Comment et pourquoi j'utilise Perl/CPAN pour enseigner le développement d'interfaces web vers des bases de données

E.Otton - Ecole des mines d'Albi-Carmaux

COMPIL - 25 novembre 2010

Le contexte

des étudiants pas forcément informaticiens,

- destinés à faire plutôt de la maitrise d'ouvrage.
- L'option systèmes d'information = 6 mois après 3 ans de formation généraliste
- ... partagés à moitié entre maitrise d'ouvrage et maitrise d'oeuvre...
- Dont 20h pour apprendre à développer des interfaces web sur des bases de données.

On pose un problème :

- Un modèle de données simple : client -> commandes -> produits,
- Comment faire une interface web ?

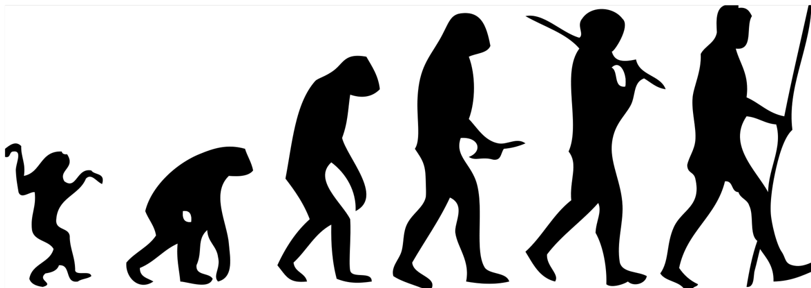
Comment on démarre...

```
package MonSchema;                              # Modèle
use base qw/DBIx::Class::Schema::Loader/;
__PACKAGE__->loader_options( constraint => qr/client|commande/);

package MonAppli;
use CGI::Application::Plugin::TT;               # Vue
use base 'CGI::Application';                    # Controleur
use MonSchema;
my $schema = MonSchema->connect('dbi:mysql:test','moi','mot2pass')

sub lister_clients : StartRunmode { my $cgiapp = shift;
    my @clients = $schema->resultset('Client')->all;
    $cgiapp->tt_params( clients => \@clients);
    return $cgiapp->tt_process('liste.tt2'); }

#!/usr/local/perl/bin/perl5.8.5 -Tw
my $appli = MonAppli->new(); $appli->run();

<table class="liste"> <tr><th>No</th><th>NOM</th>
[% FOREACH client = clients %]
<tr><td>[% client.numero %]</td><td>[% client.nom %]</td></tr>
[% END %]
```

L'évolution de l'humanité...

des gestes...

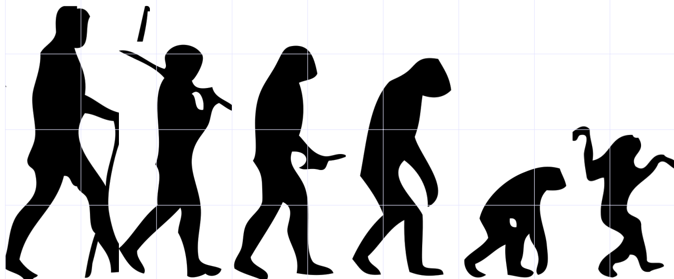
...au langage articulé



... et celle de l'informatique

du langage articulé...

...à la souris



L'approche verticale

- Approche verticale : pratiquer une pile de protocoles complète,
- pour acquérir un savoir-faire "universel".

SQL	HTML
DBI	TT2
CGI::Application	
Perl	
CGI	
HTTP	
TCP	
IP	
Ethernet	

Quelques bonnes pratiques

- Methodes de conception des applications web
  - documenter les relations entre specifications (besoins) et les composants logiciels,
  - documenter les decisions de conception,
  - produire des prototypes/squelettes d'applications en fonction de modeles (Model Driven Architecture), ou utiliser des environnements integrant ces modeles (Frameworks).
  - automatiser les tests,
- Un objectif : le projet = 80h * 2 (binôme)

On insiste sur une architecture de base : MVC

- changement facile de modèle (SGBD support, ...) si API respectée
- changement facile de vue (écran, smartphone, voix...) idem...
- testabilité accrue de chacun des composants

Perl sait faire du MVC (et pas que d'une façon)

- C -> CGI::Application, etc...
- M -> Class::DBI, DBIx::Class, etc...
- V -> TT, Mason, HTMLTemplate, etc...
- Frameworks complets : Catalyst, Openinteract, ...

Les dangers se cachent partout...

