

Projekt Anwendungsintegration

M. Sc. Informatik

Duff Brewery

TSD Brand Owner



Gliederung

- Aufgabenstellung und Ideen zur Lösung
- Umsetzung
 - Architektur
 - Artikel
 - Datenbankanzbindung
 - BO → DA Interface
 - Ausgabe von XML
 - Maintenance
- Demo
- Fazit



Aufgabenstellung und Ideen zur Lösung



Aufgaben

- Artikelstammdaten verwalten
- REST-Interface zu einem TSD Data Aggregator
- Optional: Web-Oberfläche



Resultierende Arbeitspakete

- Artikel erfinden
- Kennenlernen des BO → DA Interface
- Datenstruktur verstehen
- Artikel in gewünschter Datenstruktur beschreiben
- Artikel in Datenbank pflegen
- BO → DA Interface umsetzen
- Web-Oberfläche entwickeln
- Testen



Ideen

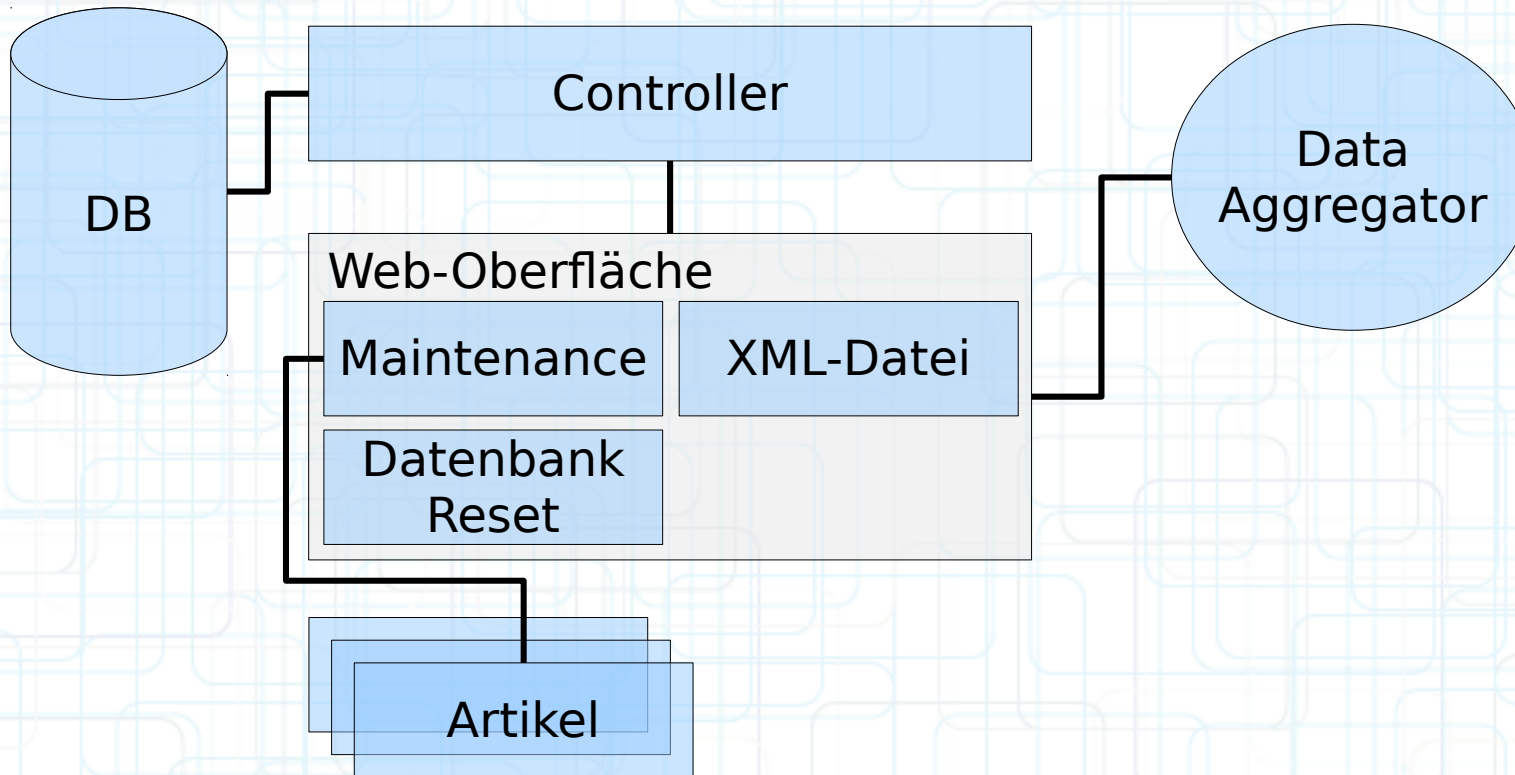
- Wir sind die Bierbrauerei Duff Brewery (Anlehnung an die Serie „The Simpsons“)
- Verschiedene Biersorten (Pils, Alkoholfrei, usw.) zum Testen des Prototyps
- Hauptaspekt der Web-Oberfläche ist Maintenance
- Nutzen des Rails-Frameworks
 - RESTful
 - Datenbankanbindung
 - XML-Builder



