

Programare PHP orientată pe obiecte

Programarea orientată pe *obiecte* (OOP – *Object Oriented Programming*) a apărut ca o necesitate în contextul creșterii complexității codului aplicațiilor software. Pentru aplicațiile de mari dimensiuni, o dezvoltare structurată a codului (orientată pe funcții/proceduri) implicând existența unui număr foarte mare de linii de cod (uneori puternic redundant), prin modul de organizare a codului conduce la o lizibilitate scăzută a acestuia și implicit, la mari dificultăți privind realizarea unor modificări ulterioare în cadrul aplicației.

OOP oferă o modalitate diferită de organizare a codului și a datelor în cadrul unui program. Din acest punct de vedere, elementele constructive de cod specifice OOP sunt clasa, respectiv obiectul. Clasa reprezintă definiția unui obiect (“planul obiectului”). Prin instanțierea unei clase este creat un obiect (evident se pot face instanțieri multiple, construindu-se mai multe obiecte ale aceleiași clase). În cadrul clasei (definiția obiectului) sunt precizate:

- atribute sau proprietăți – (practic partea de date a obiectului) reprezentate prin declarații de variabile, inclusiv posibile inițializări ale acestora;
- metode – (partea de cod a obiectului) reprezentate prin funcții (sau proceduri) constituind totodată și interfața obiectului destinată manipulării datelor acestuia.

OOP este fundamentată pe 3 principii de bază:

- încapsulare – fiecare obiect este de sine stătător și complet autonom (conținând atât date - proprietăți-, cât și cod –metode). Un obiect are o interfață clară, bine definită, folosită pentru a manipula obiectul;
- moștenire;
- polimorfism.

Asupra ultimelor două se va reveni în contextul programării în PHP (existând câteva particularități specifice acestuia). Particularizat la PHP, în continuare sunt tratate următoarele aspecte:

- Crearea unei clase (proprietăți, metode)

- Crearea unui obiect (instanțierea clasei)

- Utilizarea proprietăților obiectului

- Apelul metodelor obiectului

- Moștenirea

- Polimorfismul

În limbajul PHP, crearea unei clase se face utilizând instrucțiunea *class*. O definiție minimală a unei clase, este prezentată mai jos:

```
class denumire_clasa {  
  
}
```

În vederea atribuirii unei funcționalități clasei este necesară definirea unor proprietăți și metode. Proprietățile se creează prin declararea unor variabilelor la începutul definiției unei clase folosind instrucțiunea *var*. Următoarea secvență PHP creează o clasă denumită *Clasa1* având două atribute (proprietăți): *\$message*, *\$data* (ultimul fiind și inițializat).

```
class Clasa1 {  
var $message;  
var $data="initializat";  
}
```

Metodele se creează prin declararea unor funcții PHP în definiția clasei. Codul metodei *setMessage*, având un parametru de intrare (*\$param*) și permițând o setare a valorii proprietății *\$message*. De remarcat că, în cadrul definiției clasei, referirea unei proprietăți a acesteia se face folosind operatorul *\$this* care precede numele proprietății (nume utilizat fără \$ în față). În cazul de față: *\$this->message*.

- metoda *getMessage*, fără parametri de intrare, care afișează un mesaj, respectiv returnează valoarea proprietății *\$message*.

```
<?php  
class Clasa1  
{
```

```

var $message;
var $data="initializat";
function setMessage($param)
{
    $this->message = $param;
}
function getMessage()
{
    echo "Mesajul pentru obiectul 1 este:<br>";
    return $this->message;
}
}
// . . .cod PHP instanțiere clasă
?>

```

Evident s-a inclus codul în tag-urile de delimitare specifice PHP. După ce a fost creată clasa, în continuare se prezintă modul de instanțiere a clasei în vederea creării unui obiect, precum și modul de setare a proprietăților acestuia și de apel al metodelor (prin completarea secvenței anterioare, după încheierea definiției clasei):

```

$Obiect1 =new Clasa1();
$Obiect1->setMessage("Setarea proprietate folosind o metoda");
echo $Obiect1->getMessage();
$var=$Obiect1->getMessage();
echo $var;
$Obiect1->message="Setare directa a proprietatii printr-o atribuire ";
echo $Obiect1->getMessage();
echo $Obiect1->data;

```

Rezultatul rulării scriptului este următorul:

Mesajul pentru obiectul 1 este:

Setarea proprietate folosind o metoda

Mesajul pentru obiectul 1 este:

Setarea proprietate folosind o metoda:

Mesajul pentru obiectul 1 este:

Setare directa a proprietatii printr-o atribuire

initializat

Ulterior declarării unei clase, este necesara crearea un obiect cu care acesta să opereze. Operația este denumita instanțierea unei clase sau crearea unei instanțe. În limbajul PHP pentru aceasta operație, se utilizează instrucțiunea, cuvântul cheie new.

