

Fachhochschule Gießen-Friedberg

■ Fachbereich Mathematik, Naturwissenschaften und Informatik

Seminararbeit
im Studienschwerpunkt Systemtechnik
zum Thema

LARA

Logistics and Repair Application

A PHP Enterprise Application Framework

Eingereicht von:	Kevin Wennemuth
Email:	kevin.wennemuth@mni.fh-giessen.de
Matrikelnummer:	712893
Betreuer:	Prof. Dr. P. Kneisel
Erstellt am:	1. September 2007

Eingereicht im Sommersemester 2007
im Fachbereich Mathematik, Naturwissenschaften und Informatik (MNI)
der Fachhochschule Gießen-Friedberg

Wiesenstraße 14
D-35390 Gießen

Inhaltsverzeichnis

1	Einleitung	1
1.1	Widmung	1
1.2	Motivation	1
1.3	Überblick	2
2	Begriffe und Techniken	3
2.1	Java ist auch nur ein Kaffee	3
2.2	PHP mal anders	4
2.2.1	Von Interfaces, Klassen und dem ganzen Rest	4
2.2.2	Die Geister, die ich rief... Magic Methods	5
2.2.3	Mehr Power! Die Funktionalität von PHP	6
3	LARA - Logistic And Repair Application	9
3.1	Was ist ein Application Framework?	10
3.2	Das Rad	11
3.3	Ordnung im Chaos: Die Architektur	13
3.3.1	Model-View-Controller Model 2	13
4	Kernkomponenten	15
4.1	Einmal Objekt, immer Objekt: LCoreObject und LObject	15
4.2	Eine Klasse für sich: LCoreClassLoader	16
4.3	Ausnahme der Ausnahme: LException	18
5	Basis... hier spricht LARA	21
5.1	Instanzierung != Instanzierung	21
5.2	Instanzierung en Detail	22
6	Kleine Helferlein	23
6.1	XML oder die Suche nach dem heiligen Gral	23
6.2	Wer, wann, was: Logging	23
6.3	Babel und der Fisch: Von ALE und SOA	24
7	Am Horizont	27
7.1	Database Pooling	27
7.2	Session-Clustering	28
8	Fazit	29

Abbildungsverzeichnis

3.1	LARA Kernsystem (schematisch)	13
3.2	Model-View-Controller Model 1	14
3.3	Model-View-Controller Model 2	14
4.1	LARA Kernsystem (UML)	15
4.2	LCoreObject und LObject	16
4.3	Chained-Exceptions	18
5.1	LARA Instanzierung en Detail	21
6.1	LXmlObject Formate	23
6.2	LLogger Logging-Targets	24
6.3	Serviceorientierte Architektur (SOA)	24
6.4	LARA Application Link Enabling	25
7.1	LARA Datenbank-Pooling	27
7.2	LARA Session-Clustering	28

Listings

2.1	Objektorientierung und Vererbungskonzepte in PHP	4
2.2	Magic Methods in PHP	5
4.1	Verwendung des LCoreClassLoaders	16
4.2	Auswertung durch LCoreClassLoader	17
4.3	Chained-Exceptions in LARA	18
5.1	LARA Instanziierung	22


1 Einleitung

1.1 Widmung


Ich widme diese Ausarbeitung meinen Freunden, denn sie geben, ohne zu fragen...

„Mut steht am Anfang des Handelns, Glück am Ende.“
Demokritos von Abdera

1.2 Motivation

Seit nunmehr zehn Jahren verfolge und programmiere ich hauptsächlich *PHP*¹ . Anfangs noch zum Spaß und aus Neugierde über das aufkommende World-Wide-Web mit seinen Möglichkeiten, später dann aus informationstechnischem Interesse an dieser relativ neuen Programmiersprache (genauer Skriptsprache) und seinen Möglichkeiten. Letztendlich ist die Konzipierung und Implementierung von größeren Projekten in PHP zu meinem (bezahlten) täglich Brot geworden. Bei einer sehr intensiven Beschäftigung mit PHP gibt es meiner Meinung nach zwei Möglichkeiten: Man *liebt* PHP oder man *haßt* sie.

Wie man sicherlich durch die Existenz dieser Seminararbeit erkennen kann, tendiere ich eher zu ersterem. Obwohl manchmal auch zweiteres der Fall ist, ist PHP wie eine Droge: Man kann nicht ohne. Die Einfachheit der Entwicklung unter PHP ist bis jetzt von keiner mir bekannten Programmiersprache, außer JavaTM, übertroffen worden. Als völlig typfreie Programmiersprache ist sie in meinen Augen sehr angenehm zu programmieren. Doch gerade dies bringt auch Tücken mit sich: Nächtelanges Debuggen vor gedimmten Laptops und schlaflose Nächte, weil es einfach nicht richtig ist, dass ein Server plötzlich zusammenbricht, weil man *eine* Zeile geändert hat. PHP zwingt den Programmierer dazu aufzupassen, was er tut. Das hält mich im Training und verhilft mir zu differenzierten Perspektiven auf andere Programmierprojekte und Programmiersprachen.

Seit der Freigabe von PHP3 im Jahre 1997 (vgl. *PHP Historie*² ) hat sich der Funktionsumfang und die Palette der Funktionen in PHP fast jedes Jahr exponentiell erweitert. Mittlerweile ist es sehr zeitaufwändig, mit der Entwicklung Schritt zu halten. Spätestens seit der Veröffentlichung von Version 5.0.0 in 2004 hat PHP seinen Ritterschlag als vollwertige Programmiersprache bekommen.

Für mich als angehenden Informatiker ist es sehr spannend zu beobachten, wie sich diese Programmiersprache langsam aber stetig aus der Belächelung als Programmiersprache für Kinder und Hobbyisten zu einem ernst zu nehmenden Pendant von JavaTM entwickelt.

¹PHP: <http://www.php.net>

²PHP Historie: <http://www.php.net/manual/de/history.php>

1.3 Überblick

Ich möchte mit dieser Seminararbeit einen kurzen aber prägnanten Überblick über die derzeitige Situation von PHP geben und ein Beispiel für eine konkrete Konzipierung und Implementation eines Applikations-Frameworks in PHP demonstrieren.

Beginnend mit einem Vergleich zwischen PHP und Java™ und einem kurzen Einblick in die Konzeptfähigkeit von PHP folgt als Überleitung zur erwähnten Implementation ein Überblick über derzeit vorhandene und eingesetzte PHP Frameworks mit deren Stärken und Schwächen.

Nach diesem kurzen Überblick über die derzeitige “Marktsituation” der PHP Frameworks möchte ich den geneigten Leser auf eine Reise in die Weiten der Softwarearchitektur entführen. Anhand einer konkreten Eigenentwicklung des LARA PHP Frameworks sollen neue wie alte Konzepte solcher Frameworks demonstriert und verdeutlicht werden.

Nach einer Vorstellung der Entwicklungsmodelle werde ich tiefer in die Sphären der Implementierung eintauchen und exemplarisch anhand der Initialisierung des Frameworks einige Konzepte dieser Eigenentwicklung darstellen. In diesem Kapitel werden außerdem einige Basismodule des LARA Frameworks konzeptuell erläutert.

Im nächsten Kapitel möchte aus den Untiefen des LARA-Kerns wieder auftauchen und die Integrationsfähigkeit des LARA Frameworks behandeln. *Serviceorientierte Architekturen* sind hier ebenso mit von der Partie wie *Application Link Enabling*. Ein kurzer Überblick über die Fähigkeiten von LARA in diesen Bereichen ist der Inhalt dieses Kapitels.

Am Ende dieser Reise soll ein kurzer Ausblick auf die weitere Entwicklung des LARA Frameworks gegeben werden, der sie anregen soll, selbst einmal in die Welt von PHP abzutauchen.

Ich hoffe, Ihre Koffer sind gepackt...





2 Begriffe und Techniken

Um einen gezielten Einstieg in das nachfolgende Thema zu erleichtern, möchte ich vorher einige Begrifflichkeiten in den Fokus der Aufmerksamkeit rücken. PHP ist im Gegensatz zu Java™ eine sehr leicht zu erlernende Programmiersprache. Doch wie bei allen mir bekannten, leichten Programmiersprachen ist es umso schwieriger, gute Programme zu schreiben, je einfacher strukturiert eine Programmiersprache ist.

Die Betrachtung der verschiedenen nachfolgenden Themen erfolgt unter dem Grundgedanken des Einsatzes in webbasierten Bereichen.

2.1 Java ist auch nur ein Kaffee

Von einfachen Applets bis hin zu komplexen verteilten Anwendungen in allen Bereichen der Informatik ist Java™ die am meisten eingesetzte Programmiersprache. Java™ ist eine klassische, sehr typorientierte, strukturierte und damit auch zum großen Teil restriktive Programmiersprache. Gerade die detailreiche Typorientierung ist Segen und Fluch zugleich.

