Institut für Neuroinformatik, Prof. Dr. Günther Palm

Automatisierte Verifikation des Designs von grafischen Benutzeroberflächen mit Bayes'schen Netzen

Konzeptualisierung und praktische Anwendung am Beispiel des Audi Infotainmentsystems MMI

Dissertation zur Erlangung des Doktorgrades Dr. rer. nat.der Fakultät für Ingenieurwissenschaften und Informatik der Universität Ulm

von

Miriam Sarah Katharina Heilemann

aus Berlin

2013

Amtierender Dekan: Prof. Dr. Klaus Dietmayer

Erster Gutachter: Prof. Dr. Günther Palm

Zweiter Gutachter: Prof. Dr. Michael Weber

Tag der Promotion: 25.03.2014

Danksagung

Die vorliegende Dissertation entstand während meiner Zeit als Doktorandin bei der AUDI AG in Ingolstadt. Aus diesem Umfeld danke ich insbesondere Herrn Dieter Niederkorn und Herrn Alexander Höfer von der Abteilung „Integrationserprobung Infotainment“, dass sie es mir ermöglicht haben in so einem interessanten Gebiet zu promovieren. Ebenfalls danke ich meinem damaligen Kollegen Herrn Marcus Beccu für die gute und nette Zusammenarbeit im Projekt „Bildverarbeitung“ und seine stetige Unterstützung meines Promotionsvorhabens. Ich danke auch den Studenten, die im Rahmen von Abschlussarbeiten die praktische Umsetzung der Arbeit gefördert haben. Ebenso bedanke ich mich für die interessanten Gespräche und das entgegengebrachte Interesse an meiner Promotion bei den Doktoranden der automotiveHMI-Arbeitsgruppe.

Mein ganz besonderer Dank gilt Herrn Prof. Dr. Günther Palm, Leiter der Abteilung Neuroinformatik an der Universität Ulm, der mir als Betreuer und Erstgutachter stets zur Seite stand und mir in zahlreichen aufschlussreichen Gesprächen wertvolle Anregungen zur Gestaltung meiner Dissertation gab. Ebenso danke ich herzlich meinem Zweitgutachter Herrn Prof. Michael Weber, Leiter der Abteilung Medieninformatik an der Universität Ulm, für die freundliche Unterstützung und interessanten Gespräche. Des Weiteren geht mein Dank an Herrn Prof. Dr. Uwe Schöning, Herrn Prof. Dr. Wolfgang Minker und Herrn Prof. Helmuth Partsch für das Interesse an meiner Arbeit und ihre Bereitschaft die Rolle der Prüfer zu übernehmen.

Zu guter letzt bedanke ich mich bei meinem Mann Robert Deininger für die stets liebe, geduldige und aufmunternde Unterstützung meiner Promotion sowie all meiner sonstigen Vorhaben.

Kurzfassung

Heutige Infotainmentsysteme wie das Audi Infotainmentsystem *MMI* bieten eine große Funktionsvielfalt und erfordern eine umfangreiche Testautomatisierung. Im Gegensatz zur Verhaltenslogik und den Textinhalten wurde das Design der grafischen Benutzeroberfläche (GUI) bislang allerdings nicht verifiziert. Das in dieser Arbeit vorgestellte Designverifikationssystem schließt diese Lücke.

Das Designverifikationssystem ist als wissensbasiertes System konzipiert und besteht aus voneinander unabhängig erweiterbaren Komponenten. Den Systemkern bildet ein Bayes'sches Netz, die Wissensbasis ist zweigeteilt in das Soll- und das Ist-Design. Das Soll-Design wird im Designentwurf spezifiziert und umfasst die Position, die optische Darstellung, die semantische Bedeutung und die globalen Beziehungen von Texten und Symbolen einer GUI. Im Gegensatz zu dem klar vorgegebenen Soll-Design, wird das Ist-Design mittels Bildverarbeitungsmethoden aus dem Displayinhalt erworben und ist naturgemäß mit Unsicherheiten behaftet. Der Vergleich von sicheren Soll- mit unsicheren Ist-Designinformationen stellt ein neues Einsatzgebiet für Bayes'sche Netze dar.

