type="text/css" href="<?php echo $dReponse['urlstyle'] ?>" rel="stylesheet" />
  </head>
  <body>
     <?php
      include $dReponse['vue1'];
    ?>
     <hr>
     <?php
      include $dReponse['vue2'];
    ?>
  </body>
</html>
```

Ce script a trois éléments dynamiques placés dans un dictionnaire **$dReponse** et associés aux clés suivantes :

1. **urlstyle** : url de la feuille de style du modèle
2. **vue1** : nom du script chargé de générer la vue **vue1**
3. **vue2** : nom du script chargé de générer la vue **vue2**

Un générateur de vue désirant utiliser le modèle **modèle1** devra définir ces trois éléments dynamiques. Nous définissons maintenant les vues élémentaires qui peuvent prendre la place des éléments [vue1] et [vue2] du modèle.

4.14.3 La vue élémentaire v-bandeau.php

Le script **v-bandeau.php** génère une vue qui sera placée dans la zone [vue1] :

```
<table>
  <tr>
    <td><img src="univ01.gif"></td>
     <td>
       <table>
         <tr>
           <td><div class='titre'><?php echo $dReponse['titre'] ?></div></td>
         </tr>
         <tr>
          <td><div class='resultat'><?php echo $dReponse['resultat']?></div></td>
         </tr>
       </table>
     </td>
  </tr>
</table>
```

Le générateur de vue devra définir deux éléments dynamiques placés dans un dictionnaire **$dReponse** et associés aux clés suivantes :

1. **titre** : titre à afficher
2. **resultat** : montant de l'impôt à payer

4.14.4 La vue élémentaire v-formulaire.php

La partie [vue2] correspond soit à la vue **[v-formulaire]** soit à la vue **[v-erreurs]**. La vue **[v-formulaire]** est générée par le script [**v-formulaire.php]** :

```
<form method="post" action="main.php?action=calculerimpot">
  <table>
    <tr>
      <td class="libelle">Etes-vous marié(e)</td>
       <td class="valeur">
        <input type="radio" name="optmarie" <?php echo $dReponse['optoui'] ?> value="oui">oui
        <input type="radio" name="optmarie" <?php echo $dReponse['optnon'] ?> value="non">non
```

```
      <td>
    <tr>
     <td class="libelle">Nombre d'enfants</td>
      <td class="valeur">
       <input type="text" class="text" name="txtenfants" size="3" value="<?php echo $dReponse['enfants'] ?
>"
      </td>
    </tr>
    <tr>
     <td class="libelle">Salaire annuel</td>
      <td class="valeur">
       <input   type="text"   class="text"   name="txtsalaire"   size="10"   value="<?php   echo   $dReponse
['salaire'] ?>"
      </td>
    </tr>
  </table>
  <hr>
  <input type="submit" class="submit" value="Calculer l'impôt">
</form>
<form method="post" action="main.php?action=effacerformulaire">
  <input type="submit" class="submit" value="Effacer le formulaire">
</form>
```

Les parties dynamiques de cette vue qui devront être définies par le générateur de vue sont associées aux clés suivantes du dictionnaire **$dReponse** :

- **optoui** : état du bouton radio de nom [optoui]
- **optnon** : état du bouton radio de nom [optnon]
- **enfants** : nombre d'enfants à placer dans le champ [txtenfants]
- **salaire** : salaire annuel à placer dans le champ [txtsalaire]

4.14.5 La vue élémentaire v-erreurs.php

La vue **[v-erreurs]** est générée par le script **v-erreurs.php** :

```
Les erreurs suivantes se sont produites :
<ul>
  <?php
    for($i=0;$i<count($dReponse["erreurs"]);$i++){
      echo "<li class='erreur'>".$dReponse["erreurs"][$i]."</li>\n";
    }//for
  ?>
</ul>
<div class="info"><?php echo $dReponse["info"] ?></div>
<br>
<a href="<?php echo $dReponse["href"] ?>"><?php echo $dReponse["lien"] ?></a>
```

Les parties dynamiques de cette vue à définir par le générateur de vue sont associées aux clés suivantes du dictionnaire **$dReponse** :

- **erreurs** : tableau de messages d'erreurs
- **info** : message d'information
- **lien** : texte d'un lien
- **href** : url cible du lien ci-dessus

4.14.6 La feuille de style

L'ensemble des vues est "habillée" par une feuille de style. Pour modifier l'aspect visuel de l'application, on modifiera sa feuille de style. La feuille de style **style1.css** est la suivante :

```
div.menu {
  background-color: #FFD700;
  color: #F08080;
  font-weight: bolder;
  text-align: center;
}
td.separateur {
  background: #FFDAB9;
  width: 20px;
}

table.modele2 {
  width: 600px;
}

BODY {
  background-image : url(standard.jpg);
  margin-left : 0px;
  margin-top : 6px;
  color : #4A1919;
  font-size: 10pt;
```

