**Hochschule für Technik, Wirtschaft und Kultur Leipzig**

Fakultät Informatik, Mathematik und Naturwissenschaften
Bachelorstudiengang Medieninformatik

Bachelorarbeit
zur Erlangung des akademischen Grades

**Bachelor of Science (B.Sc.)**

# Integration des Tacton Produktkonfigurators in ein Open Source Shopsystem

**Autor:** Philipp Anders
philipp.anders.pa@gmail.com

**Betreuer:** Prof. Dr. Michael Frank (HTWK Leipzig)
Dipl.-Wirt.-Inf. (FH) Dirk Noack
(Lino GmbH)

**Abgabedatum:** 29.09.2015

# Kurzfassung

Das ganze auf Deutsch.

# Abstract

Das ganze auf Englisch.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Listing-Verzeichnis

# Abkürzungsverzeichnis

**ATO** Assemble-to-Order
**CSP** Constraint Satisfaction Problem
**ETO** Engineer-to-Order
**MC** Mass Customization
**MTO** Make-to-Order
**PTO** Pick-to-Order
**RPC** Remote Procedure Call
**WSDL** Web Service Description Language

# 1 Einleitung

## 1.1 Ziel der Arbeit

## 1.2 Aufbau der Arbeit

# 2 Grundlagen

## 2.1 Konfiguration

Im Folgenden Abschnitt wird geklärt: was ist ein Konfigurator und wofür brauchen wir ihn? Aus Gründen der Nachvollziehbarkeit werden diese Fragen allerdings in umgedrehter Reihenfolge bearbeitet. Dazu wird zunächst eine marktwirtschaftliche Situation erläutert, welche die Notwendigkeit hybrider Wettbewerbsstrategien begründet. Diese wiederum stellen zu ihrer Umsetzbarkeit Bedingungen an Produktionskonzepte, welche daraufhin vorgestellt werden. Abschließend wird geklärt, welchen Beitrag die Produktkonfiguration dabei leistet und dessen Funktionsweise erläutert.

### 2.1.1 Ausgangssituation

Neue Wettbewerbsbedingungen und gesteigerte Kundenansprüche haben den Druck auf Industrieunternehmen zur Produktion individualisierter Produkte erhöht (Piller, 1998). Dies erfordert eine stärkere Leistungsdifferenzierung(Lutz, 2011). Die Differenzierung ist ein der von Porter beschriebenen „generischen Wettbewerbsstrategien“. Weitere sind die Kostenführerschafts- und Fokussierungsstrategie. Die Kostenführerschaft bezieht sich klassischerweise auf Anbieter von Massenproduktion. Durch die Unterbietung der Konkurrenzpreise wird der Marktanteil erhöht. Dem Gegenüber entspricht die Leistungsdifferenzierung dem Verkauf von Produkten mit höherem individuellen Kundennutzen zu höheren Preisen.

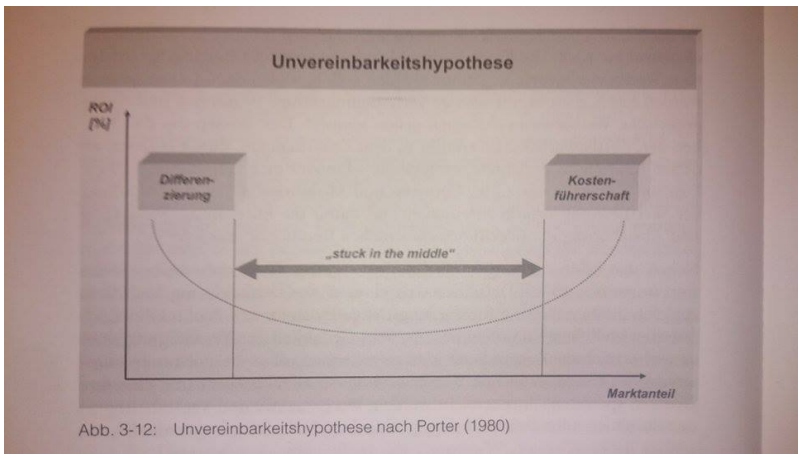



Abbildung 1: Unvereinbarkeitshypothese nach Porter (1980), zitiert von Schuh (2005)

In diesem Zusammenhang formulierte Porter die Unvereinbarkeitshypothese. So sollen Kostenführerschaft und Leistungsdifferenzierung nicht gleichzeitig erreichbar sein.

Eine uneindeutige Positionierung führe zu einem „stuck in the middle“ und damit zur Unwirtschaftlichkeit, wie Abbildung 1 darstellt.

Die Beobachtung der Unternehmensrealität zeichnet jedoch ein anderes Bild. Neue Organisatzionsprinzipien, Informationsverarbeitungspotentiale und Produktstrukturierungsansätze ermöglichen einen Kompromiss aus Preis- und Leistungsführerschaft (Schuh, 2005). Das Ergebnis wird als hybride Wettbewerbsstrategien bezeichnet. Eine dieser Strategien ist die sogenannte Mass Customization (MC).

MC ist die „Produktion von Gütern und Leistungen für einen (relativ) großen Absatzmarkt, welche die unterschiedlichen Bedürfnisse jedes einzelnen Nachfragers dieser Produkte treffen, zu Kosten, die ungefähr denen einer massenhaften Fertigung vergleichbarer Standardgüter entsprechen“ (Piller, 1998). Mit anderen Worten: Preisvorteil (i.d.R. durch Massenfertigung) wird mit Individualisierung (i.d.R. durch Variantenvielfalt) verbunden.

### 2.1.2 Produktklassifizierung

MC setzt zu deren Umsetzbarkeit gewisse Bedingungen an die Produktionsweisen der abgesetzten Güter. Im Folgenden wird eine Klassifizierung von Produkten in Bezug auf deren Herstellung vorgestellt, was von Schuh (2006) als Produktionskonzepte bezeichnet wird. Abgestimmt auf diese Produktionskonzepte lässt sich daraufhin der jeweilige Nutzen einer Produktkonfiguration ableiten.

Folgende Produktionskonzepte werden unterschieden (nach Schuh 2006, zitiert von Lutz 2011):

- **Pick-to-Order (PTO):** Herstellung ohne Kundenauftrag. Lagerhaltung ganzer Produkte. Keine Abhängigkeit dieser Produkte untereinander. Keine Berücksichtigung von Kundenanforderungen.
- **Assemble-to-Order (ATO):** Herstellung ohne Kundeauftrag. Lagerhaltung von Baugruppen/-teilen. Teile mit Abhängigkeiten untereinander. Berücksichtigung von Kundenanforderungen.
- **Make-to-Order (MTO):** Herstellung teilweise erst nach Kundenenauftrag. Lagerhaltung von standardisierten Baugruppen/-teilen sowie Produktion von vorgedachten Komponenten nach Kundenanforderung. Abhängigkeiten zwischen Teilen. Keine unendliche Anzahl von Varianten.
- **Engineer-to-Order (ETO):** Herstellung erst nach Kundenauftrag. Wenig Lagerhaltung von Baugruppen/-teilen. Entwicklung und Fertigung von Teilen nach Kundenspezifikation. Unendliche Variantenanzahl möglich.