Am konkreten Beispiel der MMI-Testautomatisierung zeigt die Dissertation auch die Realisierbarkeit wissenschaftlicher Konzepte in der Praxis durch die geeignete Anpassung bewährter Prozesse. Die Automatisierung von Testanforderungen sowie der produktive Einsatz Bayes'scher Netze scheitern oft daran, dass das Wissen nicht formal vorliegt. Auch für das GUI-Design vom MMI existierte zu Beginn des Projekts keine formale Designspezifikation. Eine Evaluation existierender GUI-Spezifikationssprachen zeigte zudem deren Unfähigkeit das GUI-Design geeignet zu beschreiben. Aus diesen Beweggründen heraus entstand basierend auf dem Datenaustauschformat *JSON* ein neues Spezifikationsformat, das exakt auf die kompakte und effiziente Beschreibung von GUI-Design zugeschnitten ist. Ohne konkrete Unterstützung der Spezifikation im MMI-Entwicklungsprozess ist der produktive Einsatz jedoch nicht möglich. Als Alternative wurde deshalb zusätzlich diskutiert, inwieweit das maschinelle Lernen eines Bayes'schen Netzes für die Designverifikation möglich ist und eine formale Designspezifikation ersetzen kann.

Die gewonnen Erkenntnisse der Arbeit zeigen, dass die automatisierte Verifikation von GUI-Design realisierbar ist und ein Bayes'sches Netz hierfür ein geeignetes und mächtiges Werkzeug darstellt. Das entwickelte System zeichnet sich durch Eigenschaften wie automatisierte und effiziente Verarbeitung unsicherer Informationen,

Vollständigkeit in der Fehlerabdeckung, Korrektheit, Erweiterbarkeit und Universalität aus. Darüber hinaus beschränkt sich der Beitrag dieser Arbeit nicht auf die Designverifikation. U.a. dienen die entwickelte JSON-Spezifikation und die Bildverarbeitung der Realisierung weiterer automatisierter Tests für das MMI. Das Lernen Bayes'scher Netze zur Repräsentation von GUI-Design, der produktive Einsatz in der MMI-Testautomatisierung sowie die Unterstützung dynamischer Designinhalte liefern weitere interessante Erkenntnisse und Fragestellungen für zukünftige Arbeiten.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Anwendungsumfeld 1

1.2 Motivation und Zielsetzung 2

1.3 Bayes'sche Netze in der Praxis 5

1.4 Gliederung der Dissertation 6

2 Infotainmentsysteme und Testautomatisierung **9**

2.1 Infotainmentsysteme 9

2.1.1 Funktionen 10

2.1.2 HMI 10

2.2 Audi Infotainmentsystem 11

2.2.1 MMI GUI 12

2.2.2 GUI-Design 15

2.3 HMI-Testautomatisierung 21

2.3.1 Verifikation und Validierung 21

2.3.2 Fehlervermeidung 22

2.3.3 Modellbasiertes Testen 23

2.3.4 Ansätze aus der Künstlichen Intelligenz 23

2.3.5 Bildverarbeitung in der Testautomatisierung 24

2.4 MMI-Testautomatisierung 28

2.4.1 Testmethoden 28

2.4.2 Aktueller Umfang 29

3 Formale Spezifikationssprachen **31**

3.1 Formale Sprachen und Spezifikationen 31

3.1.1 Vorteile formaler Spezifikationen 32

3.1.2 Nachteile formaler Spezifikationen 32
3.2 Extensible Markup Language (XML) 33
3.2.1 XML-Syntax 34
3.2.2 XML-Schema 34
3.2.3 Einsatz und Nachteile 35
3.3 JavaScript Object Notation (JSON) 35
3.3.1 JSON-Syntax 36
3.3.2 JSON-Schema 36
3.3.3 Einsatz 38
3.3.4 JSON vs. XML 39
3.4 Formale Spezifikation von GUIs 40
3.4.1 GUI-Spezifikationssprachen 40
3.4.2 GUI-Spezifikationssprachen und GUI-Design 48
3.4.3 Spezifikation von GUI-Design 50
3.4.4 Integration in den GUI-Entwicklungsprozess 52

4 Wissensverarbeitung **53**
4.1 Wissensbasierte Systeme 53
4.1.1 Aufbau 53
4.1.2 Kategorisierung 55
4.2 Schlussfolgern mit unsicherem Wissen 56
4.2.1 Arten von Unsicherheit 56
4.2.2 Schlussfolgern unter Unsicherheit mit quantitativen Methoden 57
4.2.3 Gegenüberstellung und weiterführende Ansätze 59