- Deci prima linie din secvența de cod anterioară creează un nou obiect Obiect1 prin instanțierea clasei Clasa1.

- În continuarea se face un apel al metodei setMessage (\$Obiect1->setMessage) pasându-i un parametru, metoda care permite setarea proprietății message. Linia următoare apelează metoda getMessage (fără parametrii), care afișează un mesaj și returnează o valoare (a proprietății message, afișată pe ecran folosind echo).

- Valoarea returnată putea fi evident memorată într-o variabilă (\$var în cazul de față).

- În acest caz, setarea proprietății message a fost realizată prin apelul unei metode. Setarea unei proprietăți a unui obiect se poate face și prin atribuirea direct a unei valori către proprietatea referită (\$Obiect1->message="Setare directa a proprietatii") sau chiar printr-o inițializare în definiția clasei (vezi proprietatea \$data). Toate comenzile de afișare (echo) au fost utilizate doar pentru a evidenția modul de referire a atributelor/metodelor obiectului.

Observație: În cazul limbajul PHP nu se limitează accesul la proprietăți. Implicit toate proprietățile sunt de tip public, neputând fi declarate private sau protected.

Orice proprietate declarată în definiția clasei poate fi referită în exteriorul ei printr-o construcție de tipul: `$Obiect->nume_proprietate;`

O metodă specială a unei clase este așa numita metodă constructor. O metoda constructor are același nume cu al clasei, fiind apelata automat la crearea unui obiect (realizând operații de inițializare, crearea de alte obiecte necesare obiectului în cauza etc.). Un constructor se declară similar cu celelalte metode, singura deosebire fiind ca are același nume ca și clasa. În PHP definirea unei metode constructor nu este obligatorie. Clasa anterior creata nu avea definit un constructor (unele proprietăți fiind însă inițializare direct odată cu declararea lor).

Pentru exemplificare, se va crea un constructor pentru Clasa1, care va afișa un mesaj și va inițializa proprietatea \$message. Definiția anterioară a clasei se va completa cu încă o metodă (având același nume cu al clasei):

```
function Clasa1()  
{  
    echo "Obiect creat<br>";  
    $this->message = "initializare_obiect1";  
}
```

O simplă secvență de creare a obiectului va apela imediat metoda constructor:

```
$Obiect1 = new Clasa1();  
echo $Obiect1->message;
```

iar rezultatul va fi:

Obiect creat

initializare_obiect1

Pentru ca o clasă deja creată să poate fi instanțiată în mai multe scripturi PHP, fără a se face o replicare (copiere) a codului clasei în fiecare script, de regulă definițiile claselor sunt păstrate în fișiere distincte de cele în care sunt create obiectele. Pentru cazul de față, spre exemplu, într-un fișier clase.php este salvată definiția clasei, în timp ce într-un alt fișier PHP (obiect.php spre exemplu) se face instanțierea clasei (și utilizarea obiectului):

```
<?php  
// fișier obiect.php  
include("clase.php");  
$Obiect1 = new Clasa1();
```

```
echo $Obiect1->data;  
?>
```

Evident fișierul clase.php poate conține definițiile mai multor clase.