- Le langage de template inclut tout ce qu'il faut (procédural, présentation..)
- Des "plugins" étendent les fonctionnalités de TT, par exemple Dumper

```
[% USE Dumper %]
[% Dumper.dump_html(structure_de_donnees) %]
```

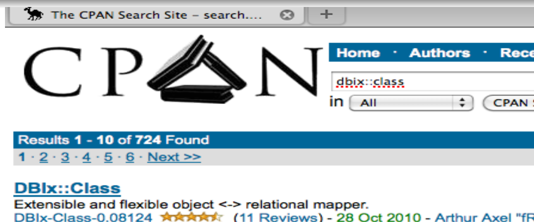
- Mais aussi (malheureusement) DBI :

```
[% USE DBI('dbi:mysql:test') %]
<table border=0 width="100%">
<tr><th>User ID</th> th>Name</th><th>Email</th></tr>
[% FOREACH user = DBI.query('SELECT * FROM user ORDER BY id') %]
   <tr>
      <td>[% user.id %]</td>
      <td>[% user.name %]</td>
      <td>[% user.email %]</td>
   </tr>
[% END %]
</table>
```

On se construit son IDE personnalisé

- Ajout de plugins de debug.

```
use CGI::Application::Plugin::DevPopup;
use CGI::Application::Plugin::DevPopup::Timing;
use CGI::Application::Plugin::ViewCode;
use CGI::Application::Plugin::DebugScreen;
BEGIN {
    $ENV{'CAP_DEVPOPUP_EXEC'} = 1;
    $ENV{'CGI_APP_DEBUG'} = 1;
}
use Data::Dumper::HTML qw(dumper_html);
$cgiapp->ajouter_message_information(dumper_html($q));
```



- à chaque table correspond une classe :

```
package GestCli::Client;
use base 'DBIx::Class';
__PACKAGE__->load_components(qw/InflateColumn::DateTime .../);
__PACKAGE__->table('client');
__PACKAGE__->add_columns(
  "cli_numero", {data_type=>"INT", is_nullable=>0, size=>11 },
  "cli_nom", {data_type=>"CHAR", is_nullable=>1, size=>25},
.../... );
__PACKAGE__->set_primary_key("cli_numero");
__PACKAGE__->has_many( "commandes", "GestCli::Commande",
    { "foreign.cmd_cli_numero" => "self.cli_numero" }, );
```

Génération automatique des classes de l'ORM

```
use DBIx::Class::Schema::Loader qw/ make_schema_at /;
my $classe_base     = 'GestCli';
my $repertoire_base = '.';
my $dsn = 'dbi:mysql:test:localhost';

make_schema_at( $classe_base,
    {
        relationships => 1,
        components    => [qw/InflateColumn::DateTime Core/],
        dump_directory => $repertoire_base,
    },
    [ $dsn, $user, $password, ],);
```

l'ORM facilite l'écriture des méthodes du modèle

- La méthode de connexion ramène un objet schema,

```
package MonAppli;
use GestCli;
my $schema = GestCli->connect('dbi:mysql:test','otton','');
```

- Ce schéma possède autant de "resultsets" que de tables.

```
my @clients = $schema->resultset('Client')->all;
my @clients_albigeois
  = $schema->resultset('Client')->search({cli_cp=>'81000'});
```

- Les objets obtenus disposent d'accesseurs aux attributs :

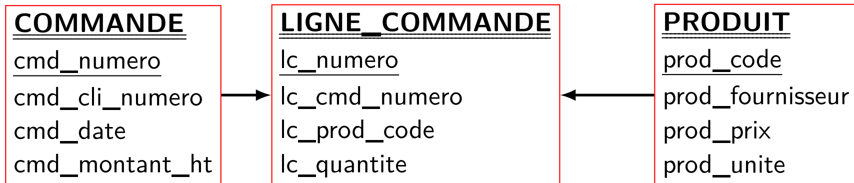
```
my $client = $schema->resultset('Client')->find(17);
my $nom_client = $client->cli_nom;
$client->cli_cp('31000');
$client->update;
```

Gestion des liens 1->n et n->m

- Les accesseurs 1->N sont génerés automatiquement (si contrainte foreign key)

```
__PACKAGE__->has_many( "commandes", "GestCli::Commande",
    { "foreign.cmd_cli_numero" => "self.cli_numero" }, );
.../...
my $client = $schema->resultset('Client')->find(17);
for my $commande ($client->commandes) {
    .../...
}
```

Association N vers M




```
GestCli::Commande->has_many(
    "ligne_commandes", "GestCli::LigneCommande",
    { "foreign.lc_cmd_numero" => "self.cmd_numero" }, );
GestCli::LigneCommande->belongs_to(
    "lc_cmd_numero", "GestCli::Commande",
    { cmd_numero => "lc_cmd_numero" }, );
GestCli::LigneCommande->belongs_to(
    "lc_prod_code","GestCli::Produit",
    { prod_code => "lc_prod_code" }, );
GestCli::Produit->has_many(
    "ligne_commandes","GestCli::LigneCommande",
```

