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
B2B Business-to-Business
B2C Business-to-Customer
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 Bezugsrahmen

Im Folgenden wird ein Anwendungsrahmen für Produktkonfigurationssysteme (Konfiguratoren) geschaffen. Dazu wird zunächst die marktwirtschaftliche Situation erläutert, welche die Notwendigkeit hybrider Wettbewerbsstrategien begründet. Diese wiederum stellen zu ihrer Umsetzbarkeit Bedingungen an Produktionskonzepte, welche daraufhin vorgestellt werden.

2.1.1 Ökonomischer Bezug

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

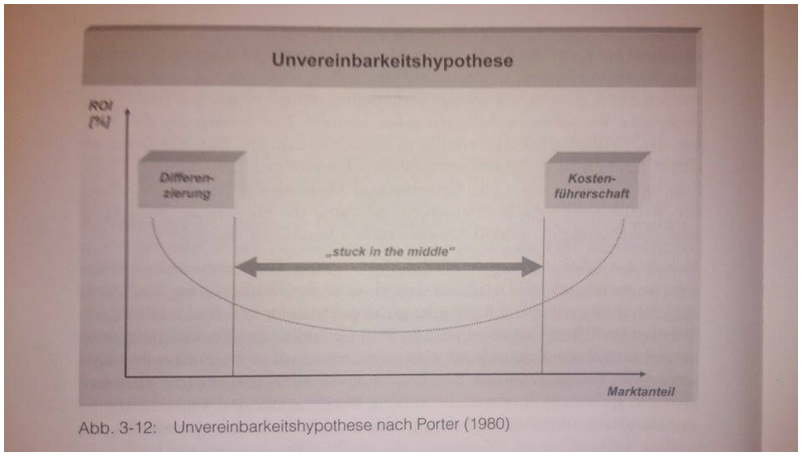


Abbildung 1: Unvereinbarkeitshypothese nach Porter (1980), zitiert von Schuh (2005)

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 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) vereint.

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. Sie werden als Produktionskonzepte bezeichnet (nach Schuh 2006, zitiert von Lutz 2011):

- **Pick-to-Order (PTO):** Herstellung ohne Kundenauftrag; Lagerhaltung auf Ebene ganzer Produkte; Keine Abhängigkeit dieser Produkte untereinander; Beispiel: Ein Standardnotebook.
- **Assemble-to-Order (ATO):** Herstellung ohne Kundeauftrag; Lagerhaltung auf Ebene der Baugruppen/-teilen; Teile mit Abhängigkeiten untereinander; Beispiel: Ein Notebook, bei welchem auf Kundenwunsch statt des CD-Laufwerks eine zusätzliche Festplatte eingebaut wird. Die Festplatte lag bereits im Lager vor.
- **Make-to-Order (MTO):** Herstellung teilweise erst nach Kundenenauftrag; Lagerhaltung auf Ebene der Baugruppen/-teilen; Produktion oder regelbasierte (parametrisierte) Konstruktion von Komponenten nach Kundenanforderung; Abhängigkeiten zwischen Teilen; keine unendliche Anzahl von Varianten; Beispiel: Kauf eines Notebooks, wobei der Kunde eine Displaygröße abweichend von den Standarddiagonallängen bestimmen kann. Display und Notebookgehäuse müssen konstruiert/hergestellt und die technischen Standardkomponenten (z.B. Festplatte, Motherboard) eingepasst werden.
- **Engineer-to-Order (ETO):** Produkt ist nicht komplett vom Hersteller vorhersehbar; Wenig bis keine Lagerhaltung auf Ebene der Baugruppen/-teilen; Entwicklung und Fertigung von Teilen nach Kundenspezifikation; unendliche

Variantenanzahl möglich;
Beispiel: Herstellung eines Notebooks mit Kaffeehalterung.

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. Ein hoher Komponentenanteil, der erst nach Auftragseingang hergestellt oder sogar konstruiert werden muss, spricht für eine starke Kundenindividualisierung (Lutz, 2011). Die unterschiedlichen Produktionskonzepte stellen jeweils einen Anwendungsbezug zu Konfiguratoren her, welche im Folgenden vorgestellt werden.

2.2 Produktkonfiguration

Aus der im vorangegangenen Kapitel vorgestellten MC resultiert mehr Produktvariabilität und damit 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).

2.2.1 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 Constraints 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 ist 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 Constraints. **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).

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

Zur besseren Nachvollziehbarkeit der 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“. Varianten sind also Gegenstandsmengen. Ein Element dieser Menge ist

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 korrekten Lösungen ab und definiert so implizit alle Varianten eines Produktes (Soininen u. a., 1998). 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). Auf dessen Grundlage kann die gewünschte Konfiguration errechnet werden. 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. Der Konfigurationsprozess, der zur Konfigurationslösung führt, wird von einem System durchgeführt. Dieses wird als **Konfigurator** bezeichnet.

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.

2.2.2 Wissenrepräsentation

Das Konfigurationswissen beschreibt Wissen, welches über ein konfigurierbares Produkt besteht (Soininen u. a., 1998). Dieses Wissen kann auf unterschiedliche Art und Weise repräsentiert, d.h. dargestellt werden. Die Repräsentation kann zur Definition eines Konfigurationsmodells genutzt werden (Felfernig u. a., 2014). Auf konzeptioneller Ebene können die Begriffe Wissensrepräsentation und Konfigurationsmodell äquivalent verwendet werden. Das Konfigurationsmodell bezeichnet jedoch letztendlich das spezifische Format, welches von einem Konfigurator verstanden wird (Soininen u. a., 1998).

Es existieren verschiedene Wissensrepräsentationskonzepte mit unterschiedlicher Ausdruckskraft. Jede hat verschiedene Stärken. Im Folgenden wird ein grafische sowie eine formelle Repräsentationsvariante vorgestellt. Ein Visualisierungskonzept erleichtert den Einstieg und ermöglicht die Bildung einer Vorstellung über die möglichen Varianten eines Produktes. Über die Formalisierung des Konfigurationswissens lässt

sich hingegen eine Definition der Konfigurationsaufgabe ableiten.

UML-Visualisierung

Von den bestehenden Visualisierungskonzept wird eine UML-basierte Variante besprochen, da diese Sprache in der Informatikdomäne eine besondere Verbreitung aufweist. Im Folgenden wird eine Notebook-Konfiguration eingeführt und von späteren Erklärungen wieder aufgegriffen. Die Modellierung basiert auf dem Arbeitsbeispiel von (Felfernig u. a., 2014).

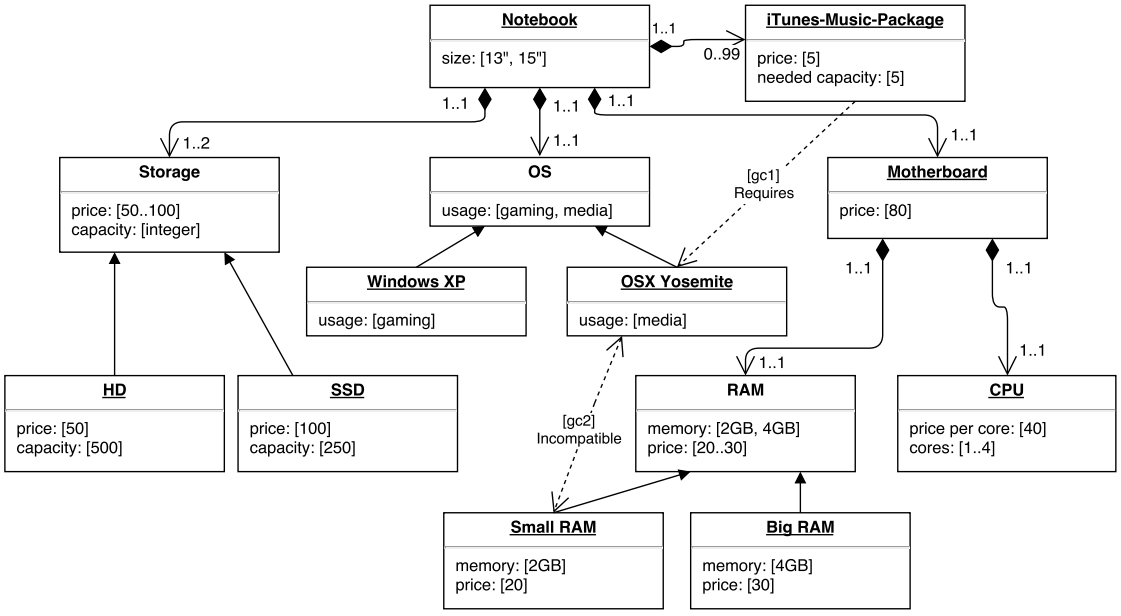


Abbildung 2: UML-Visualisierung einer Notebook-Konfiguration[a]

[a]Das 'iTunes-Music-Package' stellt ein Überraschungspaket mit Musik für iTunes dar.