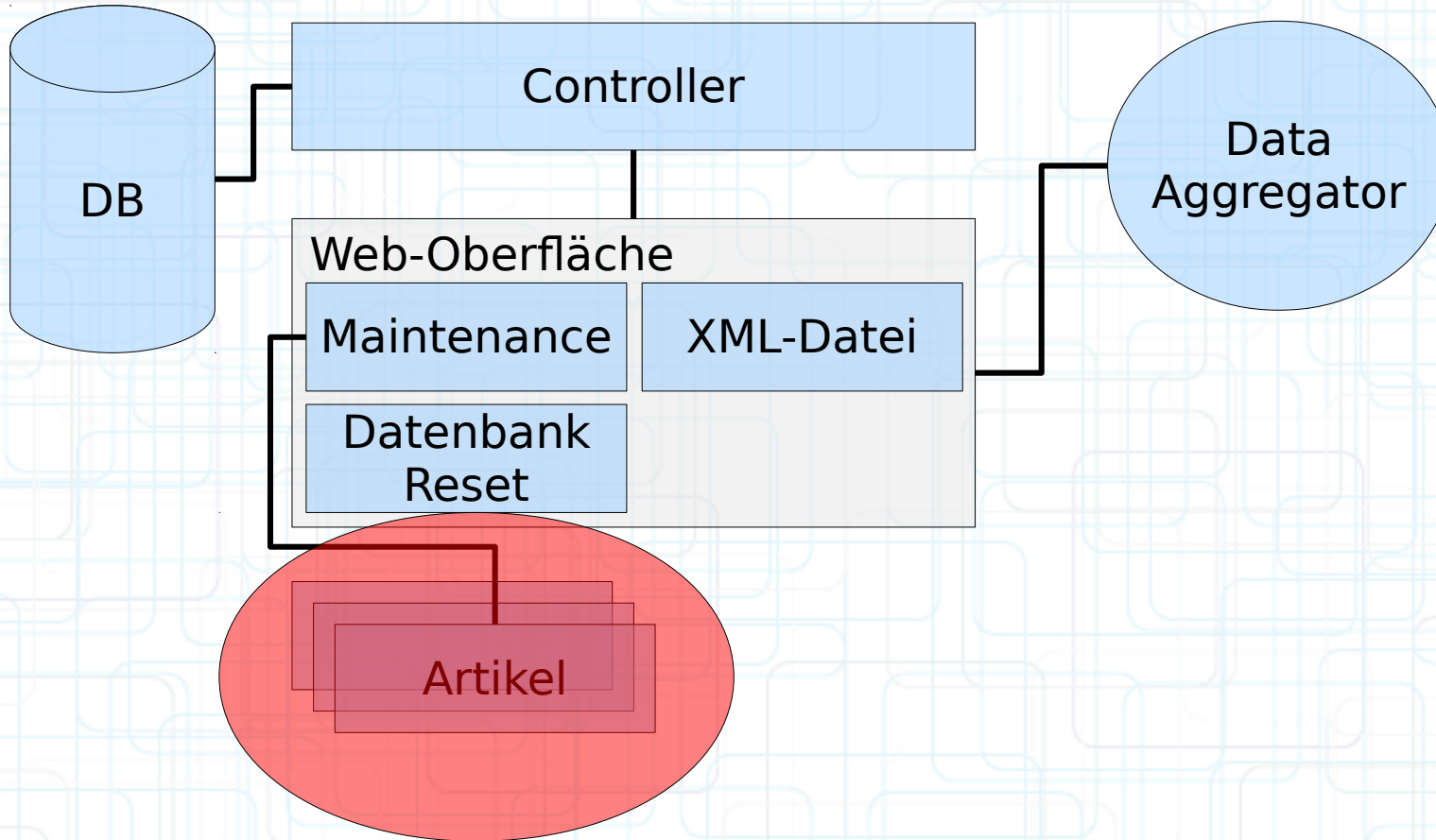
Umsetzung



Architektur



Artikel (1)



Artikel (2)

- Welche Informationen gibt es zu den Artikeln?
 - Welche TSD-Module brauchen wir?
 - Welche Module machen überhaupt Sinn?
 - Absprache mit IAP (Problematisch, Teufelskreis)
- Entscheidung:
 - BasicProductInformation (Pflicht)
 - FoodAndBeverageIngredientInformation
 - ProductQuantityInformation