Association N vers M

- En suivant les liens "commande has many lignes", puis "ligne belongs to produit", on peut écrire :

```
my $commande = $schema->resultset('Commande')->find(1);
for my $ligne ( $commande->ligne_commandes ) {
    my $produit = $ligne->lc_prod_code; .../...  }
```

- Des accesseurs N->M peuvent être ajoutés facilement :

```
# DO NOT MODIFY THIS OR ANYTHING ABOVE! md5sum:atBclI/rjLktNT48oY/
GestCli::Commande->many_to_many(
    'produits', 'ligne_commandes', 'lc_prod_code' );
```

- On peut alors simplifier en ceci :

```
my $commande = $schema->resultset('Commande')->find(1);
for my $produit ( $commande->produits ) { .../...  }
```

- Une méthode de création "add_to_xxxxs" est disponible :

```
$commande->add_to_produits( { prod_code => 'P1', });
```

On fait rapidement un premier écran

- dans le runmode liste, on lit les données dans un tableau,
- puis on transfère la référence du tableau au template :

```
my @lignes = $schema->resultset('Client')->all;
$cgiapp->tt_params( liste => \@lignes);
return $cgiapp->tt_process('liste.tt2');
```

- Il suffit d'ajouter une boucle d'affichage dans le template :

```
<table class="liste">
<tr><th>No</th><th>NOM</th>
[% FOREACH ligne = liste %]
    <tr><td>[% ligne.cli_numero %]</td>
        <td>[% ligne.cli_nom %]</td>
        <td><a href="?rm=formulaire_modification
            &cli_numero=[% ligne.cli_numero %]">
            modifier</a></td>
    </tr>
[% END %]
</table>
```

On sensibilise aux dangers du web

- récupérer (et valider !) des données saisies dans un formulaire web : tache courante mais difficile ;
- CGI::param ne fait qu'une partie du travail ;
- la saisie externe est potentiellement dangereuse
- le mode taint : tout ce qui vient d'une saisie externe est marqué comme potentiellement dangereux, et se trouve interdit d'utilisation s'il y a un risque.
- pour utiliser des valeurs "tainted" dans des usages dangereux, il faut les "dé-tainter" (untaint), et donc les vérifier.
- pour un-tainter une variable, il faut la faire passer par un "match" sur une expression régulière.
- Le module CGI::Untaint est un cadre permettant de vérifier/untainter les données venant d'un formulaire.

Dans un runmode ou on souhaite vérifier des données de requete (formulaire ou GET), il faut effectuer les etapes suivantes :

- Créer un objet CGI::Untaint en lui passant le tableau associatif des parametres venant de la requete HTTP (methode Vars de l'objet CGI, lui-meme obtenu par la methode query sur l'objet CGI::Application).
- L'objet CGI::Untaint obtenu permet à la fois de vérifier et d'extraire les données, par la methode "extract", à laquelle on specifie le mode de vérification à effectuer.

```
sub modification_client : Runmode {
  my $cgiapp = shift;
  my $q = $cgiapp->query;
  my $cli_numero = $q->param('cli_numero');
  my $untaint_handler = CGI::Untaint->new($q->Vars);
  my $name = $untaint_handler->extract(-as_printable => 'name');
```

Règles de validation indiquées sous forme déclarative

```
use Data::FormValidator;
my $results = Data::FormValidator->check(\%input_hash, \%dfv_profi
if ($results->has_invalid or $results->has_missing) {
    # traiter erreurs, afficher $results->msgs ...
}
else { # utiliser les données de $results->valid
}
```

- Data::FormValidator fournit la méthode check, deux arguments :
  - référence aux données à valider (objet "query" du module CGI),
  - référence à un "profil" de validation = structure décrivant les règles de validation :

```
my $profil = {
    optional => [qw( cli_cp cli_ville)],
    required => [qw( cli_nom cli_adresse )],
    constraint_methods => { cli_mail => email(), },
};
```

Profil de validation

- La structure du profil de validation est très riche, elle permet d'exprimer des dépendances :

```
dependencies    => {
    # si adresse saisie, alors ville obligatoire:
    "cli_adresse" => [ qw( cli_ville ) ],
    # si mode de paiement='cheque', alors numéro chèque obligatoir
    "mode_paiement" => {
        cheque => [ qw( numero_cheque ) ],
        }
},
```

- des dépendances de groupe :

```
dependency_groups  => {
# si l'un des champs est rempli, les deux sont obligatoires:
    password_group => [qw/password password_confirmation/],
}
```

- On peut aussi appliquer filtres de transformation, contraintes.

