Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Fachbereich IV
Angewandte Informatik

Bachelorarbeit

über das Thema

Konzeption und prototypische Entwicklung einer Schnittstelle zwischen der Multichannel Marketing Solution *iTool 3* und dem Mercateo Maketplace

Hendrik Hofmann
Matr.Nr.: 539721
27. Januar 2017

Prüfer:
Prof. Dr.-Ing. habil.
Dierk Langbein,

I Inhaltsverzeichnis

1 Einleitung

Vorliegende Arbeit entstand im Rahmen meiner Werkstudententätigkeit bei der Firma i-Ways Sales Solutions GmbH und meinem Mitwirken an der von i-Ways entwickelten, auf einem PHP-Webframework basierenden Multichannel-Marketing Software iTool3, die es – unter Anderem – gestattet über eine Weboberfläche Produkte verschiedener Verkäufer auf Onlinemarktplätzen wie Amazon oder eBay anzubieten. Ziel dieser Arbeit ist es die Anwendung, im Kundenauftrag, um die Möglichkeit der Anbindung an den Mercateo Marktplatz, einer Online B2B Beschaffungsplattform, zu erweitern. Da Mercateo ein API bereitstellt das Produktdaten über einen XML-Katalog empfängt, soll im folgenden untersucht werden, wie die Anbindung der Software iTool3 an den Mercateo Marktplatz realisiert werden kann. Ziel der Arbeit ist es einen funktionsfähigen Prototypen zu entwickeln, der einen fehlerfreien Transfer der mit iTool verwalteten Produkte an Mercateo ermöglicht.

2 Grundlagen

2.1 iTool3

iTool3 ist eine auf dem CakePHP 3.3 - Framework basierende eCommerce Software Lösung zur Steuerung von Produktsortimenten auf verschiedenen Marktplätzen mit dem Ziel, den Vertriebsprozess zu automatisieren. Es ermöglicht dem Nutzer, über eine einzelne Benutzeroberfläche, Produkte auf Marktplätzen wie eBay, Amazon oder auch einem Magento-Store zu verwalten. Produkte können dabei händisch erstellt oder aus bestehenden Datenquellen in die Software eingepflegt werden. Im Anschluss ist es möglich diese Produkte auf einem oder mehreren Marktplätzen anzubieten. Die Verwaltung und Abwicklung der eingehenden Bestellungen läuft dabei komplett über das iTool. Da für jeden Marktplatz unterschiedliche Daten benötigt werden um auf ihm erfolgreich zu verkaufen, können für jedes Produkt unterschiedliche Attribute mit wiederum unterschiedlichen Werten angelegt werden. Die Produktverwaltung der Software folgt daher dem Entity-Attribute-Value Modell.

2.1.1 Verkäufer und Benutzer

Es wird unterschieden zwischen Verkäufern (Core-Seller) und Benutzern (Core-User). Einem Verkäufer können mehrere Benutzer zugeordnet werden, die, mit mehr oder weniger Rechten ausgestattet, die Produkte nur einsehen, oder Kontrolle über die gesamte Produkt- und Bestellverwaltung haben können.

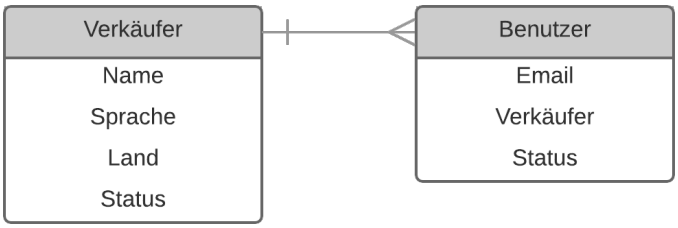

Abbildung 1: ER-Diagramm: Verkäufer - Benutzer

Benutzerdaten werden dabei in der Tabelle `core_users` gespeichert, Verkäuferdaten in `core_sellers`.

2.1.2 Produktverwaltung

Die Produktverwaltung ist aufgeteilt in Produkte und Kategorien. **Produkte** besitzen Attribute wie Titel, Preis, Beschreibung etc. die für jeden Marktplatz auf denen diese angeboten werden sollen unterschiedlich ausfallen können. Es kann gewählt werden, ob ein Produkt auf einem bestimmten

Marktplatz angeboten werden soll oder nicht. Ein Produkt kann dabei mehreren **Kategorien** zugeordnet sein.

Abbildung 2: ER-Diagramm: Kategorie - Produkt

Eine Kind-Kategorie hat jeweils genau eine Eltern-Kategorie. Eine Eltern-Kategorie kann jedoch mehrere Kind_Kategorien haben.Die einzelnen Produkte sind genau einem Verkäufer zugeordnet.

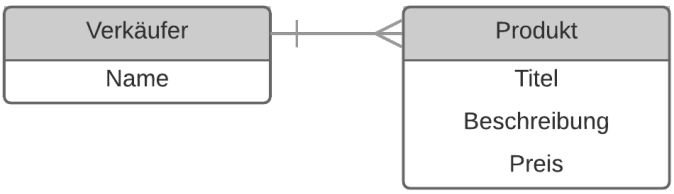

Abbildung 3: ER-Diagramm: Verkäufer - Benutzer

Alle Produktdaten sind in der Tabelle `core_products` und den damit verknüpften Tabellen hinterlegt.

2.1.3 Dashboard

Auf dem Dashboard werden Informationen über die Anzahl der insgesamt eingegangenen Bestellungen, den durchschnittlichen Bestellwert, die Gesamtzahl der Kunden, den insgesamt erwirtschafteten Umsatz, eine Übersicht der zuletzt eingegangenen Bestellungen sowie eine graphische Übersicht der während eines Jahres erwirtschafteten Umsätze angezeigt.

2.2 Der BMECat

Der BMECat ist ein vom „Bundesverband Materialwirtschaft, Einkauf und Logistik e.V“ in Zusammenarbeit mit dem „eBusiness Standardization Committee“ entwickelter XML Standard mit dem Ziel den Austausch von Produktkatalogen zwischen Lieferanten und beschaffenden Organisationen zu standardisieren und somit zu vereinfachen[1].

[1] BMECat V1.2 Spezifikation, Seite 5

2.2.1 Terminologie

Ein **Produktkatalog** ist die Menge aller benötigten Daten, welche vom katalogerzeugenden Unternehmen an das katalogempfangende Unternehmen übermittelt werden sollen.
Ein **Katalogdokument** ist eine XML-Datei, in der der Produktkatalog im BMECat-Format gespeichert und zum Katalogemfänger übermittelt wird.
Eine **Kataloggruppe** ist ein Datenbereich, der eine Gruppe definiert, welcher gleichartige Artikel zugeordnet werden können. Diese wird im BMEcat-Format durch das Element **CATALOG_STRUCTURE** abgebildet.
Ein **Kataloggruppensystem** ist ein hierarchischer Baum von verknüpften Kataloggruppen. Es wird im BMEcat-Format durch das Element **CATALOG_GROUP_SYSTEM** abgebildet.[2]

2.2.2 Transaktionen

Im BMECat wird zwischen 3 verschiedenen Transaktionsarten unterschieden:

- **T_NEW_CATALOG** - Übertragung eines neuen Produktkataloges
- **T_UPDATE_PRODUCTS** - Aktualisierung von Produktdaten
- **T_UPDATE_PRICES** - Aktualisierung von Preisinformationen

Die Unterscheidung geschieht um die Größe eines Katalogdokumentes zu reduzieren. Es muss so z.B. nicht ein kompletter Produktkatalog übertragen werden, falls sich bei einem *odermehreren* Artikel*n* der Preis ändert.

2.2.3 Aufbau

Ein BMECat-Dokument besteht aus einer Folge von „Kann“ und „Muss“ Feldern, den dazugehörigen Datentypen und Feldlängen und ist folgendermaßen aufgebaut:

1. XML Deklaration:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE BMECAT SYSTEM "bmecat_new_catalog.dtd">
<BMECAT version="1.2" xml:lang="de" xmlns="http://www.bmecat.org/
    bmecat/1.2/bmecat_new_catalog">
```

2. Header-Bereich (mit Informationen über Kataloganbieter und Empfänger, Bezeichnung und Erstellungsdatum des Kataloges etc.)

```
  <GENERATOR_INFO> Kann </GENERATOR_INFO>
  <CATALOG> Muss </CATALOG>
  <BUYER> Kann </BUYER>
  <SUPPLIER> Muss </SUPPLIER>
</HEADER>
```

[2] vgl. hierzu:BMECat V1.2 Spezifikation, Seite 7

3. Produktgruppensystem (Baumstruktur der Produktgruppen mit den Attributwerten „root“, `Node` und *Leaf*)

```
<CATALOG_STRUCTURE type="root">
    <GROUP_ID>1</GROUP_ID>
    <GROUP_NAME>Katalog</GROUP_NAME>
    <PARENT_ID>0</PARENT_ID>
    <GROUP_ORDER>1</GROUP_ORDER>
</CATALOG_STRUCTURE>
  <CATALOG_STRUCTURE type="node">
    <GROUP_ID>2</GROUP_ID>
    <GROUP_NAME>Spiele &amp;amp; Konsolen</GROUP_NAME>
    <PARENT_ID>1</PARENT_ID>
 </CATALOG_STRUCTURE>
 <CATALOG_STRUCTURE type="leaf">
    <GROUP_ID>7</GROUP_ID>
    <GROUP_NAME>PlayStation 4</GROUP_NAME>
    <PARENT_ID>2</PARENT_ID>
 </CATALOG_STRUCTURE>
```

4. Artikel (mit Attributen und Werten)

```
<ARTICLE mode="new">
  <SUPPLIER_AID>9057320097280</SUPPLIER_AID>
    <ARTICLE_DETAILS>
      <DESCRIPTION_SHORT>GTA 5</DESCRIPTION_SHORT>
      <DESCRIPTION_LONG>Das tolle neue Spiel</DESCRIPTION_LONG>
      <EAN>87126723434</EAN>
        ... weitere Attribute ...
    </ARTICLE_DETAILS>
        ...weitere Felder ...
</ARTICLE>
```

5. Zuordnung der Artikel zu den Produktgruppen.

