Création d'une extension PHP

Extension pour le moteur d'inférence CLIPS

Notre besoin

- Choix d'un langage ouvert dans lequel décrire une quantité importante de règles à partir de d'informations récoltées.
- L'application :
  - En 'stand-alone' sur un poste client,
  - En client-serveur,
  - Au travers un serveur Web.
- Portable sous Win32 et Unix/Linux.

La maintenance des règles

- Maintenir les règles de dédection des anomalies au travers un nombre important de faits détectés:
  - En ajouter,
  - Intégrer les cas d'exceptions, pour affiner le règle,
  - Comparer des faits entre eux,
  - ...

Le choix de langage

- Aucun langage classiquement utilisé ne répond facilement à ces besoins.
- Seul un langage possédant un moteur d'inférence est adapté.
- Le choix c'est porté sur CLIPS : sources libres, langage ouvert, interfaçable, évolutif.
- Ce langage est lui même développé en C Ansi.

Le choix de langage (2)

- Mais pour faire une application adaptée au Web, il fallait l'associer à un autre langage.
- Php, Java, C/C++ semblent eligibles.
  - PHP et son processus d'extension.
  - Java et sa passerelle JNI.
  - C/C++ associé à des sources C Ansi.

Cas extension à PHP

- Pour notre cas nous ne traiterons que du couple PHP - CLIPS.
- Et nous allons étudier comment intégrer le langage CLIPS dans le PHP en créant une extension.
- Mais avant tout une définition de l'inférence.

Moteur d'inférence ?

- Définition de l'inférence:
  - Opération qui consiste à tirer une conclusion d'une série de propositions reconnues pour vraies (premisses).

- Notre besoin était d'extérioriser toutes les règles de déductions sur des faits précis (le dom HTML, le rendu du navigateur...) pour emettre d'autres faits : les remarques à faire sur les flux analysés.

Configuration de l'extension

Récupération des sources PHP

- Pour permettre la recompilation de PHP il faut récupérer les sources.

  http://www.php.net/downloads.php

- Les répertoires importants de `php4` (ou `php5`):

  `ext` : les modules déjà intégrés, c'est ici qu'il faut créer le notre.

  `main` : dossier comprenant le fichier `php.h` et `php_ini.h`.

  `Zend` : dossier du moteur Zend..

Squelette d'un module d'extension

- Système de squelette automatique dans dossier `ext`.
- Fichier de commande : `ext_skel`
- Exemple pour générer le squelette du module clips:

```
> cd <chemin de PHP>/ext
> ./ext_skel --extname=clips
```

- Création du dossier `clips` dans `ext`.

Le fichier de configuration

- Le squelette crée un fichier de configuration `config.m4`
- Ce fichier M4 permet de paramétrer les options de ligne de commande passées au script `configure`
- Il crée pour notre cas deux directives:

```
--with-clips
--enable-clips
```

Les directives de configuration

- `--with-clips` : cette option fait référence à des fichiers externes et permet d'intégrer le module dans PHP.
- `--enable-clips` : cette option permettra l'intégration du module de façon externe avec PHP (appel avec `dl()`).

Modification de `config.m4`

- Par défaut sont définies:
  - Les deux directives :
    - `PHP_ARG_WITH`,
    - `PHP_ARG_ENABLE`,
    - Elles sont en commentaire : `dnl` en début de ligne.
    - Il faudra donc choisir son mode d'intégration statique ou dynamique.
  - L'activation du module :
    - `PHP_NEW_EXTENSION`.

Modification de `config.m4` (2)

- Si on crée avec `ext_skel` le module `mon_module`, la directive statique :

```
dnl PHP_ARG_WITH(mon_module, for mon_module support,
dnl Make sure that the comment is aligned:
dnl [ --with-mon_module Include mon_module support])
```

- La directive dynamique :

```
dnl PHP_ARG_ENABLE(mon_module, whether to enable mon_module support,
dnl Make sure that the comment is aligned:
dnl [  --enable-mon_module Enable mon_module support])
```

Modification de `config.m4` (3)

- L'extension sera définie si le module est bien référencé :

```
if test "$PHP_MON_MODULE" != "no"; then
....
fi
```

- La variable `$PHP_MON_MODULE` est le résultat de la directive 'with' ou 'enable'.