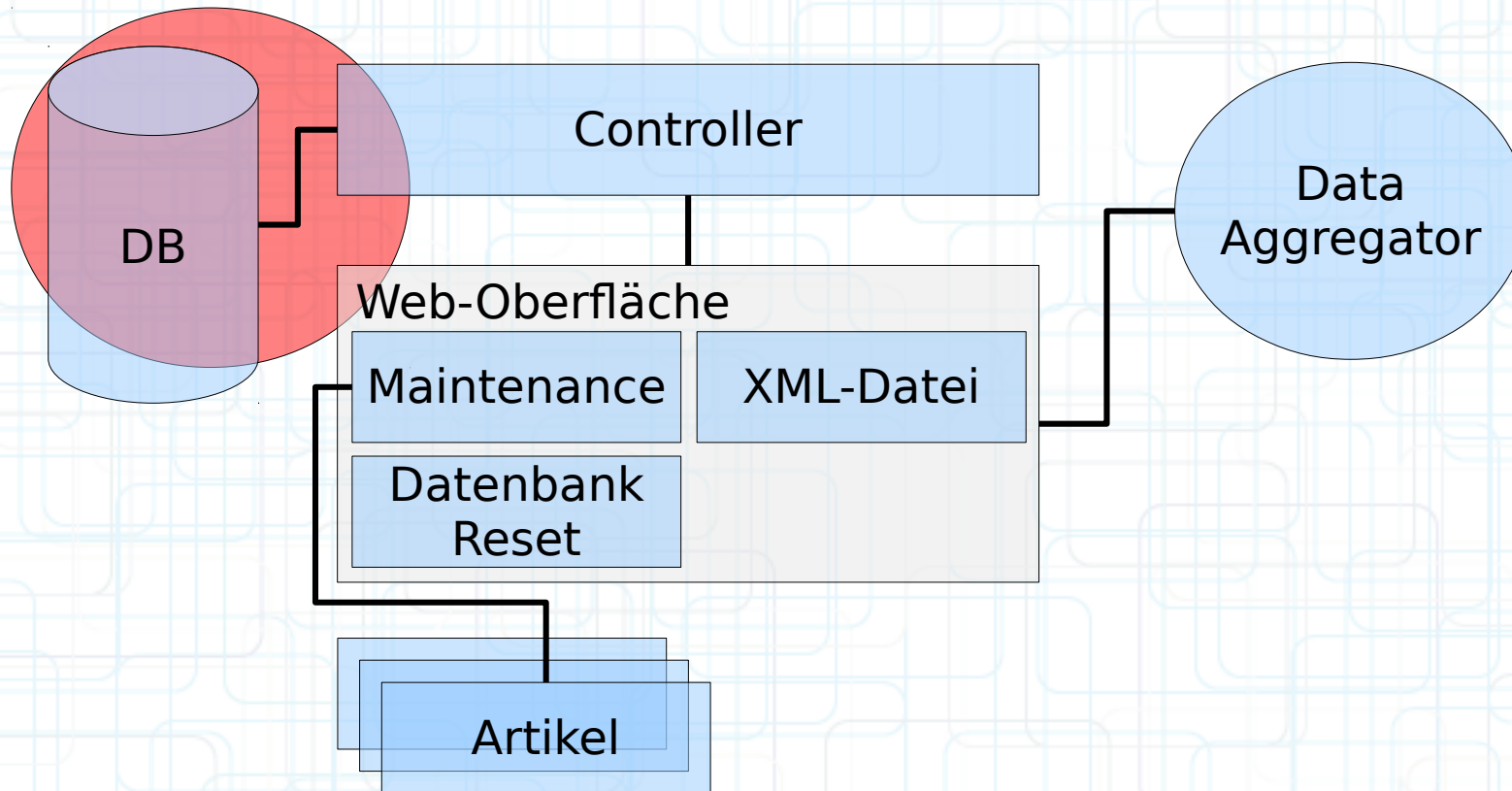


Artikel (3)

- Sehr sehr hoher Zeitaufwand
- Was wird in den einzelnen Elementen spezifiziert?
- Beispiele:
 - VariantDescription
 - ProductName ↔ RegulatedProductName
 - BrandName
 - ConsumerMarketingDescription
- Beispiel XML-Datei absolut nicht hilfreich



Datenbankanbindung (1)



Datenbankanbindung (2)

- Rails arbeitet mit Migrations und Models
- Grundlage/Root: XML-Schema (ProductData.xsd)
- Modellierung der Artikel mit Schema-Regeln
 - Typen (ComplexType, SimpleType, usw.)
 - Restriktionen (Länge, Anzahl Ziffern, usw.)
