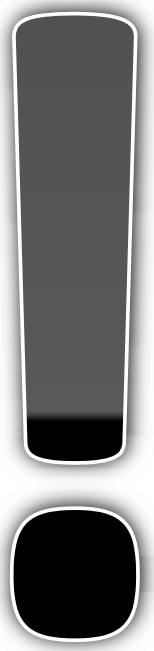
Wer nutzt die PHP-Reflection API?
Und für was?



Fragen können jederzeit gestellt werden

Metaprogrammierung und Reflection

Tool-Entwicklung für und mit PHP

Stefan Marr
PHP Unconference
26./27. April 2008

HPI Hasso Plattner Institut

Agenda

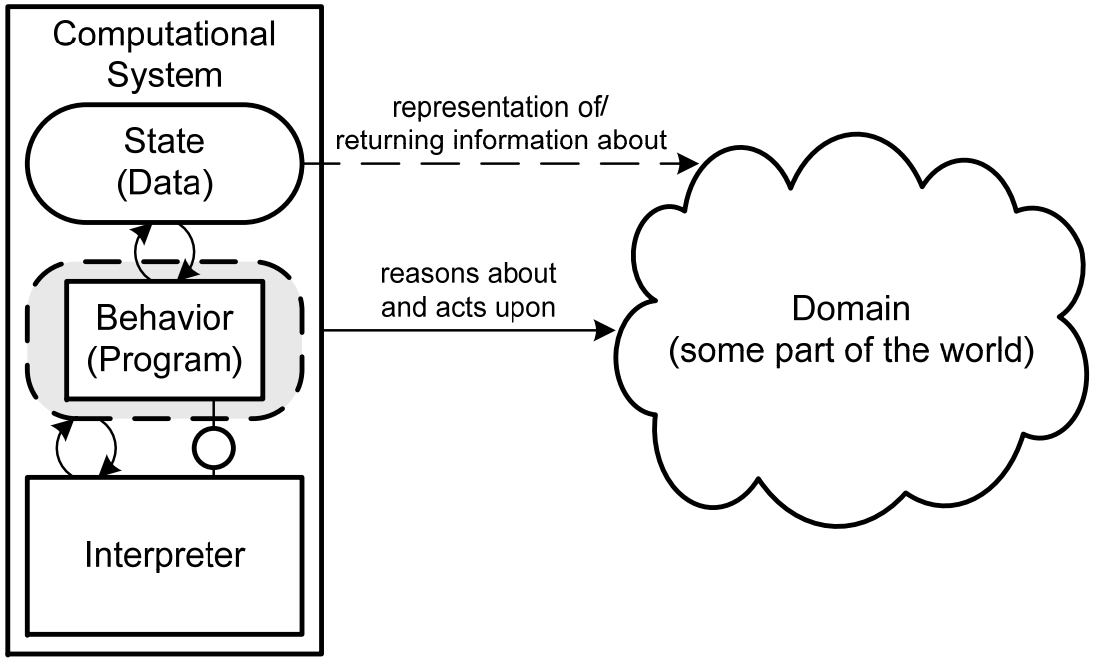
1. Begriffsklärung
2. Open Implementations
   - Metaprogrammierung und Frameworkdesign
3. Relevante Sprachfeatures von PHP
4. Reflection und Runkit für die Tool-Entwicklung
5. Zusammenfassung

Kurze Begriffsklärung zur Einleitung

METAPROGRAMMIERUNG UND REFLECTION

Verarbeitendes System

nach [1]

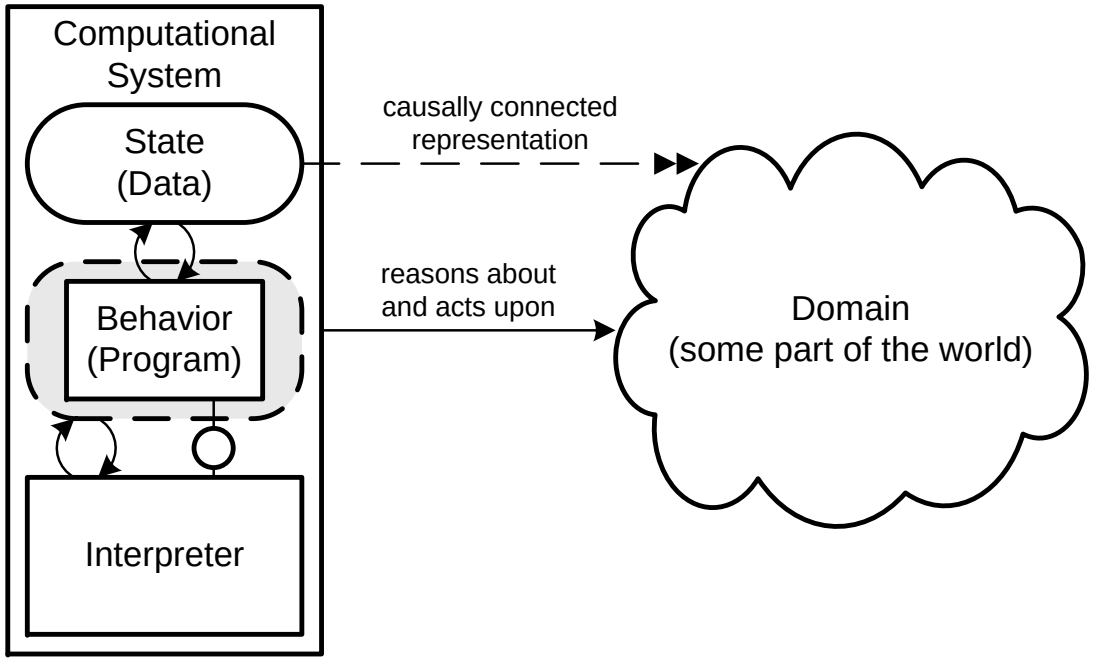


- Daten repräsentieren Domäne
- Programm beschreibt Veränderungen
- Interpreter führt Programm aus

- Domäne
  - Buchhaltung, Kundendaten, Logistik, Webinhalte, …

Kausal verbundenes System

nach [1]