Modification de `config.m4` (4)

- On peut définir plusieurs 'directives' dans le même module. Exemple avec le module `CURL`:

```
PHP_ARG_WITH(curl, for CURL support,
[  --with-curl[=DIR]       Include CURL support])

PHP_ARG_WITH(curlwrappers, if we should use CURL for url
streams,
[  --with-curlwrappers     Use CURL for url streams], no, no)
...
if test "$PHP_CURL" != "no"; then
....
  if test "$PHP_CURLWRAPPERS" != "no" ; then
  ....
  fi
...
fi
```

Modification de `config.m4` (5)

- Activation de l'extension pour `mon_module` :

```
PHP_NEW_EXTENSION(mon_module, mon_module.c, $ext_shared)
```

- On fait référence ici au source C du module et aux extensions partagées.

Modification de `config.m4` (6)

- Pour notre cas `clips` :

```
PHP_ARG_WITH(clips, for CLIPS support,
[  --with-clips      Include clips support],yes)
if test "$PHP_CLIPS" != "no"; then
  extra_sources="clipssrc/agenda.c \
    clipssrc/analysis.c clipssrc/argacces.c \
    ...
    clipssrc/watch.c"
  PHP_NEW_EXTENSION(clips,
      clips.c $extra_sources,
      $ext_shared)
fi
```

Configuration automatique

- Pour inclure le module créé dans la phase de configuration automatique et de compilation il faut utiliser le script `buildconf`.
- Ce script est à la racine du code source PHP.
- Il se peut qu'il faille utiliser l'argument `--force` pour lancer l'exécution du script.
- A chaque changement de `config.m4` il faudra relancer le script `buildconf`.

Code source de l'extension

Le code généré par `ext_skel`

- Dans notre cas, le squelette a créé dans le dossier de l'extension `clips` le fichier source C `clips.c`.
- La principale en-tête standard :

```
#include "php.h"
```

- Cet en-tête est défini dans le dossier `main` de PHP.
- Ce fichier inclut toutes les macros et API nécessaires pour la fabrication du module.

Le code généré par `ext_skel` (2)

- Ce script utilise un dossier d'extension spécifique `skeleton`
- Attention les fichiers squelettes n'utilisent pas les recommandations de la documention en terme de nommage des macros.
- PHP_FE au lieu de ZEND_FE...
- En réalité les macros PHP_* et ZEND_* sont équivalentes.

Le fichier clips.c

- La première zone importante, le tableau de fonctions :

```
function_entry clips_functions[] = {
  PHP_FE(confirm_clips_compiled, NULL)
  {NULL, NULL, NULL}  /* Must be the last line in
                         clips_functions[] */
};
```

- Ici on doit déclarer les fonctions à exporter vers PHP.

Le tableau de fonctions

- Logiquement on aurait du créer :

```
zend_function_entry clips_functions[] = {
  ZEND_FE(confirm_clips_compiled,NULL)
  {NULL, NULL, NULL}  /* Must be the last line in
                         clips_functions[] */
};
```

- Si l'on veut donc se conformer à la documentation, il faut modifier.

Le tableau de fonction (2)

- `{NULL, NULL, NULL}` est destiné à Zend, pour qu'il puisse reconnaître la fin de la liste de fonctions exportées.
- Par défaut le squelette fait référence à une fonction de test qu'il a construit :

```
confirm_clips_compiled
```

- Suivent les déclarations des informations spécifiques au module.