Abbildung 2 beschreibt den Strukturteil der Visualisierung. Folgende Spracheinheiten sind enthalten (Felfernig u. a., 2014):

- **Komponententypen** sind die dargestellten Entitäten. Sie besitzen einen eindeutigen Namen (z.B. 'Storage') und werden durch eine Menge von Attributen beschrieben (z.B. 'price', 'capacity'). Ein Attribute hat einen Datentyp, welcher eine Konstante, ein Wertebereich (z.B. die Zahlen von 50 bis 100 als mögliche Werte des Attributs 'capacity') oder eine Enumeration (z.B. ['gaming', 'media'] als mögliche Werte des Attributes 'usage') sein kann.
- **Generalisierungen** stellen die Verbindungen zwischen einem spezialisierten Subtyp zu einem einem generalisierten Supertyp her. Damit muss der Wer-

Tabelle 1: Constraints des Konfigurationsmodells aus Abbildung 2

Name	**Beschreibung**
GC_1	Wird das **iTunes-Music-Package** gewählt, muss auch das Betriebssystem (OS) vom Typ**OSX Yosemite** gewählt werden.
GC_2	Das OS vom Typ **OSX Yosemite** und der Arbeitsspeicher (RAM) vom Typ **Small RAM** können nicht gleichzeitig gewählt werden
PRC_1	Der Preis des Notebooks ist die Summe der **price**-Attribute der Storage-, Motherboard-, RAM-, CPU- und iTunes-Music-Package-Komponenten
$RESC_1$	Die Summe der **needed capacity**-Attribute aller iTunes-Music-Package-Komponenten darf die Summe der **capacity**-Attribute aller Storage-Komponenten nicht überschreiten
CRC_1	Das OS vom Typ **OSX Yosemite** benötigt mindestens einen **core**-Wert der CPU-Komponente von 2
$COMPC_1$	Das OS vom Typ **Windows XP** ist inkompatibel mit dem **size**-Wert der Notebook-Komponente von 13"

tebereich eines Attributs eines Subtypen eine Teilmenge des entsprechenden Attributwertebereichs des Supertypen sein. Angewendet auf das Konfigurationsmodell entsteht so die Unterscheidung zwischen Komponententypen und Komponenten: ein nicht mehr weiter spezialisierter Komponententyp wird als Komponente (unterstrichen dargestellt) bezeichnet. Also sind Generalisierungen die Verbindungen zwischen Komponententypen (z.B. 'Storage') und Komponenten (z.B. 'HD'). Durch die Zuweisung eines Komponententypen zu einer Komponente entsteht eine Instanz. Diese Zuweisung ist disjunkt und vollständig. Disjunkt bedeutet, dass jede Instanz eines Komponententypen nur genau eine der Komponenten entsprechen kann. Beispiel: Eine Instanz eine 'Storage' kann eine 'HD' oder eine 'SSD' sein, aber nicht beides. Vollständig bedeutet, dass die dargestellten Komponenten alle tatsächlich möglichen Instanzen darstellen (z.B. gibt es für diese Konfiguration keine Komponente 'DVD' als möglichen Storage).

- **Assoziationen mit Kardinalitäten** beschreiben die Beziehungen zwischen Komponententypen. Die hier verwendete Variante ist die Komposition. Das bedeutet, dass keine Instanz eines Komponententypen Teil von mehr als einer anderen Instanz sein kann. Kardinalitäten beschreiben Assoziationen noch näher, indem sie sie durch Mengeninformationen ergänzen. Beispiel: Eine Notebook-Instanz besitzt ein oder zwei Storage-Instanzen. Eine Storage-Instanz kann nur Teil einer Notebook-Instanz sein.

Die Darstellung wird ergänzt durch Constraints. Sie gelten zwischen Komponenten-

typen und/oder deren Attribute. Wenn möglich, werden sie direkt im Diagramm dargestellt werden. Anderenfalls werden sie in einer Tabelle aufgelistet (siehe Tabelle 1). Es werden unterschiedliche Constrainttypen unterschieden (Felfernig u. a., 2014):

- **Grafische Constraints** GC können im Gegensatz zu anderen Constraints direkt im UML-Diagramm dargestellt werden. Ansonsten entsprechen sie einem der folgenden Typen.
- **Preisbildungs-Constraints** 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 Preisconstraints jedoch meistens nicht Teil des Konfigurationsmodells. Stattdessen wird die Preisbildung durch einen eigenen Mechanismus realisiert.
- **Ressourcen-Constraints** $RESC$ beschränken die Produktion und den Verbrauch bestimmter Ressourcen. Beispiel: Jedes 'iTunes-Music-Package' verbraucht 5(MB) Festplattenkapazität. Der verfügbare Speicher wird wiederum durch die Storage-Instanzen bestimmt. Wird nur ein Speichermedium in Form einer 'SSD' gewählt, hat das Notebook 250(MB) Festplattenkapazität. Somit die Obergrenze für 'iTunes-Music-Package' Instanzen gleich 50.
- **Abhängigkeits-Constraints** CRC beschreiben, unter welchen Voraussetzungen zusätzliche Komponenten Teil der Konfiguration sein müssen.
- **Kompatibilitäts-Constraints** $COMPC$**:** Beschreiben die Kompatibilität oder Inkompatibilität bestimmter Komponenten.

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

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

2.2.3 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) ist eine andere Definition für ein Konfigurationsmodell. (B) und (C) sind als Kundenanforderungen zusammenfassbar. Informell entsteht so die im Begriffsüberblick

vorgestellte Definition: Die Konfigurationsaufgabe besteht aus dem Konfigurationsmodell sowie den Kundenanforderungen (Felfernig u. a., 2014).

Auf formeller Ebene ist eine Konfigurationsaufgabe auch auf Grundlage eines Constraint Satisfaction Problem (CSP) beschreibbar. Eine Vorstellung dieser Variante ist sinnvoll, da so eine Definition der Konfigurationslösung abgeleitet werden kann. Das CSP ist ein Problem, bei dem für eine gegebene Menge Variablen und deren Wertebereiche unter Berücksichtigung einer Regelmenge versucht wird, eine zulässige Wertekombination 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).

2.2.4 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 Constraints) (Falkner u. a., 2011). Eine solche Lösung wird als korrekt bezeichnet (Soininen u. a., 1998).

Eine korrekte Lösung kann durch ein UML-Instanz-Diagramm visualisiert werden. Abbildung 3 zeigt eine korrekte Lösung der Notebook-Konfiguration. Dargestellt wird also eine mögliche Variantenausprägung. Anstatt der Komponententypen sind nur noch konkrete Instanzen enthalten.

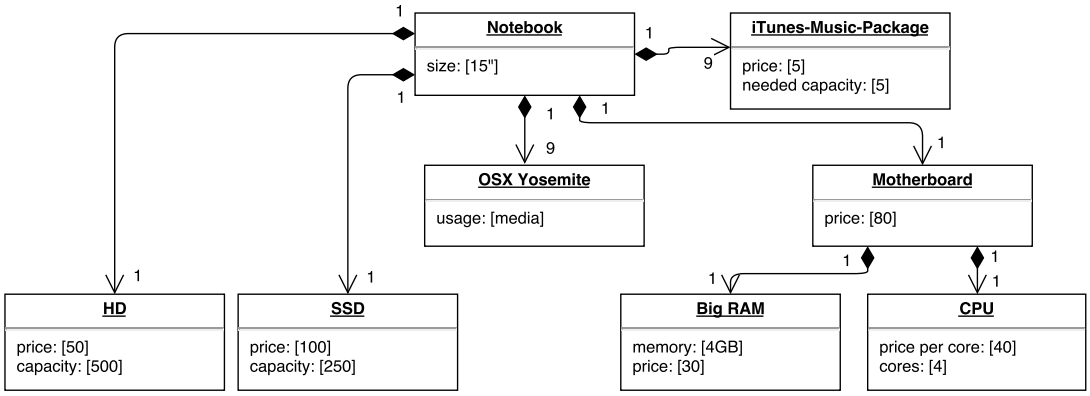


Abbildung 3: Visualisierung einer Konfigurationslösung als UML-Instanz-Diagramm

Lieferte eine Konfigurationsaufgabe mehr als eine korrekte Lösung, ist das Ergebnis eine Variantenmenge. Eine nicht erfüllbare Konfigurationsaufgabe führt 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 wird als statische Konfiguration bezeichnet. Demgegenüber erlaubt die interaktive Konfiguration das schrittweise Treffen und Revidieren von Entscheidungen (Hadzic und Andersen, 2004).

2.2.5 Konfigurationssysteme

Der Konfigurator ist das System, welche die Schnittstelle zum Benutzer darstellt und den (interaktiven) Konfigurationsprozess durchführt. Es bekommt die Konfigurationsaufgabe als Eingabe und liefert als Ausgabe die Konfigurationslösung (Felfernig u. a., 2014). Konfiguratoren "[...] 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 drei Komponenten:

- Die **Konfigurationskomponente** führt den Konfigurationsprozess durch. Sie wird auch als Konfigurationsengine bezeichnet (Tacton Systems AB, 2007).
- Die **Präsentationskomponente** erstellt eine Konfigurationsdarstellung in zielgruppenspezifischer Form. Daraus lässt sich ableiten, dass sie gleichzeitig als Schnittstelle zur Aufnahme der Kundenanforderungen dient.

- Die **Auswertungskomponente** präsentiert der Konfiguration in einer Form, welche eine Interpretation der Variantenausprägung außerhalb des Konfigurators erlaubt. Dies können zum Beispiel Stücklisten, Konstruktionszeichnungen und Arbeitspläne sein.

Konfiguratorarten

Konfiguratoren für die Erhebung komplexer Anforderungen technischer Systeme müssen von Konfiguration für den Einsatz für MC unterschieden 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. Diese sogenannte Selbstkonfiguration ist eine Vorraussetzung für MC, indem der zeitkonsumierende Prozess der Erhebung der Kundenbedürfnisse auf 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 bei der Auswahl eines fertigen Produktes aus einer Produktpalette unterstützt. Bei ATO verhält sich der Konfigurator wie ein Variantengenerator, der den Anwender bei der Auswahl der richtigen Variantenausprägung unterstützt. Wohlgemerkt: der Hersteller hat alle möglichen Variante vordefiniert, sie sind also herstellerspezifisch (Schomburg, 1980). Der Anwendungsfall MTO ist ähnlich, jedoch werden Komponenten kundenspezifisch hergestellt oder regelbasiert konstruiert. Es wird von kundenspezifischen Varianten gesprochen (Schomburg, 1980). Bei ETO besteht ein erheblicher Neukonstruktionsbedarf. Dies widerspricht der Definition der Konfiguration als Designaktivität aus Abschnitt 2.2.1 - die Spezifikation der beteiligten Objekte ist nicht vollständig bekannt. Konfiguration können hier nur einen begrenzt Aus dieser Erläuterung lässt sich Ableiten, dass das Haupteinsatzgebiet von Konfiguratoren im ATO/MTO Umfeld liegt.