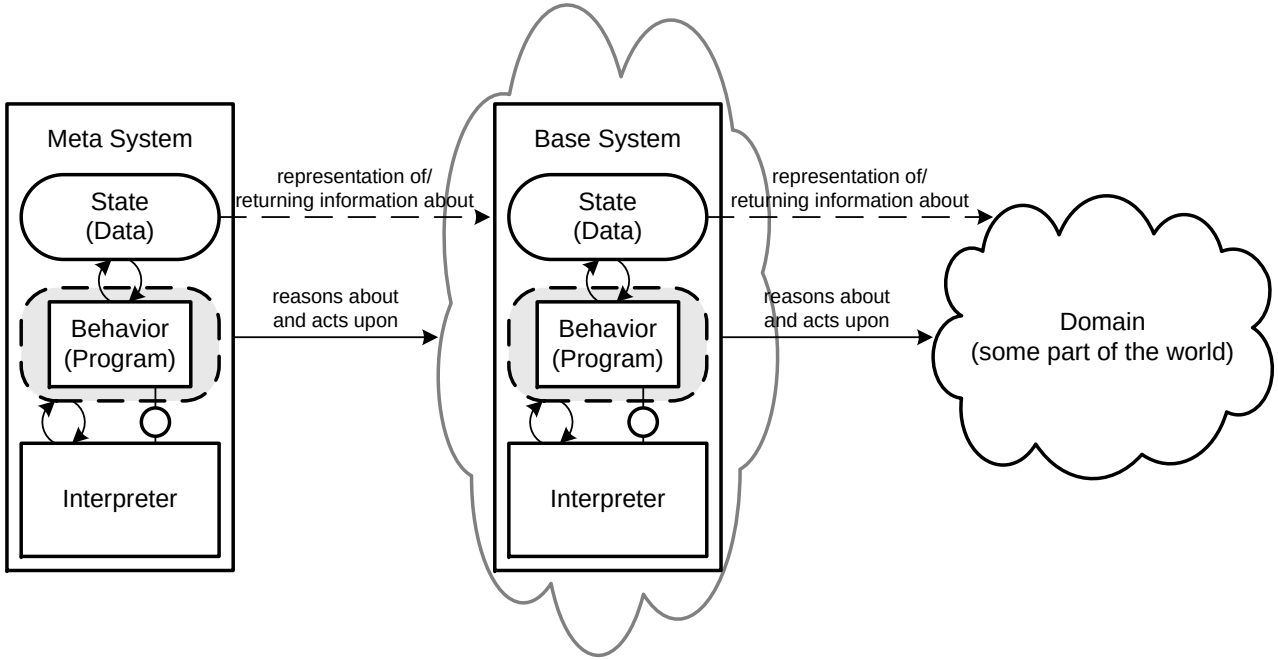


- Ursache-Wirkungsbeziehung zwischen System und Domäne
- Direkte wechselseitige Auswirkungen

- Regellungstechnik
  - Klimaanlagen
  - Autopiloten
  - …

Meta-System

nach [1]

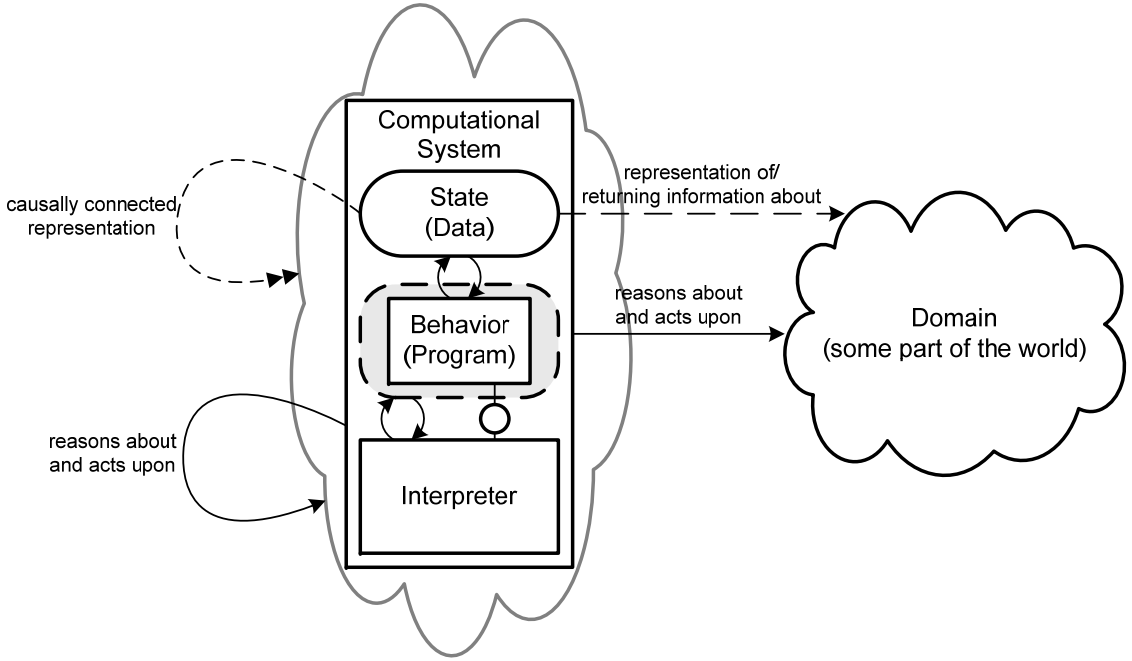


- Spezielle Domäne: Programme
- Meta-Programme und -system arbeiten auf Basis-Programmen und -systemen

- Compiler
- Code-Transformation, Generierung
- …

Reflektives System

nach [1]

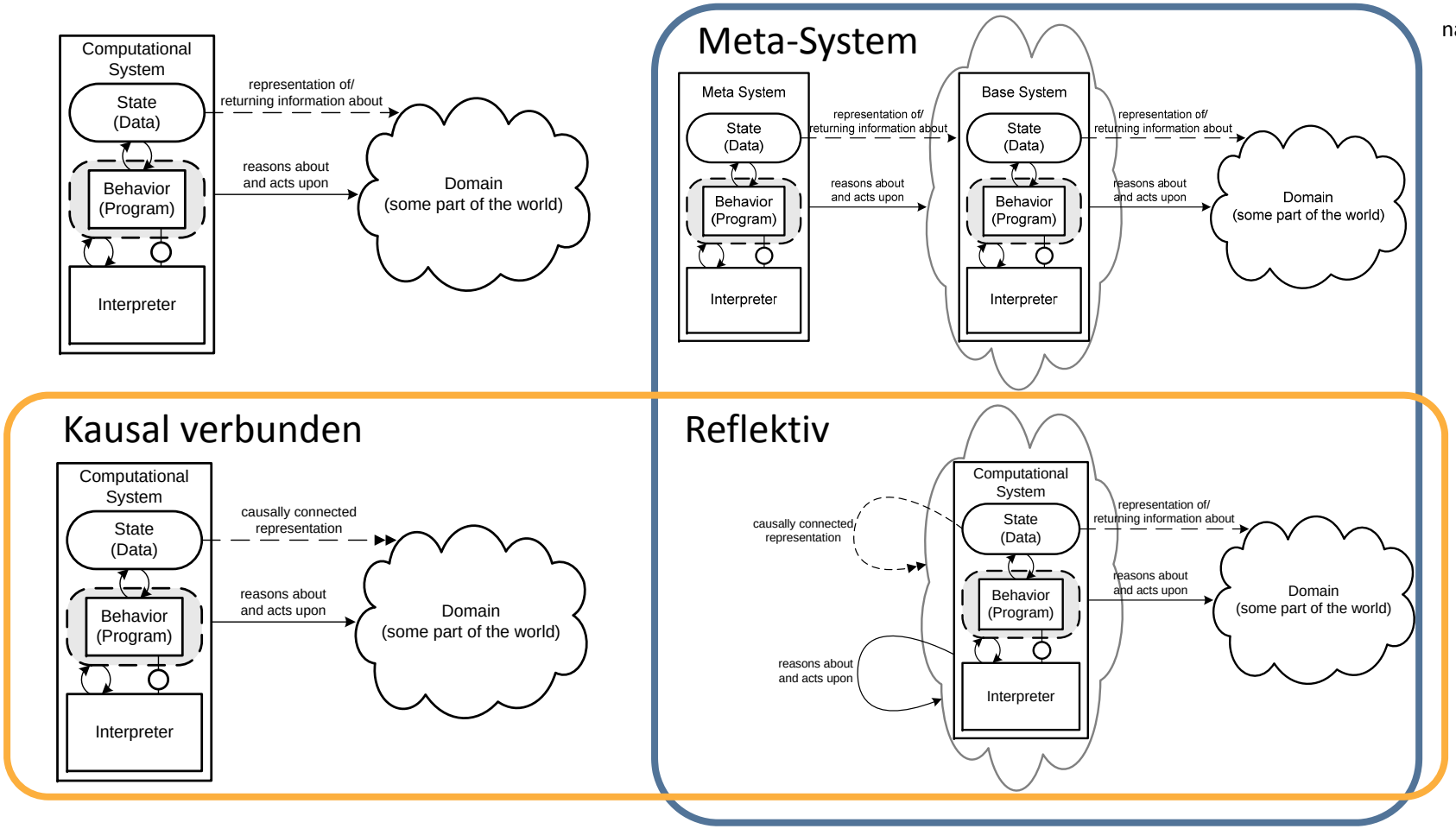


- Spezielle Meta-Systeme
- Selbstrepräsentation bestimmter Strukturen
- Kausal verbunden