Déclaration des informations du module

```
zend_module_entry clips_module_entry = {
   STANDARD_MODULE_HEADER,
   "clips",
   clips_functions,
   PHP_MINIT(clips),
   PHP_MSHUTDOWN(clips),
   PHP_RINIT(clips),     /* Replace with NULL if there's nothing to
                            do at request start */
   PHP_RSHUTDOWN(clips), /* Replace with NULL if there's nothing to
                            do at request end */
   PHP_MINFO(clips),
   "0.1", /* Replace with version number for your extension */
   STANDARD_MODULE_PROPERTIES

};
```

Déclaration des informations du module (2)

- Conformément à la documention, il faudrait changer PHP_ en ZEND_.
- Si vous n'avez pas de fonction spécifique à l'initialisation et l'arrêt du module on peut mettre à NULL à la place des références concernées.

Déclaration des informations du module (3)

- Par contre si elles sont utilisées elles sont associées à : `<référence>_FUNCTION`.
- Exemple :

`PHP_MINIT` est associé à `PHP_MINIT_FUNCTION`.

Ou

`ZEND_MINIT` est associé à `ZEND_MINIT_FUNCTION`.

Déclaration des informations du module (4)

- Une des références interessante, `ZEND_MINFO`.
- Elle déclare la fonction d'affichage des informations dans `php_info()`.
- Elle est associée à `ZEND_MINFO_FUNCTION`.

Déclaration de module dynamique

- La zone de déclaration du module :

```
#ifdef COMPILE_DL_CLIPS
    ZEND_GET_MODULE(clips)
#endif
```

- Cette fonction ne sera compilée que si le module est dynamique.
- C'est la définition `COMPILE_DL_CLIPS` qui permet au compilateur d'intégrer la déclaration.
- Donc si statique => inutile.

La fonction externe créée en standard

- La fonction exportée intégrée par le squelette :

```
PHP_FUNCTION(confirm_clips_compiled)
{
    char *arg = NULL;
    int arg_len, len;
    char string[256];

    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
            "s", &arg, &arg_len) == FAILURE) {
        return;
    }

    len = sprintf(string, "Congratulations! You have \
        successfully modified ext/%.78s/config.m4. \
        Module %.78s is now compiled into PHP.", "clips", arg);
    RETURN_STRINGL(string, len, 1);
}
```

La fonction externe créée en standard (2)

- Normalement la déclaration devrait être faite avec `ZEND_FUNCTION`
- Elle est associée à la déclaration `PHP_FE` (`ZEND_FE`) de la table des fonctions.

La fonction externe créée en standard (3)

- Que fait cette fonction exportée :
  - Elle teste et récupère l'argument string passé,
  - Elle génére et renvoie à l'appelant une chaîne de caractères contenant notemment l'argument passé.
  - Cette fonction est mise en place dans le fichier test clips.php généré par le squelette dans le dossier du module.

Traitement de l'argument passé

- La fonction teste et récupère l'argument passé :

```
if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
          "s", &arg, &arg_len) == FAILURE) {
       return;
}
```

- On teste avec la fonction Zend `zend_parse_parameters`
- Si on n'est pas satisfait (`FAILURE`) on ne retourne rien! Dans le cas contraire (`SUCCESS`) on continue.

zend_parse_parameters

- Le premier argument de la fonction Zend est le nombre d'arguments passé à la fonction externe.
- Ici normalement 1 seul.
- Le deuxième paramètre doit toujours être la macro TSRMLS_CC.
- Le troisième argument est une chaîne qui spécifie le nombre et le type des arguments attendus par la fonction externe,

`zend_parse_parameters` (2)

- Ce troisème argument est dans un format proche de celui utilisé par printf.
- Le reste des arguments sont des pointeurs vers les variables qui recevront les valeurs de ces paramètres.
- Cette fonction Zend effectue des conversions de type lorsque c'est possible, on doit recevoir les données dans le format demandé.

Spécificateurs de type - Référence :
http://www.zend.com/apidoc/zend.arguments.retrieval.php

- Ce troisième argument, est un de ces spécificateurs de type :
  - l : long,
  - d : double,
  - s : chaîne de caractères (avec possibilité de caractère null) et sa longueur,
  - b : booléen,

Spécificateurs de type (2)

- r : ressource, stockée dans un zval* ,
- a : tableau, stocké dans un zval* ,
- o : objet (d'une classe), stocké dans un zval*,
- O : objet (de classe spécifiée par un « zend_class_entry »), stocké dans un zval* ,
- z : un zval*,

Spécificateurs de type (3)

- Dans notre exemple on a simplement la valeur :

  `"s"`

- L'argument passé par le fonction externe doit être une chaîne de caractère qui sera lue ainsi que sa longueur.
- Cet argument est récupéré par référence `&arg`,et la longueur de la chaîne passée dans `&arg_len`.