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). MC resultiert in Variantenvielfalt und damit in Produktkomplexität. In Abschnitt 2.2,

durch welche Funktionsweise Konfiguratoren zur Beherrschung dieser Komplexität beitragen.

2.3 Webservices

Das W3C (2004) definiert Webservices lose als:

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

Die Definition schließt die Kommunikation heterogener Systeme ein. „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 Dienst anbietet, der von einem Clienten über Webtechnologien angesprochen werden kann. Webservices sind demzufolge eine Möglichkeit zur Realisierung von Integrationsszenarien webbasierter Systeme.

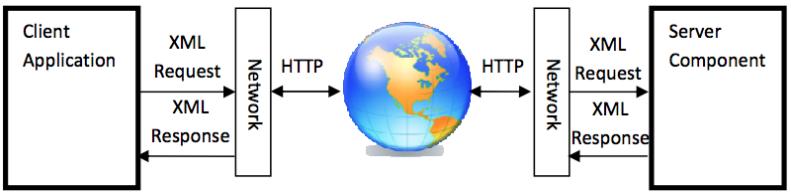


Abbildung 4: Generische Client-Server Kommunikation bei Webservices

Abbildung 4 entspricht im wesentlichen der klassischen Client-Server Kommunikation im Web. Exemplarisch werden XML-Daten übertragen, was die zugrunde liegende Idee der Webservices illustriert: die Übertragung anderer Daten als Webseiten mittels HTTP.

Wilde und Pautasso (2011) reden von zwei etablierten „Geschmäkern“ (flavors) in der Webservice Welt: SOAP und REST. Die erste Geschmacksrichtung bedeutet Web Services „auf Basis von SOAP, WSDL und den WS-*Standards - bzw. [...] deren Architektur“ (Tilkov, 2011). Hier wird also ein XML-basierter 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 detaillierteren 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 zur Übertragung von Daten. Richardson und Ruby (2007) haben 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 sagt aus, 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/pc). Bei SOAP steht diese Information in einer XML-Datei welche Entity-Body übertragen wird; die sogenannte Payload. Die Methodeninformation sagt aus, was mit dem identifizierten Datenelement geschehen soll (Bsp:: lege einen neuen PC-Artikel an). Bei REST steht dies im Methodenfeld der HTTP-Headers, bei SOAP wieder im Entity-Body. Daraus lässt sich als grundlegender Unterschied ableiten: SOAP verwendet HTTP nur als Transportprotokoll, REST auch dessen Ausdruckskraft (Wilde und Pautasso, 2011).

2.3.1 SOAP

Bei SOAP-Web Services wird ein Remote Procedure Call (RPC) durchgeführt. Dabei handelt es sich um eine generelle Technik zur Realisierung von Systemverteilung. Ein System ruft die Funktion eines Systems aus einem anderen Adressraum auf. SOAP ist ein XML-basiertes Umschlagsformat, welches wiederum die Beschreibung eines Methodenaufrufs in XML-Form enthält. Bei SOAP-Web Services werden also RPCs über HTTP getunnelt (Wilde und Pautasso, 2011). Das ist Konvention, aber keine Notwendigkeit: der SOAP-Umschlag ist Transportunabhängig, könnte also auch von anderen Protokollen als HTTP übertragen werden (Tilkov, 2011). Solange es sich bei dem Transportprotokoll um eine Webtechnologie handelt, wird die Webservicedefinition nicht verletzt.

Wie die Beschreibung des RPC 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. Außerdem werden Schemata der XML-Dokumente festgehalten, 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 (2009) zu entnehmen.

2.3.2 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)

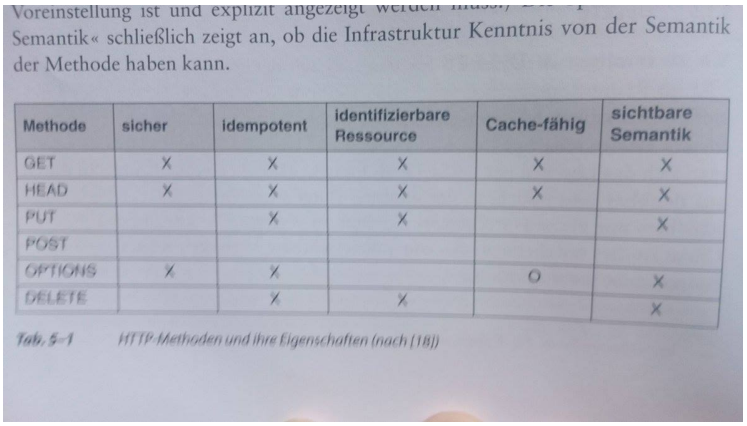
Dies ist in der Dissertation von Fielding geschehen, in der REST als Architekturstil definiert wird. Ein Architekturstil ist ein stärkerer Abstraktionsgrad als eine Architektur. Beispielsweise ist das Web eine HTTP-Implementierung von REST (Tilkov, 2011). Tatsächlich wurden die Einschränkungen von REST 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, es dem Wesen des Webs anzupassen und dessen stärken zu nutzen (Tilkov, 2011).

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

Einschränkungen
Einschränkungen sind - in eigenen Worten - Implementierungskriterien. Während Fielding in seiner theoretischen Abhandlung explizit vier solcher 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 Ressourcen nicht das gleiche sind wie die Datenelemente aus der Persistenzschicht einer Anwendung. Sie befinden sich auf einem anderen Abstraktionsniveau. Beispiel: eine Warenkorbressource kann eine Auflistung von Artikeln sein, welche allerdings nicht einzeln als Ressource ansprechbar sind. Tilkov nimmt in diesem Zusammenhang eine Typisierung von Ressourcen vor. Von den sieben verschiedene Ressourcentypen sind folgende im Rahmen der Fragestellung interessant:

   a. Bei einer **Projektion** wird die Informationsmenge verringert, indem eine sinnvolle Untermenge der Attribute einer abgerufenen Ressource gebildet wird. Zweck ist die 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 sich aus Prozessen ergeben, wie etwa ein Schritt innerhalb einer Verarbeitung. Beispiel: Ein Schritt einer nicht abgeschlossenen Konfiguration.
- **Hypermedia** beschreibt das Prinzip verknüpfter Ressourcen. So wird dem Client ermöglicht, neue Ressourcen zu entdecken oder bestimmte Prozesse anzustoßen. Beispiel: Zur einer Bestellbestätigungsressource wird der zugehörige Stornierungslink hinzugefügt.
- **Standardmethoden/uniforme Schnittstelle**: Oben wurde beschreiben, dass jede Ressource durch (mindestens) eine ID identifiziert wird. Jede URI unterstützt dabei den gleichen Methodensatz, welche mit den HTTP-Methoden korrespondieren. Das bedeutet - übertragen auf die objektorientierte Programmierung: jedes Objekt implementiert das gleiche Interface. Folgende Teilmenge der neun verfügbaren HTTP-Methoden finden in der Literatur am häufigsten Erwähnung:
   a. **GET:** Das Abholen einer Ressource.
   b. **PUT:** Das Anlegen oder Aktualisieren einer Ressource. Je nachdem, ob unter dieser URI bereits 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:** Das Löschen einer Ressource.



Voreinstellung ist und explizit angezeigt werden muss). Die Spalte »sichtbare 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 Fragestellung: „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 die URL garantiert eine Ressource identifiziert.

Abbildung 5 fast die Eigenschaften der Methoden aus der HTTP-Spezifikation 1.1 zusammen. 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 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. In der Praxis wird meist eine serialisierte Variante eines Objektes als JSON zur Verfügung gestellt. Beispiel: Bereitstellung einer Bestellbestätigung als PDF und HTML.
- **Statuslose Kommunikation** bedeutet die Nichtexistenz eines serverseitig abgelegten, transienten, clientspezifischen Status über die Dauer eines Requests hinweg. Der Service benötigt also nie Kontextinformationen zur Bearbeitung eines Requests. Beispiel: Ein Warenkorb wird nicht in einem Sessionobjekt, sondern als persistentes Datenelement gehalten.

Diese Auflistung legt folgende Frage nahe: ist ein Webservice nur dann REST-konform, wenn alle Kriterien erfüllt werden? Was ist mit einem Webservice, der

allen Einschränkungen gerecht wird, 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 die abgestufte Bewertung eines Webservices nach dessen REST-Konformität erlaubt. Es wird im Auswertungsteil vorgestellt und zur Evaluierung der Implementierung genutzt.

Eigenschaften

Aus den vorgestellten Kriterien resultieren folgende Eigenschaften (Tilkov, 2011), welche die Vorteile REST-basierter Webservices gegenüber der SOAP-Konkurrenz darstellen (Richardson und Ruby, 2007):

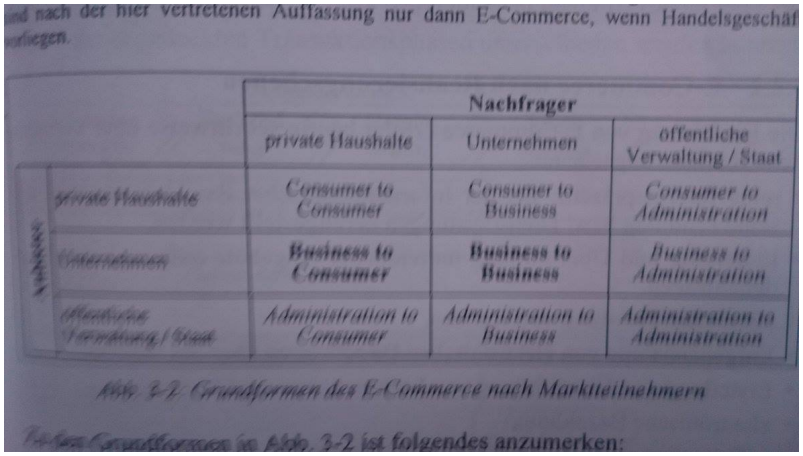
- **Lose Kopplung:** Beschreibt isolierte Systeme mit größtmöglicher Unabhängigkeit, die über Schnittstellen miteinander kommunizieren. Hierzu tragen die Standardmethoden bei.
- **Interoperabilität:** Beschreibt die Möglichkeit der Kommunikation von Systemen unabhängig von deren technischen Implementierung. Dies ergibt sich durch die Festlegung auf Standards. Bei der Anwendung von REST auf Webservices sind dies die Webstandards (z.B. HTTP, URIs).
- **Wiederverwendbarkeit**: 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**: Im Ideal sollen Webservices schnell antworten, unabhängig von der Anzahl von Anfragen in einem definierten Zeitraum. Dies wird durch Cachebarkeit (siehe HTTP-Methodenspezifikation) und Zustandslosigkeit erreicht. Da der Service keinen clientspezifischen Kontext aufbauen muss, müssen aufeinanderfolgende Requests nicht vom gleichen System beantwortet werden.