- Reflektives Verhalten
  - Introspection (beobachten)
  - Intercession (Selbstveränderung)

Meta- und Reflektive Systeme

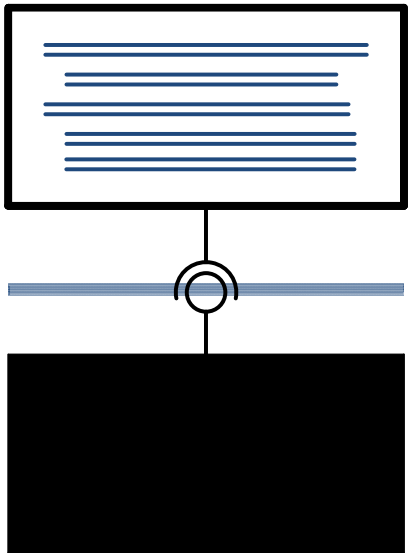
nach [1]



Metaprogrammierung und Moduldesign

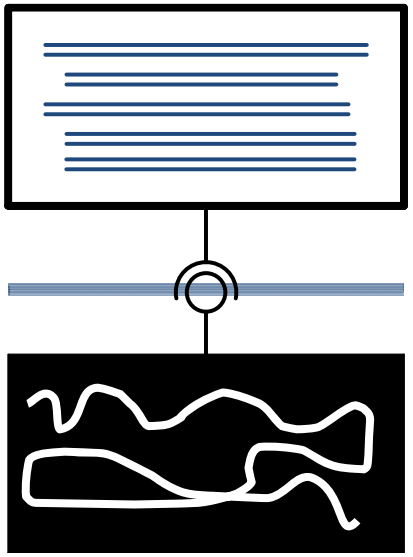
OPEN IMPLEMENTATIONS

Abstraktion in einer Black Box



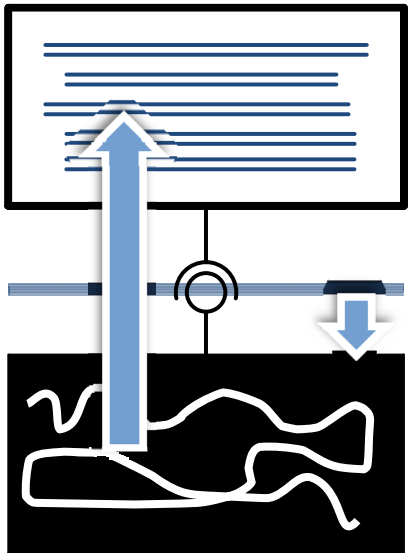
- Verstecken von Implementierungsdetails
  - Vereinfachung des Client-Codes
- Verringerung von Abhängigkeiten zwischen Modulen und Clients
  - Erhöhung der Wiederverwendbarkeit
  - Client-Code lesbarer und verständlich ohne Wissen über unterliegende Implementierung

Probleme der Black Box



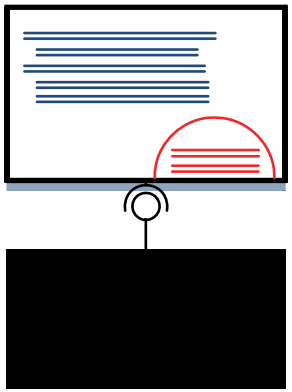
- Geeignete Implementierungsstrategie abhängig von der Art der Nutzung des Moduls
- Implementierungs-entscheidungen
  - selten dokumentiert
  - für bestimmte Fälle ungeeignet
  - nicht anpassbar
  - zu starke Abstraktion
- Client weiß meist am besten welche Strategie geeignet wäre

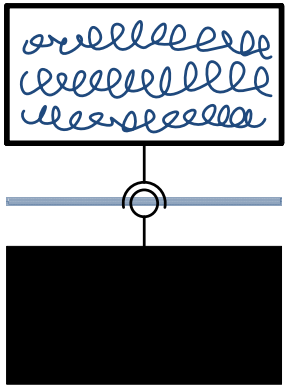
Keine 100%ige Black Box möglich



- Implementierungs-entscheidungen nicht im Interface sichtbar, aber:
  - Performanceauswirkungen
- Abbildung von High-Level auf Low-Level Funktionen vielfältig möglich
  - Beeinflussen Performance
  - Entscheidung daher feststehend, aber nicht verborgen

Probleme und Workarounds



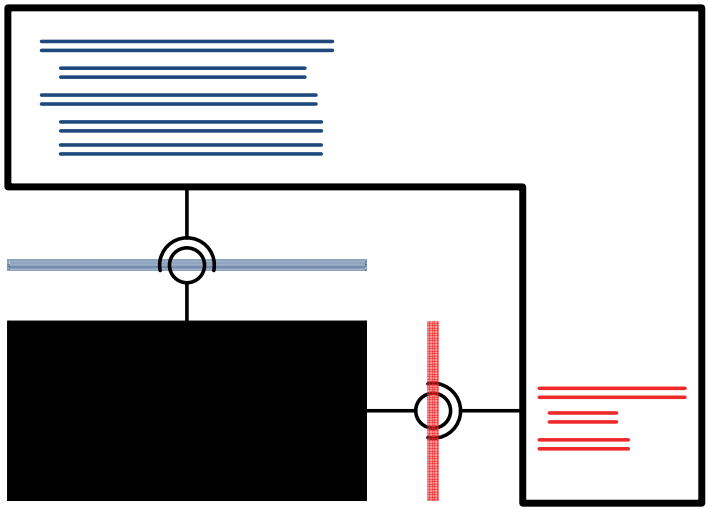


- Neuimplementierung von Interfacefunktionen
  - Um gewünschte Eigenschaften zu erreichen
  - Code wird größer und komplexer
- „Code zwischen den Zeilen“
  - Behandelt Probleme die eigentlich vom Interface verborgen werden sollten
  - Basiert auf undokumentierten Interna, verborgen vom Interface, aber dem Programmierer bekannt