Die Produktionskonzepte unterscheiden sich hauptsächlich nach dem Kriterium, wann die Produktion der Baugruppen/-teile beginnt - vor oder nach Auftragsspezifikation. Eine Produktion vor Auftragseingang, also ohne Kundenspezifikation, erlaubt Lagerhaltung. Die Einflussnahme des Kunden auf das Endprodukt - abgesehen von der Kaufentscheidung - beginnt bei ATO. Lutz kommt zu dem Schluss, dass MTO am ehesten der Definition individualisierter Güter im Rahmen der MC entspricht. Während sie einen Grundanteil standardisierter Komponenten besitzen, sind sie doch im ausreichenden Maße durch den Kunden anpassbar.

**Zwischenfazit**
In Abschnitt 2.1.2 wurde dargestellt, wie bestimmte Produktionskonzepte die Herstellung individualisierter Produktvarianten bei gleichzeitiger Lagerfertigung ermöglichen. Produkte werden mit dem Ziel gestaltet, so individuell und auftragsunabhängig wie möglich zu sein. Damit wurde eine der Schlüsselfaktoren für die Ermöglichung der hybriden Wettbewerbsstrategie MC erläutert. Diese verbindet die Vorteile effizienter Massenproduktion mit denen der kundenspezifischen Einzelfertigung(Piller, 1998).

### 2.1.3 Produktkonfiguration

Aus der im vorangegangenen Kapitel vorgestellten MC resultiert mehr Produktvariabilität und damit mehr Produktkomplexität. Die Produktkonfiguration (Konfiguration) ist ein Werkzeug zur Beherrschung dieser Komplexität. Sie unterstützt das Finden einer Produktvariante, die auf Kundenanforderungen angepasst und gleichzeitig machbar ist (Lutz, 2011). Diese wird von einem Produktkonfigurationssystem (Konfigurator) durchgeführt. Es folgt nun eine Definition der in diesem Zusammenhang wichtigen Begriffe.

**Begriffsüberblick**
**Konfiguration** ist eine spezielle Designaktivität, bei der der zu konfigurierende Gegenstand aus Instanzen einer festen Menge wohldefinierter Komponententypen zusammengesetzt wird, welche entsprechend einer Menge von Konfigurationsregeln kombiniert werden können (Sabin und Weigel, 1998).

Die Einordnung als Designaktivität erlaubt außerdem die Beschreibung der Konfiguration als ein Designtyp. Es werden das „Routine Design“, „Innovative Design“ und „Creative Design“ unterschieden. Die Konfiguration entspricht dem „Routine Design“. Dabei handelt es sich um ein Problem, bei der die Spezifikation der Objekte, deren Eigenschaften sowie kompositionelle Struktur gegeben sind und die Lösung auf Basis einer bekannten Strategie gefunden wird (Brown und Chandrasekaran, 1989). Damit

ist „Routine Design“ die simpelste der drei Formen. Die anderen Designtypen enthalten hingegen Objekte und Objektbeziehungen, die erst während des Designprozesses entwickelt werden.,

Die Schlüsselbegriffe der Definition von Sabin und Weigel sind Komponententypen und Konfigurationsregeln. **Komponententypen** sind Kombinationselemente, welche durch Attribute charakterisiert werden und eine Menge alternativer (konkreter) Komponenten repräsentieren. Übertragen auf die objektorientierte Programmierung verhalten sich Komponententypen zu Komponenten wie Klassen zu Instanzen. Komponententypen stehen zueinander in Beziehung. Diese kann entweder eine „Teil-Ganzes“-Beziehung oder Generalisierung sein (Felfernig u. a., 2014).

**Konfigurationsregeln** (Constraints) im engeren Sinne sind Kombinationsrestriktionen (Felfernig u. a., 2014). Weitere Arten werden später vorgestellt.

Zur besseren Nachvollziehbarkeit den weiteren Terminologie ist eine Definition des Variantenbegriffs angebracht. DIN 199 beschreibt Varianten als „Gegenstände ähnlicher Form und/oder Funktion mit einem in der Regel hohen Anteil identischer Gruppen oder Teile“. Damit ist also eine Gegenstandsmenge gemeint. Ein Element dieser Menge ist demzufolge eine konkrete Variantenausprägung. Eine Variantenausprägung unterscheidet sich von einer anderen durch mindestens eine Beziehung oder ein Element (Lutz, 2011).

Die Einheit aus Komponententypen sowie das Wissen um deren Kombinierbarkeit in Form von Constraints wird als **Konfigurationsmodell** bezeichnet. Es bildet die Menge der validen Lösungen und definiert so implizit alle Variantenausprägungen eines Produktes. Dadurch muss nicht jede Variantenausprägung explizit definiert und abgespeichert werden (z.B. in einer Datenbank). Die Anzahl möglicher Kombinationen kann in die Millionen gehen, was die Suche nach einer bestimmten sehr zeitaufwändig machen würde (Falkner u. a., 2011).

Die Einheit aus Konfigurationsmodell und den Kundenanforderungen wird als **Konfigurationsaufgabe** bezeichnet (Felfernig u. a., 2014). Diese ist die Eingabe des Konfigurators, welcher als Ausgabe eine Konfiguration und somit eine konkrete Variantenausprägung liefert. Demzufolge ist der Begriff Konfiguration überladen: er bezeichnet sowohl den Prozess als auch dessen Ergebnis. Im Folgenden werden daher die Begriffe Konfigurationsprozess sowie Konfigurationslösung verwendet.

Aus diesem Begriffsüberblick geht hervor, dass die auf dem Konfigurationsmodell basierende Konfigurationsaufgabe der Schlüssel zur Bildung einer kundenspezifischen Variantenausprägung ist. Aus diesem Grunde werden diese Begriffe im Folgenden

genauer erläutert.

**Konfigurationsmodell**

Das Konfigurationsmodell kapselt das Wissen über die Komponenten eines variierbaren Produktes, sowie deren Beziehungen und Kombinierbarkeit. Es wird auch als Wissensbasis $C_K B$ bezeichnet. Durch dieses Modell kann eine Vorstellung aller abbildbaren Variantenausprägungen gewonnen werden.

