I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Abkürzungsverzeichnis

MVC	Model View Controller
PHP	Hypertext Preprocessor
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
UML	Unified Modelling Language
XML	Extensible Markup Language

1 Einleitung

1.1 Die dotSource und die E-Commerce Plattform Magento

Die dotSource GmbH in welcher der Autor arbeitet befasst sich seit 2006 mit der Entwicklung von E-Commerce Lösungen. Ein Teilbereich der Firma entwickelt Software mithilfe von Magento. In diesem, auf der Skriptsprache PHP (Hypertext Preprocessor) basierenden Shopsystem, kommen vielzählige Design Patterns zu Einsatz.

1.2 Zielsetzung der Arbeit

Im Bereich des E-Commerce gibt es verschiedene Design Patterns (auch Entwurfsmuster genannt). Ziel ist es nun dem Leser eine anschauliche Übersicht verwendeter Design Patterns in Magento zu geben. Diese Projektarbeit wird sich dabei mit den Vor- und Nachteilen von Design Patterns befassen. Dabei werden einige in Magento verwendete Entwurfsmuster genannt und deren Verwendung aufgeführt. Zur Unterstützung werden Programmierabschnitte aus Magento und UML Diagramme der jeweiligen Verwendung des Design Patterns hinzugefügt. Im weiteren Verlauf der Arbeit wird ein Magento Request unter Betrachtung der verwendeten Entwurfsmuster und deren Zusammenspiel dargestellt.

2 Patterns in der IT

2.1 Was sind Design Patterns

„Entwurfsmuster lösen bekannte, wiederkehrende Entwurfsprobleme. Sie fassen Design und Architekturwissen in kompakter und wiederverwertbarer Form zusammen“[1] und sind von Entwicklern verfasstes, geteiltes Wissen zur Umsetzung von Softwareprojekten. Dabei wird das gesamte Projekt in kleinen Teilkomponenten betrachtet und für diese Teile entsprechende, sich in der Praxis bewährte, Entwurfsmuster zur Konzeption verwendet.[2] Die entstandenen Design Patterns erstrecken sich von Hinweisen zu Datenbanken (wie dem Sperren von Datensätzen bei Updates), über die Kommunikation zwischen Server und Client (beispielsweise mit Data Transfer Objekten), bis hin zur Trennung von Benutzerschnittstelle, grafischer Ausgabe und Business Logik in Form des MVC-Patterns. Die festgehaltenen Entwurfsmuster sind sehr vielfältig und können in unterschiedliche Kategorien gegliedert werden.[3] Eine beispielhafte, in dieser Arbeit verwendete Kategorisierung, stellt die folgende dar:

- Erzeugungsmuster
- Verhaltensmuster und
- Strukturmuster

Erzeugungsmuster werden verwendet um die Art und Weise der Instanziierung (Erzeugung) von Objekten zu beeinflussen. Dabei wird der eigentliche Instanziierungsprozess abstrahiert, sodass verschiedene Ausprägungen von Objekten in einer Funktion generiert werden können.[4] Es werden die nachfolgenden Entwurfsmuster genannt:

- Factory Method

---

[1] [Eil10], S. 1

[2] [Fow03], S. 1 f.

[3] [Eil10], S. V f.

[4] [Gam06], S. 101

- Singleton und
- Registry

Verhaltensmuster wiederum beschreiben die Interaktion zwischen den Objekten. Die Elemente des Entwurfsmusters sind dabei meist lose gekoppelt. Dies bedeutet, dass sie keine starke Abhängigkeit zu den aufrufenden Teilen des Systems besitzen. Dies begünstigt eine einfache Erweiterung des Systems.[5] Die in dieser Arbeit genannten Verhaltensmuster sind:

- Command
- Template Method und
- Observer

Die letzte genannte Kategorie von Entwurfsmustern sind die Strukturmuster.
Sie „befassen sich mit der Komposition von Klassen und Objekten, um größere Strukturen zu bilden.“[6] Sie können Klassenhierarchien oder auch Schnittstellen beschreiben. Das verwendete Beispiel ist das MVC Pattern. Es wird in Kapitel *2.4 Das MVC Pattern* erläutert.

2.2 Vorteile

Wie bereits erwähnt sind Design Patterns von Entwicklern für ebensolche verfasst. Der Grund dafür besteht meist in dem Wissenstransfer, welcher durch die Weitergabe von praktischen Erfahrungen und angewandtem Wissen besteht.[7] Eine Anwendung solcher Entwurfsmuster bedeutet eine Wiederverwendung erprobter Designentscheidungen. Des Weiteren nimmt das Suchen nach dem richtigen Design Pattern weniger Zeit in Anspruch, als durch mehrere Refaktorierungszyklen selbst entdeckte Lösungsmöglichkeiten.[8]

---

[5] [Gam96], S. 271 f.

[6] [Gam96], S. 169 f.

[7] [Eil10], S. 165

[8] [Sch09], S. 155

2.3 Nachteile

Durch die Verwendung von Design Patterns ist es möglich, dass die Anzahl an Entwicklungscode und Komplexität zunimmt. Es sollte darauf geachtet werden, dass Entwürfe einfach und verständlich gehalten werden.[9] Weiterhin kann es passieren, dass ein Pattern nicht genau als Lösung für ein Problem verwendet werden kann.

2.4 Das MVC Pattern

Softwareprojekte haben meist die Anforderung flexibel in der Entwicklung zu sein. Ein Lösungsansatz dafür ist es den Klassen in der Software klare Aufgabenbereiche zu geben. Dazu dient das Model View Controller Pattern (MVC). Dieses, zu den Strukturmustern gehörende, Design Pattern trennt Verantwortlichkeiten in:

- Datenverarbeitung (Model),
- die Kontrolle von Benutzereingaben (Controller) und
- der Darstellung (View). [10]

Das Design Pattern selbst besteht aus eine Zusammensetzung mehrerer Entwurfsmuster.[11] Der Controller ist die erste Instanz in einem MVC Seitenaufruf. In ihm wird beispielsweise die Zugangsberechtigung für eine Seite geprüft und im Bedarfsfall auf eine andere Seite umgeleitet. Im Model wird die Datenhaltung und Aufbereitung durchgeführt. Diese Daten werden dann von den beiden anderen Komponenten verwendet. Es kann beispielsweise nach dem Observer Pattern funktionieren. Dieses wird Kapitel *2.7 Das Observer Pattern* näher erläutert. Die View beinhaltet alle visuellen Elemente wie HTML Ausgaben mit Formularen, Bildern oder auch Frames. Sie kann sich nach dem Template Pattern