```
<ARTICLE_TO_CATALOGGROUP_MAP>
  <ART_ID>9057320097280</ART_ID>
  <CATALOG_GROUP_ID>7</CATALOG_GROUP_ID>
</ARTICLE_TO_CATALOGGROUP_MAP>
```

— Übersicht der im BMECat verwendeten Datentypen — noch einfügen —

Im Folgenden wird jeder Teilbereich mit seinen Unterelementen graphisch dargestellt und erläutert. Farblich rot markiert sind jeweils die „Muss“-Felder, welche zwingend in einem gültigen BMECat Dokument vorkommen müssen, grün die „Kann“-Felder. Ein Plus + Zeichen hinter dem Elementnamen indiziert, dass dieses Element mehrfach an dieser Stelle vorkommen muss, jedoch mindestens einmal. Ein Asterisk $*$ zeigt an, dass dieses Element einmal, mehrfach oder gar nicht vorkommen kann.

Header

Im Header werden allgemeine Informationen über das Katalogdokument hinterlegt und Default Werte gesetzt. Das Element **`CATALOG`** enthält dabei

Informationen zur Identifikation und Beschreibung des Produktkataloges, wie z.B. die KatalogID, die Katalogversion oder die für das Katalogdokument geltende Sprache sowie Elemente zum Setzten von Standard-Werten wie z.B. die für das Katalogdokument geltende Währungsangabe [3].

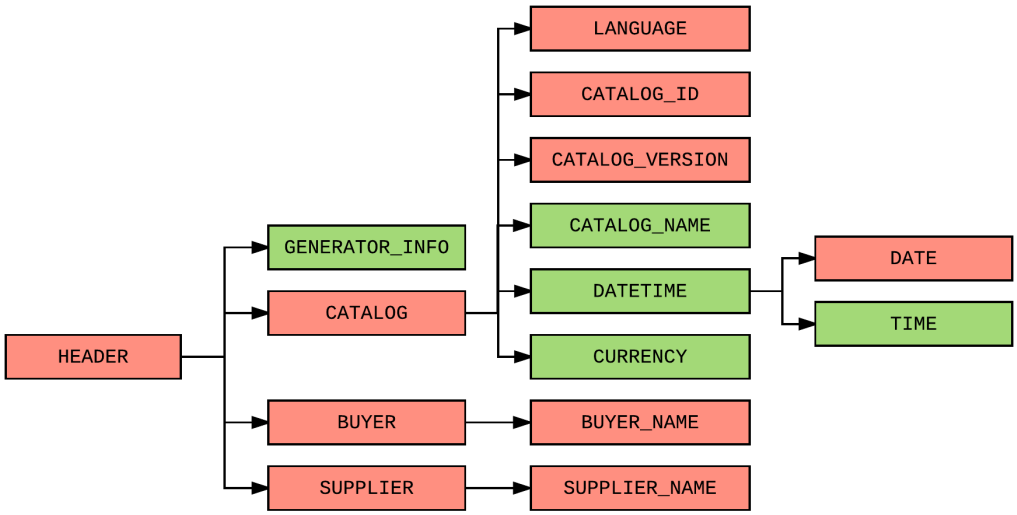

Abbildung 4: Headerstruktur

Die Transaktion T_NEW_CATALOG

Diese Transaktion wird verwandt, um einen initialen Produktkatalog zu übertragen. Das empfangende System reagiert dabei je nach übertragener `CATALOG_ID`, `CATALOG_VERSION` und `LANGUAGE` unterschiedlich. Dieser Zusammenhang wir später noch erläutert.

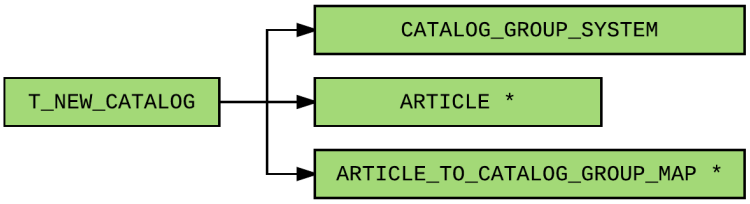

Abbildung 5: T_NEW_CATALOG

Die Transaktion T_UPDATE_PRODUCTS

Bei dieser Transaktion werden Artikeldaten übertragen und gegebenenfalls einer Kataloggruppe zugeordnet. Je nach Kennung des Artikels werden die übertragenen Artikel im Zielsystem entweder hinzugefügt, gelöscht oder die Artikeldaten werden komplett ersetzt. Der Artikel wird immer komplett

[3] BMECat V 1.2 Spezifikation, Seite 27,29

ausgetauscht, eine Änderung von einzelnen Datenfeldern innerhalb eines Artikels ist nicht möglich. Wie der Grafik entnommen werden kann ist bei dieser Transaktion nur die Übertragung von Produktdaten und die Zuordnung von Produkten zu Kataloggruppen möglich, nicht jedoch das Erstellen eines neuen Kataloggruppensytems [4].

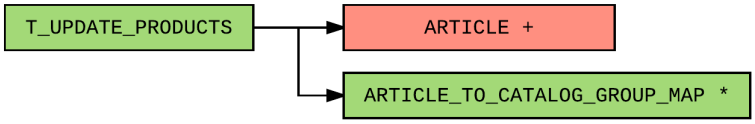

Abbildung 6: T_UPDATE_PRODUCTS

Das Element `T_UPDATE_PRODUCTS` verfügt über das Attribut `prev_version`, welches die Anzahl der vorausgegangenen Updates enthält. Der Wert dieses Attributes wird nach jedem Update um 1 erhöht.

```
<T_UPDATE_PRODUCTS prev_version="91">...</T_UPDATE_PRODUCTS>
```

Die Elemente CATALOG_GROUP_SYSTEM und CATALOG_STRUCTURE

Im Element `CATALOG_GROUP_SYSTEM` werden die `GROUP_SYSTEM_ID` und der `GROUP_SYSTEM_NAME` bekannt gemacht sowie die Katalogstruktur – `CATALOG_STRUCTURE` – beschrieben. Dabei gibt es genau ein Wurzelelement, sowie beliebig viele Knoten und Blätter. Jedes Element hat dabei eine als `GROUP_ID` bezeichnete ID und wird über `PARENT_ID` die dem jeweiligen Elternelement zugeordnet. Die Zuordnung der Artikel zu den Artikelgruppen erfolgt mit dem Element `ARTICLE_TO_CATALOG_GROUP_MAP` das weiter unten beschrieben wird.

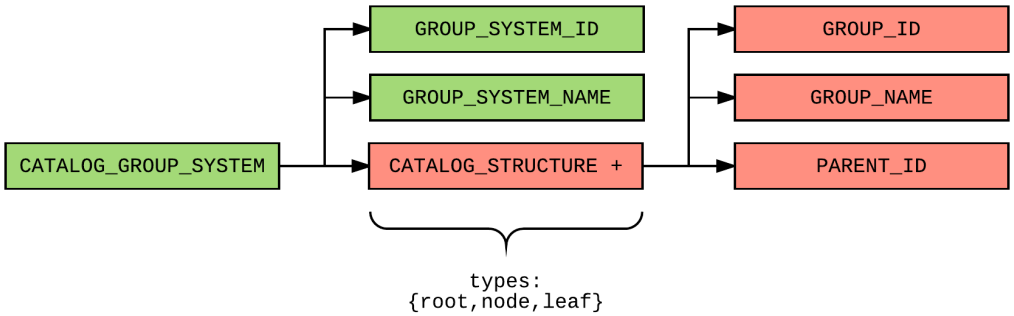

Abbildung 7: CATALOG_GROUP_SYSTEM und CATALOG_STRUCTURE

[4] vgl. BMECat V 1.2 Spezifikation, Seite 52

Das Element ARTICLE

Das Artikelelement schließlich enthält Informationen über einen Artikel, wie Überschrift, Beschreibung, Bilder, Preisinformationen, eine **eindeutige** Artikelnummer usw. Die Artikelnummer wird über das Element `SUPPLIER_AID` bekanntgegeben, handelt es sich um einen Variantenartikel, so bildet sich die Artikelnummer aus der `SUPPLIER_AID` und der `SUPPLIER_AID_SUPPLEMENT`. Dies ist hier jedoch nicht umgesetzt. Die als *eCl@ass* und *Zolltarifnummer* zusammengefassten `ARTICLE_FEATURES` werden explizit von Mercateo verlangt.

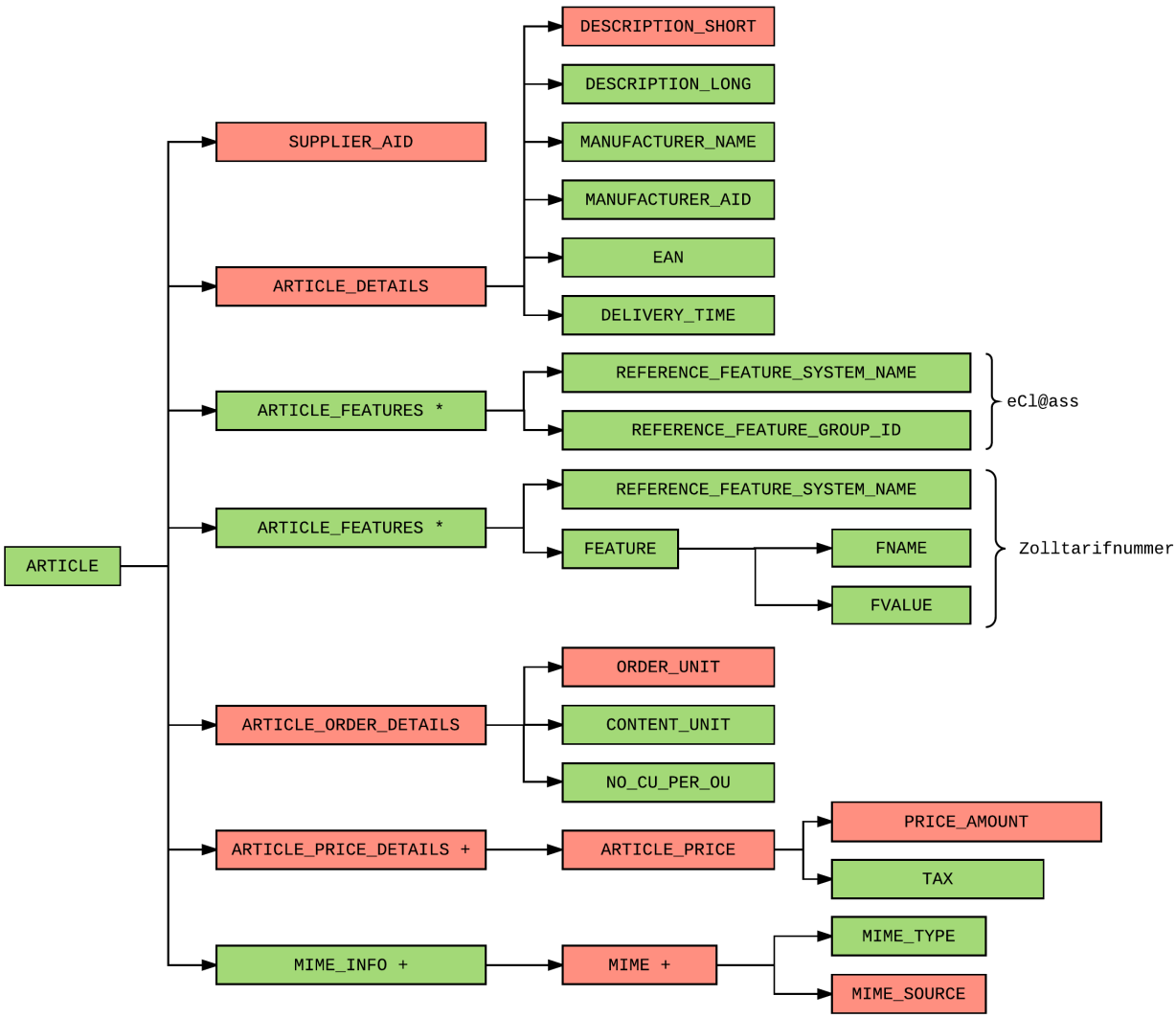

Abbildung 8: Article

Das Element `ARTICLE` verfügt über das Attribut `mode`, welches Informationen darüber enthält, ob es sich um die Anlage eines neuen Artikel, ein Update der Artikelinformationen oder die Löschung eines Artikels handelt.

```
<ARTICLE mode="new">...</ARTICLE>
<ARTICLE mode="update">...</ARTICLE>
```

```
<ARTICLE mode="delete">...</ARTICLE>
```

Das Element ARTICLE_TO_CATALOG_GROUP_MAP

Um Produkte ihren Kategorien zuordnen zu können wird das Element `ARTICLE_TO_CATALOGGROUP_MAP` verwandt. Es erfolgt hier eine Verknüfung aus der eindeutigen Artikelnummer und der `GROUP_ID` welcher der Artikel zugeordnet werden soll. Eine Mehrfachzuordnung ist möglich, d.h. ein Artikel kann in unterschiedliche Kategorien „eingehängt“ werden.

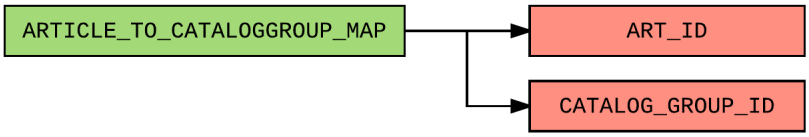

Abbildung 9: ARTICLE_TO_CATALOG_GROUP_MAP

Im Kontext der Transaktion `T_UPDATE_PRODUCTS` verfügt das Element zusätzlich über das Attribut `mode`, mit welchem angegeben wird, ob es sich um eine Neuzuweisung zu einer Kategorie handelt oder der Artikel aus einer Kategorie entfernt werden soll.