Spécificateurs de type (4)

- Certains caractères on des significations dans la chaîne des spécificateurs :
  - « | » : indique que la suite des paramètres sont optionnels. Les variables de stockage de ces paramètres doivent être initialisées à leur valeur par défaut (définie par l'extension).

Spécificateurs de type (5)

- Exemple :
  - Lit un objet de classe spécifiée par le `zend_class_entry` my_ce, et un double optionnel.

```
zval *obj;
double d = 0.5;
if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
          "O|d", &obj, my_ce, &d) == FAILURE) {
    return;
}
```

Spécificateurs de type (5)

- « / » : La fonction d'analyse va appeler `SEPARATE_ZVAL_IF_NOT_REF()` sur le paramètre précédent, pour fournir une copie du paramètre, à moins que cela ne soit une référence.

  Exemple, qui lit un tableau :

```
zval *arr;
if (zend_parse_parameters(ZEND_NUM_ARGS()
TSRMLS_CC,
       "a/", &arr) == FAILURE) {
    return;
}
```

Spécificateurs de type (6)

- « ! » : Le paramètre précédent peut être du type spécifié ou NULL (uniquement pour a, o, O, r, et z). Si NULL est passé par l'utilisateur, le pointeur de stockage sera NULL .

  Exemple:

  Lit un objet ou null, et un tableau. Si null est passé comme objet, obj prendra la valeur NULL :

```
zval *arr;
zval *obj;
if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
   "O!a", &obj, &arr) == FAILURE) {
      return;
}
```

Traitement de la fonction externe

- Une fois le test du passage de l'argument passé, la fonction fabrique une chaîne destinée a être retournée :

```
len = sprintf(string,
   "Congratulations! You have successfully \
   modified ext/%.78s/config.m4. Module %.78s \
   is now compiled into PHP.", "clips", arg);
RETURN_STRINGL(string, len, 1);
```

Utilisation de la fonction externe

- Le squelette a créé un fichier de test PHP, permettant entre autre de tester la fonction standard intégrée. Ce fichier est ici `clips.php` :

```
$module = 'clips';
...
$function = 'confirm_' . $module . '_compiled';
if (extension_loaded($module)) {
    $str = $function($module);
} else {
    $str = "Module $module is not compiled into PHP";
}
echo "$str\n";
```

Utilisation de la fonction externe (2)

- La fonction renvoie la chaîne :

```
Congratulations! You have successfully
modified ext/clips/config.m4. Module clips
is now compiled into PHP.
```

- Le premier **clips** correspond à la chaîne fixée dans le sprintf.
- Le deuxième **clips** correspond au paramètre passé par l'appel de la fonction.

Intégration des fonctions externes spécifiques

- Pour intégrer maintenant les programmes de notre extension, il faut:
  - Spécifier les fonctions externes que l'on veut utiliser dans PHP, à l'aide de la macro `PHP_FUNCTION` (`ZEND_FUNCTION`).

```
PHP_FUNCTION(clips_init)
...
PHP_FUNCTION(clips_messages)
...
PHP_FUNCTION(clips_reset)
...
PHP_FUNCTION(clips_clear)
...
```

Intégration des fonctions externes spécifiques (2)

- Faire référence à ces fonctions dans le tableau de fonctions à l'aide de la macro `PHP_FE` (`ZEND_FE`).

```
PHP_FE(clips_init, NULL)
PHP_FE(clips_messages, NULL)
PHP_FE(clips_reset, NULL)
PHP_FE(clips_clear, NULL)
...
```

La fonction spécifique 'callback'

- Clips permet de définir une fonction externe callback qui sera appelée par le moteur pour renvoyer un flux de caractères sur un canal spécifique: sortie standard, erreur, alerte, ...
- Vous pouvez appeler des fonctions définies par l'utilisateur dans le script PHP (dites fonctions utilisateurs)...
- Il est donc possible du côté extension de récupérer une fonction définie par l'utilisateur et de la passer au moteur d'inférerence.