O caracteristică importantă a OOP o reprezintă moștenirea care permite crearea unei relații ierarhice între clase, folosind subclase. O subclasa, practic moștenește proprietățile și metodele unei superclase. Prin intermediul moștenirii, pe lângă elementele moștenite (proprietăți și metode), în cadrul noii clase se pot construi și adăuga noi elemente (proprietăți sau metode). Astfel, pornind de la clase de baza simple se pot deriva clase mai complexe și mai specializate pentru o anumită aplicație. Utilizarea moștenirii crește gradul de reutilizare și lizibilitate al codului sursă, avantaj important al OOP (reducându-se substanțial munca programatorului în cazul în care aceleași metode pot fi scrise doar o singură dată într-o superclasă, în loc să fie scrise de mai multe ori în subclase separate). Pe baza superclasei Clasa1 (în același fișier script), se construiește prin derivarea acesteia o nouă subclasa Clasa2 (utilizând cuvântul cheie extends), care moștenește toate proprietățile și metodele superclasei, dar în același timp are și alte noi proprietăți și metode:

```
class Clasa2 extends Clasa1  
{  
    var $message2="gama";  
    function getMessage()  
    {  
        echo "mesajul nou pentru obiectul 2 este<br>";  
        return $this->message2;  
    }  
    function plus()  
    {  
        echo "<br>ceva nou<br>";  
    }  
}
```

De remarcat că, în cadrul subclasei Clasa2 se definește o metodă având un nume similar cu al unei metode din superclasa (getMessage), realizându-se în cadrul subclasei o suprascriere a metodei moștenite, precum și o nouă metodă (plus). De asemenea, noua clasă are și o proprietate în plus (\$message1). O secvență de instanțiere și utilizare a obiectelor subclasei se poate face prin liniile de cod următoare:

```
$Obiect2 =new Clasa2();  
$Obiect2->plus();  
$Obiect2->setMessage("beta");  
echo $Obiect2->message."<br>";  
echo $Obiect2->getMessage();
```

are rezultatul următor:

Obiect creat

ceva nou
beta

mesajul nou pentru obiectul 2 este

gama

Crearea obiectului prin instanțierea subclasei Clasa2 conduce și la moștenirea constructorului corespunzător (afișându-se Obiect creat). Apelul metodei plus conduce la afișarea mesajului ceva nou. Apelul metodei moștenite setMessage permite setarea proprietății moștenite message. Apelul metodei suprascrise getMessage (plasat într-un echo), conduce la afișarea ultimelor două rânduri.

Observație:

Mecanismul de moștenire funcționează într-un singur sens, subclasa (copil) moștenește de la superclasa (părinte). Orice modificare a clasei părinte este automat preluată și de clasa copil.

O altă caracteristică importantă a OOP o reprezintă polimorfismul, prin intermediul căruia clase diferite pot conține metode cu același nume, dar cu comportamente diferite. Chiar în exemplul cu moștenire, superclasa respectiv subclasa conțin metode cu același nume (getMessage), dar funcționalități diferite.

Se consideră încă un exemplu. În același script cu definiția clasei Clasa1, fie definiția unei clase Clasa3:

```
class Clasa3
{
var $message;
function setMessage($param)
{
$this->message = $param;
echo "Clasa 3";
return $this->message;
}
}
```

La o instanțiere a claselor și folosire a obiectelor create:

```
$Obiect1=new Clasa1();
$Obiect3=new Clasa3();
$Obiect1->setMessage("Setarea proprietate folosind o metoda:");
echo $Obiect1->getMessage();
$Obiect3->setMessage("Setarea proprietate clasei 3");
```

rezultatele sunt:

Obiect creat

Mesajul pentru obiectul 1 este:

Setarea proprietate folosind o metoda:

Clasa 3

Observație:

În PHP 4 nu este permisă supraîncărcarea metodelor (supraîncărcare – overload- o aceeași metodă având în cadrul aceleași clase definiții multiple, fiecare cu un număr diferit de parametri de intrare). Începând cu PHP 5 este introdusă experimental o funcție `overload()` pentru a putea beneficia de acest avantaj.

Un exemplu concret de utilizare a programării orientate pe obiecte pentru realizarea conexiunii la un server MySQL, respectiv la baza de date, este prezentat în continuare, în contextul în care aceste operații implică o refolosire repetată a aceleiași secvențe de cod pentru fiecare script operând cu baza de date.

Ca punct de start, se definește o clasă (Clasa1) implementând o metodă (`setServer`) executând operațiile dorite (clasă salvată într-un fișier script distinct -Clase.php-, care poate fi referit și inclus în orice alt script este necesar utilizând comanda `include`):

- Fișier Clase.php:

```
<?php
class Clasa1
{
function setServer($var1, $var2, $var3, $var4)
{
$returnez = @mysql_connect ("$var1", "$var2", "$var3") or die ("Eroare SERVER");
mysql_select_db("$var4") or die ("Eroare BD");;
return $returnez;
}
}
?>
```

Metoda clasei are patru parametri de intrare (nume server, user, parolă, numele bazei de date), returnând în caz de reușită un identificator de conectare (util pentru cazul conectării la mai multe servere SQL, pentru identificarea unei anumite conexiuni).