```
  font-family: Arial, Helvetica, sans-serif;
  scrollbar-face-color:#F2BE7A;
  scrollbar-arrow-color:#4A1919;
  scrollbar-track-color:#FFF1CC;
  scrollbar-3dlight-color:#CBB673;
  scrollbar-darkshadow-color:#CBB673;
}

div.titre {
  font: 30pt Garamond;
  color: #FF8C00;
  background-color: Yellow;
}

table.menu {
  background-color: #ADD8E6;
}


A:HOVER {
  text-decoration: underline;
  color: #FF0000;
  background-color : transparent;
}
A:ACTIVE {
  text-decoration: underline;
  color : #BF4141;
  background-color : transparent;
}
A:VISITED {
  color : #BF4141;
  background-color : transparent;
}

.error {
  color : red;
  font-weight : bold;
}

INPUT.text {
  margin-left : 3px;
  font-size:8pt;
  font-weight:bold;
  color:#4A1919;
  background-color:#FFF6E0;
  border-right:1px solid;
  border-left:1px solid;
  border-top:1px solid;
  border-bottom:1px solid;
}
td.libelle {
  background-color: #F0FFFF;
  color: #0000CD;
}

td.valeur {
  background-color: #DDA0DD;
}

DIV.resultat {
  background-color : #FFA07A;
  font : bold 12pt;
}

div.info {
  color: #FA8072;
}

li.erreur {
  color: #DC143C;
}

INPUT.submit {
  margin-left : 6px;
  font-size:8pt;
  font-weight:bold;
  color:#4A1919;
  background-color:#FFF1CC;
  border-right:1px solid;
  border-left:1px solid;
  border-top:1px solid;
  border-bottom:1px solid;
}
```

4.15 Les générateurs de vues

4.15.1 Rôle d'un générateur de vue

Rappelons la séquence de code du contrôleur qui fait exécuter un générateur de vue :

```
// traitement de l'action
$scriptAction=$dConfig['actions'][$sAction] ?
  $dConfig['actions'][$sAction]['url'] :
  $dConfig['actions']['actioninvalide']['url'];
include $scriptAction;

// envoi de la réponse(vue) au client
$sEtat=$dSession['etat']['principal'];
$scriptVue=$dConfig['etats'][$sEtat]['vue'];
include $scriptVue;
```

Un générateur de vue est lié à l'état dans lequel va être placée l'application. Le lien entre état et générateur de vue est fixé par configuration :

```
// configuration des états de l'application
$dConfig['etats']['e-formulaire']=array(
  'actionsautorisees'=>array('post:calculerimpot','get:init','post:effacerformulaire'),
   'vue'=>'e-formulaire.php');
$dConfig['etats']['e-erreurs']=array(
  'actionsautorisees'=>array('get:retourformulaire','get:init'),
  'vue'=>'e-erreurs.php');
$dConfig['etats']['sansetat']=array('actionsautorisees'=>array('get:init'));
```

Nous avons déjà indiqué quel était le rôle d'un générateur de vue. Rappelons-le ici. Un générateur de vue :

- fixe dans **$dReponse['vuereponse']** le nom du modèle de réponse à utiliser. Cette information sera passée au contrôleur. Un modèle est une composition de vues élémentaires qui rassemblées forment la vue finale.
- prépare les informations dynamiques à afficher dans la vue finale. Ce point est indépendant du contrôleur. Il s'agit de l'interface entre le générateur de vues et la vue finale. Elle est propre à chaque application.
- se termine obligatoirement par l'appel à la fonction **finSession** du contrôleur. Cette fonction va
    - sauvegarder la session
    - envoyer la réponse

4.15.2 Le générateur de vue associée à l'état [e-formulaire]

Le script chargé de générer la vue associée à l'état **[e-formulaire]** s'appelle **e-formulaire.php** :

```
$dConfig['etats']['e-formulaire']=array(
  'actionsautorisees'=>array('post:calculerimpot','get:init','post:effacerformulaire'),
   'vue'=>'e-formulaire.php');
```

Son code est le suivant :

```
<?php
  // on prépare la réponse formulaire
  $dReponse['titre']=$dConfig['webapp']['titre'];
  $dReponse['vuereponse']='modele1';
  $dReponse['vue1']=$dConfig['vues']['bandeau']['url'];
  $dReponse['vue2']=$dConfig['vues']['formulaire']['url'];
  $dReponse['urlstyle']=$dConfig['style']['url'];
  $dReponse['titre']=$dConfig['webapp']['titre'];

  // configuration selon le type de formulaire à générer
  $type=$dSession['etat']['secondaire'];
  if($type=='init'){
    // formulaire vide
    $dReponse['optnon']='checked';
  }//if
  if($type=='calculimpot'){
    // il nous faut réafficher les paramètres de saisie stockés dans la requête
    $dReponse['optoui']=$_POST['optmarie']=='oui' ? 'checked' : '';
    $dReponse['optnon']=$dReponse['optoui'] ? '' : 'checked';
    $dReponse['enfants']=$_POST['txtenfants'];
    $dReponse['salaire']=$_POST['txtsalaire'];
    $dReponse['resultat']='Impôt à payer : '.$dReponse['impot'].' F';
  }//if
  if($type=='retourformulaire'){
    // il nous faut réafficher les paramètres de saisie stockés dans la session
    $dReponse['optoui']=$dSession['requete']['optmarie']=='oui' ? 'checked' : '';
    $dReponse['optnon']=$dReponse['optoui']=='' ? 'checked' : '';
    $dReponse['enfants']=$dSession['requete']['txtenfants'];
```