2.4 eCommerce

Im Folgenden wird durch die Charakterisierung des Begriffs eCommerce ein Anwendungsrahmen für eShops hergestellt. Deren softwaretechnische Umsetzung wird durch eShop-Systeme realisiert. Durch eine Kategorisierung der Systeme nach Anbieterstrategie wird abschließend die Menge der Open-Source-Lösungen für eine Konfiguratorintegration identifiziert.

2.4.1 Anwendungsrahmen

Eshops gehören zur Domäne des elektronischen Handels (eCommerce). Ecommerce ist „die elektronisch unterstützte Abwicklung von Handelsgeschäften auf der Basis der Internet“ (Schwarze und Schwarze, 2002). Je nachdem, welche Marktpartner an dem Handelsgeschäft teilnehmen, werden verschiedene Formen des eCommerce unterschieden. Die in Abbildung 6 fett hervorgehobenen Varianten werden von Meier und Stormer (2012) als „die zwei Geschäftsoptionen des eCommerce“ bezeichnet: Business-to-Customer (B2C) und Business-to-Business (B2B). Bei B2C erfolgt der Handel von Produkten und Dienstleistungen zwischen Unternehmen and Endverbraucher, bei B2B zwischen Unternehmen.



nach der hier vertretenen Auffassung nur dann E-Commerce, wenn Handelsgeschäf

		Nachfrager		
		private Haushalte	Unternehmen	öffentliche Verwaltung / Staat
Anbieter	private Haushalte	Consumer to Consumer	Consumer to Business	Consumer to Administration
	Unternehmen	**Business to Consumer**	**Business to Business**	Business to Administration
	öffentliche Verwaltung / Staat	Administration to Consumer	Administration to Business	Administration to Administration

Abb. 3-2: Grundformen des E-Commerce nach Marktteilnehmern

Bei den Grundformen in Abb. 3-2 ist folgendes anzumerken:

Abbildung 6: Grundformen des eCommerce nach Marktpartnern (Quelle: Schwarze und Schwarze (2002))

Für die Umsetzung von eCommerce existieren unterschiedliche Geschäftsmodelle. Timmers (1998) nennt 11 verschiedene Formen. eShops sind eine eine davon. Es handelt sich dabei um ein „Geschäftsmodell der Angebotsveröffentlichung, bei dem ein Anbieter seine Waren oder Dienstleistungen über das Web den Nachfragern offeriert“ (Bartelt u. a., 2000).

Ein eShop bildet den traditionellen Einkaufsvorgang nach: Kunden können mittels einer Katalog- oder Suchfunktion über den Produktbestand navigieren. Produkte können ausgewählt und ausführliche, mit Medien angereicherte Beschreibungen abgerufen werden. Wunschartikel werden einem virtuellen Warenkorb hinzugefügt. Ist die Produktauswahl abgeschlossen, begibt sich der Kunde zur „Kasse“, wo die Zahlungsmodalitäten erledigt werden (Boles und Haber, 2000). Ein eShop beschreibt das Geschäftsmodell, jedoch noch nicht dessen Umsetzung als Softwaresystem. Diese wird als eShop-System bezeichnet (Boles und Haber, 2000) und im Folgenden behandelt.

2.4.2 eShop-Systeme

„eShop-Systeme sind Software-Systeme, die den Aufbau, die Verwaltung und den Einsatz von eShops unterstützen“ (Boles und Haber, 2000). Abbildung 7 zeigt die Grobarchitektur eines eShop-Systems nach Meier und Stormer. Darin wird die Unterteilung zwischen Storefront und Backfront deutlich, welche in der Terminologie realer Shopsysteme als Front- und Backend bezeichnet werden (vgl. Shopware AG, 2015a). Das Frontend ist der Interaktionsraum der Kunden, das Backend der adminsitrative Bereich des Shopbeitreibers.

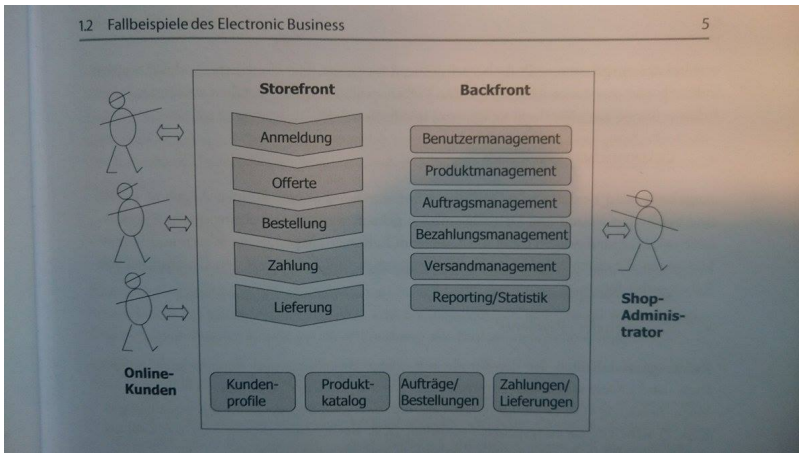


Abbildung 7: Grobarchitektur eines eShop-Systems (Quelle: Meier und Stormer (2012))

Die Hauptaufgaben eines eShop-Systems sehen Boles und Haber (2000) in den Bereichen Merchandising (z.B. Management von hierarchisch strukturierten Produktkatalogen, Beeinflussung des Shopdesigns), Auftragsbearbeitung (z.B. Festlegung der Abarbeitungs-Pipeline, Integration von Bezahlverfahren) und Sonstiges (z.B. die Kopplung mit externen ERP-Systemen). Der konkrete Funktionsumfang hängt vom gewählten eShop-System ab.

Die Systeme sind nach Strategie der Anbieter kategorisierbar:

- **Open-Source**-Systeme sind kostenlos verfügbar. Sie bieten völlige Gestaltungsfreiheit, aber keinen Herstellersupport. Die Dokumentationen sind schwacher und der Funktionsumfang geringer als bei kostenpflichtigen alternativen. Andererseits existieren Communities, die Unterstützung bieten und die Entwicklung von Erweiterungen vorantreiben (Stahl u. a., 2015). Beim kommerziellen Handel der modularen Erweiterungen auf shopspezifischen Stores liegt auch eine der wesentlichen Erlösquellen der Open-Source Strategie (z.B. der Addon Marketplace der PrestaShop SA oder das Extensionverzeichnis der Opencart Limited).
- **Kauf-Lösungen** können kostenpflichtig lizensiert werden (z.B. Shopware AG). Sie bieten Herstellersupport, zusätzliche Dienstleistungen (z.B. Installation des Shops) und einen höheren Funktionsumfang (z. B. Schnittstellen zu verschiedenen Warenwirtschaftssystemen oder Zahlungsdienstleistern) (Stahl u. a., 2015). Die Hersteller bieten verschiedene Editionen mit teilweise erheblichen Preisunterschieden an (Bsp.: die Preisdifferenz der Magento Enterprise Edition zu Enterprise Premium liegt bei über 35.000 $, vgl FWP shop, 2014).
  a. Im Rahmen eines Dual-License-Modells ist eine Open-Source **Community Edition** Teil des Editionsspektrums (t3n, 2014) (vgl. das Shopangebot der Magento Inc., Shopware AG oder OXID eSales AG). Durch den offenen Quellcode existiert auch hier der Handel modularer Erweiterungen, von dem auch die kostenpflichtigen Varianten profitieren (vgl. der Plugin Store der Shopware AG). Aufgrund der gleichen Codebasis alles Editionen kann zu einer Kauf-Lösung migriert werden, was Flexibilität für wachsende Shopanforderungen bietet.
- **Miet-Shops** entsprechen einer Cloud-Lösung als Software-as-a-Service (z.B. Strato AG, Shopify). Die technische Infrastruktur wird vom Provider zur Verfügung gestellt. Systemwartung, Bereitstellung der Shopsoftware und Hosting werden unter dem Mietpreis abgerechnet. Stahl u. a. (2015) bewertet diese Variante als Einstiegslösung mit geringer Gestaltungsfreiheit.
- **Eigenentwicklungen** eignen sich für individuelle Bedürfnisse, wenn die Standardsysteme die Anforderungen nicht mehr erfüllen (Stahl u. a., 2015; Graf, 2014).

Aus der Liste sind die (zumindest initial) kostenfreien Varianten ersichtlich: reine Open-Source eShop-Systeme sowie die Community-Editionen der Dual-License Modelle. Eine Anbieterübersicht ist t3n (2014) zu entnehmen. Eine Kategorisierung der Systeme nach Anforderungsklassen ist Graf (2014) zu entnehmen.

