Universität Passau

Fakultät für Informatik und Mathematik

Bachelorarbeit

Entwurf und Implementierung einer optimalen Benutzerobfläche zur Darstellung komplexer Produktkonfigurationen mit Schwerpunkt auf der Anzeige und Auswahl von alternativen Konfigurationsschritten

Autor:
Patrick Kuglmeier

Betreuer:
Dr.-Ing. Sven Apel

March 23, 2013

Zusammenfassung

Benutzeroberflächen von webbasierten Produktkonfiguratoren sind mittlerweile immer größeren Ansprüchen ausgesetzt. Die Komplexität der angebotenen Produkte und die Zahl der darzustellenden Elemente steigt stetig an. Gleichzeitig soll jedoch die Bedienung durch den Benutzer immer intuitiver und einfacher möglich sein. Um dies zu gewährleisten müssen diese Benutzeroberflächen die Regeln guter Usability einhalten und darauf optimiert werden.

In der vorliegenden Arbeit wurden zu diesem Zweck die Anforderungen an die Benutzeroberflächen analysiert, neue Ansätze zur intelligenten Benutzerführung konzipiert und prototypisch implementiert. Dabei ist zum einen eine neue Konfigurationsoberfläche entstanden - das *intelligente Formular* - zum anderen wurde ein Verfahren entwickelt, um Konfigurationskonflikte schnell und benutzerfreundlich zu lösen - der *Alternativen-Wizard*.

Abschließend wurden die Ergebnisse evaluiert. Die Bewertung ergab, dass die neuen Konzepte eine deutliche Verbesserung der Benutzerfreundlichkeit mit sich bringen, diese in Zukunft aber im Detail noch optimiert und weiterentwickelt werden müssen.

Inhaltsverzeichnis

Abbildungsverzeichnis **iv**

Tabellenverzeichnis **vi**

1 Einleitung **1**
1.1 Motivation . . . 1
1.2 Ziel der Arbeit . . . 2
1.3 Aufbau der Arbeit . . . 2

2 CAS Configurator Merlin **4**
2.1 Architektur . . . 4
2.2 Datenmodell . . . 5
2.2.1 Element . . . 6
2.2.2 Regel . . . 6
2.2.3 Regelwelt . . . 6
2.2.4 Produktplan . . . 6
2.2.5 Produkt . . . 7
2.2.6 Gültiges Produkt . . . 7
2.2.7 Das Datenmodell des CAS Configurator Merlin . . . 8
2.3 Konfigurationsregeln . . . 9
2.3.1 Aktiviert-Regel . . . 9
2.3.2 Nicht-mit-Regel . . . 10
2.3.3 Impliziert-Regel . . . 10
2.3.4 Operatorbäume . . . 11
2.4 Bisherige Benutzeroberfläche . . . 11

3 Usability bei Produktkonfiguratoren **13**
3.1 Allgemeine Gestaltungsgesetze . . . 13
3.1.1 Das Gesetz der Nähe . . . 13
3.1.2 Das Gesetz der Kontinuität . . . 15
3.1.3 Das Gesetz der Geschlossenheit . . . 15
3.1.4 Das Gesetz der Gleichheit . . . 16
3.2 Usability - Was ist das? . . . 17
3.3 Kundenbedürfnisse . . . 17
3.3.1 Bedienbarkeit und Selbsterklärung . . . 18
3.3.2 Laden der Anwendung . . . 19
3.3.3 Hilfestellungen . . . 20
3.3.4 Feedback . . . 20
3.4 Grundsätze der Dialoggestaltung - ISO 9241-110 . . . 21

3.4.1 Aufgabenangemessenheit . . . 21
3.4.2 Selbstbeschreibungsfähigkeit . . . 22
3.4.3 Erwartungskonformität . . . 22
3.4.4 Lernförderlichkeit . . . 23
3.4.5 Steuerbarkeit . . . 23
3.4.6 Fehlertoleranz . . . 23
3.4.7 Individualisierbarkeit . . . 24
3.5 Ausgewählte Gründe für benutzerinitiierte Prozessabbrüche . . . 24
3.5.1 Vielzahl an Konfigurationsmöglichkeiten . . . 25
3.5.2 Fehlende persönliche Beratung . . . 25
3.5.3 Irreführende Benutzerführung . . . 25
3.6 Interaction Design Patterns für die Produktkonfiguration . . . 26
3.6.1 Ziele und Angebote . . . 26
3.6.2 Navigation und Führung . . . 27
3.6.3 Auswahl und Feedback . . . 28

4 Intelligentes Formular zur Produktkonfiguration 29
4.1 Verwendete Technologie . . . 29
4.2 Spezifikation . . . 29
4.2.1 Die Merkmalsstruktur . . . 29
4.2.2 Vertriebsprodukte und Vertriebsbausteine . . . 30
4.2.3 Kardinalitäten . . . 31
4.2.4 Grafischer Entwurf des intelligenten Formulars . . . 32
4.3 Implementierung . . . 33
4.3.1 Wichtige Datentypen . . . 33
4.3.2 Die CatalogBean . . . 36
4.3.3 Die ConfigurationBean . . . 38

5 Dialog zur Darstellung alternativer Konfigurationen 40
5.1 Ursache alternativer Konfigurationen . . . 40
5.1.1 Ablauf eines Konfigurationsschrittes . . . 40
5.1.2 Lösungsalternativen . . . 41
5.2 Stand heute . . . 42
5.3 Konzepte einer intelligenten Benutzerführung . . . 43
5.3.1 Auswahl über Präferenzfragen . . . 43
5.3.2 Der Alternativen-Wizard . . . 44
5.4 Spezifikation des Alternativen-Wizard . . . 45
5.4.1 Grafischer Entwurf des Alternativen-Wizard . . . 45
5.5 Implementierung . . . 46
5.5.1 Erweiterung des Regeleditors . . . 46
5.5.2 Erweiterung der ConfigurationBean . . . 47

6 Evaluation der Ergebnisse **51**
6.1 Interview mit Kundenbetreuer . . . 51
6.2 Usability-Testcase . . . 51
6.2.1 Das AttrakDiff2-Verfahren . . . 51
6.2.2 Die Testpersonen . . . 52
6.3 Auswertung des Usability-Tests . . . 52
6.3.1 Das Profil der Wortpaare . . . 53
6.3.2 Das Diagramm der Mittelwerte . . . 54
6.3.3 Das Ergebnis als Überblick in Portfoliodarstellung . . . 55

7 Zusammenfassung und Ausblick **56**
7.1 Zusammenfassung . . . 56
7.2 Ausblick . . . 56

A Regelwerk des PC-Konfigurator Beispiels **59**
A.1 Produktstruktur . . . 59
A.2 Regelwerk Laptop . . . 62
A.3 Regelwerk PC . . . 64
A.4 Regelwerk Tablet . . . 65

B Interview mit Projektleiter **66**

Abbildungsverzeichnis

Abbildungsverzeichnis

1 Systemarchitektur CAS Configurator Merlin 5
2 Datenmodell des CAS Configurator Merlin 8
3 Beispiel für zwei NAND-verknüpfte Operatorbäume 9
4 Beispiel der Aktiviert-Regel 10
5 Beispiel der Nicht-mit-Regel 10
6 Beispiel der Impliziert-Regel 11
7 Bisherige Oberfläche im Warenkorbstil 12
8 Hier liest man nach dem Gesetz der Nähe vertikal, obwohl die Wörter keinen Sinn ergeben 14
9 Es ist nicht ganz eindeutig, was zusammengehört, aber man kann die Wörter entziffern 14
10 Hier kann man die Wörter problemlos lesen 14
11 Wir nehmen die einzelnen Striche als zusammengehörende gerade Linie wahr 15
12 Hier sehen wir in den einzelnen Strichen eine Wellenlinie 15
13 Wir sehen zwei übereinanderliegende Quadrate, obwohl diese in der Realität nicht existieren 16
14 Zusammengehörigkeit von Elementen gleicher Form 16
15 Zusammengehörigkeit von Elementen gleicher Farbe 17
16 Zweistufiges Standardlayout 19
17 Gründe, die gegen den Online-Einkauf sprechen (vgl. [CM02]) 24
18 Vierstufige Hierarchie der Merkmalsstruktur 30
19 Grafischer Entwurf des intelligenten Formulars 33
20 Übersicht über die wichtigetsn Datentypen 34
21 Ausschnitt der Methode *buildProductCatalog* spannt den *Merkmalsbaum* auf 37
22 Die Methode *processBuildIn* als Ausschnitt der *ConfigurationBean* . . 39
23 Ablauf eines Konfigurationsschrittes am Beispiel des CAS Configurator Merlin 41
24 Visualisierung der Lösungsalternativen 42
25 Herkömmlicher Alternativendialog des CAS Configurator Merlin . . . 43
26 Wizard zur intelligent geführten Alternativenauswahl 45
27 Code zur Erweiterung des Regeleditors 46
28 Unterscheidungskraft der einzelnen Elemente 48
29 Code zur Erweiterung der *ConfigurationBean* zur Darstellung des Alternativen Wizard 50

30 Profil der Wortpaare 53
31 Diagramm der Mittelwerte 54
32 Übersicht über das Testergebnis in Portfoliodarstellung 55
33 Produktstruktur PC-Konfigurator - Teil 1 59
34 Produktstruktur PC-Konfigurator - Teil 2 60
35 Produktstruktur PC-Konfigurator - Teil 3 61
36 Regelwerk für das Vertriebsprodukt Laptop - Teil 1 62
37 Regelwerk für das Vertriebsprodukt Laptop - Teil 2 63
38 Regelwerk für das Vertriebsprodukt PC 64
39 Regelwerk für das Vertriebsprodukt Tablet 65

Tabellenverzeichnis

Tabellenverzeichnis

1 Rollen der Bedürfnisgruppen (vgl. [Dre08]) 18
2 Übersicht über alle für diese Arbeit implementierten Klassen 34

1 Einleitung

1.1 Motivation

In vielen Branchen steigt die Zahl von Wettbewerbern aus dem In- und Ausland stetig an. Um als Unternehmen bei einem derart hohen Marktdruck wettbewerbsfähig zu bleiben, ist es wichtig, den Angebotsprozess so einfach wie möglich zu halten, um auch bei komplexen Produkten eine schnelle Angebotserstellung zu gewährleisten. Außerdem steigt die Anzahl möglicher und angebotener Produktvarianten zunehmend an und den Kunden wird ein immer größerer Spielraum bei der Ausgestaltung individueller Produkte geboten. Man spricht hierbei vom Prinzip des *Mass Customization*.

> "Der Ausdruck Mass Customization ist ein Oxymoron, das die an sich gegensätzlichen Begriffe 'Mass Production' und 'Customization' verbindet." [Pil06]

Man spricht auch von der individuellen Massenfertigung, welche als hybride Strategie ein gleichzeitiges Anstreben von Kostenführerschaft und Produktdifferenzierung bedeutet.

Mit Hilfe eines webbasierten Produktkonfigurators hat der Kunde die Möglichkeit, sich sein individuelles Produkt online auf Basis von standardisierten Bauteilen zusammenzustellen. Dabei reicht die Bandbreite von *Match-to-order*[1]-Angeboten, über *Assemble-to-order*[2], bis hin zu *Make-to-order*[3]- und *Develop-to-order*[4]-Alternativen. Durch die Einbindung des Konfiguratorsystems in die Prozesskette eines Unternehmens bildet es im Idealfall die Schnittstelle zwischen *ERP-System*[5] und *CRM-System*[6]. Der Produktkonfigurator bezieht also zum Beispiel Daten über mögliche Bauteile aus einem ERP-System, kombiniert diese nach individuellen Kundenwünschen und gibt ein fertiges Angebot an das CRM-System weiter.

Häufig spiegelt die Benutzeroberfläche bei webbasierten Produktkonfiguratoren ein einfaches Warenkorbprinzip wieder. Dieses bietet lediglich die Möglichkeit, zwischen dem Einbau eines Bauteils oder dessen Weglassen zu entscheiden. Bei den

[1] Hier kann sich der Kunde aus bestehenden Standardprodukten sein individuelles Produkt zusammenstellen

[2] Beim *Assemble-to-order* handelt es sich um ein Verfahren zur auftragsbezogenen Endmontage aus Bauteilen vom Lager

[3] Ermöglicht auftragsbezogene Komponentenfertigung

[4] Individualisierung der Produktentwicklung

[5] Enterprise-Resource-Planning-System: System zur IT-gestützten Bearbeitung betriebswirtschaftlicher Aufgaben aus den verschiedensten Bereichen eines Unternehmens (z.B. Finanzwesen, Produktion, Logistik, Personalwesen)[HG07]

[6] Customer-Relationship-Management-System: System zur Sammlung und Auswertung von Kundendaten sowie zur Automatisierung kundenbezogener Prozesse[HHW11]

immer komplexer werdenden Produkten reicht es oft nicht mehr aus, die Konfiguration mit Hilfe eines solchen Schemas darzustellen.

Aufgrund des booleschen Regelwerks eines Produktkonfigurators kann der Konfiguratorkern bei entsprechendem Umbauwunsch in eine Situation kommen, bei der es keinen eindeutigen Weg hin zu einem gültigen Produkt gibt. Der Konfiguratorkern des CAS Configurator Merlin berechnet dann Alternativen, in denen alle Umbauten hin zu einem gültigen Produkt enthalten sind. Auch hier mangelt es oft an zufriedenstellenden Lösungen zur Darstellung dieser alternativen Konfigurationsmöglichkeiten auf der Benutzeroberfläche.

Durch einen Produktkonfigurator, der trotz hoher Komplexität der Produkte den Kunden benutzerfreundlich und intelligent durch die Konfiguration führt und aktiv bei der Auswahl von alternativen Konfigurationen unterstützt, könnten sowohl die Kundenzufriedenheit als auch die Kundenbindung erhöht und gleichzeitig die Anzahl kundeninitiierter Prozessabbrüche bei Präferenzlosigkeit gesenkt werden. Dadurch können sich Unternehmen einen deutlichen Wettbewerbsvorteil verschaffen.

1.2 Ziel der Arbeit

In dieser Arbeit soll das Konzept einer intelligenten grafischen Oberfläche für die Produktkonfiguration entwickelt und umgesetzt werden, welches vor allem die Problematik der Darstellung einer Vielzahl alternativer Konfigurationen löst. Dazu soll die Möglichkeit geschaffen werden, für einzelne Bauteile im Regelwerk einen Grad der Priorität zu definieren. Abhängig von dieser Priorität soll der Konfigurator auf verschiedene Art und Weise, zum Beispiel durch intelligente Präferenzfragen oder durch gezieltes Fragen nach der gewünschten Ausprägung, den Lösungsraum eingrenzen und somit die Anzahl der alternativen Konfigurationen auf ein Minimum beschränken. Außerdem soll eine interne Kostenfunktion den Aufwand des jeweiligen Umbaus evaluieren und die übrigen Alternativen nach diesem Aufwand sortiert auflisten. Das Entwickelte System soll prototypisch implementiert und in den existierenden Produktkonfigurator CAS Configurator Merlin integriert werden.

1.3 Aufbau der Arbeit

Der erste Teil beschreibt den CAS Configurator Merlin als Grundlage dieser Arbeit (siehe Kapitel 2). Insbesondere wird hier auf dessen Architektur und die bisherige Oberfläche im Stil eines Warenkorbprinzips eingegangen. Danach werden in Kapitel 3 *Usability bei Produktkonfiguratoren* die Bedürfnisse der Benutzer bei der Verwendung eines Konfiguratorsystems analysiert, Gründe für benutzerinitiierte Prozessabbrüche bei der Konfiguration eruiert und spezielle *Interaction Design Pattern* eingeführt.

Der Hauptteil dieser Arbeit besteht aus den beiden Kapiteln *Intelligentes Formular zur Produktkonfiguration* (siehe Kapitel 4) und *Dialog zur Darstellung alternativer Konfigurationen* (siehe Kapitel 5). In diesen Kapiteln werden Konzepte und Entwürfe einer optimalen Benutzeroberfläche für die Produktkonfiguration erläutert und Implementierungsbeispiele angeführt, wobei auch detailliert auf die verwendeten Technologien eingegangen wird.

In Abschnitt 6 *Evaluation der Ergebnisse* wird die Qualität der im Rahmen dieser Arbeit erstellten Konzepte bewertet und in Kapitel 7 *Zusammenfassung und Ausblick* werden die Ergebnisse dieser Arbeit zusammengefasst und Möglichkeiten zur Verbesserung der gefundenen Lösung aufgezeigt.

2 CAS Configurator Merlin

2.1 Architektur

Die Architektur des CAS Configurator Merlin setzt sich aus drei in *Java* implementierten Modulen zusammen, die mit *SOAP-Webservices* untereinander kommunizieren (siehe Abbildung 1):

- Regeleditor
- Konfiguratorkern (*XPS Merlin*)
- Konfiguratoroberfläche

Grundlage für eine Konfiguratorlösung mit dem CAS Configurator Merlin ist immer der Konfiguratorkern. Die einzelnen Module - bis auf den Konfiguratorkern *XPS Merlin* - sind beliebig austauschbar und bei Bedarf auch für jeden einzelnen Kunden individualisierbar. Die Grundidee ist hierbei, ein Basisprodukt zu verkaufen, das für kleine und mittelständische Unternehmen meist ohne größere Anpassungen verwendet werden kann. Um dieses Basisprodukt trotzdem auf die speziellen Anforderungen des Kunden zuschneiden zu können, können zum Beispiel folgende Anpassungen und Serviceleistungen vorgenommen werden:

- Anpassung der Oberfläche
- Implementierung individueller Schnittstellen
- Erstellung einer Modellsprache

Die Produktmodellierung findet auf der Pflegeoberfläche des Regeleditors statt, die auf *Eclipse RCP*[7] basiert.

Der webbasierte *CAS Configurator Merlin* hat eine in JSF implementierte Konfiguratoroberfläche.

[7] Eclipse Rich Client Platform zur Entwicklung von Anwendungen basierend auf dem Eclipse Framework

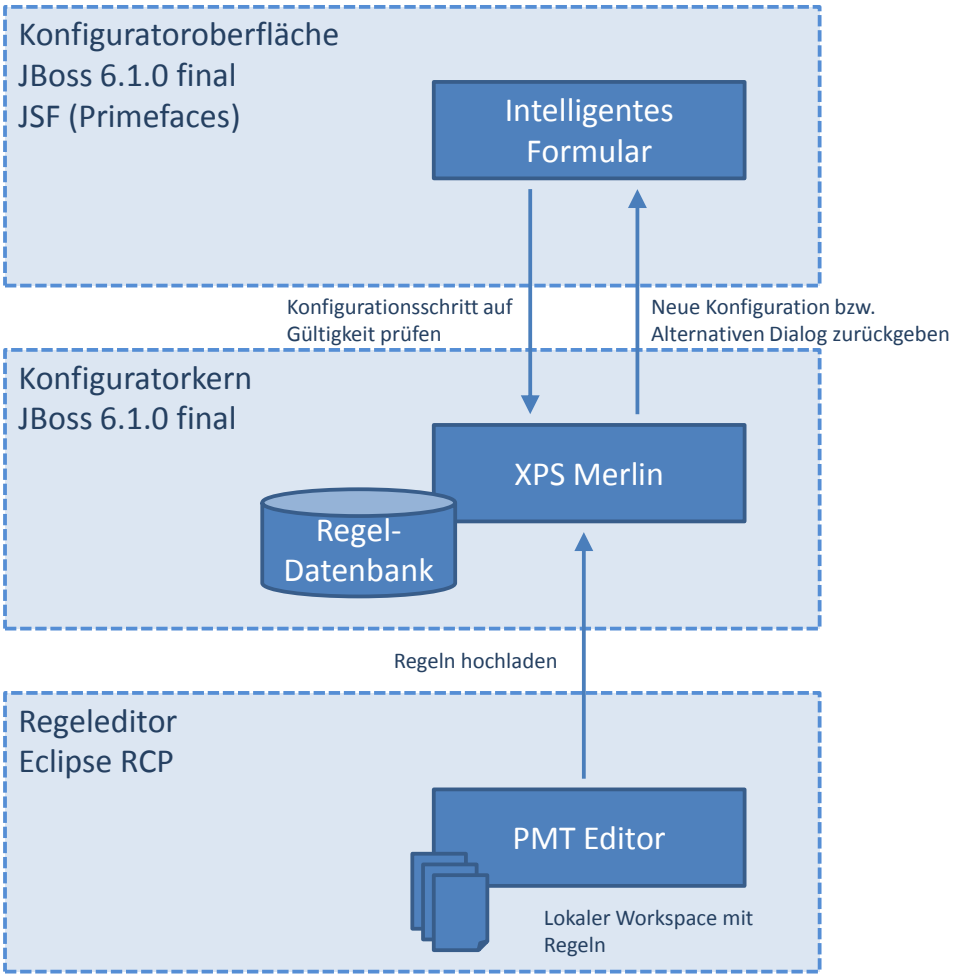


Abbildung 1: Systemarchitektur CAS Configurator Merlin

2.2 Datenmodell

Der Konfiguratorkern greift auf ein festes Datenmodell zurück, was diesen unabhängig vom modellierten Produkt macht. Dadurch kann der Konfiguratorkern flexibel für jedes beliebige Produkt verwendet werden, da das zu konfigurierende Produkt für diesen nicht relevant ist, es wird nur das zugrunde liegende Datenmodell betrachtet (siehe Abbildung 2).

Um das grundsätzliche Datenmodell greifbar zu erklären, wird im Folgenden das Beispiel eines PC-Konfigurators verwendet und die einzelnen Bestandteile des Datenmodells an diesem Beispiel erläutert. Das komplette Regelwerk und die Pro-

duktstruktur des PC-Konfigurator Beispiels ist im Anhang in Abschnitt A zu finden.

2.2.1 Element

Ein *Element* ist ein Bauteil bzw. eine Funktionseinheit, die in ein Produkt eingebaut werden kann. Alle diese *Elemente* bilden die Menge der möglichen verbaubaren *Elemente* $E = \{e_1, e_2, e_3, ..., e_n\}$.

Für einen PC-Konfigurator setzt sich die Menge der verbaubaren *Elemente* zum Beispiel folgendermaßen zusammen (ausschnittsweise):

Beispiel 2.1. $E_{PC-Konfigurator} := \{prozessor_1, prozessor_2, festplatte_1, festplatte_2, festplatte_3\}$

2.2.2 Regel

Eine *Regel* r ist eine boolesche Formel über den *Elementen* aus der Menge der möglichen verbaubaren *Elemente* E als deren Variablen. Häufig verwendete *Regeln* sind *in-Verbindung-mit-* und *nicht-in-Verbindung-mit-Regeln* (mehr dazu in Kapitel 2.3 Konfigurationsregeln).