Utilizarea clasei, în orice script este necesară o conexiune la MySQL, presupune o instanțiere a ei (creând un nou obiect), urmată de un apel al metodei ei, setată cu parametri adecvați:


```

<?php
include("clase.php");
$Obiect1=new Clasa1();
$db=$Obiect1->setServer("localhost","root", "parola","baza");
echo 'Conectare OK!';
$query = "SELECT * FROM test";
$interoghez=mysql_query($query);
...

```

O soluție și mai compactă ca și cod, presupune folosirea unui constructor și astfel, la o instanțiere a clasei se face și un apel automat al metodei realizând conectarea la server, respectiv la baza de date. Fișierul conținând clasa cu codul următor:

- Fișier Clase.php:

```

<?php //folosind constructor
class Clasa2
{
var $returnez;
function Clasa2($var1, $var2,$var3,$var4)
{
$this->returnez = mysql_connect("$var1", "$var2", "$var3") or die ("Error connecting to mysql");
mysql_select_db("$var4");
return $this->returnez;
}
}
?>

```

Scriptul în care se face un apel la această clasă pentru realizarea unei conexiuni la o bază de date MySQL poate conține următorul cod:

```

<?php
include("clase.php");
$Obiect2=new Clasa2("localhost","root","parola", "baza");
$db=$Obiect2->returnez; // referire identificator conectare
//cod interogare
...
?>

```

Informații sistem

Informații asupra datei și timpului curent. Sunt prezentate câteva funcții utile pentru obținerea datei și timpului curent al sistemului (time(), strftime(), getdate()). Este importantă uneori obținerea exactă a datei (inclusiv a momentului de timp) aferente execuției unei anumite operații, astfel încât doar timpul de pe serverul Web (același pentru toți clienții apelanți) poate fi luat în considerare. Codul sursă al unui exemplu de utilizare este următorul:

```
<?php

//data curentă ca string

echo "Timpul ca string: ".time()."<hr>";

// funcția strftime( ) convertește timpul curent într-o dată formatată

$data=strftime("%d.%m.%Y",time());

echo "Data ca string an, luna, zi: ".$data."<br>";

// tot funcția strftime( ) convertește timpul curent într-un timp formatat

$time=strftime("%H%M",time());

$time1=$time/100;

echo "Timpul (ora, minut): ".$time1."\n"."<hr>";

//extragere zi, luna, an din time() ca elemente ale unui șir ( data ca array)

$data1=getdate(time());

// construcție data si timp afișat formatat

$var="Data: ".$data1["mday"].".".$data1["mon"]."."

.$data1["year"]." "."Timp: ".$time1.".";

print ($var."<br>"); // afișare

// strtoupper - conversie la caractere mari

echo "<strong>".strtoupper($var); // afișare

?>
```

Figura de mai jos este elocventă asupra efectului fiecărei funcții utilizate.

```
Timpul ca string: 1342418725
```

```
Data ca string an, luna, zi: 16.07.2012
```

```
Timpul (ora, minut): 9.05
```

```
Data: 16.7.2012 Timp: 9.05.
```

```
DATA: 16.7.2012 TIMP: 9.05.
```

Funcția `time()` returnează timpul în secunde (ca string), scurs de la începutul —Epocii Unix|| (1 Ian. 1970). Funcția este utilă, prin utilizarea împreună cu alte funcții, pentru determinarea datei sau timpului curent (folosit de regulă ca referință). Funcția `strftime()` formatează timpul/data, conform unui șablon dat ca argument, returnând un string, pe baza informației furnizate de `time()`.

Funcția `getdate()` returnează un șir cu data calendaristică, permițând referirea și afișare element cu element (zi, lună, an). Funcțiile `strtoupper()`, `strtolower()` realizează o conversie a unui șir de caractere (furnizat ca argument) la caractere mari, respectiv mici.