```
    $dReponse['salaire']=$dSession['requete']['txtsalaire'];
  }//if
  // on envoie la réponse
   finSession($dConfig,$dReponse,$dSession);
?>
```

On remarquera que le générateur de vue respecte les conditions imposées à un générateur de vue :
- fixer dans **$dReponse['vuereponse']** le modèle de réponse à utiliser
- passer des informations à ce modèle. Ici elles sont passées par l'intermédiaire du dictionnaire **$dReponse**.
- se terminer par l'appel à la fonction **finSession** du contrôleur

Ici, le modèle utilisé est **'modèle1'**. Aussi le générateur définit-il les deux informations dont a besoin ce modèle **$dReponse['vue1']** et **$dReponse['vue2']**.

Dans le cas particulier de notre application, la vue associée à l'état **[e-formulaire]** dépend d'une information stockée dans la variable **$dSession['etat']['secondaire']**. C'est un choix de développement. Une autre application pourrait décider de passer des informations complémentaires d'une autre façon. Par ailleurs, ici toutes les informations nécessaires à l'affichage de la vue finale sont placées dans le dictionnaire **$dReponse**. Là encore, c'est un choix qui appartient au développeur. L'état **[e-formulaire]** peut se produire après quatre actions différentes : **init**, **calculerimpot**, **retourformulaire**, **effacerformulaire**. La vue à afficher n'est pas exactement la même dans tous les cas. Aussi a-t-on distingué ici trois cas dans **$dSession['etat']['secondaire']** :

- **init** : le formulaire est affiché vide
- **calculimpot** : le formulaire est affiché avec le montant de l'impôt et les données qui ont amené à son calcul
- **retourformulaire** : le formulaire est affiché avec les données initialement saisies

Ci-dessus, le script **e-formulaire.php** utilise cette information pour présenter la réponse selon ces trois variantes.

4.15.3 La vue associée à l'état [e-erreurs]

Le script chargé de générer la vue associée à l'état **[e-erreurs]** s'appelle **e-erreurs.php** est le suivant :

```
<?php

  // on prépare la réponse erreurs
  $dReponse['titre']=$dConfig['webapp']['titre'];
  $dReponse['vuereponse']='modele1';
   $dReponse['vue1']=$dConfig['vues']['bandeau']['url'];
   $dReponse['vue2']=$dConfig['vues']['erreurs']['url'];
  $dReponse['urlstyle']=$dConfig['style']['url'];
  $dReponse['titre']=$dConfig['webapp']['titre'];
  $dReponse['lien']='Retour au formulaire de saisie';
  $dReponse['href']='main.php?action=retourformulaire';

  // infos complémentaires
  $type=$dSession['etat']['secondaire'];
  if($type=='database'){
    $dReponse['info']="Veuillez avertir l'administrateur de l'application";
  }

  // on envoie la réponse
   finSession($dConfig,$dReponse,$dSession);
?>
```

4.15.4 Affichage de la vue finale

Les deux scripts générant les deux vues finales se terminent tous les deux par l'appel à la fonction **finSession** du contrôleur dont on rappelle ci-dessous une partie du code :

```
  function finSession(&$dConfig,&$dReponse,&$dSession){
    // $dConfig : dictionnaire de configuration
    // $dSession : dictionnaire contenant les infos de session
    // $dReponse : le dictionnaire des arguments de la page de réponse

....

    // on présente la réponse
    include $dConfig['vuesReponse'][$dReponse['vuereponse']]['url'];

    // fin du script
    exit(0);
  }//finsession
```

La vue envoyée à l'utilisateur est définie par l'entité **$dReponse['vuereponse']** qui définit le modèle à utiliser pour la réponse finale. Pour les états **[e-formulaire]** et **[e-erreurs]**, ce modèle a été défini égal à **modele1** :

```
  $dReponse['vuereponse']='modele1';
```

Ce modèle correspond par configuration au script **m-reponse.php** :

```
  $dConfig['vuesReponse']['modele1']=array('url'=>'m-reponse.php');
```

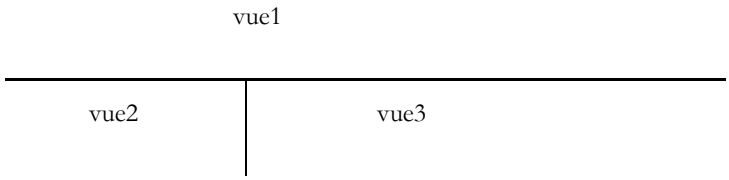
C'est donc le script [**m-reponse.php**] qui va envoyer la réponse au client.

4.16 Modifier le modèle de réponse

Nous supposons ici qu'on décide de changer l'aspect visuel de la réponse faite au client et nous nous intéressons à comprendre les répercussions que cela engendre au niveau du code.

4.16.1 Le nouveau modèle

La structure de la réponse sera maintenant la suivante :



Ce modèle s'apellera **modele2** et le script chargé de générer ce modèle s'appellera **m-reponse2.php.** Ces informations sont placées dans [**config.php**] :

```
  $dConfig['vuesReponse']['modele2']=array('url'=>'m-reponse2.php');
```

Le script correspondant à ce nouveau modèle est le suivant :