Eine *Regel* setzt sich also aus mehreren *Elementen* zusammen, die mit logischen Operatoren miteinander verknüpft sind.

Beispiel 2.2. $r_1 := prozessor_1 \rightarrow (festplatte_1 \vee festplatte_2)$

2.2.3 Regelwelt

Die Menge aller *Regeln* über E wird als *Regelwelt* R bezeichnet.

Beispiel 2.3. $r_1 := prozessor_1 \vee prozessor_2$
$r_2 := (festplatte_1 \not\leftrightarrow festplatte_2) \wedge (festplatte_2 \not\leftrightarrow festplatte_3) \wedge (festplatte_3 \not\leftrightarrow festplatte_1)$
$r_3 := prozessor_1 \rightarrow (festplatte_1 \vee festplatte_2)$
$R := \{r_1, r_2, r_3\}$

2.2.4 Produktplan

Der *Produktplan*[8] *pp* stellt den *Bauplan* für die zu konfigurierenden Produkte dar und besteht aus folgenden Bestandteilen:

- Menge möglicher verbaubarer *Elemente* E
- *Regelwelt* R

[8] Im weiteren Verlauf auch *ProductPlan* genannt

Beispiel 2.4. $E_{PC-Konfigurator} := \{prozessor_1, prozessor_2, festplatte_1, festplatte_2, festplatte_3\}$
$r_1 := prozessor_1 \vee prozessor_2$
$r_2 := (festplatte_1 \not\Leftrightarrow festplatte_2) \wedge (festplatte_2 \not\Leftrightarrow festplatte_3) \wedge (festplatte_3 \not\Leftrightarrow festplatte_1)$
$r_3 := prozessor_1 \rightarrow (festplatte_1 \vee festplatte_2)$
Produktplan $pp_{PC} = (E, R) = (E, \{r_1, r_2, r_3\})$

2.2.5 Produkt

Ein *Produkt*[9] p zu einem *Produktplan* pp ist eine Teilmenge von E.

Beispiel 2.5. $p_1 := \{prozessor_1, festplatte_1\}$
$p_2 := \{prozessor_1, festplatte_2\}$
$p_3 := \{prozessor_1, prozessor_2\}$
$p_4 := \{\}$

Die Gültigkeit der im Beispiel angeführten *Produkte* ist nicht geklärt, diese stellen nur mögliche *Produkte* dar. Informal beschrieben liegt ein *gültiges Produkt* p zu einem *Produktplan* $pp = (E, R)$ vor, wenn alle *Regeln* aus R gleichzeitig erfüllt sind.

2.2.6 Gültiges Produkt

Ein *Produkt* p zu einem *Produktplan* $pp = (E, R)$ heißt gültig für pp, wenn gilt:

- β_p ist ein Modell für die Formel R^+
- mit der Gesamtformel R^+ definiert als $\wedge_{r \in R} r$
- mit der Variablenbelegung β_p wie folgt definiert:
  - für alle $e \in p$ ist definiert: $\beta_p(e) := 1$
  - für alle $e \in E \setminus p$ ist definiert: $\beta_p(e) := 0$

Nichtformal kann man also die Vorgehensweise zur Bestimmung der Gültigkeit folgendermaßen beschreiben. Man wertet alle in p eingebauten *Elemente* als 1, alle in p nicht-eingebauten *Elemente* wertet man hingegen als 0. Damit kann man den Wahrheitswert aller *Regeln* bestimmen. Ist das Ergebnis der logischen Auswertung aller *Regeln* 1, ist das *Produkt* p gültig für pp, andernfalls nicht.

[9] Im weiteren Verlauf auch *Product* genannt

2.2.7 Das Datenmodell des CAS Configurator Merlin

Jedes *Product* basiert auf einem *ProductPlan*. Ein solcher *ProductPlan* besteht aus *Slots*, die angeben, was in das *Produkt* eingebaut werden darf. Ein *Element* ist ein Bauteil, das ein *SlotFilling* füllt und dieses wiederum füllt den *Slot*. Ein *Element* kann nur in einen *Slot* eingebaut werden, wenn beide über das gleiche *Feature* verfügen. Man kann den *Slots* zusätzlich noch Kardinalitäten zuweisen und damit bestimmen, wie oft ein *Element* mit bestimmtem *Feature* in den passenden *Slot* eingebaut werden kann bzw. ob ein Einbau gezwungenermaßen erforderlich oder fakultativ ist.

Die Summe aller gefüllten *Slots*, also die *SlotFillings*, stellt die aktuelle Konfiguration bzw. das *Produkt* dar.

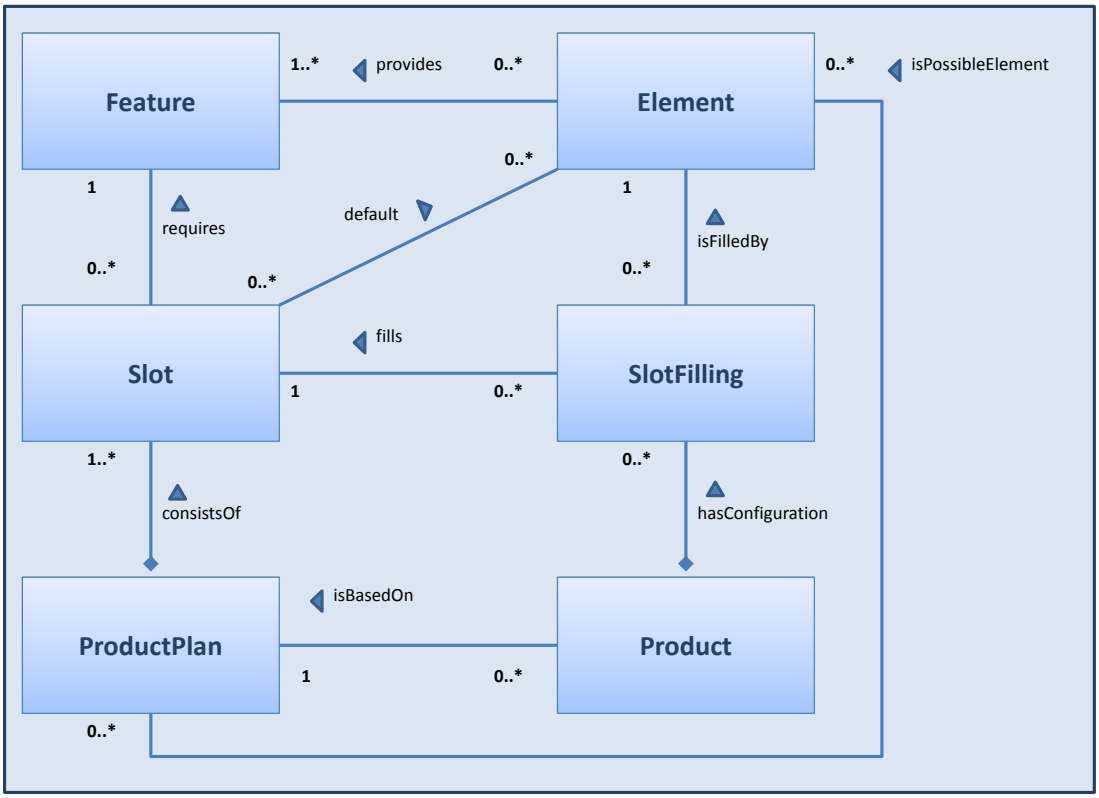


Abbildung 2: Datenmodell des CAS Configurator Merlin

2.3 Konfigurationsregeln

Das Regelwerk für den CAS Configurator Merlin basiert auf boolescher Logik. Dabei sind die einzelnen Elemente entweder eingebaut oder nicht, was eine binäre Entscheidung darstellt. Logische Aussagen modellieren Einschränkun-gen über die verbaubaren Elemente, so genannte *Constraints*, weshalb man in diesem Fall auch von einem *constraintbasierten* Konfigurator spricht. Im weiteren Verlauf dieser Arbeit werden diese *Constraints* als *Regeln* bezeichnet. Sehr häufig angewandte *Regeln* sind die *nur-in-Verbindung-mit-Regel*, also die *Implikation*, und die *nicht-in-Verbindung-mit-Regel*, welche ein *NAND-Gatter* darstellt. Zusätzlich finden auch andere Gatter-Typen, wie *AND*, *OR* und *XOR* Verwendung, um boolesche Operatorbäume rechts und links der *nur-in-Verbindung-mit-* bzw. *nicht-in-Verbindung-mit*-Regel aufzuspannen (siehe Abbildung 3).

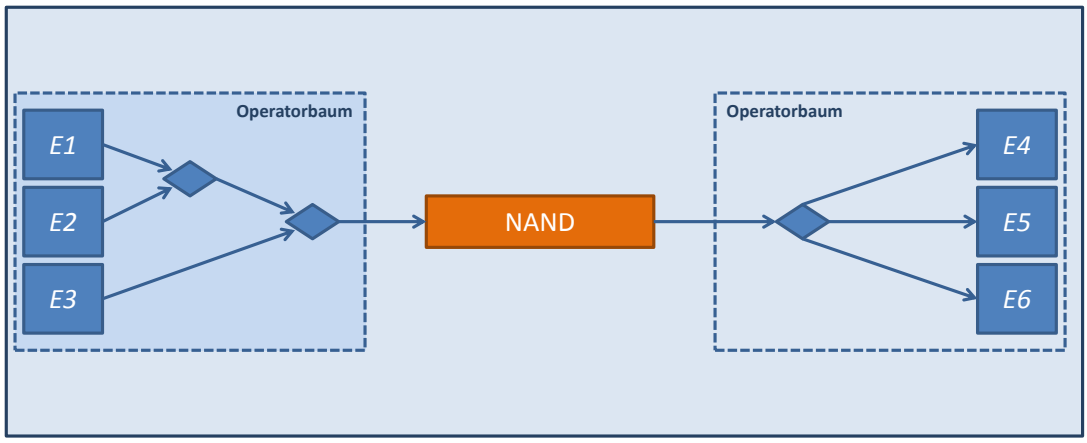


Abbildung 3: Beispiel für zwei NAND-verknüpfte Operatorbäume

Ein Produkt ist genau dann gültig, wenn es alle Regeln des Regelwerkes gleichzeitig erfüllt. Ist nur eine einzige Regel nicht erfüllt, so ist das Produkt ungültig.

Im Folgenden wird auf für das Beispiel des PC-Konfogurators relevante Regeln und Operatoren eingegangen.

2.3.1 Aktiviert-Regel

Laut der *Aktiviert-Regel* aktiviert ein bestimmtes Element ein anderes und dessen Kindknoten. So können ganze Bereiche der Produktstruktur auf einmal aktiviert werden. Im *CAS Configurator Merlin* können nur aktivierte Elemente in die Konfiguration eingebaut werden. Zum Beispiel aktiviert in Abbildung 4 der Einbau eines

Laptops das Element Prozessor, wodurch im Anschluss ein Prozessortyp ausgewählt werden kann.

Abbildung 4: Beispiel der Aktiviert-Regel

2.3.2 Nicht-mit-Regel

Die *Nicht-mit-Regel* besagt, dass zwei Elemente in keinem Fall gleichzeitig Bestandteil eines Produktes bzw. einer Konfiguration sein können und stellt deshalb eine symmetrische Konfigurationsregel dar. Diese Regel entspricht der negierten Konjunktion, also dem *NAND-Gatter* und wird formal mit dem $\not\Leftrightarrow$-Symbol dargestellt. Im in Abbildung 5 angeführten Beispiel kann ein Laptop niemals ein Display der Größe 10 Zoll erhalten und ein Display der Größe 10 Zoll kann wiederum niemals in ein Laptop eingebaut werden.

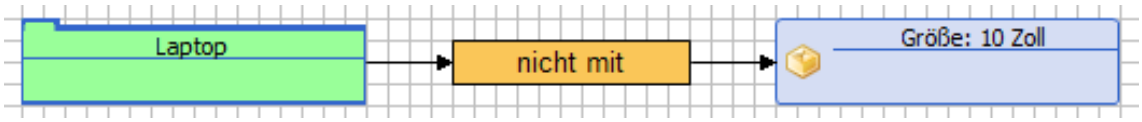


Abbildung 5: Beispiel der Nicht-mit-Regel

2.3.3 Impliziert-Regel

Die *Impliziert-Regel* ist indentisch zur *nur-in-Verbindung-mit-Regel*, also zur booleschen Implikation und bedeutet, dass das Element auf der rechten Seite der Implikation zwingend eingebaut werden muss, wenn das Element links eingebaut wird. Andersrum muss aber das linke Element nicht zwingend Bestandteil des Produktes sein, wenn das rechte Element eingebaut wird. Man spricht deshalb von einer asymmetrischen Konfigurationsregel, die formal mit dem $\Rightarrow$-Symbol dargestellt wird. In Abbildung 6 muss die Größe der zweiten Festplatte gewählt werden, sofern eine solche eingebaut wird.

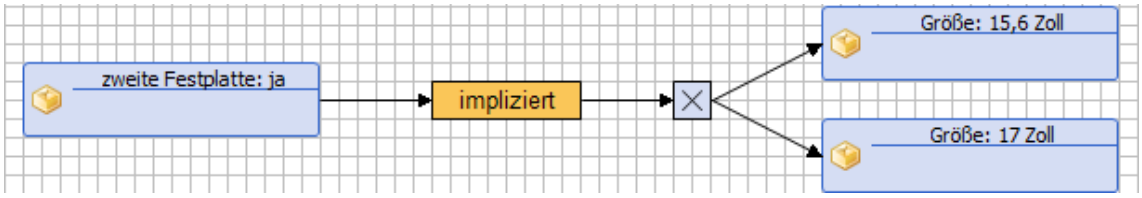


Abbildung 6: Beispiel der Impliziert-Regel

2.3.4 Operatorbäume

Wie man in Abblidung 6 erkennen kann, muss ein Operand einer Konfigurationsregel nicht unbedingt aus einem einzelnen Element bestehen, sondern kann auch einen ganzen Operatorbaum beinhalten. Um solche Operatorbäume zu ermöglichen, benötigt man boolesche Operatoren wie *AND*, *OR* und *XOR*, um Elemente logisch miteinander zu verknüpfen.

AND Der *AND-Operator* bedeutet, dass die verknüpften Elemente gleichzeitig eingebaut werden müssen. Im Regeleditor wird dafür das Symbol $\wedge$ verwendet.

OR Der *OR-Operator* bedeutet, dass mindestens eines der verknüpften Elemente eingebaut werden muss, aber auch mehrere gleichzeitig möglich sind. Dieser Sachverhalt wird wird im Regeleditor durch das $\vee$-Symbol ausgedrückt.

XOR Der *XOR-Operator* bedeutet, dass genau eines der verknüpften Elemente eingebaut werden muss, nicht mehr und nicht weniger. Bei der grafischen Produktmodellierung im *Regeleditor* wird der *XOR-Operator* mit dem x-Symbol dargestellt.

2.4 Bisherige Benutzeroberfläche

Die bisher verwendete Benutzeroberfläche ist im Warenkorbstil implementiert (siehe Abbildung 7). Im linken Bildschirmbereich werden alle möglichen einbaubaren Elemente - der sogenannte *Katalog* - angezeigt und diese können über das *Plus-Symbol* zur Konfiguration hinzugefügt werden. Rechts daneben wird die aktuelle Konfiguration, also alle eingebauten Elemente, angezeigt. Um eine Ausprägung zu ändern, muss das aktuell eingebaute, zu verändernde Element ausgebaut und die neue gewünschte Ausprägung wieder eingebaut werden.

Für die Darstellung komplexer Produkte ist diese Oberfläche nicht mehr geeignet, da der *Katalog* durch eine Vielzahl von Elementen und Ausprägungen schlicht und ergreifend überladen ist. Auch der Bereich zur Darstellung der aktuellen Konfiguration wird schnell unübersichtlich und eine rasche Umkonfiguration ist oft nicht

möglich. Abhilfe dafür sollen die im Rahmen dieser Arbeit entwickelten Konzepte zur Darstellung komplexer Produktkonfigurationen schaffen.

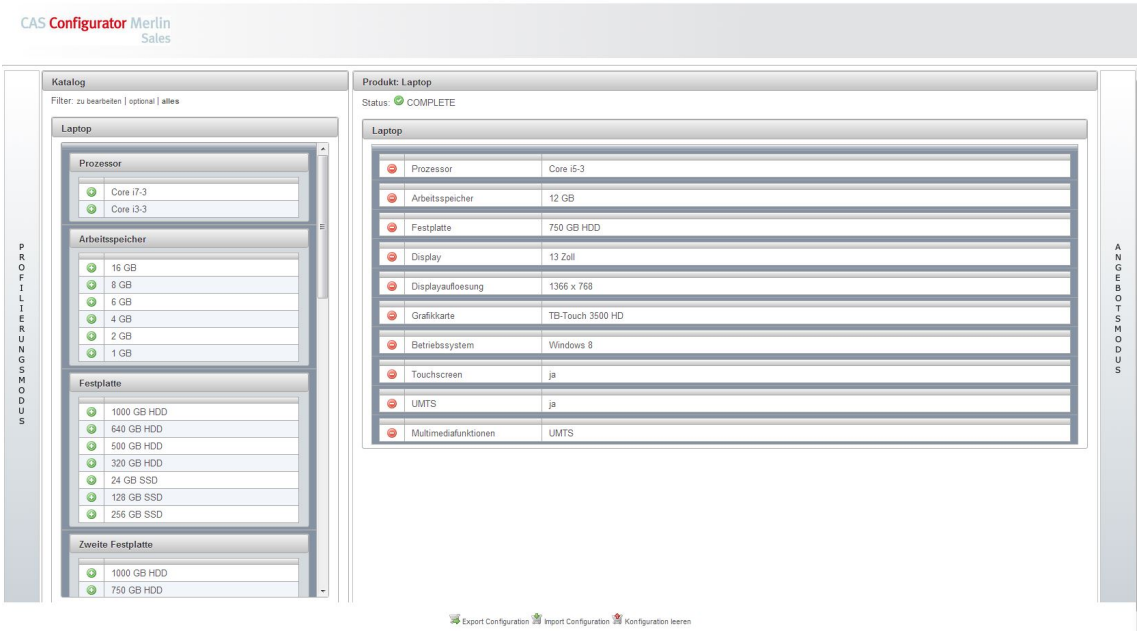


Abbildung 7: Bisherige Oberfläche im Warenkorbstil

3 Usability bei Produktkonfiguratoren

3.1 Allgemeine Gestaltungsgesetze

Bei der Gestaltung grafischer Werke, worunter auch die Gestaltung von Benutzeroberflächen zu verstehen ist, gibt es einige grundlegende Wahrnehmungsgesetze, die beachtet werden sollten. Zu den Gestaltungsgesetzen auf die im Folgenden näher eingegangen wird zählen (vgl. [Wä10]):

- Das Gesetz der Nähe
- Das Gesetz der Kontinuität
- Das Gesetz der Geschlossenheit
- Das Gesetz der Gleichheit

3.1.1 Das Gesetz der Nähe

Dieses Wahrnehmungsgesetz ist eines der stärksten Gesetze und sollte bei jeder Gestaltungsaufgabe berücksichtigt werden.

> "Das Gesetz der Nähe besagt, dass Elemente, die nahe beieinander stehen, als zusammengehörend wahrgenommen werden."[Wä10]

In Abbildung 8 lesen wir *TRO*, *OOR* und *RTT*. Diese Wörter ergeben zwar keinen Sinn, aber unser Gehirn verarbeitet diese Informationen gemäß dem Gesetz der Nähe. Alternativ könnte man auch die farblich gleichen Buchstaben als zusammengehörend empfinden. "Das Gesetz der Nähe wirkt wesentlich stärker als formale oder farbliche Übereinstimmung"[Wä10], weshalb man die farblich gleichen Buchstaben zunächst gar nicht bemerkt. In Abbildung 9 fällt es schon leichter, die Wörter *TOR*, *ROT* und *ORT* zu erkennen, wenn auch nicht ganz eindeutig. Erst in Abbildung 10 kann man die Wörter durch die optimale Formatierung nach dem Gesetz der Nähe problemlos entziffern.

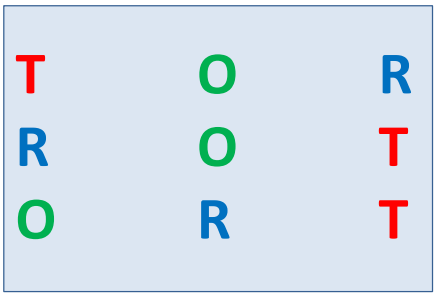


Abbildung 8: Hier liest man nach dem Gesetz der Nähe vertikal, obwohl die Wörter keinen Sinn ergeben

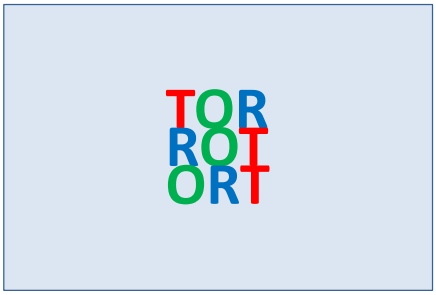


Abbildung 9: Es ist nicht ganz eindeutig, was zusammengehört, aber man kann die Wörter entziffern

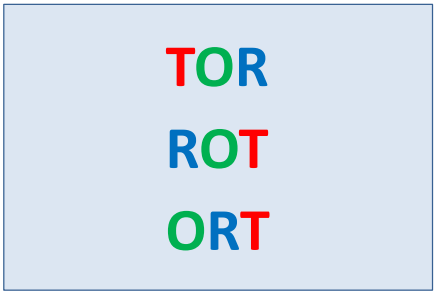


Abbildung 10: Hier kann man die Wörter problemlos lesen

3.1.2 Das Gesetz der Kontinuität

> "Elemente, die entlang einer geraden oder gebogenen Linie angeordnet sind, werden eher als zusammengehörend wahrgenommen als solche Elemente, die keiner gemeinsamen Linie folgen." [Wä10]

In Abbildung 11 bilden die einzelnen Striche eine gerade Linie, wohingegen sie in Abbildung 12 eine Wellenlinie formen. In beiden Fällen empfinden wir die einzelnen Striche als zusammengehörend. Um diesen Effekt zu erzielen ist es wichtig, die einzelnen Elemente an einer deutlich erkennbaren Linie anzuordnen und einen festen Rhythmus in den Abständen der Elemente und deren Größenverhältnissen einzuhalten.