 - Relationen (Anzahl von erlaubten Elementen)
- Hoher Zeitaufwand (evtl. überflüssig?)
- Alternative: Einfache Key → Value Paare in Datenbank für jedes Produkt



Datenbankanbindung (3)

- Beispiel: Model

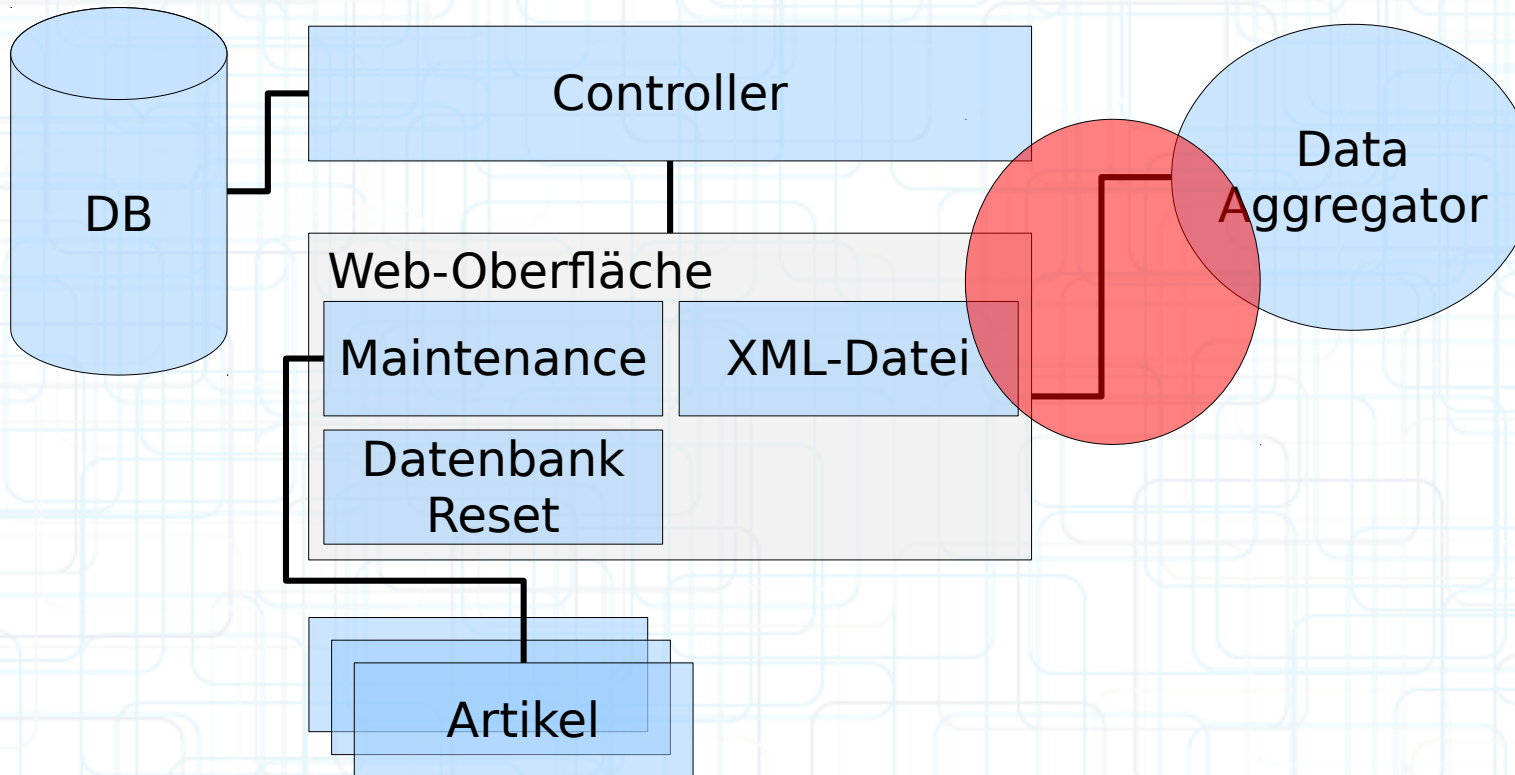
```
class ProductInformationLink < ActiveRecord::Base
  attr_accessible :url
  belongs_to :basic_product_information
  belongs_to :product_information_type_code_type
  belongs_to :language_type_code_type
  validates :product_information_type_code_type, :length => { :minimum => 1 }
end
```