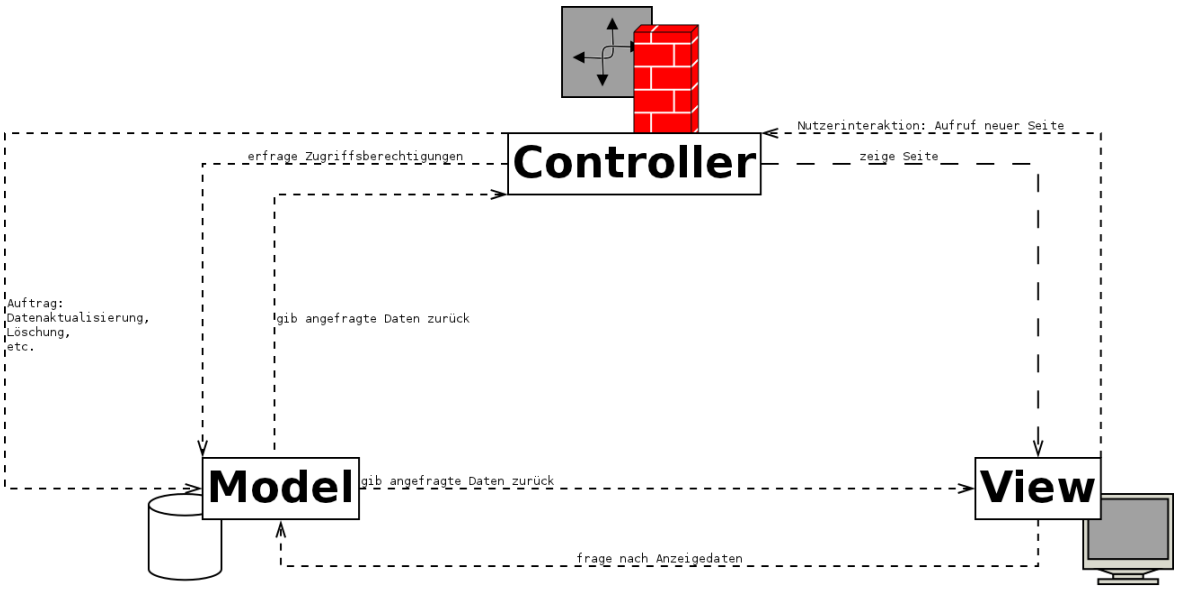
[9] [Eil10], S. 2

[10] [Eil10], S. 77 ff.

[11] [Fre06], S. 534 f.

richten. Dieses Entwurfsmuster wird in Kapitel *3.1 Beispielrequest von Magento* erläutert. Die folgende Abbildung verdeutlicht das Zusammenspiel der einzelnen Komponenten innerhalb des MVC Patterns.

Abbildung 1: MVC - Zusammenspiel der Komponenten



Quelle: eigene Abbildung

Die dabei entstehenden Vorteile dieses Design Patterns liegen in der klaren Trennung von Abhängigkeiten. Will man beispielsweise etwas am Design ändern, so kann dies unabhängig der Datenaufbereitung (Model) geschehen. Weiterhin kann man eine View unterschiedlichen Models zuordnen (z.B. Produktliste, Wunschliste, . . .) wodurch sich der Wiederverwendungswert erhöht und Coderedundanzen vermieden werden. Nachteil an diesem Verfahren ist der Mehraufwand an Implementierungen. Ebenso gibt es einen Anspruch dieses Verfahrens „allen eventuellen Anforderungen gerecht zu werden“[12]. Das bedeutet, dass viele Eventualitäten im Entwurf der MVC Komponenten bedacht werden müssen, da beispielsweise die Views keine Daten nachladen können und auf die korrekten Übergaben der Models angewiesen sind.

[12] [Eil10], S. 79

Magento nutzt das MVC Pattern ebenfalls zur Strukturierung von Zugriffskontrolle, Verarbeitung und Ausgabe. In *Kapitel 3.1 Beispielrequest von Magento* findet sich eine Seitenanfrage mit Erklärungen zur Verwendung des MVC Patterns.

2.5 Die Design Patterns: Singleton und Registry

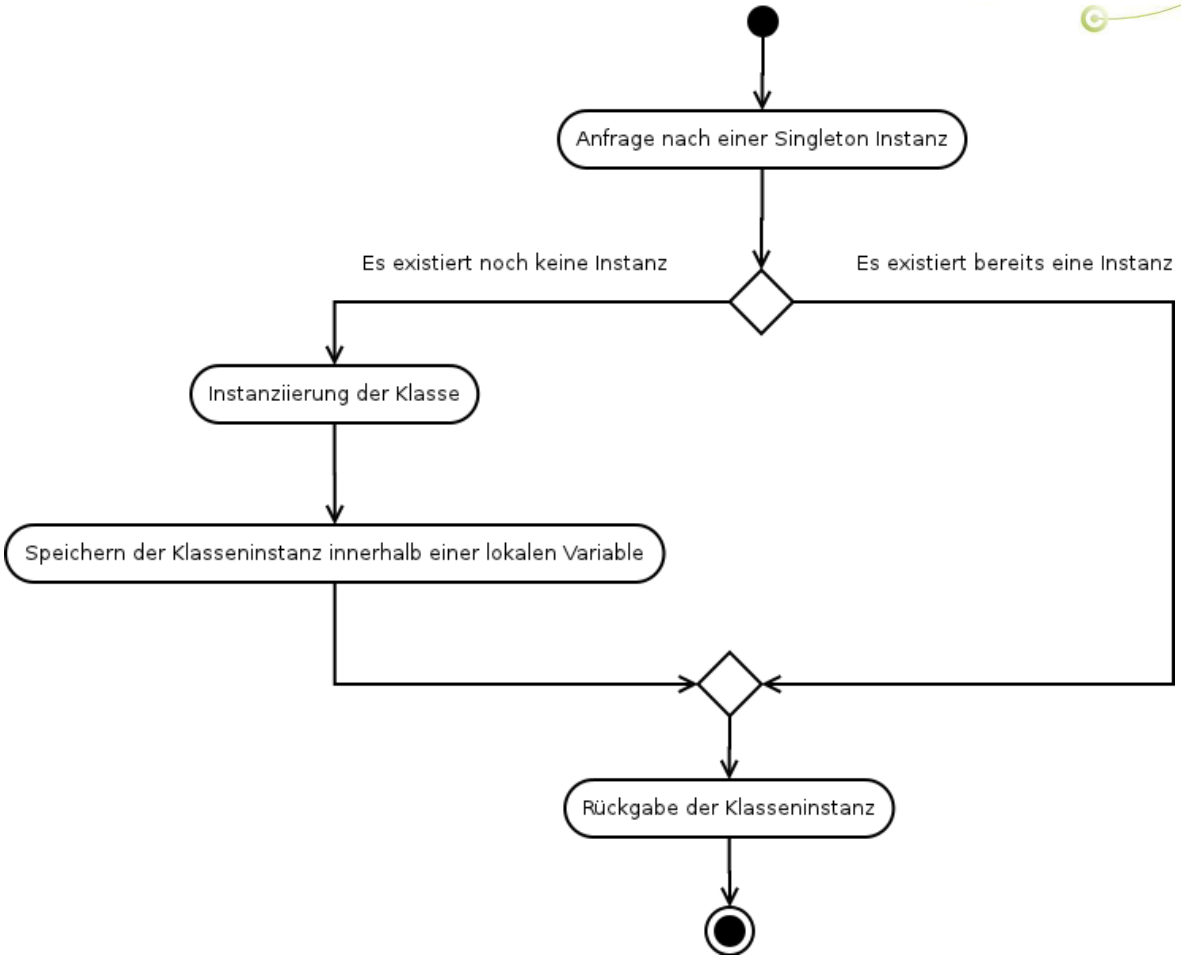
Mithilfe des Singleton Patterns existiert eine Umsetzungsstrategie für das Erstellen einer einzigen Instanz von Objekten. Es zählt zu den Erzeugungsmustern. Bei Anfrage nach einer Objektinstanz wird zuvor abgefragt ob bereits eine Instanz existiert, im Bedarfsfall generiert und zurückgegeben. Dabei kann diese Logik auf zwei verschiedene Arten implementiert werden[13]. Durch Nutzung:

- einer „Instanziierungsklasse“, welche nach Existieren einer Instanz prüft oder
- der eigenen Klasse, d.h. Privatisieren des Konstruktors und Aufruf einer statischen *::getInstance()* Methode.

Die folgende Abbildung symbolisiert einen Aufruf des Singleton Patterns mit der Entscheidung zur Rückgabe des bereits erstellten Objekts und eventueller vorheriger Instanziierung.

[13] [Eil10], S. 37 f.

Abbildung 2: Singleton - Erstellung und Rückgabe einer Instanz



Quelle: eigene Abbildung

In Magento ist die Implementierung des Entwurfsmuster über das Verfahren der Instanziierungsklasse gegeben. Die Anfrage zur Instanziierung geschieht über die Klasse *Mage*. Über diese werden die Erstellung und der Aufruf für Helper und Models gesteuert. Die nachfolgende Abbildung zeigt einen Ausschnitt der Implementierung.

Abbildung 3: Singleton - Magento Rückgabe einer Instanz

```
/* @var $catalogHelper = Mage_Catalog_Helper_Config */
$catalogHelper = Mage::helper('catalog/config');

/* @var $product = Mage_Catalog_Model_Product */
$product = Mage::getSingleton('catalog/product');
```

Quelle: eigene Abbildung

Der Vorteil dieses Entwurfsmusters besteht darin, dass weniger Speicherauslastung stattfindet, da nur eine Instanz zur Laufzeit erstellt wird.[14] Jedoch kann diese Logik umgangen werden, wenn, wie in Magento implementiert, die Instanziierung nicht über den privaten Konstruktor, sondern über Instanziierungsklassen gelöst wird.

Werden die Singleton Objekte über die gesamte Anwendung benötigt, so kann dieses Entwurfsmuster um das nachfolgende Registry Pattern erweitert werden. Dabei bedient sich die Registry eines zentralen Zugriffspunkts. Das Setzen und Erhalten von Registrydaten in Magento findet mit dem Aufruf der folgenden Methoden statt:

Abbildung 4: Registry - Magento Setzen und Erhalten der Objekte

```php
// Set product in registry.
Mage::register('current_product', $product);

// ...
// ~
// ...

// Call product from registry.
/* @var $product Mage_Catalog_Model_Product */
$product = Mage::registry('current_product');

echo $product->getName();
```

Quelle: eigene Abbildung

Verwendung findet dieses Pattern beispielsweise im Umgang mit MVC. Dabei kann der Controller die aktuell anzuzeigenden Produktdaten in die Registry ablegen. Innerhalb der View wird dann auf diesen globalen „Datenpool“ zugegriffen. Der Vorteil dieses Verfahrens besteht in der Einsparung von Funktionsparametern. Benötigte Daten einer Funktion müssen nicht mehr übergeben, sondern können innerhalb dieser selbst geladen und verarbeitet

[14] [Sch09], S. 164 ff

werden. Ein Nachteil besteht in der verlorengegangenen Datenkapselung. [15] So wäre es beispielsweise in der MVC View möglich Daten eines Models abzuändern.

2.6 Das Method Factory Pattern

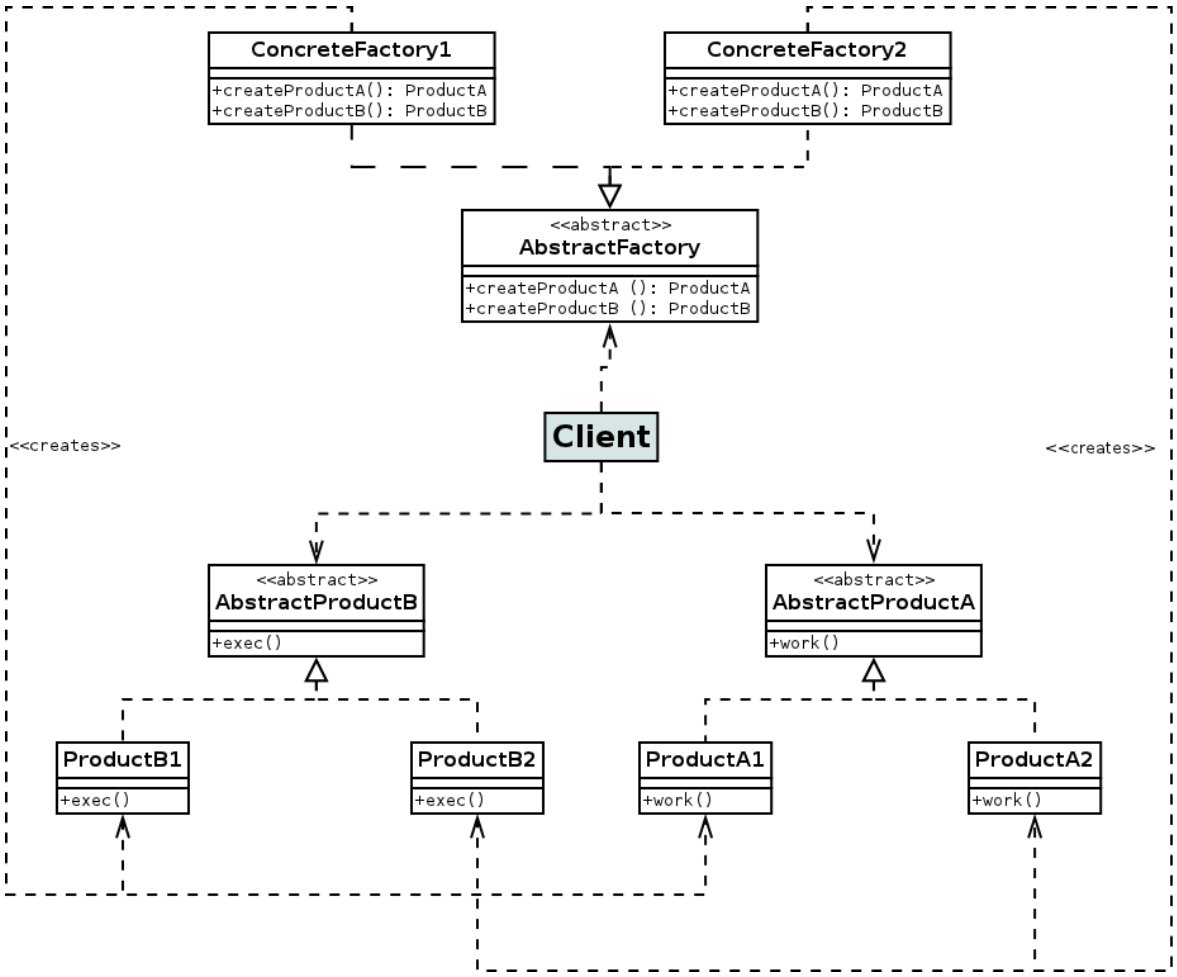
Soll der Instanziierungsprozess von Klassen zentral verwaltet werden, so bietet sich die Verwendung dieses, zu den Erzeugungsmustern gehörenden, Design Patterns an. Es beeinflusst die beiden Fragen:

- Was ist zu erzeugen und
- Wie ist es zu erzeugen[16]

Realisiert wird es mithilfe von Factory Methoden (Fabrikmethoden). Dabei werden Klassen nicht mehr direkt instanziiert, sondern über die Factory Schnittstelle erzeugt. Innerhalb dieser Funktionen werden benötigte Parameter zur Erstellung der gewünschten Klassen bearbeitet. Des Weiteren kann anhand des Aufrufs der Factory entschieden werden, von welchem Typ die Rückgabeklasse sein soll. Dazu ist es nötig, dass die Factory Methode als Rückgabewert eine abstrakte Klasse beinhaltet, in der Implementierung jedoch auch eine konkrete Klasse zurückgibt.[17]
Die folgende Abbildung beinhaltet einen Entwurf des Factory Method Patterns:

[15] [Sch09], S. 402 f.
[16] [Eil10], S. 33
[17] [Sch09], S. 173 ff.

Abbildung 5: Factory - Komponenten



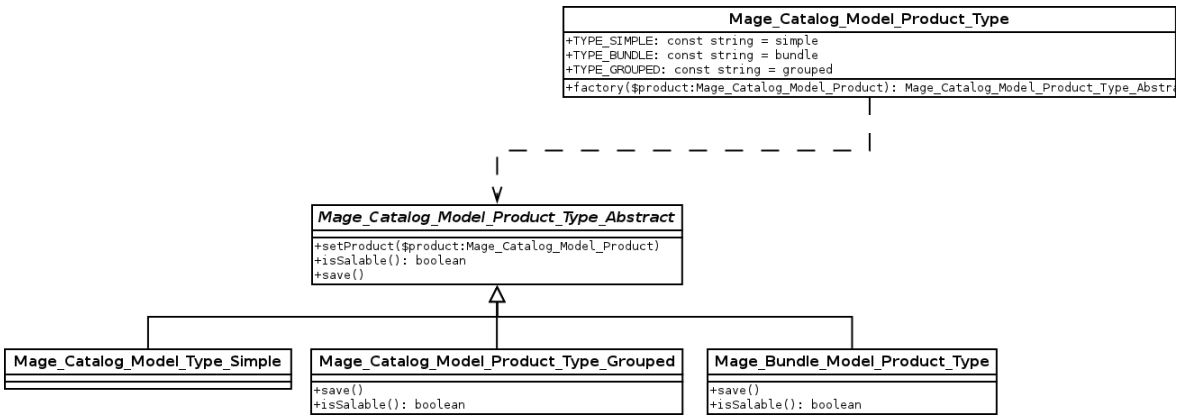
Mit Änderungen entnommen aus: [Sch09], S. 192

In diesem Beispiel ist die Erstellung einer Instanz von AbstractProductA und AbstractProduktB abhängig von der jeweiligen Verwendung einer der beiden Factory Klassen. Durch Verwendung einer der Beiden wird die Frage des „Was wird instanziiert“ beantwortet. Des Weiteren wird innerhalb der einzelnen Factory Methoden die Datenübergabe an die zu erstellenden Objekte realisiert („Wie ist es zu erzeugen“). Dies wären z.B. Anfangsbelegungen, welche an den Konstruktor einer neu zu erstellenden Klasse übergeben werden können.

Magento nutzt dieses Pattern beispielsweise bei der Instanziierung von Produkttypen. Es unterscheidet zwischen Typen wie Simple-, Bundle- oder

Grouped Produkten. Beim Aufruf der Factory Methode wird das aktuelle Produkt übergeben und anhand dessen Daten der entsprechende Produkttyp instanziiert („Was wird erstellt“). Anschließend wird diesem das eigentliche Produkt selbst zugeordnet („Wie wird es erstellt“).
Die folgende Abbildung repräsentiert die Abhängigkeit der unterschiedlichen Produkttypen.

Abbildung 6: Method Factory - Magento Produkttypen



Quelle: eigene Abbildung

Nachfolgend wird ein Aufruf der Factory Methode dargestellt. Wie bereits erwähnt wird innerhalb dieser die Instanziierung der weiteren Produkttypen Klasse anhand der übergebenen Produktdaten ermittelt.

