UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Aspekt-Orientierung in PHP

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Computervisualistik

vorgelegt von

Markus Schulte

Erstgutachter:	Prof. Dr. Ralf Lämmel Institut für Informatik
Zweitgutachter:	M.Sc. Informatik Andrei Varanovich Institut für Informatik

Koblenz, im September 2012

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Objekte-Orientierte Programmierung (OOP) stellt das Programmierparadigma der Gegenwart dar, 57% der aktuellen Programmiersprachen sind Objekt-Orientierte [Tio]. Die weite Verbreitung der OOP ergibt sich aus den Vorteilen während der Software-Entwicklung für die Beteiligten, wie etwa Entwickler und Software-Architekten. OOP ermöglicht es, sehr komplexe Systeme soweit zu zerlegen, dass ein einfaches Verständnis des Gesamtsystems bei gleichzeitigem Verständnis der Anwendungsdetails möglich ist. Nichtsdestotrotz bietet das OOP-Paradigma nicht für alle Anforderungen an ein Software-System einen Lösungsansatz.

Ein spezielle Art von Anforderung in der Software-Entwicklung stellen die Cross-Cutting Concerns (CCCs) dar. Dabei handelt es sich um Anforderungen, die sich in vielen bis hin zu sämtliche Schichten einer ansonsten modularisierten Software finden. Das OOP-Paradigma bietet für diese Art von Anforderung aufgrund der Idee der Funktionalitäts-Kapselung in einer Modul-/Klassen-Hierarchie keinen direkten Lösungsweg, so dass einzelne CCCs in sämtlichen Schichten der Software semantisch wiederholt behandelt werden.

Die Problematik der CCCs und eine mögliche Unterstützung in Form des Aspekt-Orientierte Programmierung (AOP)-Paradigmas wurden bereits 1997 vorgestellt [Kic+97]. Das Team um Kiczales erfasste diese Art der Anforderungen an eine Software und erarbeitet eine mögliche Lösung in Form der Aspekt-Orientierten Programmierung. Als Ergänzung des OOP-Paradigmas ermöglichen es

Aspekte, diese Art der Belange zu zentralisieren und gleichzeitig sicherzustellen, dass auch spätere Software-Erweiterungen die CCCs behandeln.

Die weit verbreitete Programmiersprache PHP bietet keine native Möglichkeit, CCCs zu unterstützen. Die negativen Auswirkungen der mangelnden CCCs-Unterstützung durchziehen somit eine in PHP geschriebene Software und implizieren, dass die Software schwerer zu verstehen, zu testen und zu warten ist.

Andere Programmiersprachen neben PHP unterstützen CCCs durch AOP oder andere Ansätze. Für die Sprache PHP selbst wurden Versuche unternommen, die fehlende Unterstützung zu ergänzen. Die meisten dieser Versuche sind nicht (mehr) verwendbar, auch die aktiven Projekte können nicht alle Anforderungen des AOP-Paradigmas erfüllen.

Diese Diplomarbeit hat das Thema der fehlenden CCCs-Unterstützung in PHP zum Inhalt. Die Basis bilden zu definierende Anforderungen an eine AOP-Realisierung im PHP-Umfeld. Es wird analysiert, wie und ob verwandte Sprachen und Paradigmen es gestatten, CCCs zu unterstützen. Darüber hinaus wird die Möglichkeit erörtert, AOP in PHP ohne PHP-Erweiterung zu realisieren. Weiter werden die bisherigen Ansätze, AOP in PHP umzusetzen, qualitativ untersucht. Die vorliegende Arbeit zielt darauf ab, eine eigene AOP-PHP-Lösung zu präsentieren, die nicht die Schwächen existierender Lösungen teilt.

1.2 Aufbau der Diplomarbeit

Zu Beginn dieser Diplomarbeit werden dem Leser in Kapitel 2 die notwendigen Grundlagen präsentiert. Zunächst wird in die Herausforderung der CCCs eingeführt. Aufbauend hierauf wird die Aspekt-Orientierte Programmierung vorgestellt. Den Abschluss des Kapitels bildet eine Vorstellung der Programmiersprache PHP, wobei auf Spezifika der Sprache bzw. der Sprachumgebung eingegangen wird, die in dieser Diplomarbeit von Belang sind.

Anschließend werden die Anforderungen an eine AOP-PHP-Lösung in Kapitel 3 analysiert. Es wird dokumentiert, welche minimalen Anforderungen solch eine Lösung erfüllen muss, und welche Anforderungen darüber hinaus die Lösung erfüllen sollte.

Im 4. Kapitel wird das Thema der CCCs außerhalb des PHP-Kontextes betrachtet. Mit AspectJ wird eine AOP-Implementierung in der Programmiersprache Java vorgestellt, welche als Referenzimplementierung gesehen werden kann.

Die Möglichkeit der Metaprogrammierung in der Objekt-Orientierten Programmiersprache Ruby bieten einen Einblick in die (teilweise) Unterstützung von CCCs ohne die Notwendigkeit, AOP anzuwenden. Das in diesem Kapitel vorgestellte Context-Orientierte Programmierung (COP)-Paradigma bietet u. a. einen alternativen Ansatz zu AOP, um CCCs (teilweise) zu unterstützen.

Im folgenden Kapitel 5 wird die Möglichkeit analysiert, in der aktuellen PHP-Umgebung AOP ohne weitere Sprachanpassung anzuwenden. Dabei wird die Umsetzung durch PHP-Reflexion oder Traits behandelt.

Anschließend beschäftigt sich das Kapitel 6 mit den bisherigen Ansätzen, AOP in PHP zu unterstützen. Es werden die inaktiven und aktiven Projekte vorgestellt und die relevanten Projekte an den definierten Anforderungen an eine AOP-PHP-Lösung aus Kapitel 3 evaluiert.

In Kapitel 7 wird eine im Zuge dieser Diplomarbeit entwickelte AOP-PHP-Lösung namens YAPAF beschrieben und diskutiert. Aus diesem Grund werden das zugrundeliegende Konzept sowie dessen Umsetzung präsentiert. Der mögliche Einsatz der entwickelten Lösung wird anhand der Realisierung von Dependency Injection (DI) als Anwendungsbeispiel erläutert.

Im abschließenden Kapitel 8 werden die Ergebnisse dieser Diplomarbeit zusammengefasst und analysiert, sowie ein Ausblick auf mögliche weitere Entwicklungen gegeben.

Kapitel 2

Grundlagen

Dieses Kapitel umfasst das nötige Grundlagenwissen, auf dem die folgenden Kapitel aufbauen.

Kapitel 2.1 gibt eine allgemeine Einführung in das Thema Cross-Cutting Concerns und erklärt diese im Kontext der Objekt-Orientierten Sprachen. Das anschließende Kapitel 2.2 ist eine Einleitung in das Thema Aspekt-Orientierung. Das Kapitel 2.3 beschäftigt sich mit der Programmiersprache PHP. Nach einer Einführung in die Sprache und der Wiedergabe der geschichtlichen Entwicklung werden PHP-Spezifika analysiert, die in dieser Diplomarbeit von Bedeutung sind.

2.1 OOP und Cross-Cutting Concerns

Die Objekt-Orientierte Programmierung stellt das derzeit verbreiteste Programmierparadigma dar (vgl. Kapitel 1.1). Eine typische OOP-Software wendet die folgenden Softwaretechnik-Prinzipien an: Abstraktion, Hierarchisierung, Modularisierung, Geheimnisprinzip, Lokalisierung und Einfachheit [Som06]. So wird aus einer komplexen Anforderung nach dem „Divide-and-conquer“-Prinzip eine Software bestehend aus Schichten, Paketen bzw. Modulen und zuletzt Klassen. Jedes Teilsystem übernimmt klar abgegrenzte Aufgabengebiete, wobei eine Aufgabe genau einmal umgesetzt wird. Diese übersichtliche Struktur wird durch CCCs gestört. Ein Concern umfasst eine Anforderung, die sich eben nicht auf ein Teilsystem sondern auf eine Vielzahl von Teilsystemen erstreckt [Kic+97].

Ein triviales Beispiel sei eine Model View Controller (MVC)-Applikation mit Zugriff auf ein Enterprise Information System (EIS). Der Zugriff auf dieses EIS kann aus sämtlichen Schichten der Applikation erfolgen. Eine Anforderung soll sein, die tatsächliche Ausführungsdauer zugehörig zu einer Anfrage ans EIS zu messen. In einem klassischen OOP-Ansatz muss an jeder EIS-aufrufenden Codestelle ein Tracing implementiert werden. Es entstehen die Gefahren von semantischem Copy and Paste (C+P): Es kann beispielsweise bei neuen Modulen vergessen werden, das Tracing zu implementieren; oder Änderungen am Tracing werden nicht an allen nötigen Vorkommen durchgeführt. Das AOP-Paradigma beschäftigt sich mit diesen Überlegungen und bietet eine mögliche Lösung, CCCs in einer OOP-Software zu ermöglichen, ohne o. g. Nachteile in Kauf nehmen zu müssen.

2.2 Aspekt-Orientierte Programmierung

AOP hat den Zweck, die Umsetzung eines CCCs (vgl. Kapitel 2.1) an zentraler Stelle zu implementieren, so dass dieser Concern nicht die gesamte Software durchzieht. Die Kapselung erfolgt dabei in sogenannten „Aspekten“. Kiczales et al. definieren bei ihrer Vorstellung von AOP 1997 Aspekte wie folgt: „[A property is] An aspect, if it can not be cleanly encapsulated in a generalized procedure.“[Kic+97]. Im Gegensatz hierzu stehen Komponenten, die sich sauber kapseln lassen, beispielsweise in Klassen.

Das Ziel von AOP ist es, Komponenten und Aspekte sauber voneinander zu trennen, und so disjunkte Funktionalitäts-Bereiche zu schaffen, die den softwaretechnischen Prinzipien genügen (vgl. Kapitel 2.1). Ein AOP-Programm besteht demnach aus Komponenten als auch Aspekten, die zusammen die Gesamtfunktionalität ergeben.

2.2.1 Sprachkonstrukte

Um Aspekte zu realisieren führt die Aspekt-Orientierung neue Sprachkonstrukte ein, welche die OOP-Sprachwelt aus Klassen, Methoden u. v. m. ergänzen [Kic+01].

JoinPoint Ein JoinPoint definiert einen Punkt der Programmlogik eindeutig, beispielsweise das Lesen einer Klassenvariablen. JoinPoints dienen damit der Koordination von bestehender Programm- und Aspektlogik.

Pointcut Ein Pointcut setzt sich aus mehreren JoinPoints zusammen, und enthält ggf. zusätzliche Werte aus dem Kontext der JoinPoint-Ausführung.

Advice „[An] Advice is a method-like mechanism used to declare that certain code should execute at each of the join points in a pointcut.“ [Kic+01]. Ein Before-Advice ist also bspw. Programmlogik, die vor einem Pointcut ausgeführt werden soll. Ein Call-Advice bezieht sich auf einen konkreten Aufruf, ein After-Advice soll nach einem Pointcut ausgeführt werden.

Kombiniert ergeben diese Sprachkonstrukte einen Aspekt. Dieser enthält demzufolge bestimmte Logiken, die zu bestimmten Zeitpunkten der Ausführung des Programmes aktiviert werden sollen.

Beispiele für Aspekte in anderen Programmiersprachen werden in dieser Diplomarbeit in Kapitel 4 präsentiert.

2.2.2 Weben

Das Zusammenführen der Funktionalität von Komponenten und Aspekten wird durch das sogenannte „Weben“ in einem Aspect_Weaver realisiert. Dieser akzeptiert als Eingabe die Komponenten bzw. ihre programmiersprachliche Umsetzung und die Aspekte. Das Ergebnis des Webens ist das Gesamtsystem, also die ausführbare Gesamtfunktionalität.

Das Weben kann zu unterschiedlichen Zeiten der Programmausführung geschehen [PGA02]:

Compile-time Bei diesem Ansatz wird das Weben während des Kompilierens vorgenommen. Dies kann durch Preprocessing erfolgen, aber auch durch das Modifizieren des vorhandenen Compilers oder den Einsatz eines eigenen Compilers.

Load-time Beim Weben zur Ladezeit werden Aspekte während des Ladens von Klassen angewendet. Zu diesem Zweck muss der Classloader das Weben unterstützen.

Run-time Beim Anwenden von Aspekten zur Laufzeit wird die Programmausführung während der Laufzeit angepasst, um die Funktionalität der Aspekte umzusetzen.

2.3 PHP

PHP (ein rekursives Akronym für „PHP Hypertext Preprocessor“ [Big10]) ist eine der derzeit gefragtesten Programmiersprachen überhaupt [Lan]. Gleichzeitig gehört PHP zu den Programmiersprachen, die einer breiten Front an Kritik gegenüberstehen[1]. 77,9% der Webseiten werden mit PHP betrieben [W3t].

Der Erfolg von PHP im Sinne von Verbreitung ergibt sich aus einigen Kerneigenschaften der Sprache selbst:

Einfachheit „The key technical contributor to PHP success is its simplicity, which translates into shorter development cycles, easier maintenance and lower training costs.“[Gar09].

Mächtigkeit PHP bietet neben der OOP-Unterstützung auch solche für prozedurales/funktionales Vorgehen. Über verschiedene Erweiterungen ist es einfach möglich, sämtliche Alltagsaufgaben wie Datenbankzugriff, I/O Manipulation, Bildmanipulation etc. durchzuführen.

Dokumentation Die Dokumentation unter php.net [Php] bietet zumindest für die Alltags-Spracheigenschaften eine ausführliche Dokumentation inklusive hilfreicher Kommentare durch PHP-Anwender.

Breite Unterstützung Es gibt zahlreiche PHP-Frameworks, so dass Standard-Aufgaben leicht gelöst werden können. Die Funktionalitäts-Reichweite erstreckt sich dabei von Object Relational Mapping (ORM)-Frameworks wie Doctrine[2], Komponenten-Frameworks wie dem Zend Framework[3], bis hin zu vollständigen Content Management System (CMS)-Lösungen wie Joomla[4].

Für gewöhnlich wird PHP eingesetzt, um Webapplikationen zu betreiben. Die hierfür eingesetzt Ausführungsumgebung besteht in der Regel aus einem Webserver wie dem Apache httpd[5] mit der Einbettung von PHP.

[1] „PHP has no formal semantics, no rigorous test suite, and an incomplete manual.“ [Big10, Kapitel 6.2]. Das Handbuch ist insbesondere außerhalb der Kernfunktionalität lückenhaft, etwa bei PHP-Interna.

[2] http://www.doctrine-project.org/

[3] http://framework.zend.com/

[4] http://www.joomla.de/

[5] http://httpd.apache.org/

Diese Diplomarbeit beschäftigt sich ausschließlich mit der Programmiersprache PHP selbst, in seiner aktuellen Version 5.4. Daher wird PHP nur auf Kommandozeilenebene verwendet, andere Aspekte wie die Einbindung in einen Webserver sind nicht relevant.