3 Analyse

Die Tacton Systems AB (Tacton) wurde 1998 als Spin-Off des Schwedischen Instituts für Informatik (SICS) gegründet (Tacton Systems AB, 2007). In der Forschungseinrichtung wurde als Resultat der Untersuchungen im Bereich Wissensbasierte Systeme und Künstliche Intelligenz der Tacton Produktkonfigurator entwickelt (Tacton Systems AB, 2015). Dieser interaktive Konfigurator ist die Basis der verschiedenen Produkte der Firma.

Tacton bietet Lösungen im Bereich Vertriebskonfiguration und Design Automation (Automatisierung der Konstruktion in CAD-Systemen).

3.1 Konfigurationsmodell

Das in Abschnitt 2.2.2 vorgestellte Visualisierungskonzept abstrahiert Konfigurationswissen in einen Struktur- und einen Regelteil. Im Tacton-Konfigurationsmodell wird ebenfalls abstrahiert, jedoch in andere Domänen (**?**):

a. Strukturinformation: welche Teile-Hierarchie hat das Produkt?

b. Komponenteninformation: welche Komponententypen stehen für die Teile zur Verfügung?

c. Constraintinformationen: wann ist das Produkt korrekt?

d. Ausführungsinformationen: welche Fragen bekommt der Nutzer gestellt? Bekommt er sie in einer bestimmten Reihenfolge gestellt? Welche Optimierungskritieren werden verwendet?

Dabei sind zwei Dinge festzuhalten:

(1) Typisch für einen modellbasierten Konfigurator wird Produktwissen und Problemlösungswissen bei der Wissensmodellierung separiert. Beispiel: ein Constraint, der die Kompatibilität bestimmter Betriebssystems mit einer bestimmten Anzahl Prozessorkerne ausdrückt, soll wirken, auch ohne die konkreten CPU-Arten (z.B. Intel i7) zu kennen. Darum ist - genauer ausgedrückt - die Rede von generischen Constraints: sie beziehen sich auf alle Komponenten eines Typs (Felfernig u. a., 2014).

(2) Im Konfigurationsmodell wird auch die Nutzerinteraktion festgelegt. Es wird also auch definiert, welche Entscheidungen in welcher Reihenfolge getroffen werden können. Damit geht der Funktionsumfang eines Tacton-Modells über die Definition aus Abschnitt 2.2.1 hinaus.

3.1.1 Components und Configuration

Bei der UML-Wissensrepräsentation aus Abschnitt 2.2.2 wurden die Domänen b und a im Diagramm ausgedrückt, c in der Tabelle und teilweise im Diagramm. Tacton wählt eine andere Aufteilung. b wird als „Components“ konzipiert, a und c als „Configuration“.

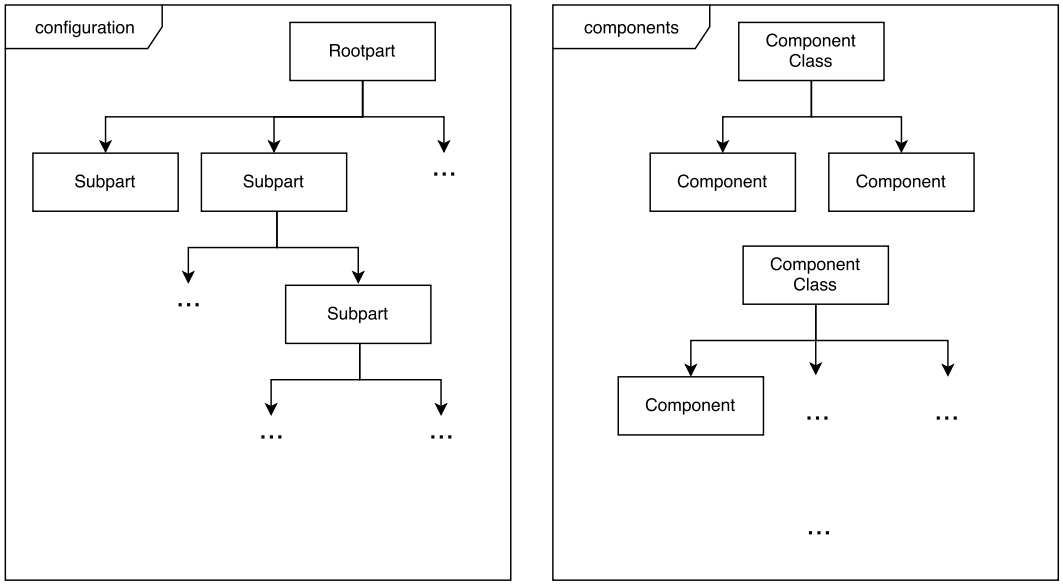


Abbildung 8: High-Level Architektur des Tacton-Konfigurationsmodells

Abbildung 8 zeigt das High-Level Konzept der Modellarchitektur. Unter ’configuration’ wird die Produktstruktur als hierarchischer Baum von ’Part’-Objekten dargestellt. Jeder Part kann Constraints enthalten, die sich auf den Knoten selbst und alle seine Kinder beziehen. Ein Part ist ansonsten nur ein Komponentenplatzhalter. Noch ist keine Information darüber hinterlegt, welches Bauteil dort eigentlich verkörpert wird. Abbildung 9 veranschaulicht das Konzept am Notebook-Beispiel.

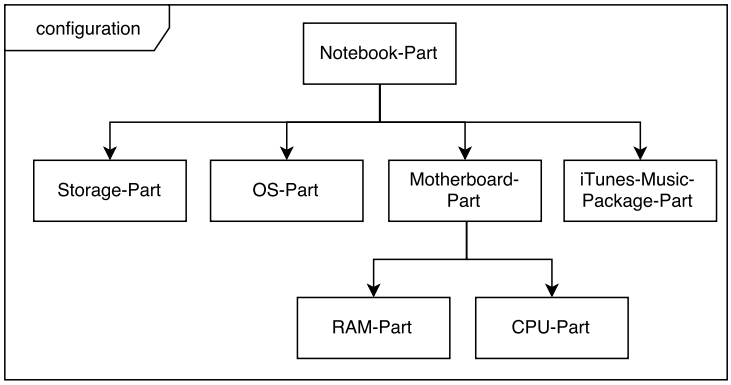


Abbildung 9: Part-Struktur der Notebook-Konfiguration

Die Informationen über die eben erwähnten Bauteile werden isoliert unter 'components' definiert (Abbildung 8). Analog zu den Komponententypen und Komponenten aus Abschnitt 2.2.2 wird ein Bauteil in 'Component Classes' und 'Components' abstrahiert. Hier wurde einfach eine andere Terminologie gewählt. Eine Component Class (z.B. ein 'RAM') wird durch Eigenschaften beschrieben, die als Features bezeichnet werden. Jedes Feature besitzt einen Namen (z.B. 'memory') und einen Wertebereich (z.B. [2GB, 4GB]), welche als Domain bezeichnet wird. Wertebereiche können Integer, Float, Boolean, andere Component Classes oder selbstdefinierte Enumerationen sein. Components sind das, was eine Component Class konkret sein kann (z.B. ein 'Small RAM'). Sie übernehmen alle Features der übergeordneten Component Class und besitzen konkrete Werte ('Values') aus dem jeweiligen Wertebereich.

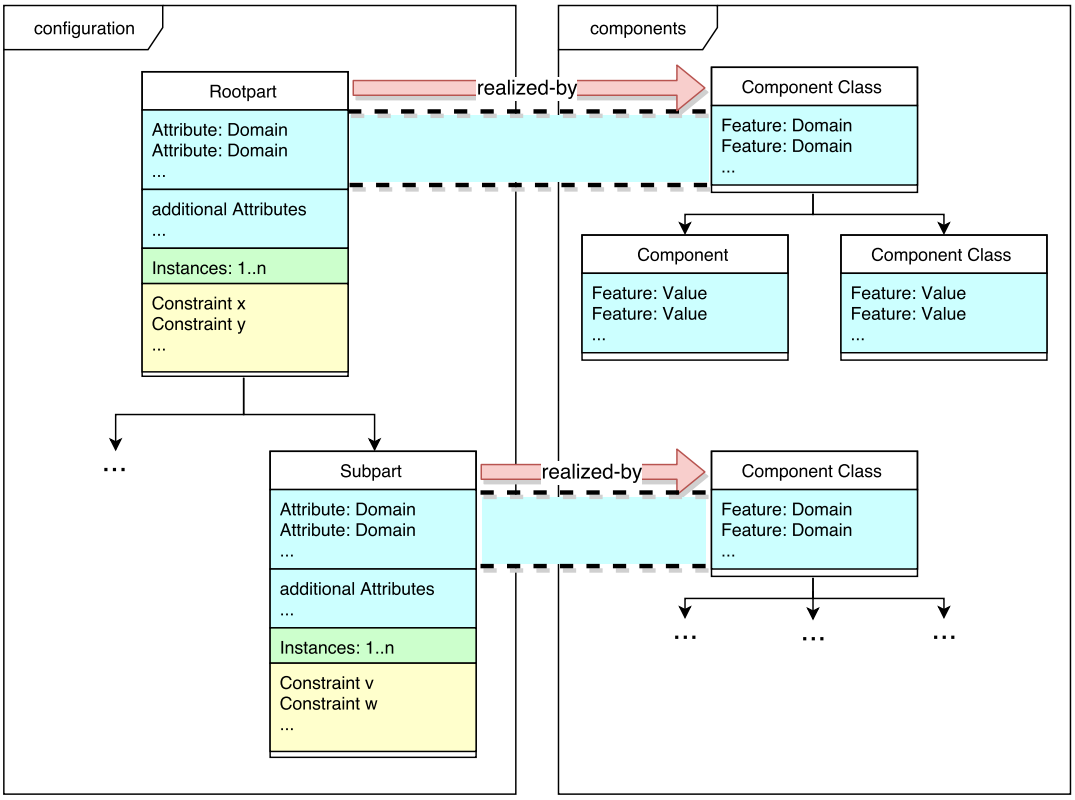


Abbildung 10: Zuordnung von Parts und Component Classes.