5 Bayes'sche Netze **63**
5.1 Terminologie 64
5.1.1 Definition des Wahrscheinlichkeitsbegriffs 64
5.1.2 Bayestheorem 65
5.1.3 Eigenschaften Bayes'scher Netze 66
5.1.4 Bayes'sche Netze zur Klassifikation 70
5.2 Inferenzberechnungen 75
5.2.1 Exakte Inferenzverfahren 75
5.2.2 Approximative Inferenzverfahren 83
5.2.3 Inferenz mit unsicheren Evidenzen 84

5.3 Generierung von Bayes'schen Netzen . . . 88
5.3.1 Netzgenerierung aus formaler Wissensbeschreibung . . . 89
5.3.2 Netzgenerierung durch datenbasiertes Lernen . . . 90
5.3.3 Generierung von Bayes'schen Netzen in der Praxis . . . 101
5.4 Korrektheit . . . 102
5.4.1 Gütebestimmung für Generierungsverfahren . . . 104
5.4.2 Gütebestimmung für Bayes'sche Netze . . . 105
5.4.3 Feststellung und Vermeidung von Konflikten . . . 107
5.5 Effizienz . . . 110
5.6 Erklärbarkeit Bayes'scher Netze . . . 112
5.6.1 Arten von Erklärbarkeit und ihre Darstellung . . . 112
5.6.2 Vermittelnde Variablen . . . 114
5.6.3 Erklärbarkeit gelernter Netze . . . 115
5.7 Einordnung und Erweiterungen . . . 116
5.7.1 Ketten- und Faktorgraphen . . . 119
5.7.2 Dynamische Bayes'sche Netze . . . 121
5.7.3 Graphenmodelle in der Praxis . . . 123

6 Designverifikationssystem 125
6.1 Anforderungen . . . 125
6.2 Realisierbarkeit . . . 127
6.3 Systemarchitektur . . . 128
6.3.1 Inferenzkomponente . . . 128
6.3.2 Wissensbasis . . . 129
6.3.3 Wissenserwerbskomponente . . . 129
6.3.4 Erklärungskomponente . . . 131
6.4 Schlussfolgerung . . . 132

7 Bayes'sches Netz zur Designverifikation 133
7.1 Allgemeiner Aufbau . . . 133
7.1.1 Variablen . . . 134
7.1.2 Netzstruktur . . . 135
7.1.3 Parameter . . . 137
7.1.4 Verifikationsergebnisse . . . 138
7.2 Verifikationsprozess . . . 138

7.3 Verifikation eines Wizardmenüs . . . 139
7.3.1 Parameter . . . 141
7.3.2 Verifikation . . . 143
7.3.3 Erweiterungen . . . 152
7.4 Begründung und Einordnung . . . 154
7.4.1 Verwendung eines Bayes'schen Netzes . . . 154
7.4.2 Effizienz . . . 156
7.4.3 Behandlung von Unsicherheit . . . 158
7.4.4 Kombination der Wahrscheinlichkeiten . . . 160
7.4.5 Merkmalseingabe . . . 166
7.4.6 Verifikation globaler Merkmale . . . 166
7.4.7 Interpretation der Ergebnisse . . . 167
7.4.8 Korrektheit . . . 168
7.4.9 Erweiterbarkeit . . . 169
7.5 Schlussfolgerung . . . 170

8 Generierung des Bayes'schen Netzes aus einer Spezifikation 173
8.1 Grundlegende Anforderungen und Überlegungen . . . 173
8.1.1 Allgemeine Anforderungen . . . 174
8.1.2 Struktur der formalen Soll-Designspezifikation . . . 175
8.1.3 Netzgenerierung . . . 176
8.2 Soll-Designspezifikation in JSON . . . 177
8.2.1 Lokales Design . . . 177
8.2.2 Globales Design . . . 181
8.3 Generierung des Bayes'sche Netzes . . . 183
8.3.1 Bayes'sches Netz aus JSON-Spezifikation . . . 184
8.3.2 Vervollständigung mit Zusatzwissen . . . 184
8.3.3 Generierungsprozess . . . 187
8.4 Spezifikation eines Wizardmenüs . . . 187
8.4.1 Soll-Designspezifikation . . . 188
8.4.2 Zusatzwissen . . . 191
8.5 Begründung und Einordnung . . . 192
8.5.1 JSON als Spezifikationsformat . . . 192
8.5.2 Zusatzwissen . . . 200