2.3.1 Geschichtliche Entwicklung

Der erste Version von PHP erschien im Jahr 1995. Zu diesem Zeitpunkt war PHP die Abkürzung für „Personal Home Page/Forms Interpreter“ und in Perl geschrieben. PHP1 war als spezifische Sprache gedacht, um Vorlagen für HTML-Dokumente zu benutzen. PHP2 erschien im Jahre 1996 und war das erste Mal komplett in C geschrieben – C ist seitdem die Grundlage für PHP. PHP3 erschien 1998 und bringt die erste Unterstützung für OOP. Mit PHP4 aus dem Jahre 2000 ist zum ersten Mal die neue Zend Engine Bestandteil von PHP. Bei der Zend Engine handelt es sich um Interpreter für PHP in Form einer virtuellen Maschine. PHP5 beinhaltet im Jahr 2004 die Zend Engine 2, die das Objektmodell von PHP um wertvolle OOP-Funktionen erweitert und die Performance verbessert [Big10].

2.3.2 Spracheigenschaften von PHP

Bei PHP handelt es sich um eine hybride Sprache mit Unterstützung für prozedurales, funktionales und Objekt-Orientiertes Vorgehen. Diese duale Eigenschaft ergibt sich aus den prozeduralen Eigenschaften der ersten PHP-Versionen, die nicht aus der Sprache entfernt worden sind, sondern parallel zur Objekt-Orientierung ab Version3 weiter gepflegt worden sind [Big10].

Das Typsystem von PHP ist dynamisch und schwach. Die dynamische Eigenschaft bedeutet, dass eine Variable zur Laufzeit keine Restriktionen bzgl. des referenzierten Wertes unterliegt. Das schwache Typsystem bewirkt, dass vorhandene Werte ihren Typ automatisch zur Laufzeit ändern, falls der aktuelle Werttyp nicht dem geforderten entspricht.

Weitere Eigenschaften von PHP sind [Big10]:

Skalare Typen PHP besitzt die skalaren Typen int, real, string, bool, resource und null.

Klassensystem Das Klassensystem von PHP ist statisch, d. h. der vorhandene Ableitungsbaum lässt sich nicht manipulieren. PHP unterstützt ausschließlich Einfachvererbung. PHP-Objekte können nicht ihren Klassentyp ändern.

Operatoren Die Vergleichsoperatoren liegen jeweils in einer Version für den einfachen Wertvergleich „==“ sowie einer weiteren Version zur zusätzlichen Prüfung des Werttyps „===“ vor.

Zuweisung PHP unterstützt die Zuweisung per Kopie „$a = $b“ als auch per Referenz „$a = &$b“.

Inkludieren zur Laufzeit Das wichtige Inkludieren weiterer PHP-Resourcen erfolgt zur Laufzeit, der eingebundene PHP-Code wird (erst) zur Laufzeit analysiert und ausgeführt.

Garbage Collection PHP unterstützt Garbage Collection und entbindet damit den Software-Entwickler von den Pflichten des Speichermanagements.

2.3.3 PHP-Ausführung

Ein wesentlicher Bestandteil der Aspekt-Orientierung stellt das Weben dar, also das Anwenden von Aspekten auf bestehenden Sourcecode (vgl. Kapitel 2.2.2). Dafür ist es notwendig, im Detail und nötiger Tiefe zu verstehen, wie genau bei PHP die Ausführung stattfindet, um eine gut begründete Wahl für die Umsetzung des Webens treffen zu können, insbesondere da PHP als interpretierte Sprache keinen Compiler im klassischen Sinne mitbringt. Es wird an dieser Stelle die reine PHP-Ausführung auf Kommandozeilenebene betrachtet, sämtliche sich anschließenden Operationen wie PHP-Opcode-Caching (vgl. Kapitel 2.3.4) oder die PHP-Ausführung durch einen (Web-)Server bauen hierauf auf.

Die Ausführung von PHP-Sourcecode erfolgt in vier Schritten [Aho+06][CT09, Kapitel 4]:

Lexikalische Analyse Ein Lexer übernimmt die Umwandlung einer Zeichenkette in eine Folge von PHP_Token.

Syntaktische Analyse Analyse der PHP_Token.

Bytecode-Generierung Umwandlung der PHP_Token in PHP-Opcode, den sogenannten „Bytecode“.

Listing 2.1: Ein einfaches PHP-Beispiel [Ber]

```
1 <?php
2 if (TRUE) {
3   print '*';
4 }
5 ?>
```

Tabelle 2.1: PHP-Source mit zugehörigen PHP_Token [Ber]

Line	Source	PHP_Token
1	<?php	T_OPEN_TAG
2	if	T_IF
		T_WHITESPACE
	(	(
	TRUE	T_STRING
	)	)
		T_WHITESPACE
	{	{
3		T_WHITESPACE
	print	T_PRINT
		T_WHITESPACE
	'*'	T_CONSTANT_ENCAPSED_STRING
	;	;
4		T_WHITESPACE
	}	}
5		T_WHITESPACE
	?>	T_CLOSE_TAG

Bytecode-Ausführung Ausführen des Bytecode.

Lexikalische Analyse

Die Grundlage der PHP-Ausführung bildet der PHP-Sourcecode. Im ersten Schritt der Ausführung wird aus dem Sourcecode durch den Lexer eine Aufeinanderfolge sogenannter PHP_Token[6] erzeugt.

Zur Verdeutlichung dient das einfache PHP-Beispiel Listing 2.1 und die zugehörige Zuordnung von PHP-Instruktionen zu PHP_Token in der Tabelle 2.1. Die linke Spalte der Tabelle „Line“ gibt die Zeile an, in der der zugehörige PHP-

[6]PHP_Token Referenz: http://php.net/manual/en/tokens.php

Listing 2.2: Ein PHP-Beispiel mit syntaktischem Fehler „parseerror.php“

```
1 <?php
2 if (TRUE {
```

Listing 2.3: PHP-Syntax-Fehler (vgl. Listing 2.2)

```
1 root@debian-vt: php parseerror.php
2 PHP Parse error:  syntax error, unexpected '{' in /root/
    parseerror.php on line 2
```

Sourcecode „Source“ vorhanden ist. Die rechte Spalte „PHP_Token“ visualisiert das Ergebnis des PHP-Lexers. Der Lexer analysiert den Sourcecode. Falls der Lexer bekannte Ausdrücke erkennt, werden diese als PHP_Token interpretiert. Der erste vom Lexer erkannte Ausdruck ist die Zeichenkette „<?php“ (vgl. Zeile 1), dies entspricht dem PHP_Token „T_OPEN_TAG“. Auf diese Art wird der komplette Sourcecode in der Reihenfolge der Anweisungen analysiert und die zugehörigen PHP_Token produziert.

Syntaktische Analyse

Die syntaktische Analyse folgt der lexikalischen und arbeitet auf dessen Ergebnis, den PHP_Token. In dieser Phase wird die Abfolge der PHP_Token auf Korrektheit anhand der PHP-Grammatik[7] untersucht. Die syntaktische Korrektheit ist zwingende Voraussetzung für die weitere Verarbeitung bzw. Ausführung des Programmes. Ein Beispiel für eine Regel der Grammatik ist die Regel „unticked_statement: T_IF ’(’ expr ’)’ statement“; der PHP-Ausdruck „if (true) ...“ erfüllt diese Regel.

Listing 2.3 zeigt beispielhaft die Ausführung syntaktisch fehlerhaften PHP-Sourcecodes. Die Regel, dass bei einem if-Statement eine Expression in öffnende „(“ und schließende „)“ Klammer gesetzt werden muss, wird verletzt; demzufolge wird ein Parse-Error ausgegeben.

[7] Die PHP-Grammatik findet sich unter https://github.com/php/php-src/blob/master/Zend/zend_language_parser.y

Tabelle 2.2: Zend_Opcodes zugehörig zu Listing 2.1

line	#	*	op	fetch	ext	return	operands
2	0	>	> JMPZ				true, ->4
3	1	>	PRINT				'%2A'
	2	FREE					
4	3	> JMP					->4
6	4	>	> RETURN				1

Bytecode-Generierung

Nachdem durch die syntaktische Analyse die Korrektheit der vorliegenden PHP-_Token garantiert ist, wird in der nun folgenden Phase aus den Token der PHP-Bytecode bestehend aus einer Folge von sogenannten Zend_Opcodes[8] generiert.

In der Tabelle 2.2 ist dieser Zwischenschritt zu sehen, die Ermittlung erfolgte durch die php-vld-extension[9]. Die ursprüngliche PHP-Anweisung „print '*'“ (Zeile 3, Listing 2.1) liegt in dieser Phase als die Zend_Opcode Anweisung „PRINT“ mit dem Operator „%2A“[10] vor.

Bytecode-Ausführung

Im letzten Schritt der PHP-Ausführung wird der erzeugte Stack von Zend_Opcode-Anweisungen durch die Zend Engine ausgeführt [CT09, Kapitel 4]. Bei der Zend Engine handelt es sich um einen PHP-Interpreter bzw. eine Virtual Machine (VM), die je nach unterliegendem Betriebssystem unterschiedlich implementiert ist. Diese Ausführung in einer VM ist der Grund dafür, dass PHP plattformunabhängig entwickelt werden kann.

2.3.4 Opcode-Caching

Kapitel 2.3.3 stellt die Zend_Opcodes vor – ein (Zwischen-)Ergebnis in der PHP-Ausführung. Die Zend_Opcodes werden während der Ausführung von PHP bei jeder Programmausführung erneut für den vorhandenen Sourcecode erstellt. Die-

[8] http://www.php.net/manual/en/internals2.opcodes.list.php

[9] http://hengrui-li.blogspot.de/2011/07/review-php-opcode-with-vld.html

[10] „2A“ entspricht dem '*' der Anweisung in utf8 (http://en.wikipedia.org/wiki/Asterisk)

Listing 2.4: Explizite Abhängigkeiten zwischen PHP-Klassen

```
ClassA.php:

<?php
class ClassA {
}


ChildOfClassA.php:

<?php
require_once("ClassA.php");
class ChildOfClassA extends ClassA {
}
```

se Erstellung geschieht auch, wenn keine Sourcecode-Änderungen gegenüber der letzten Ausführung des Codes vorliegen. In diesem Fall besteht die Möglichkeit, die Ausführung eines PHP-Programmes zu beschleunigen, indem die generierten Opcodes zwischengespeichert anstatt verworfen werden.

Die Zend_Engine (vgl. Kapitel 2.3.3) gestattet das Einhängen von Programmen als PHP-Erweiterungen, um dieses Zwischenspeichern zu bewerkstelligen. Diese Erweiterungen werden unter dem Begriff „PHP accelerators“ zusammengefasst. Tatsächlich ist das angedeutete Szenario beim praktischen Einsatz von PHP der Normalfall, eingesetzte Opcode-Caches sind beispielsweise „Alternative PHP Cache“[11] und „eAccelerator“[12]. Die PHP accelerators übernehmen neben dem reinen Zwischenspeichern des Opcodes auch Optimierung an diesem, beispielsweise durch Entfernen unnötiger Anweisungen. Die Opcode-Caches beschleunigen die PHP-Ausführung merklich. Messungen bei dem PHP-CMS Drupal[13] belegen eine Verdreifachung der Ausführungsgeschwindigkeit [Dru].

2.3.5 Autoloading

Ein modernes OOP-PHP-Programm besteht aus einer Vielzahl von Klassen, die sich normalerweise auf jeweils eine Datei verteilen.

[11] http://www.php.net/manual/en/intro.apc.php
[12] http://sourceforge.net/projects/eaccelerator/
[13] http://www.drupal.de/

Listing 2.5: Einsatz des PHP-Autloaders

```
1 <?php
2 function __autoload($class_name) {
3     include $class_name . '.php';
4 }
5
6 $obj  = new MyClass();
```

Das Listing 2.4 zeigt das explizite Einbinden von Abhängigkeiten innerhalb einer PHP-Datei. Die Kindklasse „ChildOfClassA“ inkludiert durch die PHP-Methode require_once() die Datei „ClassA.php“ (vgl. Zeile 11), in der die Mutterklasse „ClassA“ enthalten ist. Dieses Vorgehen ist mit sehr hohem manuellen Aufwand für den Entwickler verbunden.

Eine oft genutzte Alternative zur expliziten Deklaration ist das Benutzen des PHP-Autoloaders. Dabei handelt es sich um eine im PHP-Programm zu definierende Funktion Namens „__autoload“, die einen Klassennamen entgegennimmt und für diese Klasse die entsprechende PHP-Quelltext-Datei lädt. Der Aufruf des Autoloaders findet statt, sollte eine angeforderte PHP-Klasse zum Aufrufzeitpunkt noch nicht verfügbar sein. Weiter muss in diesem Fall der Autoloader natürlich verfügbar sein, eine frühe Platzierung in der Programmausführung bietet sich an.

Das Listing 2.5 zeigt die Benutzung des PHP-Autoloaders[14]. Im gezeigten Beispiel ist zum Zeitpunkt des Aufrufes die Klasse „MyClass“ nicht verfügbar (vgl. Zeile 6). Infolgedessen wird der PHP-Autoloader aktiviert, der die include()-Funktion aufruft, um die Datei „MyClass.php“ zu inkludieren. Im Fehlerfall verhält sich der Autoloader analog zu dem Versuch, eine nicht vorhandene Datei mittels include() einzubinden – es wird mit einem Fehler terminiert.

Eine Alternative zum Verwenden einer globalen __autoload()-Funktion stellt die Registrierung einer Autoloader-Methode oder -Funktion durch spl_autoload_register() dar. Durch Nutzung dieser Funktion entfällt der Zwang, eine im globalen Programm-Kontext vorhandene __autoload()-Funktion definieren zu müssen, es kann eine lokale Methode zum Autoloading bestimmt werden.

[14] http://php.net/manual/de/language.oop5.autoload.php

Listing 2.6: Beispiel für einen PHP-include_path nach [Php]

```
1 include_path=".:/php/includes"
```

2.3.6 include_path

PHP bietet die Möglichkeit, Streams über verschiedene Funkionen einzubinden. Im Kontext dieser Diplomarbeit sind vor allem die Funktionen zum Einbinden lokaler Dateien wie include() und require() von Bedeutung.

Das Einbinden von Dateien durch diese Methoden kann auf zwei Arten erfolgen:

Absolute Einbindung In diesem Fall wird der Funktion ein absoluter Dateipfad, beginnend mit einem „/“, übergeben. Die Funktion inkludiert genau diese Datei. Falls diese Datei nicht eingebunden werden kann, terminiert die Funktion mit einem Fehler.

Relative Einbindung Im Falle einer relativen Einbindung wird der Funktion ein relativer Dateipfad übergeben, beispielsweise „Class.php“. Die einbindende Funktion versucht infolgedessen nicht nur, diese Datei relativ zum Current Working Directory (CWD) einzubinden, sondern auch relativ zu Verzeichnissen, die im sogenannten „include_path“ angegeben sind.