```
<ARTICLE_TO_CATALOGGROUP_MAP mode="new">...</<ARTICLE_TO_CATALOGGROUP_MAP>
<ARTICLE_TO_CATALOGGROUP_MAP mode="delete">...</<ARTICLE_TO_CATALOGGROUP_MAP>
```

Zusammenspiel verschiedener Transaktionen

Die folgende Grafik zeigt, wie das empfangende System bei der Transaktion `T_NEW_CATALOG` je nach übergebener `CATALOG_ID`, `CATALOG_VERSION` und `LANGUAGE` reagiert.

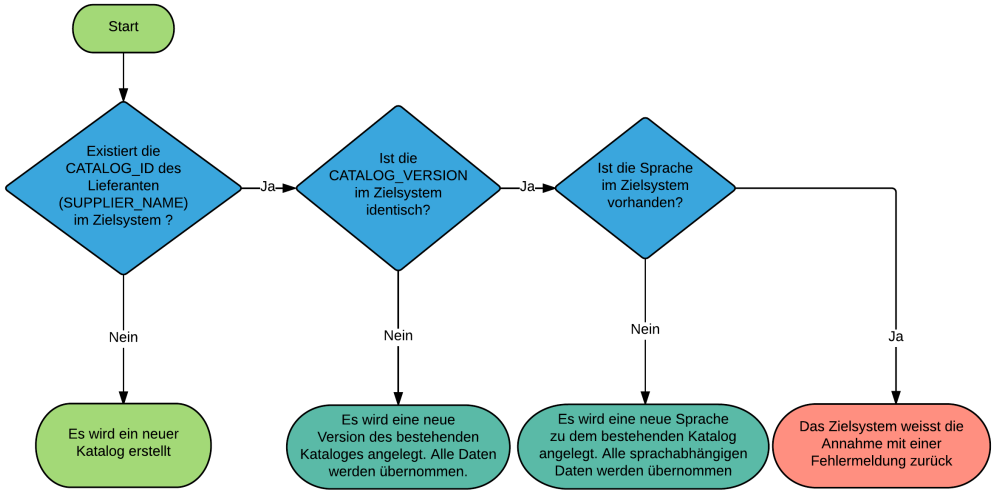

Abbildung 10: Logik des empfangenden Systems bei der Transaktion `T_NEW_CATALOG`

Kommt die Transaktion T_UPDATE_PRODUCTS zur Anwendung, gilt es folgendes zu beachten[5]:

- Die übertragene CATALOG_ID des jeweiligen Lieferanten und die dazugehörige CATALOG_VERSION müssen im Zielsystem bereits vorhanden sein.
- Das Attribut prev_version muss bei der ersten anderen Transaktionsart nach T_NEW_CATALOG, (T_UPDATE_PRODUCTS,T_UPDATE_PRICES) auf „0“ gesetzt werden.
- Danach wird es bei jeder solchen Transaktion um „1“ erhöht.

—- Übersicht, tabellarisch oder nicht über die wichtigsten Felder und ihre Einschränkungen, vor allem die von Mercateo —-

2.3 Das Cake-PHP Framework

Cake PHP ist ein Webframework, das dem MVC (Model-View-Controller) Schema folgt und dabei die Softwaredesignparadigmen DRY (Don’t repeat yourself) und „Convention over configuration“ umsetzt.

2.3.1 Convention over Configuration in CakePHP

In CakePHP wird das Softwaredesign-Paradigma der „Konvention vor Konfiguration“ konsequent umgesetzt.

Die Klassennamen von **Controllern** sind im Plural verfasst, „CamelCased“ und enden auf *Controller*. `UsersController` und `ArticleCategoriesController` sind Beispiele dafür. Eine öffentliche Methode eines solchen Controllers kann über einen Webbrowser aufgerufen werden. Per Konvention werden Controllernamen in URLs klein geschrieben und mit Bindestrich verbunden. `http://samplesite.com/article-categorie/view` ruft demnach die öffentliche `view()` Methode des ArticleCategoriesControllers auf.

Die Namen von **Model** Klassen sind „CamelCased“ und im Plural. Der Name der zum Model gehörenden Tabelle ist im Plural verfasst und mit einem Unterstrich verbunden.
`article_categories` ist die dem Model `ArticleCategories` zugrundeliegende Tabelle. Um einen Fremdschlüssel auf eine Tabelle zu vergeben genügt es das Suffix `_id` an den kleingeschriebenen Namen dieser Tabelle anzuhängen. Wenn Users eine hasMany Beziehung zu Articles hat, kann mit dem Fremdschlüssel `user_id` in der `articles`-Tabelle auf den entsprechenden Eintrag in der `users`-Tabelle verwiesen werden.

[5] vgl. hierzu: BMECat V 1.2 Spezifikation, Seite 52

Die Template Datei einer **View** ist nach der entsprechenden Methode im Controller benannt, die sie darstellen soll. Die `view()` Methode der `ArticlesController` Klasse würde demnach unter `src/Template/Articles/view.ctp` nach einem View-Template suchen[6].

2.3.2 Model

Das Backend einer CakePHP Anwendung wird von einer SQL Datenbank gebildet. Das Model repräsentiert die Daten einer Anwendung und enthält die Geschäftslogik zur Datenmanipulation. Nach der CakePHP Konvention wird die Datenbankverbindung einmal in der Daeti `config/app.php` konfiguriert. Die Model-Klasse stellt dabei Methoden zur Verfügung, über die es möglich ist, den Zustand der Daten abzufragen, die Daten zu filtern und zu verändern. Die CRUD-Funktionalität (CREATE-READ-UPDATE-DELETE) ist so direkt im Model integriert.[7]. Die Beziehungen einzelner Models zueinander werden über *Associations* hergestellt. Die vier Assoziationstypen in CakePHP sind:

Nr.	Beziehung	Typ	Beispiel
1	one to one	hasOne	Ein Museum hat eine Adresse.
2	one to many	hasMany	In einem Museum hängen mehrere Kunstwerke.
3	many to one	belongsTo	Mehrere Bilder gehören zu einem Museum.
4	many to many	belongsToMany	Ein Student hat mehrere Professoren. Ein Professor hat mehrere Hörer.

Tabelle 1: Übersicht der Assoziationstypen in CakePHP

Es ist möglich ein *Model* um ein oder mehrere *Behavior* zu erweitern. Dabei handelt es sich um Klassen, in denen, ähnlich einem Trait, Funktionen zur Erweiterung des Models gekapselt sind. Ein Beispiel hierfür ist das Tree-Behavior, das es ermöglicht hierarchische Datenstrukturen in der Datenbank zu pflegen. Anwendung hierfür kann z.B. die Abbildung einer Kategoriestruktur sein [8].

Mit Hilfe von im Model definierten Validatoren können zu speichernde Daten auf Vollständigkeit und Konsistenz geprüft werden.

```
    $validator
    ->requirePresence('catalog_name', 'create')
    ->notEmpty('catalog_name')
    ->add('catalog_name', [
            'maxLength' => [
                'rule' => ['maxLength', 100],
                'message' => 'maxLength = 100.'
            ]
]);
```

[6]vgl. hierzu: `http://book.cakephp.org/3.0/en/intro/conventions.html`
[7]vgl. hierzu: Webentwicklung mit CakePHP, 2. Auflage, O'Reilly, Seite 7
[8]vgl. hierzu: `http://book.cakephp.org/3.0/en/orm/behaviors/tree.html`

2.3.3 View

Die View ist für die Darstellung der Daten in der Anwendung zuständig. Eine View ist in CakePHP immer auf einen bestimmten Controller bezogen und wird nicht für die Darstellung anderer Daten verwendet[9]. CakePHP View Template Dateien Enden auf „.ctp“ und bedienen sich der alternativen PHP Syntax für Kontrollstrukturen und Ausgabe. In einer View kann direkt auf Variablen zugegriffen werden die in der entsprechenden Controller Methode gesetzt wurden:

```
$this->set('articleCategories', $articleCategories);
```

Die Codebeispiele zeigen, wie die Variable `$articleCategories` im Controller für die View freigegeben wird und dort z.B. mit einer foreach-Schleife durchlaufen werden kann um ihren Inhalt auszugeben.