Informații privind accesul client

În exemplul din paragraful anterior s-a realizat o contorizare a numărului de accese la o pagină Web. O altă informație utilă, referitoare la accesarea unei pagini Web, rezidă din întrebarea —Cine a accesat pagina Web?|| Acest —cine|| înseamnă mai precis —Care este adresa IP a clientului care s-a conectat la pagina Web curentă?||

Pentru a răspunde la această întrebare și nu numai, se consideră scriptul prezentat în continuare:

```
<?php
//obținere informații de mediu
$I=getenv(—REMOTE_ADDR||);
$j=getenv(—SERVER_NAME||);
//afișare informații de mediu
printf(— Your IP number is : %s\n||,$I);
printf(— The server name is : %s\n||,$j);
?>
```

Funcția `getenv` cu sintaxa:

string getenv (string nume_variabilă)

are ca parametru de intrare o variabilă predefinită de mediu. În cazul de față, cele două variabile au următoarea semnificație:

REMOTE_ADDR – adresa IP a clientului

SERVER_NAME – numele serverului

Evident, utilizarea funcției getenv () având ca parametru o variabilă predefinită de mediu, are ca rezultat returnarea unui string conținând însăși informația asociată semnificației acelei variabile.

Afișarea rezultatelor pentru exemplul considerat s-a realizat folosind o funcție de afișare formatată: printf().

În cazul de față %s înseamnă că, conținutul variabilei de afișat va fi tratat și formatat ca un string (\n înseamnă deja cunoscutul _linie nouă'). Alte argumente de formatare posibile pentru printf(): %d –întreg cu semn în reprezentare zecimală, %f

– număr în virgulă flotantă, %b – întreg în reprezentare binară etc.

Generare e-mail

Realizarea operației de trimitere a unui e-mail utilizând un script PHP implică în primul rând existența unei conexiuni la un server de mail SMTP, urmată de o configurare corespunzătoare a câtorva opțiuni din fișierul php.ini.

Presupunând că, numele serverului SMTP este aut.upt.ro, iar adresa de email a emitentului este adm@domeniu1.ro, configurarea fișierului php.ini implică următoarele (pentru o platforma Win32):

[mail function]

; For Win32 only.

SMTP= aut.upt.ro ; for Win32 only

; For Win32 only.

Sendmail_from= adm@domeniu1.ro; for Win32 only

În acest moment, totul se rezumă la utilizarea funcției mail(), așa cum se poate observa și în scriptul următor:

<?php

\$a=mail("dan@aut.upt.ro", "Acesta e un subject!!!", "Asta este continutul");

echo \$a;

// 1- pt. Reușită, 0 – pt. Nereușită

?>

În exemplul prezentat, s-a utilizat o formă sintactică redusă a funcției mail(), având doar 3 argumente de intrare: adresa destinatarului, subjectul mailului, conținutul propriu-zis. Funcția returnează `_1` în caz de reușită, respectiv `_0` în caz de nereușită.

Servicii Web cu PHP

Un serviciu Web este o componentă Web oferind o serie de metode care pot fi invocate (apelate, consumate) prin intermediul Web-ului (deci remote, la distanță), folosind standarde deschise de comunicare. Practic, un apel de serviciu Web asigură o funcționalitate similară cu cea a unei RPC - Remote Procedure Call (apel de procedură la distanță), diferențele fiind la nivelul tipului de apel, serviciul fiind invocat printr-un apel HTTP (utilizând protocolul SOAP). Mai mult, apelul serviciului poate fi făcut atât dintr-o aplicație WEB, cât și dintr-o aplicație desktop.

Avantajul major al utilizării serviciilor Web constă în faptul că, anumite funcționalități dorite și deja existente ca implementare, pot fi integrate direct în cadrul unei aplicații, ne mai trebuind a fi codate, fiind rapid accesibile prin simple apeluri la componente deja existente. Un serviciu Web dispune de câteva proprietăți intrinseci, putându-se enumera următoarele:

- este o componentă software de sine stătătoare (auto-încapsulată și autodescriptivă),

apelabilă (consumabilă) dintr-o altă aplicație (Web sau Windows desktop);

- are ca element de baza pentru transferul informației formatul XML;

- comunica utilizând protocoale deschise SOAP (Simple Object Access Protocol - protocol standard pentru transmiterea de mesaje între diferite module, definind regulile de procesare a unui mesaj, convenții de apelare a procedurilor la distanță (Remote Procedure Call - RPC) și de reprezentare a răspunsurilor la aceste apeluri;

- vizibilitate asigurată de sistemul UDDI (Universal Description, Discovery

and Integration), ca sistem de bază pentru localizarea serviciilor Web, acesta făcându-le publică existența pentru a putea fi consumate;

- autodescriere asigurată de metoda WSDL (Web Services Description Language) care asigură o descriere a interfeței pentru utilizarea serviciului, creând documentele care descriu metodele serviciului, parametri și argumentele acceptate ale acestora, date returnate etc.)