Beim include_path handelt es sich um eine Liste von Verzeichnissen (getrennt durch „:“), die PHP im Falle einer relativen Dateieinbindung mit einbezieht. Listing 2.6 zeigt ein Beispiel für diese Direktive, es sind zwei Ordner im include_path vorhanden: „.“ (also das CWD) sowie „/php/include“. Diese php.ini-Direktive ist durch die PHP-Applikation konfigurierbar[15] und kann in Verbindung mit dem Autoloader (vgl. Kapitel 2.3.5) verwendet werden, um automatisch PHP-Klassen einzubinden.

2.3.7 stream-wrapper

PHP bringt mit der Funktion stream_wrapper_register()[16] die Möglichkeit mit, Streams durch einen eigenen Handler beim Einlesen zu manipulieren. „[This functi-

[15] http://de.php.net/manual/de/function.set-include-path.php

[16] http://php.net/manual/en/function.stream-wrapper-register.php

Listing 2.7: Benutzung der PHP-Funktion stream_wrapper_register

```
1  <?php
2
3  class VariableStream {
4    ...
5  }
6
7  stream_wrapper_register("var", "VariableStream");
8  $myvar = "";
9
10 $fp = fopen("var://myvar", "r+");
```

on] allows you to implement your own protocol handlers and streams for use with all the other filesystem functions“ [Php]. Die Funktion erwartet als Eingabe im ersten Parameter die Angabe eine Namens für den Wrapper, im zweiten Parameter den Namen einer PHP-Klasse, die als Wrapper dient.

Listing 2.7 zeigt beispielhaft die Verwendung von stream_wrapper_register. Die PHP-Klasse „VariableStream“ dient hier als Wrapper, sie hat die nötige Funktionalität für das Einlesen eines Streams bereitzustellen. Dies bedeutet, dass die Methoden stream_open(), stream_read(), stream_write(), stream_tell(), stream_eof(), stream_seek() und stream_metadata() enthalten sein müssen, natürlich unter Einhaltung der nötigen Schnittstelle. Die Funktion stream_register_wrapper() registriert dieses Klasse als Wrapper-Klasse (in diesem Beispiel) unter dem Protokoll „var“. Bei Stream-Operationen wie der fopen()-Methode lässt sich nun dieses Protokoll angeben, so dass statt des Standard-PHP-Handlers die Streamverarbeitung durch die eigene Wrapper-Klasse stattfindet.

2.3.8 Metadata

Bei Metadaten handelt es sich um Daten, die zusätzliche Informationen über vorhandene Daten zur Verfügung stellen. Diese zusätzlichen Informationen sind nicht für die normale Ausführung eines Programmes notwendig.

PHP bietet keine Möglichkeit, ein Programm um Metainformationen zu bereichern. Projekte wie PHPUnit (vgl. Kapitel 2.3.10) und PHPDoc[17] nutzen Eigenentwicklungen, um Metainformationen in Form von speziellen PHP-Kommentaren,

[17] http://www.phpdoc.org/

Listing 2.8: Verwendung von addendum

```
1 /** @Persistent */
2 class Person {
3    // some code
4 }
5
6 $reflection = new ReflectionAnnotatedClass('Person'); // by
     class name
7 $reflection->hasAnnotation('Persistent'); // true
```

sogenannten Annotationen, zu erfassen. Weiter liegt mit „addendum“[18] eine PHP-library vor, die das Nutzen von Annotationen in PHP erlaubt.

Listing 2.8 zeigt ein addendum-Beispiel, dieses entstammt der addendum-Projektseite[19]. In diesem Beispiel wird einer Klasse „Person“ die Metainformation „@Persistent“ in Form einer Annotation zugeordnet (vgl. Zeile 1). Die Library addendum stellt die Klasse „ReflectionAnnotatedClass“ bereit, mit der auf Annotationen zugegriffen werden kann (vgl. Zeile 6). Im Beispiel wird auf das Vorhandensein der Metainformation „Persistent“ geprüft (vgl. Zeile 7).

2.3.9 Reflexion-API

Bei Reflexion handelt es sich um die Fähigkeit einer Programmiersprache, die Daten und das Verhalten eines Programmes zur Laufzeit zu analysieren und zu manipulieren. PHP besitzt seit der Version 5.0 eine einheitliche Reflexion-API[20,21], mit der sich Laufzeiteigenschaften ermitteln und manipulieren lassen.

Introspektion

Introspektion ist eine Untermenge von Reflexion: Introspektion beschreibt die Fähigkeit, lesend auf Eigenschaften eines Programmes zuzugreifen.

[18] http://code.google.com/p/addendum/

[19] http://code.google.com/p/addendum/wiki/ShortTutorialByExample

[20] Version 5.0.0 Beta 2, „Added Reflection API“ http://www.php.net/ChangeLog-5.php

[21] Referenz: http://de3.php.net/manual/en/book.reflection.php

In seiner Objekt-Orientierten Ausprägung bietet PHP mit den Klassen ReflectionClass, ReflectionMethod und ReflectionProperty die Möglichkeit, Informationen über Klassen zu erhalten, u. a.:

- Klassenkonstanten, Informationen über den vorhandenen Konstruktor, implementierte Methoden, sowie Sichtbarkeit der Elemente.
- Für Klassenmethoden lassen sich neben der Sichtbarkeit auch die Eingabeparameter bestimmen.
- Bei Klassenvariablen lassen sich für eine vorhandene Objektinstanz die aktuellen Werte bestimmen.

Daneben deckt PHP auch die funktionale Seite der Sprache ab, die Klasse ReflectionFunction[22] dient diesem Zweck.

Reflexion

Die Introspektion gestattet es, Laufzeit-Eigenschaften eines Programmes zu ermitteln. Daneben bietet die Reflexion-API von PHP auch die Möglichkeit, das ausführende Programm in gewissem Umfang zu manipulieren:

- Es lassen sich neue Objektinstanzen erzeugen.
- Das Setzen von statischen Klassenvariablen ist möglich.
- Die Sichtbarkeit einer Klassenmethoden ist veränderbar.
- Eine Objektmethode lässt sich via Reflexion ausführen.
- Die Sichtbarkeit und der Wert einer Klassenvariablen ist veränderbar.

PHP-Metaprogrammierung außerhalb der Reflexion-API

Neben der Reflexion-API bietet PHP noch weitere Möglichkeiten für Metaprogrammierung.

Nennenswert ist hierbei die Umsetzung von Überladung im PHP-Sinne. „Overloading in PHP provides means to dynamically "create"properties and methods.

[22] /manual/en/class.reflectionfunctionabstract.php auf [Php]

Listing 2.9: Magische Methode „__set()“

```
1  <?php
2  class PropertyTest {
3    public function __set($name, $value) {
4      echo "Setting '$name' to '$value'\n";
5    }
6  }
7
8  $obj = new PropertyTest;
9  $obj->a = 1;
10
11 > Setting 'a' to '1'
```

[...] The overloading methods are invoked when interacting with properties or methods that have not been declared or are not visible in the current scope.“[23].

Dies bedeutet, dass für den Fall des Zugriff auf undefinierte Klassenvariablen oder -methoden sogenannte magische Methoden[24] innerhalb der Klasse genutzt werden können, um dieses Ereignis zu behandeln. Im Falle des Setzens einer Klassenvariable kann auf den neuen Wert zugegriffen werden, beim Aufruf einer unbekannten Methode sind die übergebenen Parameter verfügbar.

Listing 2.9 verdeutlicht dies am Beispiel (das Beispiel ist angelehnt an der o. g. Dokumentation zu Overloading). Es ist eine Klasse „PropertyTest“ mit vorhandener magischer Methode __set() zu sehen (vgl. Zeile 3). Auf einer Instanz dieser Klasse wird eine nicht vorhandene Klassen-Eigenschaft „a“ mit dem Wert „1“ gesetzt (vgl. Zeile 9). Durch das Fehlen von „a“ wird die magische Methode __set() durch PHP aufgerufen, die Eingabe ist der Name der fehlenden Eigenschaft so wie der angegebene Wert „1“. In diesem Beispiel werden diese Informationen direkt ausgegeben (vgl. Zeile 11).

2.3.10 PHPUnit

Bei PHPUnit[25] handelt es sich um eine Library für das Testen von PHP-Software. Das Listing 2.10 zeigt ein einfaches Beispiel für den Einsatz von PHPUnit: Die be-

[23] http://de2.php.net/manual/en/language.oop5.overloading.php

[24] http://www.php.net/manual/en/language.oop5.magic.php

[25] https://github.com/sebastianbergmann/phpunit/

Listing 2.10: Ein einfaches PHPUnit-Beispiel

```
1  class HelloWorld {
2    public function returnHello() {
3      return 'Hello World!';
4    }
5  }
6
7  class HelloWorldTest extends PHPUnit_Framework_TestCase {
8    /**
9     * @test
10    */
11   public function returnHelloWorks() {
12     $helloWorld = new HelloWorld();
13     $this->assertEquals('Hello World!', $helloWorld->returnHello
           ());
14   }
15 }
16
17 > phpunit HelloWorldTest.php
18 > PHPUnit 3.6.11 by Sebastian Bergmann.
19 > Configuration read from [...]/phpunit.xml
20 >
21 > .
22 >
23 > Time: 0 seconds, Memory: 2.50Mb
24 > OK (1 test, 1 assertion)
```

kannte Rückgabe der Methode returnHello() der PHP-Klasse „HelloWorld“ wird mittels PHPUnit validiert (vgl. Zeile 13).

Eine Besonderheit von PHPUnit stellt die Möglichkeit dar, jeden definierten Test in einem eigenen Prozess auszuführen[26]. Dadurch lässt sich unter anderem eine bereits in einem vorherigen Test inkludierte Klasse erneut einbinden, ohne dass es zu Fehlern aufgrund des doppelten Einbindens einer Klasse käme.

[26] http://www.phpunit.de/manual/3.6/en/phpunit-book.html#textui.clioptions

Kapitel 3

Anforderungen an eine AOP-PHP-Lösung

Diese Diplomarbeit beschäftigt sich mit bestehenden AOP-PHP-Lösungen und präsentiert eine selbst entwickelte. Zwecks Validierung definiert dieses Kapitel die Anforderungen an eine AOP-PHP-Lösung.

3.1 Minimale Anforderungen

Die minimalen Anforderungen umfassen solche Anforderungen, die eine Implementation in jedem Fall erfüllen muss, um als AOP-PHP-Lösung zu gelten.

- Eine AOP-PHP-Lösung muss es ermöglichen, Aspekte zu realisieren.

Dies ist die genau eine Anforderung an eine AOP-PHP-Lösung. Es muss möglich sein, CCCs durch Aspekte (vgl. Kapitel 2.2) zu unterstützen.

3.2 Optionale Anforderungen

Die optionalen Anforderungen definieren solche, die neben der Kernfunktionalität wünschenswert sind, um eine AOP-PHP-Lösung möglichst einsetzbar und benutzbar zu machen [Kic+01].

Aufwärts-Kompatibilität Jedes korrekte PHP-Programm muss ein korrektes AOP-PHP-Programm sein.

Plattform-Kompatibilität Alle korrekten AOP-PHP-Programme müssen in einer Standard-PHP-Umgebung ausführbar sein.

Werkzeug-Kompatibilität Es muss für bestehende Werkezeuge wie IDEs möglich sein, AOP-PHP zu unterstützen.

Programmierer-Kompatibilität Der Umgang mit AOP-PHP muss sich möglichst ähnlich gestalten wie der Umgang mit einer „normalen“ PHP-Erweiterung.

Diese optionalen Anforderungen beschäftigen sich mit der Anwendbarkeit des Aspektframeworks und verfolgen den Zweck, das Framework möglichst einfach einsetzbar und benutzbar zu halten.

Kapitel 4

Relevante Sprachunterstützung

Es gibt Unterstützung für Cross-Cutting Concerns bereits in einer Vielzahl von Programmiersprachen durch Realisierung von AOP. Daneben gibt es alternative Konzepte und Realisierungen zu dem AOP-Paradigma. Anhand bekannter Sprachen sollen im Folgenden unterschiedliche Lösungsansätze zur Unterstützung von CCCs präsentiert werden.

Zunächst wird mit AspectJ eine AOP-Realisierung für Java vorgestellt. Anschießend behandelt Kapitel 4.2 die Möglichkeit der Metaprogrammierung in Ruby, und setzt dies in den Kontext zu AOP. Kapitel 4.3 analysiert Context-Orientierte Programmierung (COP), ein verwandtes Programmier-Paradigma zu AOP.

4.1 Java: AspectJ

Bei AspectJ[1] handelt es sich um eine Realisierung von AOP (vgl. Kapitel 2.2) in der Programmiersprache Java. AspectJ wurde 2001 am Xerox Palo Alto Research Center durch Gregor Kiczales et al. [Kic+01] präsentiert. Dieses Team zeichnete sich auch für die ersten allgemeinen Überlegungen zu AOP verantwortlich [Kic+97].

Sowohl PHP in seiner Objekt-Orientierten Ausprägung (vgl. Kapitel 2.3.2) als auch Java sind Objekt-Orientierte Programmiersprachen mit dementsprechend ähnlicher Syntax.

[1] www.eclipse.org/aspectj/

Listing 4.1: Ein Aspekt in AspectJ nach [Kic+01]

```
1  aspect SimpleErrorLogging {
2    Log log = new Log();
3
4    pointcut publicEntries():
5      receptions(public * com.xerox.printers.*.*(..));
6
7    after() throwing (Error e): publicEntries() {
8      log.write(e);
9    }
10 }
```

Ein wesentlicher Unterschied zwischen Java und PHP besteht in der Ausführung. Während PHP interpretiert ausgeführt wird (vgl. Kapitel 2.3.3), wird Java-Programmcode zunächst kompiliert. Bei der Kompilierung von Java-Sourcecode handelt es sich um die Übersetzung des Java-Programmcodes in Java-Bytecode zur Ausführung in einer Java Virtual Machine (JVM) [Die98].

Das Listing 4.1 zeigt einen Aspekt in AspectJ. Bei der modellierten Funktionalität handelt es sich um ein Logging von Fehlerfällen innerhalb öffentlicher Bereiche des Programm-Packages „com.xerox.printers“. Das Beispiel besteht aus einem After-Advice, das für den Pointcut „publicEntries“ gilt. Dieser Pointcut referenziert genau einen JoinPoint der Programmausführung (Zeile 5) (AOP-Sprachkonstrukte, vgl. Kapitel 2.2).

AspectJ ermöglicht die Umsetzung von Aspekten gemäß Kapitel 3. Mit AJDT[2] (einer Eclipse-Erweiterung) liegt eine konkrete Realisierung der optionalen Anforderung der „Werkzeug-Kompatibilität“ (vgl. Kapitel 3.2) vor.