Abbildung 7: Method Factory - Magento Aufruf

```
Mage::getSingleton('catalog/product_type')->factory($this);
```

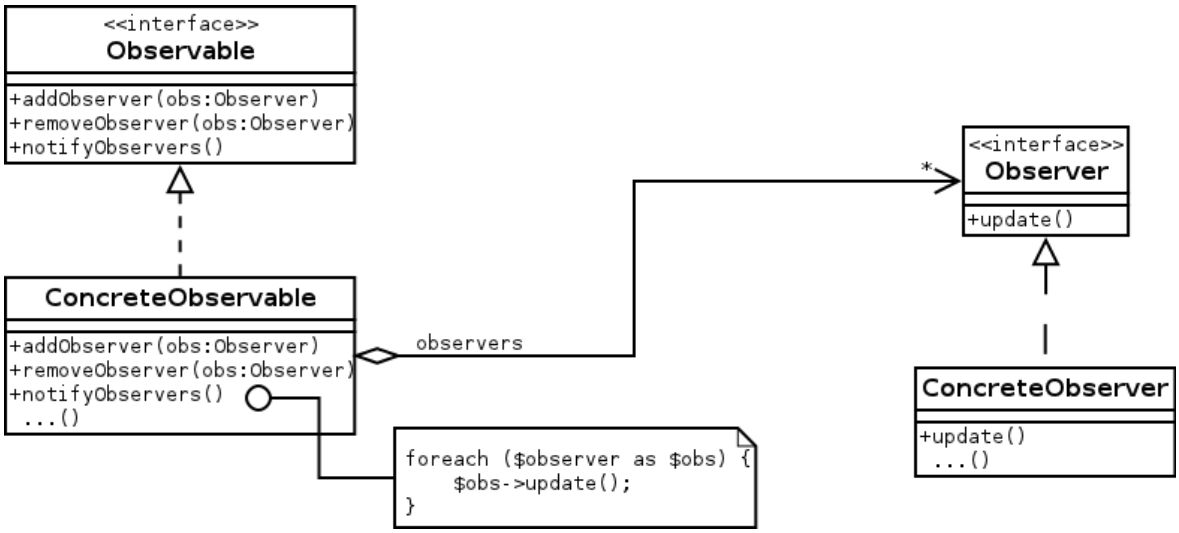
Quelle: eigene Abbildung

Ein Vorteil dieses Verfahrens liegt in der zentralen Verwendung. Ändern sich Konstruktor-Parameter der zu instanziierenden Klasse, so muss diese Änderung nicht in allen Aufrufen der Klasse, sondern nur in den Factory Methode(n) geschehen. Ein Nachteil besteht in der schweren Erweiterbarkeit. Muss beispielsweise eine weitere Rückgabeklasse zu einer Factory Methode hinzukommen, so müssen sämtliche, aufrufenden Klassen ebenfalls abgeändert werden, um weiterhin das zuvor gewünschte Objekt zu erhalten.

2.7 Das Observer Pattern

Wird in Magento ein Produktpreis geändert, so ergibt sich daraus eine Anpassung verschiedener Komponenten. Die Änderung kann beispielsweise den Warenkorb anderer Nutzer oder auch die Rabattberechnung des Produkt selbst beeinflussen. Kommen nun noch Weitere hinzu kann der entsprechende Quelltext schnell unübersichtlich und stark abhängig zu den einzelnen anzupassenden Komponenten werden. Dem entgegenwirken soll das Observer Pattern. Es gehört zu den Verhaltensmustern, da es die Zustands- oder auch Verhaltensänderungen von Objekten bekannt macht. Die sogenannten Observer (Überwacher) werden dazu innerhalb einer zentralen Stelle bestimmten, sogenannten Events, zugeordnet. Nun kann ein Objekt ein solches Event aufrufen und alle registrierten Observer werden über diesen Aufruf informiert. Die nachfolgende Abbildung stellt die Zusammenhänge zwischen Observer und der zentralen Aufrufschnittstelle dar.

Abbildung 8: Observer - Komponenten



Mit Änderungen entnommen aus: [Eil10], S. 62.

In Magento besteht die Nutzung eines Observers aus drei Teilen:

Zur Registrierung der zugehörigen Observer dient ein Eintrag in der config.xml des jeweiligen Moduls (nachfolgende Abbildung).

Abbildung 9: Observer - Magento XML Deklaration

```
<config>
    <global>
        <events>
            <alter_product_price>
                <observers>
                    <catalog_product_alter_product_price>
                        <class>catalog/observer</class>
                        <method>catalogProductAlterPrice</method>
                    </catalog_product_alter_product_price>
                </observers>
            </alter_product_price>
        </events>
    </global>
</config>
```

Quelle: eigene Abbildung

Das Starten eines Events genügt einem einzigen Funktionsaufruf. Dieser ist in der folgenden Abbildung dargestellt.

Abbildung 10: Observer - Magento Event werfen

```
Mage::dispatchEvent('alter_product_price', array('product' => $product));
```

Quelle: eigene Abbildung

Nachdem der Observer deklariert (config.xml) und das Event gestartet wurde, werden alle Überwacher aufgerufen. Eine Beispielfunktion kann der nachfolgenden Abbildung entnommen werden.

Abbildung 11: Observer - Magento Event abfangen

```php
<?php
class Mage_Catalog_Model_Observer
{
    /**
     * Add new price on event throwing.
     *
     * @param Varien_Event_Observer $observer
     */
    public function catalogProductAlterPrice(Varien_Event_Observer $observer)
    {
        $oldPrice = $observer->getProduct()->getPrice();
        $additionalPrice = microtime();

        $observer->getProduct()->setPrice($oldPrice + $additionalPrice);
    }
}
```

Quelle: eigene Abbildung

Vorteil dieses Entwurfsmusters ist die lose Kopplung der Observer zu dem aufrufenden Prozess. Sie sind nur über den Aufruf des aktuellen Events verbunden. Ein Nachteil dieses Verfahrens ist, dass bei steigenden Zahlen an Beobachtern die Komplexität und Überwachung der Datenänderungen immer schwerer wird.[18]

2.8 Vielzählige Entwurfsmuster

Magento selbst basiert auf dem Zend Framework. Dieses ist wiederum beinhaltet ebenfalls vielfältige Patterns, welche zum Teil von Magento benutzt werden. Dazu gehört z.B. das bereits erwähnte MVC Pattern. Weitere, noch nicht genannte Design Patterns sind:

- Adapter
- Dekorator
- EAV
- Front Controller
- Iterator

[18] [Eil10] S. 63

- usw.

Zur Definition und Anwendung dieser Entwurfsmuster sei auf weiterführende Lektüre verwiesen.