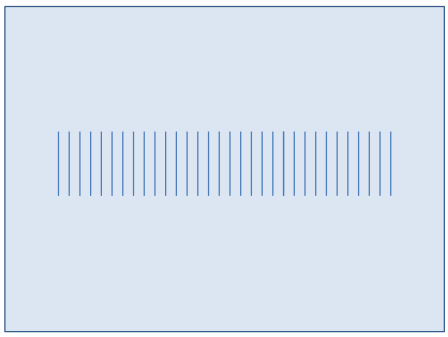


Abbildung 11: Wir nehmen die einzelnen Striche als zusammengehörende gerade Linie wahr

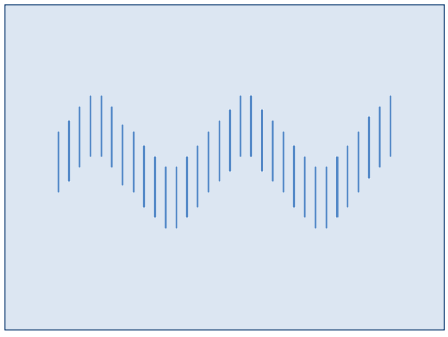


Abbildung 12: Hier sehen wir in den einzelnen Strichen eine Wellenlinie

3.1.3 Das Gesetz der Geschlossenheit

> "Die Wahrnehmung des Menschen ist darauf ausgerichtet, in allem, was wir sehen, Zusammenhänge zu entdecken. Aufgrund der kräftigen Aus-

prägung dieser Konditionierung erkennen wir sogar dort zusammenhängende Figuren, wo gar keine sind." [Wä10]

In Abbildung 13 sehen wir zum Beispiel zwei übereinanderliegende Quadrate, das hintere um 45° gedreht. Diese beiden Quadrate existieren in der Realität nicht, sondern entstehen erst in unserem Gehirn. Wir nehmen die vier 3/4-Kreise und die vier Dreiecke so wahr, dass nach dem Gesetz der Geschlossenheit zwei Quadrate entstehen.

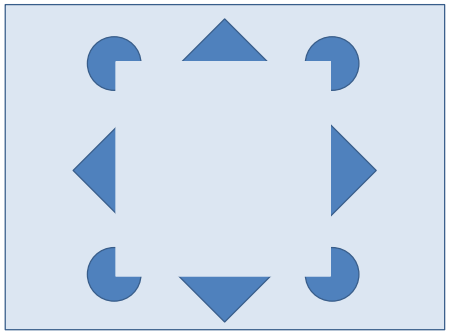


Abbildung 13: Wir sehen zwei übereinanderliegende Quadrate, obwohl diese in der Realität nicht existieren

3.1.4 Das Gesetz der Gleichheit

Objekte mit gleicher Form, Größe oder Farbe werden als zusammengehörend empfunden. In Abbildung 14 fallen sofort die vier kleinen Vierecke ins Auge, wobei in Abbildung 15 die vier orangen Elemente eine Einheit bilden. Dadurch kann man erkennen, dass farbliche Gleichheit vorrangig vor gleicher Form bzw. Größe wahrgenommen wird, da hier die vier kleinen Vierecke nicht primär als zusammengehörend empfunden werden.

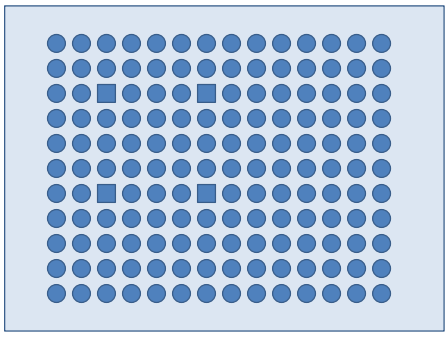


Abbildung 14: Zusammengehörigkeit von Elementen gleicher Form

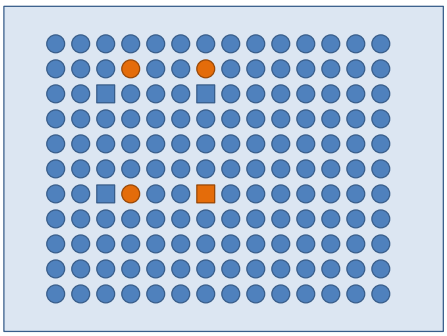


Abbildung 15: Zusammengehörigkeit von Elementen gleicher Farbe

3.2 Usability - Was ist das?

Verbergen sich hinter der bloßen Übersetzung des Begriffs *Usability* nur Worte wie *Nutzbarkeit*, *Brauchbarkeit* oder *Benutzerfreundlichkeit*, steckt doch eine noch viel weitreichendere Bedeutung dahinter.

> "Usability is the quality of a system that makes it easy to learn, easy to use, easy to remember, error tolerant, and subjectively pleasing".[PR02]

Speziell im Bereich der Produktkonfiguration spielt also primär die Bedienbarkeit und Selbsterklärung eines Systems eine entscheidende Rolle. Das beste Produkt und die innovativsten Funktionen sind wenig nützlich, wenn der Produktkonfigurator bereits bei der Gewährleistung einer intuitiven Bedienung und Navigation, sowie beim Erreichen eines Ziels und dem damit verbundenen Erfolgserlebnis scheitert.

Um feste Regeln für die Bedienbarkeit eines Konfigurationssystems definieren zu können, müssen vorher die Ansprüche der Kunden an ein solches System analysiert werden.

3.3 Kundenbedürfnisse

Bei der Gestaltung einer benutzerfreundlichen Oberfläche für einen webbasierten Produktkonfigurator müssen die Bedürfnisse des Kunden sowohl in der Rolle als Konsument, als auch in der Rolle als Anwender berücksichtigt werden (vgl. [Dre08]). Zu beachten ist, dass es neben Endkundenkonfiguratoren auch so genannte Vertriebskonfiguratoren gibt, die von einem Vertriebsmitarbeiter bedient werden. Hierbei ist nicht mehr der Kunde der Anwender, sondern eben der Vertriebsmitarbeiter, welcher eventuell andere Bedürnisse hat als der Kunde. Aus der Menge der Anforderungen von Anwendern und Konsumenten lassen sich eine ganze Reihe von Kundenbedürfnissen festhalten (vgl. [PR02]).

Kundenbedürfnisse	
Anwender	Konsument

Tabelle 1: Rollen der Bedürfnisgruppen (vgl. [Dre08])

3.3.1 Bedienbarkeit und Selbsterklärung

Diese beiden Bedürfnisse zählen mit zu den bedeutendsten im Bereich der Produktkonfiguration.

> "Ein Konfigurator sollte den Benutzer ständig informieren, welchen Konfigurationsschritt er gerade macht, welche er schon gemacht hat und welche er noch machen muss."[PR02]

Zentrale Aufgabe des Konfigurationssystems im Hinblick auf die oftmals sehr komplexen Produkte ist hierbei, den User durch die Konfiguration zu führen und die Lernschwelle möglichst niedrig zu halten. Weder die Bedienung der Anwendung selbst, noch die Komplexität der Produkte dürfen den Benutzer überfordern. Aus der Navigation und dem Layout der Seite, sollen sich dem User stets folgende Fragen erschließen (vgl. [PR02]):

- Wo bin ich?
- Wo war ich?
- Wohin kann ich gehen?

Bei der Gestaltung einer Webseite wird häufig auf ein zweistufiges Standardlayout (siehe Abbildung 16) zurückgegriffen.

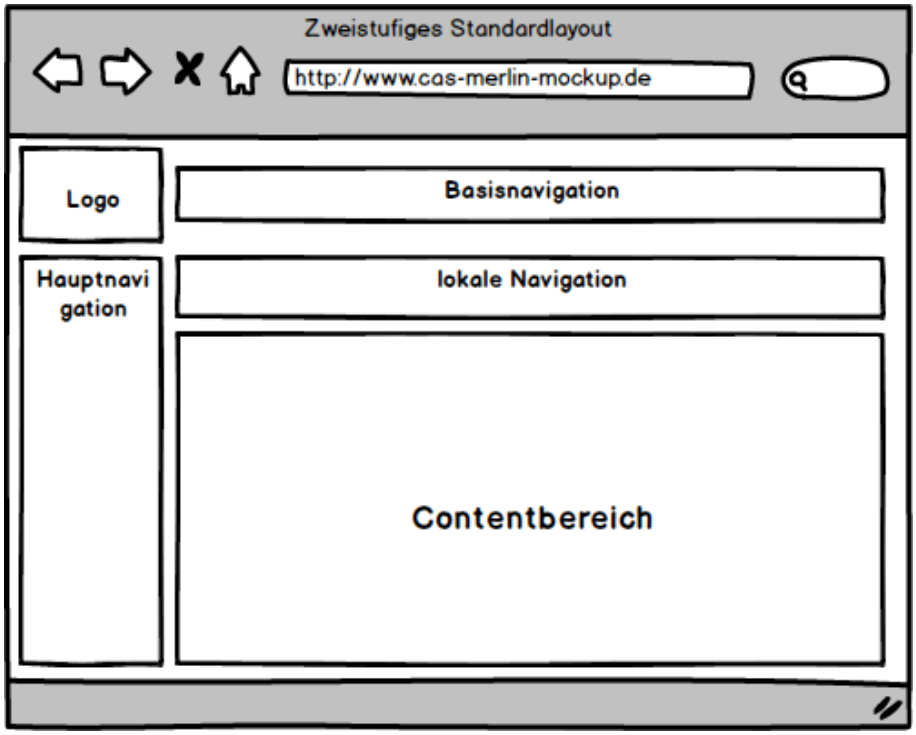


Abbildung 16: Zweistufiges Standardlayout

Hauptnavigation Die *Hauptnavigation* beinhaltet in der Regel "allgemeine Links zur Seite wie die Startseite, häufige Fragen (FAQ), und Informationen zum Produkt und den Geschäftsbedingungen"[PR02].

Basisnavigation Der Konfigurator selbst, der Warenkorb oder Links zu persönlichen Einstellungen bzw. zum Benutzerprofil finden in der *Basisnavigation* ihren Platz.

Lokale Navigation Die Steuerelemente des Konfigurationssystems befinden sich in der *lokalen Navigation* direkt über dem *Contentbereich*, also in unmittelbarer Nähe zum eigentlichen Konfigurationsformular.

3.3.2 Laden der Anwendung

Wie auch bei jeder herkömmlichen Webseite stellen die Ladezeiten der Anwendung einen entscheidenden Faktor für die Benutzerfreundlichkeit dar. Lange Ladezeiten sollten unbedingt vermieden werden.

3.3.3 Hilfestellungen

Hilfestellungen dienen dazu, Vertrauen beim Benutzer zu wecken und Prozessabbrüche zu vermeiden, die aus Unsicherheit bzw. Unwissenheit und Überforderung resultieren.

Im folgenden werden zwei unterschiedliche Konzepte der Hilfestellung vorgestellt:

- Häufig gestellte Fragen - FAQ's
- Kontextsensitive Hilfe

Häufig gestellte Fragen - FAQ's Die einfachste Form der Hilfestellung für den Benutzer ist die Auflistung von *häufig gestellten Fragen*, sogenannten *FAQ*[10]. Der Vorteil von solchen *FAQ*'s liegt in deren Einfachheit und der Vertrautheit der Benutzer mit deren Verwendung. Meist stellen die *häufig gestellten Fragen* eine Liste von Fragen mit den jeweiligen Antworten dar, die in bestimmte Kategorien bzw. nach bestimmten Oberbegriffen gruppiert sind. Dass der Benutzer jedoch aktiv nach Antworten für sein Problem suchen muss, stellt einen großen Nachteil von *FAQ*'s dar. Vielen Anwendern fällt es schwer, ihr Problem detailliert und korrekt zu beschreiben, außerdem ist die Liste der *häufig gestellten Fragen* oftmals unvollständig.

Kontextsensitive Hilfe Ein anderes Konzept, das den Anwender aktiv durch die Konfiguration führt und gezielte Hilfestellungen anbietet, ist das der *kontextsensitiven Hilfe* (vgl. [Thi08]). Man kann *kontextsensitive Hilfe* auf zwei unterschiedlichen Ebenen einsetzen, auf Elementebene und auf Dialogebene.

Auf Elementebene bietet die *kontextsensitive Hilfe* Hilfestellungen und Informationen bezüglich eines bestimmten einzelnen Steuerelements, zum Beispiel ein Formularfeld, ein Navigationselement, ein bestimmtes Symbol oder eine Schalftfläche. Ein Beispiel hierfür ist ein Tooltip, der erscheint, wenn der User mit der Maus über ein bestimmtes Steuerelement fährt, das mit einer Hilfestellung behaftet ist.

Sollen ganze Dialoge, Fenster oder Register als Ganzes, einschließlich der enthaltenen Elemente, beschrieben werden, so kommt die *kontextsensitive Hilfe* auf Dialogebene zum Einsatz.

> "Sie beschreibt sowohl die einzelnen Elemente als auch deren Zusammenhänge und Abhängigkeiten".[Thi08]

3.3.4 Feedback

Ein weiteres Kundenbedürfnis, das man bei der Konzeption einer Benutzer-oberfläche für einen webbasierten Produktkonfigurator nicht vernachlässigen sollte, ist der

[10] engl.: frequently asked questions

Wunsch nach Feedback. Der Anwender möchte stets Rückmeldung für seine getätigten Aktionen erhalten, zum Beispiel in der Form von Erfolgs- bzw. Fehlernachrichten. Außerdem sollten Ladezeiten durch typische Ladesymbole (Sanduhr, sich drehender Ring, usw.) visualisiert werden, damit der Benutzer nicht fälschlicherweise das Gefühl bekommt, dass das System nicht mehr antwortet und den Konfigurationsprozess abbricht.

3.4 Grundsätze der Dialoggestaltung - ISO 9241-110

Die Norm DIN EN ISO 9241 mit dem Titel *Ergonomie der Mensch-System-Interaktion* beschreibt bestimmte Richtlinien für und Anforderungen an Arbeitsumgebungen, Hardware und Software mit dem Ziel, die Gesundheit der Benutzer am Bildschirm zu schonen und das Arbeiten zu erleichtern. Für diese Arbeit ist speziell der Abschnitt 9241-110 interessant, der sich mit den Grundsätzen der Dialoggestaltung beschäftigt und das Ziel verfolgt, GUI[11]-Designs von interaktiven Systemen, also zum Beispiel von Webseiten und im Speziellen auch von Konfigurationssystemen, leicht bedienbar zu halten. Folgende sieben Grundsätze der Dialoggestaltung sollen von jedem GUI-Entwickler stets eingehalten werden:

- Aufgabenangemessenheit
- Selbstbeschreibungsfähigkeit
- Steuerbarkeit
- Erwartungskonformität
- Fehlertoleranz
- Individualisierbarkeit
- Lernförderlichkeit

Nachfolgend werden diese sieben Grundsätze theoretisch eingeführt und mit anschaulichen Beispielen hinterlegt.

3.4.1 Aufgabenangemessenheit

Die *Aufgabenangemessenheit* ist in der ISO 9241-110-Norm wie folgt definiert:

[11] grafical user interface, dt.: grafische Benutzerschnittstelle

> "Ein interaktives System ist aufgabenangemessen, wenn es den Benutzer unterstützt, seine Arbeitsaufgabe zu erledigen, d.h., wenn Funktionalität und Dialog auf den charakteristischen Eigenschaften der Arbeitsaufgabe basieren, anstatt auf der zur Aufgabenerledigung eingesetzten Technologie." [Sch08]

Es sollen dem Benutzer also alle zur Aufgabenerfüllung notwendigen Informationen angezeigt und gleichzeitig alle irrelevanten Informationen weggelassen werden. Weiterhin wird eine an die jeweilige Arbeitsaufgabe angepasste Form der Ein- bzw. Ausgabe gefordert, wobei typische Eingabewerte automatisch voreingestellt sind. Dialogschritte, die Aktivitäten enthalten, die automatisch vom System ausgeführt werden können, sind unnötig und sollen vermieden werden.[Sch08]

3.4.2 Selbstbeschreibungsfähigkeit

> "Ein Dialog ist in dem Maße selbstbeschreibungsfähig, in dem für den Benutzer zu jeder Zeit offensichtlich ist, in welchem Dialog, an welcher Stelle im Dialog er sich befindet, welche Handlungen unternommen werden können und wie diese ausgeführt werden können." [Sch08]

Gefordert wird hier eine gezielte Unterstützung bei der Aufgabenerledigung, zum Beispiel durch Anleitungen, Rückmeldungen oder Zustandsinformationen, wobei ein Wechsel zwischen der eigentlichen Systemanzeige und externen Informationen vermieden werden sollte. Dadurch sollen selbsterklärende Menüpunkte und Navigationselemente, sowie ein (kontextsensitives) Hilfesystem zum Einsatz kommen. Auch die Eingabemöglichkeit an sich und deren Format oder Einheit sollte für den Benutzer klar sein.[Sch08]

3.4.3 Erwartungskonformität

> "Ein Dialog ist erwartungskonform, wenn er den aus dem Nutzungskontext heraus vorhersehbaren Benutzerbelangen sowie allgemein anerkannten Konventionen entspricht." [Sch08]

Der GUI-Entwickler sollte sich bei der Dialoggestaltung also an gängige Konventionen halten und für den Benutzer aus Erfahrung vertraute Begriffe, Symbole oder Strukturierungen/Formatierungen verwenden. Im Idealfall erfolgt nach jeder Aktion des Benutzers ein entsprechendes kurzes, objektives und konstruktives Feedback des Systems und auch über längere Lade- bzw. Antwortzeiten sollte der Benutzer unmittelbar informiert werden. Innerhalb der Anwendung sollen Dialoge und Informationen konsistent dargestellt und dadurch eine gewisse Routine beim Benutzer erreicht werden.[Sch08]

3.4.4 Lernförderlichkeit

> "Ein Dialog ist lernförderlich, wenn er den Benutzer beim Erlernen der Nutzung des interaktiven Systems unterstützt und anleitet."[Sch08]

Lernförderliche Systeme bieten Oberflächen mit erläuternden Informationen und Lernhilfen, wobei die Lernschwelle am besten relativ gering gehalten wird. Eine minimale Eingabe von Informationen soll erreicht werden.[Sch08]

3.4.5 Steuerbarkeit

> "Ein Dialog ist steuerbar, wenn der Benutzer in der Lage ist, den Dialogablauf zu starten sowie seine Richtung und Geschwindigkeit zu beeinflussen, bis das Ziel erreicht ist."[Sch08]

Die Geschwindigkeit der Interaktion ist vom Benutzer steuerbar, das bedeutet auch, dass der Interaktionsprozess pausiert und zu einem späteren Zeitpunkt wieder aufgenommen werden kann. Der Nutzer sollte die Möglichkeit haben, seine Dialogschritte (mindestens einen Dialogschritt) rückgängig zu machen.[Sch08]

3.4.6 Fehlertoleranz

> "Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand seitens des Benutzers erreicht werden kann. Fehlertoleranz wird mit den Mitteln erreicht:
>
> - Fehlererkennung und -vermeidung (Schadensbegrenzung);
> - Fehlerkorrektur oder
> - Fehlermanagement, um mit Fehlern umzugehen, die sich ereignen.
>
> "[Sch08]

Dabei sollte die Anwendung den Nutzer bei der Fehlererkennung bzw. Fehlervermeidung unterstützen und undefinierte Systemzustände verhindern. Durch ein Hilfesystem sollte der User bei der Beseitigung von Fehlern mit möglichst geringem Aufwand angeleitet und nach einer eventuell automatischen Fehlerkorrektur benachrichtigt werden. Eine Validierung der Eingaben vor der Verarbeitung durch das System ist ratsam.[Sch08]

3.4.7 Individualisierbarkeit

> "Ein Dialog ist individualisierbar, wenn Benutzer die Mensch-System-Interaktion und die Darstellung von Informationen ändern können, um diese an ihre individuellen Fähigkeiten und Bedürfnisse anzupassen." [Sch08]

Liegen Dialoge vor, die häufig von Benutzern mit unterschiedlichen Belangen bedient werden, so kann die Möglichkeit der individuellen Anpassung der Dialoge durch den Benutzer sinnvoll sein, zum Beispiel durch die Auswahl der Dialogtechnik.[Sch08]

3.5 Ausgewählte Gründe für benutzerinitiierte Prozessabbrüche

Trotz der immer weiteren Verbreitung des Internets in der Gesellschaft und der wachsenden Vertrautheit der Nutzer mit E-Commerce-Prozessen, gibt es weiterhin einige Hürden, die für viele Kunden gegen den Online-Einkauf sprechen oder zu benutzerinitiierten Prozessabbrüchen führen.

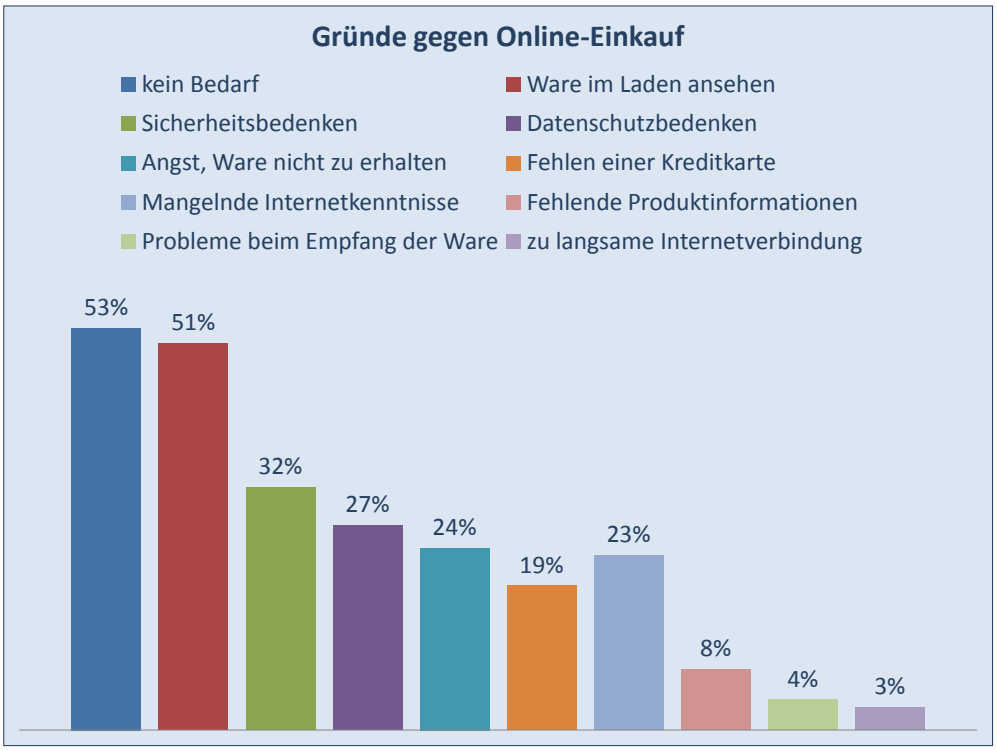


Abbildung 17: Gründe, die gegen den Online-Einkauf sprechen (vgl. [CM02])

Die Gründe für benutzerinitiierte Prozessabbrüche kann man in drei Kategorien einteilen (vgl. [HL06]):

- Vernunftsentscheidung
- Technische Probleme
- Unsicherheit bzw. Unentschlossenheit des Kunden

Zur Gruppe der Vernunftsentscheidungen gehört zum Beispiel, dass dem Kunden das jeweilige Produkt bzw. die angebotenen Produktkonfigurationen nicht zusagen. Lange Antwortzeiten hingegen stellen ein Beispiel für technische Probleme dar. Im weiteren Verlauf wird die dritte Gruppe, die benutzerinitiierten Prozessabbrüche auf Grund von Unsicherheit bzw. Unentschlossenheit des Kunden, beschrieben.