Es gibt verschiedene Formalisierungs- und Visualisierungskonzepte von Konfigurationsmodellen. Für eine vollständige Übersicht sei auf Felfernig u. a. (2014) verwiesen. Im Folgenden wird eine UML-basierte Visualisierung vorgestellt, da diese im Bereich der Informatik eine besondere Verbreitung aufweist.

**UML-Konfigurationsmodellvisualisierung**

Die Visualisierung eines Konfigurationsmodells wird in Anlehnung an Felfernig u. a. anhand einer exemplarischer PC-Konfiguration erklärt.

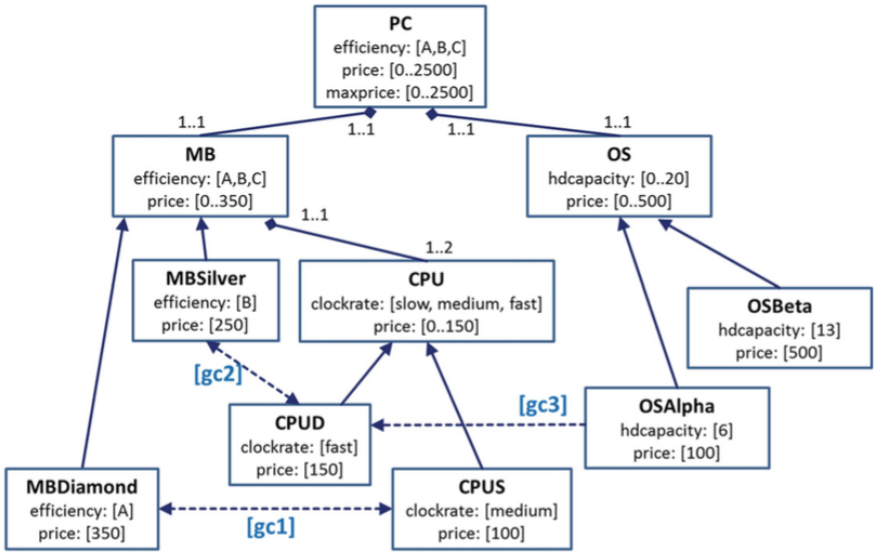



**FIGURE 6.9**

Fragment of the PC model (adapted part of Figure 6.7).

**Table 6.3** Constraints related to the configuration model in Figure 6.9.

| Name | Description |
|---|---|
| gc1 | CPUs of type CPUS are incompatible with motherboards of type MBDiamond |
| gc2 | CPUs of type CPUD are incompatible with motherboards of type MBSilver |
| gc3 | Each OS of type OSAlpha requires a CPU of type CPUD |
| prc2' | The price of one personal computer (PC) is determined by the prices of the motherboard (MB), the CPUs, and the operating system (OS) |
| resc1 | The computer price must be less or equal to the maxprice defined by the customer |

Abbildung 2: UML-Konfigurationsmodellvisualisierung einer PC-Konfiguration (Quelle: Felfernig u. a. (2014))

Abbildung 2 besteht aus einer Strukturdarstellung und einer Regeltabelle. Vor der genauen Beschreibung der Modellierungsmöglichkeiten erfolgt zur besseren Nachvollziehbarkeit eine unformale Beschreibung der Grafik:

Ein PC besteht aus einem Motherboard (MB) und Betriebssystem (OS). Ein Motherboard besitzt wiederum einen oder zwei CPUs. Ein Motherboard kann in der Ausfertigung "MBDiamondöder "MBSilver"vorliegen. Ein CPU in den Ausfertigungen "CPUDöder "CPUS", ein Betriebssystem als OSBETA oder OSALPHA. MBDiamond und CPUS dürfen nicht gleichzeitig gewählt werden. Gleiches gilt für MBSilver und CPUD. Wird OSAlpha gewählt, muss CPUD auch gewählt werden. Der Preis errechnet sich aus der Summer der Einzelkosten des verwendeten Motherboards, Betriebssystems und der CPUs. Er darf das Preislimit des Kunden nicht überschreiten.

Es stehen folgende Modellierungsmöglichkeiten des Strukturteils zur Verfügung:

- **Komponententypen:** Kann mit Klassen aus der Objektorientierten Programmierung verglichen werden. Es besitzt einen eindeutigen Namen (z.B. MB) und wird durch eine Menge Attribute beschrieben (z.B. efficiency und price). Ein Attribute hat einen Datentyp, welcher eine Konstante, ein Wertebereich oder eine Enumeration (z.B. efficiency kann die Werte 'A', 'B' oder 'C' annehmen) sein kann.
- **Assoziationen mit Kardinalität:** Aus UML sind die Associationstypen Aggregation und Komposition bekannt. Im Rahmen der Konfigurationsmodellbeschreibung geschieht eine Beschränkung auf die Komposition. Auf diese Art und Weise steht jede Komponente in einer „Teil-Ganzes“ Beziehung zu genau einer anderen Komponente. Kardinalitäten beschreiben Assoziationen noch näher, indem sie sie durch Mengeninformationen ergänzen (ein Motherboard hat ein bis zwei CPUs, welche zu genau einem Motherboard gehören).
- **Generalisierung**: Stellt die Verbindungen zwischen einem spezialisierten Subtyp (z.B. OSAlpha) zu einem einem generalisierten Supertyp (z.B. OS) her. Sie ist disjunkt und vollständig. Disjunkt bedeutet, dass ein Supertyp genau einem seiner Subtypen zugewiesen werden kann. Vollständig bedeutet, dass die Subtypen alle Instanzen darstellen, die ein Supertyp annehmen kann.

Die Darstellung wird ergänzt durch Konfigurationsregeln. Sie gelten zwischen Komponententypen und/oder deren Attribute. Wenn möglich, werden sie direkt im Diagramm dargestellt werden. Anderenfalls werden sie in einer Tabelle aufgelistet. Man unterscheidet unterschiedliche Arten von Konfigurationsregeln Felfernig u. a.:

- **Grafische Konfigurationsregeln** $GC$**:** Können im Gegensatz zu anderen Regeln direkt im UML-Diagramm dargestellt werden. Ansonsten entsprechen sie einem der folgenden Typen.
- **Preisbildungs-Konfigurationsregeln** $PRC$**:** Nehmen eine Sonderstellung ein, da sie keinen direkten Einfluss auf die Kombinierbarkeit haben. Stattdessen kann aus der Auswertung dieser Regel eine Preisinformation gewonnen werden. Bei tatsächlichen Konfigurationsanwendungen sind Preisregeln jedoch meistens nicht Teil des Konfigurationsmodells. Der Konfigurator hat dann einen eigenen Mechanismus zur Preisbildung.
- **Ressourcen-Konfigurationsregeln** $RESC$**:** Beschränken die Produktion und den Verbrauch bestimmter Ressourcen.
- **Abhängigkeits-Konfigurationsregeln** $CRC$**:** Beschreiben, unter welchen Voraussetzungen zusätzliche Komponenten Teil der Konfiguration sein müssen.
- **Kompatibilitäts-Konfigurationsregeln** $COMPC$**:** Beschreiben die Kompatibilität oder Inkompatibilität bestimmter Komponenten. In der Regeln finden sich in einer Regeltabelle entweder erstere oder zweitere Art, je nachdem, was den selteneren Fall darstellt.