AspectJ kann das Weben (vgl. Kapitel 2.2) auf zwei der drei möglichen Arten umsetzen:

Compile-time AspectJ bietet einen eigenen Compiler, der zum Kompilieren des Java-Programmes eingesetzt werden kann. Dieser Compiler akzeptiert als Eingabe das Programm mit seiner normalen OOP-Struktur als auch den Aspekten, und erzeugt auf dieser Grundlage Java-Bytecode [EH04].

[2] www.eclipse.org/ajdt/

Listing 4.2: Ruby-Metaprogrammierung nach [Wam07]

```
# foo.rb
class Foo
  def concat *args
    args.join(" ")
  end
end

# foo_advice.rb
class Foo
  alias old_concat concat
  def concat *args
    "{[< "+ old_concat(args) + " >]}"
  end
end

> puts Foo.new.concat("hello", "world")
> {[< hello world >]}
```

Load-time AspectJ bietet die Möglichkeit, die JVM um Aspekt-Unterstützung zu erweitern[3].

Bei AspectJ handelt es sich um die Referenzimplementierung des Aspekt-Konzeptes von 1997 [Kic+97]. AspectJ wird dem Anspruch an eine Vorlage für AOP-Lösungen gerecht – und kann damit auch als Orientierungspunkt für AOP-PHP-Lösungen dienen. Dies erreicht die Spracherweiterung nicht nur durch die Mächtigkeit der Aspekt-Realisierung, sondern auch die einfache Erlernbarkeit und Handhabung.

4.2 Ruby: Metaprogrammierung

Die Programmiersprache Ruby[4] bietet umfassende Möglichkeiten der Reflexion (vgl. Kapitel 2.3.9); in der Ruby-Gemeinde wird weithin der Begriff „Metaprogrammierung“ verwendet [FM08, Kapitel 8]. U. a. bietet Ruby die Möglichkeit, das Programmverhalten zur Laufzeit zu manipulieren, etwa durch Erweitern/Ändern des Verhaltens einer Methode [FM08, Kapitel 8.11].

[3] http://www.eclipse.org/aspectj/doc/released/devguide/ltw.html

[4] http://www.ruby-lang.org/en/

Listing 4.2 zeigt die Ruby-Fähigkeit, ein Methodenverhalten zu verändern. In diesem Beispiel wird das Verhalten der Methode „concat“ der Klasse „Foo“ (ursprünglich definiert in „foo.rb“, vgl. Zeile 3ff.) verändert. In „foo_advice.rb“ wird die Klasse wieder geöffnet, und durch Ruby-Sprachmechanismen wird eine neue „concat“-Methode definiert, die ein eigenes Verhalten einführt und dabei die ursprüngliche „concat“-Methode nutzt (vgl. Zeile 10ff.).

In Bezug auf CCCs (vgl. Kapitel 2.1) bedeutet die breite Unterstützung für Reflexion, dass Concerns in Ruby nicht den ursprünglichen Geschäftslogik-Code einer Methode durchziehen müssen, sondern sich auslagern lassen – Dean Wampler hat gezeigt, dass dies ein praktikables Vorgehen sein kann [Wam07]. Jedoch lassen sich klassenübergreifende Concerns in Ruby mit dieser Methodik nicht nativ behandeln, da diese Art der Concern-Behandlung an die vorhandene Klassenstruktur oder alternativ die Programmausführung gebunden ist.

Nichtsdestotrotz entschärft Ruby durch die Fähigkeit der Metaprogrammierung das Problem der CCCs teilweise und zeigt damit, dass CCCs auch durch reine Sprachunterstützung gelöst werden können.

4.3 Context-Orientierte Programmierung

Die COP ist ein Programmierparadigma, das sich mit dem unterschiedlichen Verhalten eines Softwaresystems abhängig von Anwender und Umgebung beschäftigt. Eine Ähnlichkeit zwischen COP und AOP ist in Bezug zu CCCs zu nennen: COP bietet u. a. die Möglichkeit, CCCs zu unterstützen [HCN08].

Das COP-Paradigma wurde 2008 von R. Hirschfeld et al. vorgestellt [HCN08]. Es beschäftigt sich mit der grundsätzlichen Problematik, dass sich Software in Abhängigkeit von Nutzereingaben und Umgebungsbedingungen – dem Ausführungs-Kontext – unterschiedlich verhalten soll; die Möglichkeit dieses abhängige Verhalten explizit zu unterstützen wird nach Ansicht der Autoren in Programmier-Hochsprachen nur unzulänglich unterstützt.

Bei COP sind drei Rollen für diesen Kontext von Relevanz [HCN08]:

Akteur Der Akteur ist der Benutzer des Systems. Sein Verhalten bzw. seine Eingaben lösen Reaktionen im System aus.

System Das berechnende System reagiert auf den Akteur und stellt gewünschtes Verhalten zur Verfügung.

Umgebung Die Umgebung umfasst alles in der Relation zwischen Akteur und System, also bspw. die Position oder die Uhrzeit.

Dieses drei Rollen können unabhängig voneinander den Kontext einer Programmausführung verändern und führen so zu Verhaltensvariationen der Software. Bei COP wird die gewünschte Verhaltensveränderung in Abhängigkeit einer der drei Rollen unabhängig und separat beschrieben.

COP führt folgende Spracheigenschaften ein, um Kontext-abhängige Programmierung zu ermöglichen [HCN08]:

Verhaltensvariationen Verhaltensvariationen bestehen aus neuem oder geändertem Verhalten der Software.

Schichten Schichten gruppieren Kontext-abhängige Verhaltensvariationen.

Aktivierung Schichten lassen sich dynamisch zur Laufzeit in Abhängigkeit des Kontext deaktivieren und aktivieren.

Kontext Beim Kontext handelt es sich um Informationen zur Laufzeit.

Bereich Je nach Bereich innerhalb der Software werden Schichten deaktiviert oder aktiviert.

Bei COP werden Schichten Kontext-abhängig deaktiviert oder aktiviert. COP-Sprachunterstützung soll das Beschreiben und Deaktivieren bzw. Aktivieren von Schichten zur Laufzeit ermöglichen.

Mit ContextS liegt eine Realisierung von COP in Squeak[5] vor, einem Smalltalk-Dialekt. Neben der Realisierung in Squeak gibt es auch für weitere Sprachen wie Java, Lisp, Python, Ruby u. v. m. COP-Unterstützung[6].

Listing 4.3 zeigt ContextS im Einsatz. Dieses Listing zeigt eine Schicht „MyLayer“ und eine teilweise Methoden-Definition „adviceCopLeafEvaluate“, die zur Schicht „MyLayer“ gehört. In Zeile 14 ist das Nachrichten-Konstrukt „receiver useAsLayersFor: argument“ zu sehen. Diese Nachricht kann in Squeak an alle Instanzen (in diesem Fall „receiver“) geschickt werden, die ihrerseits Schichten enthalten, um so den Kontext zu aktivieren.

[5] http://www.squeak.org/

[6] http://www.hpi.uni-potsdam.de/hirschfeld/cop/implementations/index.html

Listing 4.3: ContextS nach [HCH08]

```
CsLayer subclass: #MyLayer
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: ''
  category: 'My Category'

MyLayer>>adviceCopLeafEvaluate
  ^ CsAroundVariation
    target: [MyTargetClass -> #myTargetSelector]
    aroundBlock: [:receiver :arguments :layer :client :
       clientMethod |
      "my layer-specific code"
      ...]

receiver useAsLayersFor: argument
```

Wie bereits beschrieben bietet neben AOP auch COP die Möglichkeit, CCCs zu unterstützen. Diese Unterstützung ist jedoch nur teilweise: Bei COP wird die notwendige Verhaltensänderung für CCCs durch die Schichten und deren Aktivierung realisiert, so dass die ursprüngliche Geschäftslogik nicht vom Concern betroffen ist. Anders als bei AOP wird die Logik zum Behandeln von CCCs also nicht in Aspekten, sondern in Schichten zugehörig zur betroffenen Klassen gekapselt. Anders als AOP ist COP jedoch an die Klassenhierarchie der Software gebunden, so dass der Fall eines CCCs, der Klassen-übergreifend die Software betrifft (bspw. eine allgemeine Fehlerbehandlung), mit COP nicht explizit zentralisiert werden kann und weiterhin den Sourcecode durchzieht – wenn auch in Schichten statt im Programmcode selbst.

Kapitel 5

Alternativen in PHP

Dieses Kapitel beschäftigt sich mit den Möglichkeiten, die in PHP bereits alternativ vorliegen um Aspekte gemäß Kapitel 2.2 zu realisieren, und gibt anhand der Anforderungsdefinition aus Kapitel 3 eine Einschätzung zur Umsetzung von Aspekten ab.

Die Umsetzung von AOP verlangt die Änderung eines Programmes „von außen“, es handelt sich um Metaprogrammierung. Daher werden die PHP-Fähigkeiten im Bereich von Metaprogrammierung in Bezug auf CCCs analysiert.

5.1 Reflexion

PHP besitzt die Fähigkeit der Reflexion und Überladung (vgl. Kapitel 2.3.9). Die Kombination aus Reflexion und Überladung bzw. magischen Methoden kann dazu benutzt werden, das Weben von Aspekten in bestehende OOP-Konstrukte zu übernehmen. Denkbar ist eine Implementierung, bei der sämtliche Klassen von einer Mutterklasse ableiten, die die nötigen magischen Methoden bereitstellt. Die Idee ist, sämtliche Objekt-Aufrufe durch die magischen Methoden umzusetzen, also indirekte Aufrufe auszuführen. Dies würde es gestatten, das Einweben von Aspekten in den magischen Methoden zu realisieren.

Das Listing 5.1 und seine Ausführung Listing 5.2 verdeutlichen dies am Beispiel des Setzens einer Klassenvariable. Im Beispiel wird eine nicht-existente Objektvariable „var“ mit einem neuen Wert „testValue“ belegt. Die tatsächlich vorhandene Variable trägt jedoch den Bezeichner „_var“; die nicht-Existenz der Klassenvariablen „var“ führt zur Ausführung der magischen Methode __set() in der

Listing 5.1: Umsetzung des Webens

```
1  magicSet.php:
2  <?php
3
4  abstract class AspectWeaver {
5
6    public function __set($name, $value) {
7      echo "Before, name: $name, value: $value\n";
8      $this->{'_' . $name} = $value;
9      echo "After, set value is: {$this->_var}\n";
10   }
11 }
12
13 /**
14  * @property mixed $var
15  */
16 class ConcreteClass extends AspectWeaver {
17
18   protected $_var;
19 }
20
21 $obj = new ConcreteClass();
22 $obj->var = 'testValue';
```

Listing 5.2: Ausführung (vgl. Listing 5.1)

```
1  root@debian-vt: php magicSet.php
2  Before, name: var, value: testValue
3  After, set value is: testValue
```

Mutterklasse; an genau dieser Stelle ließe sich ein „before“ und „after“-Advice (vgl. Kapitel 2.2) realisieren, verdeutlicht durch die php-echo()-Befehle in Zeile 7 und 9.

Das benutzte Prefix verdeutlicht, dass der Programmierer sich an ein striktes Pattern halten muss, um zum Einen die Instanz der Klasse „ConcreteClass“ korrekt zu verwenden und zum Anderen die Nutzung der JoinPoints einzuhalten.

Bedeutung für CCCs

Eine solche Implementierung hat folgende Nachteile:

- Es wird von dem Programmierer erwartet, sich strikt an ein vorhandenes Pattern für Klassenvariablen und -methoden zu halten. Ein Verletzen des

Patterns führt jedoch nicht dazu, dass das Programm selbst nicht mehr ausführbar wäre, lediglich die Möglichkeit des Einwebens geht (unbemerkt) verloren.

- Vorhandene Code-Completion Unterstützung von IDEs wird wirkungslos (dies lässt sich zumindest für Klassenvariablen durch die Klassen-Dokumentationsart „@property“[1] vermindern).
- Vorhandene Applikationen müssen mit großen Aufwand angepasst werden.
- Der Umweg über die magischen Methoden ist eben nicht für die magischen Methoden selbst möglich - insbesondere der Konstruktor construct()[2] ist hiervon betroffen.
- Die before- und after-Advices sollten nicht innerhalb der Klassenstruktur der betroffenen Klasse implementiert sein, sondern an der aufrufenden Code-Stelle – so dass beispielsweise der Aufruf einer Methode durch einen Aspekt unterbunden werden kann.

Zusammengefasst lässt sich sagen, dass eine Umsetzung des Webens in PHP durch hauseigene Metaprogrammierung zwar möglich ist, aber in der Praxis insbesondere die Nachteile für die Programmierer überwiegen. Diese Art der Umsetzung von Aspekten ist nicht realistisch einsetzbar.

5.2 Traits

Bei Traits handelt es sich um ein PHP-Sprachkonzept, das es ermöglicht, Klassenübergreifende Funktionalität wiederzuverwenden ohne den Ableitungsbaum zu beeinflussen. Von der Idee her erinnern Traits an die Möglichkeit der Mehrfachvererbung; das Diamond-Problem wird jedoch bei Traits umgangen, da eindeutige Regeln bei Namenskonflikten greifen.

Das Listing 5.3 zeigt ein einfaches Beispiel für die Nützlichkeit von Traits. In diesem Beispiel wird das Singleton-Pattern durch einen Trait realisiert (vgl. Zeile

[1] http://manual.phpdoc.org/HTMLSmartyConverter/PHP/phpDocumentor/tutorial_tags.property.pkg.html

[2] http://www.php.net/manual/en/language.oop5.decon.php#object.construct

Listing 5.3: Implementierung des Singleton-Patterns durch Traits

```
1  <?php
2
3  trait SingletonTrait {
4
5    protected static $instance;
6
7    final public static function getInstance() {
8
9      if (!static::$instance) {
10       static::$instance = new static();
11     }
12
13     return static::$instance;
14   }
15
16   final private function __construct() {
17   }
18   ...
19 }
20
21 class SingletonClass {
22   use SingletonTrait;
23 }
```

3). Sämtliche Klassen, die das Singleton-Pattern nach [Gam+10] implementieren, können dieses Trait verwenden anstatt die Funktionalität selbst implementieren zu müssen – die Klasse „SingletonClass“ dieses Beispiel macht genau dies (vgl. Zeile 22).

Traits wurden in PHP mit der Version 5.4 im März 2012 eingeführt[3].

Bedeutung für CCCs

Die Umsetzung des Webens ließe sich mittels Traits analog zu der Umsetzung in Kapitel 2.3.9 realisieren, jedoch ohne den Ableitungsbaum beeinflussen zu müssen, die magischen Methoden ließen sich über Traits realisieren.

Die analoge Umsetzung führt konsequenterweise zu den gleichen Problemen. Eine Realisierung des Webens durch Traits ist nicht praxistauglich.

[3] Version 5.4.0, „Added support for Traits.“ http://php.net/ChangeLog-5.php

Kapitel 6

Stand der Dinge von AOP in PHP

Es existiert bereits eine Zahl von Ansätzen und Implementierungen, um AOP in PHP zu unterstützen. Dieses Kapitel setzt sich mit diesen Projekten auseinander, gibt die aktuellen Entwicklungsstatus wieder und validiert die relevanten Projekte gemäß den in Kapitel 3 aufgestellten Anforderungen.