Lösung: Reflektive Module

Basis Interface

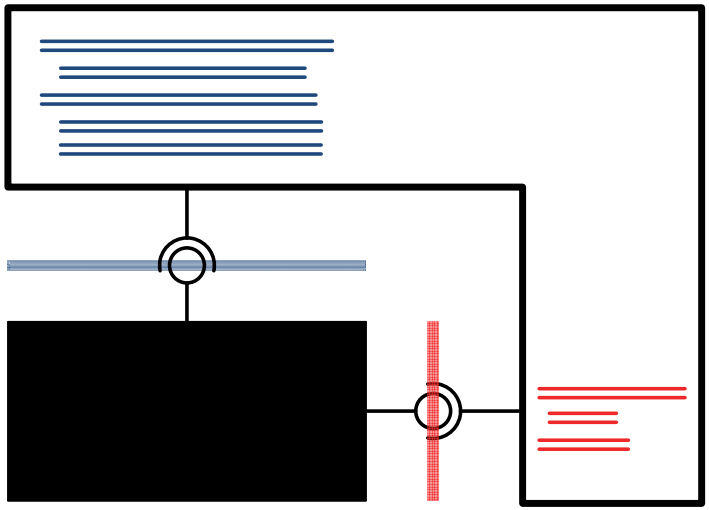
- Bietet Funktionalität an
- Das WAS



Meta Interface

- Ermöglicht Steuerung des Basis Interfaces
- Das WIE

Reflektive Module/Programme



- Vorteile
  - Schnittstellennutzung jetzt wieder als Black Box
  - Konfiguration bzw. Implementierungsstrategie der Black Box vom Client wählbar
- Programmierer können sich auf Modulnutzung konzentrieren
- Optimierung/Konfiguration separat möglich

Umsetzungsmöglichkeiten

- Frameworks
  - Aufteilung in Client API und Framework API
  - Konfigurationsdateien, Konstanten, Parameter, Flags, …
- Dependency Injection als Pattern verbreitet
- Funktionen mit Callbacks altbekannt

Zusammenfassung Open Implementations

- Frameworks sollten Erweiterungs- bzw. Spezialisierungspunkte anbieten
  - Problematische Teile vom Client anpassbar
- Anpassung, Konfiguration über Meta-APIs
- Design trotzdem nicht mit den zu kapselnden Details überfrachten
  - Auf geeignete Trennung zwischen WAS und WIE achten

Magic Methods und die Reflection API

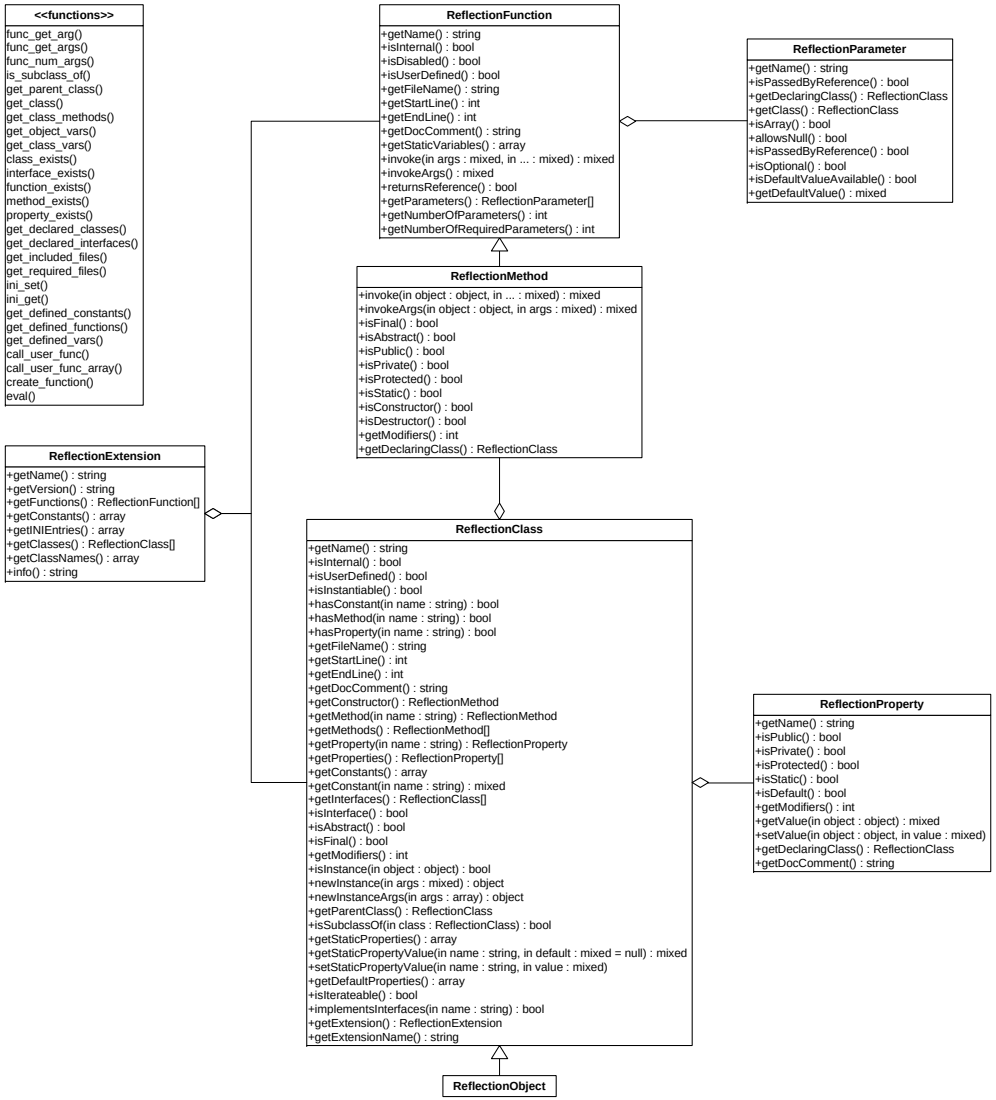
RELEVANTE SPRACHFEATURES VON PHP

Magic Methods

- Dynamische Object-Properties
  - `__get()`, `__set()`
    - Werden verwendet wenn die angefragten Properties nicht vorhanden sind
    - z. B. für Validierung und Vermeidung von unspezifizierten Properties
    - `__isset()`, `__unset()`
- `__clone()`: ändert Semantik von `clone`
- `__toString()`, `__sleep()`, `__wakeup()`, `__set_state()`

Magic Methods

- `__autoload($className)`
  - Ermöglicht echtes Lazy-loading von Klassen
  - Implementiert eigene Laderoutine
- `__call($methodName, $args)`
  - Ermöglicht echtes Late-Binding
  - Instanzverhalten zur Laufzeit entscheidbar
- `__callStatic($methodName, $args)`
  - Late-Binding für Klassenmethoden