Die Menge aller Konfigurationsregeln wird auch als Wissensbasis $C_{KB}$ beschrieben. Es gilt:

$$C_{KB} = GC \cup PRC \cup RESC \cup CRC \cup COMPC$$

**Konfigurationsaufgabe**

Frayman und Mittal (1989) definieren einen Konfigurationsaufgabe wie folgt:

> (A) a fixed, pre-defined set of components, where a component is described by a set of properties, ports for connecting it to other components, constraints at each port that describe the components that can be connected at that port, and other structural constraints; (B) some description of the desired configuration; and (C) possibly some criteria for making optimal selections.

(A) entspricht den Informationen, die aus dem oben vorgestellte Konfigurationsmodell resultieren. (B) und (C) sind als Kundenanforderungen zusammenfassbar. Informell wird daher die Aussage getroffen, dass eine Konfigurationsaufgabe aus den Konfigurationsmodell sowie den Kundenanforderungen besteht (Felfernig u. a., 2014).

Auf formeller Ebene ist eine Konfigurationsaufgabe auch auf Grundlage eines Constraint Satisfaction Problem (CSP) als „Regelbasierte Wissensrepräsentation“ beschreibbar. Eine Diskussion dieser Variante ist sinnvoll, da so die Konfigurations-

lösung definiert werden kann. Es handelt sich dabei um ein Problem, bei dem für eine gegebene Menge Variablen und deren Wertebereiche unter Berücksichtigung einer Regelmenge versucht wird, eine zulässige Kombination von Werten aus den Wertebereichen der Variablen zu ermitteln. Bei einer Konfigurationsaufgabe wird die Regelmenge um die Menge der Kundenanforderungen erweitert (Felfernig u. a., 2014).

Eine Konfigurationsaufgabe ist demzufolge ein Tripel $(V, D, C)$, wobei $V = \{v_1, ..., v_n\}$ eine endliche Menge Variablen , $D = \{dom(v_1), ..., dom(v_n)\}$ die Menge der Werte der Variablen und $C = C_{KB} \cup REQ$, wobei $C_{KB}$ die oben beschriebene Wissensbasis und $REQ$ die Menge der Kundenanforderungen ist (Felfernig u. a., 2014).

**Konfigurationslösung**

Auf Grundlage des CSP kann eine Konfigurationslösung formell definiert werden. Es handelt sich dabei um eine Insttanziierung $I = \{v_1 = i_1, ..., v_n = i_n\}$, wobei $i_j$ ein Element aus $dom(v_1)$ ist. I ist vollständig (jede Variable bestitzt einen zugewiesenen Wert) und konsistent (erfüllt alle Konfigurationsregeln) (Falkner u. a., 2011). Eine solche Lösung wird im folgenden als valide bezeichnet.

Eine Lösung kann durch ein UML-Instanz-Diagramm visualisiert werden, wie Abbildung 3 darstellt. Anstatt von Komponententypen sind nur noch konkrete Instanzen enthalten.

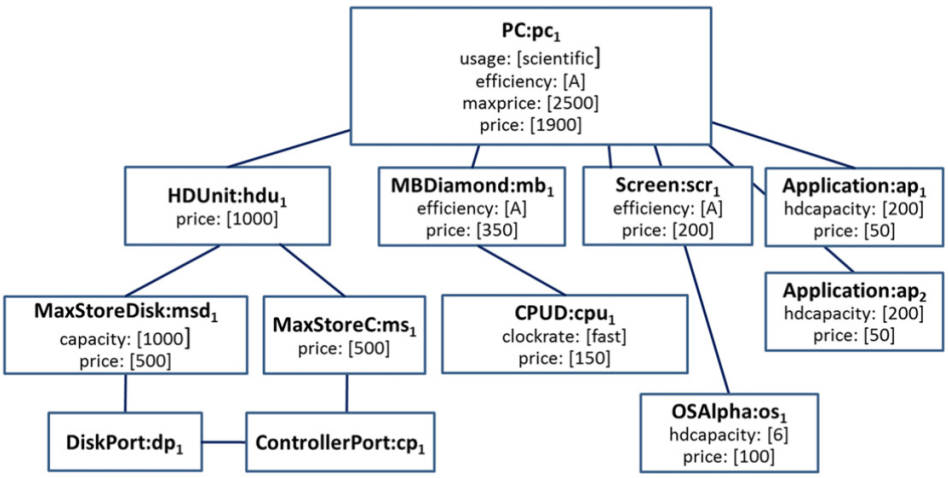



Abbildung 3: Visualisierung eines Konfigurationsergebnisses als UML-Instanz-Diagramm (Quelle: Felfernig u. a. (2014))

Eine Konfigurationsaufgabe kann mehr als eine valide Lösung liefern. In diesem Falle ist es Möglich, dem Benutzer als Ergebnis eine Variantenmenge zu präsentieren. Nicht

erfüllbare Constraints führen hingegen zu einer leeren Lösungsmenge.

Diese Beschreibung einer Konfigurationslösung suggeriert, dass zu Beginn des Konfigurationsprozesses einmalig die Kundenanforderungen aufgenommen und daraufhin die Lösungsmenge ermittelt wird. Diese Form ohne interaktive Eingriffsmöglichkeiten wird als statisch bezeichnet. Demgegenüber erlaubt die interaktive Konfiguration einen schrittweisen Entscheidungsprozess (**?**). Dieser Prozess muss durch eine Software unterstützt werden. Sie wird als Konfigurator bezeichnet und wird im Folgenden vorgestellt.

### 2.1.4 Konfigurator

Der Konfigurator ist das System, welche die Schnittstelle zum Benutzer darstellt und den Konfigurationsprozess durchführt. Es bekommt die Konfigurationsaufgabe als Eingabe und liefert als Ausgabe die Konfigurationslösung (Felfernig u. a., 2014). Sie "[...] führen den Abnehmer durch alle Abstimmungsprozesse, die zur Definition des individuellen Produktes nötig sind und prüfen sogleich die Konsistenz sowie Fertigungsfähigkeit der gewünschten Variante"(Piller, 2006).