Abbildung 10 zeigt die Zuordnung zwischen Parts und Component Classes via Verzeigerung. Einem Part wird eine Component Class durch eine 'realized-by' Beziehung zugewiesen. Dem Part werden dabei die Eigenschaften (Features) der entsprechenden Component Class vererbt (türkis dargestellt). Nur werden sie zur besseren Differenzierung beim Part nicht mehr als Features, sondern als Attribute bezeichnet. Für einen Part können auch noch zusätzliche Attribute definiert werden ('additional Attributes'), falls notwendig. Die Angabe 'Instances' (grün dargestellt) entspricht den Kardinalitäten in Abschnitt 10. Constraints (gelb dargestellt) werden als logische Ausdrücke formuliert. Sie bestehen aus Attributwerten, die mit mathematischen Zeichen in Relation gebracht werden.

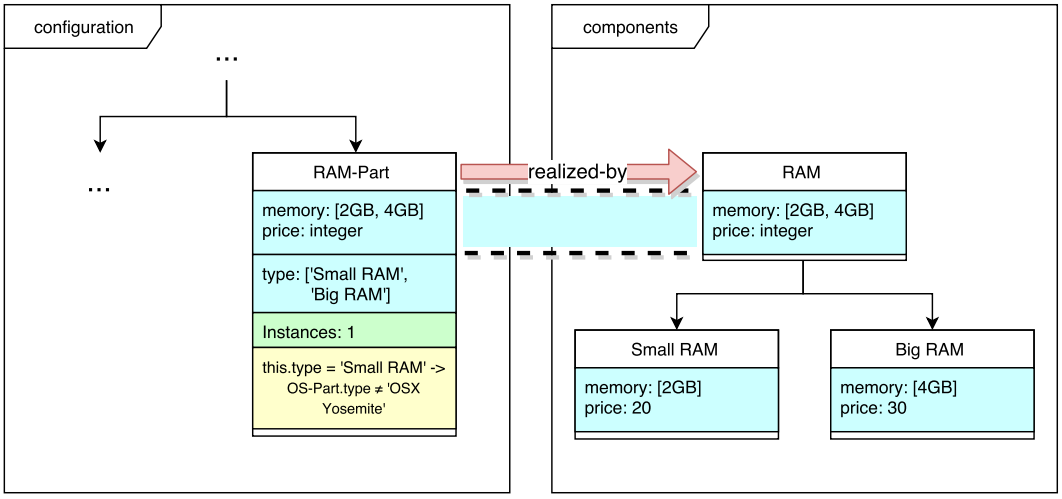


Abbildung 11: Part-Struktur der Notebook-Konfiguration

Abbildung 11 veranschaulicht die 'realized-by' Beziehung an einem Ausschnitt aus der Notebook-Konfiguration. Der Part übernimmt die Attribute 'memory' und 'price'. Das zusätzliche Attribut 'type' kapselt Beispielsweise die Information über das gewählte Component (z.B. Small RAM). Der Constraint veranschaulicht exemplarisch, wie die Inkompatibilität mit dem Betriebssystem vom Typ 'OSX Yosemite' formuliert werden würde.

3.1.2 Execution

Es wurde dargestellt, wie das Konfigurationswissen im Tacton Konfigurationsmodell definiert wird. Dadurch ist aber noch nicht gesagt, welche Entscheidungen ein Nutzer während des Konfigurationsprozesses treffen kann. Nicht jeder Part und nicht jedes Attribut muss eine relevante Wahl darstellen. Vielleicht sollen dem Nutzer sogar Fragen auf einem anderen Abstraktionsniveau als auf Komponentenebene gestellt werden. Statt „Soll eine HD oder eine SSD als Festplatte in das Notebook eingebaut werden?“ kann auch gefragt werden: „Möchten Sie viel Speicherplatz oder einen schnellen Speicherzugriff?“.

Die Interaktionsspielraum des Anwenders mit der Konfiguration wird unter dem Begriff 'Execution' zusammen gefasst. Dabei wird nicht einfach nur eine Menge an Optionen festgelegt, die dem Nutzer am Ende als Liste präsentiert wird. Stattdessen werden die Optionen hierarchisch gegliedert.

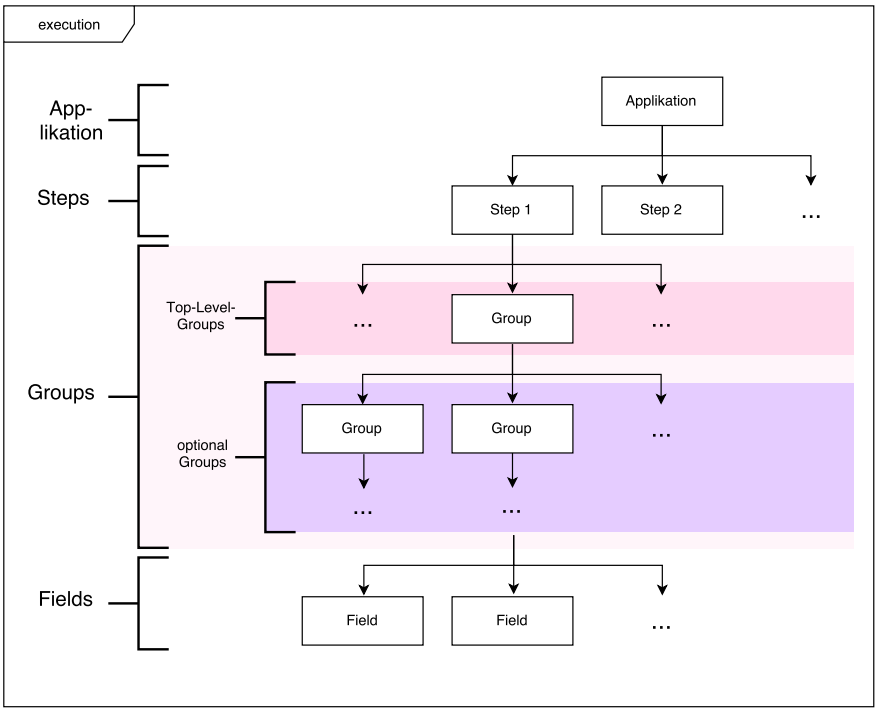


Abbildung 12: Generische Executionstruktur

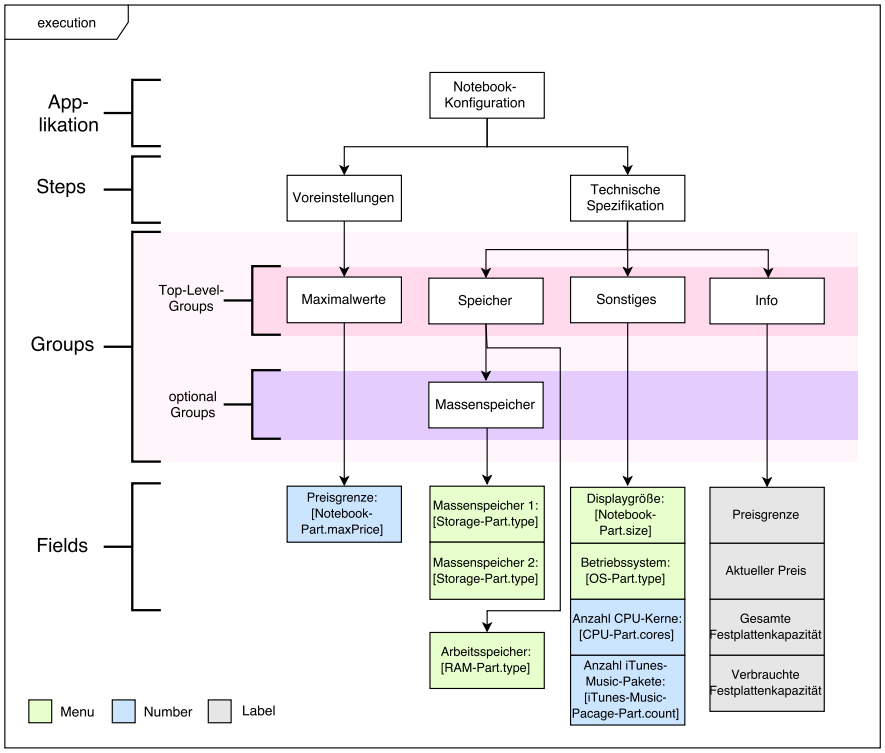


Abbildung 13: Exemplarische Executionstruktur einer Notebook-Konfiguration

Abbildung 12 zeigt die generische Struktur der Execution als Baum. Abbildung 13

veranschaulicht das Konzept durch die beispielhafte Umsetzung einer Execution der Notebook-Konfiguration.

- Die Wurzel wird als **Applikation** bezeichnet und kapselt den Konfigurationsprozess aus Anwenderperspektive.
  Beispiel: Eine Notebook-Konfiguration.
- Die nächste Knotenebene wird als **Steps** bezeichnet. Sie legen fest, in welcher Reihenfolge die Optionen präsentiert werden. Das ist in verschiedenen Fällen sinnvoll. Zum Beispiel dann, wenn komplexe Probleme in mehrere Schritte unterteilt oder Werte für spätere Schritte gesetzt werden sollen.
  Beispiel: In Step 1 legt der Anwender eine Preisgrenze fest. In Schritt 2 wird die technische Spezifikation getroffen.
- Nun folgen 1..n Knotenebenen, die als **Groups** bezeichnet werden. Sie fassen Optionen zu logischen Einheiten zusammen. Die in Gruppen angeordneten Optionen müssen nicht in einer bestimmten Reihenfolge beantwortet werden.
  (a) Die oberste Group-Ebene (**Top-Level-Groups**) ist obligatorisch.
  Beispiel: Eine 'Speicher'-Group in Step 2.
  (b) Die Unterteilung in weitere Groups ist optional. Es kann beliebig tief geschachtelt werden.
  Beispiel: unter 'Speicher' werden die Groups 'Massenspeicher' und 'RAM' angelegt.
- Die tatsächlichen Optionen werden als **Fields** bezeichnet. Sie bestehen aus einer Beschreibung und einem Interaktionselement, über das der Anwender eine Entscheidung treffen kann. Eine Entscheidung bedeutet: ein Wert aus dem Wertebereich von Part-Attributen.
  Folgende Interaktionselemente stehen zur Verfügung:
  (a) **Menu:** Wahl aus einem Wertebereich
  Beispiele:
  Beschreibung: Massenspeicher / Wertebereich: ['HD', 'SSD']
  Beschreibung: Anzahl CPU-Kerne / Wertebereich: [1..4].
  (b) **Number:** Eingabe eines eigenen Wertes in ein Textfeld.
  Beispiel: Eingabe der Preisgrenze in Schritt 1.
  (c) **Label:** Anzeige eines Wertes aus Informationsgründen.
  Beispiel: Anzeige des Gesamtpreises.