Intégration de CGI et Data::FormValidator

- Pour minimiser le code d'interface entre CGI et Data::FormValidator,
- Ajouter le composant FromValidators" aux options du Schema::Loader permet à toutes les DataSources créées d'heriter des méthodes create_from_fv et update_from_fv .
- Ces méthodes prennent en entrée le résultat de l'opération de validation,
- et transmettent à DBIx::Class ces données,
- puis effectuent la création ou la mise à jour.

```
my $profil = GestCli->dfv_profile_client;
my ($resultats) = Data::FormValidator->check($q, $profil);
if ($resultats->has_invalid or $resultats->has_missing) {
    $cgiapp->ajouter_message('erreur', "Il y a des erreurs");
    $cgiapp->tt_process('formulaire_client.tt2');
} else {
    my $nouvelle_ligne = $schema->resultset('Client')
        ->create_from_fv($resultats);
}
```

J'enrichis mon interface facilement

- Plusieurs modules permettant de paginer des jeux de données existent : Data::Page, Data::Pageset, Data::SpreadPagination.
- DBIx::Class utilise par défaut Data::Page.
- Il suffit de fournir un paramètre page à la création du resultset, un objet Data::Page est créé simultanément.
- Il est accessible par la méthode "pager", il FAUT le transmettre au template pour pouvoir l'utiliser.

```
my $lignes_par_page = 5;
my $page = 1;
if (my $p_demandee = $untaint_handler->extract(-as_integer => 'pag
    $page = $p_demandee }
.../...
my $rs = $schema->resultset('Client')->search( $where,
    { page => $page, rows => $lignes_par_page,
.../...
$cgiapp->tt_params( pager => $rs->pager);
```

- dans le template, on utilise les données de l'objet pager pour afficher des liens de pagination :

```
[% IF pager.previous_page %]
<a href="?rm=liste&page=[% pager.previous_page %]">
les [% pager.entries_per_page %] precedents</a> |
[% END %]

[% FOREACH num = [pager.first_page .. pager.last_page] %]
[% IF num == pager.current_page %]  page [% num %] 
[% ELSE %]<a href="?rm=liste&page=[% num %]">[[% num %]]</a>[% END
[% END %]

[% IF pager.next_page %]
<a href="?rm=liste&page=[% pager.next_page %]">
les [% pager.entries_per_page %] suivants</a>
[% END %]
```

On insiste sur les tests !

- une application non testée ne peut être mise en exploitation
- une application qui n'a pas de plan de tests ne peut être testée
- une application qui ne peut être testée que manuellement sera un jour modifée sans être testée...
- Tous les langages proposent des modules de test automatise :

```
use Test::More tests => 2;
$a = 2;
$b = 3;
$produit = $a * $b;      # calcul produit
$somme = $a * $b;        # calcul somme
ok( $produit == 6, "Le produit de $a * $b = 6" );
ok( $somme == 5, "La somme $a + $b = 5" );
```

client web "mécanisé" : WWW::Mechanize

- WWW::Mechanize permet de programmer une navigation ;

```
use WWW::Mechanize;
use Test::More tests => 1;

my $mech = WWW::Mechanize->new();
$mech->get('http://sevres.enstimac.fr/cgi-bin/essai.cgi');

$mech->form_number(1);
$mech->submit_form(
    fields      => {
        champ1      => 'valeur1',
        champ2      => 'valeur2',
    }
);

like( $mech->content(), qr/tout va bien/,
    'le résultat contient "tout va bien"');
```

WWW::Mechanize::CGI

- Test::WWW::Mechanize : :CGI permet de naviguer sur un programme CGI sans serveur http.

```
use Test::WWW::Mechanize::CGI;
use Test::More tests => 6;
my $mech = Test::WWW::Mechanize::CGI->new;
$mech->cgi_application('./gestcli99.cgi');

$mech->get_ok('http://localhost/');
$mech->title_is('Liste des éléments');
$mech->content_contains("Créer un nouveau client");
$mech->follow_link( text_regex => qr/Créer un nouveau client/i );
like( $mech->content(), qr/Creation du client/, "formulaire OK");
$mech->submit_form( fields => {
    cli_nom     => 'LE-NOM',
    cli_prenom  => 'LE-PRENOM',
    .../...
    cli_ville   => 'ALBI', }
);
like( $mech->content(), qr/Client créé/,
    "OK: présence message réussite création");
```