3.5.1 Vielzahl an Konfigurationsmöglichkeiten

Eine stetig wachsende Vielfalt und Komplexität der Produkte und damit auch der Konfigurationsmöglichkeiten, vor allem im Bereich des Mass Customization, birgt die Gefahr, dass der Kunde bei der Auswahl seiner Wunschkonfiguration überfordert ist. Das führt zu Verzögerungen in der Kaufentscheidung, suboptimalen Konfigurationsentscheidungen und im schlimmsten Fall stellt dies einen benutzerinitiierten Prozessabbruchgrund dar.

> "Product variety and choice have the potential to add significant complexity to the selling process, thus making the process more difficult for the consumer. Confusion from the complexity of customization might even lead to purchase abandonment." [GT02]

3.5.2 Fehlende persönliche Beratung

> "66% of all on-line shoppers during last year's holiday shopping season abandoned their shopping carts before checkout." [SSK00]

Einer der Hauptgründe für diesen Umstand ist die fehlende persönliche Beratung[12].

3.5.3 Irreführende Benutzerführung

Oftmals besteht das Problem, dass der Benutzer sein gewünschtes Produkt nicht findet oder Schwierigkeiten hat, es zu konfigurieren.[13]

[12] vgl. [SSK00]
[13] vgl. [SSK00]

3.6 Interaction Design Patterns für die Produktkonfiguration

Nachdem im Vorangegangenen die Bedürfnisse der Kunden im Bezug auf webbasierte Produktkonfigurationslösungen eruiert, Grundsätze der Dialoggestaltung nach dem ISO 9241-110 Standard eingeführt und ausgewählte Gründe für benutzerinitiierte Prozessabbrüche identifiziert wurden, wird im Nachfolgenden auf einige *Interaction Design Patterns*[14] für die Konfigurationssysteme eingegangen (vgl. [Dre08]). Durch die Einhaltung dieser *Design Pattern* kann eine Vielzahl von Usability-Problemen vermieden werden.

Man kann diese Entwurfsmuster in drei Kategorien einteilen:

- Ziele und Angebote
- Navigation und Führung
- Auswahl und Feedback

3.6.1 Ziele und Angebote

Vorkonfigurationen Um die Komplexität der Produkte zu verringern und somit den Konfigurationseinstieg für den Benutzer zu erleichtern, ist es sinnvoll, einige vorkonfigurierte Basisprodukte anzubieten. Diese Basisprodukte kann der Kunde dann im weiteren Verlauf nach individuellen Vorstellungen konfigurieren. Durch diese Vorgehensweise soll verhindert werden, dass der Kunde von einer unüberschaubaren Vielzahl an Konfigurationsmöglichkeiten überfordert wird und den Konfigurationsprozess abbricht.

Wiederkauf Die genaue Konfiguration von bereits fertiggestellten und vom Kunden bestellten Produkten wird gespeichert und steht dem Kunden zum erneuten Bestellen zur Verfügung. Dadurch wird die Kundenbindung gefördert, da für das gleiche Produkt kein mehrmaliges Konfigurieren nötig ist.

Konfiguration erweitern Eine gespeicherte Konfiguration soll aber nicht nur für den identischen Wiederkauf zur Verfügung stehen, sondern kann auch in einem neuen Konfigurationsprozess erweitert werden.

Sofort verfügbare Produkte Zusätzlich zum Konfigurationsangebot sollte der Kunde stets die Möglichkeit haben, ein bereits fertiges Produkt sofort zu erwerben. Dadurch kann man Kunden an ein Unternehmen binden, die ihr Produkt nicht selbst konfigurieren möchten.

[14] Entwurfsmuster für die Interaktionsgestaltung

Produkt-Service-Paket Man kann den Nutzen für den Kunden steigern, indem man zusätzlich zum konfigurierten Produkt ein passendes individuelles Service-Angebot bietet. Das überzeugt vor allem Kunden, die bevorzugt offline einkaufen und auf den vermeintlich besseren Service von lokalen Anbietern setzen.

Sonderwünsche Zusätzlich zur normalen Konfiguration sollte dem Anwender die Möglichkeit gegeben werden, individuelle Sonderwünsche zu kommunizieren, welche dann in das Produkt integriert werden.

Individualisierte Broschüre Nach der abgeschlossenen Konfiguration wird dem Kunden im Idealfall eine individuelle Broschüre bzw. ein individuelles Angebot übermittelt. Der Kunde kann seine Konfiguration somit noch einmal in Ruhe betrachten, überdenken und ggf. auch ausdrucken und mit anderen Personen besprechen.

Konfigurationen vergleichen Mehrere unterschiedliche gespeicherte Konfigurationen sollten miteinander verglichen werden können. Das bietet Transparenz und erleichtert den Entscheidungsprozess für den Kunden.

3.6.2 Navigation und Führung

Story-Navigation Eine sogenannte *Story-Navigation* führt den Benutzer gezielt Schritt für Schritt durch die Konfiguration. Dadurch wird ein Prozessfluss hin zum fertigen Produkt erreicht und verhindert, dass der Kunde den roten Faden verliert und aus Frust die Konfiguration abbricht.

Konfigurationsfortschritt anzeigen Durch das Anzeigen des Konfigurationsfortschritts ist der bevorstehende Aufwand für den Kunden jeder Zeit klar ersichtlich. Das verhindert Prozessabbrüche von ungeduldigen Benutzern oder solchen, die meinen, sich in einem endlosen Fragenkatalog zu befinden.

Expertenmodus Für verschiedene Benutzergruppen sollten auch unterschiedliche Konfigurationsmodi zur Verfügung stehen. So benötigt zum Beispiel ein Endkunde mehr Führung und Unterstützung bei der Konfiguration, als ein Vertriebsmitarbeiter, der bevorzugt in einem schnellen Konfigurationsprozess sein Expertenwissen einbringt.

Konfiguration speichern Zu jedem Zeitpunkt soll ein Speichern der aktuellen Konfiguration möglich sein, um diese später wieder aufzugreifen und im Konfigurationprozess fortzufahren.

3.6.3 Auswahl und Feedback

Farbkacheln Farben, Formen und Muster sollen dem Anwender realitätsnah angezeigt werden. Damit fällt dem User der Vergleich und die Entscheidung leichter.

Visuelle Konfiguration Eine Konfiguration direkt am realen Produktmodell erleichtert es dem Anwender, sich sein konfiguriertes Produkt vorzustellen. Es gibt mehrere Verfahren, um dieses Ziel zu erreichen:

- 3D-Darstellung des realen Produktes
- Verwendung von Renderbildern (Abbildung entspricht exakt der aktuellen Konfiguration)
- Produkt im Einsatz bzw. in seiner Anwendungsumgebung zeigen

Konfigurationsübernahme Bei einem Wechsel zwischen unterschiedlichen Basisprodukten sollen die bereits getätigten Konfigurationsentscheidungen übernommen werden, sofern dies nach dem Regelwerk möglich ist.

Automatische Vervollständigung Die Implementierung einer Auto-Vervollständigung verkürzt den Konfigurationsprozess und reduziert dadurch benutzerinitiierte Prozessabbrüche.

Konfliktvorwegnahme Der Benutzer sollte niemals ein ungültiges Produkt erhalten. Deswegen ist es Ziel des Konfigurationssystems, Konflikte zum Beispiel durch einen intelligenten Alternativen Wizard[15] zu umgehen.

Persönliches Design Einen hohen Grad der Individualisierung erreicht man, indem man dem Kunden Funktionen zur kreativen Gestaltung (individuelle Verpackung kreieren, persönlichen Text hinzufügen, usw.) des Endproduktes zur Verfügung stellt.

[15] siehe auch Kapitel 5.3.2

4 Intelligentes Formular zur Produktkonfiguration

Ein Hauptziel dieser Arbeit war die Konzeption und Implementierung einer intelligenten grafischen Oberfläche für die Produktkonfiguration, im Folgenden auch *intelligentes Formular* genannt. Als Basis für die Entwicklung diente die bisherige Oberfläche des CAS Configurator Merlin im Stil eines einfachen Warenkorbs (vgl. Kapitel 2.4).

4.1 Verwendete Technologie

Das intelligente Formular wurde in *JSF 2.0* unter Verwendung von *Primefaces 3.4* implementiert. Für die Implementierung der Programmlogik wurde auf *Java 7* und die *JavaBeans-Technologie* zurückgegriffen.

4.2 Spezifikation

4.2.1 Die Merkmalsstruktur

Im Falle des CAS Configurator Merlin wird die Merkmalsstruktur in einer vierstufigen Hierarchie abgebildet, die sich am Produktaufbau orientiert. Die vier Hierarchiestufen werden mit Hilfe einer Baumstruktur dargestellt, weshalb im weiteren Verlauf von einem *Merkmalsbaum* gesprochen wird. Spezifische Merkmalsausprägungen finden sich nur auf der vierten Hierarchieebene wieder. In Abbildung 18 wird dies am Beispiel eines PC-Konfigurators verdeutlicht. Dabei ist *Speicher* ein Oberpunkt auf erster Hierarchieebene. Dieser hat drei *Kind-Knoten*, um verschiedene Speichertypen darzustellen, nämlich *Arbeitsspeicher*, *Festplatte* und *Zusätzliche Festplatte*. Auch der Knoten zweiter Hierarchieebene *Arbeitsspeicher* hat einen Kind-Knoten *Größe*, welche als vierte Stufe mehrere Ausprägungen enthält.

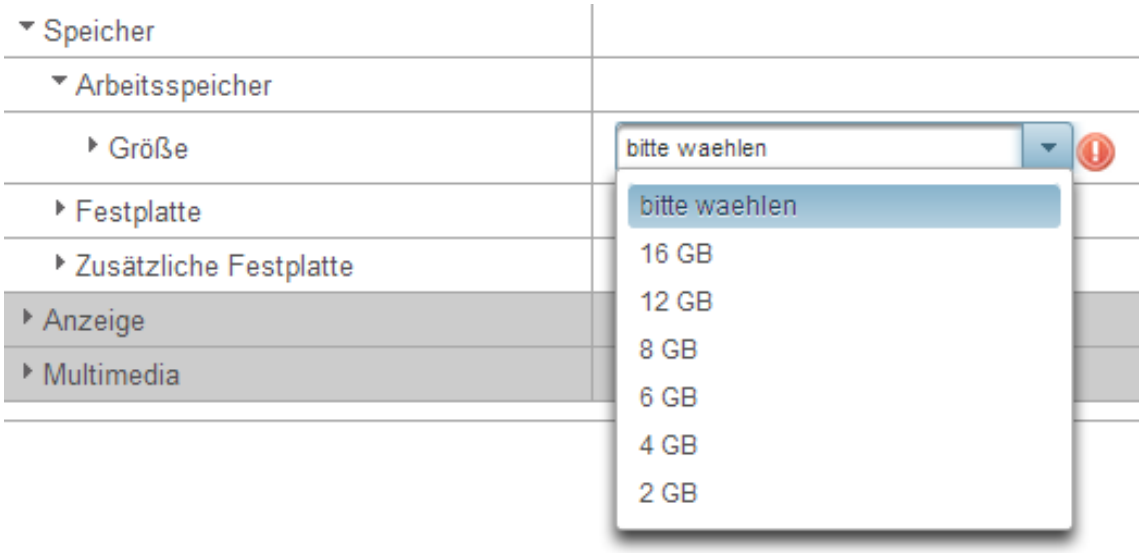


Abbildung 18: Vierstufige Hierarchie der Merkmalsstruktur

4.2.2 Vertriebsprodukte und Vertriebsbausteine

Vertriebsprodukte stellen das in Kapitel 3.6.1 vorgestellte Konzept der Vorkonfigurationen dar und verhelfen dem Benutzer zu einem einfacheren und schnelleren Einstieg in die Konfiguration. Um beim oben eingeführten Beispiel des PC-Konfigurators zu bleiben, sind hier unter anderem folgende Vertriebsprodukte denkbar:

- Der neue Highend-Gaming-PC
- Das meistverkaufte Laptop
- Das ultimative Highspeed-Tablet

Der Anwender kann ein vorhandenes Vertriebsprodukt auswählen und im Anschluss um- bzw. fertigkonfigurieren oder auch unverändert lassen.

Vertriebsbausteine liegen eine Abstraktionsebene unter den Vertriebsprodukten und stellen keine fertigen Vorkonfigurationen dar. Vielmehr definieren sie die Rahmenbedingungen für die weitere Konfiguration und aktivieren die für die Konfiguration relevanten Bereiche im *Merkmalsbaum*. Im PC-Konfigurator-Beispiel entscheidet sich der Benutzer zwischen den drei grundlegenden Konfigurationskategorien:

- PC
- Laptop

- Tablet

Die einzelnen Ausprägungen der Elemente sind hierbei noch nicht vorgegeben.

4.2.3 Kardinalitäten

> "Die Kardinalität gibt an, wie viele Ausprägungen einer Entität an einer Beziehung minimal und maximal beteiligt sein können." [Wie08]

Für die Darstellung von Elementen mit verschiedenen *Kardinalitäten* müssen unterschiedliche Auswahltypen zur Verfügung stehen, die sich an den *Kardinalitäten* orientieren. In unserem Fall ist die Kardinalität der dritten Hierarchieebene relevant, die angibt, wie viele verschiedene Ausprägungen in der vierten Ebene für das jeweilige Elternelement möglich sind. Im Folgenden werden vier mögliche *Kardinalitäten* und die jeweils passenden Auswahltypen vorgestellt.

$< 1,1 >$ **Kardinalität** Jede Entität dritter Hierarchieebene steht in Beziehung mit genau einer Entität vierter Hierarchieebene. Mit anderen Worten: Jedes Element auf dritter Ebene hat genau eine Ausprägung. Im *intelligenten Formular* wird eine Dropdown-Liste als Steuerelement zur Konfiguration verwendet.

$< 1,* >$ **Kardinalität** Jede Entität dritter Hierarchieebene steht in Beziehung mit mindestens einer Entität vierter Hierarchieebene. Mit anderen Worten: Jedes Element auf dritter Ebene hat mindestens eine Ausprägung. Im *intelligenten Formular* werden Listen mit Checkboxen als Steuerelemente zur Konfiguration verwendet.

$< 0,1 >$ **Kardinalität** Jede Entität dritter Hierarchieebene steht in Beziehung mit keiner oder genau einer Entität vierter Hierarchieebene. Mit anderen Worten: Jedes Element auf dritter Ebene hat gar keine oder genau eine Ausprägung. Im *intelligenten Formular* wird eine Dropdown-Liste als Steuerelement zur Konfiguration verwendet, wobei keine Auswahl auch eine mögliche Option darstellt.

$< 0,* >$ **Kardinalität** Jede Entität dritter Hierarchieebene steht in Beziehung mit beliebig vielen Entitäten vierter Hierarchieebene. Mit anderen Worten: Jedes Element auf dritter Ebene hat beliebig viele Ausprägungen, also von keiner Ausprägung bis zu vielen Ausprägungen ist alles möglich. Im *intelligenten Formular* werden Checkboxen als Steuerelemente zur Konfiguration verwendet.

4.2.4 Grafischer Entwurf des intelligenten Formulars

In Abbildung 19 ist der grafische Entwurf des *intelligenten Formulars*, orientiert am in Kapitel 3.3.1 eingeführten zweistufigen Standardlayout, dargestellt. Durch einen Oberflächenaufbau nach einem weit verbreiteten Schema wird der Grundsatz der Erwartungskonformität (siehe Kapitel 3.4.3) umgesetzt und dem Seitenbesucher das Gefühl von Vertrautheit und Sicherheit vermittelt. Auch die in Kapitel 3.1 eingeführten allgemeinen Gestaltungsgesetze finden im Design der Benutzeroberfläche Anwendung. So erfüllt die Anordnung der verschiedenen Bereiche (Hauptnavigation, lokale Navigation, Contentbereich) das Gesetz der Kontinuität, da die einzelnen Elemente an gerade verlaufenden Linien angeordnet sind.

Hauptnavigation - Die View Selection Der Bereich der Haupnavigation (roter Bereich) wird für die Darstellung der *View Selection* verwendet. Dabei wird zwischen zwei unterschiedlich granularen Konfigurationsmodi unterschieden:

- Express-View
- Non-Guided-View

Die *Express-View* eignet sich für den schnellen Einstieg in die Konfiguration über bereits vorkonfigurierte Vertriebsprodukte, wohingegen die *Non-Guided-View* Vertriebsbausteine mit größter Flexibilität anbietet.

Lokale Navigation - Filter und Steuerelemente In der lokalen Navigation (grüner Bereich) werden dem Anwender Filteroptionen und Steuerelemente zur Verfügung gestellt. Zu diesen Elementen zählen:

- "Alle ausklappen"-Funktion
- "Alle zuklappen"-Funktion
- "Schritt rückgängig machen"-Funktion
- "Konfiguration speichern"-Funktion
- "Konfiguration laden"-Funktion
- Filter für noch zu bearbeitende Elemente
- Filter für zuletzt eingebaute Elemente

Mit dieser Navigationsleiste wird vor allem der Grundsatz der Steuerbarkeit (siehe Kapitel 3.4.5) erfüllt.

Contentbereich - Der Konfigurationsdialog Der Contentbereich ist in drei Spalten unterteilt und stellt den *Merkmalsbaum* und Funktionen zur Produktkonfiguration bereit. Die linke Spalte (blau) beinhaltet Hierarchieebene eins bis drei und dient zur übersichtlichen Strukturierung komplexer Produkte in einzelne Produktbestandteile. Die mittlere Spalte (lila) enthält alle Merkmalsausprägungen, also die vierte Hierarchieebene, und stellt den Bereich der eigentlichen Konfiguration dar. Die rechte Spalte (gelb) bietet Platz für zusätzliche Operationen und Hilfe- bzw. Feedbacksymbole.

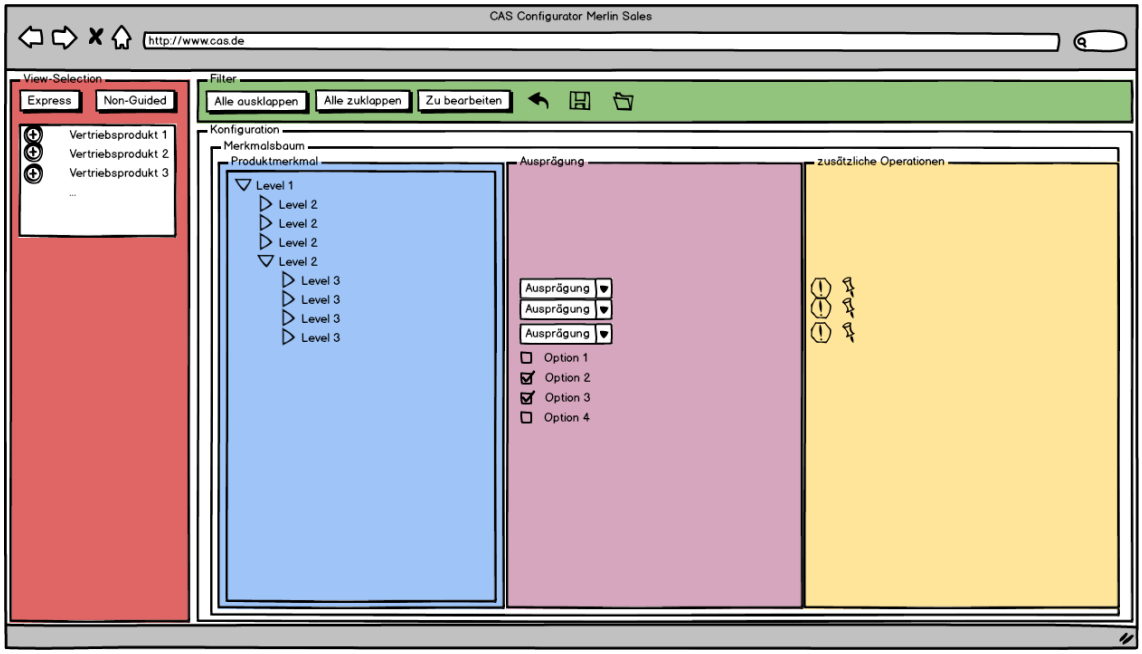


Abbildung 19: Grafischer Entwurf des intelligenten Formulars

4.3 Implementierung

Die Implementierung des *intelligenten Formulars* unterteilt sich in zwei verschiedene Bereiche - die Gestaltung der Oberfläche in JSF und die Programmlogik in den jeweiligen Java Beans. Eine Übersicht über alle für diese Arbeit implementierten Klassen ist in Tabelle 2 zu finden.

4.3.1 Wichtige Datentypen

In Abbildung 20 sind die wichtigsten und für diese Arbeit relevanten Datentypen des *CAS Confgurator Merlin* dargestellt.