Das Modell wird in einem XML-basierten Modelldatei mit der Endung '.tcx' abgelegt. Die Bildung der Datei in einer vom Konfigurator interpretierbaren Struktur wird durch Anwendungswerkzeuge unterstützt. Hierfür kann zum Beispiel TCstudio genutzt

werden, welches eine grafische Oberfläche zur Modellentwicklung bietet (Tacton Systems AB, 2015).

Zwischenfazit

Das Tacton-Modellierungskonzept wurde vorgestellt. Neben dem Konfigurationswissen wird auch festgelegt, welche Entscheidungen der Anwender treffen kann. Die Optionen werden durch Steps und Groups logisch gegliedert. So wird festgelegt, welche Struktur die Interaktion des Anwenders mit der Konfiguration hat. Diese Struktur muss jedoch noch dargestellt werden. TCsite realisiert eine entsprechende Darstellung. Diese wird im Folgenden analysiert.

3.2 TCSite

TCsite ist ein webbasierter Vertriebskonfigurator. Er könnte zwar theoretisch von Endkunden genutzt werden, bildet aber nicht die Geschäftsprozesse eines eShops nach - stattdessen handelt es sich um eine CPQ-Lösung (Configure-Price-Quote). Das bedeutet: der Anwender wird durch den Konfigurationsprozess geführt, was in einer Preiskalkulation resultiert, woraus wiederum ein Angebot erstellt werden kann. Eine unmittelbare Bestellung ohne Angebotsprozess ist nicht vorgesehen. Der Anwendungsbereich liegt also im B2B (Tacton Systems AB, 2015).

Technisch betrachtet ist TCsite eine Webanwendung. Im Lieferumfang sind 'Apache Tomcat' als Application Server sowie TCserver enthalten. TCserver ist das, was man gemäß Kapitel 2.2 als eigentlicher Konfigurator bezeichnet wird. Er beherbergt die Konfigurationsengine. Damit ist das System gemeint, welches Konfigurationsaufgaben verarbeitet und Ergebnisse in unterschiedlicher Form präsentiert (**?**). Abbildung 14 veranschaulicht das High-Level Architekturkonzept des Standardsetups.

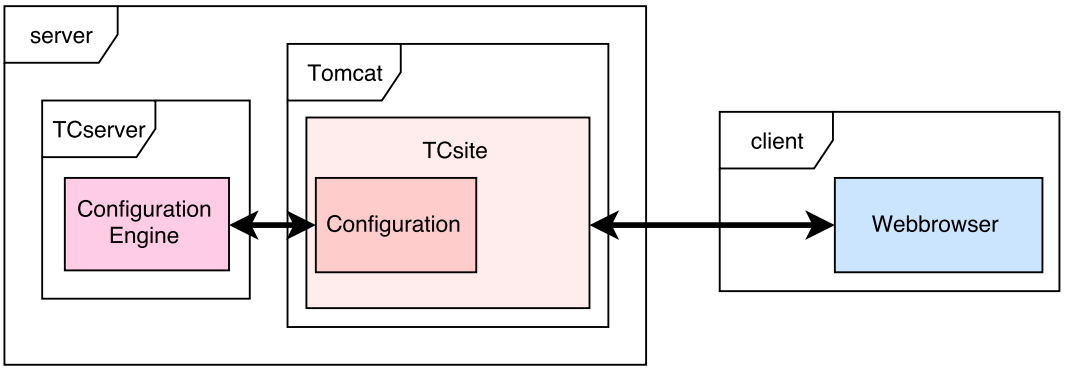


Abbildung 14: High-Level-Architektur von TCsite

3.2.1 Architektur

TCsite basiert auf einer offenen Schichtenarchitektur, wie Abbildung 15 visualisiert. Offen bedeutet, dass jede Schicht mit allen darunter liegenden Schichten kommunizieren kann. Das Fundament bildet die Persistenzschicht. Sie realisiert die Datenhaltung. Die Plattform bildet API mit den Basisfunktionalitäten für die darüber liegenden Schichten. Die Module bilden jeweils eine der drei Oberflächenbereiche der Anwendung. Außerdem bieten sie Services und Erweiterungspunkte für die oberste Schicht: die Plugins. Durch sie kann die Funktionalität von TCsite erweitert werden (**?**).

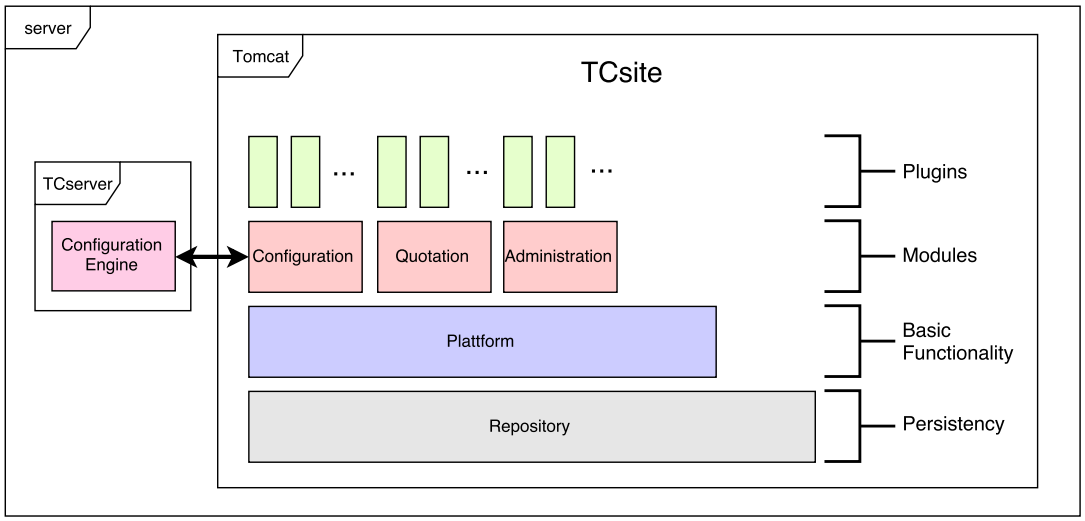


Abbildung 15: Schichtenarchitektur von TCsite

Im Folgenden werden die Architekturkomponenten von unten nach oben vorgestellt.

Repository

Das Repository bietet eine Datenbank sowie ein Reihe von Funktionen zum Lesen und Schreiben der TCsite-Objekte. Die Interaktion mit der Datenhaltung wird so für darüber liegende Schichten abstrahiert. Das Repository ist in einen lokalen und einen globalen Speicher strukturiert. Wird ein Objekt erstellt oder geöffnet, geschieht die Bearbeitung immer auf einer Arbeitskopie in dem für jeden Nutzer spezifischen lokalen Speicher. Änderungen sind solange für andere Nutzer unsichtbar. Erst die Übertragung (commit) eines Objektes in den globalen Speicher sorgt für nutzerübergreifende Sichtbarkeit. Bei dieser Übertragung wird eine Revisionshistorie über Objektänderungen geführt, so dass alte Zustande wieder abrufbar sind (**?**).

Administration

Über die Administrationsoberfläche werden Einstellungen und Verwaltungsaspekte realisiert. Dazu gehört zum Beispiel die Verwaltung der Nutzergruppen und Produkte. Vorangelegte Nutzergruppen sind Standarduser, Systemadministrator und der Integrationuser. Während der Handlungsspielraum von Standardusern eingeschränkt werden kann, dürfen Administratoren alle Einstellungen an der Installation vornehmen. Der Integration User dient der Authentifizierung bei der Kommunikation mit externen Systemen, zum Beispiel bei einem Integrationsszenario. Anschaulich ist die Vorstellung, dass jede externe Webanfrage einen Gast personifiziert, der die Maske des Integration Users aufgesetzt und dessen Rechtespektrum bekommt. Außerdem wird der Produktkatalog verwaltet. Jedem Produkt wird dabei eine '.tcx'-Modelldatei und so mit der entsprechenden Konfiguration verbunden. Entsprechend der Definition eines Konfigurationsmodells in Kapitel 2.2 steht ein hier angelegtes Produkt nicht im klassischen Sinne für einen bestimmten Artikel, sondern für alle seine Varianten **??**.

Quotation

Das Quotation-Modul erstellt die Quotation-View (Abbildung x) und verwaltet das zentrale Objekt von TCsite: die Quotation (zu Deutsch: das Angebot). TCsite kann in einer abstrakten Sicht als Liste von Quotations in Verbindung mit Methoden für deren Manipulation, Anzeige und Dokumentengenerierung betrachtet werden. Aus Anwendersicht ist eine Quotation ein Angebot, dass die für ihn konfigurierten Produkte enthält. Diese werden als QuotationItems bezeichnet. Betrachtet man eine Quotation als Warenkorb, sind die QuotationItems die Warenkorbartikel. Abbildung 16 veranschaulicht das Verhältnis von Quotations zu QuotationItems (**?**).