- Beispiel: Migration

```
[...]
  create_table :product_information_links do |t|
    t.string :url, limit: 2500, null: false
    t.belongs_to :basic_product_information
    t.belongs_to :product_information_type_code_type
    t.belongs_to :language_type_code_type
  end
[...]
```



BO → DA Interface (1)



BO → DA Interface (2)

- Produkte hinzufügen/löschen/ändern
- Zusätzlich: XML von DA auslesen
- Zusätzlich: Alle Produkte aus unserer Datenbank hinzufügen/entfernen



BO → DA Interface (3)

- POST (Hinzufügen)
 - /service/v1/product_data/gtin
 - clientGln, mac
 - Request body: XML-Datei
 - Response body: Leer
- PUT/DELETE (Ändern/Hinzufügen)
 - /service/v1/product_data/gtin/<gtin>
 - clientGln, mac, targetMarket
 - Request body: XML-Datei bzw. leer
 - Response body: Leer



BO → DA Interface (4)

- GET (Auslesen)
 - /service/v1/product_data/gtin/<gtin>
 - clientGln, mac, targetMarket
 - Request body: Leer
 - Response body: XML-Datei



BO → DA Interface (5)

- Ruby HTTP Gem

```
requi = '/service/v1/product_data/gtin/%014d' % gtin
requi += '?clientGln=2865195100007'
requi += '&targetMarket=276'
key = ':'
hmac = Digest::HMAC.hexdigest(requi, key, Digest::SHA256)
request = Net::HTTP::Delete.new(requi + '&mac=' + hmac.upcase)
Net::HTTP.new("1.1.1.1", 1111).request(request)
```