```
<html>
  <head>
    <title>Application impots</title>
    <link type="text/css" href="<?php echo $dReponse['urlstyle'] ?>" rel="stylesheet" />
  </head>
  <body>
    <table class='modele2'>
      <!-- début bandeau -->
      <tr>
        <td colspan="2"><?php  include $dReponse['vue1']; ?></td>
       </tr>
      <!-- fin bandeau -->
       <tr>
        <!-- début menu -->
        <td><?php include $dReponse['vue2']; ?></td>
        <!-- fin menu -->
        <!-- début zone 3 -->
        <td><?php include $dReponse['vue3']; ?></td>
        <!-- fin zone 3 -->
       </tr>
    </table>
  </body>
</html>
```

Les éléments dynamiques du modèle sont les suivants :

$dReponse['urlstyle'] : la feuille de style à utiliser
$dReponse['vue1'] : le script à utiliser pour générer [vue1]
$dReponse['vue2'] : le script à utiliser pour générer [vue2]
$dReponse['vue3'] : le script à utiliser pour générer [vue3]

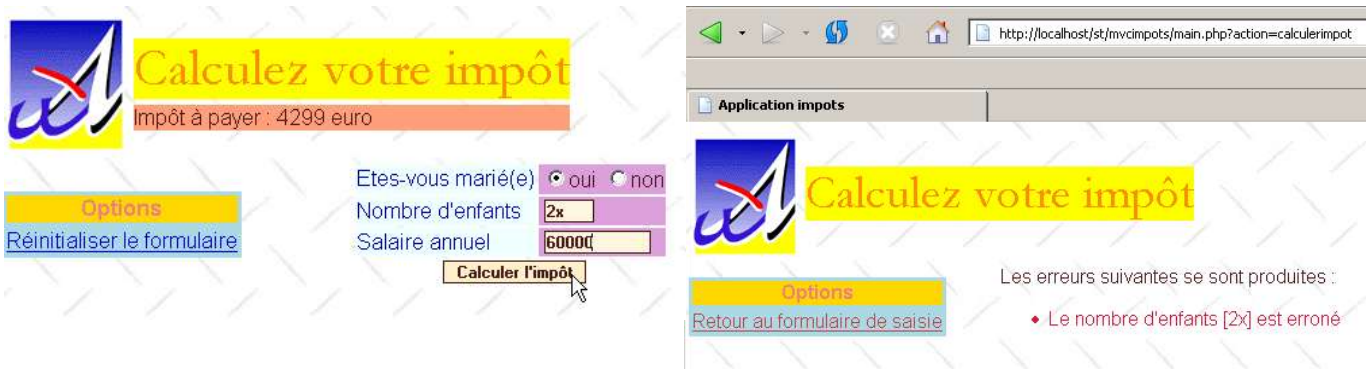
Ces éléments devront être fixés par les générateurs de vues.

4.16.2 Les différentes pages réponse

L'application présentera désormais les réponses suivantes à l'utilisateur. Lors de l'appel initial, la page réponse sera la suivante :

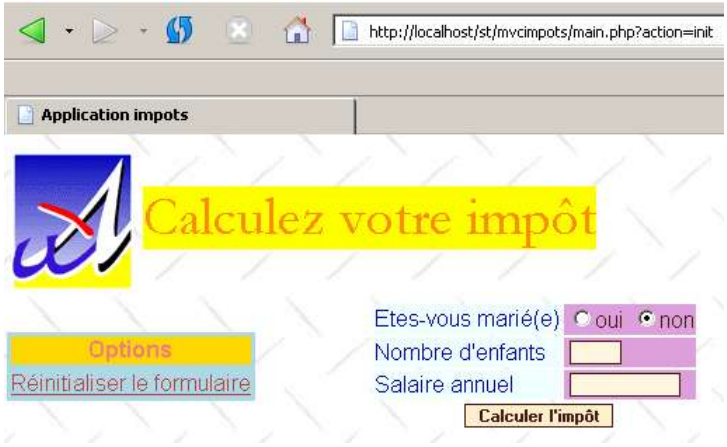
Si l'utilisateur fournit des données valides, l'impôt est calculé :

S'il fait des erreurs de saisie, la page d'erreurs est affichée :



S'il utilise le lien [Retour au formulaire de saisie], il retrouve le formulaire tel qu'il l'a validé :

Si ci-dessus, il utilise le lien [Réinitialiser le formulaire], il retrouve un formulaire vide :



On notera que l'application utilise bien les mêmes actions que précédemment. Seul l'aspect des réponses a changé.

4.16.3 Les vues élémentaires

La vue élémentaire [vue1] sera comme dans l'exemple précédent associé au script **v-bandeau.php** :

```
<table>
  <tr>
    <td><img src="univ01.gif"></td>
     <td>
       <table>
         <tr>
           <td><div class='titre'><?php echo $dReponse['titre'] ?></div></td>
         </tr>
         <tr>
          <td><div class='resultat'><?php echo $dReponse['resultat']?></div></td>
         </tr>
       </table>
     </td>
  </tr>
</table>
```

Cette vue a deux éléments dynamiques :

1. **$dReponse['titre']** : titre à afficher
2. **$dReponse['resultat']** : montant de l'impôt à payer

La vue élémentaire [vue2] sera elle, associée au script **v-menu.php** suivant :