Warum nimmt man also in Zeiten von so mächtigen Werkzeugen für Java™, wie *Spring*¹ , *Apache Struts*² , *Hibernate*³  oder *Tomcat*⁴  dennoch PHP zum Implementieren von webbasierten Anwendungen? Zu dieser einfachen Frage gibt es bei genauerem Hinsehen zwei Antworten:

- Weil es einfacher, schneller und unkomplizierter ist: Implementierungen in Java™ ziehen bei komplexeren Systemen einen ganzen Rattenschwanz an Anforderungen und Anpassungen nach sich. So ist es zum Beispiel im Webbereich unentbehrlich, einen gut konfigurierten Apache Webserver *und* einen Apache-Tomcat als Anwendungskontainer zu haben. Die Plattformen, die nötig sind, um eine Java™ Umgebung zu betreiben, sind sehr wartungs- und installationsaufwändig. Die Implementierung als solche ist meist komplizierter, da Programmierer per Java™ Definition dazu verpflichtet sind, bestimmte Paradigmen einzuhalten.
- Weil es kostengünstiger bei späteren Anpassungen ist: Nicht jedes Projekt, gerade im webasierten oder hochindividualisierten Bereichen, ist von Anfang an zu 100% ausgearbeitet oder konzipiert. Oft kommen erst gegen Ende eines Projektes Anforderungen oder Anforderungsänderungen zum Tragen, die gar nicht ersichtlich waren. Sollte es gar ein Individualprodukt sein, das ständig nach Vorgaben anderer (zum Beispiel Hersteller) angepasst werden muss, ist der Aufwand einer Anpassung in Java™ sehr hoch.

¹Spring: <http://www.springframework.org>

²Apache Struts: <http://struts.apache.org>

³Hibernate: <http://www.hibernate.org>

⁴Tomcat: <http://tomcat.apache.org>

2.2 PHP mal anders

Die Entwicklung von PHP als vollwertige Programmiersprache wird von Außenstehenden immer gerne belächelt. Dieses Vorurteil wurde fast allen bekannten Programmiersprachen der neueren Zeit entgegengebracht. Selbst Java™ galt lange Zeit als unausgereift und Spielerei. Diese Vorurteile gründen meist auf veraltetem Wissen oder gar Unwissenheit über die Entwicklung und Anwendungsmöglichkeiten. PHP bietet mittlerweile eine Alternative zu Java™ basierten Anwendungen. Hierbei muss nicht auf moderne Konzepte wie Objektorientierung oder Vererbung verzichtet werden. Ein großer Vorteil, der gerne als Nachteil für PHP angesehen wird, ist die Eigenschaft einer reinen Skriptsprache. Hierbei wird der auszuführende Code von PHP nicht in Maschinensprache übersetzt, sondern direkt ausgeführt. Bei einer Java™ Umgebung müsste ein entsprechend geänderter Code erst neu übersetzt und eingespielt werden. Hierbei entstehen Ausfallzeiten, die gerade im Web-Bereich nicht gern gesehen sind. PHP hat diesen Nachteil nicht.

2.2.1 Von Interfaces, Klassen und dem ganzen Rest

Ähnlich wie in Java™ kann man in PHP ebenfalls Interfaces zur restriktiven Definition von Objekt-API's verwenden. Differenzierte Klassen und Objekthierarchien sind in PHP ebenfalls realisierbar. Auch abstrakte und finale Klassentypen sowie die Konzepte von *public*, *protected* und *private* für Vererbungssteuerung von Methoden und Variablen sind integraler Bestandteil von PHP [MA07]. Nachfolgend soll ein einfaches Beispiel für die Verwendung aller Methoden und Konzepte gezeigt werden.

Listing 2.1: Objektorientierung und Vererbungskonzepte in PHP

```

1 interface life {
2     public function getName();
3     public function getAge();
4 }
5
6 abstract class abstractHuman implements life {
7     public $age = null;
8     public $name = null;
9     public function getName() {
10         return $this->name;
11     }
12 }
13
14 class parentCitizen extends abstractHuman {
15     public function getAge() {
16         return $this->age;
17     }
18
19     public sayHelloFrom() {
20         return 'Hello from '.$this->getName().'!';
21     }
22 }
23
24 class childCitizen extends parentCitizen {
25     public function getName() {
26         return parent::getName();
27     }
28 }
29
30 $foo = new parentCitizen(); # Neues Vater-Objekt instanzieren
31 $bar = new childCitizen();  # Neues Kind-Objekt instanzieren

```

```

32 $foo->name = "Meier";           # Vater::name == "Meier"
33 $foo->sayHelloFrom();           # Liefert "Hello from Meier"

```

PHP verwendet für Operationen standardmäßig Referenzen, wodurch die Verwendung von Objekten sehr speicherschonend ist. Natürlich können auch tiefe Kopien erstellt werden. Hierzu bietet PHP sogenannte *Magic Methods*.

2.2.2 Die Geister, die ich rief... Magic Methods

Ähnlich wie Java™ besitzt auch PHP spracheninterne Methoden zur Objektverwaltung unter bestimmten Kriterien. Diese so genannten *Magic Methods*⁵ bieten die Möglichkeit Callback-Funktionen für bestimmte interne Ereignisse von PHP zu überladen. Das nachfolgende Beispiel stellt ein Objekt zur Verbindung mit einer MySQL-Datenbank dar. Dieses Objekt soll persistent speicherbar sein. Das Problem, das hierbei auftritt, ist die eigentliche Verbindung zur Datenbank selbst. Diese wird wie alle Streams oder Socket-Verbindungen in PHP als *Ressource* dargestellt, was allgemein gesagt nichts weiter als eine Referenz auf ein internes anonymes Objekt ist. Diese Ressource kann nicht mit PHP Mitteln serialisiert, also persistent gespeichert werden. Abhilfe schaffen hier die exemplarisch für die *Magic Methods* aufgeführten Funktionen `__sleep` und `__wakeup`:

Listing 2.2: Magic Methods in PHP

```

1 class Connection {
2     protected $link;
3     private $server, $username, $password, $db;
4
5     public function __construct($server, $username, $password,
6     $db)
7     {
8         $this->server = $server;
9         $this->username = $username;
10        $this->password = $password;
11        $this->db = $db;
12        $this->connect();
13    }
14
15    private function connect()
16    {
17        $this->link = mysql_connect($this->server, $this->
18        username, $this->password);
19        mysql_select_db($this->db, $this->link);
20    }
21
22    public function __sleep()
23    {
24        mysql_close($this->link);
25    }
26
27    public function __wakeup()
28    {
29        $this->connect();
30    }
31 }
32
33 # Create new object instance

```

⁵Magic Methods: <http://de3.php.net/manual/de/language.oop5.magic.php>

```
32 $mysqlObject = new Connection('localhost','root','geheim','  
    meineDaten');  
33  
34 # Serialize object to string  
35 $serializedObject = serialize($mysqlObject);  
36  
37 # Write object to file persistently  
38 file_put_contents('/var/www/persistentCache/mysqlConnection01.  
    pdo', $serializedObject);  
39  
40 ...  
41  
42 # Read persistent object from file  
43 $objectString = file_get_contents('/var/www/persistentCache/  
    mysqlConnection01.pdo');  
44  
45 # Reinstanciate object  
46 $mysqlObject = unserialize($objectString);
```

Was passiert in diesem Beispiel? Zuerst wird eine Klassendefinition einer Datenbankverbindung erstellt, die persistent gespeichert werden soll. Damit dies passieren kann, muss bei dem Serialisierungsvorgang die eigentliche Verbindung abgebaut und bei dem Wiederherstellen des Objektes aus seinem persistenten Zustand wieder aufgebaut werden. Wie schon beschrieben übernehmen die Funktionen `__sleep` und `__wakeup` diesen Teil der Funktionalität. PHP bietet natürlich für alle wichtigen internen Ereignisse solche Methoden.

2.2.3 Mehr Power! Die Funktionalität von PHP

Die Stärke von PHP liegt in seinem immensen, ständig erweiterten Funktionsumfang. Von Modulen für Apache-Funktionen über Pakete für Streams und Softwareanbindungen für freie und kommerzielle Software bis hin zu verschiedenen Treibern und Abstraktionsmodellen für relationale und objektrelationale Datenbanken (zum Beispiel MySQL, postGREsql, DB2, Oracle, Paradox, Informix) bietet PHP alles, was das Programmiererherz begehrt. Selbst eine PHP2Java™-Bridge ist im Funktionsumfang enthalten, um bestehende PHP und Java™ Applikationen reibungslos ineinander greifen zu lassen. Nachfolgend eine kurze Liste der PHP internen Pakete: Diese Pakete werden durch eine Heerschaar an externen,

- Apache
- DOM XML
- IBM DB2, Cloudscape and Apache Derby
- IMAP, POP3 and NNTP
- LDAP
- MaxDB and MySQL/AB
- Microsoft SQL Server
- Oracle 8
- PostgreSQL und SQLite
- SDO XML Data Access Service
- PHP bytecode Compiler
- COM Support für Windows
- Netzwerkfunktionen
- Mcrypt Encryption
- PDF
- SNMP and TCP
- SOAP and XML-RPC
- XSLT and WDDX
- SSH2 and OpenSSL
- Streaming, Tokenizer and Strings

in PHP oder C/C++ entwickelten Projekten ergänzt, abstrahiert oder erweitert. Als Fazit zu PHP kann man sagen: Ein PHP Benutzer konzentriert sich nicht auf das *wie*, sondern auf das *was*.