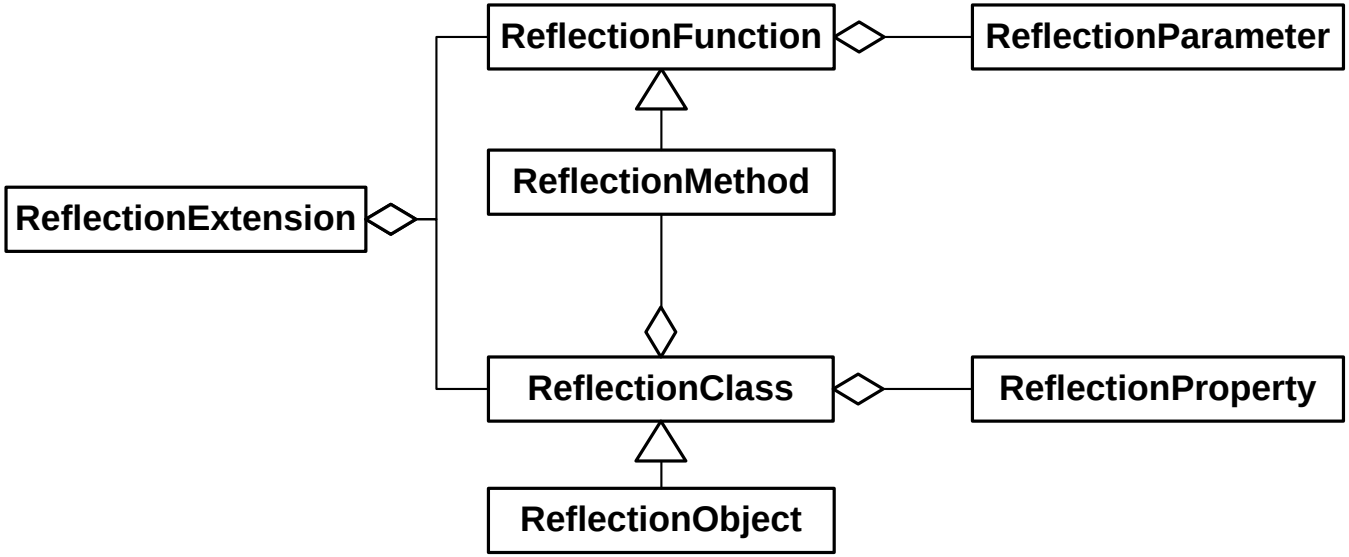


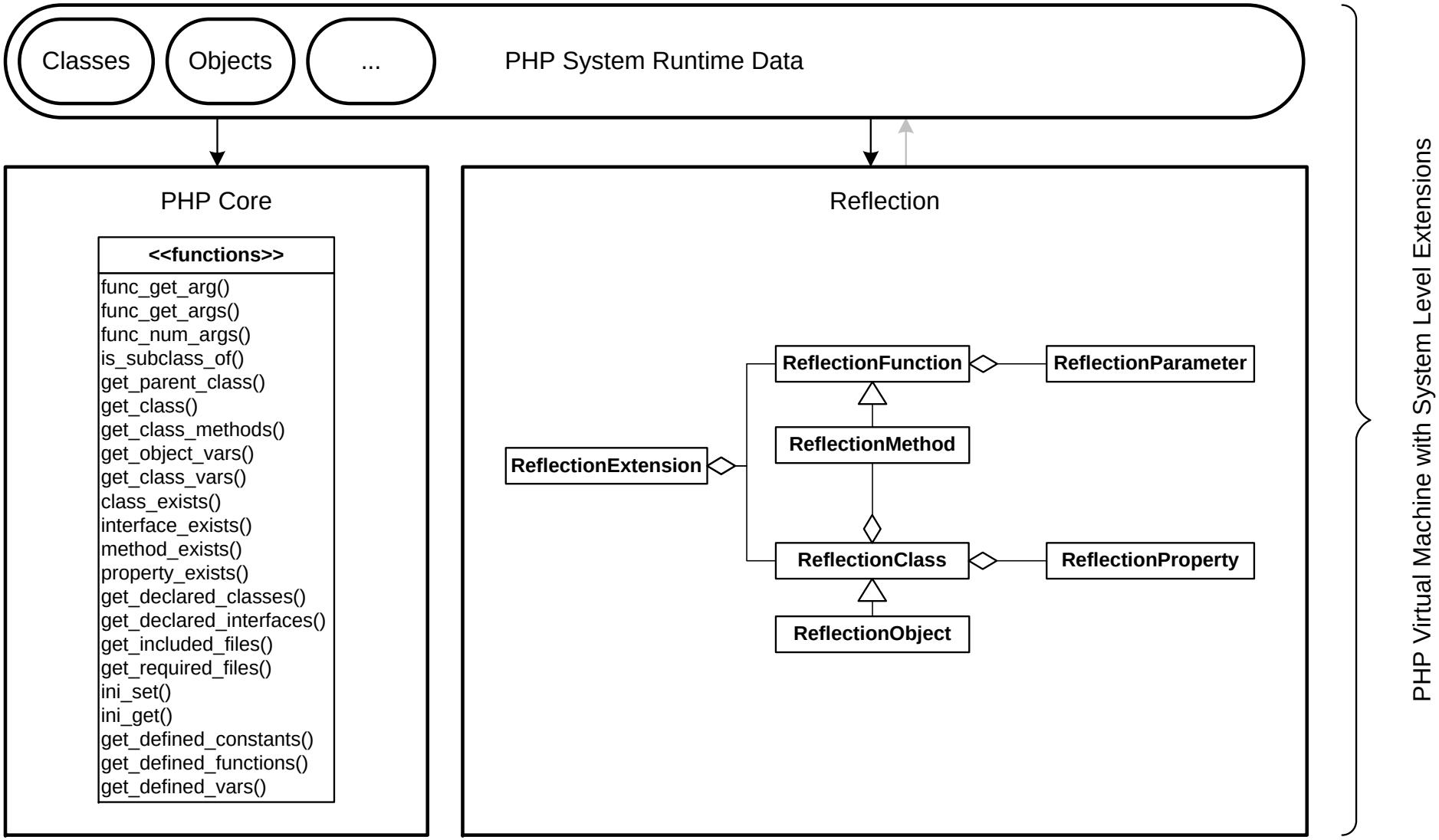
Reflection API

Klassendiagramm der Reflection API von PHP 5.2

<<functions>>

func_get_arg()
func_get_args()
func_num_args()
is_subclass_of()
get_parent_class()
get_class()
get_class_methods()
get_object_vars()
get_class_vars()
class_exists()
interface_exists()
function_exists()
method_exists()
property_exists()
get_declared_classes()
get_declared_interfaces()
get_included_files()
get_required_files()
ini_set()
ini_get()
get_defined_constants()
get_defined_functions()
get_defined_vars()
call_user_func()
call_user_func_array()
create_function()
eval()




Classes
Objects
...
PHP System Runtime Data
PHP Core
<<functions>>
func_get_arg()
func_get_args()
func_num_args()
is_subclass_of()
get_parent_class()
get_class()
get_class_methods()
get_object_vars()
get_class_vars()
class_exists()
interface_exists()
method_exists()
property_exists()
get_declared_classes()
get_declared_interfaces()
get_included_files()
get_required_files()
ini_set()
ini_get()
get_defined_constants()
get_defined_functions()
get_defined_vars()
Reflection
ReflectionFunction
ReflectionParameter
ReflectionMethod
ReflectionExtension
ReflectionClass
ReflectionProperty
ReflectionObject
PHP Virtual Machine with System Level Extensions

phpCrow, ezcReflection und isvcRunkit

REFLECTION UND RUNKIT FÜR DIE TOOL-ENTWICKLUNG MIT PHP

phpCrow

Classes and Methods | S01_debug_backtrace | S03_invoke_method | S02_create_function

Class	Method
CodeOutline	__construct
Crow	draw
ezcBase	drawCore
ezcReflectionApi	drawInternal
ezcReflectionClass	drawMethodRect
ezcReflectionMethod	drawOnCanvas
ezcReflectionTypeMapper	drawOutline
iscReflectionClass	drawPixel

User Internal | Only Functions | execute code | Set Code

```
1        $this->method = $method;
2        $this->gtkImage = $gtkImage;
3
4        if ($this->filename == null) {
5            $this->drawInternal();
6            return;
7        }
8
9
10       $content = file_get_contents($this->filename);
11
12       $this->tokens = token_get_all($content);
13       $this->content = split("\n", $content);
14
15       $maxLength = 0;
16       foreach ($this->content as $line) {
```

Intercession mit Standard-PHP

- Generierten Code ausführen: `eval()`
- Funktionen erstellen: `create_function()`
- Dynamische Aufrufe:
  - `call_user_function()`
  - `call_user_function_array()`
  - `call_user_method()`
  - `call_user_method_array()`
  - `$functionName($a, $b);`