```
<table class="menu">
  <tr>
    <td><div class="menu">Options</div></td>
  </tr>
  <?php
    for($i=0;$i<count($dReponse['liens']);$i++){
      echo '<tr><td><div class="option"><a href="'.
        $dReponse['liens'][$i]['url'].
         '">'.$dReponse['liens'][$i]['texte']."</a></div></td></tr>\n";
```

```
    }//$i
  ?>
</table>
```

Cette vue a les éléments dynamiques suivants :

- **$dReponse['liens']** : tableau des liens à afficher dans [vue2]. Chaque élément du tableau est un dictionnaire à deux clés :
  - **'url'** : url cible du lien
  - **'texte'** : texte du lien

La vue élémentaire [vue3] sera elle, associée au script **v-formulaire2.php** si on veut afficher le formulaire de saisie ou au script **v-erreurs2.php** si on veut afficher la page d'erreurs. Le code du script **v-formulaire2.php** est le suivant :

```
<form method="post" action="main.php?action=calculerimpot">
  <table>
    <tr>
      <td class="libelle">Etes-vous marié(e)</td>
       <td class="valeur">
        <input type="radio" name="optmarie" <?php echo $dReponse['optoui'] ?> value="oui">oui
        <input type="radio" name="optmarie" <?php echo $dReponse['optnon'] ?> value="non">non
       </td>
    </tr>
    <tr>
      <td class="libelle">Nombre d'enfants</td>
       <td class="valeur">
        <input type="text" class="text" name="txtenfants" size="3" value="<?php echo $dReponse['enfants'] ?
>"
       </td>
    </tr>
    <tr>
      <td class="libelle">Salaire annuel</td>
       <td class="valeur">
        <input   type="text"   class="text"   name="txtsalaire"   size="10"   value="<?php   echo   $dReponse
['salaire'] ?>"
       </td>
    </tr>
    <tr>
      <td colspan="2" align="center"><input type="submit" class="submit" value="Calculer l'impôt"></td>
    </tr>
  </table>
</form>
```

Les parties dynamiques de cette vue et qui devront être définies par le générateur de vue sont associées aux clés suivantes du dictionnaire **$dReponse** :

- **optoui** : état du bouton radio de nom optoui
- **optnon** : état du bouton radio de nom optnon
- **enfants** : nombre d'enfants à placer dans le champ txtenfants
- **salaire** : salaire annuel à placer dans le champ txtsalaire

Le script qui génère la page d'erreurs s'appelle **v-erreurs2.php**. Son code est le suivant :

```
Les erreurs suivantes se sont produites :
<ul>
  <?php
    for($i=0;$i<count($dReponse["erreurs"]);$i++){
      echo "<li class='erreur'>".$dReponse["erreurs"][$i]."</li>\n";
    }//for
  ?>
</ul>
<div class="info"><?php echo $dReponse["info"] ?></div>
```

Les parties dynamiques de cette vue à définir par le générateur de vue sont associées aux clés suivantes du dictionnaire **$dReponse** :

- **erreurs** : tableau de messages d'erreurs
- **info** : message d'information

4.16.4 La feuille de style

Elle n'a pas changé. C'est toujours **style1.css**.

4.16.5 Le nouveau fichier de configuration

Pour introduire ces nouvelles vues, nous sommes amenés à modifier certaines lignes du fichier de configuration.

```
<?php

  // configuration de php
```

```
  ini_set("register_globals","off");
  ini_set("display_errors","off");
  ini_set("expose_php","off");

  // liste des modules à inclure
  $dConfig['includes']=array('c-impots-data.php','c-impots-calcul.php');

  // contrôleur de l'application
  $dConfig['webapp']=array('titre'=>"Calculez votre impôt");

  // configuration des vues de l'aplication
  $dConfig['vuesReponse']['modele1']=array('url'=>'m-reponse.php');
   $dConfig['vuesReponse']['modele2']=array('url'=>'m-reponse2.php');
  $dConfig['vues']['formulaire']=array('url'=>'v-formulaire.php');
  $dConfig['vues']['erreurs']=array('url'=>'v-erreurs.php');
   $dConfig['vues']['formulaire2']=array('url'=>'v-formulaire2.php');
   $dConfig['vues']['erreurs2']=array('url'=>'v-erreurs2.php');
  $dConfig['vues']['bandeau']=array('url'=>'v-bandeau.php');
   $dConfig['vues']['menu']=array('url'=>'v-menu.php');
  $dConfig['style']['url']='style1.css';

  // configuration des actions de l'application
  $dConfig['actions']['get:init']=array('url'=>'a-init.php');
  $dConfig['actions']['post:calculerimpot']=array('url'=>'a-calculimpot.php');
  $dConfig['actions']['get:retourformulaire']=array('url'=>'a-retourformulaire.php');
  $dConfig['actions']['post:effacerformulaire']=array('url'=>'a-init.php');
  $dConfig['actions']['enchainementinvalide']=array('url'=>'a-enchainementinvalide.php');
  $dConfig['actions']['actioninvalide']=array('url'=>'a-actioninvalide.php');