3 LARA - Logistic And Repair Application

Um einen Einblick in das Konzept und die Hintergründe einer Eigenentwicklung eines PHP Frameworks wie LARA zu bekommen, ist es notwendig, sich einige bereits bestehende Projekte und Frameworks genauer anzusehen. Ein kurzer Überblick über die aktuelle Situation am “Frameworkmarkt” ist hierbei unerlässlich. Im nachfolgenden Kapitel sind die momentan wichtigsten PHP Frameworks aufgelistet, die auch produktiv in verschiedenen Einsatzszenarien genutzt werden.

3.1 Was ist ein Application Framework?

Im Bereich der Softwareentwicklung definiert man ein Framework folgendermaßen:




„Ein Framework ist ein wiederverwendbarer Entwurf, der durch eine Menge von abstrakten Klassen, sowie dem Zusammenspiel ihrer Instanzen beschrieben wird.“

Ausgehend von dieser Definition (vgl. [Lu5]) können zu Frameworks einige wichtige Charakteristiken beschrieben werden:

- **Reuse of analysis:** Dem Entwurf eines Frameworks geht eine gründliche Analyse der Anwendungsdomäne voraus.
- **Hot spots und hooks:** Das Framework stellt an bestimmten Stellen einfache Methodiken zur Verfügung, mit denen ein Anwender arbeiten kann. Dies geschieht meist durch abstrakte Klassen und Methoden, die vom Anwender gezielt überschrieben und an die gegebenen Anforderungen angepasst werden können.
- **Inversion of control:** Der Anwendungsfluss wird nicht vom Anwender bestimmt, sondern vom Framework, das den Anwendungscode ausführt. Dieses Kriterium bildet die Abgrenzung zu reinen Klassenbibliotheken, da bei reinen Klassenbibliotheken die Anwendungssteuerung bei der Anwendung selbst verbleibt.

Ein Application Framework ist also eine strukturierte Sammlung von Methoden und Objekten für wiederkehrende Prozesse. Ein einfaches Beispiel eines solchen Prozesses ist ein OK/Abbrechen-Dialog. Diese Prozessform kommt in vielen verschiedenen Anwendungen immer wieder vor. Warum also jedesmal den Dialog neu programmieren? Nicht nötig. Mit der richtigen Abstraktion der Dialogfunktionen und -eigenschaften ist es sehr gut möglich für diesen Prozess eine generische Komponente zu erstellen, die in verschiedenen Projekten oder Applikationen auf die gleiche Weise benutzt werden kann.

Wenn dies nun mit allen notwendigen Komponenten geschieht, die für ein Projekt notwendig sind, spricht man von einem Application Framework. Solche Frameworks vereinfachen wiederkehrende Strukturen und bieten die Möglichkeit, sehr schnell die gesetzten Aufgaben zu erreichen, ohne jedesmal alle Komponenten neu entwickeln zu müssen. Im Web-Bereich heißen diese Frameworks *Web-Application-Frameworks* oder kurz *Web-Frameworks*. Die nachfolgende Betrachtung bezieht sich ausschließlich auf *Web-Frameworks* der neueren Generation. Ein gutes *Web-Framework* sollte einige konzeptuelle Kriterien erfüllen:

- eine Schnittstelle für abstrakten Datenbankzugriff (O/R-Mapper)
- Templating- und Caching-Mechanismen
- *MVC*¹  oder *MVC2*²  Design-Patterns
- *Scaffolding*³ 
- gute Dokumentation
- Objektorientierung
- Schnittstellen zu anderen Systemen (SOAP, XML-RPC)

Ausgehend von diesen sechs Grundideen möchte, im nächsten Kapitel näher auf verschiedene Frameworks eingehen.

¹MVC: <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>






²MVC2: <http://de.wikipedia.org/wiki/MVC>

³Scaffolding: <http://de.wikipedia.org/wiki/Scaffolding>

3.2 Das Rad

Wenn man die Überlegung antritt, ein Application Framework in PHP zu konzipieren, tritt früher oder später die Frage auf: Warum das Rad neu erfinden? Es gibt das alles doch schon [PHP07]. Die Frage ist auf den ersten Blick nicht leicht zu beantworten, jedoch könnte man genauso gut fragen, warum nicht alle Menschen auf der Welt Unix/Linux benutzen. Ein Framework ist im Grundsatz immer auch eine Konzession an das mögliche Einsatzgebiet [Cre07].

Eine kurze Übersicht der aktuellen Application Frameworks für PHP:

<i>Prado</i> ⁴ 	Vorteile: <ul style="list-style-type: none"> - Plug-in Fähigkeit - integrierte ACL - Templating und Caching - AJAX integriert Nachteile: <ul style="list-style-type: none"> - keine Datenbankabstraktion - kein einheitliches Entwicklungspattern - schlechte Dokumentation
<i>symfony</i> ⁵ 	Vorteile: <ul style="list-style-type: none"> - großer Funktionsumfang - sehr gute Dokumentation - sehr gute Konzepte Nachteile: <ul style="list-style-type: none"> - kein Templating oder Caching - sehr langsame Ausführungsgeschwindigkeit (<i>Benchmark</i> ⁶ ) - keine Plug-in Fähigkeit
<i>CakePHP</i> ⁷ 	Vorteile: <ul style="list-style-type: none"> - sehr gute Konzepte - Plug-in Fähigkeit - guter Funktionsumfang Nachteile: <ul style="list-style-type: none"> - schlechte Dokumentation - kein Templating oder Caching - keine Modulunterstützung - alter PHP4 Code
<i>ZendFramework</i> ⁸ 	Vorteile: <ul style="list-style-type: none"> - leichte Syntax - angelehnt an PHP-PEAR - reine PHP5 Implementation - Modularisierung - vom PHP "Hersteller" Nachteile: <ul style="list-style-type: none"> - keine integrierte AJAX Unterstützung - kein Templating

⁴Prado: <http://www.xisc.com/>

⁵symfony: <http://www.symfony-project.com/>

⁶Benchmark: <http://www.sellersrank.com/web-frameworks-benchmarking-results/>

⁷CakePHP: <http://cakephp.org/>

⁸ZendFramework: <http://framework.zend.com/>




	<ul style="list-style-type: none"> - nur Funktionssammlung a la PHP-PEAR - kein objekt-relacionales Mapping - geringer Funktionsumfang
<i>CodeIgniter</i> ⁹ 	Vorteile: <ul style="list-style-type: none"> - reiner PHP5 Code - sehr gute Dokumentation - großer Funktionsumfang - sehr schnelle Ausführungsgeschwindigkeit (<i>Benchmark</i>¹⁰  Nachteile: <ul style="list-style-type: none"> - kein Design Pattern - Beta Stadium
<i>PHP-PEAR</i> ¹¹ 	Vorteile: <ul style="list-style-type: none"> - gute Funktionalitäten - exotische Aufgaben schon implementiert - z.T. sehr gute Implementationen Nachteile: <ul style="list-style-type: none"> - reine Funktionssammlung - PHP4 und keine Objektorientierung - z.T. sehr alter Code - Namenskonflikte in Paketen - keine konsistente API bzw. Coding Standards - sehr schlechte Dokumentation

Tabelle 3.1: Stärken und Schwächen von PHP Frameworks

Meist sind die bereits vorhandenen Frameworks veraltet oder so speziell konzipiert, dass sie sich nur selten anderweitig einsetzen lassen, als für ihren eigentlich erdachten Zweck. Ein zweiter wichtiger Faktor ist die Anpassungsfähigkeit: Viele der hier vorgestellten Frameworks lassen sich nur mit erheblichem Aufwand anpassen. Ein dritter, noch schwerwiegenderer Faktor ist die Herrschaft über die Entwicklung. Interna dieser Frameworks können entweder durch das Konzept selbst oder durch die Unstrukturiertheit der Architektur nur schwerlich durchschaut bzw. getestet oder gar geändert werden. Gerade im marktwirtschaftlichen Bereich würde dies bedeuten ein Produkt zu verwenden, das man nur *von außen* kennt.

⁹CodeIgniter: <http://codeigniter.com/>

¹⁰Benchmark: <http://www.sellersrank.com/web-frameworks-benchmarking-results/>

¹¹PHP-PEAR: <http://pear.php.net/>

3.3 Ordnung im Chaos: Die Architektur

Die Architektur von LARA basiert auf dem simplen Prinzip eines Microkernels. Ein sehr kleiner Teil der vorhandenen Klassen bildet das so genannte Runtime-Environment. Diese Kernfunktionalitäten übernehmen die wichtigsten Aufgaben wie Konfiguration, Verwaltung von Klassen und Objekten sowie das Exception-Handling. Zu den Aufgaben des Kerns gehört neben der Verwaltung von Sessiondaten auch die Bereitstellung der meisten abstrakten Klassen und Interfaces für den gesamten Rest des Systems. Er bietet außerdem eine Reihe kleinerer Kernmodule, wie etwa den Umgang mit XML Daten oder die Bereitstellung von Basistypen wie Datums- und Stringfunktionen. Basierend auf diesen

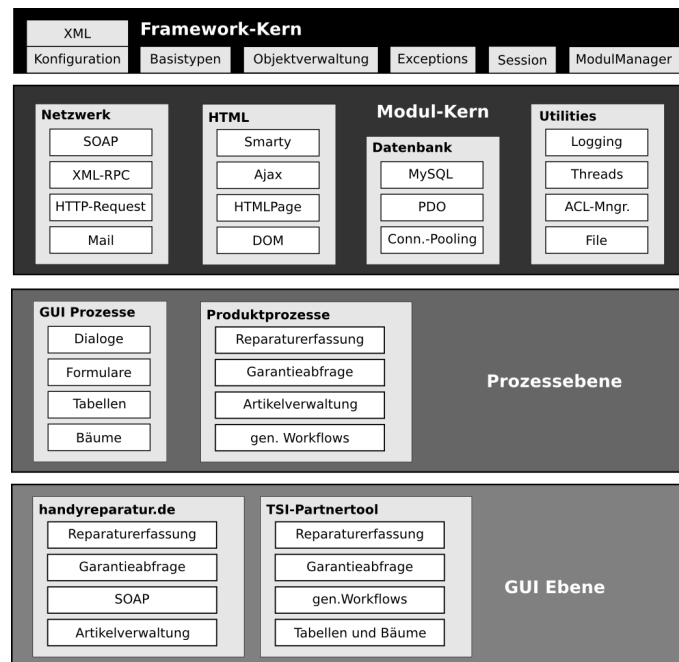


Abbildung 3.1: LARA Kernsystem (schematisch)

Kernfunktionalitäten setzt die Ebene der Module auf. Module sind in diesem Kontext als Funktionalitäten mit strikter Objekthierarchie zu sehen. Sie erweitern den Kern um komplexere Aufgaben oder Workflows.