Pot fi de asemenea punctate câteva proprietăți funcționale:

- permit crearea de aplicații independente de platformă (sisteme de operare, instrumente, limbaje de programare folosite);
- implementează componente care pot comunica între ele (indiferent de platformă);
- un serviciu Web furnizează informații pentru descoperirea și descrierea lui (și a metodelor lui, parametrilor de apelare, date returnate etc.).

Limbajul PHP (ca și alte limbaje de programare Web) permite operarea cu serviciile Web, oferind în acest sens biblioteca extensie `php_soap.dll` (necesitând activarea ei prin configurarea corespunzătoare în fișierul `php.ini`). Problematika serviciilor Web este tratată la ora actuală într-o mulțime de cărți și site-uri dedicate unele chiar exclusiv acestui subiect. În paragraful de față nu se va intra în detalii privind modul de creare a serviciilor Web utilizând PHP-ul, fiind oferită o simplă exemplificare privind consumarea unui serviciu Web. Pentru cazul tratat s-a considerat apelul unui serviciu Web oferind cursul valutar curent, disponibil pe Internet la adresa:

<http://www.infovalutar.ro/curs.asmx?wsdl>

Codul unui posibil script PHP care realizează consumarea acestui serviciu este prezentat în continuare, cu comentariile de rigoare:

```
<?php

// Configurare php.ini -> php_soap.dll

// Instanțiere obiect SOAP

$client = new SoapClient("http://www.infovalutar.ro/curs.asmx?WSDL");

// Apel metodă cu data ultimei actualizări (fără parametri)

$result = $client->lastdateinserted();

// afișare dată - se apelează singura proprietate pe care o are obiectul returnat

echo "Data ultimei actualizari: ".

$result->LastDateInsertedResult."<br><b>";

// Se apelează metoda (cu un parametru de intrare)

$result = $client->GetLatestValue

(array('Moneda'=>'EUR'));

// afișare curs - proprietatea pe care o are obiectul returnat
```

```

echo 'Euro: ' . $result->GetLatestValueResult.PHP_EOL;

// apel metodă cu un parametru

$result = $client->GetLatestValue
(array('Moneda'=>'USD'));

// afișare curs valutar apelând proprietatea pe care o are obiectul returnat

echo 'USD: ' . $result->GetLatestValueResult;

?>

```

Rezumând, etapele necesare consumării unui serviciu Web folosind biblioteca extensie php_soap.dll sunt următoarele:

- Instanțiere obiect SOAP:

```
$client = new SoapClient;
```

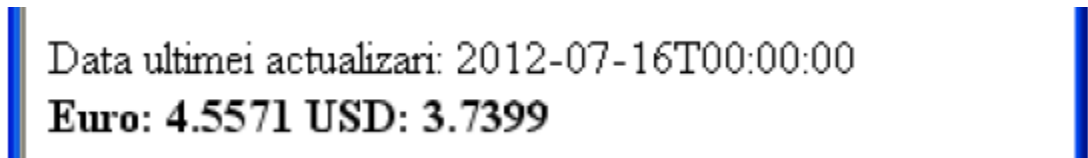
- Apel metoda (cu parametri):

```
$result = $client->metoda(array('param1'='val1', 'param2'='val2'....));
```

- Apel proprietate pentru afișare:

```
echo $result->Proprietate;
```

Rezultatul rulării scriptului anterior este prezentat în figura:



The screenshot shows the output of a PHP script. It displays the date of the last update as '2012-07-16T00:00:00'. Below this, it shows the exchange rates: 'Euro: 4.5571' and 'USD: 3.7399'. The text is displayed in a monospaced font, with the currency names in bold.

Observație: PHP permite dezvoltarea de aplicații integrând servicii Web bazate și pe altor pachete extensie (spre exemplu, nuSOAP ca suport conținând un grup de clase PHP destinate creării și operării cu servicii Web).