Nach Piller (2006) besitzt ein Konfigurator typischerweise drei Komponenten:

- **Konfigurationskomponente:** Führt den Konfigurationsprozess durch. Lutz (2011) bemerkt, dass hierzu eine explizite Wissensbasis (das Konfigurationsmodell, Anmerkung des Authors) und eine Problemlösungskomponente zur Lösungsfindung gehört.
- **Präsentationskomponente:** Erstellte eine Konfiguration in Zielgruppenspezifischer Form. Der Author bemerkt, dass sie gleichzeitig als Schnittstelle zur Aufnahme der Kundenanforderungen dienst.
- **Auswertungskomponente:** Präsentation der Konfiguration in einer Form, welche eine Interpretation der letztendlichen Variantenausprägung außerhalb des Konfigurators erlaubt. Dies können zum Beispiel Stücklisten, Konstruktionszeichnungen oder Arbeitspläne sein.

Konfiguratoren für die Erhebung komplexer Anforderungen technischer Systeme unterschieden müssen von Konfiguration für den Einsatz im MC werden (Felfernig u. a., 2014). Erstere sind für den Experteneinsatz gedacht oder dienen nach Piller (2006) als Vertriebskonfiguratoren der Unterstützung des Verkaufsgespräches. Letztere werden von Kunden in einer Company-to–Customer Beziehung genutzt und werden auch als Mass Customization Toolkit bezeichnet. Die sogenannte Selbstkonfiguration ist wesentlich sie die Verheiratung von kundenindividueller Fertigung und Massenpro-

duktion, indem der zeitkonsumierende Prozess der Erhebung der Kundenbedürfnisse auch die Seite des Kunden verlagert wird(Piller, 2006)

Konfiguratoren können bei allen in Abschnit 2.1.2 genannten Produktionskonzepten zum Einsatz kommen. Je nach Produktionskonzept erfüllen sie für den Anwender eine unterschiedliche Funktion. Bei PTO erfüllt der Konfigurator eine Katalogfunktion, indem er den Anwender er den Anwender bei der Auswahl eines fertigen Produktes aus einer Produktpalette unterstützt. Bei ATO verhält sich der Konfigurator wie eine Variantengenerator, der den Anwender bei der Auswahl der richtigen, vom Hersteller vordefinierten Variante unterstützt. Der Anwendungsfall MTO ist ähnlich, jedoch sind die Varianten vom Anwender definiert. Bei ETO können Konfiguratoren durch den erheblichen Neukonstruktionsbedarf nur begrenzten Beitrag leisten. Aus dieser Erläuterung lässt sich Ableiten, dass das Haupteinsatzgebiet von Konfiguratoren im ATO/MTO Umfeld liegt.

## 2.2 Webservices

Das W3C (2004) definiert Web-Services lose als:

> „[...] a software system designed to support interoperable machine-to-machine interaction over a network“

Die Definition hebt hervor, dass die Kommunikation zwischen heterogenen Systemen stattfinden kann. „Zwischen Systemen“ differenziert gleichzeitig klar zur klassischen Verwendung eines Programms, bei der ein (menschlicher) Nutzer mit einem System kommuniziert. Tilkov (2011) bemerkt, dass Web Service damit sehr weich definiert ist; „nämlich eigentlich gar nicht“. Fest steht, dass hier ein Service einen gewissen Dienst anbietet, die von einem Clienten über Webtechnologien angesprochen werden kann. Webservices sind also eine Möglichkeit zur Realisierung von Integrationsszenarien.

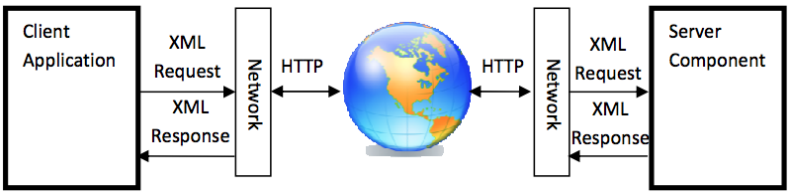



Abbildung 4: Generische Client-Server Kommunikation bei Web Services

Abbildung 4 entspricht im wesentlichen der klassischen Client-Server Kommunikation im Web. Exemplarisch werden XML-Daten übertragen, was die zugrunde liegende

Idee der Web Services illustriert: die Übertragung anderer Daten als Webseiten mittels HTTP.

Wilde und Pautasso (2011) reden von zwei "Geschmäkern"(flavors) in der Web Service Welt: SOAP und REST. Die erste Geschmacksrichtung bedeutet Web Services „auf Basis von SOAP, WSDL und den WS-*Standards - bzw. mit deren Architektur“ (Tilkov, 2011). Hier wird also ein Technologiestack beschrieben. REST hingegen ist ein Archikturstil, der 2000 in der Dissertation von Fielding vorgestellt wurde. Der Versuch, beide Varianten direkt gegenüberstellen stellen zu wollen, ist ein „[...] klassischer Apfel-Birnenvergleich: ein konkretes XML-Format gegen einen abstrakten Architekturstil“ (Tilkov, 2011).

Vor einer detailierteren Diskussion von SOAP und REST wird zur Einordnung eine Grundlegende Unterscheidung der Ansätze vorgestellt. Gemeinsam ist beiden, dass HTTP als Transportprotkoll zur Übertragung der Frage (Request) verwendet wird, die vom Server (Response) beantwortert werden soll. HTTP wiederum besteht aus einem Header und einem Entity-Body, in welchem ein Datenformat übertragen werden kann. Richardson und Ruby (2007) haben hierbei zwei Leitfragen herausgearbeitet, die von den jeweiligen Ansätzen unterschiedlich beantwortet werden: wo in diesem Paket sagt der Client dem Service, mit welchen Daten (Fokusinformation) was (Methodeninformation) gemacht werden soll?

Die Fokusinformation besagt, für welche Datenelemente sich der Client interessiert (z.B. ein Artikel eines Onlineshops). Bei REST ist dies der URI zu entnehmen (z.B. http://onlineshop.com/artikel/ski). Bei SOAP steht diese Information in einer XML-Datei, welche im Entity-Body liegt. Die Methodeninformation besagt, was mit dem identifizierten Datenelement geschehen soll (z.B. einen neuen Artikel anlegen). Bei REST steht dies im Methodenfeld der HTTP-Headers, bei SOAP wieder um Entity-Body. Als Ausgangsbasis ist zu bemerken: SOAP verwendet HTTP nur als Transportprotokoll, REST auch dessen Ausdruckskraft (Wilde und Pautasso, 2011).