Oberhalb dieser zwei Grundebenen befindet sich die Abstraktionsschicht für Design-Pattern. Momentan werden die verfügbaren Komponenten der ersten zwei Ebenen in MVC2 (Model-View-Controller Model 2) Pattern zusammengefasst und bilden somit jeweils einen Schritt oder Teilaspekt von wiederkehrenden Prozessen ab.

Als letzte Ebene kommt die eigentliche GUI Entwicklung. Hier werden die Prozessketten mit den einzelnen Prozessstücken befüllt und in eine Applikation eingebracht.

3.3.1 Model-View-Controller Model 2

Zur Ausgestaltung der Prozessebene kommt das Model-View-Controller Pattern [Sin04] in der konkreten Ausprägung des *MVC Model 2* zum Einsatz. Beim Model-View-Controller Model 1 Pattern allgemein hat man einen geschlossenen Kreislauf zwischen allen Beteiligten. Ein Client fragt beim Controller nach einem Datenmodell. Dieser meldet die Anfrage an das Datenmodell, welches wiederum seine registrierte Ansicht aktualisiert. Die Ansicht

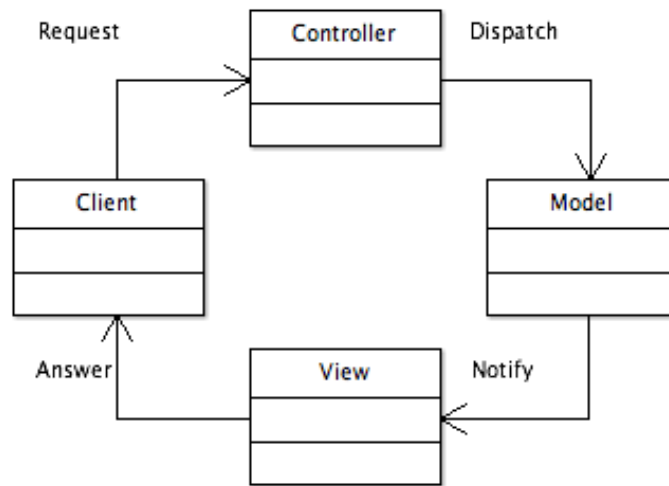


Abbildung 3.2: Model-View-Controller Model 1

wird beim Client angezeigt. Dies stellt eine sehr gute Möglichkeit zur Trennung von Daten, Geschäfts- und Präsentationslogik dar.

Bedingt durch den Einsatz in der verbindungslosen Umgebung des HTTP kann in dieser Umgebung der Model-View-Controller Model 1 nicht gänzlich laut seines Konzeptes zum Einsatz kommen. Die derzeitig angezeigte Ansicht des Datenmodells kann nicht direkt vom Datenmodell selbst angesprochen werden, wie dies von MVC1 verlangt wird. Im

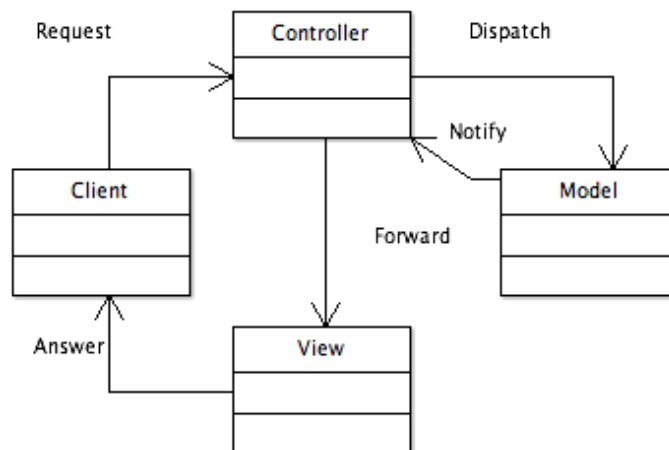


Abbildung 3.3: Model-View-Controller Model 2

Gegensatz zu MVC1 ist bei Model-View-Controller Model 2 ein größeres Gewicht auf den Controller verlegt worden. Der Client bei einer Änderung des Datenmodells den aktuellen Stand bei dem registrierten Controller erfragen bzw. die Änderungen weitergeben. Dieser gibt die Änderungen wiederum an das jeweilige Datenmodell weiter und aktualisiert die dazugehörige Ansicht mit den geänderten Daten. Die Ansicht wird anschließend beim Client angezeigt.

4 Kernkomponenten

Nach dieser allgemeinen Betrachtung von Frameworks sowie deren Vertreter und dem allgemeinen Überblick über die Systemarchitektur von LARA kommen wir in diesem Kapitel zu den internen Objekthierarchien (vgl. [Sch06]). LARA stellt im Kern nur wenige Klassen zur Verfügung, da die überwiegende Funktionalität, wie in Kapitel 3 beschrieben, von separaten Modulen übernommen wird.

Abbildung 4.1 zeigt einen kleinen Überblick über die Klassenstruktur des Kernsystems: Ich möchte im folgenden auf einige dieser Komponenten näher eingehen, da sie essentiell

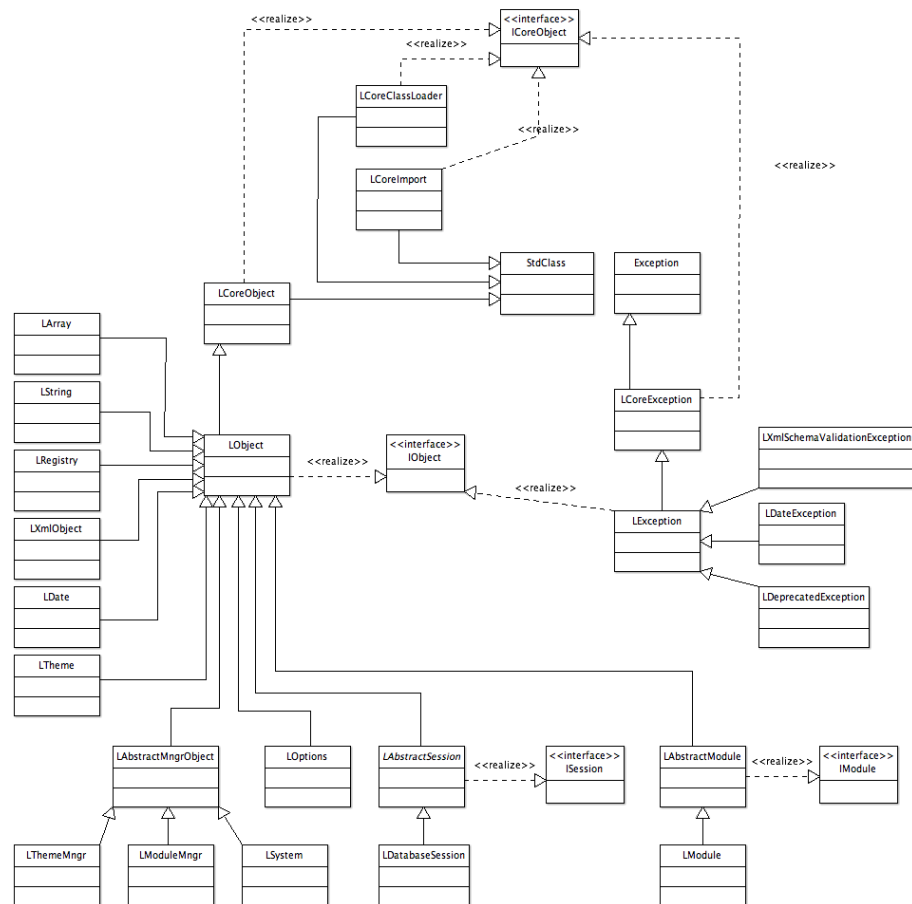


Abbildung 4.1: LARA Kernsystem (UML)

sowohl für die Entwicklung, als auch für die Handhabung weiterer Funktionalitäten sind.