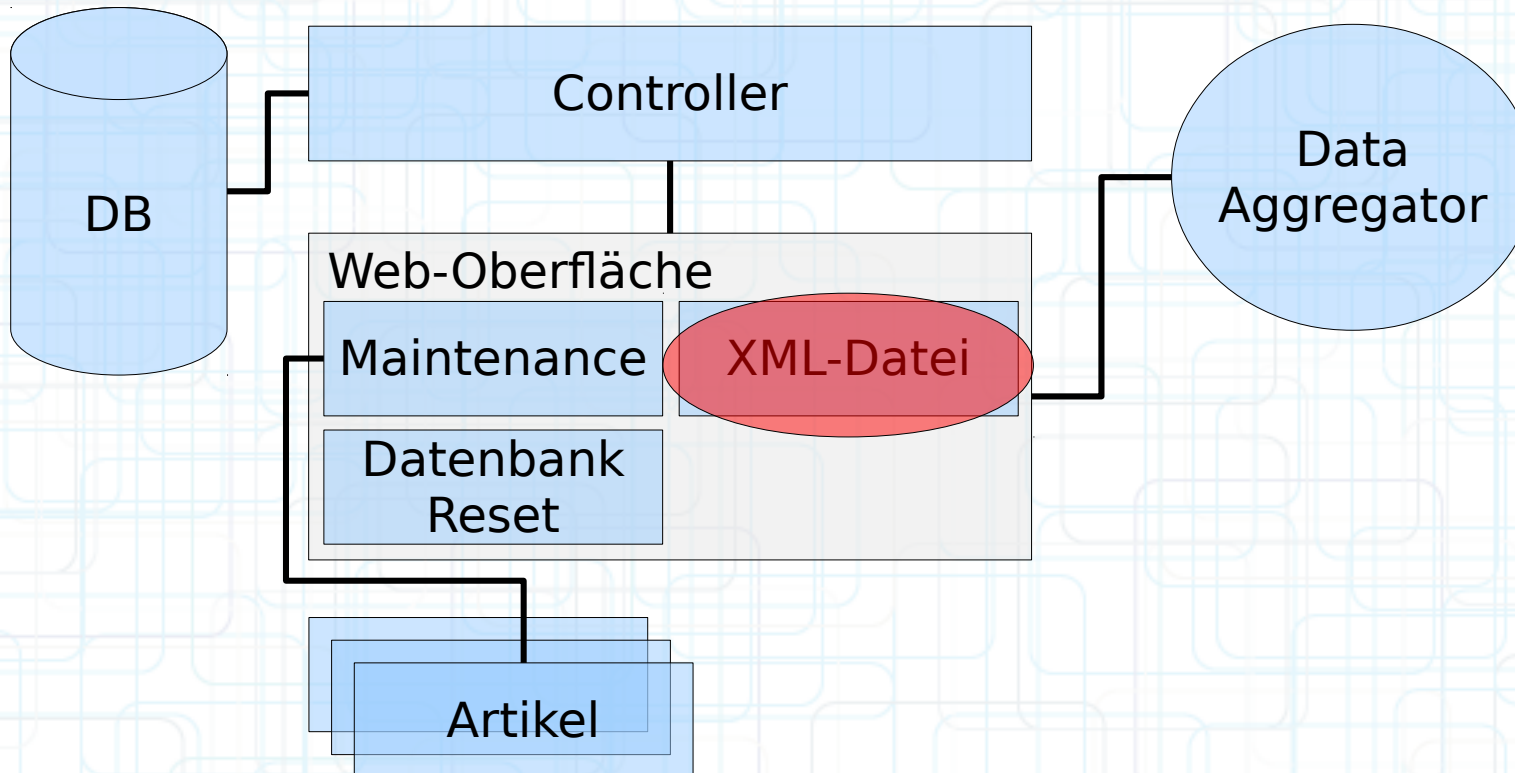


BO → DA Interface (6)

- Keine Fehlerbehandlung
 - Status wird in einem „Gut-Abschnitt“ des Web-UI angezeigt
 - Exceptions werden in einem „Schlecht-Abschnitt“ des Web-UI angezeigt
 - Keine Unterscheidung zwischen 2xx und 4xx Status-Codes (Benutzer reagiert entsprechend)
- Für optimale Fehlerbehandlung evtl. XML-Exceptions notwendig



Ausgabe von XML (1)



Ausgabe von XML(2)

- XML vom Data Aggregator
 - passthrough
- XML mit Produkt aus der Datenbank
 - GTIN Parameter
 - Datenbankabfrage
- Ruby Builder Gem
- Einfaches Anlegen einer Datei mit Endung .xml.builder genügt (hier nicht)