Klassenname	Typ	Kurzbeschreibung
CatalogBean	Java Bean	In dieser Klasse wird der Merkmalsbaum aufgespannt
ConfigurationBean	Java Bean	Diese Klasse implementiert die Konfigurationslogik der Oberfläche
content_configuration	JSF View	Diese View beinhaltet den Merkmalsbaum und den Konfigurationsdialog
viewselection	JSF View	Diese View enthält die Auswahl des Konfigurationsmodus
filter	JSF View	Diese View enthält die Steuerelemente

Tabelle 2: Übersicht über alle für diese Arbeit implementierten Klassen

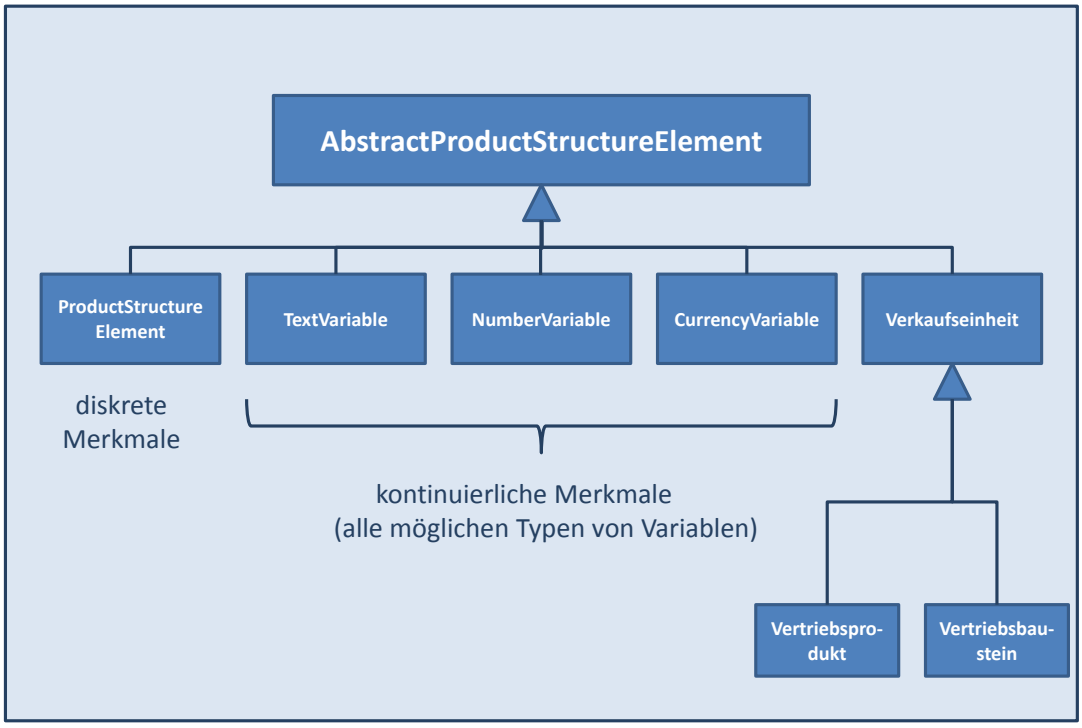


Abbildung 20: Übersicht über die wichtigetsn Datentypen

AbstractProductStructureElement Jedes in der Konfiguration verwendete Element ist vom Typ *AbstractProductStructureElement* (siehe Quellcode auf CD). Ein *AbstractProductStructureElement* hat eine eindeutige *UUID*[16] über welche es identifiziert und referenziert werden kann und einen Namen. Außerdem beinhaltet es die für den *Alternativen-Wizard*[17] benötigten Metadaten zum Beispiel zur Bestimmung der Priorität und der Wizard-Frage. Weiterhin erhält man über die Methode *getChildren()* die Kindknoten eines *AbstractProductStructureElement*.

Da es im *CAS Configurator Merlin* aber mehrere verschiedene Elementtypen gibt, muss es auch unterschiedliche Klassen mit individuellen Eigenschaften geben, die von der Klasse *AbstractProductStructureElement* erben und diese erweitern:

- ProductStructureElement
- TextVariable
- NumberVariable
- CurrencyVariable
- Verkaufseinheit

ProductStructureElement Alle Merkmale und Merkmalsausprägungen im *Merkmalsbaum*, die keine Variablen sind, sind vom Typ *ProductStructureElement*. Ein solches *ProductStructureElement* hat eine bestimmte Kardinalität und beinhaltet die Information, ob das Element aktiviert oder selektiert ist. Man bezeichnet diese Elemente auch als *diskrete Merkmale*.

Variablen Die Variablen *TextVariable*, *NumberVariable* und *CurrencyVariable* sind wie ihre Namen bereits verraten textuelle und numerische Variablen bzw. Währungsvariablen und werden als *kontinuierliche Merkmale* bezeichnet. Diese Datentypen werden in den für diese Arbeit entwickelten Konzepten nicht berücksichtigt.

Verkaufseinheit Der Datentyp *Verkaufseinheit* ist der Obertyp für Vertriebsprodukte und Vertriebsbausteine, welche für die *Express-View* bzw. *Non-Guided-View* relevant sind und einen Einstieg in die Konfiguration bieten.

[16] Universally Unique Identifier - Standard für Identifikatoren
[17] siehe auch Kapitel 5.3.2

4.3.2 Die CatalogBean

Die *CatalogBean* hat die Hauptaufgabe, den *Merkmalsbaum* aufzuspannen (siehe Abbildung 21 für einen Ausschnitt aus der *CatalogBean*). Hierfür werden vier ineinander verschachtelte *For-Each-Schleifen* durchlaufen - für jede Hierarchiebene eine - angefangen mit der ersten Hierarchieebene und der zugehörigen *Collection psElementList* mit Elementen vom Typ *AbstractProductStructureElement*. Gleichzeitig werden die jeweiligen Elemente als *DefaultTreeNode*[18] des Levels 1 bis 4 - abhängig von ihrer Hierarchieebene - in die Baumstruktur eingefügt und eine neue Collection mit deren Kindknoten erstellt, die in der nächst inneren *For-Each-Schleife* durchlaufen wird. Dieser Ablauf findet für jedes Element statt, bis der komplette vierstufige *Merkmalsbaum* abgebildet ist.

Weitere Aufgaben der *CatalogBean* sind:

- Herausfinden der passenden Darstellungsweise für die Ausprägungen in Abhängigkeit von deren Kardinalität[19] in folgenden Methoden:
  - *isOptionalDropdown*
  - *isRequiredDropdown*
  - *isOptionalCheckbox*
  - *isRequiredCheckbox*
- Komplettes Auf- bzw. Zuklappen des *Merkmalsbaumes* in der Methode *expandAll*
- Bestimmung ob aktuell beim Durchlaufen der Produkstruktur ein Element vierter Hierarchieebene vorliegt, welches in der zweiten Spalte des Konfigurationsformulars dargestellt werden muss, in der Methode *hasNoLeaves*

[18] siehe Primefaces 3.4: org.primefaces.model.DefaultTreeNode
[19] siehe Kapitel 4.2.3 für weitere Informationen bezüglich der Kardinalitäten

```
public void buildProductCatalog(Collection<AbstractProductStructureElement>
         psElementList, IProductStructureRepository salesUnitRepository) throws IOException {
            root = new DefaultTreeNode("root", null);
            Collection<AbstractProductStructureElement> secondLevelElementList =
                           new ArrayList<AbstractProductStructureElement>();
            Collection<AbstractProductStructureElement> thirdLevelElementList =
                           new ArrayList<AbstractProductStructureElement>();
            Collection<AbstractProductStructureElement> fourthLevelElementList =
                           new ArrayList<AbstractProductStructureElement>();

            // get List of all Elements of the second level in product-structure
            for (AbstractProductStructureElement firstLevelElement : psElementList) {
                  secondLevelElementList.clear();
                  level1 = null;
                  secondLevelElementList.addAll(firstLevelElement.getChildren());
                  if (firstLevelElement.getType() == ProductStructureElementType.ELEMENT) {
                        ProductStructureElement p1 = (ProductStructureElement) firstLevelElement;
                        p1.setFeature(firstLevelElement.getName());
                        p1.setCharacteristic("");
                        if(p1.getisActivated()) {
                        level1 = new DefaultTreeNode(firstLevelElement, root);
                        }
                  }

                  // get List of all Elements of the third level in product-structure
                  for (AbstractProductStructureElement secondLevelElement : secondLevelElementList) {
                        level2 = null;
                        ProductStructureElement p2 = (ProductStructureElement) secondLevelElement;
                        p2.setCharacteristic("");
                        p2.setFeature(secondLevelElement.getName());
                        thirdLevelElementList.clear();
                        thirdLevelElementList.addAll(secondLevelElement.getChildren());
                        if (p2.isVisible() && p2.getisActivated()) {
                         level2 = new DefaultTreeNode(p2, level1);
                        }

                        // get List of all Elements of the fourth level in
                        // product-structure
                        for (AbstractProductStructureElement thirdLevelElement : thirdLevelElementList) {
                              level3 = null;
                              ProductStructureElement p3 = (ProductStructureElement) thirdLevelElement;
                              p3.setCharacteristic("");
                              p3.setFeature(thirdLevelElement.getName());
                              fourthLevelElementList.clear();
                              fourthLevelElementList.addAll(thirdLevelElement
                                          .getChildren());
                              if (p3.isVisible() && p3.getisActivated()) {
                                    level3 = new DefaultTreeNode(p3, level2);
                              }
                              for (AbstractProductStructureElement fourthLevelElement : fourthLevelElementList) {
                                    ProductStructureElement p4 = (ProductStructureElement) fourthLevelElement;
                                    p4.setCharacteristic(fourthLevelElement
                                                .getName());
                                    p4.setFeature("");
                                    if (p4.isVisible() && p4.getisActivated()) {
                                          temp.add(p4);
                                    }
                              }
                              TreeNode level4 = null;
                              level4 = new DefaultTreeNode(temp, level3);
                        }
                  }
            }
```

Abbildung 21: Ausschnitt der Methode *buildProductCatalog* spannt den *Merkmalsbaum* auf

4.3.3 Die ConfigurationBean

Die *ConfigurationBean* ist vor allem dafür zuständig, die eigentliche Konfiguration zu steuern. Zu ihren Aufgaben zählen unter anderem:

- Verarbeitung von Ausprägungsveränderungen im *intelligenten Formular*
- Einbau von Elementen in die Konfiguration (siehe Abbildung 22)
- Ausbau von Elementen aus der Konfiguration

Im Folgenden wird der Ausschnitt aus der *ConfigurationBean* erläutert, der den Einbau eines Elements in die aktuelle Konfiguration implementiert.

Wie auch schon beim Aufspannen des *Merkmalsbaumes* in der *CatalogBean* werden zu Beginn der Methode *processBuildIn* die vier ineinander verschachtelten *For-Each-Schleifen* durchlaufen. Allerdings werden hier nicht die unterschiedlichen *DefaultTreeNode's* definiert, sondern unter allen möglichen Elementen das *featureToBuildIn* - des vom Benutzer im aktuellen Konfigurationsschritt ausgewählte Element - gesucht und zur *ArrayList*[20] *elementsToBuildIn* hinzugefügt.

Anschließend wird die Methode *reconfigureCurrentConfiguration* aufgerufen, die die Aufgabe hat, den Konfigurationsschritt zu vollziehen und vom Konfiguratorkern *XPS Merlin* die neue Konfiguration berechnen zu lassen. Dieser Methode wird mit *currentConfiguration* die aktuelle Konfiguration vor dem Konfigurationsschritt des Benutzers und mit *elementsToBuildIn* das einzubauende Element übergeben. *elementsToDelete*, *pinnedElements* und *null* sind für meine prototypische Implementierung in diesem Schritt irrelevant, müssen jedoch der Vollständigkeit halber mit übergeben werden.

Als Ergebnis der Neukonfiguration erhält man mit *configurationWithAlternativeSupport* eine neue Konfiguration, die entweder keine Alternativen enthält und sofort als *currentConfiguration* verwendet werden kann, oder Alternativen enthält und eine Lösung des Konflikts per *Alternativen-Dialog* bzw. *Alternativen-Wizard* vom Benutzer verlangt. In diesem Fall wird die vom Anwender gewählte Alternative zur *currentConfiguration*. Abschließend wird der *Merkmalsbaum* von der *CatalogBean* neu aufgespannt.

Ähnlich verhält es sich auch beim Ausbau eines Elements aus der aktuellen Konfiguration in der Methode *processBuildOut*.

[20] siehe java.util.ArrayList⟨E⟩

```
private void processBuildIn(String id) throws IOException {
            Collection<AbstractProductStructureElement> psElementList =
                    configurationService.getProductStructureRepository().getElements();
            Collection<AbstractProductStructureElement> secondLevelElementList =
                    new ArrayList<AbstractProductStructureElement>();
            Collection<AbstractProductStructureElement> thirdLevelElementList =
                    new ArrayList<AbstractProductStructureElement>();
            Collection<AbstractProductStructureElement> fourthLevelElementList =
                    new ArrayList<AbstractProductStructureElement>();

            for (AbstractProductStructureElement firstLevelElement : psElementList) {
                  secondLevelElementList.clear();
                  if (firstLevelElement.getId().toString().equals(id)
                              && firstLevelElement.getType().equals(
                                          ProductStructureElementType.ELEMENT)) {
                        ProductStructureElement firstLevelElementCast =
                              (ProductStructureElement) firstLevelElement;
                        featureToBuildIn = firstLevelElementCast;
                  }
                  secondLevelElementList.addAll(firstLevelElement.getChildren());

                  // get List of all Elements of the third level in
                  // product-structure
                  for (AbstractProductStructureElement secondLevelElement : secondLevelElementList) {
                        ProductStructureElement p2 = (ProductStructureElement) secondLevelElement;
                        if (p2.getId().toString().equals(id)) {
                              featureToBuildIn = p2;
                        }
                        thirdLevelElementList.clear();
                        thirdLevelElementList.addAll(secondLevelElement.getChildren());
                        // get List of all Elements of the fourth level in
                        // product-structure
                        for (AbstractProductStructureElement thirdLevelElement : thirdLevelElementList) {
                              ProductStructureElement p3 = (ProductStructureElement) thirdLevelElement;
                              if (p3.getId().toString().equals(id)) {
                                    featureToBuildIn = p3;
                              }
                              fourthLevelElementList.clear();
                              fourthLevelElementList.addAll(thirdLevelElement
                                          .getChildren());

                              for (AbstractProductStructureElement fourthLevelElement : fourthLevelElementList) {
                                    ProductStructureElement p4 = (ProductStructureElement) fourthLevelElement;
                                    if (p4.getId().toString().equals(id)) {
                                          featureToBuildIn = p4;
                                    }

                              }
                        }
                  }
            }

            List<ISlotFilling> elementsToDelete = new ArrayList<ISlotFilling>();
            List<ISlotFilling> elementsToBuildIn = new ArrayList<ISlotFilling>();
            elementsToBuildIn.add(new SlotFilling(configurationService
                        .getProductStructureRepository().getElementById(
                              featureToBuildIn.getId())));
            Set<IReferencable> pinnedElements = new HashSet<IReferencable>();
            try {
                  IConfigurationWithAlternativeSupport configurationWithAlternativeSupport =
                        configurationService.reconfigureCurrentConfiguration(currentConfiguration,
                                    elementsToBuildIn, elementsToDelete,
                                    pinnedElements, null);
                  if (!configurationWithAlternativeSupport.hasAlternatives()) {
                        showAlternativeSupport = false;
                        currentConfiguration = configurationWithAlternativeSupport;
                        configHistory.add(currentConfiguration);
                        removeUnselectedElements(currentConfiguration);
                        setCurrentlySelectedElements(currentConfiguration);
                  } else {
                        showAlternativeWizard = true;
                        configurationWithAlternatives = configurationWithAlternativeSupport;
                        alternativeList.clear();
                        alternativeList.add(configurationWithAlternatives);
                        alternativeList.addAll(configurationWithAlternatives
                                    .getAlternatives());
                        getWizardQuestions();
                  }
            } catch (RuleGenerationSetChangedException ex) {

            }
            catalogBean.buildProductCatalog(psElementList, salesUnitRepository);
      }
```

Abbildung 22: Die Methode *processBuildIn* als Ausschnitt der *ConfigurationBean*

5 Dialog zur Darstellung alternativer Konfigurationen

5.1 Ursache alternativer Konfigurationen

Bevor näher auf die Darstellung alternativer Konfigurationsmöglichkeiten eingegangen wird, soll die Ursache für die Entstehung von Alternativen im *CAS Configurator Merlin* geklärt werden. Dabei ist sowohl der Ablauf eines Konfigurationsschrittes, also auch das Auffinden von Lösungsalternativen im Konfliktfall relevant.

5.1.1 Ablauf eines Konfigurationsschrittes

Der schematische Ablauf eines Konfigurationsschrittes am Beispiel des *CAS Configurator Merlin* ist in Abbildung 23 dargestellt. Als *Input* für den Konfigurationsschritt i wird stets ein gültiges Produkt[21] benötigt, das durch eine vom Anwender gewünschte Änderung im *intelligenten Formular* ergänzt bzw. modifiziert wurde. Das dadurch entstandene neue Produkt wird im Konfiguratorkern *XPS Merlin* auf Gültigkeit geprüft. Dabei wird unter anderem getestet, ob alle Konfigurationsregeln und Kardinalitäten erfüllt sind und ob alle Elemente noch für das neue Produkt gültig sind. Ist dies der Fall, so entspricht das neue Produkt dem *Output* des Konfigurationsschrittes. Liegt allerdings ein ungültiges neues Produkt vor, so ist es Aufgabe des *XPS Merlin*, die Gültigkeit wiederherzustellen, da der Anwender des *CAS Configurator Merlin* zu keinem Zeitpunkt ein ungültiges Produkt als *Output* erhält. Der Anwender erhält als *Output* eine Menge von Lösungsalternativen, wobei jede Lösungsalternative jeweils wieder - über das Ein- und Ausbauen von Elementen - zu einem gültigen Produkt führt. Der *Output* von Konfigurationsschritt i stellt gleichzeitig wieder das für den *Input* von Konfigurationsschrit i+1 benötigte gültige Produkt dar.

[21] siehe Kapitel 2.2.6

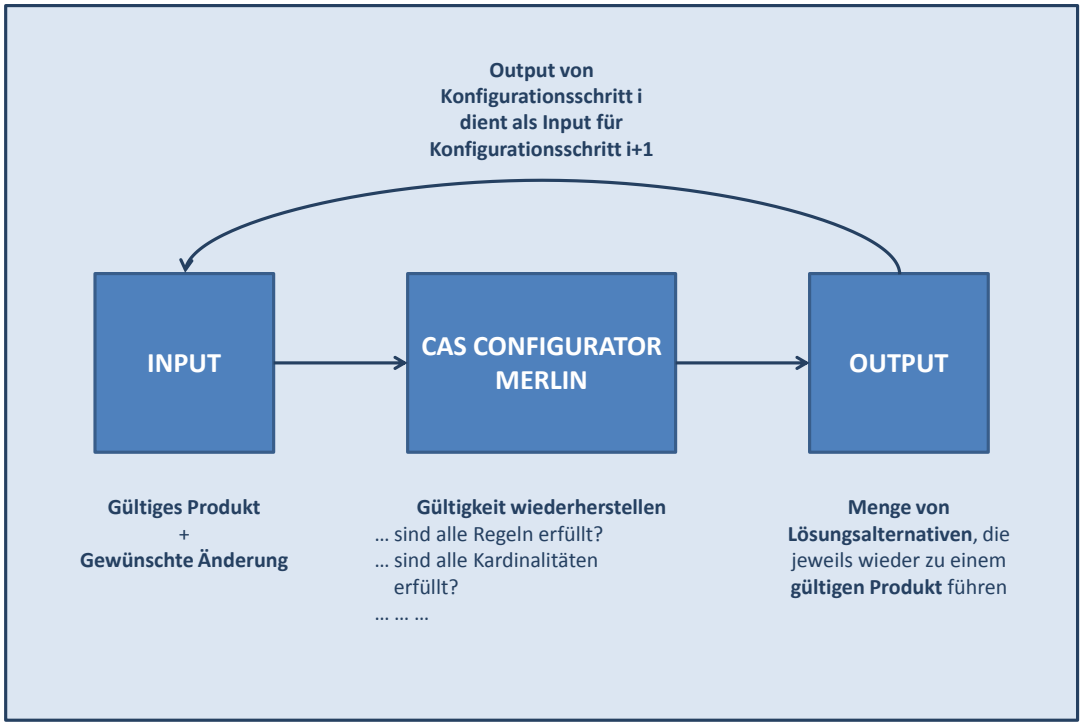


Abbildung 23: Ablauf eines Konfigurationsschrittes am Beispiel des CAS Configurator Merlin

5.1.2 Lösungsalternativen

Wenn durch einen bestimmten Konfigurationsinput des Anwenders der Raum der gültigen Produkte verlassen wird, ist es Aufgabe des Konfiguratorkerns *XPS Merlin*, durch gezielte Änderungen am konfigurierten Produkt, nämlich dem Hinzufügen und Weglassen von Elementen, einen Weg zurück in den gültigen Lösungsraum zu finden. Diese Änderungsoperationen spiegeln sich in verschiedenen Lösungsalternativen wider (siehe Abbildung 24). Da für verschiedene Lösungsalternativen auch unterschiedliche Umbaumaßnahmen vorgenommen werden, kann man die *Kosten* bzw. den Aufwand eines solchen Umbaus berechnen und mit diesem Maß die einzelnen Alternativen bezüglich des Änderungsaufwandes vergleichen. Dem Konfiguratorkern *XPS Merlin* können auch bestimmte Kostenfunktionen übergeben werden. Je geringer die *Kosten* eines Umbaus sind, desto ähnlicher ist das neue Produkt dem ursprünglichen *Input*. Ein Umbau eines ganzen Vetriebsbausteines ist zum Beispiel

teurer als die Änderung einer einzelnen Merkmalsausprägung. Aber auch unterschiedliche Merkmalsumbauten haben verschieden hohe Kosten.