4.1 Einmal Objekt, immer Objekt: LCoreObject und LObject

Die Klasse `LCoreObject` bietet eine Erweiterung der Standardklasse `StdClass` von PHP dar. Neben einigen Standardmethoden von PHP besitzt diese Klasse einige erweiterte Funktionalitäten wie eine eindeutige ID zur Referenz auf das jeweilige Objekt. Auch Methoden zum Hashing des Objekts und zur Darstellung in XML sind ebenso erweitert worden wie

die direkte Integration der *Reflection-Funktionalität*¹  von PHP.

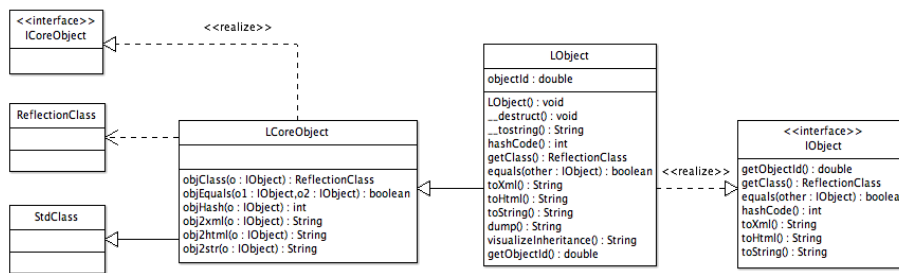



Abbildung 4.2: LCoreObject und LObject

Die von LCoreObject abgeleitete Klasse LObject erweitert diese Funktionalitäten nochmals um Debugging- und Ausgabefunktionen. Alle Basis- und erweiterten Objekte des Frameworks basieren im Grunde immer auf einer Ableitung von LObject.

4.2 Eine Klasse für sich: LCoreClassLoader

Dieses Objekt ist als Kern für die gesamte Objekt- und Klassenverwaltung zuständig. Die Klasse LCoreClassLoader ist als *Singleton-Pattern*²  implementiert und existiert somit nur ein einziges Mal pro LARA-Instanz. LCoreClassLoader ist zuständig für das Auffinden und Laden einer Klassendatei anhand ihres Klassenpfades. Ähnlich wie in Java™, werden Klassen im LARA Framework anhand eines Klassenpfades adressiert. Die Funktionalität des LCoreClassLoaders erlaubt es verschiedene *Aliase* für bestimmte Basispfade anzugeben. Diese *Aliase* werden bei der Adressierungsangabe durch die jeweilige Aliasentsprechung ersetzt.

Nachfolgend sei ein kurzes Beispiel vorgestellt, welches ein typisches Szenario des Ladens und Instanzierens darstellt. Die Funktion `import` ist hierbei normalerweise separat und außerdem global in einer LARA-Instanz definiert. Zu Verdeutlichung des Ablaufs jedoch erscheint sie bei diesem Beispiel nochmals:

Listing 4.1: Verwendung des LCoreClassLoaders

```

1  /**
2   * Import a dotclass ("lara.base.LFoo" imports "%LARA_CLASSPATH
   *   %/lara/base/LFoo.php")
3   * @return LImport
4   */
5  function import($dotname, $file, $line) {
6      return LCoreClassLoader::getInstance()->loadClass($dotname,
   *   $file, $line);
7  }
8
9  /** register 'lara' as alias for '/var/www' */
10 define('LARA_BASE_PATH', '/var/www');
11 define('LARA_CLASSPATH', LARA_BASE_PATH.'/framework/lara');
12 LCoreClassLoader::getInstance()->registerClasspath('lara',
   *   LARA_CLASSPATH);
13

```


¹Reflection-Funktionalität: <http://www.php.net/manual/de/language.oop5.reflection.php>


²Singleton-Pattern: http://en.wikipedia.org/wiki/Singleton_pattern

```

14 /** imports "%LARA_CLASSPATH%/lara/base/LObject.php" */
15 import('lara.base.LObject', __FILE__, __LINE__);
16
17 /** imports "%LARA_CLASSPATH%/lara/base/LException.php" */
18 import('lara.base.LException', __FILE__, __LINE__);
19
20 $foo = new LObject();      # Instanciate new LObject
21 $bar = new LException();   # Instanciate new LException

```

Anhand dieses Pfades kann die Klasse inkludiert und entsprechend im weiteren Programm verwendet werden. Die Adressierung erfolgt immer absolut zum Basispfad des Frameworks und ist somit unabhängig von der Konfiguration des Apache (*DokumentRoot*³ )³. Der LCoreClassLoader verwaltet hierbei intern die bereits bekannten Klassen und lädt nur bei Bedarf oder wenn die Klasse bisher unbekannt sein sollte die entsprechende Klassendefinition.

Sollte sich die Klassendefinition nicht im angegebenen Pfad befinden, beherrscht der LCoreClassLoader ebenfalls einige Fallbackstrategien, um die Klassendefinition dennoch zu laden. So sucht er zuerst im angegebenen Klassenpfad, danach in den globalen *include-Pfaden*⁴ , von PHP und wenn auch dies fehlschlägt, traversiert der LCoreClassLoader automatisch durch das gesamte Framework, um die Klassendefinition zu finden.

Zur erweiterten Analyse und Optimierung bestehender Projekte erlaubt der LCoreClassLoader außerdem eine gezielte Ausgabe seiner Verwaltungsdaten. Ausgegeben werden die Anzahl der Ladevorgänge pro Klassendefinition, sowie Pfad- und Timingangaben.

Listing 4.2: Auswertung durch LCoreClassLoader

```

1 List of made imports at position:
2 File: /var/www/framework/lara/test/index.php
3 Line: 39
4
5 lara.core.ICoreObject 1 import(s):
6
7     File: /var/www/framework/lara/lara.php
8     Line: 58
9     Time: 1187436801.1431291103
10
11 lara.base.LObject 13 import(s):
12
13     File: /var/www/framework/lara/lara.php
14     Line: 81
15     Time: 1187436801.1621630192
16
17     File: /var/www/framework/lara/base/LXmlObject.php
18     Line: 16
19     Time: 1187436801.1663339138
20
21     File: /var/www/framework/lara/util/LUrl.php
22     Line: 16
23     Time: 1187436801.1770050526
24
25 ...

```

³DokumentRoot: <http://httpd.apache.org/docs/2.0/de/mod/core.html>

⁴include-Pfaden: <http://www.php.net/manual/en/function.set-include-path.php>

Ebenfalls lassen sich hier die so genannten *Reverse-Imports*, d.h. von wo wurden die einzelnen Klassendefinitionen angefordert, analysieren und gegebenenfalls anpassen.

4.3 Ausnahme der Ausnahme: LException

Für ein ausgereiftes Application Framework wie LARA ist natürlich eine ausgedehnte Fehlerdiagnose und -behandlung unerlässlich. PHP bietet genau wie Java™ eine sehr gute Möglichkeit der Fehlerbehandlung durch *PHP-Exceptions*⁵. Diese Basisfunktionalität wird durch LARA-eigene Exceptions, hier LCoreExceptions und LExceptions, nochmals erweitert. Angelehnt an Exception-Pattern von Java™ können im Fehlerfall auch Exceptions verschachtelt werden, sie werden zu so genannten *Chained-Exceptions*⁶. Gerade bei



Abbildung 4.3: Chained-Exceptions

Exceptions, die über Anwendungsdomänen hinweg geworfen werden, etwa durch SOAP-Schnittstellen oder direkte PHP2Java™-Bridge Projekte, ist diese Funktionalität sehr hilfreich und unerlässlich. In Abbildung 4.3 kann man die Darstellung einer solchen SOAP-Exception von einem entfernten Service als Ausgabe und wie sie in LARA wieder gefangen und verschachtelt wird sehen. Wie man erkennen kann, unterscheiden sich die Typen der Exceptions, je nachdem ob sie lokal oder entfernt geworfen wurden.

Listing 4.3: Chained-Exceptions in LARA

```

1  ...
2
3  try {
4      if (true) {
5          throw new LRemoteException('DUMMYFAULT');
6      }
7  } catch (LException $e) {
8      throw new LException('Try-Catch-Block failed!', $e);
9  }
10
```

⁵PHP-Exceptions: <http://www.php.net/manual/de/language.exceptions.php>

⁶Chained-Exceptions: <http://java.sun.com/docs/books/tutorial/essential/exceptions/chained.html>

¹¹ | . . .


Dieses kleine Beispiel verdeutlicht den einfachen Umgang mit Exceptions und Chained-Exceptions in LARA. Hier werden zwei unterschiedliche Exceptions ineinander verschachtelt. Dies kann beliebig tief geschehen und ermöglicht eine gezielte Fehlerdiagnose, da man den Fehlerweg durch die Exceptionkette zurückverfolgen kann.

5 Basis... hier spricht LARA

Nachdem wir eine kleine Reise in den Kern von LARA unternommen haben, um einige wichtige Mechanismen von LARA kennenzulernen, werden wir jetzt ein wenig höher aufsteigen, um die eigentliche Frameworkfunktionalität zu betrachten.