Ausgabe von XML(3)

```
xml.instruct! :xml, :version => '1.0', :encoding => 'UTF-8'
```

```
xml.pd(:productData,
```

```
  'xsi:schemaLocation' => 'urn:gs1:tsd:product_data:xsd:1  
tsd/ProductData.xsd',
```

```
  'xmlns:xsi' => 'http://www.w3.org/2001/XMLSchema-instance',
```

```
  'xmlns:pd' => 'urn:gs1:tsd:product_data:xsd:1') do
```

```
  xml.gtin ("%014d" % product[:gtin])
```

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<pd:productData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

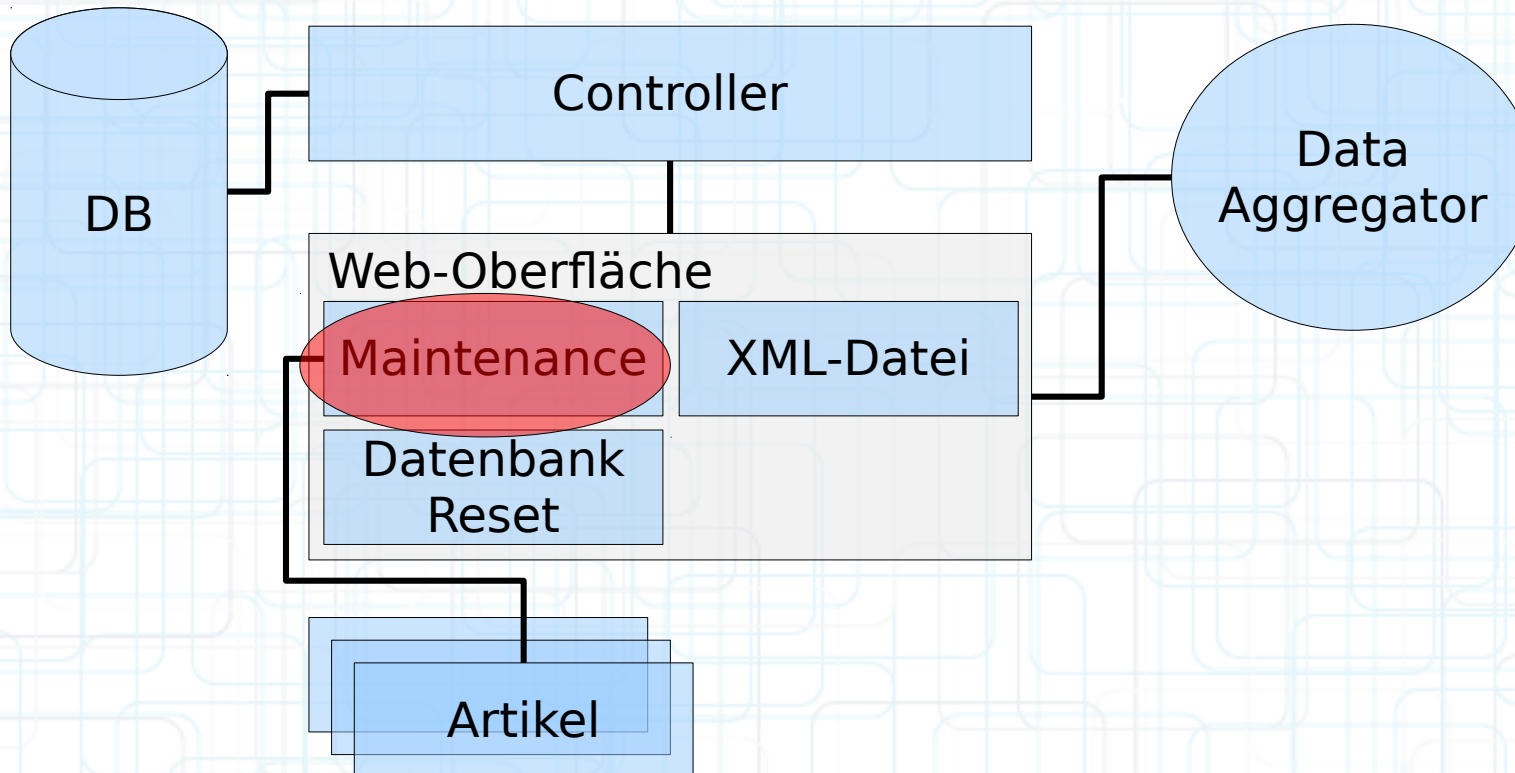
```
  xmlns:pd="urn:gs1:tsd:product_data:xsd:1"
```

```
  xsi:schemaLocation="urn:gs1:tsd:product_data:xsd:1  
tsd/ProductData.xsd">
```

```
<gtin>02865195100014</gtin>
```