La fonction spécifique 'callback' (2)

- Côté Clips on définit dans l'extension à l'initialisation la référence aux fonctions captureQuery et capturePrint:

```
if(AddRouter("capture", 20,
   (int (*)(char *))captureQuery,
   (int (*)(char *,char *))capturePrint,
   NULL, NULL, NULL))  {
         m_fClipsInit = true;
         return true;
   }
```

- Ici c'est surtout la fonction capturePrint qui nous intéresse c'est elle qui va donner la référence de la fonction côté PHP qu'il faut 'réveiller' si le moteur d'inférence a quelque chose à dire!

La fonction spécifique 'callback' (3)

- On définit dans l'extension la fonction capturePrint, qui a pour rôle de donner le nom de la fonction PHP qui a été mise dans le buffer `_clips_out_msg` :

```
int capturePrint(char *router, char *str)
  {
  ...
  if(call_user_function_ex(CG(function_table),
     NULL, _clips_out_msg, &retval, 2, args, 0,
     NULL TSRMLS_CC) != SUCCESS)
  {
     zend_error(E_ERROR, "Function call failed");
     return 1;
  }
  ...
  }
```

La fonction spécifique 'callback' (4)

- Cette fonction utilisateur est appelée avec la fonction `call_user_function_ex`.
- Cette fonction nécessite une valeur de hash pour la table de fonctions `CG(function_table)`, un pointeur vers un objet (pour nous NULL), le nom de la fonction (que l'on a mis dans un zval), une valeur de retour, un nombre d'arguments, un tableau d'arguments, et une option qui indique si on souhaite faire une séparation zval (NULL pour nous).

La fonction spécifique 'callback' (5)

- Maintenant il faut créer la fonction externe qui va permettre de véhiculer le nom de la fonction PHP qui sera 'réveiller' par le moteur d'inférence.
- Ce nom est déposer dans le zval utilisé par capturePrint.
- La fonction clips_messages met à jour `_clips_out_msg`:

```
MAKE_STD_ZVAL(_clips_out_msg);
_clips_out_msg->type = IS_STRING;
_clips_out_msg->value.str.len = (*fonction)->value.str.len;
ZVAL_STRINGL (_clips_out_msg, (*fonction)->value.str.val,
      (*fonction)->value.str.len, 1);
```

Utilisation de fonction spécifique 'callback'

- L'ordre logique côté PHP sera de donner la fonction 'messages' avant d'initialiser.
- Puisque c'est la fonction d'initialisation qui donne à Clips le nom de la fonction côté PHP.
- Dans notre exemple :

```
echo "clips_messages() = " . clips_messages("message").
  "<br>\n";

echo "clips_init() = " . clips_init() . "<br>\n";
```

Utilisation de fonction spécifique 'callback' (2)

- La fonction messages est déclarée dans le fichier `clips.inc`.

```
function message($router,$msg) {
   echo "message - $router : $msg\n";
}
```

- Quand le moteur de Clips a quelque chose à dire, par exemple :

```
message - wtrace : ==>
message - wtrace :  instance
...
message - wdialog : *** MEMORY DEALLOCATED ***
....
```

Fabrication du nouveau PHP avec son extension

Make

- Notre source de l'extesion finalisé, on peut lancer la fabrication du nouvel exécutable et ses composants associés.
- Pour cela : `make`.
- A partir lu répertoire racine des source php.
- Le résultat du nouveau php se trouve, dans dossier `sapi`.

Nouveau PHP

- Sur répertoire `cli`, l'exécutable en mode client.
- Sur répertoire `cgi`, l'exécutable en mode CGI.
- Plus tout le code pour les différentes interfaces avec les serveurs HTTP.

Démonstration

Merci de votre attention!