6.1 Inaktive Projekte

In der Vergangenheit wurde viele Versuche unternommen, AOP in PHP zu ermöglichen. Den hier vorgestellten Projekten ist gemein, dass sie nicht mehr aktiv entwickelt werden.

6.1.1 AOPHP

Das AOPHP Projekt war im Jahr 2005 ([SSC05], [SSB05]) und 2006 aktiv [Aug+06], es ist derzeit aber nicht mehr verfügbar. Die seinerzeit offizielle Webpräsenz `www.aophp.net` ist mittlerweile unter anderer Verwendung im Einsatz. Auf `github.com` findet sich Sourcecode `/uberchicgeekchick/aophp`, der von der Benennung und den Kommentaren her zum AOPHP-Projekt gehört. Jedoch ist dieses Projekt aufgrund fehlender Dokumentation und syntaktischer Fehler im Sourcecode nicht benutzbar; weiter liegt auch hier die letzte Aktivität zwei Jahre zurück, der letzte Commit war am 23.06.2010[1].

[1] `https://github.com/uberchicgeekchick/aophp/commits/master`

Listing 6.1: AOPHP-Anwendung nach [Aug+06]

```
1  <?aoPHP filename="login_a.aoPHP"
2  function invalidLogin($p) {
3    echo "<b>Can Not Login, $p Invalid Password</b><br>";
4  }
5  ?>
6
7  after(): exec(invalidLogin($p)) {
8    $ip = $_SERVER['REMOTE_ADDR'];
9    //This is Where you Log to File or Database
10   echo "<i>$ip failed @ " . time() . "</i><br>";
11 }
```

Der Ansatz für das Weben (vgl. Kapitel 2.2.2) ist ein in Java geschriebener PHP Preprozessor [She06]. Eine AOPHP-Anwendung ist in Listing 6.1 zu sehen. Die Syntax orientiert sich an AspectJ (vgl. Kapitel 4.1). Das Besondere an AOPHP ist die Verwendung des Java-Preprocessors. Dies wird durch die Verwendung „<?aoPHP“ statt „<?php“ erwirkt (vgl. Zeile 1).

AOPHP erfüllt die Minimalanforderungen an eine AOP-PHP-Lösung (vgl. Kapitel 3), das Umsetzen von Aspekten ist möglich.

AOPHP erfüllt jedoch zwei der vier optionalen Anforderungen nicht: Der in Java geschriebene Preprocessor führt dazu, dass die Anforderung der Plattform-Kompatibilität nicht erfüllt ist, da eine lokale Installation von Java nötig ist. Weiter muss aufgrund des geänderten PHP-Datei-Deklaration „<?aoPHP“ die gesamte vorliegende PHP-Software angepasst werden, die Anforderung der Aufwärts-Kompatibilität ist somit nicht erfüllt.

6.1.2 AspectPHP / GAP

Das Projekt „Generic Aspects for PHP (GAP)“ aus dem Jahr 2006 stellt den Nachfolger zu „AspectPHP“ dar; AspectPHP wurde aufgrund der hohen Namensähnlichkeit zu dem weiteren AOP-PHP-Lösungsversuch „PHPAspect“ (vgl. Kapitel 6.1.4) in GAP umbenannt [BK06]. Die letzte Aktivität für dieses Projekt ist das Paper [BK06] aus dem Jahr 2006. AspectPHP/GAP ist aufgrund fehlender Arbeitsressourcen nicht zu Ende entwickelt worden[2], es findet sich keine Veröffentlichung von AspectPHP oder des GAP-Frameworks.

[2] http://sebastian-bergmann.de/archives/589-AspectPHP.html

Listing 6.2: Ein GAP-Aspekt nach [BK06]

```
1  <?php
2  /* @pointcut allInvocations : method(* *->*(..));
3   * @after allInvocations : Logging->log();
4   */
5  class Logging {
6    public function log($joinPoint) {
7      printf(
8        "%s->%s() called %s->%s()\n",
9        $joinPoint
10         ->getSource()
11         ->getDeclaringClass()
12         ->getName(),
13       $joinPoint
14         ->getSource()
15         ->getName(),
16       $joinPoint
17         ->getTarget()
18         ->getDeclaringClass()
19         ->getName(),
20       $joinPoint
21         ->getTarget()
22         ->getName()
23       );
24   }
25 } ?>
```

Das Einhängen in die Programmausführung bei GAP geschieht mittels der PHP-Fähigkeit auf Streams (vgl. Kapitel 2.3.7) zuzugreifen. Laut eigener Beschreibung realisiert GAP das Weben (vgl. Kapitel 2.2.2) in zwei Schritten [BK06]:

1. Die definierten Aspekte werden eingelesen und ihre Pointcuts erfasst. Dazu bedient sich GAP der Reflexion von PHP (vgl. Kapitel 2.3.9).

2. Beim Laden einer PHP-Klasse werden die Aspekte auf den Bytecode der Klasse angewendet.

Listing 6.2 zeigt einen GAP-Aspekt. Dieser Aspekt dient dem Zweck, alle Methoden-Aufrufe innerhalb einer Software zu loggen. Dazu wird durch eine Annotation (vgl. Kapitel 2.3.8) ein Pointcut „allInvocations“ definiert, der einen JoinPoint method(* *->*(..)); referenziert (vgl. Zeile 2). Dieser GAP-Aspekt definiert in einem After-Advice für den Pointcut „allInvocations“ den Aufruf der log()-Methode der „Logging“-Klasse (vgl. Zeile 3). Diese Methode bekommt als

Eingabe den aktiven JoinPoint übergeben und führt das Logging anhand von Information, die der JoinPoint liefert, durch.

GAP gestattet es, CCCs durch Aspekte zu lösen. Damit erfüllt GAP die Minimalanforderungen an eine AOP-PHP-Lösung (vgl. Kapitel 3.1). Weiter erfüllt GAP auch alle optionalen Anforderungen (vgl. Kapitel 3.2), insbesondere aufgrund nicht vorhandener Abhängigkeiten die Anforderung der Plattform-Kompatibilität; damit handelt es sich bei GAP um die einzige AOP-PHP-Lösung überhaupt, die diese optionale Anforderung erfüllt.

Eine Besonderheit an GAP bzw. seinem Vorgänger AspectPHP stellt APDT[3] dar. Dabei handelte es sich um eine Erweiterung der Entwicklungsumgebung Eclipse, die es ermöglicht, den Entwickler bei der Anwendung von AspectPHP zu unterstützen. APDT kennt die AspectPHP-Syntax. Die Spracherkennung erfolgt hierbei mittels des Java-Spracherkennungstools ANTLR[4], Die AspectPHP-Syntax-Defintion findet sich auf `https://apdt.googlecode.com/svn` unter `/trunk/org.phpaspect.apdt.core/Resources/PHPAspectLexer.g`. APDT unterstützt den Entwickler durch Hinweise bei der Verletzung der AspectPHP-Syntax. Weiter unterstützt APDT das Erstellen neuer Aspekte und integriert das (manuelle) Weben direkt in Eclipse.

Die letzte relevante Aktivität in APDT war im Mai 2009[5].

6.1.3 MFAOP

MFAOP[6] (Abkürzung für „MFA Pouw Solutions", „Pouw“ ist der Name des Entwicklers) wurde 2004 und 2005 am Leiden Institute of Advanced Computer Science entwickelt [Pou08], die letzte Version 0.9 datiert vom 15.03.2005[7]. Obwohl MFAOP unter der GNU General Public License (GPL) steht, ist kein freier Download verfügbar. MFAOP wird jedoch auf Nachfrage zur Verfügung gestellt.

MFAOP kombiniert Reflexion (vgl. Kapitel 2.3.9) und die seit Mitte 2006 nicht mehr weiterentwickelte PHP-Erweiterung Runkit[8], um AOP zu realisieren [Pou08]. Die Runkit-Extension wird hierbei eingesetzt, um das Weben zu realisieren.

[3] `http://code.google.com/p/apdt/`
[4] `http://www.antlr.org/`
[5] SVN-Revision 401 `https://apdt.googlecode.com/svn`
[6] `http://www.mfaop.com/`
[7] `http://www.mfaop.com/?pid=30`
[8] `http://pecl.php.net/package/runkit`

Listing 6.3: Anwendung von MFAOP nach [Pou08]

```
<?php
class Example {

  function Foo() {
    echo "Inside foo\n";
  }

  function Bar() {
    echo "Inside bar\n";
    return "Return value of bar";
  }
}

$pointCut = new PointCut();
$pointCut->addJoinPoint('Example', 'Foo');
$pointCut->addJoinPoint('Example', 'Bar');

new Aspect($pointCut, before, 'echo "Before $MethodName";');
new Aspect($pointCut, after, 'echo "After $MethodName";');
new Aspect($pointCut, around, '$Return = "New return value of
    $MethodName";');

echo $example->Foo();

> Before foo
> Inside foo
> After foo
> New return value of foo
```

Listing 6.3 zeigt den Einsatz von MFAOP. MFAOP unterstützt Aspekte gemäß der Anforderungen (vgl. Kapitel 3) – es ist möglich, Aspekte auf Klassen anzuwenden, ohne dass die Klassen selbst manipuliert werden müssen. Eine Schwäche der Realisierung von MFAOP stellt die Voraussetzung der PHP-Erweiterung Runkit dar; damit erfüllt MFAOP nicht die optionale Anforderung der Plattform-Kompatibilität.

6.1.4 Weitere Projekte

Die bisher vorgestellten Projekten lassen sich aus heutiger Sicht gut nachvollziehen, weiter zeichnen sie die ernsthafte Absicht aus, AOP in PHP zu realisieren. Die folgenden Projekte sind weniger gut nachvollziehbar und eher als Konzepte

Listing 6.4: Umsetzung von JoinPoints mit aop.lib.php

```
1 Class Sample4 {
2   function Sample4($Arg1, $Arg2, $Arg3) {
3     Advice::_before($this);
4     print 'Some business logic of Sample4<br />';
5     Advice::_after($this);
6   }
7 }
```

zu bewerten. Die geringe Relevanz dieser Projekte ergibt sich aus den vorhandenen Lücken, so dass keine Bewertung durchgeführt werden kann.

aop.lib.php

Das Projekt „AOP Library for PHP“[9] stammt aus dem Oktober 2005[10]. Der Oktober 2005 ist nicht nur der Monat der ersten Aktivität in diesem Projekt, sondern auch der letzten.

Der Ansatz hinter aop.lib.php ist es, die JoinPoints direkt in den bestehenden Sourcecode manuell zu integrieren. Wie im Listing 6.4 erkennbar ist, werden das Before- und After-Advice in die erste bzw. letzte Zeile eines Methodenkörpers eingepflegt (vgl. Zeile 3 und 5). Dies Vorgehen führt dazu, dass der Sourcecode einer Methode trotz des Einsatzes von AOP von einem weiteren Belang betroffen ist. Weiter werden die Aspekte nicht ausschließlich „von außen“ angewendet, für die aop.lib.php-Unterstützung muss bestehender Sourcecode manipuliert werden. Daher erfüllt dieses Projekt nicht die Minimal-Anforderung (vgl. Kapitel 3.1) an eine AOP-PHP-Lösung.

PHPAspect

Das Projekt PHPAspect[11], erschienen im Februar 2005 und verfolgte den Ansatz, mit einem eigenen Compiler die PHPAspect-Aspektsprache mit PHP zu vereinen. Ein entsprechender PHP-Patch liegt für die Version 4.3.10. vor – ist aus heutiger

[9] http://www.phpclasses.org/package/2633-PHP-Implement-Aspect-Oriented-Programming-at-run-time.html

[10] http://www.phpclasses.org/discuss/package/2633/thread/1/

[11] http://www.cs.toronto.edu/~yijun/aspectPHP/

Listing 6.5: Webpunkt-Realisierung in Seasar.PHP

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE components PUBLIC "-//SEASAR//DTD S2Container//EN"
"http://www.seasar.org/dtd/components21.dtd">
<components>
    <component name="traceInterceptor" class="TraceInterceptor"
        />
    <component class="Date">
        <aspect pointcut="getTime">
            traceInterceptor
        </aspect>
    </component>
</components>
```

Sicht also stark veraltet, diese PHP-Version erschien am 14.12.2004[12]. Zum Zeitpunkt der Veröffentlichung von PHPAspect lag PHP in der Version 5.0.3 vor, die erste Beta von PHP5 erschien am 29.06.2003[13].

Seasar.PHP

Das Seasar-Projekt verfolgt das Ziel, DI zur Verfügung zu stellen. Die Realisierung für PHP[14] verwendet zu diesem Zweck AOP. Die letzte Aktivität war am 09. Juli 2006.

Seasar.php verwendet ein eigenes XML-Format, um Aspekte zu deklarieren (vgl. Listing 6.5). Die Integration wird durch PHP selbst realisiert, der neu erzeugte Code durch die PHP-Funktion eval()[15] ausgeführt [She06]. Die Idee der Kopplung von Aspekten an Pointcuts in einer XML-Datei bringt den Nachteil, dass Aspekte sich nicht lokal befinden. Stattdessen ist der Webpunkt in einer XML-Datei definiert, das Advice an anderer Stelle. Dies erschwert die Verständlichkeit in der Anwendung.

[12] http://www.php.net/ChangeLog-4.php
[13] http://www.php.net/ChangeLog-5.php
[14] http://www.seasar.org/en/php5/index.html
[15] http://php.net/manual/de/function.eval.php

6.2 Aktive Projekte

Die bisher vorgestellten Projekte werden nicht mehr aktiv entwickelt und sind aufgrund des hohen zeitlichen Abstands zur letzten Aktivität in aktuellen PHP-Projekten nicht einsetzbar. Die folgenden zwei Projekte befinden sich im Gegensatz dazu in aktiver Entwicklung.

6.2.1 Flow3

Bei Flow3[16] handelt es sich um ein PHP-Applikations-Framework, auf dem das PHP-CMS Typo3[17] aufbaut. Es handelt sich hierbei also nicht um eine explizite AOP-PHP-Lösung; Flow3 ist ein vollständiges Framework, das u. a. AOP in PHP ermöglicht.

Listing 6.6 zeigt die Anwendung von Aspekten in Flow3, dieses Beispiel entstammt der Flow3-Dokumentation[18]. Bei Aspekten handelt es sich in diesem Framework um speziell annotierte PHP-Klassen (vgl. Kapitel 2.3.8). Diese Metadaten definieren sowohl die Aspekte als auch die Webpunkte und Advices (vgl. Kapitel 2.2).

Im Listing ist eine Klasse „Forum“ (vgl. Zeile 1) mit einer Methode zum Löschen eines Forum-Posts deletePost() (vgl. Zeile 8) sowie ein Flow3-Aspekt „LoggingAspect“ (vgl. Zeile 16) zu sehen. Der Aspekt übernimmt in diesem Beispiel die Aufgabe, das Löschen eines Posts zu loggen. Mittels „@FLOW3\Before“ (vgl. Zeile 28) wird der Bezug zwischen Aspekt und JoinPoint (vgl. Kapitel 2.2) hergestellt, so dass vor dem Ausführen der „deletePost“-Methode des Forums das Logging durchgeführt wird – es handelt sich also um einen Before-Advice.

Das Weben von Aspekten in Flow3 wird mittels PHP-Reflexion (vgl. Kapitel 2.3.9) realisiert, es ist weder ein eigener PHP-Preprocessor noch eine PHP-Erweiterung nötig [Buc08]. Mittels der PHP-Reflexion-API werden in Flow3 Kindklassen der durch Aspekte betroffenen PHP-Klassen erzeugt, die während der Programmausführung als Proxy [Gam+10] statt der eigentlichen Klassen verwendet werden und so die JoinPoints realisieren.

[16] http://flow3.typo3.org/
[17] http://typo3.org/
[18] http://flow3.typo3.org/documentation/guide/partiii/aspectorientedprogramming.html

Listing 6.6: Aspekte in Flow3