  // configuration des états de l'application
   $dConfig['etats']['e-formulaire']=array(
      'actionsautorisees'=>array('post:calculerimpot','get:init','post:effacerformulaire'),
     'vue'=>'e-formulaire2.php');
   $dConfig['etats']['e-erreurs']=array(
    'actionsautorisees'=>array('get:retourformulaire','get:init'),
    'vue'=>'e-erreurs2.php');
  $dConfig['etats']['sansetat']=array('actionsautorisees'=>array('get:init'));

  // configuration modèle de l'application
  $dConfig["DSN"]=array(
    "sgbd"=>"mysql",
    "user"=>"seldbimpots",
    "mdp"=>"mdpseldbimpots",
    "host"=>"localhost",
    "database"=>"dbimpots"
  );
?>
```

La modification essentielle et quasi unique consiste **à changer les générateurs de vue** associés aux états [e-formulaire] et [e-erreurs]. Ceci fait, les nouveaux générateurs de vue sont chargés de générer les nouvelles pages réponse.

4.16.6 Le générateur de vue associé à l'état [e-formulaire]

Dans le fichier de configuration, l'état **[e-formulaire]** est maintenant associé au générateur de vue **e-formulaire2.php**. Le code de ce script est le suivant :

```
<?php
  // on prépare la réponse formulaire
  $dReponse['titre']=$dConfig['webapp']['titre'];
  $dReponse['vuereponse']='modele2';
   $dReponse['vue1']=$dConfig['vues']['bandeau']['url'];
   $dReponse['vue2']=$dConfig['vues']['menu']['url'];
   $dReponse['vue3']=$dConfig['vues']['formulaire2']['url'];
  $dReponse['urlstyle']=$dConfig['style']['url'];
  $dReponse['titre']=$dConfig['webapp']['titre'];
   $dReponse['liens']=array(
    array('texte'=>'Réinitialiser le formulaire', 'url'=>'main.php?action=init')
   );

  // configuration selon le type de formulaire à générer
  $type=$dSession['etat']['secondaire'];
  if($type=='init'){
    // formulaire vide
    $dReponse['optnon']='checked';
  }//if
  if($type=='calculimpot'){
    // il nous faut réafficher les paramètres de saisie stockés dans la requête
    $dReponse['optoui']=$_POST['optmarie']=='oui' ? 'checked' : '';
    $dReponse['optnon']=$dReponse['optoui'] ? '' : 'checked';
    $dReponse['enfants']=$_POST['txtenfants'];
    $dReponse['salaire']=$_POST['txtsalaire'];
    $dReponse['resultat']='Impôt à payer : '.$dReponse['impot'].' F';
  }//if
```

```
  if($type=='retourformulaire'){
    // il nous faut réafficher les paramètres de saisie stockés dans la session
    $dReponse['optoui']=$dSession['requete']['optmarie']=='oui' ? 'checked' : '';
    $dReponse['optnon']=$dReponse['optoui']=='' ? 'checked' : '';
    $dReponse['enfants']=$dSession['requete']['txtenfants'];
    $dReponse['salaire']=$dSession['requete']['txtsalaire'];
  }//if
  // on envoie la réponse
  finSession($dConfig,$dReponse,$dSession);
?>
```

Les principales modifications sont les suivantes :

1. le générateur de vue indique qu'il veut utiliser le modèle de réponse **modele2**
2. pour cette raison il renseigne les éléments dynamiques **$dReponse['vue1']**, **$dReponse['vue2']**, **$dReponse['vue3']** tous trois nécessaires au modèle de réponse **modele2**.
3. le générateur renseigne également l'élément dynamique **$dReponse['liens']** qui fixe les liens à afficher dans la zone [vue2] de la réponse.

4.16.7 Le générateur de vue associé à l'état [e-erreurs]

Dans le fichier de configuration, l'état **[e-erreurs]** est maintenant associé au générateur de vue **e-erreurs2.php**. Le code de ce script est le suivant :

```
<?php

  // on prépare la réponse erreurs
  $dReponse['titre']=$dConfig['webapp']['titre'];
  $dReponse['vuereponse']='modele2';
   $dReponse['vue1']=$dConfig['vues']['bandeau']['url'];
   $dReponse['vue2']=$dConfig['vues']['menu']['url'];
   $dReponse['vue3']=$dConfig['vues']['erreurs2']['url'];
  $dReponse['urlstyle']=$dConfig['style']['url'];
  $dReponse['titre']=$dConfig['webapp']['titre'];
   $dReponse['liens']=array(
    array('texte'=>'Retour au formulaire de saisie', 'url'=>'main.php?action=retourformulaire')
   );

  // infos complémentaires
  $type=$dSession['etat']['secondaire'];
  if($type=='database'){
    $dReponse['info']="Veuillez avertir l'administrateur de l'application";
  }