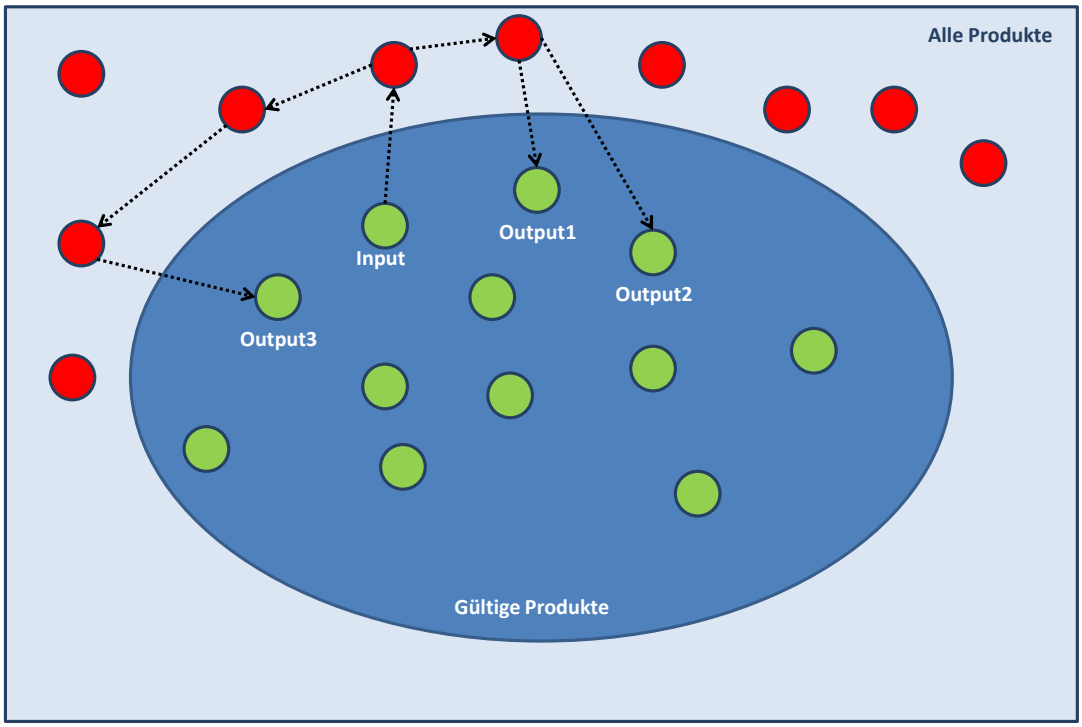


Abbildung 24: Visualisierung der Lösungsalternativen

5.2 Stand heute

In Abbildung 25 ist das herkömmliche Konzept des Alternativendialogs abgebildet. Die einzelnen Alternativen werden unter Angabe der hinzukommenden bzw. wegfallenden Elemente/Bauteile aufgelistet. Ein Vergleich der verschiedenen Alternativen untereinander ist möglich. Beim Auftreten von wenigen Alternativen ist der herkömmliche Alternativendialog eine akzeptable Lösung zur Darstellung der Alternativen. Soll aber eine größere Anzahl möglicher alternativer Konfigurationen angezeigt werden, so wird der Dialog schnell unüberschaubar. Aber auch eine kleine Anzahl von Alternativen kann bereits zu einer unübersichtlichen Darstellung führen, wenn die einzelnen Alternativen jeweils aus vielen ein- bzw. auszubauenden Elementen bestehen.

Ein weiteres Problem des aktuellen Alternativendialogs besteht darin, dass auch semantisch irrelevante Merkmale angezeigt werden. Außerdem werden wichtige und weniger wichtige Elemente vermischt dargestellt.

Zur Verbesserung der Übersichtlichkeit der Darstellung und zur Einhaltung des Grundsatzes der Aufgabenangemessenheit[22], werden im folgenden erweiterte Konzepte zur intelligenten Benutzerführung innerhalb des Alternativendialoges vorgestellt und die prototypische Implementierung eines Ansatzes angeführt.

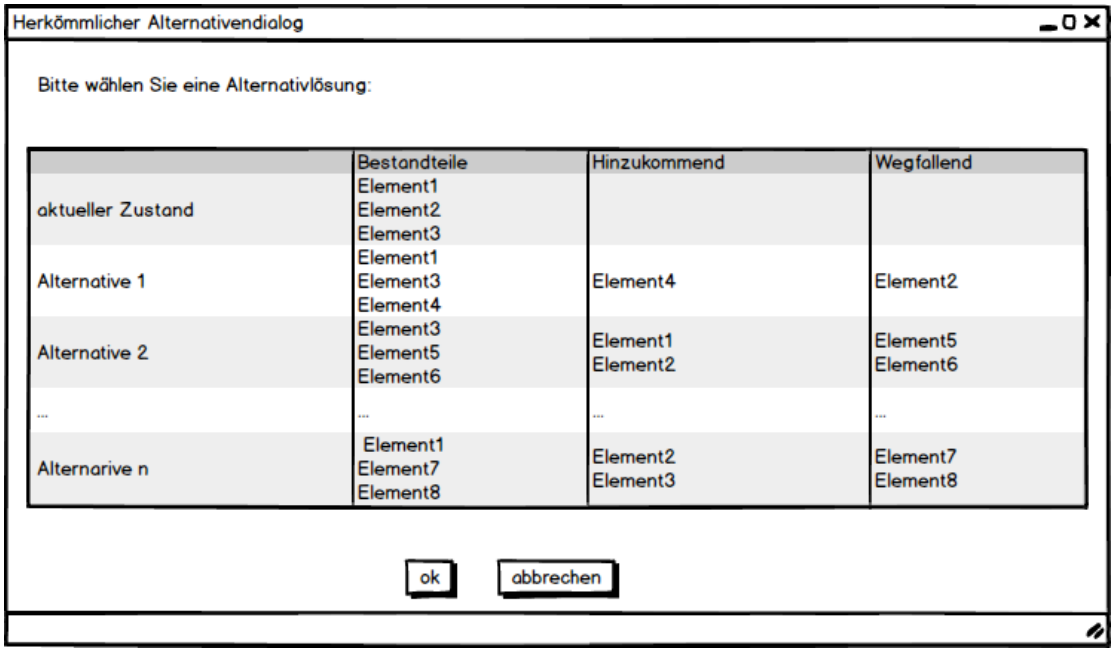


Abbildung 25: Herkömmlicher Alternativendialog des CAS Configurator Merlin

5.3 Konzepte einer intelligenten Benutzerführung

In den nächsten beiden Unterkapiteln werden zwei Konzepte zur benutzerfreundlichen Darstellung und Auswahl einer alternativen Konfiguration vorgestellt.

5.3.1 Auswahl über Präferenzfragen

Beim Konzept der Präferenzfragen werden dem Anwender vor der eigenlichen Konfiguration Präferenzfragen bezüglich seiner subjektiven Anforderungen gestellt. Dafür

[22] siehe Kapitel 3.4.1

müssen vorerst geeignete produktspezifische Eigenschaften definiert werden und die einzelnen Bauteile den verschiedenen Charakteristiken zugeordnet werden.

Im Beispielfall des PC-Konfigurators sind unter anderem folgende Eigenschaften möglich:

- leistungsstark
- energieeffizient
- leicht
- portabel
- viel Speicherplatz
- preiswert
- usw.

Tritt nun ein Konflikt auf, der den Benutzer zur Auswahl eines Prozessors zwingt, so wird bei der Präferenz für ein leistungsstarkes System wohl zum Beispiel der Einbau eines i7-Prozessors empfohlen, wobei das System eher zu einem sparsameren i3-Prozessor rät, wenn die Präferenz auf Energieeffizienz liegt.

5.3.2 Der Alternativen-Wizard

Zur benutzerfreundlichen Auswahl der passenden Alternative soll dem Anwender ein Alternativen-Wizard[23] zur Verfügung stehen. Ziel hierbei ist es, die Menge möglicher Alternativen durch gezielte Fragen über hinzukommende und wegfallende Elemente einzugrenzen bis nur noch eine passende Alternative übrig ist. Der Benutzer kann aber auch den Wizard beenden und zur Direktauswahl der Alternativen wechseln, wobei er sich dann wieder im herkömmlichen Alternativendialog befindet.

Damit der Benutzer hier nicht wieder von einer unüberschaubaren Menge an Fragen überfordert wird, soll ein Priorisierungskonzept eingeführt werden. Durch die Angabe einer Priorität für jedes Element dritter Hierarchieebene im *Regeleditor* kann die Reihenfolge und die Relevanz der gestellten Fragen vom Produktmodellierer gesteuert werden.

[23] Der Begriff *Wizard* wird gleichbedeutend mit dem Ausdruck *Assistent* verwendet und stellt eine Oberfläche dar, die den Anwender über eine nach ISO 9241-110 ergonomische Dateneingabe (siehe auch Kapitel 3.4) durch den Dialog führt.

5.4 Spezifikation des Alternativen-Wizard

5.4.1 Grafischer Entwurf des Alternativen-Wizard

In Abbildung 26 ist ein grafischer Entwurf des *Alternativen-Wizards* am Beispiel eines PC-Konfigurators dargestellt. Nachdem bei der Konfiguration im *intelligenten Formular* durch den User alternative Konfigurationen aufgetreten sind, erscheint der *Alternativen-Wizard* im Browserfenster des Anwenders.

Der Fortschrittsbalken in der rechten oberen Ecke zeigt den Fortschritt des Users bei der Beantwortung der Fragen (im Beispiel wird die erste von fünf Fragen beantwortet). Dieser Fortschrittsbalken ist eines der in Kapitel 3.6 vorgestellten *Interaction Design Pattern*.

Im mittleren Bereich des Dialogs wird die jeweilige Frage und eine Drop-down-Liste zur Auswahl der Antwort angezeigt. Rechts neben der Dropdown-Liste befindet sich das *Info-Symbol*, das kontext-sensitive Hilfe für den Anwender bereithält und somit sowohl die Lernförderlichkeit[24], als auch die Fehlertoleranz[25] des Dialogs verbessert.

Unten im Dialogfenster befindet sich die Steuerleiste. Diese gibt dem Anwender die Möglichkeit, einen Schritt zurückzuspringen, direkt zur Alternativenauswahl (herkömmlicher Alternativendialog) zu wechseln und seine Auswahl zu bestätigen und fortzufahren. Diese Steuerleiste erfüllt den Grundsatz der Steuerbarkeit[26].

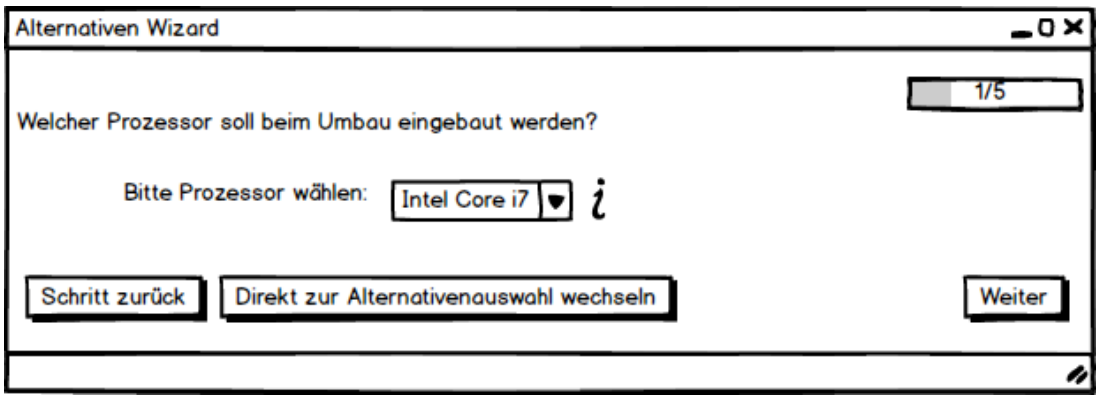


Abbildung 26: Wizard zur intelligent geführten Alternativenauswahl

[24] siehe auch Kapitel 3.4.4
[25] siehe auch Kapitel 3.4.6
[26] siehe auch Kapitel 3.4.5

5.5 Implementierung

5.5.1 Erweiterung des Regeleditors

Die herkömmliche Pflegeoberfläche des Regeleditors wurde erweitert und durch Eingabemöglichkeiten für weitere Metadaten für Elemente ergänzt (siehe Abbildung 27). Die Benutzeroberfläche wird automatisch aus der modellierten Produktstruktur und den Metadaten aus dem Regeleditor generiert.

Hinzugekommene Metadaten sind:

- Prioritätsangabe
- Fragestring
- Tooltip
- Pfad zum Bild des Elements

```
metadata.add(new StringEnumMetadataProperty(getNextMetadataId(), PRIORITY, "Priorität",
          CATEGORY_META_DATA, "1", Arrays.asList((new String[] { "1", "2", "3" }))));
metadata.add(new StringMetadataProperty(getNextMetadataId(), WIZARD_QUESTION, "Frage",
          CATEGORY_META_DATA, ""));
metadata.add(new StringMetadataProperty(getNextMetadataId(), TOOLTIP_INFO, "Tooltip",
          CATEGORY_META_DATA, ""));
metadata.add(new StringMetadataProperty(getNextMetadataId(), IMAGE_PATH, "Pfad zum Foto",
          CATEGORY_META_DATA, ""));
```

Abbildung 27: Code zur Erweiterung des Regeleditors

Prioritätsangabe In der prototypischen Implementierung wurde eine dreistufige Prioritätsskala umgesetzt, wobei Priorität 1 der höchsten und Priorität 3 der niedrigsten Priorität entspricht. Es sind aber auch granularere Skalen denkbar. Im *Alternativen-Wizard* werden die Fragen in absteigender Prioritätsreihenfolge beantwortet, wobei in der Regel die Beantwortung von Fragen erster und zweiter Priorität bereits ausreichen, um zu einem eindeutigen Ergebnis zu kommen.

Fragestring Über die Eingabe eines individuellen Fragestrings als Metadatum für die einzelnen Bauteile im Regeleditor werden die einzelnen Fragen definiert, die im *Alternativen-Wizard* abgefragt werden.

Tooltip Damit der Anwender effektiv durch die Konfiguration geleitet und dabei Unsicherheit und Irreführung vermieden werden kann, wurde ein weiteres Metadatum zur Eingabe eines Tooltips eingefügt. Dadurch wird das Konzept der kontextsensitiven Hilfe[27] verwirklicht und somit auch die Grundsätze der Selbstbeschreibungsfähigkeit[28] und Lernfoerderlichkeit[29] erfüllt.

Pfad zum Bild des Elements Weiterhin kann für jedes Element bzw. Bauteil ein passendes Bild hinterlegt werden, um die Konfiguration zu visualisieren. Dadurch erhält der Anwender einen besseren Eindruck von seinem konfigurierten Produkt, was Vertrauen fördert und Missverständnisse reduziert.

5.5.2 Erweiterung der ConfigurationBean

Für die Umsetzung des *Alternativen-Wizard* wurde die *ConfigurationBean* unter anderem um die Methode *getWizardQuestion()* erweitert (siehe Abbildung 29). In der prototypischen Implementierung des *Alternativen-Wizard* werden nur alternativenrelevante hinzukommende Elemente erster Prioritätsstufe betrachtet, da ein Beispiel mit einer relativ kleinen Regelwelt und einer schlanken Produktstruktur vorlag. Deshalb konnten die Konfigurationskonflikte meist schon mit nur ein bis zwei Fragen gelöst werden, was hier eine Priorisierung der Fragen überflüssig machte.

Die Methode *getWizardQuestion()* bestimmt die für die Fragen im *Alternativen-Wizard* relevanten Elemente und ordnet diese nach dem Grad ihrer Verschiedenheit zueinander. Je mehr unterschiedliche Ausprägungen ein Element dritter Hierarchieebene unter den möglichen Alternativen hat, desto stärker ist dessen Unterscheidungskraft und desto schneller führt die Beantwortung eines solchen Elements zu einer eindeutigen Lösung des Konfliktes.

Hierfür ein einfaches Beispiel (siehe Abbildung 28): Nach einem Konfigurationsschritt durch einen Benutzer kommt es zum Konfliktfall und es exis-tieren drei alternative Konfigurationen. Der *Alternativen-Wizard* würde nun nacheinander drei Fragen stellen, um eine Entscheidung bezüglich des Einbaus von Prozessor, Speicher und Display treffen zu können. Ziel ist es, durch die Beantwortung möglichst weniger Fragen zu einer eindeutigen Lösung zu kommen. Deshalb spielt die Reihenfolge der abgefragten Elemente eine Rolle. Die Chance auf eine schnelle Lösungsfindung

[27] siehe auch Kapitel 3.3.3
[28] siehe auch Kapitel 3.4.2
[29] siehe auch Kapitel 3.4.4

steigt mit der Unterscheidungskraft der einzelnen Elemente. Die Frage nach der Ausprägung des Prozessors ist im Beispiel irrelevant. Da jede der drei Alternativen die gleiche Prozessor-ausprägung einbaut, beträgt die Wahrscheinlichkeit, dass der Konflikt durch die Frage nach der Prozessorausprägung gelöst werden kann, 0%. Etwas besser verhält es sich mit der Frage nach dem Speicher. Während *Alternative 1* und *Alternative 2* den selben Speicher einbauen, unterscheidet sich *Alternative 3* in der Ausprägung und es besteht eine Chance von 33%, dass eine einzige Frage ausreicht, um den Konflikt zu lösen - nämlich wenn der Benutzer sich für die Ausprägung *Speicher2* entscheidet. Die höchste Unterscheidungskraft liegt in der Displayausprägung. Da jede der drei möglichen Alternativen ein anderes Display einbaut, kann der Konflikt durch die Frage nach der Displayausprägung mit einer Wahrscheinlichkeit von 100% gelöst werden.

Die Abfragereihenfolge ist also:

1. Displayausprägung
2. Speicherausprägung
3. Prozessorausprägung (wird aber tatsächlich nicht abgefragt, da irrelevant)

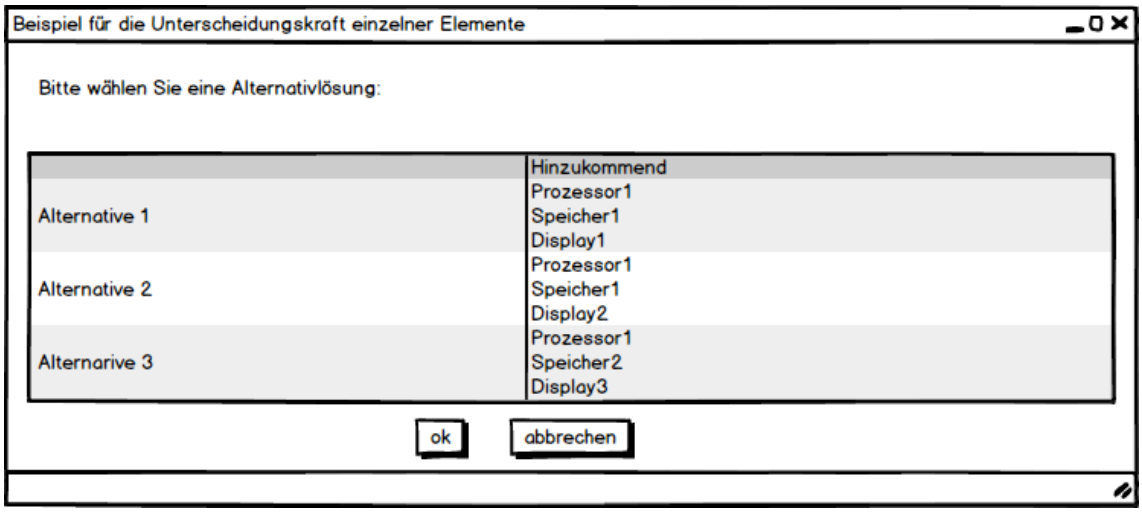


Abbildung 28: Unterscheidungskraft der einzelnen Elemente

Um wieder zur Beschreibung der *ConfigurationBean* zurückzukehren wird zuerst die *elementsMap* erläutert. Die *elementsMap* ist eine *HashMap*[30], deren *Keys*

[30] siehe java.util.HashMap⟨K,V⟩

alternativenrelevante Elemente dritter Hierarchieebene sind. Die korrelierenden *Values* sind alle für die Alternativen relevanten Ausprägungen eines *Keys* vom Typ *AbstractProductStructureElement* - also dessen Kindknoten - die in einer *ArrayList* gehalten werden.

Der *Integer*-Wert[31] *gradeOfDifference* stellt den oben beschriebenen Grad der Unterscheidungskraft dar und ist initial mit seinem maximalen Wert, nämlich mit *alternativeList.size()*, was der Anzahl möglicher Alternativen entspricht, belegt.

Für jede mögliche Alternative werden die hinzukommenden Elemente erster Prioritätsstufe ermittelt und in Form einer *ArrayList* als *Value* zur jeweiligen Alternative als *Key* in die *buildInMap* gelegt. Anschließend wird von jedem hinzukommenden Element aus der *buildInMap* der Vaterknoten, also das zuhegörige Element dritter Hierarchieebene bestimmt. Dieser Vaterknoten wird entweder als *Key* mit der *ArrayList* seiner Kindknoten als *Value* in die *elementsMap* eingefügt - sofern dort noch nicht vorhanden - oder falls in dieser schon enthalten, wird die *ArrayList* seiner Kindknoten um das neue hinzukommende Element ergänzt.

Abschließend werden alle alternativenrelevanten Elemente dritter Hierarchieebene absteigend nach dem Grad ihrer Unterscheidungskraft zur *ArrayList questionRelevantElements* hinzugefügt. Alle diese Elemente werden - sofern in der *elementsMap* enthalten - als *Key* in die *questionMap* gelegt und erhalten als *Value* die *ArrayList* ihrer Ausprägungen aus der *elementsMap*.

Die *questionMap* stellt die Grundlage für das gezielte Stellen der Fragen im *Alternativen-Wizard* dar.

[31] siehe java.lang.Integer

```
public void getWizardQuestions() {
            Map<AbstractProductStructureElement, List<AbstractProductStructureElement>> elementsMap;
            elementsMap = new HashMap<AbstractProductStructureElement, List<AbstractProductStructureElement>>();
            int gradeOfDifference = alternativeList.size();
            for (IConfigurationWithAlternativeSupport i : alternativeList) {
                  buildInMap.put(i, showElementsToBuildInWithPrioOne(i));
                  buildOutMap.put(i, showElementsToBuildOutWithPrioOne(i));
            }
            for (IConfigurationWithAlternativeSupport i : alternativeList) {
                  for (AbstractProductStructureElement apse : buildInMap.get(i)) {
                        AbstractProductStructureElement parent =
                              (AbstractProductStructureElement) apse.getParent();
                        if (!elementsMap.containsKey(parent) && parent.isVisible()) {
                              List<AbstractProductStructureElement> tmp =
                                          new ArrayList<AbstractProductStructureElement>();
                              tmp.add(apse);
                              elementsMap.put(
                                          (AbstractProductStructureElement) apse.getParent(),
                                          tmp);
                        } else if (elementsMap.containsKey(parent)
                                    && parent.isVisible()) {
                              List<AbstractProductStructureElement> test = elementsMap
                                          .get(apse.getParent());
                              if (!test.contains(apse)) {
                                    test.add(apse);
                              }
                              elementsMap.put(
                                          (AbstractProductStructureElement) apse.getParent(),
                                          test);
                        }
                  }
            }
            for (int i = gradeOfDifference; i >= 0; i--) {
                  for (IConfigurationWithAlternativeSupport iConf : alternativeList) {
                        for (AbstractProductStructureElement apse : buildInMap
                                    .get(iConf)) {
                              AbstractProductStructureElement parent =
                                    (AbstractProductStructureElement) apse.getParent();
                              if (parent.isVisible()
                                          && elementsMap.get(parent).size() > 1
                                          && parent.getPriority().equals("1")
                                          && elementsMap.get(parent).size() == i
                                          && !questionRelevantElements.contains(parent)) {
                                    questionRelevantElements.add(parent);
                              }
                        }
                  }
            }
            for (AbstractProductStructureElement apse : questionRelevantElements) {
                  if (elementsMap.containsKey(apse)) {
                        questionMap.put(apse, elementsMap.get(apse));
                  }
            }
      }
```

Abbildung 29: Code zur Erweiterung der *ConfigurationBean* zur Darstellung des Alternativen Wizard

6 Evaluation der Ergebnisse

Zur Evaluation der Ergebnisse dieser Bachelorarbeit (*intelligentes Formular* und *Alternativen-Wizard*) wurden zwei unterschiedliche Methoden herangezogen. Zum einen wurde ein kurzes Interview über die neuen Konzepte mit dem Projektleiter für den Kunden *EnBw* abgehalten (siehe Anhang B). Zum anderen wurde zusätzlich ein Usability-Test zur Bewertung der neuen GUI-Konzepte durchgeführt.