```
class Forum {
  /**
   * Delete a forum post
   *
   * @param \Examples\Forum\Domain\Model\Post $post
   * @return void
   */
  public function deletePost(Post $post) {
    $this->posts->remove($post);
  }
}

/**
 * @FLOW3\Aspect
 */
class LoggingAspect {

  /**
   * @FLOW3\Inject
   * @var \Examples\Forum\Logger\ApplicationLoggerInterface
   */
  protected $applicationLogger;

  /**
   * Log a message if a post is deleted
   *
   * @param \TYPO3\FLOW3\AOP\JoinPointInterface $joinPoint
   * @FLOW3\Before("method(Examples\Forum\Domain\Model\Forum->
       deletePost())")
   * @return void
   */
  public function logDeletePost(\TYPO3\FLOW3\AOP\
      JoinPointInterface $joinPoint) {
    $post = $joinPoint->getMethodArgument('post');
    $this->applicationLogger->log('Removing post ' . $post->
        getTitle(), LOG_INFO);
  }
}
```

Flow3 unterstützt das Umsetzen von Aspekten und erfüllt damit die Minimal-Anforderung an einen AOP-PHP-Lösung (vgl. Kapitel 3.1). Die Möglichkeit, Aspekte zu nutzen, ist jedoch an das Flow3-Framework gekoppelt, die Aspekt-Unterstützung ist nicht ohne das Framework nutzbar. Wegen dieser harten Kopplung

Listing 6.7: AOP-Anwendung nach [Pec]

```
<?php
class MyServices {
  public function doAdminStuff1() {
    ...
  }
  ...
}

function adviceForDoAdmin () {
  if ((! isset($_SESSION['user_type'])) || ($_SESSION['user_type
      '] !== 'admin')) {
    throw new Exception('Sorry, you should be an admin to do
        this');
  }
}

aop_add_before('MyServices->doAdmin*()', 'adviceForDoAdmin');
```

an das Framework erfüllt Flow3-AOP nicht die optionale Anforderung der Plattform-Kompatibilität (vgl. Kapitel 3.2).

6.2.2 AOP

Bei AOP [Pec] handelt es sich um einen jungen AOP-PHP-Lösungsansatz in Form einer PHP-Erweiterung, der sich derzeit im Entwicklungsstadium befindet. Die vorliegende Version ist 0.1-beta vom 05.07.2012[19].

Das Listing 6.7 zeigt die Anwendung von AOP. AOP ermöglicht es, Aspekte zu definieren und verschiedene Advices (vgl. Kapitel 2.2) anzuwenden. Damit erfüllt AOP die Minimal-Anforderungen an eine AOP-PHP-Lösung (vgl. Kapitel 3.1). Ein Nachteil von AOP ist es jedoch, dass diese Lösung als PHP-Erweiterung realisiert wird, die zusätzlich in Zielumgebungen installiert werden muss. Damit erfüllt AOP nicht die optionale Anforderung der Plattform-Kompatibilität (vgl. Kapitel 3.2).

Tabelle 6.1: Übersicht AOP-PHP-Lösungen

Name	Aktivität	Minimal	Optional
AOPHP	2005 - 2006	✓	2/4
AspectPHP / GAP	2005 - 2006	✓	4/4[20]
MFAOP	2004 - 2005	✓	3/4
Flow3	2011 - 2012	✓	3/4
AOP	2012	✓	3/4

6.3 Zusammenfassung

Dieses Kapitel gibt eine Übersicht über die Ansätze, Aspekt-Orientiertung in PHP umzusetzen. Die Anzahl von vielen unabhängigen Projekten zeigt die Relevanz des Themas.

Unter den hier vorgestellten Projekten ist das eingestellte GAP das ambitionierteste. GAP erfüllte nicht nur als einziger Ansatz alle Anforderungen (vgl. Kapitel 3), sondern unterstützt mit APDT auch den Entwickler in der täglichen Anwendung von Aspekten.

Die zwei derzeit aktiven Umsetzungen sind beide mit Nachteilen behaftet, wodurch der Einsatz erschwert wird. Die Umsetzung von AOP in Flow3 ist vielversprechend, jedoch ist die Migration eines Produktiv-Systems auf Flow3 nur um AOP-Unterstützung zu erhalten unverhältnismäßig; gleichzeitig würde dadurch auch eine starke Abhängigkeit zum Framework geschaffen. Auch möchten neue Projekte nicht unbedingt dieses Framework nutzen. Auch das junge Projekt „AOP“ (vgl. Kapitel 6.2.2) ist vielversprechend, jedoch noch in der Entwicklung und dadurch für Produktivsysteme nicht empfehlenswert. Ähnlich wie Flow3 kann AOP nicht die optionale Anforderung der Plattform-Kompatibilität erfüllen, es ist die Installation der PHP-Erweiterung „AOP“ nötig ist. Dies ist jedoch mit weniger Aufwand verbunden, als die alternative Adaption des Flow3-Frameworks.

Die Tabelle 6.1 fasst die Ergebnisse der Analyse bestehender und vergangener AOP-PHP-Lösungen zusammen.

[19] http://pecl.php.net/package/AOP

[20] Wobei GAP die Werkzeug-Kompatibilität nicht nur theoretisch ermöglicht, sondern tatsächlich realisiert

Kapitel 7

Yet another PHP Aspect Framework (YAPAF)

Der Stand der Dinge in Kapitel 6 hat gezeigt, dass es derzeit keine AOP-PHP-Lösung gibt, die alle Anforderungen an eine AOP-PHP-Lösung erfüllt (vgl. Kapitel 3). Daher ist es Teil dieser Diplomarbeit, ein Konzept für die mögliche Implementierung gemäß der Anforderungen aus Kapitel 3 vorzustellen. Das Projekt trägt den Namen YAPAF, die Abkürzung für „Yet another PHP Aspect Framework“. Dieses Kapitel zeigt den Quelltext des Programms nur auszugsweise. YAPAF inklusive der zugehörigen Unit-Tests, die u. a. die Funktionalität dokumentieren, finden sich im Software-Repoitory der Universität Koblenz unter `https://svn.uni-koblenz.de/softlang/main/students/markuss/YAPAF`.

Die Umsetzung gemäß der Anforderungen aus Kapitel 3 sowie die Validierung der Funktionalität stellen hierbei die Herausforderungen dar. In Kapitel 7.6.1 findet sich mit der Realisierung von Dependency Injection eine mögliche Anwendung von AOP.

7.1 Konzeptioneller Überblick

Damit alle Anforderungen aus Kapitel 3 erfüllt werden können, kann lediglich die PHP-Standardfunktionalität eingesetzt werden, um das Weben umzusetzen. In Kapitel 2.3.3 ist die normale PHP-Ausführung beschrieben - bei der Umsetzung des Webens ist es notwendig, ohne weitere Hilfsmittel die Aspekte in die

Programmausführung zu integrieren. Dies schließt für die Umsetzung den Einsatz von PHP-Interpretern Dritter aus, wie etwa „HipHop for PHP“[1] von Facebook oder „Quercus“[2]. Diese Werkzeuge böten die Möglichkeiten, in den Ausführungsprozess von PHP einzugreifen - verletzen jedoch die Anforderung der Plattform-Kompatibilität.

Das Kapitel 2.3.7 stellt die PHP-Funktionalität vor, unter Angabe eines Protokolls eigene Handler beim Einlesen von Streams zu verwenden. Diese Handler müssen nicht zwangsläufig den Original-Stream liefern, sondern können theoretisch komplett andere Daten zurückgeben.

Die Idee bei der Umsetzung des Webens hinter YAPAF beruht im Wesentlichen darauf, beim Einlesen von PHP-Dateien bzw. der enthaltenen Klasse diesen Stream zu analysieren und, falls nötig, Aspekte einzuweben. Das Einweben erfolgt dabei also während der Ausführung des PHP-Programmes. In Bezug auf den Zeitpunkt während der PHP-Ausführung (vgl. Kapitel 2.3.3) bedeutet dies, dass noch vor der lexikalischen Analyse eingegriffen wird, es handelt sich daher um Compile-time Weaving (vgl. Kapitel 2.2.2).

7.2 Einbinden der Umsetzung von Aspekten

Eine wesentliche Anforderung an ein AOP-PHP-Lösung ist es, dass die Aspekte automatisch angewendet werden - auch auf Klassen, die nach dem Erstellen eines Aspektes erstellt wurden, aber den Bedingungen vorhandener JoinPoints genügen.

Das vorherige Kapitel 7.1 zeigt eine Möglichkeit, mit PHP-Mitteln Aspekte anzuwenden. Diese Möglichkeit muss nun auch auf jede mögliche Klasse angewendet werden, damit die Aspekte bzw. ihre Advices angewendet werden. Das Kapitel 2.3.5 „Autoloading“ diskutiert den PHP-Autoloader. Diese PHP-Funktionalität bietet die Möglichkeit, Klassennamen auf Dateien aufzulösen und diese Dateien einzubinden. Der PHP-Autoloader bietet genau die Möglichkeit, die in dem Kontext des Webens gesucht wird; er bietet eine zentrale Stelle, um auf sämtliche Klasseneinbindungen einzuwirken.

[1] https://github.com/facebook/hiphop-php/
[2] http://quercus.caucho.com/

7.3 Implementierung

Die Verbindung des stream_wrapper_register() und des Autoloaders bieten die Möglichkeit, Compile-Time Weaving in PHP zu realisieren. YAPAF nutzt genau diese Möglichkeit. In einigen wenigen PHP-Klassen wird die Funktionalität des Webens umgesetzt.

Aop.php Eine Proxy-Klasse, die der einfachen Benutzbarkeit von YAPAF im eigenen Projekt dient.

libs/StreamWrapper.php Die Klasse Aop_StreamWrapper wird bei stream_per−per_register () angegeben, um Zugriff auf den Stream, konkret den einzulesende Sourcecode einer zu ladenden Klasse, zu erhalten.

libs/Weaver.php Die Klasse Aop_Weaver führt das tatsächliche Einweben von Aspekten in Klassen durch.

libs/annotations.php Enthält die Annotationen, durch die YAPAF JoinPoints umsetzt.

Die folgende Unterkapitel beschreiben im Detail die Aufgaben der jeweils an YAPAF beteiligten Dateien.

7.3.1 Aop.php

Das Listing 7.1 zeigt den Inhalt der Datei Aop.php, in der die PHP-Klasse „Aop“ enthalten ist. Diese Klasse dient der einfachen Verwendbarkeit von YAPAF, nach außen sichtbar ist ausschließlich die statische Methode init () (vgl. Zeile 4). Diese erwartet als Eingabe eine Array von Aspekten, die in Objektform (und nicht etwa als Dateipfade) vorliegen. Intern reicht die Klasse diese Aspekte an den YAPAF-Aspect_Weaver „Aop_Weaver“ durch (vgl. Kapitel 7.3.3). Weiter registriert diese Methode in Zeile 7 das eigens-definiert Protokoll „aop“ in Verbindung mit dem YAPAF-StreamWrapper „Aop_StreamWrapper“ (vgl. Kapitel 7.3.2), so dass dieser StreamWrapper beim nötigen Inkludieren noch nicht verwendeter PHP-Klassen angesprochen wird. Zuletzt wird der eigene AOP-Autoloader (vgl. Kapitel 2.3.5) aop_autoload() registriert (Zeile 8), der intern dafür sorgt, dass beim Einbinden der Datei, die eine benötigte PHP-Klasse enthält, das eigene „aop“-Protokoll verwendet wird, so dass der YAPAF-StreamWrapper beim Einbinden Verwendung findet.

Listing 7.1: Inhalt von Aop.php (Auszüge)

```
class Aop {
  const PROTOCOLL = 'aop';

  public static function init(array $aspects = array()) {
    Aop_Weaver::setAspects($aspects);

    stream_wrapper_register(self::PROTOCOLL, 'Aop_StreamWrapper'
        );
    spl_autoload_register(array('Aop', 'aop_autload'));
  }

  private static function aop_autload($class) {
    ...
    require self::PROTOCOLL . ':///' . $pathToFile;
  }
  ...
}
```

Listing 7.2: Inhalt von libs/StreamWrapper.php (Auszüge)

```
class Aop_StreamWrapper {
  ...
  protected $source;
  ...
  public function stream_open($path, $mode, $options, &
      $opened_path) {
    ...
    $proc = new Aop_Weaver($path);
    $this->source = $proc->weave();
    ...
  }
  ...
}
```

7.3.2 libs/StreamWrapper.php

Das Listing 7.2 zeigt die PHP-Klasse Aop_StreamWrapper. Diese Klasse wird durch den YAPAF-Autoloader beim Einbinden einer PHP-Klasse aktiviert, die Zeilen einer zu inkludierenden Datei werden von diesem StreamWrapper verarbeitet. Die Funktion stream_open() zeichnet sich für das Erfassen des Inhaltes zu einem Dateipfad verantwortlich. An genau dieser Stelle ruft der StreamWrapper den YAPAF-Aspect_Weaver auf, der wiederum das Einweben von Aspekten

Listing 7.3: Inhalt von libs/Weaver.php (Auszüge)