```
<ul>
<?php foreach ($todo as $item): ?>
  <li><?= $item ?></li>
<?php endforeach; ?>
</ul>
```

Listing 1: Alternative PHP Syntax

Eine View ist dabei nicht auf das Anzeigen von HTML Inhalten beschränkt, sondern kann auch dazu verwandt werden XML- oder JSON- Repräsentationen der angefragten Daten zurückzuliefern.

2.3.4 Controller

Der Controller regelt den Ablauf der Benutzerinteraktion. Er ist dafür zuständig, dass das richtige Model aufgerufen und die entsprechende Antwort oder View erzeugt wird. Er dient dabei als eine Art Vermittler zwischen dem Model und der View. Normalerweise ist in CakePHP ein Controller für ein Model verantwortlich, es ist dennoch möglich, oft auch nötig, dass ein Controller mit mehreren Models arbeitet.

Der Controller enthält eine Reihe von Methoden die HTTP Anfragen verarbeiten. Diese Methoden werden in CakePHP *actions* genannt. Per Definition ist jede öffenliche Methode in einem Controller eine *action* und über eine URL der Form `http://samplesite.com/article-categorie/view` erreichbar. Eine *action* ist für die Verarbeitung der Anfrage und das Zurückliefern einer Antwort zuständig. Im Normalfall wird dabei eine View erzeugt, es können aber auch (wie im Abschnitt Model erläutert) XML oder JSON Daten zurückgeliefert werden.[10]

[9] vgl. hierzu: Webentwicklung mit CakePHP, 2. Auflage, O'Reilly, Seite 7
[10] vgl. hierzu: `http://book.cakephp.org/3.0/en/controllers.html`

2.3.5 Component

Komponenten (Components) sind in sich geschlossene Bereiche innerhalb einer Applikation, die eine bestimmte Funktionalität kapseln und über die Grenzen eines Controllers hinaus verfügbar machen. Sollen bestimmte logische Prozesse in verschiedenen Teilen einer Anwendung zur Verfügung stehen - insbesondere in unterschiedlichen Controllern- so ist es sinnvoll diese in eine Komponente auszulagern[11]. Die Möglichkeit mit Komponenten zu arbeiten setzt das DRY Paradigma konsequent um.

2.3.6 Shell

CakePHP bietet die Möglichkeit Konsolenanwendungen zu schreiben. Dies ist nützlich für Anwendungen die per Cronjob ausgeführt werden sollen oder für solche die nicht aus einem Browser erreicht werden müssen bzw. sollen[12]. Eine der wichtigsten Funktionalität der Cake Shell ist das „Backen“ (Baking). Gemeint ist damit die automatische Generierung von Code. Der Befehl `bin/cake bake` erstellt, je nach gewählter Option, ganze MVC Grundgerüste, Controller- oder Model- Klassen, Plugin Verzeichnisstrukturen oder Shell-Klassen. Einzelne Funktionalitäten einer Shell Klasse können in Tasks ausgelagert werden.

3 Analyse der Aufgabe und der Anforderungen

3.1 Funktionale und nichtfunktionale Anforderungen

3.1.1 Funktionale Anforderungen

1. Die in iTool hinterlegten Produkt- und Herstellerdaten sollen in das BMECat Format in der Version 1.2 überführt werden.
2. Es sollen die beiden Transaktionsarten `T_NEW_CATALOG` und `T_UPDATE_PRODUCTS` umgesetzt werden[13].
3. Die Produktkategoriestruktur des iTool soll in das Kataloggruppensystem des BMECat überführt werden.
4. Die Katalogerstellung soll in einer CakeShell erfolgen.
5. Das Programm soll über einen CronJob gesteuert werden können.
6. Es soll Mercateo ermöglicht werden Bestandsdaten zu den im Katalog vorhanden Produkten über einen Webservice abzurufen. Der Aufruf erfolgt über eine URL der Form `http://itool.local/mercateo/availability/12`, wobei der letzte Wert die angefragte SKU repräsentiert.
7. Es sollen Kataloge für unterschiedliche Verkäufer erstellt werden können.

[11] vgl. hierzu: Webentwicklung mit CakePHP, 2. Auflage, O'Reilly, Seite 223
[12] vgl. hierzu `http://book.cakephp.org/3.0/en/console-and-shells.html`
[13] vgl. dazu: Kapitel 1.2.2 & 1.2.3

8. Die zu exportierenden Produkte sollen in einer Warteschlange gehalten werden um die verschiedenen Artikelmodi („new“, „update“, „delete“) ausgezeichnet werden zu können.
9. Dem Dokumentennamen sollen der Name des Verkäufers sowie Datum und Uhrzeit der Entstehung entnommen werden können.

3.1.2 Nichtfunktionale Anforderungen

1. Das Katalogdokument soll gültig sein.
   Das bedeutet, dass es fehlerfrei gegen das entsprechende XSD Schema laufen kann.
2. Das Katalogdokument soll vollständig sein.
   Das bedeutet, es müssen zum einen mindestens jene Felder im BMECat Dokument vorkommen, die die BMECat Spezifikation verlangt. Zusätzlich müssen jene Felder vorkommen, die die Mercateo Spezifikation erfordert und zwar unter zusätzlicher Beachtung der Limitierungen bzw. Besonderheiten jener Spezifikation.
3. Die zu exportierenden Produktdaten sollen über das GUI des iTool editier- und einsehbar sein.
4. Die Verkäufer- und Katalogspezifischen Daten sollen über das GUI des iTool editier- und einsehbar sein.
   - Verkäuferspezifische Daten sind:
     - `core_seller_id`
     - `core_marketplace_id`
     - `core_currency_id`
   - Katalogspezifische Daten sind (in Klammern steht das entsprechende BMECat Element):
     - Der Name des verkaufenden Unternehmens (`SUPPLIER_NAME`)
     - Der Titel des Kataloges (`CATALOG_ID`)
     - Die Katalogversion (`CATALOG_VERSION`)
     - Die Kennung des Kataloggruppensystems (`GROUP_SYSTEM_ID`)
     - Der Name des Kataloggruppensystems (`GROUP_SYSTEM_NAME`)
     - Die Beschreibung des Kataloggruppensystems (`GROUP_SYSTEM_DESCRIPTION`)
5. Der Entwurf soll den Designprinzipien der „Single Responsibility“ und des „Open Closed“ folgen.
6. Die zu exportierenden Daten sollen möglichst schon vor dem Export in das BMECat-Format validiert werden.
7. Es sollen zumindest „Single-Products“ gelistet werden können.
8. Es sollen nach Möglichkeit auch „Configurable-Products“ gelistet werden können.
9. Es soll eine große Anzahl (>10.000) an Produkten ohne Speicherüberläufe exportiert werden können.

3.2 Zielstellung

Ziel der prototypischen Entwicklung der Software ist es, zu zeigen, dass die in iTool gehaltenen Produktdaten in das BMECat-Format überführt werden können. Es soll insbesondere sichergestellt wer-

den, dass die erstellten Katalogdokumente fehlerfrei gegen die enstprechenden XSD-Schemata laufen. Falls Fehler auftreten, sollen diese geloggt werden. Da mit einer großen Anzahl zu exportierende Produkte zu rechnen ist, muss verantwortungsvoll mit den zur Verfügung stehenden Ressourcen umgegangen werden. Es sollen zunächst nur 'Simple-Products' exportiert werden. Die Verwaltung der Produkt-, Katalog- und Herstellerdaten soll einfach, nachvollziehbar und komfortabel sein. Die erwähnten Designprinzipien sollen, aufgrund der daraus resultierenden besseren Wartbarkeit des Codes, umgesetzt werden.

3.3 Einschätzung der verwendeten Technologien

3.3.1 BMECat Format

Allgemeine Vorteile die sich aus dem XML-Format ergeben sind die gleichzeitige Mensch- und Maschinenlesbarkeit sowie die Möglichkeit das Dokument gegen ein XML-Schema testen zu können. So kann schon direkt nach der Erzeugung des BMECat Dokumentes überprüft werden, ob die geschriebenen Elemente vom richtigen Datentyp sind und das Dokument der in der XSD Datei festgelegten Struktur folgt. Weitere Vorteile speziell des BMECat Standards sind[14]:

- konfigurierbare Produkte sind abbildbar.
- mehrsprachige Kataloge sind in einem Katalogdokument abbildbar.
- Übermittlung multimedialer Datenelemente ist möglich (z.B. Produktvideos).
- gilt zumindest in Deutschland als etabliertes Katalogaustauschformat.

3.3.2 Datenübertragung zu Mercateo

Die Übertragung der Katalogdatei zum Mercateo-Server geschieht über FTP. Neue Dateien werden alle 30 Minuten vom Mercateo-System verarbeitet.
Vorteile:

- einfach anzuwenden.
- Eine korrekte Datenübertragung ist durch die Fehlerbehandlung von TCP gewährleistet.

Nachteile:

- Datenübertragung nicht nach außen abgesichert.
- Übertragene Daten können mitgelesen und manipuliert werden.
- Benutzerkennung und Passwort können abgefangen werden
- Die vollständige Datenübertragung kann nicht garantiert werden (fehlende Prüfsummen o.ä.)

[14] vgl. hierzu: `http://wiki.prozeus.de/index.php/BMEcat`

Fazit:
Nicht optimal, vor allem aus Sicherheitsgründen. Zudem Fehleranfällig, wenn die Ordnerstruktur- und Dateinamenskonventionen von Mercateo nicht eingehalten werden [15].

[15] vgl. hierzu: `http://www.mercateo.com/support/verkaufen/katalog-allgemeine-informationen/datenuebertragung-per-ftp/`

4 Entwurf

Bei dem Entwurf der Software sollen die Prinzipien der *Single-Responsibility* und des *Open-Closed* zur Anwendung kommen.

- **Das Single-Responsibility-Prinzip**
  besagt, dass eine Klasse nur eine fest definierte Aufgabe hat und nur Methoden enthält, die zur Erfüllung dieser Aufgabe notwendig sind.
- **Das Open-Closed-Prinzip**
  besagt, dass Klassen sowohl offen für Erweiterungen, als auch geschlossen für Modifikationen seien sollen. Das Prinzip der Vererbung (Polymorphie) setzt dies um.

4.1 Katalogerstellung

Die Erstellung der BMECat Dokumente wird über eine CakePHP Shell und von dort aus aufrufbare Tasks realisiert. Das hat den Vorteil, dass diese, im Bedarfsfall, über einen Cronjob automatisiert werden können. Die Logik zur Erstellung eines neuen BMECat Dokumentes wird in die Klasse `AddNewCatalogTask` ausgelagert, die zur Erstellung eines Updatekataloges in die Klasse `UpdateCatalogTask`. Mit der Klasse `DeleteCatalogTask` wird das Löschen von den zu exportierenden Katalogdaten umgesetzt. Das Schreiben eines Katalogdokumentes wird durch die Componentklasse `BMECatComponent` realisiert, die sich wiederum der Komponentenklasse `XMLWriterComponent` bedient. Hier kommt das *Single-Responsibility* Prinzip zur Anwendung; Jede Klasse erfüllt genau eine Aufgabe. `AddNewCatalogTask`, `UpdateCatalogTask` und `DeleteCatalogTask` werden von der Basisklasse `CatalogToolsTask` abgeleitet. So wird das Prinzip des *Open-Closed* umgesetzt.