  // on envoie la réponse
  finSession($dConfig,$dReponse,$dSession);
?>
```

Les modifications apportées sont identiques à celles apportées au générateur de vue **e-formulaire2.php**.

5 Conclusion

Nous avons pu montrer sur un exemple, l'intérêt de notre contrôleur générique. Nous n'avons pas eu à écrire celui-ci. Nous nous sommes contentés d'écrire les scripts des actions, des générateurs de vue et des vues de l'application. Nous avons par ailleurs montré l'intérêt de séparer les actions des vues. Nous avons pu ainsi changer l'aspect des réponses sans modifier une seule ligne de code des scripts d'action. Seuls les scripts impliqués dans la génération des vues ont été modifiés. Pour que ceci soit possible, le script d'action ne doit faire aucune hypothèse sur la vue qui va visualiser les informations qu'il a calculées. Il doit se contenter de rendre ces informations au contrôleur qui les transmet à un générateur de vue qui les mettra en forme. C'est une règle absolue : une action doit être complètement déconnectée des vues.

Nous nous sommes approchés dans ce chapitre de la philosophie **Struts** bien connue des développeurs Java. Un projet 'open source' appelé **php.MVC** permet de faire du développement web/php avec la philosophie Struts. On consultera le site **http://www.phpmvc.net/** pour plus d'informations.

6 ANNEXE - PEAR DB

Note : le texte ci-dessous est tiré de la documentation officielle de PEAR DB [http://pear.php.net/]. Il n'est là que pour faciliter le travail du lecteur de ce document.

6.1 PEAR DB: a unified API for accessing SQL-databases

This chapter describes how to use the PEAR database abstraction layer.

DSN -- The data source name
Connect -- Connecting and disconnecting
Query -- Performing a query against a database.
Fetch -- Fetching rows from the query

6.1.1 DSN

To connect to a database through PEAR::DB, you have to create a valid DSN - data source name. This DSN consists in the following parts:

phptype: Database backend used in PHP (i.e. mysql, odbc etc.)
dbsyntax: Database used with regards to SQL syntax etc.
protocol: Communication protocol to use (i.e. tcp, unix etc.)
hostspec: Host specification (hostname[:port])
database: Database to use on the DBMS server
username: User name for login
password: Password for login
proto_opts: Maybe used with *protocol*

The format of the supplied DSN is in its fullest form:

```
phptype(dbsyntax)://username:password@protocol+hostspec/database
```

Most variations are allowed:

```
phptype://username:password@protocol+hostspec:110//usr/db_file.db
phptype://username:password@hostspec/database_name
phptype://username:password@hostspec
phptype://username@hostspec
phptype://hostspec/database
phptype://hostspec
phptype(dbsyntax)
phptype
```

The currently supported database backends are:

```
mysql  -> MySQL
pgsql  -> PostgreSQL
ibase  -> InterBase
msql   -> Mini SQL
mssql  -> Microsoft SQL Server
oci8   -> Oracle 7/8/8i
odbc   -> ODBC (Open Database Connectivity)
sybase -> SyBase
ifx    -> Informix
fbsql  -> FrontBase
```

With an up-to-date version of **DB**, you can use a second DSN format

```
phptype(syntax)://user:pass@protocol(proto_opts)/database

example: Connect to database through a socket
mysql://user@unix(/path/to/socket)/pear

Connect to database on a non standard port
pgsql://user:pass@word@tcp(localhost:5555)/pear
```

6.1.2 Connect

To connect to a database you have to use the function **DB::connect()**, which requires a valid DSN as parameter and optional a boolean value, which determines wether to use a persistent connection or not. In case of success you get a new instance of the database class. It is strongly recommened to check this return value with **DB::isError()**. To disconnect use the method **disconnect ()** from your database class instance.

```
<?php
require_once 'DB.php';

$user = 'foo';
$pass = 'bar';
$host = 'localhost';
$db_name = 'clients_db';

// Data Source Name: This is the universal connection string
$dsn = "mysql://$user:$pass@$host/$db_name";

// DB::connect will return a PEAR DB object on success
// or an PEAR DB Error object on error
$db = DB::connect($dsn, true);

// Alternatively: $db = DB::connect($dsn);

// With DB::isError you can differentiate between an error or
// a valid connection.
if (DB::isError($db)) {
    die ($db->getMessage());
}
....
// You can disconnect from the database with:
$db->disconnect();
?>
```

6.1.3 Query

To perform a query against a database you have to use the function **query()**, that takes the query string as an argument. On failure you get a DB Error object, check it with **DB::isError()**. On succes you get *DB_OK* (predefined PEAR::DB constant) or when you set a *SELECT*-statment a DB Result object.

```
<?php
// Once you have a valid DB object...
$sql = "select * from clients";
$result = $db->query($sql);

// Always check that $result is not an error
if (DB::isError($result)) {
    die ($result->getMessage());
}
....
?>
```

6.1.4 Fetch

The **DB_Result** object provides two functions to fetch rows: **fetchRow()** and **fetchInto()**. **fetchRow()** returns the row, null on no more data or a **DB_Error**, when an error occurs. **fetchInto()** requires a variable, which be will directly assigned by reference to the result row. It will return null when result set is empty or a **DB_Error** too.

```
<?php
...
$db = DB::connect($dsn);
$res = $db->query("select * from mytable");

// Get each row of data on each iteration until
// there is no more rows
while ($row = $res->fetchRow()) {
   $id = $row[0];
}
// Or:
// an example using fetchInto()
while ($res->fetchInto($row)) {
   $id = $row[0];
}
?>
```

6.1.4.1 Select the format of the fetched row

The fetch modes supported are:

- *DB_FETCHMODE_ORDERED* (default) : returns an ordered array. The order is taken from the select statment.