6.1 Interview mit Kundenbetreuer

Im Interview mit dem Projektleiter Björn-Oliver Hartmann wurden vier einfache Fragen gestellt:

1. "Welche Verbesserungen bringt der neue Alternativen-Wizard im Bezug auf die benutzerfreundliche Dialogführung mit sich?"
2. "Glauben Sie, dass der neue Alternativen-Wizard benutzerinitiierte Prozessabbrüche wegen Überforderung aufgrund einer zu großen Anzahl an Alternativen reduzieren kann?"
3. "Was gefällt Ihnen bzw. dem Kunden besonders gut?"
4. "Was sollte in Zukunft noch verändert bzw. ergänzt werden?"

Zusammenfassend kann man folgendes Fazit aus dem Interview ziehen: Die neuen Konzepte verbessern die Übersichtlichkeit der Oberfläche und bieten die Möglichkeit einer schnelleren Auswahl einer Alternative. Dadurch wird sich die Zahl der benutzerinitiierten Prozessabbrüche sicherlich reduzieren. Besonders gefallen haben der Fortschrittsbalken und die Chance, einen Konfliktfall mit wenigen einfachen Fragen zu lösen. Für die Zukunft wird die intensivere Verwendung von Produktbildern und Icons gewünscht.

6.2 Usability-Testcase

6.2.1 Das AttrakDiff2-Verfahren

Für die Bewertung der Qualität des *intelligenten Formulars* und des *Alternativen-Wizards* wurde das Bewertungswerkzeug *AttrakDiff2*[32] verwendet. Dieses Bewertungsverfahren zählt zur Kategorie der semantischen Differentiale.

> "Bei einem semantischen Differential werden die Befragten gebeten, ein Einstellungsobjekt anhand von Adjektiven zu beschreiben. Diese sind bipolare Gegensatzpaare mit skalierten Abstufungen." [MS10]

[32] siehe [HBK08]

Für das *AttrakDiff2-Verfahren* werden 28 solcher siebenstufiger Gegensatzpaare abgefragt, die in vier verschiedene Skalen gruppiert werden. Diese vier Skalen sind:

- Pragmatische Qualität (PQ)

  > "Die wahrgenommene Fähigkeit eines Produkts, Handlungsziele zu erreichen, indem es nützliche und benutzbare Funktionen bereitstellt." [HBK08]

- Hedonische Qualität - Stimulation (HQS)

  > "Die Fähigkeit eines Produkts, das Bedürfnis nach Verbesserung der eigenen Kenntnisse und Fertigkeiten zu befriedigen." [HBK08]

- Hedonische Qualität - Identität (HQI)

  > "Die Fähigkeit eines Produkts, relevanten Anderen selbstwertdienliche Botschaften zu kommunizieren." [HBK08]

- Attraktivität (ATT)

  > "Globale positiv-negativ Bewertung des Produkts." [HBK08]

Grundsätzlich wird also zwischen pragmatischer und hedonischer Qualität unterschieden, wobei sich die pragmatische Qualität darauf konzentriert, was man mit einem bestimmten Produkt tut und die hedonische Qualität darauf, wofür ein Produkt steht und was es symbolisiert.

6.2.2 Die Testpersonen

Die Testpersonen für den Usability-Test waren acht Kollegen aus der Softwareentwicklung bei der *CAS Software AG* in Karlsruhe. Das *intelligente Formular* und der *Alternativen-Dialog* war manchen bereits bekannt, für einige aber war der Usability-Test der erste Berührungspunkt mit den neuen Konzepten.

6.3 Auswertung des Usability-Tests

Für die Auswertung des Usability-Tests werden im Folgenden drei unterschiedlich granulare Diagramme - einmal mit Blick auf die einzelnen Wortpaare, dann mit Fokus auf den Mittelwerten für die vier Bewertungskategorien und zuletzt eine Übersicht in Portfoliodarstellung - angeführt.

6.3.1 Das Profil der Wortpaare

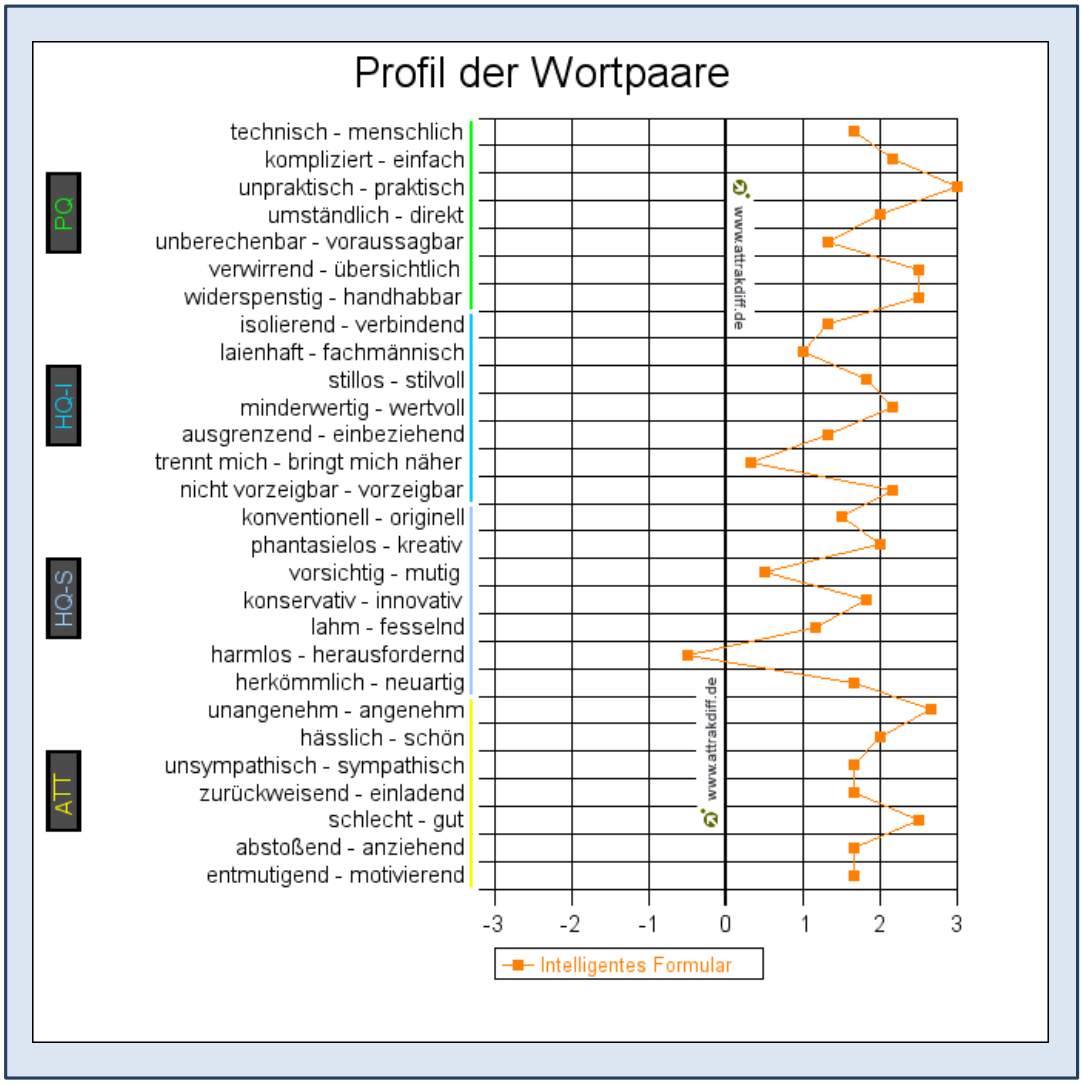


Abbildung 30: Profil der Wortpaare

In Abbildung 30 ist die Übersicht der durchschnittlichen Bewertungen der einzelnen Wortpaare auf der siebenstufigen Skala dargestellt. Von oben nach unten sind die verschiedenen Bewertungskategorien *Pragmatische Qualität*, *hedonische Qualität - Simulation*, *hedonische Qualität - Identität* und *Attraktivität* zu erkennen. Für fast alle Begriffspaare liegt das Ergebnis auf der positiven Seite der Skala, was dafür

spricht, dass das *intelligente Formular* und der *Alternativen-Dialog* bei den Testpersonen Gefallen findet und als benutzerfreundlich empfunden wird.

Als Ausreißer kann man das Begriffspaar *harmlos* und *herausfordernd* ansehen.

6.3.2 Das Diagramm der Mittelwerte

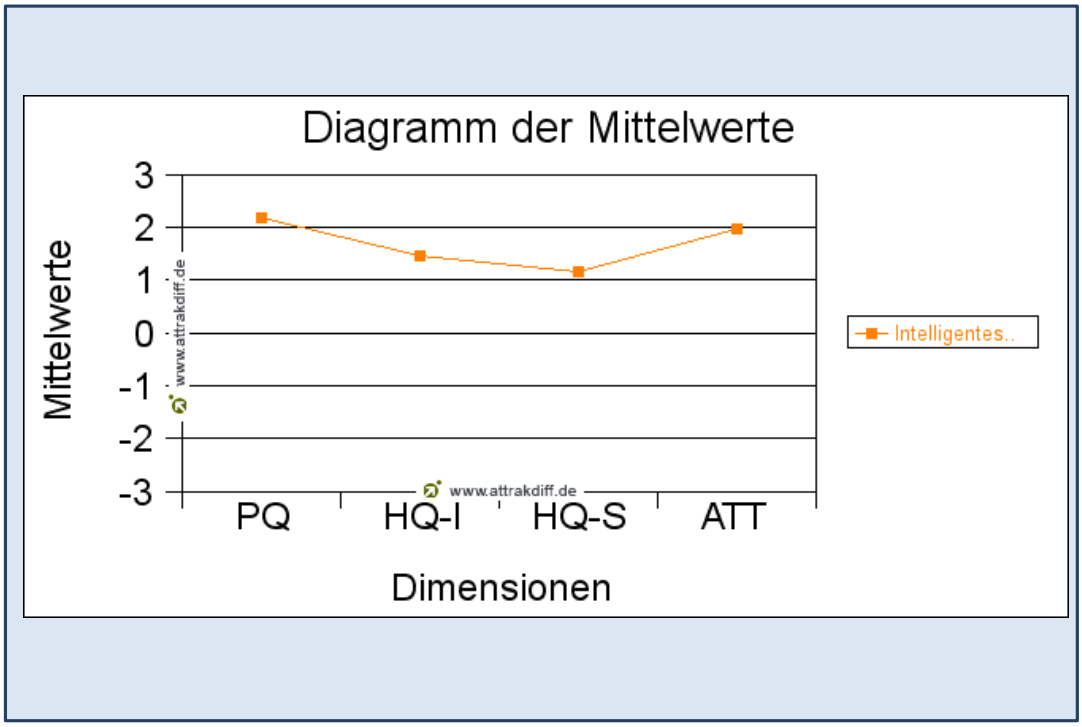


Abbildung 31: Diagramm der Mittelwerte

Abbildung 31 stellt die Mittelwerte für die vier Bewertungskategorien dar. Es fällt auf, dass die *pragmatische Qualität* mit einem Wert größer zwei am besten bewertet wurde, was für einen hohen Nutzen und eine gute Benutzbarkeit des *intelligenten Formulars* und des *Alternativen-Wizard* spricht.

Die *hedonische Qualität* ist etwas niedriger, aber dennoch im guten positiven Bereich bewertet. Da sich Hedonisches auf Bedürfnisse konzentriert und die Emotionen des Benutzers im Vordergrund stehen, kann man sagen, dass das *intelligente*

Formular und der *Alternativen-Wizard* den Anwender in geringerem Ausmaß emotional bewegen, als dass ihn der pragmatische Nutzen beeindruckt. Trotzdem kommt durch den positiven Wert eine gewisse Bindung zwischen dem Benutzer und der Benutzeroberfläche zustande.

Der Wert für die generelle *Attraktivität* der Oberfläche ist mit einer Bewertung von zwei auch äußerst positiv, wodurch das gesamte Produkt gut bei den Testpersonen ankam.

6.3.3 Das Ergebnis als Überblick in Portfoliodarstellung

In Abbildung 32 ist das Ergebnis des Usability-Tests in Portfoliodarstellung abgebildet. Die mittlere Ausprägung der Dimensionen für das *intelligente Formular* und den *Alternativen-Wizard* ist erfreulicherweise im Bereich *begehrt* angesiedelt. Das spricht insgesamt für eine sehr hohe pragmatische und hedonische Qualität. Das *Konfidenz-Rechteck* zeigt unter den Testpersonen eine höhere Übereinstimmung für die *pragmatische Qualität* als für die *hedonische Qualität*. Das *Konfidenz-Rechteck* geht leicht über den Bereich *begehrt* hinaus, wobei man trotzdem noch von einem *begehrten* Produkt sprechen kann.

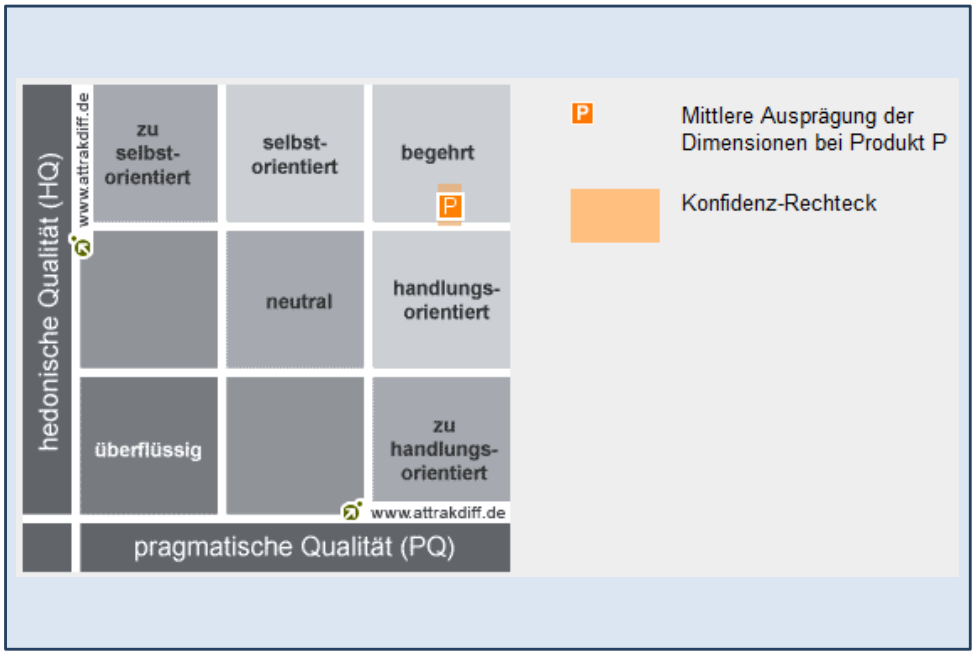


Abbildung 32: Übersicht über das Testergebnis in Portfoliodarstellung

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

Ziel dieser Arbeit war es, eine optimale Benutzeroberfläche zur Darstellung komplexer Produktkonfigurationen mit Schwerpunkt auf der Anzeige und der Auswahl von alternativen Konfigurationsschritten zu entwerfen und prototypisch zu implementieren.

Zu diesem Zweck wurden zuerst grundlegende Aspekte der Usability für Produktkonfiguratoren untersucht und in die neuen Konzepte des *intelligenten Formulars* und des *Alternativen-Wizards* basierend auf dem *CAS Configurator Merlin* eingearbeitet.

Was die Qualität und die Benutzerfreundlichkeit der im Rahmen dieser Arbeit entwickelten Konzepte betrifft, so konnte anhand eines Interviews und eines Usability-Tests ein durchaus positives Ergebnis bestätigt werden, was das Problem der benutzerinitiierten Prozessabbrüche wegen Unsicherheit und Unentschlossenheit entschärfen wird.

Andererseits musste festgestellt werden, dass im Bereich der visuellen Konfiguration mit 3D-Darstellungen oder Renderbildern noch viel Verbesserungspotential vorliegt. Zwar kann nun durch eine Erweiterung der Pflegeoberfläche des Regeleditors für jedes Element ein individuelles Bild hinzugefügt werden, doch kann die bildliche Darstellung von komplexen Konfigurationen mit dem vorliegenden Konzept nicht ermöglicht werden.

Insgesamt kann festgehalten werden, dass die Frage, ob mit den entwickelten Konzepten eine optimale Benutzeroberfläche für die Produktkonfiguration geschaffen werden konnte, differenziert beantwortet werden muss. Einerseits bringen die neuen Ansätze eine erhebliche Verbesserung der Usability und der Qualität der Benutzeroberfläche im Vergleich zur vorher verwendeten Benutzeroberfläche mit sich. Andererseits gibt es im Detail immer noch weitere Verbesserungsmöglichkeiten, die in der Zukunft umgesetzt werden sollten. Somit ist zu sagen, dass die Konzepte dieser Arbeit einen Weg hin zur optimalen Benutzeroberfläche für einen webbasierten Produktkonfigurator aufzeigen, diese jedoch in der individuellen Implementierung noch verfeinert und weiterentwickelt werden müssen.

7.2 Ausblick

Für die Zukunft gibt es in einigen Bereichen noch Potential, die Konzepte zu verfeinern und im Detail zu verbessern. Zum Beispiel wurden im Usability-Test zur Evaluation ausschließlich Softwareentwickler befragt. Interessant wäre aber auch die Meinung von Endkunden und Endanwendern. Die im Zuge dieser Arbeit entwickelten Konzepte sollten unbedingt durch Erkenntnisse eines solchen erweiterten

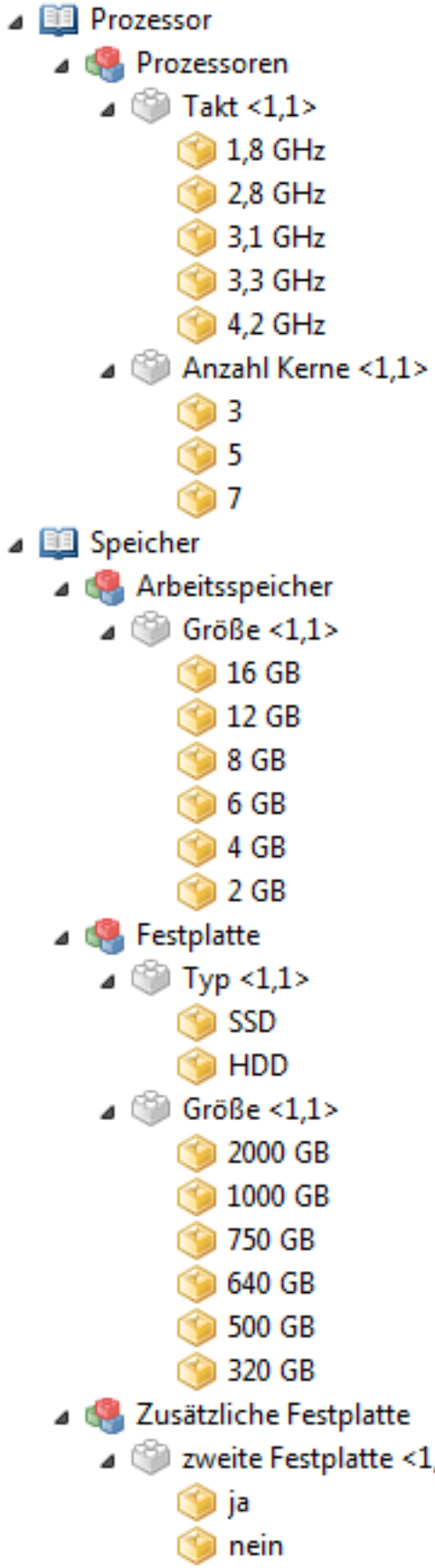
Usability-Tests ergänzt werden.

Außerdem wäre ein Test mit einer sehr großen Regelwelt und einer breiten Produktstruktur empfehlenswert. Dabei könnte man das Verhalten des *Alternativen-Wizards* im Bezug auf eine Vielzahl von Alternativen mit vielen hinzukommenden und wegfallenden Elementen testen. Dadurch würde auch das Priorisierungskonzept der Fragen besser zur Geltung kommen. Interessant wäre hier ein Mittelwert der benötigten Fragen, um einen Konfliktfall zu lösen. Mit diesem Mittelwert könnte man die Effektivität des *Alternativen-Wizards* im Vergleich zum herkömmlichen Alternativendialog einordnen.

Auch im Bereich des Feedbacks für den Anwender sind noch einige Verbesserungen denkbar. So sollte dem Benutzer zum Beispiel nach der Lösung eines Konflikts durch den *Alternativen-Wizard* ein detailliertes Feedback über alle ein- bzw. ausgebauten Elemente gegeben werden.