3 Zusammenspiel von Design Patterns in Magento

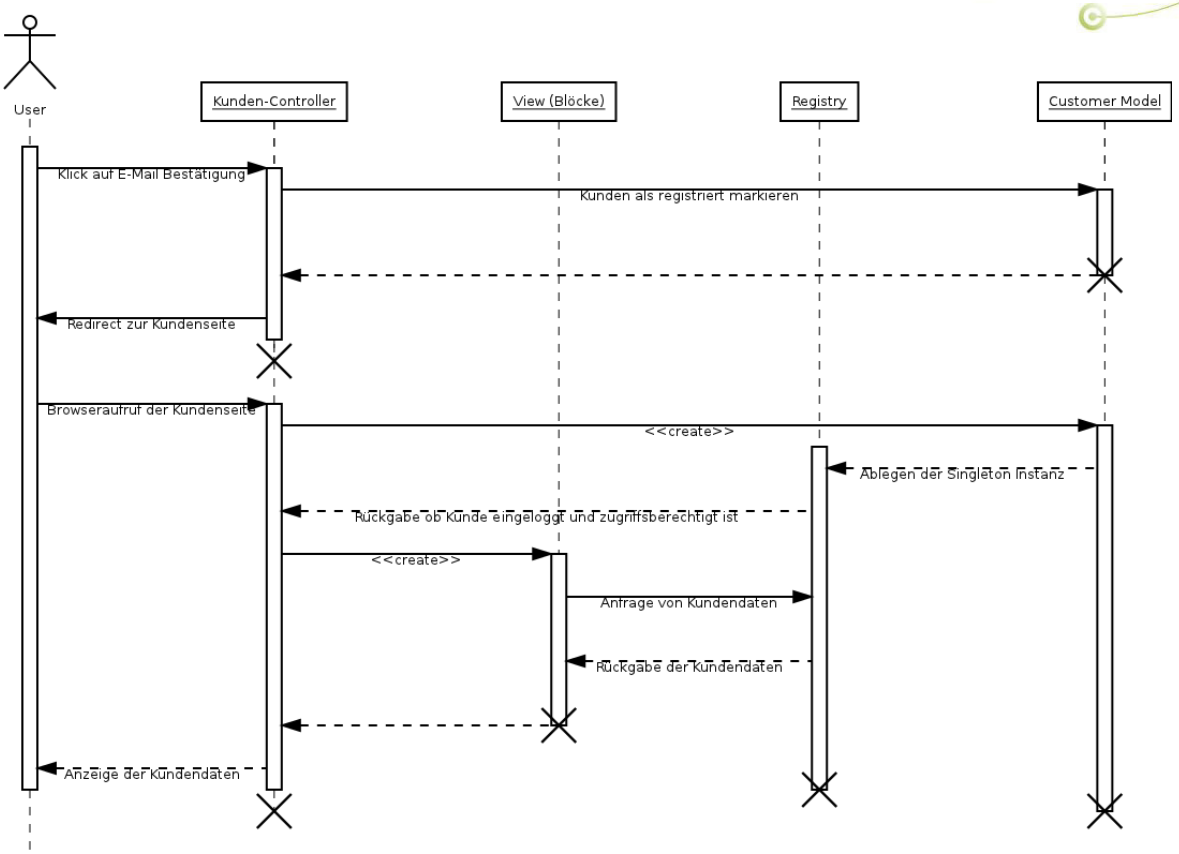
3.1 Beispielrequest von Magento

Im Folgenden wird das Zusammenspiel einzelner Design Patterns in einem Magento Request dargestellt. Es soll darlegen, dass bereits in einem einzigen Aufruf von Magento verschiedene Design Patterns zum Einsatz kommen. Das Beispiel bezieht sich auf die Aktivierung eines Kunden mit anschließendem Aufruf der Kundenseite. Es werden die Entwurfsmuster:

- MVC
- Singleton
- Registry
- Template

Genannt. Die nachfolgende Abbildung verdeutlicht das Zusammenspiel der einzelnen Elemente im MVC Pattern.

Abbildung 12: Magento - Seitenaufruf mit Design Patterns



Quelle: eigene Abbildung

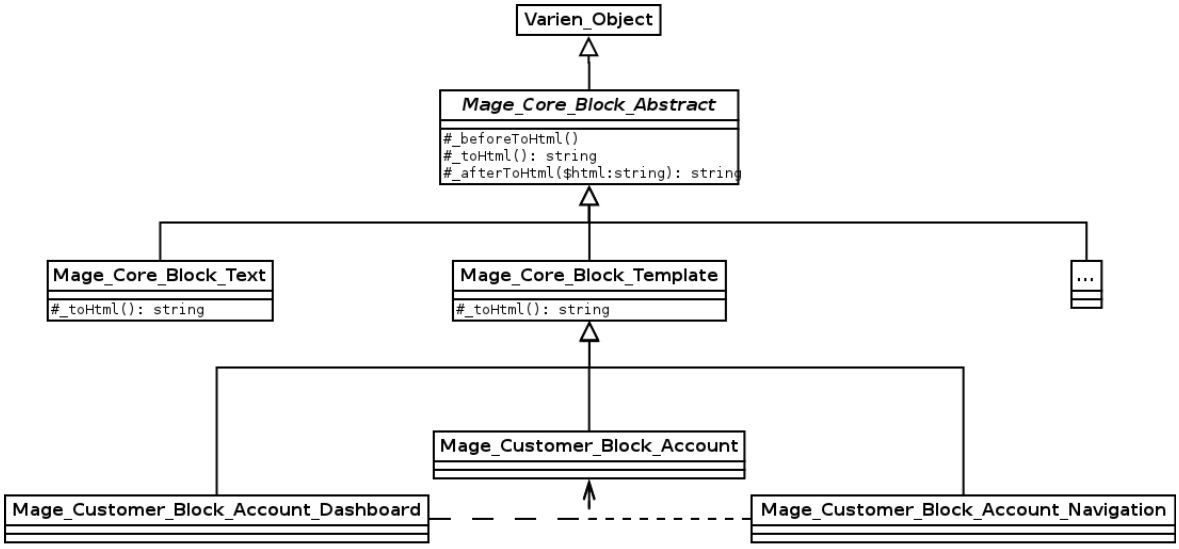
Im ersten Aufruf wird ein Customer Model innerhalb des Controllers instanziiert. Der Controller fragt beim Model nach Richtigkeit der Anfragedaten und übergibt diesem im Erfolgsfall die neuen Daten, woraufhin diese im Model bearbeitet und gespeichert werden.

Ist dieser Vorgang abgeschlossen, so leitet der Controller auf die Kundenseite um. Dabei wird zuerst eine Singleton Instanz des aktuellen Kundenmodels generiert und in die Registry abgelegt, sodass im weiteren Verlauf die Views auf die entsprechenden Daten wie Nutzernamen, E-Mail usw. zugreifen können.

Die Views wiederum wurden mit dem Template Pattern erstellt.
Beim Template Pattern geht es darum konkrete Implementierungen von Algorithmen, sprich Lösungen, an Unterklassen zu delegieren. Dies wird realisiert

indem ein Skelett (Template) mit abstrakten- oder auch Interfacemethoden erstellt und diese zur eigenen Verwendung überschrieben werden können.[19] Die nachfolgende Grafik zeigt die Vererbungsstrategie für Views in Magento am Beispiel der Kundenseite.

Abbildung 13: Magento - Views mit Template Pattern



Quelle: eigene Abbildung

Die hier genannten Funktionen _beforeToHtml(), _toHtml() und _afterToHtml() können von den einzelnen Unterklassen überschrieben und den eigenen Bedürfnissen entsprechend geändert werden. Die abstrakte Klasse Mage_Core_Block_Abstract stellt dafür ein Konstrukt von verwendbaren Funktionen zur Verfügung.