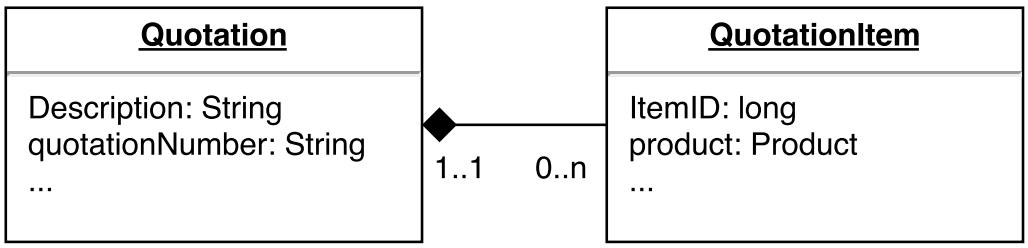


Abbildung 16: Verbindung von Quotations und QuotationItems

Quotations haben weiterhin einen Lebenszyklus. Sie befinden sich zu jedem Zeitpunkt in einem bestimmten Zustand, welche über den Administrationsbereich verwaltet werden könnnen. Die Zustände unterscheiden sich in der Sichtbarkeit für verschiedene Nutzergruppen und der Editierbarkeit der QuotationItems. Beispielsweise kann sich

eine Quotation in den Zuständen „Design“ oder „Offered“ befinden. In letzterem Zustand sind die QuotationItems nicht mehr konfigurierbar, da sonst die Quotation von dem Angebot abweichen würde, das dem Kunden vorliegt (**?**).

Die Quoation beinhaltet fertig konfigurierte Produkte, also Variantenausprägungen. Die Konfiguration eines Produktes wird vom Quotation Modul aus über die Schaltfläche sowieso gestartet. Daraufhin wählt der Anwender das gewünschte Produkt aus und der Übergang zum Configuration Modul geschieht.

Configuration
Das Configuration Modul realisiert den C-Teil des CPQ-Prozesses: die Konfiguration. Es rendert die Konfigurationsansicht und verwaltet die mit der Konfiguration im Zusammenhang stehenden Objekte. Im Modul wird zwar die Konfigurationsaufgabe zusammengestellt, aber nicht gelöst. Dies übernimmt der TCserver. Somit ist die Konfiguration als Client-Server Achitektur realisiert. Die Konfigurationsengine wird so oft aufgerufen, bis die Konfiguration beendet ist.

3.2.2 Erweiterbarkeit

Fazit
TCsite ist also webbasiert und beherbergt eine Konfigurationsengine. Grundlegende Idee und damit Gegenstand der Analyse ist also: kann die Engine als Webservice nutzbar gemacht werden? Dafür muss das Zusammenspiel der Webanwendung mit der Konfigurationsengine während des Konigurationsprozesses in TCsite untersucht werden. Ist der Konfigurationsprozess verstanden, lässt sich daraus ein Konzept für einen Konfigurations-Webservice ableiten. Voraussetzung ist jedoch Kenntnis darüber, wie Konfigurationswissen bei Tacton in Form eines Konfigurationsmodells abgebildet wird. Dies wird im Folgenden vorgestellt.

Auch das Drumherun, dass es nen TCserver gibt, der ausgelagert werden kann. Dass es in Tomcat läuft und dementsprechend eine Adresse hat.

Man muss wohl sagen, dass es Nutzer gibt. So kann man den Integration user auch erwähnen.

Eventuell das Cite-Concept völlig unter den Tisch fallen lassen.

Also irgendwie Grobe technische Übersicht. Schichtenarchitektur und so.

Analyse Konfigurationsprozess Analyse Erweiterbarkeit

3.3 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 17, wird die *minipage*-Umgebung genutzt, da die Bilder so gut positioniert werden können.

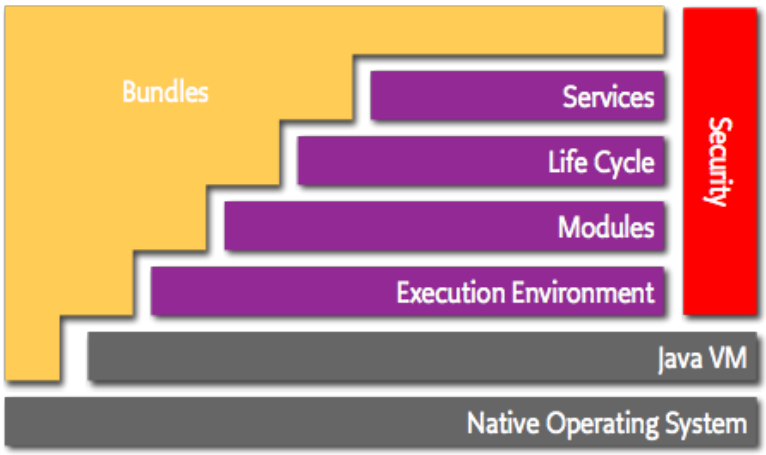


Abbildung 17: OSGi Architektur[1]

8.2 Tabellen

In diesem Abschnitt wird eine Tabelle (siehe Tabelle 2) dargestellt.

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

Tabelle 2: 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

[Bartelt u. a. 2000] Bartelt, Andreas ; Weinreich, Harald ; Lamersdorf, Winfried: Kundenorientierte Aspekte der Konzeption von Online-Shops / Universität Hamburg, Fachbereich Informatik, Verteilte Systeme (VSYS). 2000. – Forschungsbericht

[Boles und Haber 2000] Boles, Dietrich ; Haber, Cornelia: DDS: Ein Shop-System für den Informationcommerce / Informatik-Institut OFFIS Oldenburg. 2000. – Forschungsbericht

[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

[FWP shop 2014] FWP shop: *Unterschied: Magento Community vs Enterprise Edition.* 2014. – URL `http://www.fwpshop.org/shopsysteme/magento-shop/versionsunterschiede`. – Zugriff: 25.08.2015

[Graf 2014] Graf, Alexander: *Das beste Shopsystem 2014 – Erwartung vs. Wirklichkeit.* 2014. – URL `http://www.fwpshop.org/shopsysteme/magento-shop/versionsunterschiede`. – Zugriff: 26.08.2015

[Hadzic und Andersen 2004] Hadzic, Tarik ; Andersen, Henrik R.: An introduction to solving interactive configuration problems / IT University of Copenhagen. 2004. – Forschungsbericht

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

[Magento Inc. 2015] Magento Inc.: *Magento Products Overview*. 2015. – URL `http://magento.com/products/overview`. – Zugriff: 25.08.2015

[Meier und Stormer 2012] Meier, Andreas ; Stormer, Henrik: *eBusiness und eCommerce: Management der digitalen Wertschöpfungskette*. 3. Auflage. Springer Gabler, 2012

[Opencart Limited 2015] Opencart Limited: *Opencart Extensions*. 2015. – URL `http://www.opencart.com/index.php?route=extension/extension`. – Zugriff: 25.08.2015

[OXID eSales AG 2015] OXID eSales AG: *OXID eSales Produkte*. 2015. – URL `https://www.oxid-esales.com/de/produkte.html`. – Zugriff: 25.08.2015

[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*. 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

[PrestaShop SA 2015] PrestaShop SA: *Prestashop Addon Marketplace*. 2015. – URL `http://addons.prestashop.com/`. – Zugriff: 25.08.2015

[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

[Schomburg 1980] Schomburg, Eckart: *Entwicklung eines betriebstypologischen Instrumentariums zur systematischen Ermittlung der Anforderungen an EDV-gestützte Produktionsplanungs- und -steuerungssysteme im Maschinenbau*, RWTH Aachen, Dissertation, 1980

[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

[Schwarze und Schwarze 2002] Schwarze, Jochen ; Schwarze, Stephan: *Electronic Commerce: Grundlagen und praktische Umsetzung*. Herne/Berlin : Verlag Neue Wirtschafts-Brife, 2002

[Shopify 2015] Shopify: *Shopify Pricing*. 2015. – URL `http://www.shopify.com/pricing`. – Zugriff: 26.08.2015

[Shopware AG 2015a] Shopware AG: *Shopware Dokumentation*. 2015. – URL `http://community.shopware.com/Doku_cat_938.html`. – Zugriff: 25.08.2015

[Shopware AG 2015b] Shopware AG: *Shopware Plugin Store*. 2015. – URL `http://store.shopware.com/`. – Zugriff: 25.08.2015

[Shopware AG 2015c] Shopware AG: *Shopware Pricing*. 2015. – URL `https://en.shopware.com/pricing/`. – Zugriff: 25.08.2015

[Soininen u. a. 1998] Soininen, Timo ; Tiihonen, Juha ; Männistö, Tomi ; Sulonen, Reijo: Towards a general ontology of configuration. In: *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 12 (1998), Nr. 4, S. 357–372

[Stahl u. a. 2015] Stahl, Ernst ; Wittmann, Georg ; Krabichler, Thomas ; Breitschaft, Markus: *E-Commerce-Leitfaden: Noch erfolgreicher im elektronischen Handel*. 3., vollständig überarbeitete und erweiterte Auflage. Universitätsverlag Regensburg, 2015

[Strato AG 2015] Strato AG: *Strato Webshops*. 2015. – URL `https://www.strato.de/webshop`. – Zugriff: 25.08.2015

[t3n 2014] t3n: *Open Source: 16 Shopsysteme im Überblick*. 2014. – URL `http://t3n.de/news/open-source-shopsysteme-13-losungen-uberblick-286546/`. – Zugriff: 26.08.2015

[Tacton Systems AB 2007] Tacton Systems AB: *Bridging the gap between engineering and sales for complex products*. 2007. – internes Präsentionsdokument zur Produktübersicht

[Tacton Systems AB 2015] Tacton Systems AB: *About Tacton*. 2015. – URL `http://www.tacton.com/about-tacton/`. – Zugriff: 27.08.2015

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

[Timmers 1998] Timmers, Paul: Business Models for Electronic Markets. In: *Electronic Markets* 8 (1998), Nr. 2, S. 3–8

[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)