A Regelwerk des PC-Konfigurator Beispiels

A.1 Produktstruktur



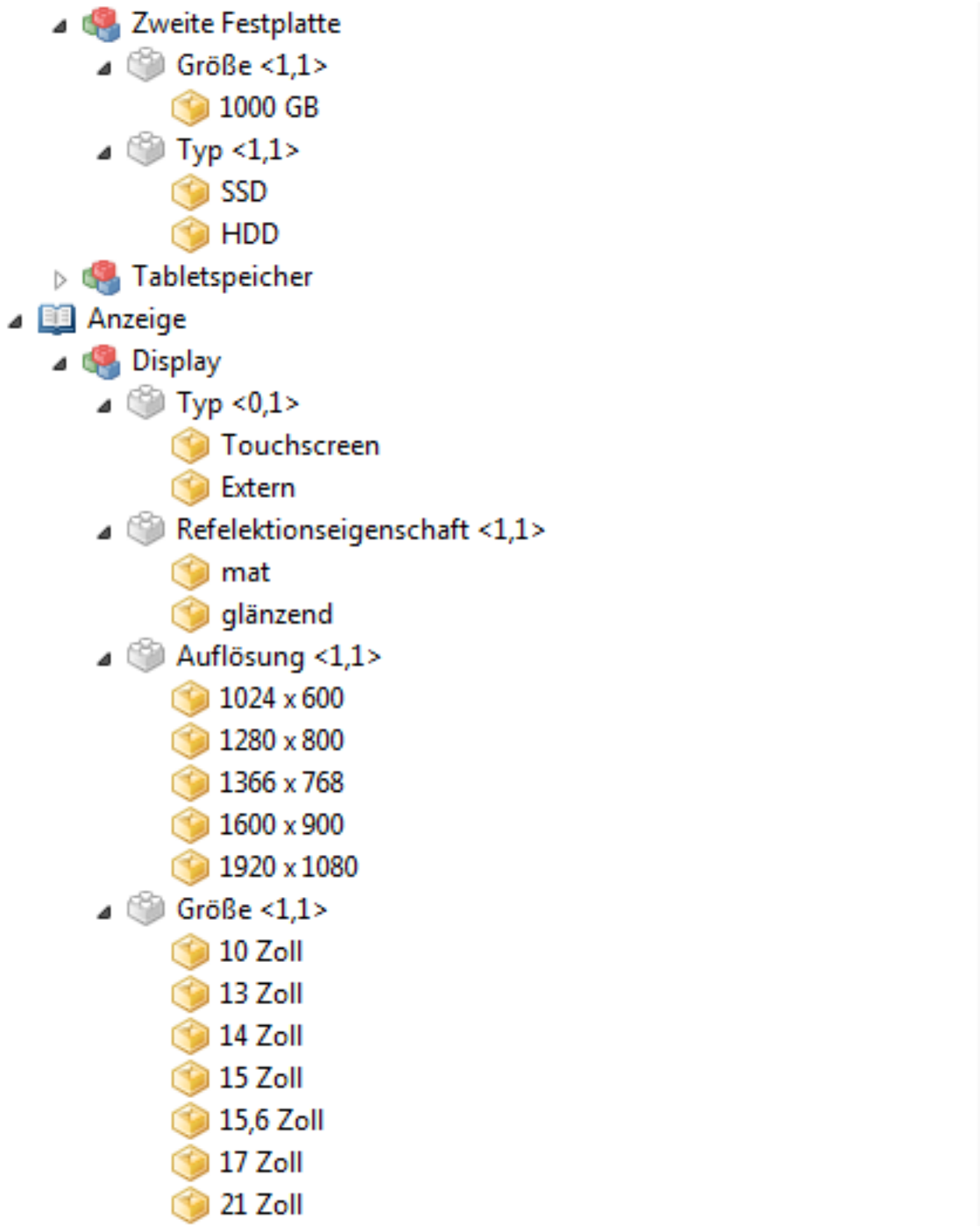


Abbildung 34: Produktstruktur PC-Konfigurator - Teil 2

Abbildung 35: Produktstruktur PC-Konfigurator - Teil 3

A.2 Regelwerk Laptop

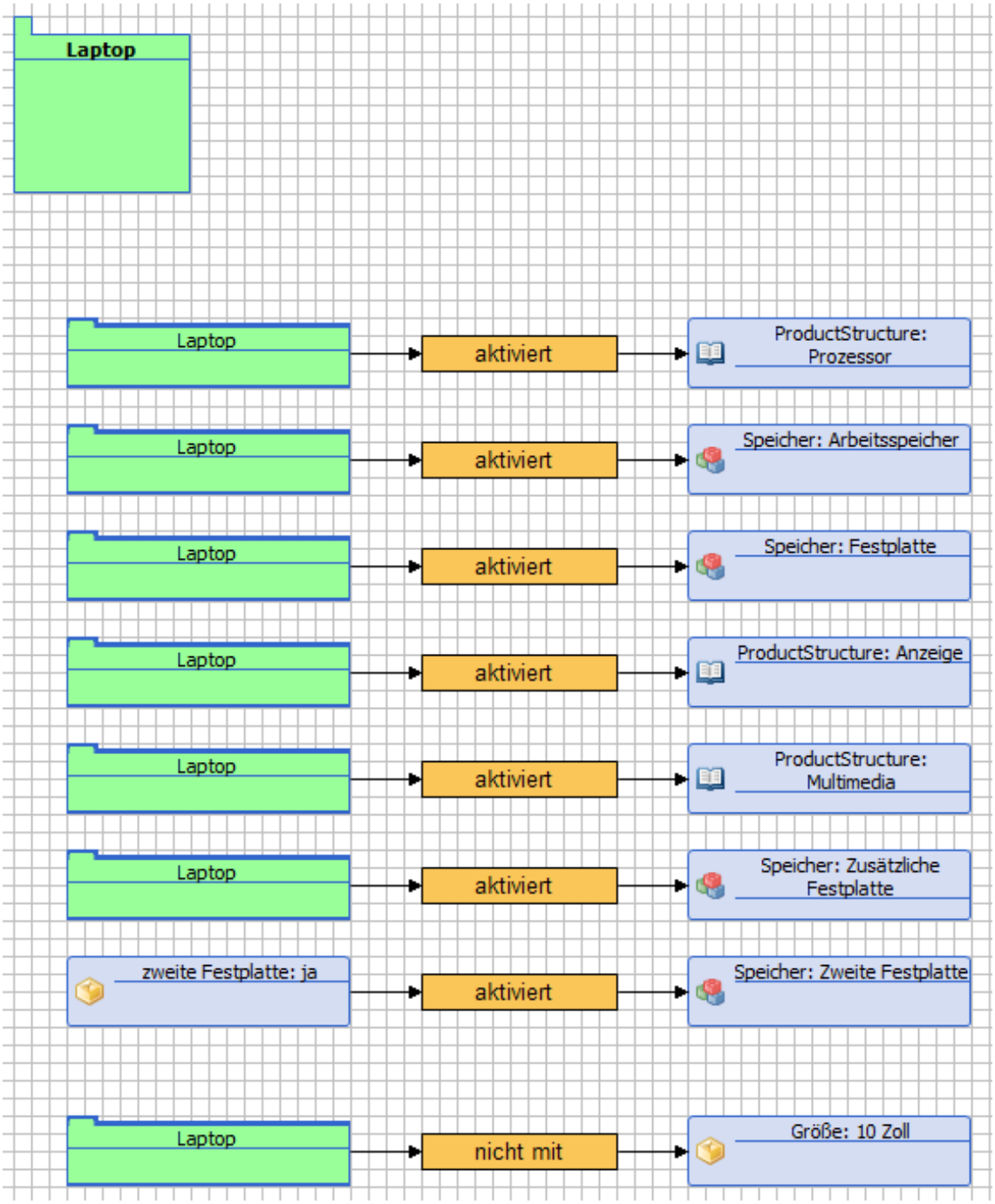


Abbildung 36: Regelwerk für das Vertriebsprodukt Laptop - Teil 1

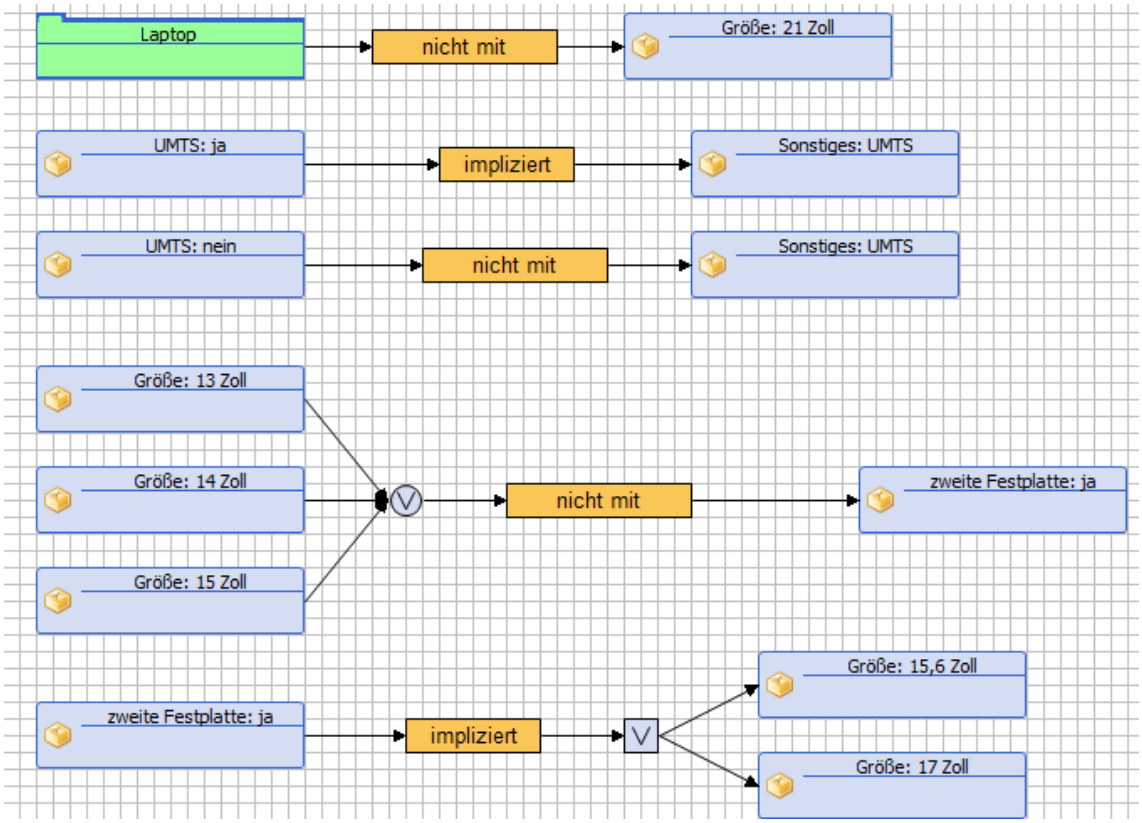


Abbildung 37: Regelwerk für das Vertriebsprodukt Laptop - Teil 2

A.3 Regelwerk PC

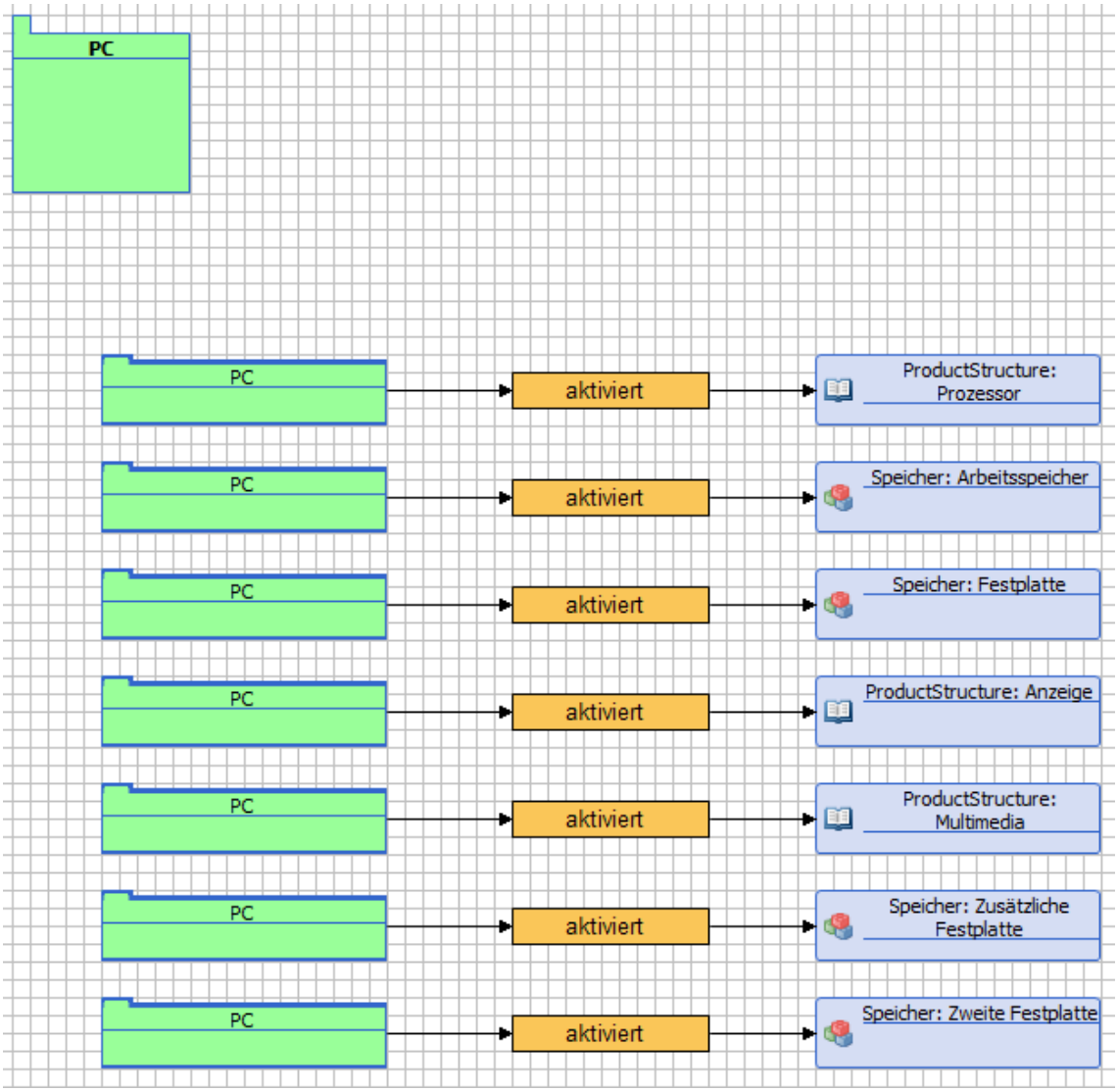


Abbildung 38: Regelwerk für das Vertriebsprodukt PC

A.4 Regelwerk Tablet

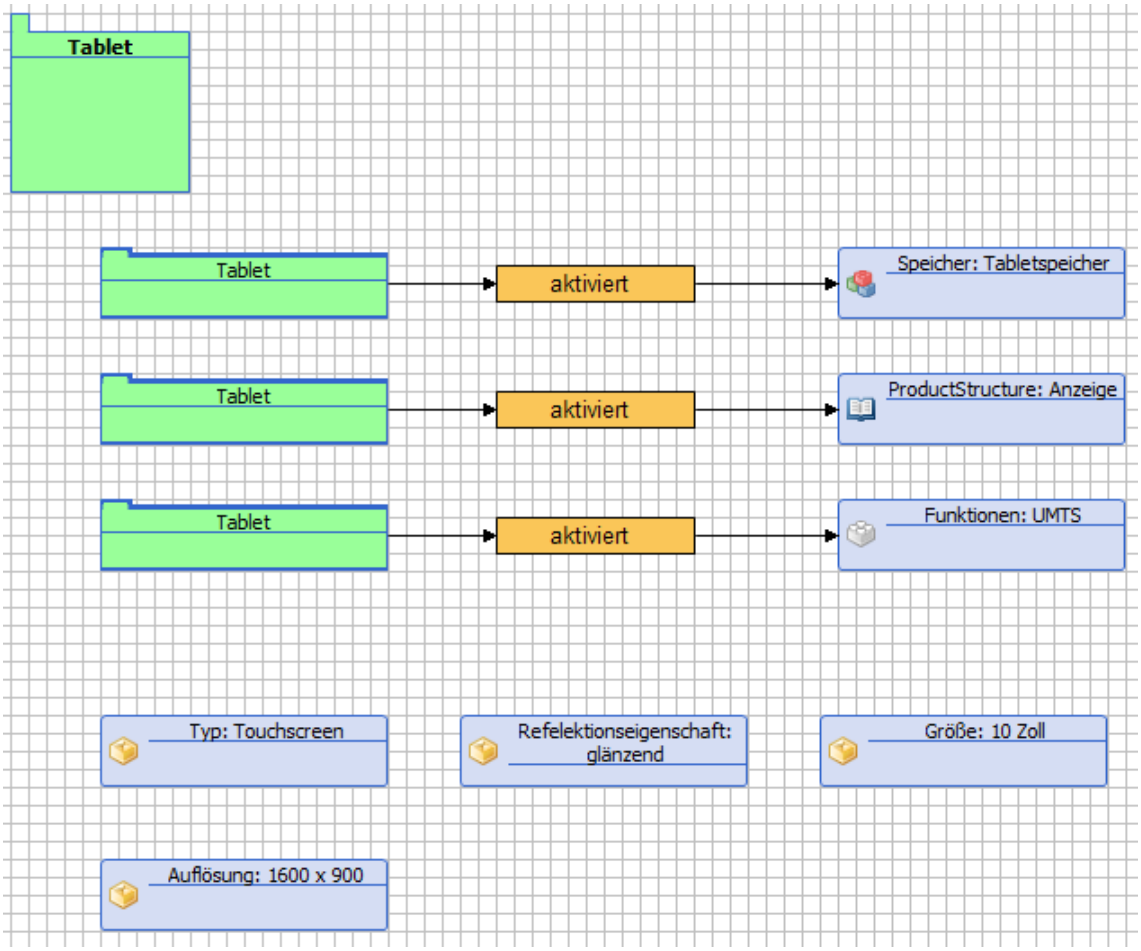


Abbildung 39: Regelwerk für das Vertriebsprodukt Tablet

B Interview mit Projektleiter

Patrick Kuglmeier: *Welche Verbesserungen bringt der neue Alternativen-Wizard im Bezug auf die benutzerfreundliche Dialogführung mit sich?*
Björn-Oliver Hartmann: *Der neue Alternativen-Wizard verbessert vor allem die Geschwindigkeit der Lösungsfindung und die Übersichtlichkeit. Man muss jetzt nicht mehr viele Alternativen mit vielen Elementen vergleichen. Ich finde es gut, dass man durch die Beantwortung einfacher Fragen zum Ziel kommt.*
Patrick Kuglmeier: *Glauben Sie, dass der neue Alternativen-Wizard benutzerinitiierte Prozessabbrüche wegen Überforderung aufgrund einer zu großen Anzahl an Alternativen reduzieren kann?*
Björn-Oliver Hartmann: *Sicherlich. Dem Kunden fällt es einfacher ein paar kurze Fragen zu beantworten, als eine lange Liste von verschiedenen Optionen zu überblicken.*
Patrick Kuglmeier: *Was gefällt Ihnen bzw. dem Kunden besonders gut?*
Björn-Oliver Hartmann: *Mir gefällt der Fortschrittsbalken ganz gut und die Navigation durch die Fragen über die Tabs. Am wichtigsten ist aber die bessere Übersichtlichkeit und dass der Kunde schneller zu einer Lösung kommt.*
Patrick Kuglmeier: *Was sollte in Zukunft noch verändert bzw. ergänzt werden?*
Björn-Oliver Hartmann: *Mir würden noch mehr Bilder und Icons gefallen.*

Literatur

[CM02] Czajka, S. ; Mohr, S.: Einkäufe privater Haushalte über das Internet. In: *Wirtschaft und Statistik* (2002), März

[Dre08] Drews, Markus: Interaction Patterns für Produktkonfiguratoren. In: Herczeg, Michael (Hrsg.) ; Kindsmüller, Martin C. (Hrsg.): *Mensch und Computer*, Oldenbourg Verlag, 2008. – ISBN 978–3–486–58900–9, 367-376

[GT02] Grenci, Richard T. ; Todd, Peter A.: Solutions-driven marketing. In: *Commun. ACM* 45 (2002), März, Nr. 3, 64–71. http://dx.doi.org/10.1145/504729.504730. – DOI 10.1145/504729.504730. – ISSN 0001–0782

[HBK08] Hassenzahl, M. ; Burmester, M. ; Koller, F.: Der User Experience (UX) auf der Spur: Zum Einsatz von www.attrakdiff.de. In: Brau, H. (Hrsg.) ; Diefenbach, S. (Hrsg.) ; Hassenzahl, M. (Hrsg.) ; Koller, F. (Hrsg.) ; Peissner, M. (Hrsg.) ; Röse, K. (Hrsg.): *Usability Professionals 2008*. Fraunhofer-Verlag, 2008, S. 78–82

[HG07] Hesseler, M. ; Görtz, M.: *Basiswissen ERP-Systeme : Auswahl, Einführung und Einsatz betriebswirtschaftlicher Standardsoftware*. Dortmund : w3l GmbH, 2007

[HHW11] Hippner, H. ; Hubrich, B. ; Wilde, K. D.: *Grundlagen des CRM : Strategie, Geschäftsprozesse und IT-Unterstützung*. Wiesbaden : Gabler, 2011

[HL06] Hansen, P. ; Loos, T.: Scenario-Driven Configuration Systems — Examining the Influence of Product Types and Mind States on Customer Satisfaction in Product Configuration Processes. In: Blecker, Thorsten (Hrsg.) ; Friedrich, Gerhard (Hrsg.) ; Hvam, Lars (Hrsg.) ; Edwards, Kasper (Hrsg.): *Customer Interaction and Customer Integration*. Gito-Verlag, 2006, S. 135–149

[MS10] Möhring, W. ; Schlütz, D.: *Die Befragung in der Medien- und Kommunikationswissenschaft: Eine praxisorientierte Einführung*. VS Verlag für Sozialwissenschaften, 2010 (Studienbücher zur Kommunikations- und Medienwissenschaft). http://books.google.de/books?id=PGjuPjsWv68C. – ISBN 9783531169941

[Pil06] Piller, F. T.: *Mass customization : ein wettbewerbsstrategisches Konzept im Informationszeitalter*. Wiesbaden : Dt. Univ.-Verl., 2006

[PR02] Piller, F. T. ; Rogoll, T. A.: *Konfigurationssysteme für Mass Customization und Variantenproduktion*. September 2002. – Marktstudie 2002/03

[Sch08] Schneider, W.: *Ergonomische Gestaltung von Benutzungsschnittstellen: Kommentar zur Grundsatznorm DIN EN ISO 9241-110*. Berlin : Beuth, 2008

[SSK00] Stanford-Smith, B. ; Kidd, P.T.: *E-Business: Key Issues, Aplications and Technologies*. Ios PressInc, 2000 http://books.google.de/books?id=6FVAfEFyMcwC. – ISBN 9781586030896

[Thi08] Thiemann, P.: *Benutzerfreundliche Online-Hilfen: Grundlagen und Umsetzung mit MadCap Flare*. Wiesbaden : Vieweg+Teubner Verlag, 2008

[Wä10] Wäger, M.: *Grafik und Gestaltung. Das umfassende Handbuch*. Bonn : Galileo Design, 2010

[Wie08] Wieken, J. H.: *SQL: Einstieg für Anspruchsvolle (Master Class)*. München : Addison-Wesley, Pearson Deutschland, 2008

Eidesstattliche Erklärung: Hiermit versichere ich an Eides statt, dass ich diese Bachelorarbeit selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und dass alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie dass ich die Bachelorarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Patrick Kuglmeier,
Passau, 23. März 2013