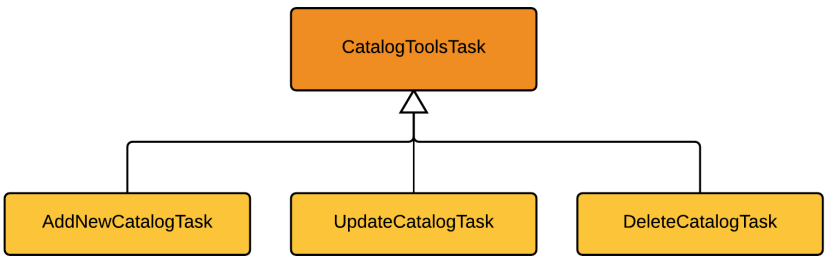

Abbildung 11: Vererbungshierarchie der Task-Klassen

Es soll möglich sein beim Aufruf des Tasks die *id* des `core_sellers` zu übergeben, dessen Produkte exportiert werden sollen. Die Spezifikation des BMECat verlangt, dass im Katalogdokument bestimmte Informationen zur Katalogversion, dem Katalognamen, der der Preisauszeichnung zu Grunde liegenden Währung *etc.* aufgeführt werden. Diese Daten werden in der Tabelle `mercateo_accounts` gespeichert und können über die GUI des iTool eingesehen, erstellt, gelöscht und geändert werden.

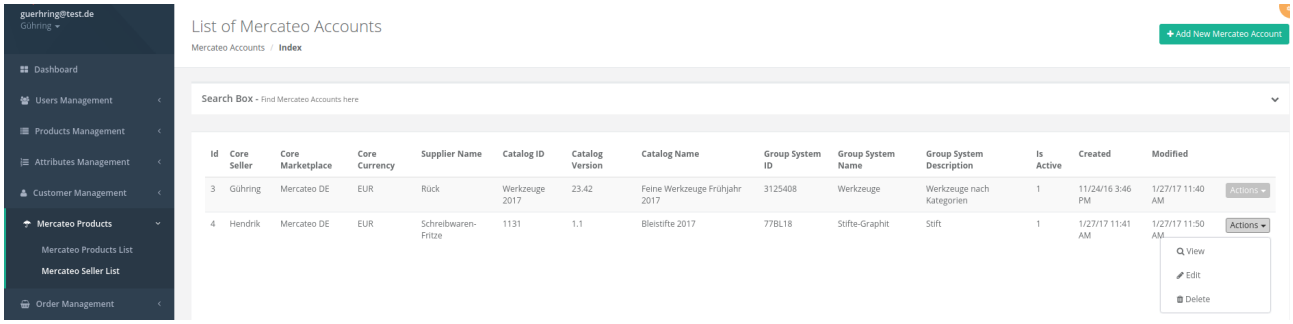

Abbildung 12: Abbildung der `mercateo_accounts` Daten im iTool

Der Zeitpunkt der Katalogerstellung sowie dessen fortlaufende Versionsnummer (das Attribut `prev_version`) werden in die bereits im iTool vorhandene Tabelle `core_configurations` geschrieben. Zur Zwischenspeicherung der zu exportierenden Daten kommt die Tabelle `mercateo_products` zum Einsatz. Die eigentlichen Produktdaten werde aus der Tabelle `core_products` geladen, die Produktkategorien aus der mit dieser verknüpften Tabelle `core_categories`. Mit Hilfe der Tabelle `core_product_updates` kann überprüft werden ob Artikeldaten aktualisiert wurde. Ist dem so, wird dort ein neuer Eintrag erstellt, der die `core_product_id` und den Zeitpunkt der Erzeugung enthält.

Tabelle	Inhalt/Zweck
mercateo_accounts	Speicherung statischer Daten wie Katalog- oder Herstellername.
mercateo_products	Zwischenspeicherung der zu exportierenden Daten.
core_products	Hält sämtliche Produkdaten.
core_categories	Enthält Produktkategoriedaten.
core_configurations	Speichert Konfigurationsgruppen, Pfade und Werte.
core_product_updates	Speichert den Timestamp der letzten Änderung eines `core_products`.

Tabelle 2: Übersicht der bei der Katalogerstellung verwendeten Tabellen

4.1.1 Die Tabelle `mercateo_accounts`

Eine Übersicht der in `mercateo_accounts` gespeicherten Werte bietet Tabelle 3. Wenn nicht anders angegeben entspricht das BMECat Element der Spaltenbezeichnung (`catalog_id` ≈ <CATALOG_ID>)

Alle BMECat spezifischen Spalten werden über eine Methode im Model validiert, so dass nur Werte entsprechend der BMECat- bzw. Mercateo Spezifikationen gespeichert werden können.

4.1.2 Die Tabelle `mercateo_products`

Im Zentrum der Katalogerstellung steht die Tabelle mercateo_products. Sie dient als Zwischenspeicher für die in `core_products` hinterlegten Daten und gibt Auskunft darüber, ob (und wann) Produk-

Spalte	Erläuterung	BMECat Element
id	Primärschlüssel.	×
core_seller_id	Fremdschlüssel auf `core_sellers`.	×
core_marketplace_id	Fremdschlüssel auf `core_marketplaces`.	×
core_currency_id	Fremdschlüssel auf `core_currencies`.	<CURRENCY>
supplier_name	Name des verkaufenden Unternehmens.	✓
catalog_id	Eindeutiger Bezeichner des Produktkataloges.	✓
catalog_version	Version des Produktkataloges.	✓
catalog_name	Beliebiger Name, der den Produktkatalog beschreibt.	✓
group_system_id	Kennung des Kataloggruppensystems.	✓
group_system_name	Name des Kataloggruppensystems.	✓
group_system_description	Beschreibung des Kataloggruppensystems.	✓
created	Timestamp, wann der Eintrag erzeugt wurde.	×
modified	Timestamp, wann der Eintrag geändert wurde.	×

Tabelle 3: Die Tabelle `mercateo_accounts`

te geändert, gelöscht oder neu hinzugefügt wurden. Dadurch wird sie zum zentralen Element zur Umsetzung der Transaktionen `T_NEW_CATALOG` und `T_UPDATE_PRODUCTS`.

Spalte	Erläuterung
id	Primärschlüssel
core_seller_id	Die Id des Verkäufers
core_product_id	Fremdschlüssel auf core_products Tabelle
core_categorie_id	Kategorie ID
status	Der Status des Eintrages
sku	Die SKU des Artikels
title	Die „DESCRITION_SHORT“ des Artikels
created	Timestamp, wann der Eintrag erzeugt wurde
modified	Timestamp, wann der Eintrag geändert wurde

Tabelle 4: Die Tabelle `mercateo_products`

Die Spalten „sku“„core_category_id“ & „title“ sind notwendig um einen Artikel in einem BMECat Dokument als *gelöscht* auszeichnen zu können. Die Spalte „status“ akzeptiert vier *Zustände*:

Zustand	Erläuterung
`new`	Produktdaten wurden neu in `core_products` angelegt.
`update`	Produktdaten wurden geändert.
`delete`	Produktdaten wurden aus `core_products` gelöscht.
`active`	Produktdaten wurden in das aktuelle BMECat Dokument übernommen.

Diese Zustände sind die Werte die das Attribut `mode` des BMECat Elements `ARTICLE` annehmen kann. Gleichzeitig geben sie an dieser Stelle Auskunft darüber, ob ein in `core_products` gespeicherter Datensatz neu ist bzw. gelöscht oder verändert wurde. Jene Datensätze die in das aktuelle BMECat Dokument geschrieben wurden, werden mit „active“ markiert. Die in `mercateo_products` gehaltenen Einträge können über das GUI manipuliert werden.

4.1.3 Allgemeiner Programmablauf bei der Katalogerzeugung

Wird die Shell unter Angabe des entsprechenden Tasks aufgerufen, wird zunächst überprüft ob der Nutzer die *id* des Verkäufers angegeben hat, dessen Produkte exportiert werden sollen. Ist das nicht der Fall bricht das Programm mit einem Hinweis zum korrekten Aufruf ab. Falls die übergebene *id* nicht in der Datenbank gefunden werden kann, so wird eine Liste aller verfügbaren Verkäufer ausgegeben. Anschließend wird, je nach gewähltem Task, ein initiales Katalogdokument oder ein Updatekatalogdokument erzeugt und im Anschluss daran mit dem entsprechenden XML Schema validiert.

4.1.4 Katalogerstellungslogik in der Klasse `AddNewCatalogTask`

Mit dem Aufruf des `AddNewCatalogTasks` wird die Umsetzung der Transaktionsart `T_NEW_CATALOG` realisiert. Zu Beginn wird die `mercateo_products` Tabelle mit den entsprechenden Werten aus `core_products` initialisiert. Allen Einträgen wird dabei zunächst der Status `new` zugewiesen. Daraufhin werden Datum & Uhrzeit der Initialisierung in die Tabelle `core_configurations` geschrieben. Anschließend wird eine BMECat Datei mit der Transaktion `T_NEW_CATALOG` erstellt. Die dazu benötigten Informationen werden aus den Tabellen `mercateo_products`, `core_products`, `mercateo_accounts` und `core_configurations` geladen. Danach wird der Status aller Einträge in `mercateo_products` auf `active` gesetzt.

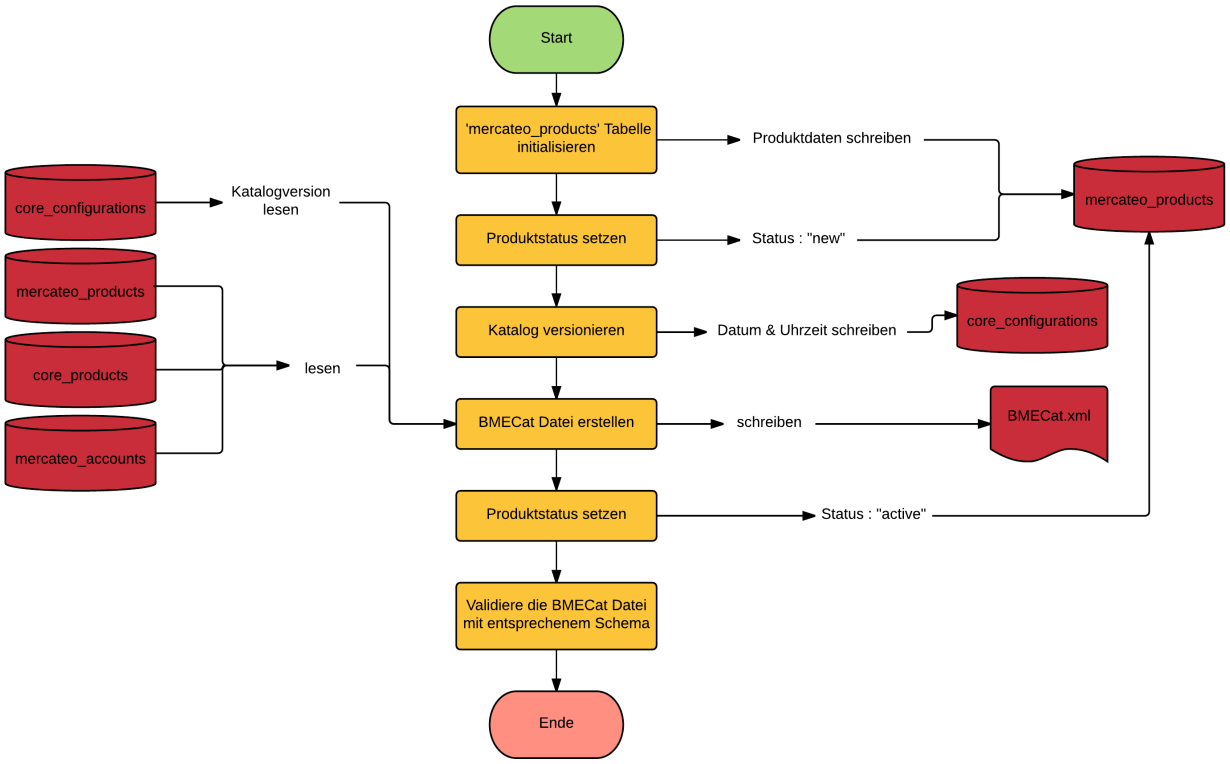

Abbildung 13: Programmlogik bei der Transaktion `T_NEW_CATALOG`

Nachdem das Katalogdokument geschrieben wurde wird es mithilfe des XSD-Schemas (`bmecat_new_catalog_1_2.xsd`) überprüft.

4.1.5 Katalogerstellungslogik in der Klasse `UpdateCatalogTask`

Die Klasse `UpdateCatalogTask` realisiert die Umsetzung der Transaktionsart `T_UPDATE_PRODUCTS`. Bei jedem Aufruf des Tasks wird zunächst - unter Zuhilfenahme der `mercateo_products`-Tabelle - überprüft ob Einträge in der `core_products` Tabelle gelöscht, neu hinzugefügt oder geändert wurden. Letzteres geschieht mit Hilfe der Tabelle `core_product_updates` in der jede Änderung an einem `core_product` mit dem Zeitstempel der Änderung erfasst wird. Ist einer der Fälle eingetreten wird der Status des Eintrages in der `mercateo_products` Tabelle entsprechend gesetzt.

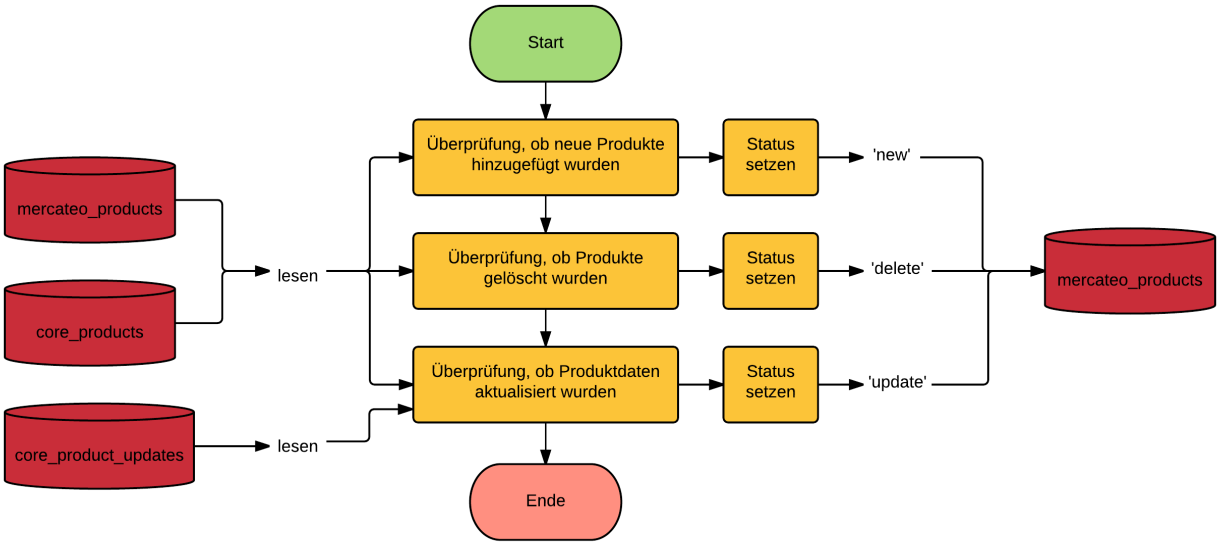

Abbildung 14: Überprüfung auf Produktbestandsänderungen

Anschließend werden wiederum Datum & Uhrzeit der Erstellung des Katalogdokumentes konstituiert. Handelt es sich um die erste Version eines Updatekatalogdokumentes wird das Attribut `prev_version` des Elementes `T_UPDATE_PRODUCTS` mit dem initialen Wert von „0“ in die `core_configurations` Tabelle geschrieben, um beim darauffolgenden Erstellen der BMECat-Datei direkt wieder ausgelesen und an entsprechender Stelle in das Dokument geschrieben werden zu können. Jene Einträge in `mercateo_products`, die den Status `delete` haben, werden gelöscht, danach wird der Status der Übrigen auf `active` gesetzt und der Wert des Attributes `prev_version` um „1“ erhöht. Abschließend erfolgt die Validierung des Dokumentes mithilfe des Schemas `bmecat_update_products_1_2.xsd`.

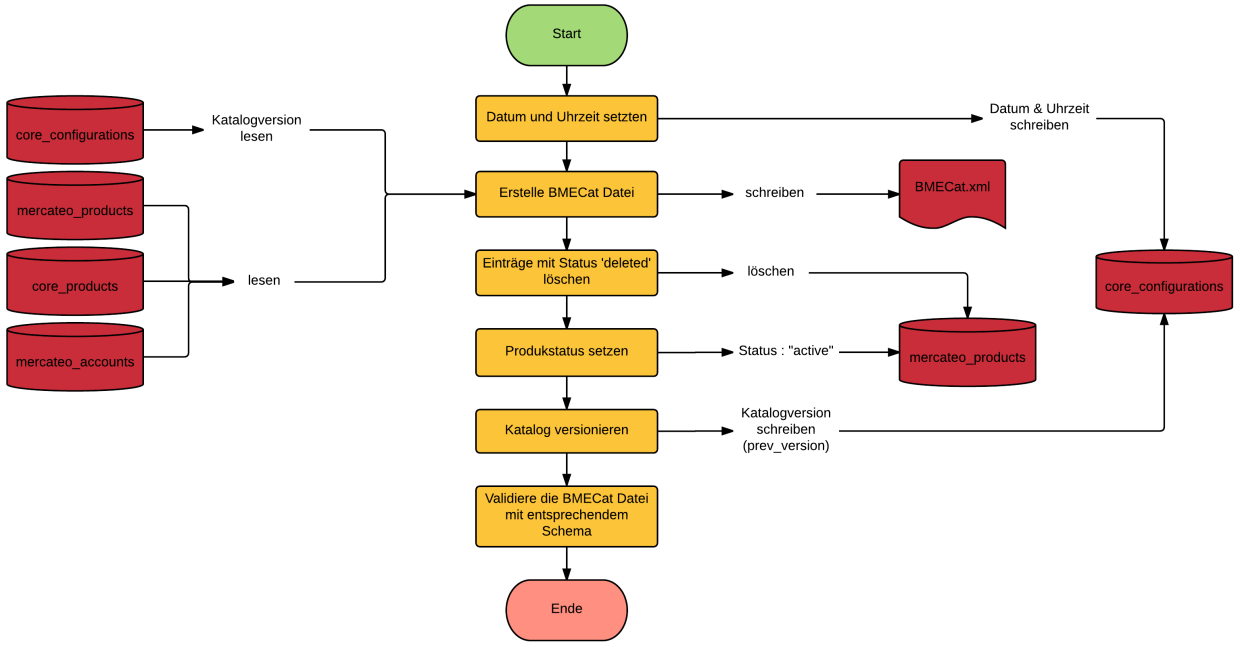

Abbildung 15: Programmlogik bei der Transaktion `T_UPDATE_PRODUCTS`

4.2 Bestandsdatenabfrage

Mercateo Die Bestandsdatenabfrage wird mit einem Controller realisiert, dessen `index()` Funktion als Parameter die angefragte SKU hat. Von Seiten Mercateos kann so eine URL der Form `http://itool.local/mercateo/availability/12` aufgerufen werden. Ist die SKU im System vorhanden, wird die Bestandsmenge als Integer Wert zurückgeliefert, ist die angefragte SKU nicht im System, wird eine Fehlermeldung ausgegeben.

5 Implementierung

Die Implementierung gliedert sich demnach in 2 Bereiche, die Konsolenanwendung zur Generierung des Katalogdokumentes und die Bestandsdatenabfrage.

5.1 Die PrepareCatalog Shell

Mit der PrepareCatalog Shell wird der Entwurf zur Erzeugung eine BMECat Katalogdokumentes umgesetzt. Die eigentliche Shell Klasse `PrepareCatalogShell` dient in der Implementierung dazu die Subcommandos `AddNewCatalogTask`, `UpdateCatalogTask` und `DeleteCatalogTask` aufzurufen und sicherzustellen, dass die notwendigen Argumente übergeben werden.