5.1 Instanziierung != Instanziierung

Bei der Verwendung eines in sich abgeschlossenen Frameworks gibt es einige Überlegungen in Bezug auf die Interaktion mit anderen Anwendungsdomänen, die nicht außer acht gelassen werden dürfen:

- **Konfiguration:** Für den Betrieb von LARA sind einige wichtige Konfigurationseinstellungen, wie Pfade, Logdateien, Datenbankangaben oder Systemvariablen nötig. Diese Angaben werden von LARA in einer feststehenden Grundkonfiguration geladen, die als Fallback dienen und vom jeweiligen Projekt, das mit LARA realisiert wird, überschrieben werden kann.
- **Session:** Die Verwendung der Session ist bei einem Framework dieser Größe eine ernste Angelegenheit. Es dürfen nicht aus Unachtsamkeit Variablen anderer Anwendungen oder Module überschrieben werden. Ebenso müssen bei Objekten, die in der Session vorgehalten werden, die Klassendefinitionen vor den Laden der Session bekannt sein. Ein beliebter Fehler in vielen Projekten, der unweigerlich zu einem unvollständigen Objekt (`__PHP_Incomplete_Class`¹ ) und somit meist zu seltsamen Fehlern führt.
- **Lazy-Init:** Um weitere Projekte oder Interaktionen gezielt zuzulassen, sollte die Instanziierung offen für Zwischenschritte von außen sein. Hierzu wird die Initialisierung in einzelne Schritte aufgebrochen, zwischen denen der Anwender eigene Programmteile ausführen kann.

Um den Instanzierungsvorgang genauer zu betrachten, folgt eine kurze schematische Darstellung der einzelnen Schritte bei diesem Vorgang:

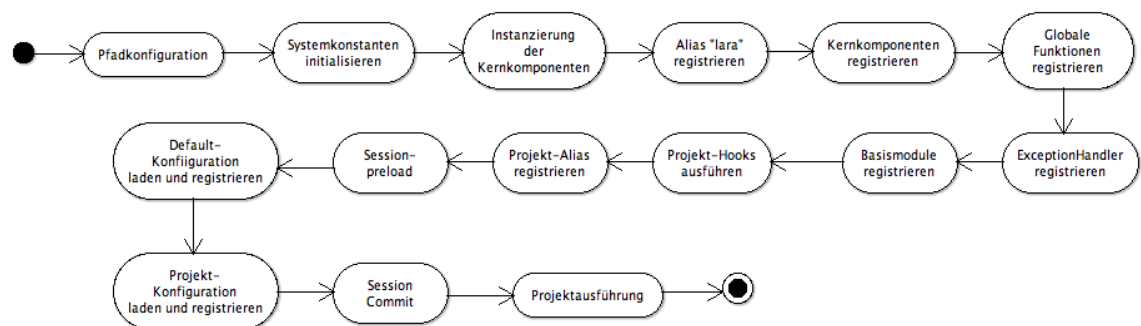


Abbildung 5.1: LARA Instanziierung en Detail

¹ `__PHP_Incomplete_Class`: <http://de.php.net/session>

5.2 Instanziierung en Detail

Um das Framework LARA zu instanzieren, sind nur einige wenige Konfigurationseinstellungen notwendig. Die wichtigsten Einstellungen werden von LARA selbst erzeugt, jedoch müssen für eine erfolgreiche Instanziierung die Basispfade angegeben werden:

Listing 5.1: LARA Instanziierung

```

1 /* Define framework and project paths */
2 define('LARA_BASE_PATH', '/var/www');
3 define('PROJECT_BASE_PATH', '/var/www/projekte/handyreparatur.
  de');
4 define('PROJECT_CONFIG', 'file:///PROJECT_BASE_PATH./config.
  xml');
5 define('PROJECT_UI', 'project.includes.gui');
6
7 /* Execute framework */
8 require_once(LARA_BASE_PATH./framework/lara/lara.php');
9
10 /* Load project XML configuration */
11 laraAddConfig('project', PROJECT_CONFIG);
12
13 /* Commit lazy init to $_SESSION */
14 laraInitCommit();
15
16 /* Execute project */
17 ...

```

Dieses kurze Beispiel verdeutlicht die standardmäßige Instanziierung von LARA. Das Framework ist nach dieser kurzen Einleitung voll funktionsfähig. Aber wie das Sprichwort schon sagt: *Der Teufel steckt im Detail*. Was hier einfach aussieht, zieht einige interne Mechanismen nach sich, die ein Höchstmaß an Flexibilität hinsichtlich des Einsatzes von LARA bieten sollen.

6 Kleine Helferlein

Nachdem ich nun einen ausführlichen Überblick über die Kernkomponenten und die Instanzierung von LARA gegeben habe, möchte ich in diesem Kapitel auf einige Anwendungsgebiete von LARA und die damit verbundenen Module eingehen. In diesem Kapitel werden keine Implementierungen mehr gezeigt. Ich möchte statt dessen einen kurzen Überblick über die derzeitigen Funktionalitäten geben. Der volle Umfang von LARA in seiner momentanen Ausprägung umfasst ca. 100 Module und ca. 60 Prozesse im Bereich des Content Management und Repair Management.

6.1 XML oder die Suche nach dem heiligen Gral

Eine der Basisfunktionalitäten von LARA beinhaltet den einfachen Umgang mit XML mittels eines speziell für diesen Zweck ausgelegten Klasse `LXmlObject`. Die Instanzen dieser Klasse haben einen ausreichenden Funktionsumfang für das Auslesen, Modulieren, Speichern und Validieren von XML Daten. Die Daten können hierbei in unterschiedlichen

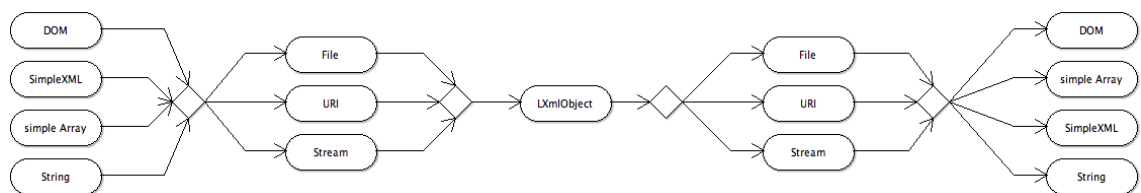


Abbildung 6.1: `LXmlObject` Formate

internen Formaten vorliegen. So ist es zum Beispiel möglich, eine DOM-Darstellung zu importieren und als reines XML oder *SimpleXML*¹ -Element wieder auszugeben. Es ist hierbei unerheblich, wohin die Daten letztendlich geschrieben werden. Die Klasse unterstützt momentan Dateien, Streams und URIs. Natürlich implementiert diese Datenabstraktion auch Funktionalitäten zum gezielten Auswerten der geladenen Daten per *xPath*² . Auch die Validierung mittels *XML-Schema*³ gehört zum Funktionsumfang.

6.2 Wer, wann, was: Logging

LARA bietet die Möglichkeit, zu jedem beliebigen Zeitpunkt Informationen gezielt in verschiedenen Log-Dateien zu speichern. So können bestimmte Abläufe dokumentiert, Performedaten gesammelt oder Fehler protokolliert werden. Das Log-System `LLogger` von LARA beherrscht verschiedenste Ziele für Log-Einträge. Von einfachen Dateien über Datenbanken bis hin zu entfernten Log-Servern oder einfacher Bildschirmausgabe oder Emails, sind dem offen konzipierten System keinerlei Grenzen gesetzt. Das Grundmodul bringt die wichtigsten Log-Funktionalitäten (so genannte *Logging-Targets*) bereits mit, kann jedoch um beliebige Prozesse erweitert werden. Eine Besonderheit des Log-Systems von LARA

¹SimpleXML: <http://de.php.net/simplexml>

²xPath: <http://www.w3.org/TR/xpath>

³XML-Schema: <http://www.edition-w3c.de/TR/2001/REC-xmlschema-0-20010502/>

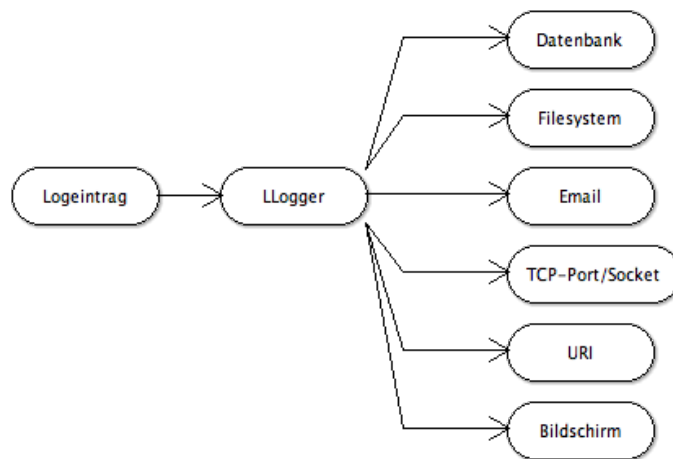





Abbildung 6.2: LLogger Logging-Targets

ist die Differenzierung der Daten: Je nachdem, welche Art von Daten verarbeitet werden sollen, können verschiedene *Logging-Targets* oder Prozesse angestoßen werden. Diese Kriterien sind ebenfalls frei erweiterbar.

