

Desenvolupament/manteniment ràpid d'Aplicacions Web

Jordi Catà Castillo - jordi.cata@dunlock.com

Dunlock – Enginyeria de Xarxes i Programari Lliure

6-9 de Juliol de 2005 – Campus de Vilanova i la Geltrú

Resum: *Independitzar les diferents capes d'una aplicació és una feina molt comú en el camp de l'Enginyeria del Software. Aquesta independència té com objectiu reduir els costos de manteniment, escalabilitat i reutilització de les aplicacions. Prenent com a partida aquest punt de vista, aquest article recull diferents eines per tal desenvolupar aplicacions web a tres nivells independents entre sí i facilitar el seu posterior manteniment.*

1. Introducció

Una manera habitual, sobretot per als programadors novells de desenvolupar aplicacions web està en programar conjuntament els nivells de disseny, negoci i físic. Aquest fet ocasiona que els futurs manteniments de les aplicacions siguin tasques costoses. Per exemple la modificació del disseny d'una aplicació web podria implicar feina de com a mínim un dissenyador i un programador.

Una tasca comú des del punt de vista de l'Enginyera del Software [1][2] està en independitzar les diferents capes d'una aplicació, per tal de facilitar el seu posterior *manteniment* i *escalabilitat*. Independitzant les capes de presentació (disseny) de la capa de negoci, ens permetrà *reduir costos*, ja que únicament en la modificació podria intervenir un dissenyador (sempre i quan sigui modificació de disseny).

L'objectiu donç, d'aquest article, està en mostrar diferents eines per tal d'independitzar les capes d'una aplicació web i al modificar per exemple la capa de presentació no calgui modificar les altres capes.

2. Disseny a 3 capes (física, negoci i presentació)

Una manera molt comú de desenvolupar aplicacions, des del punt de vista de l'Enginyeria *del Software* [1][2], és el model de 3 capes: capa física, capa de negoci i la capa de presentació. La capa física s'encarrega d'accedir a les dades, la capa de negoci s'encarrega de processar, executar, tractar les dades que li venen de la capa física i passar-les a la capa de presentació que serà l'encarregada de mostrar-les a l'usuari final.

En la *Figura 1* podem veure quines eines formaran cada capa. A continuació és descriu cada capa:

- Física: formada per la llibreria *Pear* [3] i classes que s'encarregaran d'accedir a les dades, a més de garantir la independència de l'aplicació del tipus de base de dades que utilitzem en cada moment
- Negoci: encarregada d'utilitzar les classes de la capa física per tal d'accedir a les dades i gestionar-les i enviar-les a la capa de presentació.
- Presentació: la seva funció està en mostrar la interfície a l'usuari. Està formada per la llibreria *Smarty* [4] i la seva extensió *Formsess* [5] que ens garanteixen la independència entre la capa de negoci i aquesta.

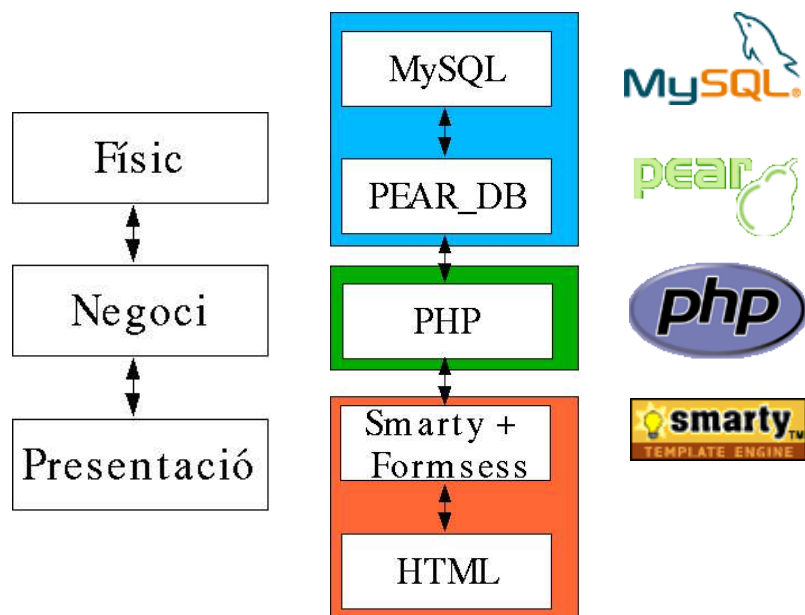


Figura 1: Esquema lògic de les tres capes

3. Independitzant la capa física mitjançant Pear

El sistema *Pear* (*PHP Extension and Application Repository*)[3], és un *framework* i sistema

de distribució per crear components de PHP reutilitzables. Entre d'altres paquets/funcionalitats destaquem la d'independitzar la base de dades. Això ens permetrà treballar amb diferents sistemes gestors de bases de dades (SGBD) com poden ser: mysql, postgres, oracle, sqlite, sybase, etc.