3.2 Magentoerweiterung um das Command Pattern

In einem aktuellen Projekt der dotSource existieren Schnittstellenanfragen an externe Systeme. Dabei kommen Protokolle wie SOAP, REST oder auch XML zum Einsatz. Diese stellen unterschiedliche Anfragen an Drittsysteme dar. Um trotz der Differenzen der Anfrageprotokolle eine einheitliche Verwendung zu

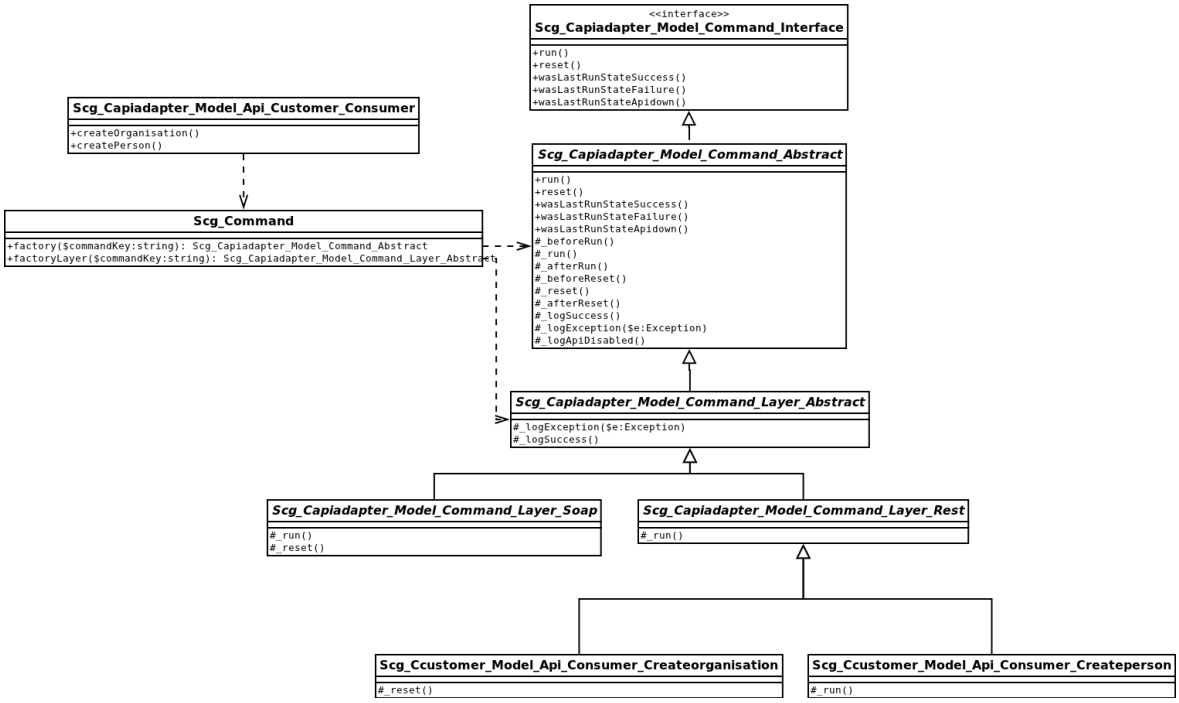
[19] [Gam96], S. 366 ff.

realisieren wurden die Schnittstellen unter Nutzung des Command Pattern implementiert. Durch dessen Gebrauch können Problemstellungen wie:

- Einzelne Anfragen unterschiedlicher Handhabung
- Log Mechanismen oder auch
- Reset Operationen bei Fehlschlagen der Operation

Realisiert werden. Dazu wird je Command eine Klasse erzeugt. Diese besitzt nur wenige, nach außen hin sichtbare Funktionen. Da diese über abstrakte Klassen definiert werden können, ist es möglich den internen Ablauf der Command Funktionen einheitlich zu verwenden. Ebenso ist dem aufrufenden Objekt die Logik des Commands unbekannt,[20] was unterschiedliche Implementierungen innerhalb der privaten Methoden begünstigt. In der nächsten Abbildung kann ein Klassendiagramm eines Command Patterns eingesehen werden. Die Klassen wurden der aktuellen Implementierung entnommen.

Abbildung 14: Magentoerweiterung - Command Pattern



Quelle: eigene Abbildung

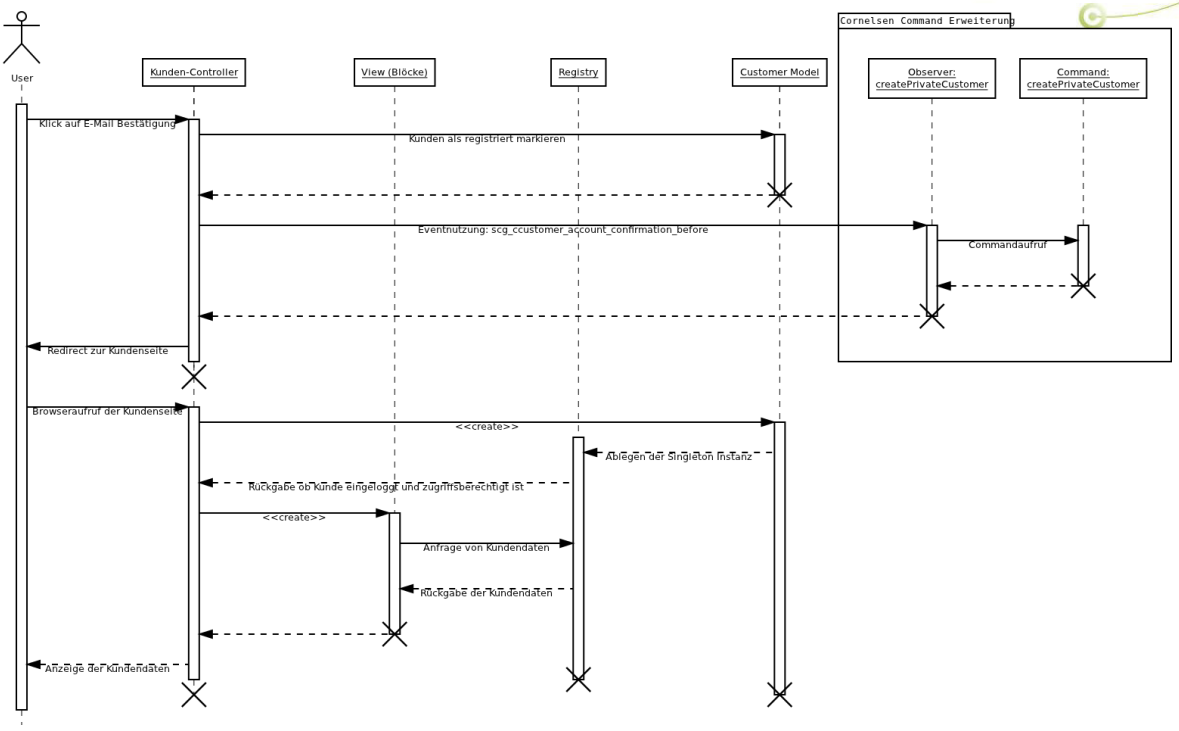
[20] [Gam96], S. 273 ff.