6.3 Babel und der Fisch: Von ALE und SOA

Ein gutes, ausgereiftes Framework wäre natürlich nicht ganz so gut, wenn es nicht einige Möglichkeiten zum Datenaustausch mit anderen Anwendungsdomänen mitbringen würde. LARA verfolgt hierbei den Ansatz der *Serviceorientierten Architektur (SOA)*⁴ , um ein möglichst breitgefächertes Repertoire an Kommunikationsmethoden zu bieten. Neben einfachem *XML-RPC*⁵  zum Datenaustausch beherrscht LARA auch die Methodiken von *SOAP*⁶ . Die nach außen exportierten Prozesse basieren auf einem abstrakten

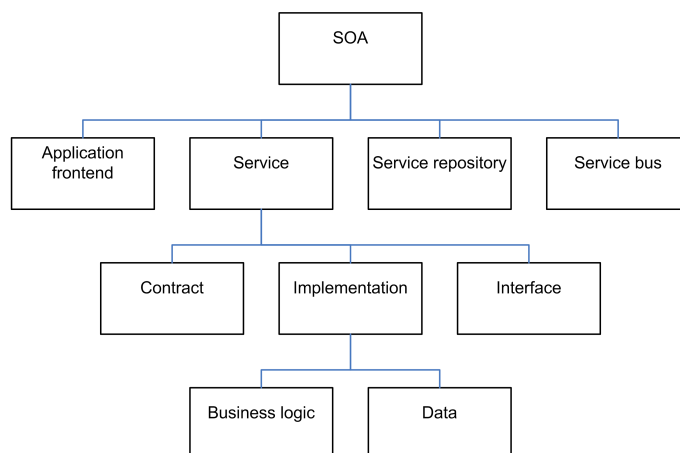





Abbildung 6.3: Serviceorientierte Architektur (SOA)

System von Basis-Services. Diese Basisklassen bedienen sowohl die Grundfunktionalität der Kommunikation und Prozessbehandlung, als auch die Authentifizierungs- und Validie-

⁴Serviceorientierten Architektur (SOA): http://de.wikipedia.org/wiki/Serviceorientierte_Architektur

⁵XML-RPC: <http://www.xmlrpc.com/>

⁶SOAP: <http://de.wikipedia.org/wiki/SOAP>

rungsmechanismen für den jeweiligen Prozess. Jeder Prozess definiert selbst seine eigenen Berechtigungen (*Access Control List*⁷ ) anhand von Rollen, Benutzergruppen, einzelnen Benutzern oder *Wildcards*⁸ . Die einzelnen Berechtigungen können hierbei zwischen *lesend*, *schreibend* und *ausführend* unterscheiden. Prozesse, die über *SOAP* angeboten werden, bringen außerdem die Prozess- und Datendefinition mittels *Web Service Description Language (WSDL)*⁹  mit. Eine Besonderheit des LARA Frameworks ist das so

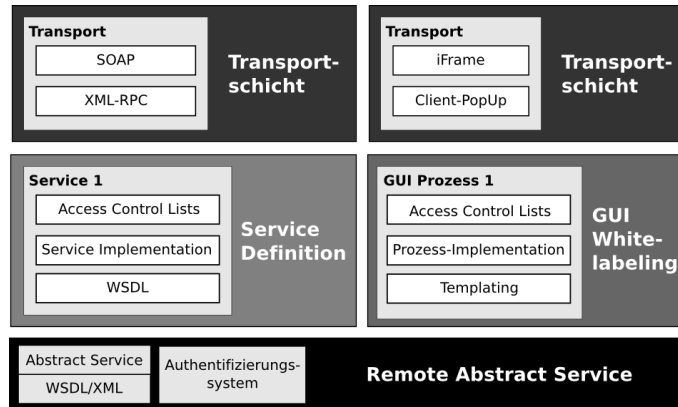




Abbildung 6.4: LARA Application Link Enabling

genannte *Whitelabeling*¹⁰ . Durch eine gezielte Abstraktion aller Prozesse in LARA ist es möglich, die jeweiligen Prozesse losgelöst von den jeweiligen Umgebungen zu exportieren. So können Prozesse für Drittabnehmer angeboten werden, ohne ein komplettes Projekt für den Kunden zu implementieren. Ausschlaggebend ist bei dieser Methode das vom Drittabnehmer abhängige Templating. Jeder Kunde kann den Prozess im eigenen Design nach eigenen Vorgaben projektieren. Änderungen am eigentlichen Kern oder den einzelnen Projektumgebungen sind hierbei nicht notwendig.

LARA stellt ebenfalls die Methodiken des *Application Link Enabling*¹¹ , kann also als Integrationsschicht zwischen verschiedenen Systemen mit gemeinsamen Datenbestand eingesetzt werden. LARA nimmt hierbei nicht nur Daten entgegen, sondern verteilt diese Daten je nach angesprochenem Prozess auf die entsprechenden Anwendungsdomänen. Momentan wird dieses Feature von mehreren großen Distributoren im Bereich des Mobilfunkmarktes benutzt, um Daten aus verschiedenen hersteller- und distributoreigenen Datenbeständen zu synchronisieren.

⁷ Access Control List: http://de.wikipedia.org/wiki/Access_Control_List

⁸ Wildcards: http://de.wikipedia.org/wiki/Wildcard_%28Informatik%29

⁹ Web Service Description Language (WSDL): <http://www.w3.org/TR/wsd.html>

¹⁰ Whitelabeling: <http://de.wikipedia.org/wiki/Whitelabel>

¹¹ Application Link Enabling: http://de.wikipedia.org/wiki/Application_Link_Enabling

7 Am Horizont

Trotz des bereits jetzt erheblich angewachsenen Stadiums des LARA-Frameworks bieten sich noch einige Kern- und Basiskomponenten zum Ausbau oder zur Erweiterung an. In diesem Kapitel möchte ich kurz auf einige wichtige, momentan in der Planungsphase befindliche Ausbaustufen eingehen.

7.1 Database Pooling

Ein weiteres wichtiges Modul betrifft den Datenbankkern von LARA. Momentan werden alle Anfragen intern auf eine persistente Verbindung zu dem jeweiligen Produktiv-Datenbankserver vermittelt. Es besteht zwar momentan die Möglichkeit, das Datenbankend (MySQL, Oracle, DB2) durch den Einsatz einer Abstraktionsschicht zu wechseln, noch ist aber nicht vorgesehen, mehrere Datenbank-Replica als Datenpool zu verwenden. Das sog. Database-Pooling würde es ermöglichen, die Anfragen und somit die Last gezielt

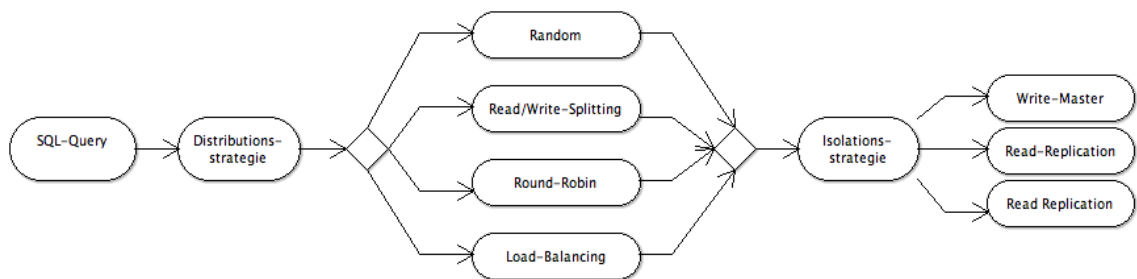



Abbildung 7.1: LARA Datenbank-Pooling


auf mehrere Datenbankinstanzen zu verteilen. Je nachdem welchen Grad der Isolation eine Anfrage erfordert, würde eine Anfrage entweder auf die Master-Datenbank oder eine der Replica laufen. Momentan ist in diesem Bereich die Implementierung von vier Verteilungsstrategien in Planung:

- **Random:** Dieses einfache Verfahren wählt nach einer `random(n)`, wobei `n` der Anzahl der vorhandenen Datenbankinstanzen entspricht, einen beliebigen Server aus.
- **Load Balancing:** Bei diesem Verfahren wird in einem bestimmten Zeitintervall die Last der sich im Pool befindlichen Datenbankinstanzen ermittelt und jeweils notiert. Diese Pool-Liste wird entsprechend der kleinsten Last zuerst sortiert. So entsteht eine rotierende Liste, bei der die jeweils am wenigsten ausgelastete Datenbankinstanz angesprochen wird.
- **Round Robin:** Die Strategie des *Round-Robin* bildet alle Abfragen sequenziell auf die vorhandenen Datenbankinstanzen ab.
- **Read/Write-Splitting:** Das komplexeste Verfahren in der Planung. Hierbei werden alle Anfrage auf ihren jeweiligen Typus hin untersucht. Als Grundsatz gilt: Schreibende Aktionen sind aufwändiger als lesende. Eine schreibende Aktion wird auf den

Write-Master weitergeleitet, während ein lesender Zugriff, je nachdem, welches Isolationslevel er beansprucht, auf eine der Datenbank-Replika umgeleitet werden kann.

Alle diese Methoden, ausgenommen das *Read/Write-Splitting*, bedingen jedoch im Backend eine so genannte *Asynchrone Multi-Master-Replication*¹ , damit alle Datenbankinstanzen als Write-Master zur Verfügung stehen. Ebenfalls besteht die Möglichkeit einer automatischen Failover-Strategie in LARA selbst. Im Falle des Ausfalls einer der Datenbankinstanzen würde einfach der nächste Server aus dem jeweiligen Pool angesprochen.