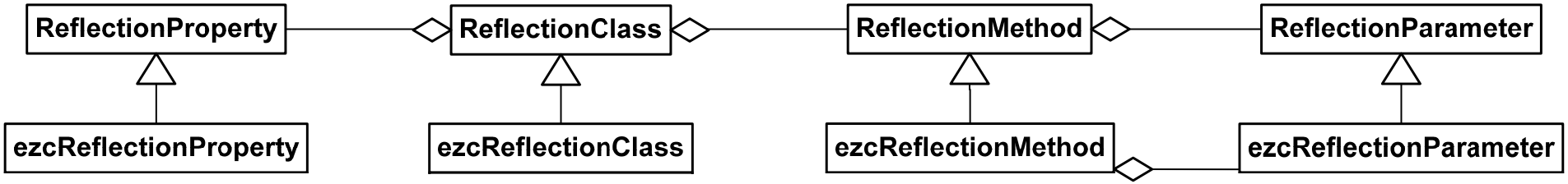
create_function

```
$func = create_function('$foo,$bar',
            'echo "CreatedFunction: $foo $bar\n";');
$func('Hello', 'World! ');
var_dump(addslashes($func));
call_user_func($func, 'FOO', 'baz');

//common usage example, with performance issues!
$av = array('the ', 'a ', 'that ', 'this ');
array_walk($av,
  create_function('&$v,$k', '$v = $v . "mango";'));

print_r($av);
```

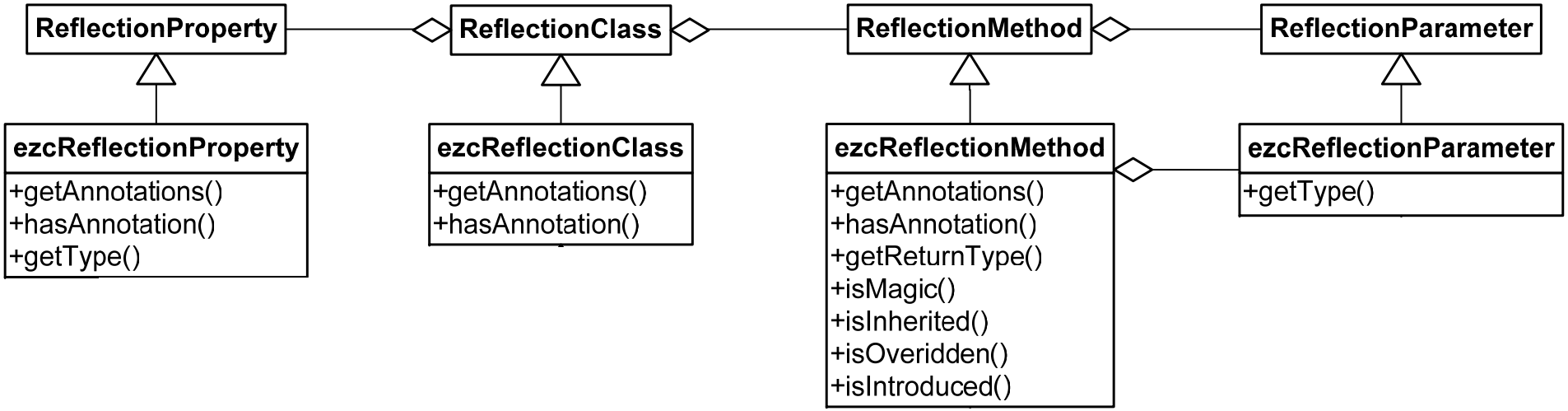
Funktionen/Methoden Aufrufe

```
class InvokeTest {
   public function callMe() {
        echo "Yeah, you called me!\n";
   }
}

$o = new InvokeTest();
$class = new ReflectionClass($o);
$method = $class->getMethod('callMe');
$method->invoke($o);

function fooFunc($bar) {
   echo "$bar\n";
}
$func = 'fooFunc';
$func('Hello World');
```