```
class Aop_Weaver extends PHP_Token_Stream {
  ...
  protected static $_aspects = array();
  ...
  public function weave() {
    foreach (self::$_aspects as $aspect) {
      $aspectReflected = new ReflectionAnnotatedClass($aspect);
      foreach ($aspectReflected->getMethods() as
          $aspectMethodReflected) {
        $aspectsJoinpoints = $aspectMethodReflected->
            getAllAnnotations('Call');
        foreach ($aspectsJoinpoints as $joinpoint) {
          ...
        }
        ...
      }
      ...
    }
    ...
  }
  ...
}
```

übernimmt und den so manipulierten Sourcecode statt des in der Datei vorhanden Sourcecodes liefert.

7.3.3 libs/Weaver.php

Bei der Klasse Aop_Weaver handelt es sich um das Herzstück von YAPAF, der Inhalt ist sichtbar im Listing 7.3. Diese Klasse übernimmt das Einweben von Aspekten in PHP-Klassen. Durch die in diesem Kapitel beschriebenen Maßnahmen kennt der Weaver die Aspekte, die angewendet werden sollen. Weiter wird der Weber automatisch ausgeführt, wenn eine PHP-Klasse aufgerufen wird. Der vom Weaver bereitgestellte Sourcecode ist derjenige, der in der weiteren Programmausführung für die angeforderte Klasse verwendet wird.

7.3.4 libs/annotations.php

Die JoinPoints in YAPAF werden durch Annotationen realisiert (vgl. Kapitel 7.4). Diese Realisierungen finden sich in der Datei „libs/annotations.php“, Auszü-

Listing 7.4: Inhalt von libs/annotations.php (Auszüge)

```
abstract class ClassPropertyJoinPoint extends Annotation {
  ...
  public $class;
  ...
  public $method;
  ...
  public $var;
  ...
  public function getMatchingTokens(array $tokens) {
    ...
    return $this->_getMatchingTokens($tokens);
  }
  ...
  abstract protected function _getMatchingTokens(array $tokens);
}

class After extends ClassPropertyJoinPoint {
  /**
   * Return tokens of the finishing ";" for machting "after"-
     statements.
   * For example, at "$this->someObject->doSth($var1, $var);"
     the token for ";" is returned;
   *
   * @param array $tokens
   * @return multitype:PHP_Token_SEMICOLON
   */
  protected function _getMatchingTokens(array $tokens) {
    $matches = array();

    $actClass = '';
    foreach ($tokens as $key => $token) {
      if ($token instanceof PHP_Token_Class) {
        $actClass = $token->getName();
      }

      if ($actClass !== $this->class &&
      $token instanceof PHP_Token_STRING &&
      (String)$token === $this->method) {

        do {
          $token = $tokens[$key++];
          if ($token instanceof PHP_Token_SEMICOLON) {
            $matches[] = $token;
            continue 2;
          }
        } while ($key < count($tokens));
      }
    }

    return $matches;
  }
}
```

ge sind sichtbar im Listing 7.4, der beispielhaften Erklärung dient der „After"-JoinPoint (vgl. Zeile 17ff.). Um Annotationen in PHP benutzen zu können, verwendet YAPAF die PHP-library „addendum", die in Kapitel 2.3.8 vorgestellt wurde.

YAPAF gestattet JoinPoints, die sich auf spezielle Klasseneigenschaften beziehen, im Konkreten auf Klassenvariablen oder -methoden. Eine Mutterklasse für JoinPoints „ClassPropertyJoinPoint" stellt die Basisfunktionalität für JoinPoints zu Verfügung und definiert die Schnittstelle zum Ermitteln passender Anwendungsstellen in betroffenen PHP-Klassen. Dieses Klasse selbst ist ein Kindklasse der addendum-Klasse „Annotation". Ein Joinpoint ist also genau eine Annotation. Die definierte Schnittstelle getMatchingTokens() erwartet als Eingabe eine Liste von PHP-Opcodes (vgl. Kapitel 2.3.3, „Lexikalische Analyse") und liefert eine Untermenge dieser Liste mit passenden Treffern, die den Bedingungen dieses JoinPoints genügen.

Die Klasse „After" im Listing 7.4 realisiert einen After-JoinPoint, also einen JoinPoint, der nach dem Ausführen einer bestimmten Bedingung greift, beispielsweise nach der Ausführung einer Klassenmethode. Dieser JoinPoint hat durch die Ableitung von der Mutterklasse „ClassPropertyJoinPoint" Zugriff auf die Information, für welche Klasse und welche Klassenvariable oder -methode er zuständig ist. Die von der Mutterklasse vorgeschriebene Methode protected function _getMatchingTokens() erhält als Eingabe die Repräsentation von PHP-Sourcecode in Opcode-Form. Die „After"-Klasse analysiert diese Opcode und liefert, falls sich Treffer für diesen JoinPoint in den Token finden, definierte Token zurück, die als Einstiegspunkte für das Weben genutzt werden.

7.4 Anwenden von Aspekten

Listing 7.5 zeigt eine gewöhnliche PHP-Klasse sowie einen YAPAF-Aspekt. Die Klasse „TestClass" beinhaltet die Methode square() (vgl. Zeile 17), die zu einem übergebenen Input das mathematische Quadrat liefert.

Der YAPAF-Aspekt „TestCallAspect" enthält einen „Call"-Advice, also einen Aspekt, der bei einem definierten Aufruf greifen soll (vgl. Zeile 4). Die Natur dieses Advice in Form einer Annotation wird in Kapitel 7.3.4 beschrieben. Dieser konkrete Call-Advice besagt, dass der Aspekt beim Aufruf der Methode square()

Listing 7.5: Ein YAPAF-Aspekte

```
class TestCallAspect {
  ...
  /**
   * @Call(class="TestClass", method="square");
   */
  public function directlyReturnSquareInput() {
    return func_get_arg(0);
  }
}

class TestClass {
  ...
  /**
   * @param   int $i
   * @return  int Squared input.
   */
  public function square($i) {
    return $i * $i;
  }
}
```

Listing 7.6: Anwenden eines YAPAF-Aspekts

```
require_once('aspects/TestCallAspect.php');
Aop::init(array(new TestCallAspect()));

$obj = new TestClass();
echo $obj->square(5);
```

in der „TestClass“ greifen soll. Für diesen Fall soll direkt der übergebene Eingabeparameter „$i“ von der Methode zurückgegeben werden.

Das Listing 7.6 zeigt die Anwendung des YAPAF-Aspektes „TestCallAspect“ aus Listing 7.5.

Dem YAPAF-Framwork wird der Aspekt über die statische Methode init() der Aop-Klasse (vgl. Kapitel 7.3.1) übergeben. Anschließend wird die Klasse „TestClass“ normal ausgeführt.

Wie durch den Aspekt zu erwarten ist, gibt die Methode square() tatsächlich den Eingabewert „5“ direkt zurück. Das normale Verhalten der Methode wäre die Rückgabe des Wertes „25“.

Das Listing 7.7 zeigt den Effekt von YAPAF. Zu sehen ist der Quelltext der „TestClass“-Klasse nach dem Weben. Wie zu sehen ist, wurde die Funktionalität

Listing 7.7: Durch einen YAPAF-Aspekt manipulierter Sourcecode

```
1 class TestClass {
2   ...
3   public function square($i) {
4     return func_get_arg(0);
5     return $i * $i;
6   }
7   ...
8 }
```

des „TestClass“-Aspektes direkt in die erste Zeile der square()-Methode eingefügt (vgl. Zeile 4).

7.5 Performance

Das Einweben von YAPAF erfolgt zur Laufzeit. Dies entspricht der Natur von PHP, das in seiner Grundeigenschaft das vollständige Parsen des Sourcecodes bei jeder Programmausführung anwendet (vgl. Kapitel 2.3.3). Das Einweben verlängert also zwangsweise die Ausführungszeit eines PHP-Pogrammes.

Aufgrund des möglichen und üblichen Opcode-Cachings von PHP (vgl. Kapitel 2.3.4) sind Performance-Überlegungen trotzdem nicht Teil des YAPAF-Konzeptes. Wie in diesem Kapitel erklärt wird (vgl. Kapitel 7.4), entspricht der PHP-Sourcecode nach dem Weben dem von Aspekten durchzogenen Sourcecode ohne Anwendung von Aspekten. Unter Verwendung von Opcode-Caching ist also kein Mehraufwand durch das Weben selbst vorhanden.

7.6 Validierung

7.6.1 Funktionalität

Listing 7.8: UnitTest zur Validierung der Funktionalität von YAPAF

```
1 #tests/classes/DITestClass.php:
2 class DITestClass {
3
4   /**
5    * @var stdClass
```

```
6     */
7   private $di_var;
8   ...
9   public function getVar() {
10    ...
11    return $this->di_var;
12  }
13 }
14
15 #tests/aspects/DI_Aspect.php:
16 class DI_Aspect {
17
18   /**
19    * @Before(class="DITestClass", var="di_var");
20    */
21   public function injectDependency() {
22     $varReflected = new ReflectionAnnotatedProperty('DITestClass
         ', 'di_var');
23      if (preg_match('/@var\s+([^\s]+)/', $varReflected->
          getDocComment(), $matches)) {
24        list(, $type) = $matches;
25        $this->di_var = new $type();
26      }
27   }
28 }
29
30 #tests/DI_Test.php:
31 function usual_autoload($class) {
32   include $class . '.php';
33 }
34
35 class DI_Test extends PHPUnit_Framework_TestCase {
36
37   public function setUp() {
38     $path = __DIR__ . DIRECTORY_SEPARATOR . 'classes';
39     set_include_path(get_include_path() . PATH_SEPARATOR . $path
         );
40   }
41
```