Imaginem ara un tros de codi des del que s'accedeixi a una base de dades mysql, postgres i oracle, aquest codi és el següent:

```
//query a la base de dades
$query = 'SELECT * FROM my_table';

//codi per accedir a mysql
$link = mysql_connect('my_host', 'my_user', 'my_password') ;
mysql_select_db('my_database');
$result = mysql_query($query);
$line = mysql_fetch_array($result)

//codi per accedir a postgres
$link = pg_pconnect('host=host port=5432 dbname=database user=user password=pwd');
$result = pg_query($link, "SELECT * FROM my_table");
$line = pg_fetch_result($result)

//codi per accedir a oracle
$conn = Ora_Logon("my_user", "my_password");
$ora_cur = ora_do( $ora_conn, "SELECT * FROM my_table");
ora_fetch_into( $ora_cur, $line);
```

Amb aquest exemple podem observar les diferents funcions que s'han d'utilitzar per tal d'accedir a diferents SGBD (mysql, postgres i oracle). Si hem de modificar una aplicació que treballa amb mysql per que passi a treballar amb postgres, els canvis que s'hauran de fer son obvis, haurem de modificar les funcions de mysql per les funcions de postgres.

Utilitzant el sistema *Pear* el codi que hem d'utilitzar per tal d'independitzar els SGBD és el següent:

```
$dbh = DB::connect ('mysql://my_user:my_pass@my_host/my_database); //mysql
$dbh = DB::connect ('pgsql://my_user:my_pass@my_host/my_database); //postgres
$dbh = DB::connect ('sqlite://my_user:my_pass@my_host/my_database); //sqlite
$result = $dbh->Query ( 'SELECT * FROM * my_table');
$line = $result->fetchRow();
```

Utilitzant *Pear* podem observar que el canvi d'un SGBD a un altre, és mínim, simplement canviant un paràmetre a la funció [DB::connect](#) en tenim suficient, a partir d'aquest paràmetre totes les consultes mitjançant la funció Query seran optimitzades per cada

SGBD sense canviar cap línia de codi.

4. Independitzant la capa de presentació mitjançant Smarty i Formsess

4.1. Smarty

El sistema *Smarty* [4] és un motor de plantilles fàcilment extensible via *plugins*, que ens permet independitzar la presentació de la capa de negoci de l'aplicació. Per tal de servir a aquest propòsit *Smarty* conté una sèrie de tags que incrustats dins d'una plantilla (fitxer HTML), exemple de tags son `{$variable}`, `{#constant#}`, `{if}{/if}`, `{html_options}`, `{html_radios}`, etc, permeten identificar quins son els paràmetres a substituir (seria com buscar el tag i reemplaçar per el valor que hi volem posar).

La següent figura 2 ens mostra el funcionament d'aquest sistema:

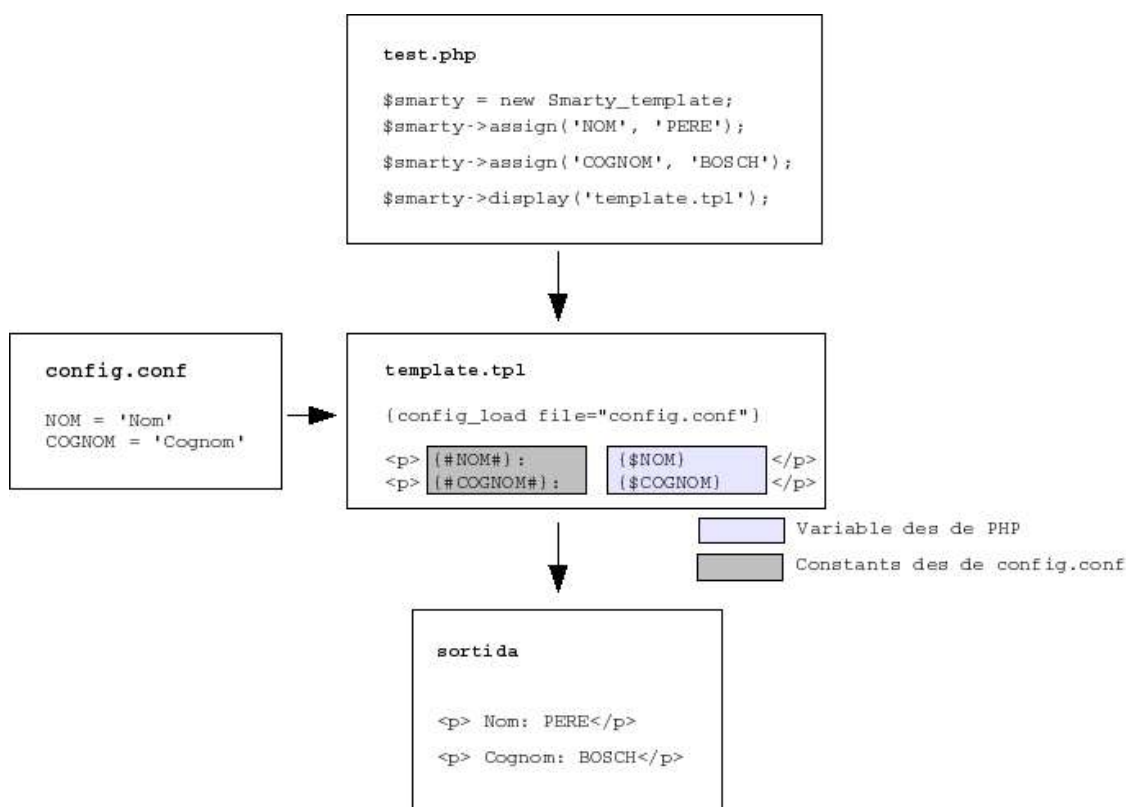


Figura 2: Esquema de funcionament d'Smarty

En la figura anterior podem veure com des del fitxer test.php s'instancia una classe Smarty, que és l'encarregada de passar mitjançant la funció *assign* les variables a la capa de presentació (fitxer template.tpl). Podem observar també com en la plantilla trobem la

sentència `{config_load file='config.conf'}`, aquesta s'encarrega de llegir el fitxer `config.conf` i assignar els valors que si troben als tags `{#nom#}` de la plantilla, d'aquesta manera no necessitem utilitzar la funció `assign` per a tots els valors, ja que podem tenir valors constants que s'assignaran automàticament des dels fitxers de configuració especificats.

El diagrama de seqüències del procés és el següent:

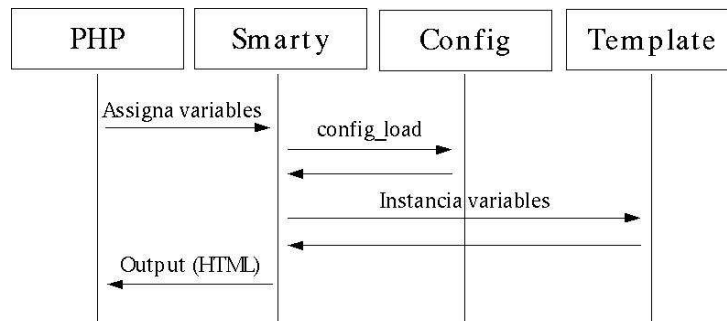


Figura 3: Diagrama de seqüències de Smarty

Smarty a part de les assignacions senzilles permet assignar arrays, tenir estructures de control, d'entre d'altres funcions. Passem a veure un exemple:

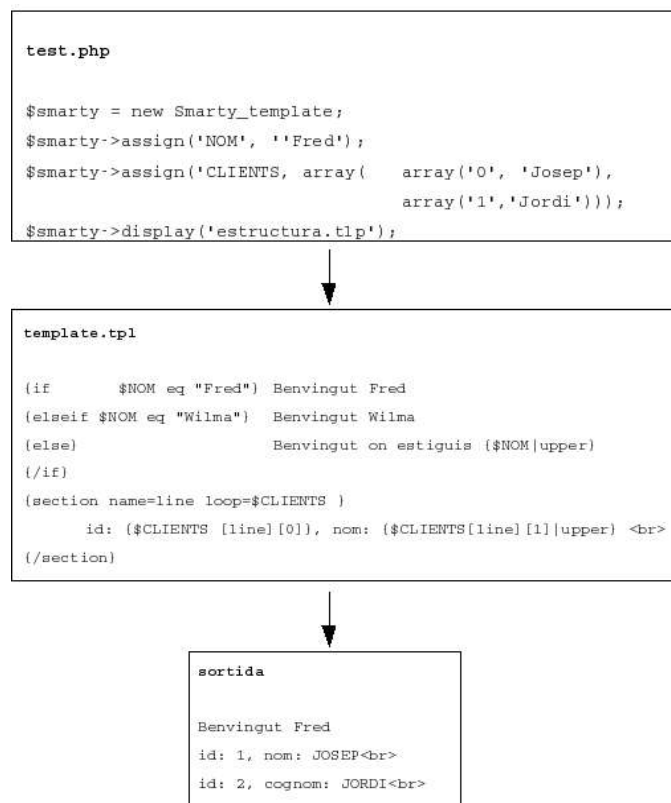


Figura 4: Exemple de plantilles en Smarty i la sortida en html

En l'exemple de la figura 4 podem observar com treballar amb estructures de control condicionals i amb arrays amb el sistema *Smarty*. Per fer- ho hem d'utilitzar els tags `{if}{/if}` i `{section}{/section}` respectivament.

La primera estructura de control dins el fitxer template.tpl corresponent al tag `{if}{/if}` i representa una estructura de control condicional que mostrarà la sentència: 'Benvingut Fred' en el cas que la variable `$NOM` (que hem assignat amb la funció assign) sigui igual a 'Fred', cas contrari comprovarà si és igual a 'Wilma' i sinò farà la branca else.

La segona estructura de control dins el fitxer template.tpl `{section}{/section}` és del tipus bucle i el seu funcionament és de que per cada element del array `{$CLIENTS}` que li hem assignat des de la funció assign farà el que hi ha entre `{section}{/section}` i el resultat en el nostre exemple serà de que faci dues iteracions, una corresponent al id: 1 i l'altre al id:2 tal i com veiem en la sortida de la figura 4.

Podeu consultar més informació referent a totes les possibilitats que ofereix *Smarty* en la seva pàgina web: smarty.php.net [4].

Resumint: utilitzar *Smarty* ens permet Independitzar la presentació. Això ens permetrà modificar les plantilles html sense necessitat de retocar cap línia de codi del negoci, per el que qualsevol dissenyador podrà modificar el codi html sense necessitat de saber php (només haurà de tenir en compte els tags propis d'*Smarty*).

4.2. Formsess

Formsess [5] és una extensió d'*Smarty* [4] per ajudar a crear formularis. Gestiona els passos necessaris de creació, manipulació i validació de formularis, generant automàticament les validacions en javascript. Aquest sistema es capaç d'utilitzar tags d'*Smarty*, però la seva potència radica en la utilització dels seus propis tags html (`<fs:form>`, `<fs:input>`, `<fs:option>`, etc) per tal de generar automàticament les validacions en javascript. Aquestes validacions solen ser una feina feixuga de programar, i *formsess* o soluciona de manera molt simple.

En la figura 5 podem veure el diagrama de seqüències d'utilització de *formsess*.

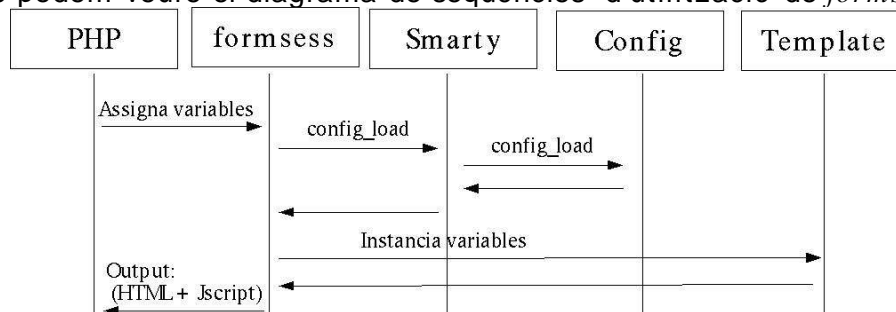


Figura 5: Diagrama de seqüències de Formsess

A continuació es mostra un exemple del funcionament i d'alguns tags del sistema *formsess*:

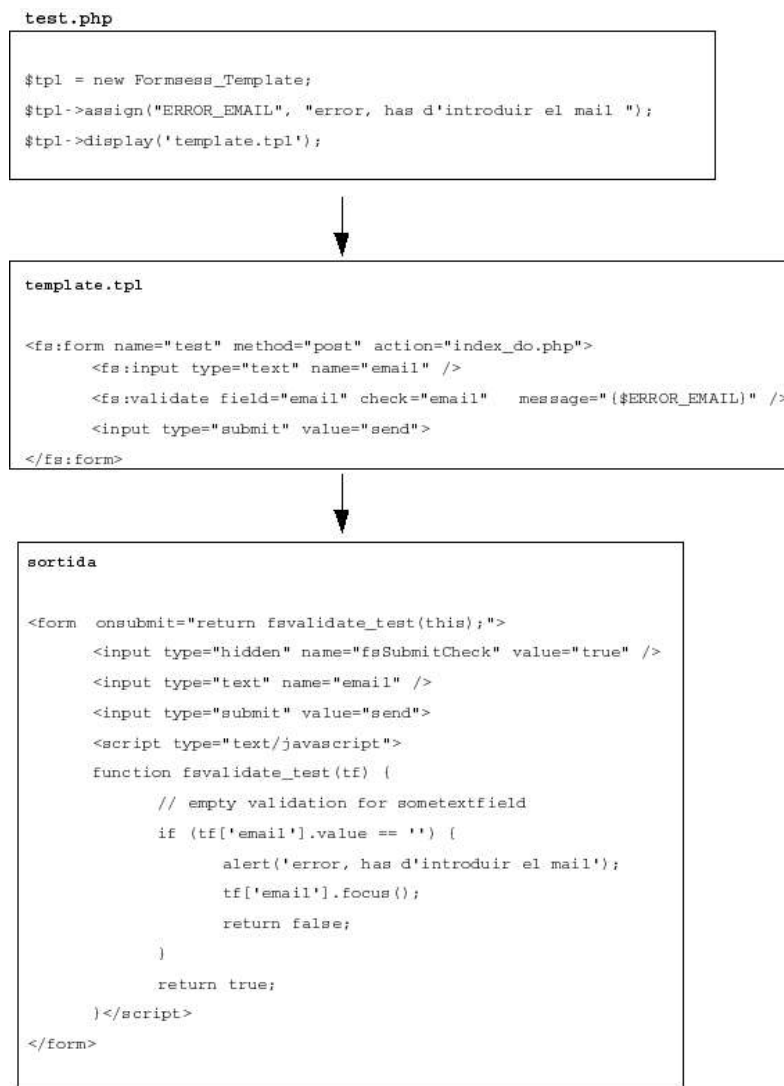


Figura 6: Exemple de plantilles utilitzant *formsess* i la sortida en html i javascript

Podem veure en el fitxer `template.tpl` com hi apareixen uns nous tags, com poden ser `<fs:input>` i `<fs:validate>`. El primer tag simplement es per indicar que aquest serà un textbox de nom email. El segon tag, ens indica que el textbox de nom email s'haurà de validar seguint la mascarà d'un email, per exemple nom@domini.com, en cas de que la validació no sigui correcte es mostrarà el text que marca la variable `{ $ERROR_EMAIL }` que hem instanciat com si fos una variable normal d'*Smarty* a partir de la funció *assign*. Un cop la plantilla sigui parsejada per *formsess* el resultat serà codi html més la validació dels camps del formulari en javascript.

Podeu consultar més informació referent a totes les possibilitats que ofereix *Formsess* en la seva pàgina web: formsess.sourceforge.net [5].

Resumint: *Formsess* afegeix la generació automàtica de validacions en javascript de camps d'un formulari a partir d'una sèrie de tags simples `<fs:input><fs:validate>`, per el que altre cop disminuïm la necessitat dels dissenyadors de conèixer la programació en

php i javascript i reduïm al mateix temps la feina dels programadors en les validacions dels camps dels formularis.

5. Conclusions

Independitzar les diferents capes d'una aplicació, a partir de la integració de les eines que hem vist: Pear, Smarty i Formsess dins les nostres aplicacions web, és una tasca senzilla i que permet reduir el temps de modificació i extensió. Sobretot el utilitzar *Smarty* i *Formsess* per aïllar la capa de presentació permet reduir el temps de modificació i programació del disseny i validacions en javascript en els aplicatius web, permetent que persones sense coneixement de programació en php pugin modificar el disseny sense necessitat d'haver de tornar a programar i/o debugar cap línia de codi per part dels programadors.

6. Referències

- [1] *Enginyeria del Software: un enfoque práctico.*
Roger S. Pressman. McGraw Hill.
- [2] *Design Patterns: Elements of Reusable Object- Oriented Software.*
Erich Gamma et all. Addison Wesley
- [3] *Pear: PHP Extension and Application Repository*
pear.php.net
- [4] *Smarty*
smarty.php.net
- [5] *Formsess*
formsess.sourceforge.net