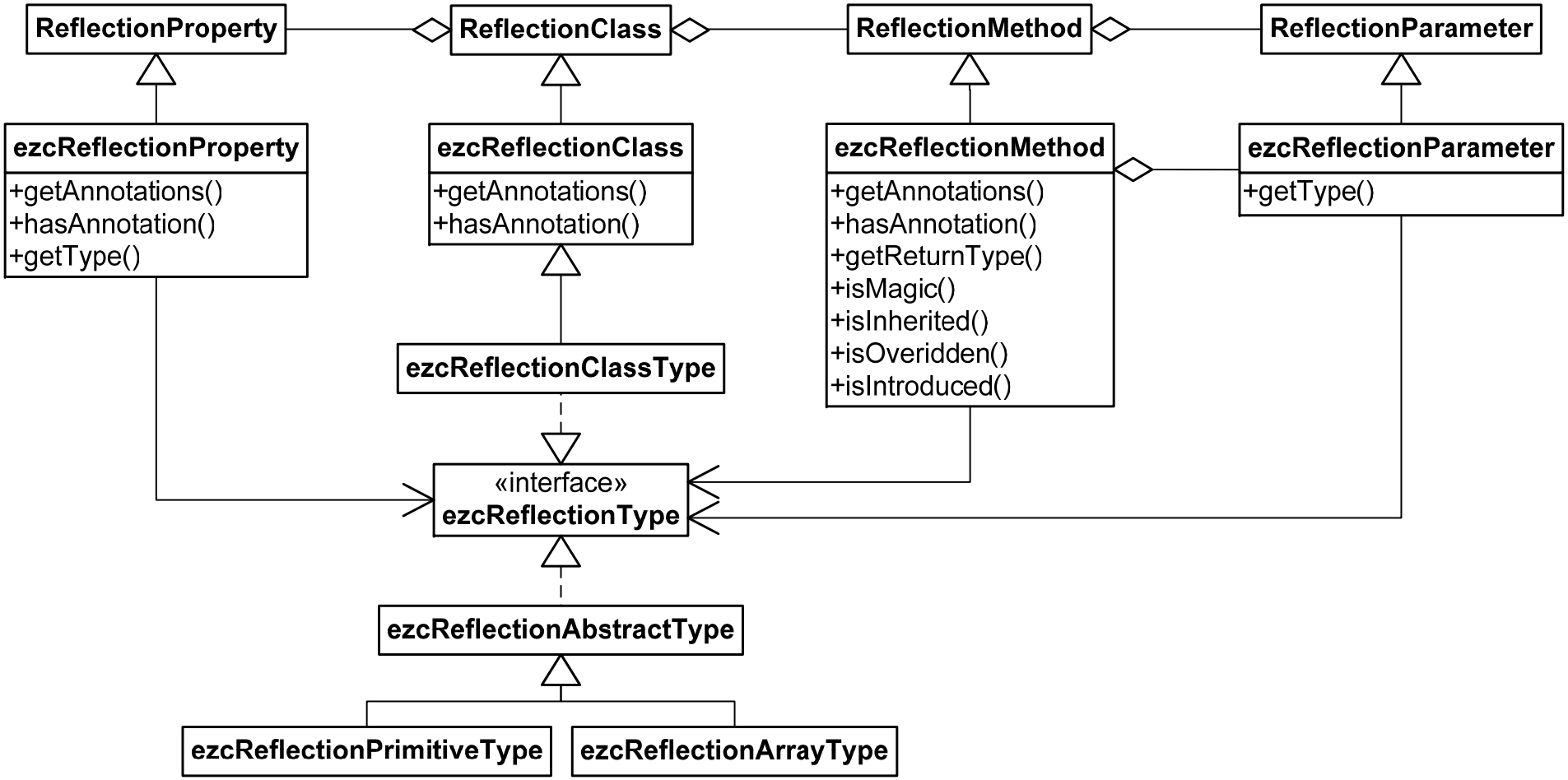
ezcReflection



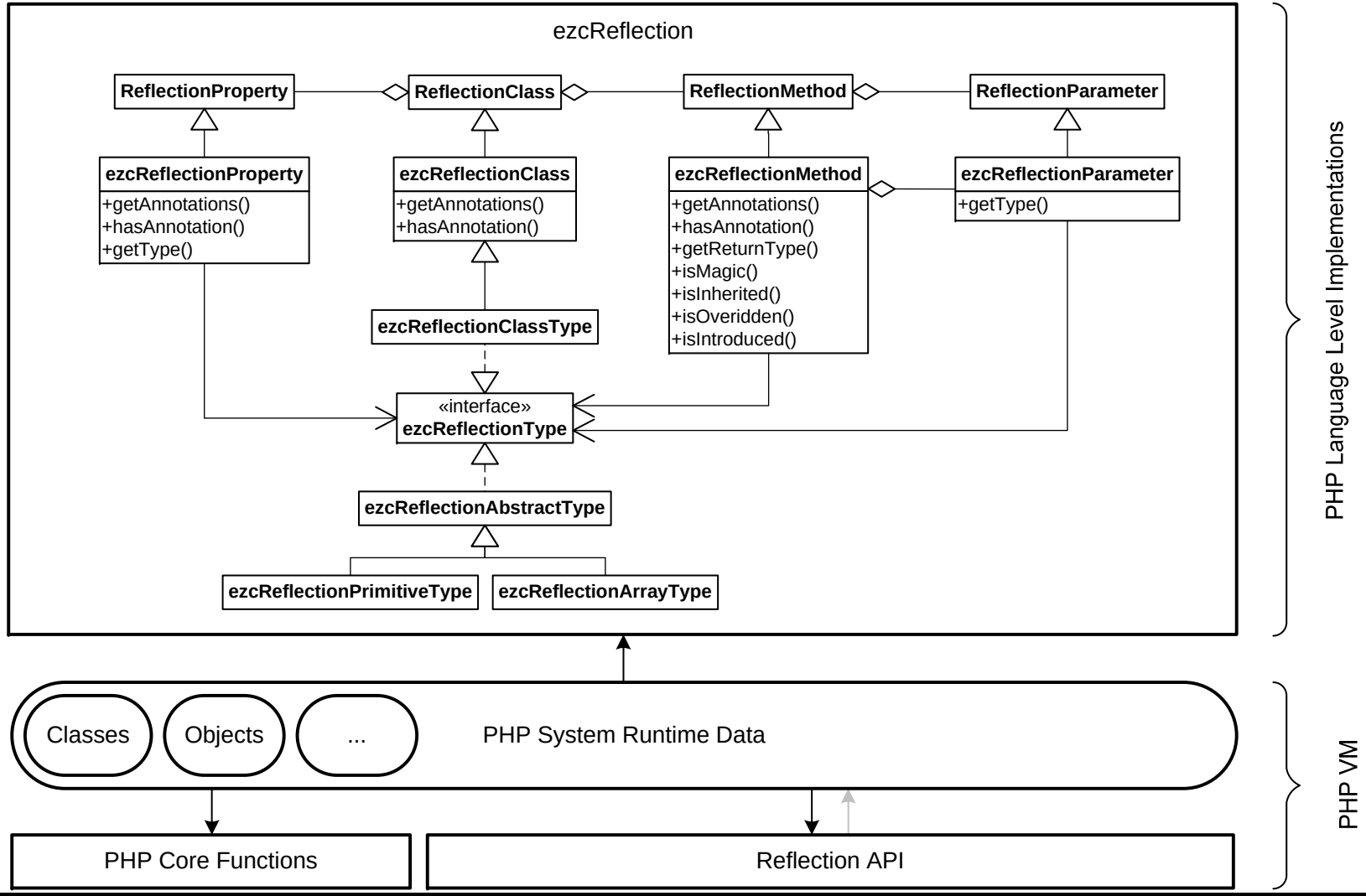
ezcReflection



ezcReflection



ezcReflection



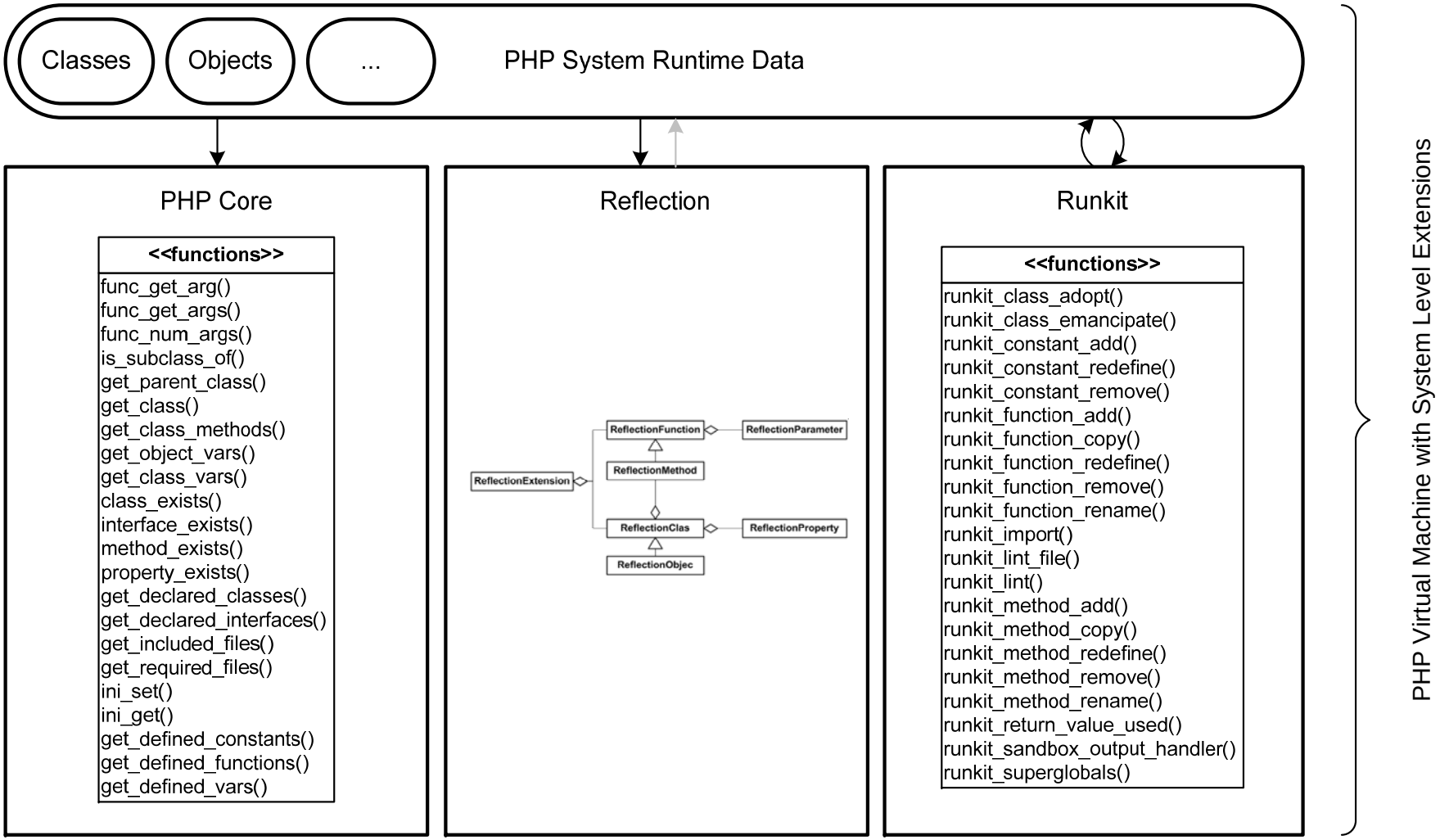
ezcReflection Überblick

- Ermöglicht Reflection auf PHPDoc-Annotationen
- Erweiterte Typ-Informationen zur Laufzeit
- Aber keine direkten Laufzeit-Einflüsse der Typ-Annotationen
- Design ermöglicht beliebige Erweiterungen der Reflection API als Datenquelle mit zu nutzen
  - z. B. eine StaticReflection
- Genutzt für z. B. WSDL Generierung