```
42    /**
43     * @test
44     */
45    public function di_disapprove() {
46      spl_autoload_register('usual_autoload');
47
48      $obj = new DITestClass();
49      $this->assertNull($obj->getVar());
50    }
51
52    /**
53     * @test
54     * @depends di_disapprove
55     */
56    public function di() {
57      require_once('aspects/DI_Aspect.php');
58      Aop::init(array(new DI_Aspect()));
59
60      $obj = new DITestClass();
61      $this->assertInstanceOf('stdClass', $obj->getVar());
62    }
63  }
64
65  > phpunit DI_Test.php
66  > [...]
67  > ..
68  > Time: 1 second, Memory: 3.50Mb
69  >
70  > OK (2 tests, 2 assertions)
```

Die Validierung der Funktionalität erfolgt in YAPAF mittels UnitTests unter Verwendung des PHPUnit-Frameworks (vgl. Kapitel 2.3.10). Diese finden sich im in der Einführung dieses Kapitels angegeben Repository unterhalb des Ordners „tests“. Beispielhaft soll hier das Testen eines Aspektes vorgestellt werden, der Dependency Injection realisiert.

Bei DI handelt es sich um eine Programmiertechnik zum Reduzieren von Abhängigkeiten zwischen Komponenten, so dass die Wiederverwendbarkeit dieser Module erhöht wird [CI05]. Ein Software-Modul muss für das korrekte Arbeiten sicherstellen, dass alle benötigen Ressourcen wie etwa andere Module verfügbar

sind. Unter Verwendung von DI wird die Auflösung der Abhängigkeiten nicht durch das Modul selbst umgesetzt, sondern die nötigen Ressourcen werden extern erstellt und von außen in das Modul eingefügt. Dies bietet u. a. den Vorteil, dass das Original-Modul nicht den Belang der Abhängigkeiten-Auflösung enthalten muss. Ein weiterer Vorteil ist, dass das einfügende Modul sich anders verhalten kann: In einer Testumgebung kann bspw. statt des Originals ein Mockup erzeugt und eingefügt werden.

PHP selbst bietet nicht die Funktionalität der DI. Mit den Frameworks „Zend-Framework“[3] und „Flow3“[4] ermöglichen zwei populäre PHP-Frameworks DI.

DI kann im AOP-Sinne als Before-Advice verstanden werden: Vor dem Zugriff auf eine Abhängigkeit wird durch einen Aspekt die benötigte Resource erstellt und an betroffener Stelle eingefügt.

Listing 7.8 zeigt eine PHP-Klasse „DITestClass“(vgl. Zeile 2ff.) und einen Aspekt „DI_Aspect“ (vgl. Zeile 16ff.), der sich auf die „DITestClass“ bezieht. Das Validieren der Funktionalität des Aspekte findet sich im UnitTest „DI_Test“ (vgl. Zeile 30ff.). Die erfolgreiche Ausführung des Tests ist im unteren Teil des Listings zu sehen (vgl. Zeile 65ff.). Vom reinen Programmausführungszeitpunkt aus betrachtet greift dieser Aspekt vor dem Zugriff auf die Variable „di_var“ der Klasse „DITestClass“.

Das Setup des Tests simuliert eine normale PHP-Applikation mit gesetztem include_path (vgl. Kapitel 2.3.6). Im UnitTest liegen zwei Tests vor: di_disapprove() sowie di(). Der erste Test, di_disapprove(), stellt das normale Verhalten der Klasse „DITestClass“ sicher. Es wird validiert, dass die Klassenvariable „di_var“ nicht initialisiert ist und dementsprechend den „null“-Wert (vgl. Kapitel 2.3.2) enthält. Der zweite Test di() wendet den Aspekt „DI_Aspect“ an (vgl. Kapitel 7.4). Es wird validiert, dass sich das Verhalten der „DITestClass“ geändert hat: In diesem Szenario ist die Variable „di_var“ der Klasse „DITestClass“ mit einem erwarteten Typen initialisiert, DI wurde also durchgeführt. Konkret ist „di_var“ vom Typ „stdClass“ [Php], die Angabe in der Annotation „@var“ (vgl. Zeile 5) wurde vom „DI_Aspect“ zur Typbestimmung verwendet (vgl. Zeile 23).

Eine Besonderheit bei diesem Test stellt die im Kapitel 2.3.10 vorgestellte Prozessisolation von PHPUnit dar. Der Test „DI_Test“ bindet die Klasse „DITestClass“ in jedem der zwei Tests erneut ein. Ohne Prozessisolation würde dies zu

[3] Ab Version 2 http://framework.zend.com/
[4] http://flow3.typo3.org/

einem Fehler im zweiten Test aufgrund der bereits erfolgten Einbindung der Klasse „DITestClass“ im ersten Test führen.

7.6.2 Anforderung

Das bisherige Kapitel 7 hat die Umsetzung und Anwendung YAPAF diskutiert. YAPAF soll eine AOP-PHP-Lösung sein und muss sich in diesem Unterkapitel an den Anforderungen an solch eine Lösung aus Kapitel 3 messen.

Die minimale Anforderung an eine AOP-PHP-Lösung ist es, Aspekte gemäß Kapitel 2.2 zu unterstützen. Diesem Anspruch wird YAPAF gerecht. Mit YAPAF ist es möglich, Aspekte zu definieren und anzuwenden. Dies belegen die zwei in diesem Kapitel vorgestellten YAPAF-Umsetzungen – zum Einen die direkte Rückgabe einer Funktionseingabe, zum Anderen die Umsetzung von DI.

Neben der minimalen Anforderung gibt es noch die optionalen Anforderungen:

Aufwärts-Kompatibilität Die Herausforderung der Aufwärts-Kompatibilität kann YAPAF nur unter Einhaltung eines benutzten PHP-Autoloaders (vgl. Kapitel 2.3.5) erfüllen. YAPAF benutzt zum Weben das eigene „aop“-Protokoll beim Einlesen von (PHP-)Dateien. Das Nutzen dieses Protokolls wird im YAPAF-Autoloader realisiert. Sollte eine PHP-Software keinen Autoloader verwenden, so würde auch nicht das „aop“-Protokoll angewendet werden können.

Plattform-Kompatibilität YAPAF verwendet ausschließlich Sprachkonstrukte, die in der Basis von PHP enthalten sind, und ist daher in jeder Standard-PHP-Umgebung ausführbar.

Werkzeug-Kompatibilität IDEs können theoretisch YAPAF unterstützen. Dies belegt das inaktive AOP-PHP-Projekt „AspectPHP“ (vgl. Kapitel 6.1.2) mit der damaligen Eclipse-Unterstützung in Form von APDT.

Programmierer-Kompatibilität YAPAF ist in PHP implementiert und verwendet größtenteils vorhandene PHP-Sprachkonstrukte, die dem PHP-Entwickler bekannt sind.

7.7 Zusammenfassung

Mit YAPAF präsentiert diese Diplomarbeit eine leichtgewichtige, einfach zu verstehende und zu verwendende AOP-Lösung.

Das Framework besteht lediglich aus einigen wenigen Klassen und verwendet ausschließlich Sprachmittel der Sprache PHP selbst. Ergänzt wird das Framework durch vorhandene Tests, die gleichzeitig die Funktionalität und den Einsatz dokumentieren. Dies führt dazu, dass PHP-Entwickler YAPAF einfach nutzen können und sich das Framework leicht in ein bestehendes PHP-Projekt integrieren lässt, so dass sich die Vorteile von Aspekt-Orientierung in einem PHP-Projekt einsetzen lassen und CCCs unterstützt werden können.

Für eine breite Unterstützung mangelt es dem jetzigen YAPAF an theoretisch möglicher IDE-Unterstützung, die Verzerrung von Programmlogik durch Aspekte gestaltet sich dem Entwickler nicht transparent. Weiter kann YAPAF noch um Möglichkeiten der AOP erweitert werden, beispielsweise weitere Join-Points, Pointcuts und die Unterstützung für eigene Aspekt-Zustände.

Kapitel 8

Zusammenfassung und Ausblick

Diese Diplomarbeit hatte sich zum Ziel gesetzt, das AOP-Paradigma im PHP-Umfeld zu untersuchen und eine AOP-PHP-Lösung zu präsentieren.

Dazu wurden die Anforderungen an eine AOP-PHP-Lösung in Kapitel 3 definiert, so dass AOP-PHP-Umsetzungen anhand dieser Anforderungen bewertet werden konnten.

In Kapitel 4 wurde die bei AOP zu Grunde liegende Herausforderung der CCCs außerhalb der PHP-Umgebung untersucht. Es wurde festgestellt, dass mit AspectJ eine Referenz-Implementierung von AOP existiert, die als Vorlage für weitere AOP-Lösungen dienen kann. AspectJ bietet zahlreiche Advices, eigene Aspektzustände, unterstützt das Weben auf mehrere Arten und präsentiert sich dem Entwickler transparent. Die Präsentation der Metaprogrammierung in Ruby zeigt, dass CCCs (teilweise) auch durch reine Sprachunterstützung gelöst werden können und so der negative Einfluss von CCCs auf die Software-Struktur gemildert werden kann, auch ohne die Anwendung von AOP. Die Analyse des COP-Paradigmas zeigt, dass AOP nicht den einzigen Ansatz darstellt, CCCs zu unterstützen.

Aufgrund der gezeigten, fehlenden CCCs-Unterstützung in PHP wurde in Kapitel 5 Überlegungen umgesetzt, AOP ohne Spracherweiterung oder den Einsatz eines Frameworks in PHP zu realisieren. Dabei wurde die Umsetzung durch die Reflexion-API von PHP und alternativ der Einsatz von Traits untersucht. Die Analyse zeigt, dass diese Ansätze nicht zielführend sind. PHP bietet keine Möglichkeit, mit vorhandenen Features CCCs zu unterstützen.

Es haben sich in der PHP-Umgebung bereits verschiedene Projekte an der AOP-Umsetzung versucht. In Kapitel 6 wurden diese inaktiven und aktiven Projekte vorgestellt und diskutiert. Das Ergebnis ist, dass derzeit keine AOP-PHP-Lösung vorliegt, die allen gestellten Anforderungen aus Kapitel 3 entspricht.

Aus diesem Grund wurde im Zuge dieser Diplomarbeit „YAPAF“ entwickelt und in Kapitel 7 vorgestellt und diskutiert. Die Umsetzung unter dem Aspekt der Erfüllung der Anforderungen aus Kapitel 3 stand dabei im Vordergrund. Die eingesetzten Mechanismen und die dafür genutzten PHP-Spracheigenschaften wurden präsentiert und der Einsatz validiert. Als Anwendungsbeispiel diente hierbei die Realisierung von Dependency Injection in PHP durch einen YAPAF-Aspekt. Besonders hervorzuheben bei der Umsetzung von YAPAF ist die Kombination aus stream_wrapper (vgl. Kapitel 2.3.7) und Autoloader (vgl. Kapitel 2.3.5). Diese Kombination ermöglicht die integrierte Metaprogrammierung von PHP-Software durch PHP-Software. Im Kontext dieser Diplomarbeit zeichnet sich YAPAF als einziges Projekt dadurch aus, AOP in PHP gemäß den Anforderungen aus Kapitel 3 zu realisieren. Weiter ist YAPAF leicht verständlich und einfach in Projekten einsetzbar.

Weitere Arbeiten zur Unterstützung von CCCs in PHP sind wünschenswert. Mit dem YAPAF-Framwork präsentiert diese Diplomarbeit einen guten Ansatz und nutzt dabei ein einmaliges Konzept, jedoch ist die Mächtigkeit dieser Realisierung noch weit von den Möglichkeiten eines AspectJ (vgl. Kaptiel 4.1) entfernt. Weiter fehlt YAPAF eine Integrated Development Environment (IDE)-Unterstützung analog zu AJDT, so dass der Entwickler im Alltag die YAPAF-Aspekt-Anwendung seiner Software schwer nachvollziehen kann.

Das Ziel war und bleibt es, die saubere Trennung funktionaler und nichtfunktionaler Belange voranzutreiben, um so Softwaresysteme und -entwicklung in PHP qualitativ zu verbessern.

Literaturverzeichnis

[Aho+06] Alfred V. Aho, Monica S. Lam, Ravi Sethi und Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. 2. Auflage. Prentice Hall, 2006.

[Aug+06] Otávio Augusto, Lazzarini Lemos, Daniel Carnio Junqueira, Marco Aurélio Graciotto Silva, Renata Pontin de Mattos Forte und John Stamey. „Using aspect-oriented PHP to implement crosscutting concerns in a collaborative web system“. In: *Special Interest Group on Design of Communication*. Association for Computing Machinery (ACM), 2006.

[Ber] Sebastian Bergmann. *Slideshare-phpcompilerinternals*. Letzter Zugriff 04.06.2012. URL: http://www.slideshare.net/sebastian_bergmann/php-compiler-internals.

[Big10] Paul Biggar. „Design and Implementation of an Ahead-of-Time Compiler for PHP“. Dissertation. Trinity College Dublin, 2010.

[BK06] Sebastian Bergmann und Günter Kniesel. „GAP: Generic Aspects for PHP“. In: *3rd European Workshop on Aspects in Software (EWAS)*. 2006.

[Buc08] Alexander Buch. „Ein Enterprise Feature übertragen in die PHP-Welt: Aspektorientierte Programmierung mit FLOW3“. In: *t3n* (Dez. 2008).

[CI05] Shigeru Chiba und Rei Ishikawa. „Aspect-Oriented Programming Beyond Dependency Injection“. In: *European Conference on Object-Oriented Programming (ECOOP)*. Herausgegeben von

Andrew Black. Band 3586. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005, Seiten 733–733.

[CT09] John Coggeshall und Morgan Tocker. *Zend Enterprise PHP Patterns*. Springer Verlag, 2009.

[Die98] Stephan Diehl. *A Formal Introduction to the Compilation of Java*. Technischer Bericht. Universität des Saarlandes, 1998.

[Dru] *Benchmarking Drupal with PHP op-code caches: APC, eAccelerator and XCache compared*. Letzter Zugriff 06.07.2012. URL: http://2bits.com/articles/benchmarking-drupal-with-php-op-code-caches-apc-eaccelerator-and-xcache-compared.html.

[EH04] Jim Hugunin Erik Hilsdale. „Advice Weaving in AspectJ“. In: *Aspect-Oriented Software Development (AOSD)*. Association for Computing Machinery (ACM), 2004.

[FM08] David Flanagan und Yukihiro Matsumoto. *The Ruby programming language*. O'Reilly, 2008.

[Gam+10] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Programmer's Choice. Addison Wesley Verlag, 2010.

[Gar09] Jorge Esparteiro Garcia. *Aspect-Oriented Web Development in PHP*. Technischer Bericht. Faculdade de Engenharia da Universidade do Porto, 2009.

[HCH08] Robert Hirschfeld, Pascal Costanza und Michael Haupt. „An Introduction to Context-Oriented Programming with ContextS“. In: *Generative and Transformational Techniques in Software Engineering II*. Herausgegeben von Ralf Lämmel, Joost Visser und João Saraiva. Band 5235. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, Seiten 396–407.

[HCN08] Robert Hirschfeld, Pascal Costanza und Oscar Nierstrasz. „Context-oriented Programming“. In: *Journal of Object Technology* 7(3) (2008), Seiten 121–151.

[Kic+01] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm und William G. Griswold. „An Overview of AspectJ“. In: *European Conference on Object-Oriented Programming (ECOOP)*. Springer Verlag, 2001.

[Kic+97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier und John Irwin. „Aspect-Oriented Programming“. In: *European Conference on Object-Oriented Programming (ECOOP)*. Springer Verlag, 1997.

[Lan] *PHP vs. Ruby vs. Python*. Letzter Zugriff 16.06.2012. URL: www.udemy.com/blog/modern-language-wars/.

[Pec] *AOP (PHP-Erweiterung)*. Letzter Zugriff 03.08.2012. URL: https://github.com/AOP-PHP/AOP.

[PGA02] Andrei Popovici, Thomas Gross und Gustavo Alonso. „Dynamic Weaving for Aspect-Oriented Programming“. In: *Aspect-Oriented Software Development (AOSD)*. Association for Computing Machinery (ACM), 2002.

[Php] *PHP: Hypertext Preproccesor*. Letzter Zugriff 06.07.2012. URL: http://www.php.net.

[Pou08] M.F.A. Pouw. *Aspect-Oriented Programming in PHP*. Technischer Bericht. University of Leiden, 2008.

[She06] Dmitry Sheiko. „Aspect Oriented Software Development and PHP“. In: *php | architect* (2006).

[Som06] Ian Sommerville. *Software Engineering*. 8. Auflage. Addison-Wesley, 2006.

[SSB05] John Stamney, Bryan Saunders und Simon Blanchard. „The Aspect-Oriented Web“. In: *Special Interest Group on Design of Communication (SIGDOC)*. Association for Computing Machinery (ACM), 2005.

[SSC05] John W. Stamney, Bryan Saunders und M. Cameron. „Introducing AOPHP“. In: *International PHP Magazine* (2005).

[Tio] *Tiobe-Sprachindex*. Letzter Zugriff 01.06.2012. 2012. URL: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

[W3t] *Usage of server-side programming languages for websites*. Letzter Zugriff 06.07.2012. URL: http://w3techs.com/technologies/overview/programming_language/all.

[Wam07] Dean Wampler. „Aspect-Oriented Design in Java/AspectJ and Ruby“. In: *International Conference on Software Engineering (ICSE)*. 2007.

Abkürzungsverzeichnis

AOP
Aspekt-Orientierte Programmierung

C+P
Copy and Paste

CCCs
Cross-Cutting Concerns

CMS
Content Management System

COP
Context-Orientierte Programmierung

CWD
Current Working Directory

DI
Dependency Injection

EIS
Enterprise Information System

GPL
GNU General Public License

IDE
Integrated Development Environment

JVM
: Java Virtual Machine

MVC
: Model View Controller

OOP
: Objekte-Orientierte Programmierung

ORM
: Object Relational Mapping

VM
: Virtual Machine

Glossar

C+P

„Copy and Paste“ (C+P) bezeichnet das Vorgehen während der Softwareentwicklung, Sourcecode innerhalb der Software zu kopieren, um so die gleiche Funktionalität an anderer Stelle zu erlangen.

CMS

Ein „Content Management System“ ist eine Software zur Verwaltung von Inhalten meist einer Website.

CWD

Das „Current Working Directory“ gibt das aktuelle Arbeitsverzeichnis an.

Eclipse

Bei Eclipse handelt es sich um eine populäre IDE.

EIS

Ein „Enterprise Information System“ dient dem persistenten Speichern betriebskritischer Daten. Bei einem EIS kann es sich also bspw. um eine Datenbank handeln.

GPL

Die „GNU General Public License“ ist eine freie Software-Lizenz zur Lizenzierung freier Software.

IDE

Ein „Integrated Development Environment“ ist eine Software, die den Entwickler beim Programmieren bspw. durch Code-Vervollständigung oder Syntax-Unterstützung behilflich sein ist.

JVM

Bei der „Java Virtual Machine“ handelt es sich um eine spezielle VM, die Java-Bytecode ausführt.

Lexer

Der lexikalische Scanner (Lexer) hat beim Kompilieren von Software die Aufgabe, aus dem vorhandenen Sourcecode die zugehörigen Token zu generieren [Aho+06].

Metaprogrammierung

Metaprogrammierung bezeichnet das Ändern von Programmen durch Programmierung. Ein Metaprogramm betrachtet ein zu änderndes Programm als Daten, die verarbeitet und manipuliert werden.

Mockup

Ein Mockup ist eine rudimentäre Implementation einer Schnittstelle. Diese reduzierte Variante kann in frühen Entwicklungsphasen oder beim Testen dazu genutzt werden, eine bekannte Schnittstelle zu verwenden, wobei diese keine eigene Funktionalität mitbringt.

MVC

Das MVC-Pattern (kurz für „Model View Controller“) ist ein Software-Architekturmuster. Dieses Muster beschreibt die Aufteilung einer Software in drei Bereiche: Das Model umfasst die Geschäftslogik einer Software, der View präsentiert ausschließlich Daten, und die Controller-Schicht vermittelt zwischen diesen Bereichen.

ORM

Das „Object Relational Mapping“-Konzept beschreibt das Verknüpfen von Objekten im Software-Sinne mit Einträgen in einer relationalen Datenbank.

Stream

Bei einem Stream handelt es sich um einen Datenstrom, dessen Abschluss unbekannt ist und der daher fortlaufend bis zum Ende des Stroms verarbeitet wird. Ein Beispiel für einen Stream stellt das zeilenweise Einlesen einer lokalen Datei dar.

VM

Eine „Virtual Machine“ ist ein Computer, die nicht auf existenter Hardware ausgeführt wird, sondern in einer bereitgestellten Software-Umgebung.

Tabellenverzeichnis

Listings