### 2.2.1 SOAP

Bei SOAP-Web Services wird ein Remote Procedure Call (RPC) durchgeführt. Es handelt sich hierbei um eine generelle Technik zur Realisierung von Systemverteilung, bei der ein System die Funktion eines Systems aus einem anderen Adressraum aufruft. SOAP ist ein XML-basiertes Umschlagsformat, welches wiederum die Beschreibung eines Methodenaufrufs in XML Form enthält. Bei SOAP-Web Services werden also im wesentlichen RPCs über HTTP getunnelt (Wilde und Pautasso, 2011). Der SOAP-Umschlag ist Transportunabhängig, könnte also auch von anderen Protokollen als

HTTP übertragen werden (Tilkov, 2011)

Wie die Beschreibung des Methodenaufrufs aussehen muss, definiert die Web Service Description Language (WSDL). Jeder SOAP basierte Service stellt eine machinenverarbeitbare WSDL-Datei bereit. Darin werden die aufrufbaren Methoden, deren Argumente und Rückgabetypen beschrieben sowie die Schemata der XML-Dokumente, die der Service akzeptiert und versendet (Richardson und Ruby, 2007).

Es existieren eine Vielzahl von middleware Interoperabilitätsstandards, die mit dem „WS-“ Prefix versehen sind. Diese sind XML-Aufkleber für den SOAP-Umschlag, die HTTP-Headern entsprechen (Richardson und Ruby, 2007). Sie erweitern die Ausdrucksmöglichkeit des SOAP-Formats (Wilde und Pautasso, 2011). Beispielsweise erlaubt WS-Security die Berücksichtigung von Sicherheitsaspekten bei der Client-Server Kommunikation. Eine Übersicht der existierenden Standards ist dem Wiki für Webservices WsWiki zu entnehmen.

### 2.2.2 REST

> „Eine Architektur zu definieren bedeutet zu entscheiden, welche Eigenschaften das System haben soll, und eine Reihe von Einschränkungen vorzugeben, mit denen diese Eigenschaften erreicht werden können.“ (Tilkov, 2011)

Dies ist in der Dissertation von Fielding geschehen, in derer REST als Architekturstil definiert wird. Ein Architekturstil ist ein stärkerer Abstraktionsgrad als eine Architektur. Beispielsweise ist das Web eine HTTP-Implementierung von REST (Tilkov, 2011). Tatsächlich wurden die Einschränkungen von REST aber dem Web entnommen, indem Fielding es post-hoc als lose gekoppeltes, dezentralisiertes Hypermediasystem konzeptualisiert (Wilde und Pautasso, 2011) und dann von diesem Konzept abstrahiert hat. Einen Webservice nach dem REST-Architekturstil zu implementieren, bedeutet, sie dem Wesen des Webs anzupassen und damit seine stärken zu nutzen (Tilkov, 2011).

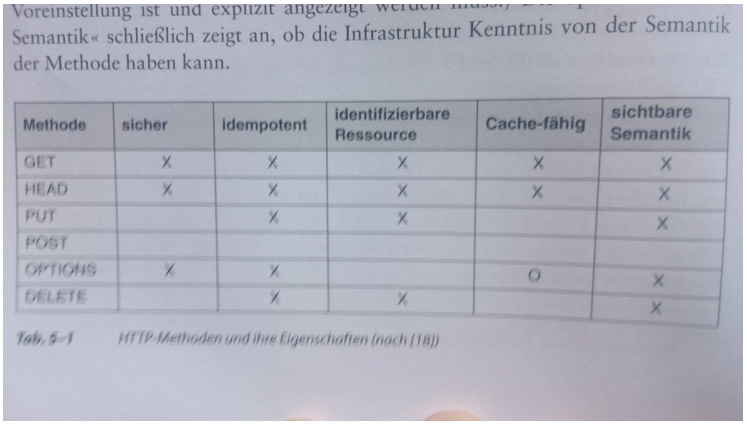
Entsprechend Tilkovs Architekturdefinition werden im Folgenden die Einschränkungen von REST sowie die daraus resultierenden Eigenschaften besprochen.

**Einschränkungen**

Einschränkungen können auch als Kriterien für die Implementierung betrachtet werden. Während Fielding explizit vier Kriterien nennt, basiert die folgende Auflistung auf der praxiserprobten Variante der Sekundärliteratur (Wilde und Pautasso, 2011; Tilkov, 2011).

- **Ressourcen mit eindeutiger Identifikation**: „Eine Ressource ist alles, was wichtig genug ist, um als eigenständiges Etwas referenziert zu werden“ (Richardson und Ruby, 2007). Identifiziert werden sie im Web durch URIs, die einen globalen Namensraum darstellen. Es ist hervorzuheben, dass eine Ressource nicht das gleiche ist wie die Datenelemente aus der Persistenzschicht einer Anwendung. Sie befinden sich auf einem anderen Abstraktionsniveau. Beispiel: eine Warenkorbressource kann ein Aggregat von Artikelelementen sein, welche allerdings nicht einzeln als Ressource zur Verfügung gestellt werden. Tilkov nimmt in diesem Zusammenhang eine Typisierung von Ressourcen vor. Im Rahmen dieser Arbeit sind folgende Ressourcentypen interessant:
  a. Bei einer Projektion wird die **Informationsmenge** verringert, indem eine sinnvolle Untermenge von Attributen eines Objektes gebildet wird. Zweck ist die zur Reduktion der Datenmenge. Beispiel: Weglassen der Beschreibungstexte von Warenkorbartikeln.
  b. Die **Aggregation** ist das Gegenteil. Hier werden Attribute unterschiedlicher Ressourcen zur Reduktion der Anzahl notwendiger Client/Server Interaktionen zusammengefasst. Beispiel: Hinzufügen der Versandkosten beim Abruf der Warenkorbartikel.
  c. **Aktivitäten** sind Ressourcen, die aus Prozessen ergeben, wie etwa ein Schritt innerhalb einer Verarbeitung. Beispiel: Ein Schritt einer nicht abgeschlossenen Konfiguration.
- **Hypermedia**: Beschreibt das Prinzip verknüpfter Ressourcen. Der Service eröffnet so die Möglichkeiten, neue Ressourcen zu entdecken oder bestimmte Prozesse anzustoßen. Beispiel: Zur einer Bestellbestätigungsressource wird ein Stornierungslink für genau dieser Bestellung hinzugefügt.
- **Standardmethoden/uniforme Schnittstelle**: Oben wurde beschreiben, dass jede Ressource durch (mindestens) eine ID identifiziert wird. Jede URI unterstützt dabei den gleichen Satz an Methoden, welche mit den HTTP-Methoden korrespondieren. Übertragen auf die objektorientierte Programmierung bedeutet dies, dass jedes Objekt das gleiche Interface implementiert. Folgende Methoden werden typischerweise unterstützt:
  a. **GET:** abholen einer Ressource
  b. **PUT:** anlegen einer Ressource unter dieser URI oder deren Aktualisierung, falls unter dieser URI bereist eine Ressource existiert.
  c. **POST:** bedeutet im engeren Sinne das Anlegen einer Ressource unter einer URI, die vom Service bestimmt. Im weiteren Sinne kann durch Post ein Prozess angestoßen werden.