```
Usage:
cake mercateo.prepare_catalog [subcommand] [-h] [-q] [-v] <Core Seller Id>

Subcommands:

addNewCatalog  Creates a new BMECat Catalog file.
deleteCatalog  Deletes Sellers Products from mercateo_products table
updateCatalog  Creates an Update Catalog file.

To see help on a subcommand use `cake mercateo.prepare_catalog [subcommand] --help`

Options:

--help, -h     Display this help.
--quiet, -q    Enable quiet output.
--verbose, -v  Enable verbose output.

Arguments:

Core Seller Id  The ID of the Seller for whom the BMECat shall be
                created
```

Kann der übergebene Parameter keinem Verkäufer zugeordnet werden wird eine Übersicht der verfügbaren Verkäufer angezeigt. Dies geschieht in den einzelnen Tasks über die geerbte Methode `validateArgument($coreSellerId)`.

```
Please choose one of the available Sellers:

ID | Name

1  | Hendrik
2  | Ben
3  | Guehring
```

Dabei liegt dem Programm folgende Aufrufhierarchie zugrunde:

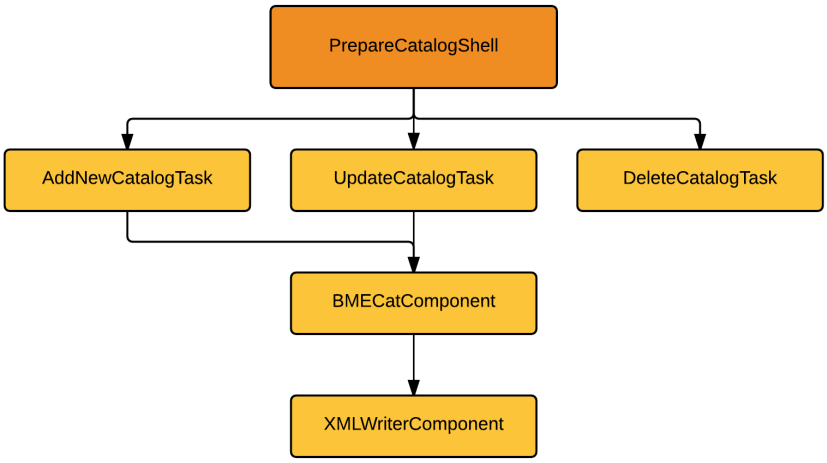

Abbildung 16: Aufrufhierarchie PrepareCatalogShell

Im Folgenden werden die einzelnen Klassen vorgestellt.

5.2 XMLWriterComponent

Die XMLWriterComponent Klasse ist insofern wichtiger Bestandteil der Implementierung, als das ohne sie nur auf umständlicherem Wege XML geschrieben werden kann. Hier seien nun in Kürze jene Methoden vorgestellt, die von der Klasse BMECatComponent genutzt werden:

1. `public function openXmlWriter($filePath, $rootElement, $attributes = null, $doctype = null)`
   Ermöglicht das Anlegen einer neuen XML Datei mit der Option Attribute (z.B. die BMECat-Version) zu übergeben, sowie über den DOCTYPE eine entsprechende .dtd Datei zu referenzieren.
2. `public function closeXmlWriter()`
   Schließt das XML Dokument ab.
3. `writeXmlElement($name, $value, $type = "text", $attributes = [])`
   Schreibt ein XML Element mit dem übergebenen Wert und den dazugehörigen Attributen und schließt es sogleich ab. (Schreibt Start- und End Tag)
4. `public function writeStartXmlElement($name, $attributes = [])`
   Öffnet ein XML Element und setzt die übergebenen Attribute. (Schreibt den Start-Tag)
5. `public function writeEndXmlElement()`
   Schließt das zuvor geöffnete Element ab. (Schreibt den End-Tag)

VORTEILE?

5.3 BMECatComponent

Die Klasse BMECatComponent dient dazu ein wohlgeformtes und gültiges XML Dokument entsprechend den BMECat- und Mercateo Vorgaben zu erstellen.

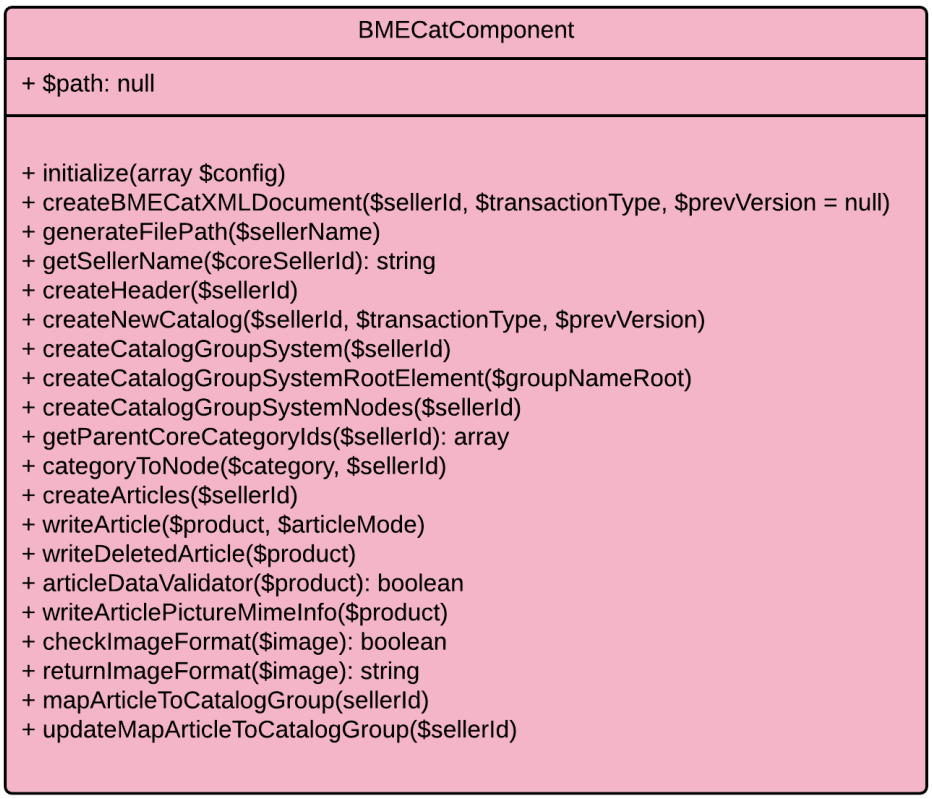

Abbildung 17: UML Diagramm der Klasse BMECatComponent

Wie im Kapitel Grundlagen bereits beschrieben gliedert sich ein BMECat Dokument in 4 Bereiche und zwar den Header, das Kataloggruppensystem, die Auflistung der einzelnen Artikel und die Zuordnung der Artikel zu ihren Kategorien. Der BMECat Komponent stellt für jeden dieser Teilbereiche Funktionen bereit die in Folge erläutert werden sollen. Zur Orientierung dient eine Übersicht der Aufrufhierarchie.

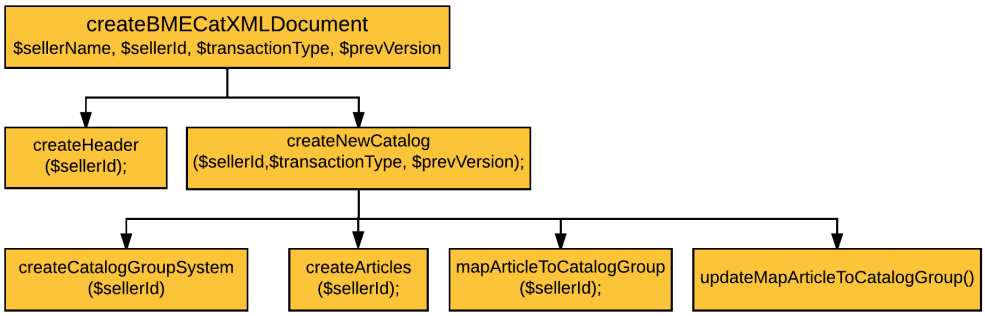

Abbildung 18: Aufrufhierarchie createBMECatXMLDocument

5.3.1 Erstellen des BMECat Dokumentes

Die Funktion `createBMECatXMLDocument($sellerName, $sellerId, $transactionType, $prevVersion)` dient der Erzeugung eines BMECat Dokumentes. Sie legt die XML Datei mit der Namenskonvention „`Verkäufername_Erzeugungsdatum_Erzeugungszeit.xml\` an und schreibt Informationen zum XML Namensraum *(xmlns)* und zur Dokumenttypdefinition *(dtd)* in die XML-Deklaration. Von ihr werden die Methoden zur Erzeugung des Headers und der restlichen Abschnitte des BMECat Dokumentes aufgerufen. Anhand des Parameters `$transactionType` wird entschieden ob bei dem zu erzeugenden Dokument die Transaktion `T_NEW_CATALOG` oder `T_UPDATE_PRODUCTS` umgesetzt werden soll. Falls die Datei, z.B. wegen fehlender Rechte, nicht angelegt werden kann, wird durch die XMLWriter Komponente eine Exception erzeugt.

5.3.2 Schreiben der Header Sektion

Die Methode `createHeader($sellerId)` schreibt die BMECat-Header-Sektion des Dokumentes. Alle benötigten Informationen, wie z.B. Herstellername oder Katalogversion werden dabei anhand der `sellerId` aus der `mercate_accounts` Tabelle geladen.

5.3.3 Schreiben des Kataloggruppensystems

Die Methode `createCatalogGroupSystem($sellerId)` steuert die Erzeugung des Kataloggruppensystems. Die von ihr aufgerufenen Methoden sind im wesentlichen für das Schreiben bestimmter

XML Elemente zuständig.

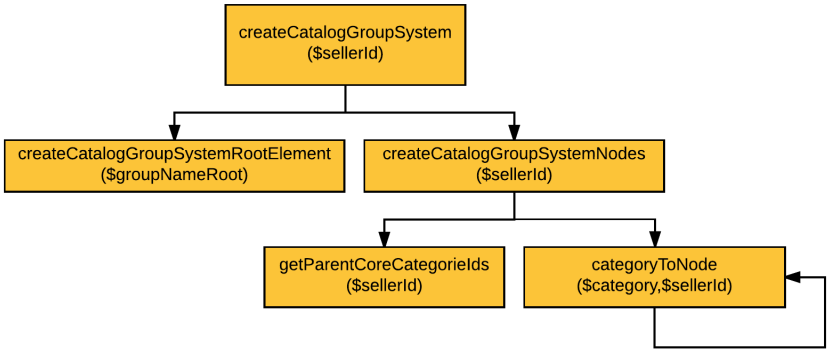

Abbildung 19: Aufrufhierarchie createCatalogGroupSystem

Das tatsächliche Abbilden der Katalogstruktur erfolgt in der Methode `categoryToNode( $category,$sellerId)`.

> Dazu ein kleiner Exkurs:
> CakePHP bietet die Möglichkeit einem Model ein sogenanntes Tree-Behaviour hinzuzufügen. Dieses basiert auf dem Nested-Set-Konzept, das es ermöglicht hierarchische Strukturen in relationalen Datenbanken abzubilden[16]. Die `core_categories` Tabelle bedient sich dieses „Behavious“, was es ermöglicht diesen Kategoriebaum rekursiv zu durchlaufen und dadurch das Kataloggruppensystem des BMECat abzubilden.

Die Methode `categoryToNode($category,$sellerId)` durchläuft, ausgehend vom Wurzelelement, alle Kindelemente und schreibt die entsprechenden Daten in das Dokument. Solange dabei die Anzahl der Kindelemente des gerade traversierten Elementes größer 0 ist wird dabei dem Attribut `type` der Wert `node` zugewiesen. Gibt es keine Kindelemente mehr, wird der Wert auf `leaf` gesetzt.

```
<CATALOG_STRUCTURE type="node">
 <GROUP_ID>207</GROUP_ID>
 <GROUP_NAME>Auto -Motorrad - Flugzeug</GROUP_NAME>
 <PARENT_ID>202</PARENT_ID>
</CATALOG_STRUCTURE>
<CATALOG_STRUCTURE type="leaf">
 <GROUP_ID>210</GROUP_ID>
 <GROUP_NAME>Oldtimer</GROUP_NAME>
 <PARENT_ID>207</PARENT_ID>
</CATALOG_STRUCTURE>
```

5.3.4 Artikelerstellung

Die Methode `createArticles($sellerId)` aggregiert die zu schreibenenden Artikeldaten. Über einen `INNER JOIN` werden die Tabellen `mercateo_products` und `core_products` verbunden, so

[16] vgl. hierzu: `https://www.sitepoint.com/hierarchical-data-database-2/`

dass über die in `mercateo_products` hinterlegte `core_product_id` die entsprechenden Daten aus der `core_products` Tabelle nachgeladen werden können. Ist der Artikelstatus gleich *„new“* oder *„update“* wird die Methode `writeArticle($product, $articleMode)` aufgerufen, die die entsprechenden XML Elemente schreibt.

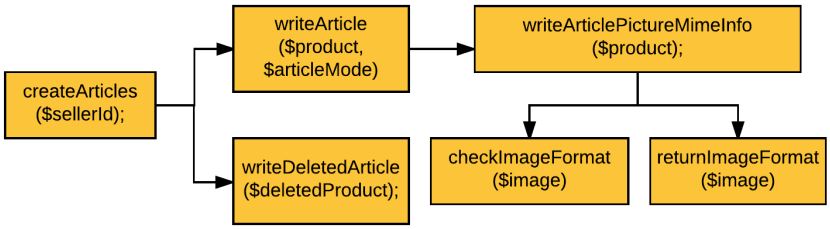

Abbildung 20: Aufrufhierarchie createArticle

Zusätzlich wird geprüft ob die dem Artikel zugeordneten Bilder der Mercateo Spezifikation entsprechen. Diese gestattet als Bildformate nur „.jpeg“ und „.gif“. Entsprechen die Bilddateien nicht diesem Format wird eine entsprechende Ausnahmebehandlung durchgeführt. Dabei wird der komplette Pfad der beanstandeten Datei zurückgeliefert, um es dem Anwender zu erleichtern diesen Fehler zu beheben.