Maintenance (1)




Maintenance (2)

- HTML5, Javascript, Bootstrap
- Eigenhändig:
 - Layout
 - Dynamisches Hinzufügen/Löschen von Eingabemasken
- Funktionen:
 - Auflisten aller Produkte mit GTIN
 - Produkte bearbeiten (jeweils deutsche und englische Bezeichnungen)
 - Interaktion mit Data Aggregator

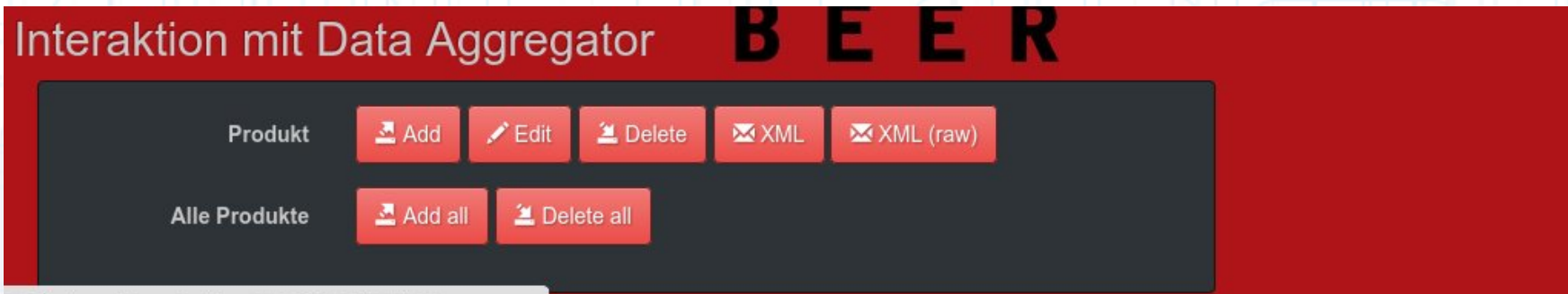


Maintenance (3)

Produktinformationen		Logo
GTIN	<input type="text" value="02865195100120"/>	
Variante	<input type="text" value="Variante"/>	
Produktname (DE)	<input type="text" value="Deutsches Produkt"/>	
Produktname (EN)	<input type="text" value="Englisches Produkt"/>	
Regulierter Produktname (DE)	<input type="text" value="Deutscher regulierter Produktname"/>	
Regulierter Produktname (EN)	<input type="text" value="Englischer regulierter Produktname"/>	
Werbespruch (DE)	<input type="text" value="Deutsche Marketingbeschreibung"/>	
Werbespruch (EN)	<input type="text" value="Englische Marketingbeschreibung"/>	
Informations-URL (WEBSITE)	<input type="text" value="http://eta-ori.net:8080/item.xml?gtin=02865195100120"/>	
<div><input type="button" value="+ Add"/> <input type="button" value="✎ Edit"/> <input type="button" value="- Delete"/> <input type="button" value="✉ XML"/></div>		



Maintenance (4)



Demo

- Bearbeiten eines Produkts
- Hinzufügen eines Produkts
- Hinzufügen/Ändern/Löschen beim Data Aggregator (jeweils mit Kontrolle)
- Funktionstest: Alle Artikel löschen und danach wieder hinzufügen
- XML-Datei eines Beispielsprodukts
- Validierung dieser XML-Datei



Fazit

- Interfaces einfach zu implementieren, da gut beschrieben
- Elemente der XML-Schemas für Außenstehende nicht verständlich
- Fehlerbehandlung nur durch Nutzen von HTTP Statuscodes schwer
- Zeitaufwand für Implementierung gering
- Auffrischen WBA-Kenntnisse
- Insgesamt: Positiver Lerneffekt



Vielen Dank fürs Zuhören!