d. **Delete:** löschen einer Ressource



Voreinstellung ist und explizit angezeigt werden

Semantik« schließlich zeigt an, ob die Infrastruktur Kenntnis von der Semantik der Methode haben kann.

| Methode | sicher | idempotent | identifizierbare Ressource | Cache-fähig | sichtbare Semantik |
|---|---|---|---|---|---|
| GET | X | X | X | X | X |
| HEAD | X | X | X | X | X |
| PUT | | X | X | | X |
| POST | | | | | |
| OPTIONS | X | X | | O | X |
| DELETE | | X | X | | X |

Tab. 5–1 HTTP-Methoden und ihre Eigenschaften (nach [18])

Abbildung 5: HTTP-Methoden und ihre Eigenschaften [a] (Quelle: Tilkov (2011))

[a] Relevante Attribute im Rahmen der Arbeit: „sicher“ bedeutet Nebenwirkungsfrei, d.h. kein Ressourcenzustand ändert sich durch diese Methode. „Idempotent“ bedeutet, dass das Resultat der Methode bei Mehrfachausführung das gleiche ist. „Identifizierbare Ressource“ bedeutet, dass unter der angegebenen URL in jedem Falle eine Ressource zu erwarten ist.

Diese Beschreibung sowie die Eigenschaften aus Abbildung 5 entsprechen der HTTP-Spezifikation. Die Implementierung einer Methode muss dem erwarteten Verhalten aus dieser Spezifikation entsprechen. Die Praxis zeigt, dass nur die Methoden unterstützt werden, die für die jeweilige Ressource sinnvoll sind. Abbildung 5 macht außerdem klar, dass es für POST keinerlei Garantien gibt. Da nicht eindeutig ist, ob über POST eine Ressource erstellt oder ein Prozess angestoßen wird, sehen Richardson und Ruby und sieht hierin eine Verletzung der uniformen Schnittstelle. Dies bedeutet in der Praxis: was bei einem Post passiert, ist nicht der HTTP-Spezifikation, sondern der Api-Beschreibung des Webservice zu entnehmen.

- **Ressourcen und Repräsentationen:** Beschreibt die Darstellungen einer Ressource in einem definierten Format. Der Client bekommt nie die Ressource selbst, sondern nur eine Repräsentation derer zu sehen. Die Praxis zeigt, dass häufig eine einfache serialisierte Variante eines Objektes als JSON zur Verfügung gestellt wird. Beispielsweise könnte eine Bestellbestätigung auch als zugänglich gemacht werden.
- **statuslose Kommunikation:** Bedeutet die Nichtexistenz eines serverseitig abgelegten, transienten, clientspezifischen Status über die Dauer eines Requests hinweg. Der Service benötigt also nie Contextinformationen zur Bearbeitung eines Requests. Beispiel: Ein Warenkorb wird nicht in einem Sessionobjekt, sondern als persistentes Datenelement gehalten.

Diese Auflistung legt folgende die Frage nahe: ist ein Webservice, der nicht jedes Kriterium unterstützt, nicht mehr REST-konform? Was ist mit einem Webservice, der allen Einschränkungen gerecht wird, aber jedoch Ressourcen nur als JSON ausliefert - ein in der Praxis häufig anzutreffender Fall und dennoch ein Verstoß gegen die Forderung nach unterschiedlichen Repräsentationen. Aus diesem Grund existiert das „Richardson Maturity Model“, welches eine abgestufte Bewertung eines Webservices dessen REST-konformität erlaubt. Es wird im Auswertungsteil vorgestellt und zur Evaluierung der Implementierung genutzt.

**Eigenschaften**
Aus den vorgestellten Kriterien resultieren folgende Eigenschaften, welche die beabsichtigten Vorteile dieses Architekturstils darstellen (Tilkov, 2011). Dies Vorteile meint Vorteile im Vergleich zu SOAP-basierten Webservices (Richardson und Ruby, 2007):

- **Lose Kopplung:** Beschreibt isolierte Systeme mit größtmöglicher Unabhängigkeit, die über Schnittstellen miteinander kommunizieren. Wird vor allem durch die Standardmethoden ermöglicht.
- **Interoperabilität:** Beschreibt die Möglichkeit der Kommunikation von Systemen unabhängig von deren technischen Implementierung. Dieser ergibt sich durch die Festlegung auf Standards, welche bei REST-konformen Webservices die Webstandards (z.B. HTTP, URIs) sind
- **Wiederverwendung**: Jeder Client, der die Schnittstelle eines REST-basierten Service verwenden kann, kann auch jeden anderen beliebigen REST-basierten Service nutzen - vorausgesetzt, das Datenformat wird von beiden Seiten verstanden.
- **Performance und Skalierbarkeit**: Befähigt Services zum schnellen antworten, unabhängig von der Anzahl von Anfragen in einem definierten Zeitraum. Dies wird durch Cachebarkeit und Zustandslosigkeit erreicht. Cachebare Antworten ermöglichen die Wiederverwendung von Prozessergebnissen. Da der Service keinen clientspezifischen Kontext aufbauen muss, müssen aufeinanderfolgende Requests nicht vom gleichen System beantwortet werden.

## 2.3 E-Commerce

# 3 Analyse

## 3.1 Tacton Produktkonfigurator

## 3.2 TCsite

## 3.3 Produktkonfiguration in TCsite

## 3.4 Shopsystem

# 4 Anforderungen

## 4.1 Funktionale Anforderungen

## 4.2 Nichtfunktionale Anforderungen

# 5 Integrationskonzept

# 6 Integrationsumsetzung

# 7 Fazit

# 8 Vorlagen

Dieses Kapitel enthält Beispiele zum Einfügen von Abbildungen, Tabellen, etc.

## 8.1 Bilder

Zum Einfügen eines Bildes, siehe Abbildung 6, wird die *minipage*-Umgebung genutzt, da die Bilder so gut positioniert werden können.