```
2017-01-05 16:30:55 Info: Product with CoreProductId: 7147 updated
Exception: 'png' ist not allowed as File Extension.
Only .gif & .jpg Files are accepted by Mercateo-Marketplace
Check Image Path: https://bild-im-rahmen.com/wp-content/uploads/2016/03/s21r.png
```

Handelt es sich um einen gelöschten Artikel, wird die Methode `writeDeletedArticle($product)` aufgerufen. Diese schreibt die in `mercateo_products` hinterlegten, um einen Artikel als gelöscht auszeichnen zu können notwendigen Informationen - Die SKU & den Titel sowie den Wert des Artikelattributes `mode` (hier `delete`)- in das Dokument.

5.3.5 Kategoriemapping

Mit der Methode `mapArticleToCatalogGroup($sellerId)` werden bei der Transaktion `T_NEW_CATALOG` die Artikel ihren Kategorien zugewiesen. Alle dazu notwendigen Informationen finden sich in der `mercateo_products` Tabelle.

Wird ein Update Katalog erstellt wird die Funktion `updateMapArticleToCatalogGroup($sellerId)` aufgerufen. Sie setzt den bei der Transaktion `T_UPDATE_PRODCUTS` geforderten Attributwert für `mode` entsprechend der Angaben in der `mercateo_products` Tabelle. Mögliche Werte sind *„new“* für neu erstellte und *„delete“* für gelöschte Produkte.

5.4 Die Klasse `CatalogToolsTask`

Die Klasse `CatalogToolsTask` stellt Methoden zur Verfügung die von den drei sie beerbenden Task Klassen verwendet werden.

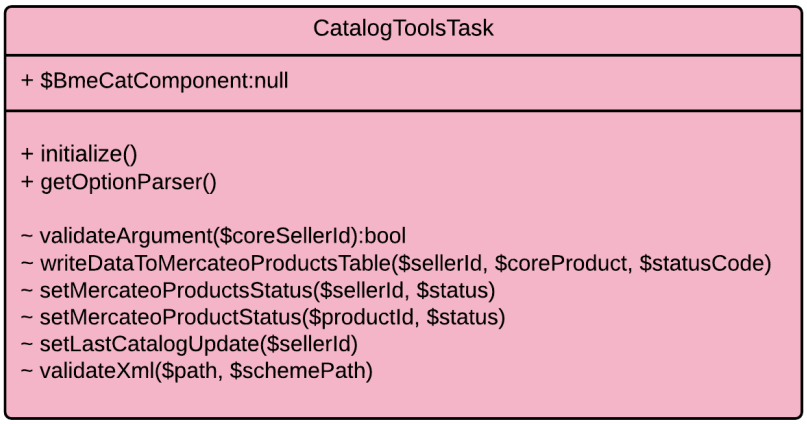

Abbildung 21: UML Diagramm der Klasse CatalogToolsTask

Die Methode `initialize()` lädt alle zur Katalogerzeugung benötigten Model-Klassen und initialisiert die Instanzvariable `BMECatComponent`. Die vom CakePHP-Framework zur Verfügung gestellte Methode `getOptionParser()` überprüft, ob beim Aufruf des Tasks die benötigte `$coreSellerId` übergeben wird. Mit `validateArgument($coreSellerId)` wird geprüft, ob die übergebene `$coreSellerId` im System vorhanden ist. Falls ja, wird `true` zurückgeliefert. Falls nicht, wird eine tabellarische Übersicht der verfügbaren Verkäufer ausgegeben und `false` zurückgegeben. Durch `writeDataToMercateoProductsTable($sellerId, $coreProduct, $statusCode)` werden Einträge in der `mercateo_products`-Tabelle erzeugt, dabei wird mit Aufruf der `MercateoProducts`-Model-Methode `allreadyInDatabase($coreProductId)` sichergestellt, dass keine doppelten Einträge erstellt werden, eine bestimmte `core_product_id` also nur einmal in der Tabelle vorkommt. Das setzten eines bestimmten Produktstatus für *einen* Eintrag in der `mercateo_products`-Tabelle wird mit der Methode `setMercateoProductStatus($productId, $status)` realisiert. Das setzten der Produkstatus aller Produkte eines *CoreSellers* geschieht mit `setMercateoProductsStatus($sellerId, $status)`. Die Methode `setLastCatalogUpdate($sellerId)` schreibt den Timestamp der letzten Katalogerstellung in die `core_configurations`-Tabelle.

> Exkurs:
> In der Tabelle `core_configurations` werden Konfigurationsdaten gespeichert.
>
> Bis dato können Einträge in die Tabelle nur über das GUI des iTool erstellt werden. Die Methode `setSellerConfiguration($coreSellerId, $configurationPath, $configurationValue)` erweitert die Klasse `CoreConfigurationsTable` um die Möglichkeit,

Spalte	Erläuterung
id	Primärschlüssel
core_seller_id	Die Id des Verkäufers für den die Konfiguration gilt
configuration_group	Der Scope in dem die Konfiguration gilt
configuration_path	Nähere Angaben zum Konfigurationswert
configuration_value	Der Konfigurationswert

Tabelle 5: Die Tabelle `core_configurations`

Einträge über einen Methodenaufruf erzeugen zu können. Ist unter dem übergeben Pfad schon ein Wert hinterlegt, so wird dieser aktualisiert. Existiert noch keine Eintrag, wird ein neuer erzeugt.

Die Methode `validateXml($path, $schemePath)` lädt eine Instanz der Klasse `XMLReaderComponent` und öffnet damit die soeben erstellte Datei, welche durch den lesenden Zugriff validiert wird. Der Dateipfad wird der Instanzvariablen `path` der BMECat-Komponente entnommen.

Exkurs:
Um die erzeugte XML Datei mit dem dazugehörigen Schema validieren zu können muss die Klasse `XMLReaderComponent` um diese Funktionalität ergänzt werden. Die dem Component zugrundeliegende `xmlReader`-Klasse stellt dazu eine Methode `setSchema($schemePath)` zur Verfügung die hier zur Anwendung kommt. Wurde ein Schema gesetzt wird mit dem ersten Aufruf der `xmlReader->read())` Methode die zu lesende Datei validiert.

5.4.1 Erzeugung eines initialen Katalogdokumentes mit der Klasse `AddNewCatalogTask`

Die Klasse `AddNewCatalogTask` dient der Erzeugung eines neuen Katalogdokumentes. Der in Kapitel 3.2.4 vorgestellte Entwurf zur Erzeugung eines neuen Katalogdokumentes wird mit den von der Methode `newCatalog($sellerId, BmeCatComponent $BmeCatComponent)` aufgerufenen Funktionen umgesetzt. `initialize()` ruft die gleichnamige Methode der Elternklasse auf. Somit stehen alles dort geladenen Model-Klassen sowie die zur Katalogerzeugung benötigte Instanz der BMECatComponent Klasse zur verfügung. In der `main()`-Methode wird der Aufrufparamter validiert und mit `checkIfMercateoProductsTableContainsSellersProducts($sellerId))` geprüft ob die `mercateo_products` Tabelle bereits Einträge des angegebenen Verkäufers enthält. Ist dies nicht der Fall, wird die Katalogerzeugung durch Aufruf von `newCatalog` angestoßen.

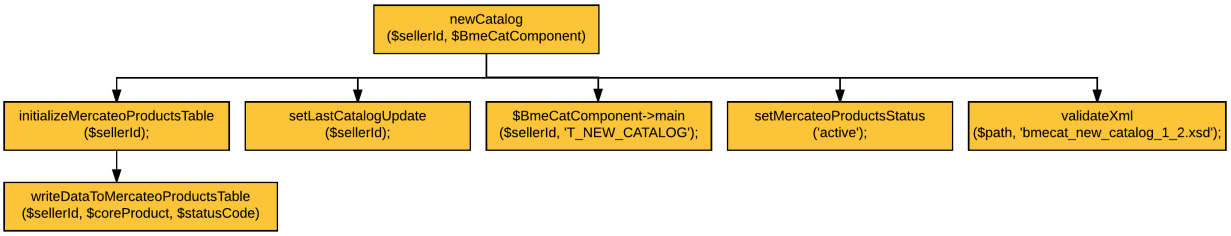

Abbildung 22: Aufrufhierarchie der Methode `newCatalog`

Mit `initializeMercateoProductsTable($sellerId)` wird die `mercateo_products` Tabelle initialisiert. Dabei werden die in `core_products` gespeicherten Produkte des angegebenen Verkäufers geladen und durch Aufruf von `writeDataToMercateoProductsTable` in die `mercateo_products` Tabelle geschrieben. Der Variablen `statusCode` wird dabei der Wert *new* zugewiesen.

Anschließend wird mit `setLastCatalogUpdate` der Timestamp der Katalogerzeugung in die `core_configurations` Tabelle geschrieben.

Durch Aufruf der Methode `createBMECatXMLDocument($sellerId, $transactionType, $prevVersion = null)` des `BmeCatComponent`-Objektes wird die zu erzeugende XML-Datei geschrieben . Dem Parameter `transactionType` wird hier der Wert `T_NEW_CATALOG` zugewiesen. Anschließend wird mit `setMercateoProductsStatus($status)` der *status* aller in `mercateo_products` gespeicherten Artikel auf *active* gesetzt.

Die Validierung des soeben erzeugten Dokumentes erfolgt mit dem Schema `bmecat_new_catalog_1_2.xsd` durch Aufruf von `validateXml($path, $schemePath)`.

5.4.2 Erzeugung eines Update-Katalogdokumentes mit der Klasse `UpdateCatalogTask`

Die Klasse `UpdateCatalogTask` dient der Erzeugung eines Update-Katalog-Dokumentes. Die `initialize()` Methode verhält sich wie im vorherigen Abschnitt beschrieben. Auch die `main()` verhält sich ähnlich, mit dem Unterschied, dass nun positiv darauf geprüft wird, ob die `mercateo_products` Tabelle bereits Einträge des angegebenen Verkäufers enthält.

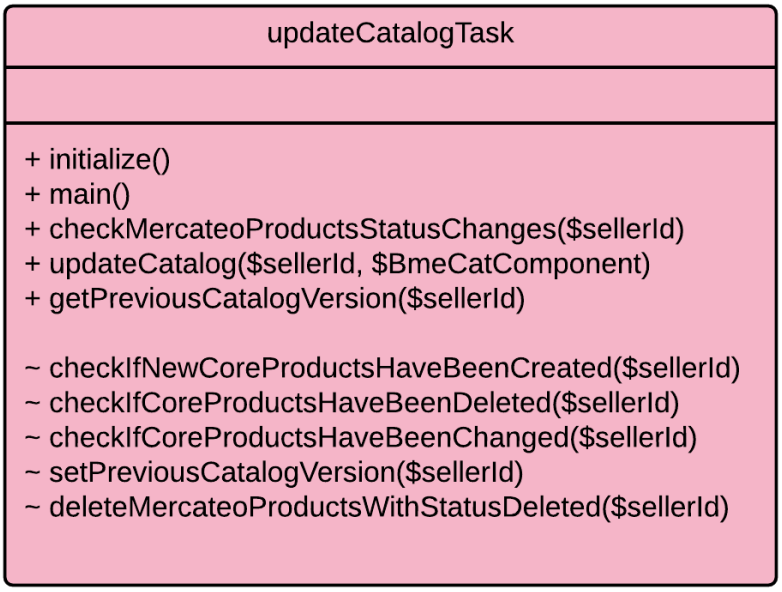

Abbildung 23: UML-Klassendiagramm der Klasse `UpdateCatalogTask`

Die Funktion `checkMercateoProductsStatusChanges($sellerId)` fasst jene 3 Methoden zusammen, die prüfen ob Produkte gelöscht, geändert oder neu hinzugefügt wurden. Diese geben jeweils `true` zurück, falls eine entsprechende Änderung stattgefunden hat. Zugleich wird in der Konsole eine Meldung der Form