8.5.3 Generierungsprozess . . . 201
8.5.4 Vergleich mit manuell erzeugtem Bayes'schen Netz . . . 202
8.6 Schlussfolgerung . . . 202

9 Generierung des Bayes'schen Netzes durch Lernen 205
9.1 Motivation und Ziel . . . 205
9.2 Voraussetzungen . . . 207
9.2.1 Lerndaten . . . 207
9.2.2 Lernverfahren . . . 209
9.3 Beispiel . . . 210
9.3.1 Beispiel A - Lokale Merkmale eines Icons . . . 210
9.3.2 Beispiel B - Globale Merkmale . . . 217
9.3.3 Weitere gelernte Bayes'sche Netze . . . 218
9.4 Begründung und Einordnung . . . 220
9.4.1 Erzeugung von Lerndaten . . . 220
9.4.2 Qualität und Performanz der Lernverfahren . . . 221
9.4.3 Erklärbarkeit . . . 226
9.4.4 Reduzierung des Lernaufwands . . . 226
9.4.5 Korrektheit . . . 229
9.4.6 Fehlerabdeckung . . . 230
9.5 Schlussfolgerung . . . 231

10 Integration in die MMI-Testautomatisierung 233
10.1 Aktueller Stand . . . 233
10.1.1 MMI-Designverifikationsprozess . . . 233
10.1.2 Ist-Design . . . 234
10.1.3 Soll-Design . . . 234
10.1.4 Offline- und Online-Tests . . . 235
10.2 Ergebnisse . . . 235
10.3 Bewertung und Ausblick . . . 238
10.3.1 Vollständigkeit . . . 238
10.3.2 Korrektheit . . . 239
10.3.3 Behandlung von Unsicherheit . . . 239
10.3.4 Anzeige der Verifikationsergebnisse . . . 240
10.3.5 Online-Tests . . . 241

10.3.6 Datenverwaltung 241
10.4 Schlussfolgerung 241

11 Schlussbewertung **243**
11.1 Zusammenfassung und Ergebnisse 243
11.2 Erreichte Ziele 246
11.2.1 Vollständigkeit 246
11.2.2 Unsichere Information 247
11.2.3 Automatisierung 248
11.2.4 Maschinenlesbarkeit 248
11.2.5 Effizienz 248
11.2.6 Erweiterbarkeit 249
11.2.7 Universalität 249
11.2.8 Korrektheit 250
11.3 Ausblick 250
11.3.1 Aktualisierung bestehender Bayes'scher Netze und Wissensbasen 251
11.3.2 Unterstützung zukünftiger MMI-GUIs 251
11.3.3 Lernen 252
11.3.4 Globale Designtests 253
11.3.5 Dynamische Designtests 253
11.3.6 Einsatzgebiet MMI-Testautomatisierung 253
11.3.7 Korrektheit 254
11.3.8 Optimierung 254
11.4 Fazit 255

A Symbol- und Abkürzungsverzeichnis **257**
A.1 Konventionen 257
A.2 Mathematische Symbole 257
A.3 Abkürzungen 258

B Beispiele für die Berechnung des Konfliktmaßes **261**
B.1 Naiver Bayes-Klassifikator mit spärlichen CPTs 261
B.1.1 Verbundwahrscheinlichkeitstabelle P(F,Z) 261
B.1.2 Wahrscheinlichkeit P(F=anders) 261

B.1.3 Verbundwahrscheinlichkeitstabelle P(H,Z) 262
B.1.4 Wahrscheinlichkeit P(H=h2) 262
B.1.5 Verbundwahrscheinlichkeit P(Z, F=anders, H=h2) 262
B.1.6 Konfliktmaß conf(F=anders, H=h2) 262
B.2 Naiver Bayes-Klassifikator mit dichten CPTs 263
B.2.1 Verbundwahrscheinlichkeitstabelle P(F,Z) 263
B.2.2 Wahrscheinlichkeit P(F=anders) 263
B.2.3 Verbundwahrscheinlichkeitstabelle P(H,Z) 263
B.2.4 WahrscheinlichkeitP (H=h2) 263
B.2.5 Verbundwahrscheinlichkeit P(Z, F=anders, H=h2) 263
B.2.6 Konfliktmaß conf(F=anders, H=h2) 264