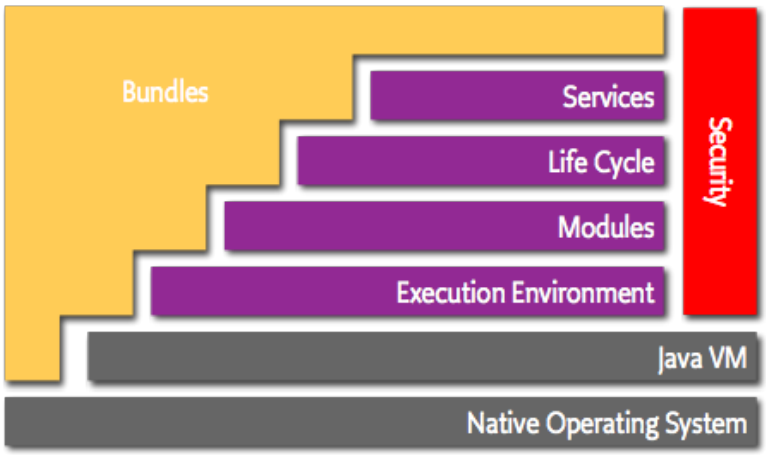



Abbildung 6: OSGi Architektur[1]

## 8.2 Tabellen

In diesem Abschnitt wird eine Tabelle (siehe Tabelle 1) dargestellt.

| Name | Name | Name |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

Tabelle 1: Beispieltabelle

[1] Quelle: `http://www.osgi.org/Technology/WhatIsOSGi`

## 8.3 Auflistung

Für Auflistungen wird die *compactitem*-Umgebung genutzt, wodurch der Zeilenabstand zwischen den Punkten verringert wird.

- Nur
- ein
- Beispiel.

## 8.4 Listings

Zuletzt ein Beispiel für ein Listing, in dem Quellcode eingebunden werden kann, siehe Listing 1.

```
1  int ledPin = 13;
2  void setup() {
3      pinMode(ledPin, OUTPUT);
4  }
5  void loop() {
6      digitalWrite(ledPin, HIGH);
7      delay(500);
8      digitalWrite(ledPin, LOW);
9      delay(500);
10 }
```

Listing 1: Arduino Beispielprogramm

## 8.5 Tipps

Die Quellen befinden sich in der Datei *bibo.bib*. Ein Buch- und eine Online-Quelle sind beispielhaft eingefügt.

Abkürzungen lassen sich natürlich auch nutzen. Weiter oben im Latex-Code findet sich das Verzeichnis.

# 9 Quellenverzeichnis

[Brown und Chandrasekaran 1989] Brown, David C. ; Chandrasekaran, B.: *Design Problem Solving: Knowledge Structures and Control Strategies. Research Notes in Artificial Intelligence.* London : Pitman, 1989

[Falkner u. a. 2011] Falkner, Andreas ; Felfernig, Alexander ; Haag, Albert: Recommendation Technologies for Configurable Products. In: *AI Magazine* 32 (2011), Nr. 3, S. 99–108

[Felfernig u. a. 2014] Felfernig, Alexander ; Hotz, Lothar ; Bagley, Claire ; Tiihonen, Juha: *Knowledge-Based Configuration: From Research to Business Cases.* Elsevier, 2014

[Fielding 2000] Fielding, Roy T.: *Architectural Styles and the Design of Network-based Software Architectures*, University of California, Irvine, Dissertation, 2000

[Frayman und Mittal 1989] Frayman, Felix ; Mittal, Sanjay: Towards a generic model of configuration tasks. In: *International Joint Conference on Artificial Intelligence (IJCAI-89)*, 1989

[Lutz 2011] Lutz, Christoph: *Rechnergestuetztes Konfigurieren und Auslegen individualisierter Produkte*, Technischen Universitaet Wien, Dissertation, 2011

[Piller 1998] Piller, Frank T.: *Kundenindividuelle Massenproduktion: die Wettbewerbsstrategie der Zukunft.* Carl Hanser Verlag München Wien, 1998

[Piller 2006] Piller, Frank T.: *Mass Customization. Ein wettbewerbsstrategisches Konzept im Informationszeitalter.* Bd. 4., überarbeitete und erweiterte Auflage. Wiesbaden : Deutscher Universitäts-Verlag | GWV Fachverlage GmbH, 2006

[Porter 1980] Porter, Michael: *Competetive Strategy: Techniques for Analyzing Industries and Competitors.* New York, 1980

[Porter 2002] Porter, Michael: *Wettbewerbsstrategie: Methoden zur Analyse von Branchen und Konkurrenten.* 11. Frankfurt, New York, 2002

[Rapp 1999] Rapp, Thomas: *Produktstrukturierung*, Universität St. Gallen, Dissertation, 1999

[Richardson und Ruby 2007] Richardson, Leonard ; Ruby, Sam: *Web Services mit REST.* Köln : O'Reilly Verlag, 2007

[Sabin und Weigel 1998] Sabin, Daniel ; Weigel, Rainer: Product Configuration Frameworks-A Survey. In: *IEEE Intelligent Systems* 13 (1998), Nr. 4, S. 42–49

[Schuh 1988] Schuh, Günther: *Gestaltung und Bewertung von Produktvarianten: ein Beitrag zur systematischen Planung von Serienprodukten*, RWTH Aachen, Dissertation, 1988

[Schuh 2005] Schuh, Günther: *Produktkomplexität managen: Strategien - Methoden - Tools*. Carl Hanser Verlag München Wien, 2005

[Schuh 2006] Schuh, Günther: *Produktionsplanung und -steuerung. Grundlagen, Gestaltung und Konzepte*. 3. Berlin : Springer, 2006

[Tilkov 2011] Tilkov, Stefan: *REST und HTTP - Einsatz der Architektur des Web für Integrationsszenarien*. 2. Heidelberg : dpunkt.verlag, 2011

[W3C 2004] W3C: *Web Services Glossary*. 2004. – URL `http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice`. – Zugriff: 21.08.2015

[Wilde und Pautasso 2011] Wilde, Eric ; Pautasso, Cesare: *REST: From Research to Practice*. Springer Verlag, 2011

[WsWiki 2009] WsWiki: *Web Service Specifications*. 2009. – URL `https://wiki.apache.org/ws/WebServiceSpecifications`. – Zugriff: 21.08.2015

# Anhang

# A GUI

Ein toller Anhang.

## Screenshot

Unterkategorie, die nicht im Inhaltsverzeichnis auftaucht.

# Erklärung

Hiermit versichere ich, dass ich meine Abschlussarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Datum: .....

(Unterschrift)