Der Funktionsablauf beginnt mit der Instanziierung des Commands. Anschließend wird die *run()* Methode aufgerufen. Durch die Implementierung der *_beforeRun()*, *_afterRun()* und Logging Aufrufe innerhalb dieser Funktion wurde eine Möglichkeit der einheitlichen Implementierung innerhalb des Aufrufs der *run()* Methode geschaffen. Ist eine Anfrage gescheitert, so kann die *reset()* Funktion aufgerufen werden.

3.3 Erweitertes Zusammenwirken der Entwurfsmuster von Magento

Im Folgenden wird der Verlauf des bereits zuvor genannten Request der Kundenregistrierung in Erweiterung einer Drittsystemanfrage eines Kudnenprojekts dargestellt. Das Augenmerk wird dabei auf das Zusammenspiel des neu verwendeten Command Patterns mit den bereits existierenden Entwurfsmustern gelegt. Die Erweiterung beginnt nach dem Speichern des Kunden mit dem Status „registriert“. Daraufhin wird das Event *scg_ccustomer_account_confirmation_before* geworfen. Dieses wird von einem Modul mittels Observer abgefangen und zur Kommunikation mit einem Drittsystem verwendet. Die nachfolgende Abbildung soll die einfache Integration weiterer Programmierlogik mithilfe von Design Patterns symbolisieren.

Abbildung 15: Magentoerweiterung - Seitenaufruf mit Design Patterns



Quelle: eigene Abbildung

In den entsprechenden Kapiteln wurde bereits erwähnt, dass die Entwurfsmuster einer Problemlösung entsprechen. Durch diese strikte Aufgabentrennung ist es einfach Weitere zu implementieren.

In diesem Beispiel ist es zum Einen das Observer Pattern. Dieses wird integriert durch einen einzigen Funktionsaufruf innerhalb bestehender Logik. Wird nun dieses Design Pattern in Magento benötigt, so kann es ungebunden vom eigentlichen Prozessfluss aufgerufen, ausgeführt und die Prozesssteuerung anschließend wieder abgegeben werden.

Der implementierte Observer führt wiederum eine Anfrage an ein externes System mittels Command Pattern durch. Die in Kapitel *3.2* Magentoerweiterung um das Command Pattern implementierte Logik ermöglicht es die abstrakte Logik mittels wenigen Änderungen zu erweitern. Diese bestehen aus Ändern der Anfrage (*run()*) ebenso wie der Anpassung des Fehlerlogging (*_logException()*) und Rücksetzen im Fehlerfalle (*reset()*).

3.4 Design Patterns, wie geht es weiter

Die in dieser Arbeit genannten Design Patterns stellen nur einen geringen Teil der bereits in Fachliteratur dokumentierten Entwurfsmuster dar sich, wie bereits mehrfach in dieser Projektarbeit erwähnt, die Verwendung nach der jeweiligen Problemstellung des Projekts richtet. Im Weiteren sei der Leser auf weiterführende Lektüre zu diesen Themen verwiesen. In *Anlage A: Design Patterns, Kurzbeschreibung und Literaturvorkommen* Können nochmals die in dieser Arbeit genannten Design Patterns in einer kurzen Übersicht mit dem jeweiligen Literaturvorkommen eingesehen werden.

4 Fazit

Design Patterns in der Softwareentwicklung sind verfasstes Wissen anderer Entwickler zur Lösung eines Softwareproblems. Sie stellen eine Vereinfachung im Entwurf von Architekturen dar. Kann man ein Entwurfsmuster der Problemstellung des eigenen Projektes zuordnen, so bietet sich eine Verwendung an. Jedoch nimmt ein Entwurfsmuster nicht die eigentliche Implementierung ab, welche eine dynamische Arbeit mit Unterstützung von Patterns darstellen kann. Jedoch bieten sie bei bewusster Anwendung den Vorteil, bereits durchdachte Lösungsansätze nicht erneut konzipieren zu müssen.

Anlage A: Design Patterns, Kurzbeschreibung und Literaturvorkommen

Design Pattern	Kurzbeschreibung	Literaturvorkommen (S. ff.)		
		Eil10	Gam96	Sch09
Command	Kapselung eines Befehls als eine Klasse.	43	273	284
Factory Method	Indirekte Instanziierung von Klassen je nach übergebenen Daten der *factory()* Methode.	31	131	173
MVC	Trennung von Zugriffskontrolle (M - Model), Darstellung (V - View) und Datenhaltung/ -Manipulation (M - Model).	77	5	354
Observer	Dient der Bekanntmachung von Änderungen an alle registrierten Überwacher.	61	287	266
Registry	Ermöglicht globalen Zugriff auf ein Singleton-Objekt innerhalb der Software.	152	-	396
Singleton	Einmalige Instanziierung einer Klasse innerhalb der Software.	35	157	164
Template Method	Deklarieren von vordefinierten, abstrakten Funktionen (Template), welche von erbenden Klassen überschrieben werden können.	59	366	276

Literaturverzeichnis

[Eil10] Eilebrecht, K., Starke, G.: „Patterns kompakt – Entwurfsmuster für effektive Software-Entwicklung“, 3. Auflage, Heidelberg, Spektrum Akademischer Verlag, 2010

[Fow03] Fowler M.: „Patterns of Enterprise Application Architecture“, 1. Auflage, Boston, Pearson Education, Inc., 2003

[Fre06] Freeman, E., Freeman E.: „Entwurfsmuster von Kopf bis Fuß“, 1. Auflage, Köln, O’Reilly Verlag GmbH & Co. KG, 2006

[Gam96] Gamma, E., Helm R., Johnson R., Vlissides J.: “Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software”, 1. Auflage, Bonn, Addison-Wesley, 1996

[Sch09] Schmidt, S.:„PHP Design Patterns - Entwurfsmuster für die Praxis“, 2. Auflage, Köln, O’Reilly Verlag GmbH & Co. KG, 2009

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich,

1. dass ich meine Studienarbeit mit dem Thema:

   Design Patterns in der Softwareentwicklung - Verwendungen in Theorie und Praxis am Beispiel Magento

   ohne fremde Hilfe angefertigt habe,
2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe und
3. dass ich meine Studienarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

_____ _____

Ort, Datum Unterschrift