```
<?php
$res = $db->query('select id, name, email from users');
$row = $res->fetchRow(DB_FETCHMODE_ORDERED);
/*
$row will contain:
array (
   0 => <column "id" data>,
   1 => <column "name" data>,
   2 => <column "email" data>
)
*/
// Access the data with:
$id    = $row[0];
$name  = $row[1];
$email = $row[2];
?>
```

- *DB_FETCHMODE_ASSOC* : returns an associative array with the column names as the array keys

```
<?php
$res = $db->query('select id, name, email from users');
$row = $res->fetchRow(DB_FETCHMODE_ASSOC);
/*
$row will contain:
array (
   'id'    => <column "id" data>,
   'name'  => <column "name" data>,
   'email' => <column "email" data>
)
*/
// Access the data with:
$id    = $row['id'];
$name  = $row['name'];
$email = $row['email'];
?>
```

- *DB_FETCHMODE_OBJECT* : returns a DB_row object with column names as properties

```
<?php
$res = $db->query('select id, name, email from users');
$row = $res->fetchRow(DB_FETCHMODE_OBJECT);
/*
$row will contain:
db_row Object
(
   [id]    => <column "id" data>,
   [name]  => <column "name" data>,
   [email] => <column "email" data>
)
*/
// Access the data with:
$id    = $row->id;
$name  = $row->name;
$email = $row->email;
?>
```

6.1.4.2 Set the format of the fetched row

You can set the fetch mode for every call or for your whole DB instance.

```
<?php
// 1) Set the mode per call:
while ($row = $result->fetchRow(DB_FETCHMODE_ASSOC)) {
   $id = $row['id'];
}

// 2) Set the mode for all calls:
$db = DB::connect($dsn);
// this will set a default fetchmode for this Pear DB instance
// (for all queries)
$db->setFetchMode(DB_FETCHMODE_ASSOC);

$result = $db->query(...);
while ($row = $result->fetchRow()) {
   $id = $row['id'];
}
?>
```

6.1.4.3 Fetch rows by number

The PEAR DB fetch system also supports an extra parameter to the fetch statement. So you can fetch rows from a result by number. This is especially helpful if you only want to show sets of an entire result (for example in building paginated HTML lists), fetch rows in an special order, etc.

```
<?php
...
// the row to start fetching
$from = 50;

// how many results per page
$res_per_page = 10;

// the last row to fetch for this page
$to = $from + $res_per_page;

foreach (range($from, $to) as $rownum) {
   if (!$row = $res->fetchrow($fetchmode, $rownum)) {
      break;
   }
   $id = $row[0];
   ....
}
?>
```

6.1.4.4 Freeing the result set

It is recommended to finish the result set after processing in order to to save memory. Use **free()** to do this.

```
<?php
...
$result = $db->query('SELECT * FROM clients');
while ($row = $result->fetchRow()) {
...
}
$result->free();
?>
```

6.1.4.5 Quick data retrieving

PEAR DB provides some special ways to retrieve information from a query without the need of using **fetch*()** and loop throw results.

getOne() retrieves the first result of the first column from a query

```
$numrows = $db->getOne('select count(id) from clients');
```

getRow() returns the first row and return it as an array

```
$sql = 'select name, address, phone from clients where id=1';
if (is_array($row = $db->getRow($sql))) {
   list($name, $address, $phone) = $row;
}
```

getCol() returns an array with the data of the selected column. It accepts the column number to retrieve as the second param.

```
$all_client_names = $db->getCol('select name from clients');
```

The above sentence could return for example: $all_client_names = array('Stig', 'Jon', 'Colin');

getAssoc() fetches the entire result set of a query and return it as an associative array using the first column as the key.

```
$data = getAssoc('SELECT name, surname, phone FROM mytable')
/*
Will return:
array(
  'Peter'      => array('Smith', '944679408'),
  'Tomas' => array('Cox', '944679408'),
  'Richard' => array('Merz', '944679408')
)
*/
```

getAll() fetches all the rows returned from a query

```
$data = getAll('SELECT id, text, date FROM mytable');
/*
Will return:
array(
   1 => array('4', 'four', '2004'),
   2 => array('5', 'five', '2005'),
   3 => array('6', 'six', '2006')
)
*/
```

The **get*()** family methods will do all the dirty job for you, this is: launch the query, fetch the data and free the result. Please note that as all PEAR DB functions they will return a **PEAR DB_error** object on errors.

6.1.4.6 Getting more information from query results

With PEAR DB you have many ways to retrieve useful information from query results. These are:

- **numRows()**: Returns the total number of rows returned from a "SELECT" query.

```
// Number of rows
echo $res->numRows();
```

- **numCols()**: Returns the total number of columns returned from a "SELECT" query.

```
// Number of cols
echo $res->numCols();
```

- **affectedRows()**: Returns the number of rows affected by a data manipulation query ("INSERT", "UPDATE" or "DELETE").

```
// remember that this statement won't return a result object
$db->query('DELETE * FROM clients');
echo 'I have deleted ' . $db->affectedRows() . ' clients';
```

- **tableInfo()**: Returns an associative array with information about the returned fields from a "SELECT" query.

```
// Table Info
print_r($res->tableInfo());
```