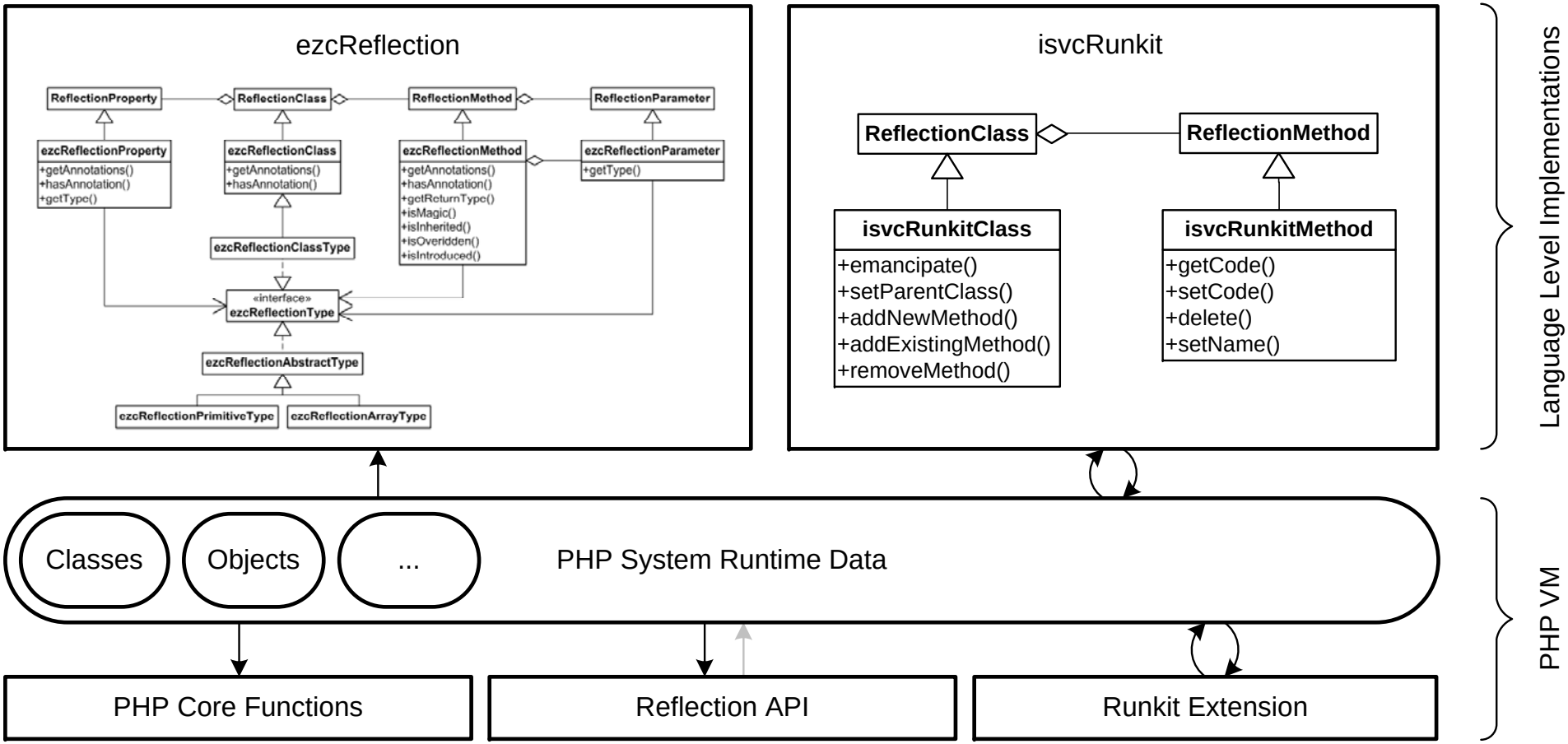
Annotationen

```
/**
* @webservice
*/
class HelloWorld {
    /**
    * @var string
    */
    private $hello = 'Hello';
    /**
    * @param string $name
    * @return string
    * @webmethod
    */
    public function getGreeting($name) {
      return "$this->hello $name!";
    }
}
```

Intercession mit Runkit



ezcReflection und isvcRunkit



Metaprogramming und Reflection

ZUSAMMENFASSUNG

Zusammenfassung

- PHP hat viele Features um Reflection einzusetzen
- Intercession über Runkit-Extension
  - Vorallem für Tool-Entwicklung
- Reflection hat Einflüsse auf Performance
  - PHP Ausführungsmodell beachten
  - Besonders in Web-Anwendungen nicht erste Wahl
- Nützlich für länger lebende Prozesse/Programme
  - z. B. GTK+, QT, MFC oder Kommandozeilen-Anwendungen

Literatur

Paper zu diesem Vortrag

G. GABRYSIAK, S. MARR, AND F. MENGE, *Meta Programming and Reflection in PHP*, Hasso-Plattner Institute, Universiy of Potsdam, Germany, February 2008.
https://instantsvc.svn.sourceforge.net/svnroot/instantsvc/branches/metaprogramming/doc/

Der Code

phpCrow – A PHP-PHP-IDE, February 2008.
https://instantsvc.svn.sourceforge.net/svnroot/instantsvc/branches/metaprogramming/src/MetaPHP

ezcReflection, SVN Repository, eZ Components, January 2008.
http://svn.ez.no/svn/ezcomponents/experimental/Reflection

Weitere Literatur und Software

[1] R. HIRSCHFELD, *Course on "Metaprogrammierung und Reflection"*, 2007. Lecture Material, Non-public.

[2] G. BRACHA AND D. UNGAR, *Mirrors: design principles for meta-level facilities of object-oriented programming languages*, in OOPSLA '04: Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, New York, NY, USA, 2004, ACM, pp. 331–344.

[3] S. GOLEMON, *Runkit*, PHP Extension, The PHP Group, January 2008. http://pecl.php.net/package/runkit

[4] A. P. GREGOR KICZALES, *Open Implementations and Metaobject Protocols*, MIT Press, Cambridge, MA, USA, 1996. http://www2.parc.com/csl/groups/sda/publications/papers/Kiczales-TUT95/for-web.pdf

[5] PHP DOCUMENTATION GROUP, *PHP Manual*, documentation, November 2007. http://www.php.net/manual/en/language.oop5.reflection.php

[6] THE PHP GROUP, *PHP Extension Community Library*, Project Site, January 2008. http://pecl.php.net/