7.2 Session-Clustering

Normalerweise existieren Benutzersitzungen, so genannten Sessions, nur als Datei im Dateisystem zur internen Verwaltung von PHP selbst. Glücklicherweise stellt PHP jedoch ein Interface für die eigene Implementierung eines so genannten *Session-Handler*²  zur Verfügung. Der Session-Handler stellt eine einfache Möglichkeit dar, die Daten einer Session in die angebundene Datenbank zu schreiben. Dies bietet den Vorteil, Load-Balancing für die

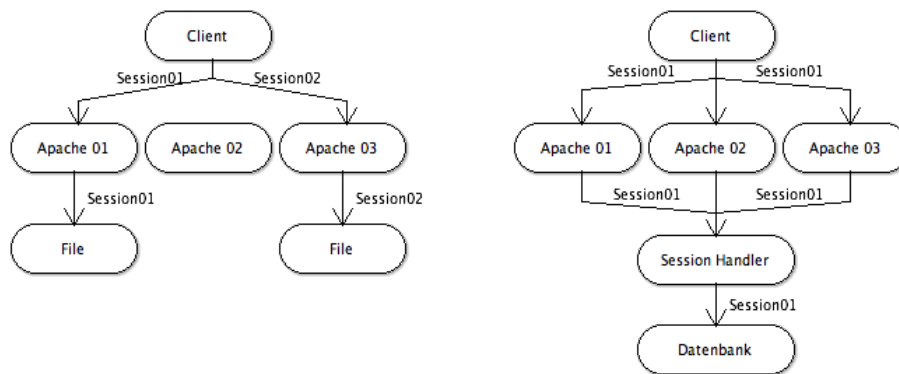



Abbildung 7.2: LARA Session-Clustering

angeschlossenen Webserver einsetzen zu können, ohne großen Implementationsaufwand auf Projektseite zu generieren. Da die Session-Daten nicht nur auf einem Server vorliegen, kann der Benutzer den Webserver wechseln (etwa bedingt durch das Apache-Load-Balancing), ohne dass seine Session verloren geht. Dieses Verfahren wird auch *Session-Clustering*³  genannt.

¹ Asynchrone Multi-Master-Replication: http://www.dbspecialists.com/presentations/mm_replication.html


² Session-Handler: <http://www.php.net/manual/de/function.session-set-save-handler.php>

³ Session-Clustering: <http://shiflett.org/articles/storing-sessions-in-a-database>

8 Fazit

Als Fazit dieser Ausarbeitung heben sich zwei Hauptbereiche hervor: Die Ausarbeitung selbst und das Thema der Ausarbeitung.

Die Ausarbeitung als solches erfolgte unter anderem im Zusammenhang der momentanen Entwicklung des LARA-Frameworks. Die hier dokumentierten Konzepte und Features dienen als Einleitung für angehende Framework-Entwickler in unserem Unternehmen. Für diesen Zweck wird diese Dokumentation noch verfeinert und erheblich erweitert, um einen umfassenden Einstieg in dieses Thema zu bekommen. Leider benötigt diese umfassende Dokumentation bei weitem mehr weißes Papier, als für diese Ausarbeitung zur Verfügung stand.

Das Thema als solches ist sehr spannend, betrachtet man die momentane Entwicklung und Umstrukturierung von Application Frameworks der heutigen Zeit. War vor fünf Jahren noch EAI der maßgebliche Standard für Entwicklungen in Unternehmen, ist es heute SOA. Direkt bemerkbar macht sich dieser Umbruch dadurch, dass die meisten von mir betrachteten Frameworks eben diesen Umstieg schmerzlich missen lassen. Die Frameworks dieser Betrachtung beschränken meist die Sichtweise ihres Zwecks auf das Programmieren von reinen Web-Anwendungen. Dies ist meiner Meinung nach nicht mehr zeitgemäß, da gerade der Trend hin zu Applikationen geht, die integriert in die Unternehmensinfrastruktur agieren. Eine Applikation ohne Standardschnittstellen wie SOAP, XML-RPC oder *UDDI*¹  ist in vielen Unternehmen nicht mehr wünschenswert oder denkbar.

Viele Unternehmen kämpfen schon heute mit erheblichen Datenmengen aus Produktions-, Service- oder Finanzbereichen, die nicht nur nicht semantisch, sondern auch nicht syntaktisch miteinander verknüpfbar sind. Das LARA-Framework soll hier gezielt Abhilfe schaffen, um neue Anwendungsdomänen und somit auch neue Kunden zu gewinnen.

Die Entwicklung von LARA als Enterprise Application Framework basiert auf jahrelanger Erfahrung in Logistik, Supply-Chain- und Repair-Management. Auch geschäftsfallorientierte Einflüsse aus dem *“after sales management”* fließen in der Entwicklung täglich mit ein und helfen, Prozesse zu optimieren und Kosten zu senken.

LARA ist für Entwickler, was LEGO für Kinder ist. . .

¹UDDI: <http://www.uddi.org/>

Index

- __PHP_Incomplete_Class, 21
- __sleep, 5, 6
- __wakeup, 5, 6
- Access Control List, 25
- Apache Struts, 3
- Application Link Enabling, 25
- Asynchrone Multi-Master-Replication, 28
- Benchmark, 11, 12
- CakePHP, 11
- Chained-Exceptions, 18
- CodeIgniter, 12
- DokumentRoot, 17
- Hibernate, 3
- import, 16
- include-Pfaden, 17
- Java™, 1–6, 16, 18
- LARA, 9, 13, 15, 16, 18, 19, 21–25, 27–29
- LClassLoader, 16, 17
- LClassLoaders, 16
- LCoreExceptions, 18
- LCoreObject, 15, 16
- LException, 18
- LExceptions, 18
- LLogger, 23, 24
- LObject, 15, 16
- Logging-Targets, 24
- LXmlObject, 23
- Magic Methods, 5
- MVC, 10
- MVC2, 10
- n, 27
- PHP, 1
- PHP Historie, 1
- PHP-Exceptions, 18
- PHP-PEAR, 12
- Prado, 11
- random(n), 27
- Reflection-Funktionalität, 16
- Scaffolding, 10
- Serviceorientierten Architektur (SOA), 24
- Session-Clustering, 28
- Session-Handler, 28
- SimpleXML, 23
- Singleton-Pattern, 16
- SOAP, 24
- Spring, 3
- StdClass, 15
- symfony, 11
- Tomcat, 3
- UDDI, 29
- Web Service Description Language (WSDL), 25
- Whitelabeling, 25
- Wildcards, 25
- XML-RPC, 24
- XML-Schema, 23
- xPath, 23
- ZendFramework, 11

Literaturverzeichnis

- [Cre07] CREDENCE, THE: *PHP frameworks - Which one is Most Suitable for you?*, August 2007. URL: <http://www.thecredence.com/php-frameworks-which-one-is-most-suitable-for-you/>.
- [Lu5] LUECKE, TIM: *Frameworks*, Februar 2005. URL: http://www.se.uni-hannover.de/documents/kurz-und-gut/ws2004-seminar-entwurf/frameworks_tluecke.pdf.
- [MA07] MEHDI ACHOUR, FRIEDHELM BETZ, ANTONY DOVGAL ET AL.: *PHP Handbuch*. PHP-Dokumentationsgruppe, April 2007. URL: <http://www.php.net/manual/de/>.
- [PHP07] PHPIT: *MVC Frameworks Written in PHP*, August 2007. URL: http://www.phpwact.org/php/mvc_frameworks.
- [Sch06] SCHMIDT, STEPHAN: *PHP Design Patterns*. Amazon.de, September 2006. URL: http://www.amazon.de/PHP-Design-Patterns-Deutsche-Ausgabe/dp/3897214423/ref=cm_taf_title_featured?ie=UTF8&tag=tellafriend-20.
- [Sin04] SINGER, LEIF: *Model View Controller (MVC)*, Dezember 2004. URL: http://www.se.uni-hannover.de/documents/kurz-und-gut/ws2004-seminar-entwurf/mvc_lsinger.pdf.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Kevin Wennemuth)

Copyright 2007 Kevin Wennemuth.

All Rights Reserved.

Aside from my professor's sole, personal review as part of his/her private, single-human, software-free grading process (checking for plagiarism with Google is acceptable), neither my professor nor my academic institution may otherwise copy, transfer, distribute, reproduce, publicly/privately perform, publicly/privately claim, publicly/privately display, or create derivative works (including „digital fingerprints“) of my copyrighted document (*intellectual property*).

The same restrictions apply to **Turnitin.com and all similar services** if my document should somehow come into their possession. Neither my professor nor my academic institution may submit my copyrighted document, in whole or in part, to be copied, transformed, manipulated, altered, or otherwise used by or stored at **Turnitin.com (iParadigms, LLC) or any other physical or electronic database or retrieval system** without my personal, explicit, voluntary, uncoerced, written permission.

Regardless of supposed intent (e.g., „to create a digital fingerprint,“), **no part of my copyrighted document** may be temporarily or permanently transferred, by any party, to Turnitin.com or any other service, program, database, or system for analysis, comparison, storage, or any other purpose whatsoever. Violators will be monetarily punished to the fullest extent allowed by the DMCA (Digital Millennium Copyright Act), the German Urheberrecht and/or international law.