```
Info: Product with CoreProductId: 1781 added
Info: Product with CoreProductId: 1782 deleted
Info: Product with CoreProductId: 1783 updated
```

ausgegeben, die zudem in der Datei *productChange.log* erfasst wird.

Die Methode `checkIfNewCoreProductsHaveBeenCreated($sellerId)` überprüft ob es seit der letzten Katalogerstellung in der Tabelle `core_products` neue Einträge gab. Dazu wird die Tabelle `mercateo_products` über einen LEFT-JOIN an `core_products` gebunden. All jene Produkte aus `core_products`, deren *core_product_id* nicht in `mercateo_products` zu finden ist müssen als neu gelten und werden demnach mit dem Statuscode `new` in `mercateo_products` geschrieben.

Mit `checkIfCoreProductsHaveBeenDeleted($sellerId)` wird geprüft ob Daten aus der `core_products` Tabelle gelöscht wurden. Auch hier wird `mercateo_products` über einen LEFT-JOIN an `core_products` gebunde. All jene Produkte deren *core_product_id* noch in der `mercateo_products` Tabelle vermerkt ist, nicht aber in `core_products`, müssen als gelöscht gelten. Entsprechend wird der Status der betroffenen Produkte in `mercateo_products` auf `delete` gesetzt.

Durch die Methode `checkIfCoreProductsHaveBeenChanged($sellerId)` schließlich wird geprüft ob sich Produktdaten seit der letzten Katalogerstellung geändert haben. Dazu wird die Tabelle `core_product_updates` über einen INNER-JOIN an `mercateo_products` gebunden. Beim Erstellen des Kataloges wurde der Zeitpunkt der Erzeugung in der `core_configurations` Tabelle gespeichert. All jene Einträge aus `core_product_updates`, deren Erzeugungsdatum nach der letzten Katalogerstellung liegt werden in `mercateo_products` mit dem Status `update` versehen.

Hat eine der soeben vorgestellten Methoden `true` zurückgeliefert, wird die Methode `UpdateCatalog($sellerId, $BmeCatComponent)` aufgerufen, die alle an der Erstellung eines Update-Katalogdokumentes beteiligten Methoden zusammenfasst.

Abbildung 24: Aufrufhierarchie der Methode `UpdateCatalog`

Zu Beginn wird mit `setLastCatalogUpdate($sellerId)` der Zeitpunkt der Katalogerstellung in der `core_configurations` Tabelle gespeichert.

Die Methode `getPreviousCatalogVersion($sellerId)` liefert den in `core_configurations` gespeicherten Wert des bei der Transaktion `T_UPDATE_PRODUCTS` benötigten Attributes `prev_version` zurück. Existiert noch kein Eintrag in der Tabelle wird dieser erstellt und der Wert des Attributes auf „0“ gesetzt. Bei der ersten Ausführung von `T_UPDATE_PRODUCTS` wird so stets „0“ zurückgeliefert. Dieser Wert wird in der Variablen `$prevCatalogVersion` gespeichert um bei Aufruf der BMECatComponent-Methode `createBMECatXMLDocument` - diesmal wird der Parameter `$transactionType` mit `T_UPDATE_PRODUCTS` initialisiert - übergeben werden zu können.

Wurde der Updatekatalog erstellt müssen die Einträge aus der `mercateo_products` Tabelle gelöscht werden, deren Status auf `delete` gesetzt ist. Die Methode `deleteMercateoProductsWithStatusDeleted($sellerId)` setzt dies um.

Anschließend bekommen die verbliebenen Einträge den Status `active` zugewiesen.Mit `setPreviousCatalogVersion($sellerId)` wird der entsprechende Wert in `core_configurations` um 1 erhöht.

Die Validierung des Katalogdokumentes erfolgt diesmal mit dem Schema `bmecat_update_products_1_2.xsd`.

5.4.3 Die Klasse DeleteCatalogTask

Das Löschen von Produkten eines bestimmten Verkäufers wird mit der Klasse `DeleteCatalogTask` realisiert.

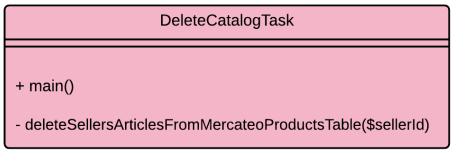

Abbildung 25: Klassendiagramm DeleteCatalogTask

In der `main()`-Methode wird die Validierung des übergebenen Arguments durchgeführt und der Nutzer gefragt, ob er sich seines Handelns sicher ist. Falls ja, werden mit deleteSellersArticlesFromMercateoProductsTable($sellerId) die Produkte des entsprechenden Verkäufers aus der `mercateo_products`-Tabelle gelöscht.

5.4.4 Ressourcenmanagement

Die XML-Knoten im BMECat-Dokument können auch, teilweise komfortabler, mit anderen in PHP zur Verfügung stehenden XML-Manipulationsklassen (wie z.B. DOMDocument) geschrieben werden. Bei der Verwendung der DOMDocument Klasse wird zunächst der komplette Objektbaum aufgebaut und im Arbeitsspeicher gehalten, bevor das XML-Dokument geschrieben werden kann. Bei mehreren zehntausend zu exportierenden Produkten kann es so passieren, dass der zur Verfügung stehende Arbeitsspeicher nicht mehr ausreicht bzw. verhältnismäßig viel Speicher beansprucht wird. Die Verwendung der XMLWriter-Klasse gestattet es die Anzahl der im Schreibpuffer gehaltenen Elemente zu limitieren und diese bei Erreichen der Obergrenze zu schreiben um anschließend den Puffer zu leeren. Dadurch wird der Speicherverbrauch beim Schreibvorgang drastisch reduziert.

Lese- und Schreibzugriffe auf die Datenbank erfolgen stets sequentiell, wie folgendes Codebeispiel illustriert:

```
        $limit = 100;
        $page = 1;

        do {
```

```
            $products = $this->CoreProducts->find('all')
                ->where($conditions)
                ->limit($limit)
                ->page($page);

            if (!empty($products)) {
                foreach ($products as $product) {
                    $this->writeDataToMercateoProductsTable($product->id, 'new');
                }
            }$page++;
        } while (count($products->toArray()) == $limit);
```

Durch diese Maßnahme wird der Speicherverbrauch bei Datenbankzugriffen klein gehalten.

5.5 Bestandsdatenabfrage

Die Bestandsdatenabfrage ist in der Controllerklasse `AvailabilityController` implementiert.

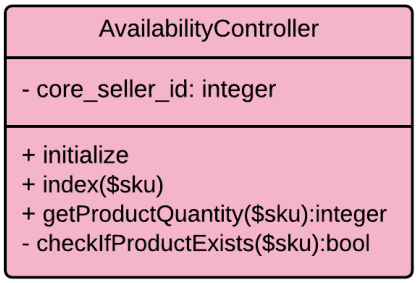

Abbildung 26: Klassendiagramm AvailabilityController

Für das Zurückliefern der Bestandsdaten einer angefragten SKU wird keine View benötigt. CakePHP versucht jedoch automatisch zu jeder aufgerufenen Controllermethode eine entsprechende View zu rendern [17]. Da jedoch dennoch auf die `index($sku)` Methode zugegriffen werden soll wird in der Funktion `initialize` das automatische Rendern einer View abgeschaltet, sowie der direkte Zugriff auf die textttindex($sku) Methode gestattet.

Die Funktion `checkIfProductExists($sku)` prüft zunächst ob sich das angefragte Produkt in der `mercateo_products` Tabelle finden lässt.Ist dies der Fall, wird zusätzlich geprüft ob es auch in der `core_products` Tabelle gefunden werden kann. Falls beides zutrifft, wird `true` zurückgeliefert, andernfalls `false`.

Die `index($sku)` Methode verarbeitet die Anfrage. Kann das angefragte Produkt in der Datenbank gefunden werden,wird über Aufruf der Methode `getProductQuantity($sku)`) die Bestandsmenge desselben abgefragt und als Text im Browserfenster ausgegeben. Zusätzlich wird der HTTP-Statuscode

[17] vgl. hierzu Kapitel 1.3.1 - Convention over Configuration in CakePHP

200 zurückgeben um das anfragenden System darüber zu informieren, dass die Anfrage erfolgreich bearbeitet werden konnte.

Können keine Produkdaten gefunden werden, wird im Browserfenster eine entsprechende Browsermeldung ausgegeben und der HTTP-Statuscode *204* zurückgeben. Das anfragende System erlangt so Kenntnis darüber, dass die Anfrage verabeitet werden konnte, jedoch kein Inhalt zurückgeliefert werden kann.

6 Test

CakePHP unterstützt „ab Werk“ Unit Testing mit PHPUnit, welches in vorliegender Arbeit verwendet wird. -ETWAS ÜEBR DIE KONVENTIONEN SCHREIBEN- Um jene Methoden testen zu können die lesend und schreibend auf die Datenbank zugreifen, werden für die betroffenen Tabellen *Fixtures* erstellt. Fixtures sind Duplikate der eigentlichen Tabellen und enthalten Testdatensätze. Der Vorteil von Fixtures ist, dass Datenbankabfragen durchgeführt werden können ohne, dass die eigentlichen Datensätze davon betroffen wären.

Im folgenden werden die einzelnen Testklassen und Testfälle vorgestellt.

6.1 Die Klasse AvailabilityControllerTest

Um die Methoden `getProductQuantity` und `checkIfProductExists` testen zu können müssen Fixtures für die Tabellen `mercateo_products`, `core_products`, `core_marketplaces` und `core_product_quantities` erstellt und mit Testdatensätzen befüllt werden. `testCheckIfProductExists()` enthält je einen Positiv- und einen Negativtest. Es wird erwartet, dass die Methode `checkIfProductExists` den booleschen Wert `true` zurückliefert, falls ein Produkt in der Datenbank gefunden werden konnte und `false`, wenn nicht. Die Methode `testGetProductQuantity` prüft ob der Rückgabewert dem erwarteten Zahlenwert entspricht und ob er vom Typ *Integer* ist.

6.2 Die Klasse UpdateCatalogTaskTest

In der Klasse `UpdateCatalogTaskTest` werden folgende Fixtures geladen:

- `core_configurations`
- `mercateo_products`
- `core_products`
- `core_product_quantities`
- `core_product_types`
- `core_product_updates`

`testGetPreviousCatalogVersion()` testet positiv und negativ auf einen in der `core_configuartions`-Fixture hinterlegten Konfigurationswert. Zudem wird geprüft, ob „0“ zurückgeliefert wird, falls für den angegeben Seller noch kein Konfigurationswert in der Datenbank angelegt wurde.

Mit `testCheckMercateoProductsStatusChanges()` werden die als *private* ausgezeichneten Methoden `checkIfNewCoreProductsHaveBeenCreated`, `checkIfCoreProductsHaveBeen-Deleted` und `checkIfCoreProductsHaveBeenUpdated` auf Funktion getestet. Für Testfall 1 - jeweils ein Produkt wurde hinzugefügt, gelöscht bzw. aktualisiert- werden entsprechende Einträge in den Fixturetabellen von `core_products`, `mercateo_products` und `core_product_updates` angelegt. Als Rückgabewert wird demzufolge `true` erwartet. Entsprechendes geschieht für den Fall, dass keine Produktstatusveränderungen stattgefunden haben sollen und als Rückgabewert `false` erwartet werden kann.

6.3 Die Klasse MercateoAccountsTableTest

Die Funktion `validateCatalogVersionFormat($value)` erweitert die CakePHP Standardvalidatormethoden. Mittels regulärem Ausdruck wird geprüft ob die übergeben Zeichenkette einem bestimmten Format entspricht. `testValidateCatalogVersionFormat()` führt einen Positiv- und einen Negativtest der Funktionalität durch.

6.4 Die Klasse BMECatComponentTest

Um die Methoden `getSellerName($sellerId)` und `getParentCoreCategoryIds($sellerId)` testen zu können werden die Fixtures für `core_sellers` und `core_categories` geladen. `test-GetSellerName()` prüft mit einem Positiv- und einem Negativtest ob die zurückgelieferte Zeichenkette der Erwartung entspricht.

Die Methode `testGetParentCoreCategoryIds()` prüft ob ein Array zurückgeliefert wird und ob die darin gespeicherten Werte vom Typ *Integer* sind.

`testArticleDataValidator()` führt einen Positiv- und einen Negativtest durch, indem jeweils eine „gültige“ bzw. „ungültige“ Instanz von *CoreProducts* an `articleDataValidator($product)` übergeben wird.

Mit `testCheckImageFormat()` wird geprüft, ob `checkImageFormat($image)` jeweils `true` zurückliefert, wenn der übergebene *URI* auf „.jpeg“, „.jpg“ oder „gif“ endet. Es wird `false` erwartet, wenn die Datei eine andere Endung hat.

1. Martin, Robert C.: Clean Code - Refactoring, Patterns, Testen und Techniken für sauberen Code : Deutsche Ausgabe. 1. Aufl.. Heidelberg: MITP-Verlags GmbH & Co. KG, 2013.