C GUI-Design-Schemata **265**
C.1 JSON-Schema 265
C.2 XML-Schema 269

D Software für Bayes'sche Netze **275**

Abbildungsverzeichnis

2.1 Einheitliche Bedienlogik des Audi MMI am Beispiel der Version 2G von 2002. Oben: Screenshots der MMI-GUI; Unten: Bedienteil mit zum Screendesign passender Funktionstastenanordnung (Tièsch-ky u. a. 2002, S. 13) 11

2.2 GUI von vier Versionen des MMI. Oben: Telefonmenü beim 2G- und 3G-System. Unten: Zieleingabe im Navigationsmenü beim 3G+-System mit Touchpadeingabe und dem MIB-System. 12

2.3 Grundlegende Screenbereiche einer MMI-GUI. 13

2.4 . View *Setup-Wizardmenü* des MIB High. Die Screenshots zeigen dieselbe View mit einem jeweils anderen selektierten Icon und verschiedenen Cursorpositionen. Im oberen Screenshot sind die einzelnen Elemente des Wizardmenüs gekennzeichnet. 13

2.5 . Die acht Icons der View *Setup-Wizardmenü* des MIB High mit ihren zustandsabhängigen Ausprägungen von oben nach unten: *aktiv, selektiert, inaktiv*. 14

2.6 Designmerkmale eines MMI-Screenshots. 15

2.7 Eine PSD-Datei mit ihrer Layerstruktur. Die aktuelle Kombination sichtbarer Layer stellt ein Wizardmenü des Medien-Kontextes dar. . 18

2.8 Ausschnitt aus dem Spezifikationsdokument [Betz 2010] zur Definition des Aufbaus und der Funktionsweise eines Wizardmenüs im 3G+. 19

2.9 Definition von Positionierungen und Iconzuständen für das 3G+ im Spezifikationsdokument [Promoli 2011]. 20

2.10 Definition der Fonts für das 3G+ im Spezifikationsdokument [Promoli 2011]. 20

2.11 Auswirkung der Kantenglättung auf GUI-Elemente am Beispiel eines Ausschnitts eines MMI-Screenshots. 26

3.1 Mit XAML spezifizierte GUI. 43

5.1 Darstellung der d-Separierungseigenschaften für eine serielle, eine divergierende und eine konvergierende Verbindung (vgl. Jensen und Nielsen 2007, Kap. 2.2). Für die Variable *E* ist jeweils Evidenz gesetzt. Die Variablen *X* und *Y* werden durch die gesetzte Evidenz entweder unabhängig voneinander (oben und mittig) oder abhängig (unten). 67

5.2 Naiver Bayes-Klassifikator zur Klassifikation von einfarbigen Regionen als Schriftzeichen basierend auf den Merkmalen *Farbe* und *Höhe*. (*Bayes-Beispiel 1*) 70

5.3 Bayes'sches Netz zur Klassifikation von einfarbigen Regionen als Schriftzeichen, basierend auf den Merkmalen *Farbe* und *Höhe* mit definierter Abhängigkeit zwischen den Merkmalsvariablen. (*Bayes-Beispiel 2*) 72

5.4 Klassifikationsergebnisse in Abhängigkeit der gegebenen Merkmale *Farbe* und *Hoehe* (*Bayes-Beispiel 2*). 73

5.5 Abhängigkeit der Variablen `Farbe` von der Variablen `Hoehe` bei gegebener Evidenz für die Variablen `Zeichen` und `Hoehe` (*Bayes-Beispiel 2*). 74

5.6 Zustandswahrscheinlichkeiten der Variablen `Farbe` und `Hoehe` bei gegebener Evidenz der Variablen `Zeichen` (*Bayes-Beispiel 2*). . . . 74

5.7 Erweitertes Bayes'sches Netz für die Schriftzeichenklassifikation mit den Merkmalen *Position*, *Farbe* und *Hoehe*. (*Bayes-Beispiel 3*) . 79

5.8 *Bayes-Beispiel 4* mit Polytree-Struktur, das durch Zusammenfassen von Variablen aus *Bayes-Beispiel 3* entstanden ist. 80

5.9 Umwandlung von einem Bayes'schen Netz zu einem Junction-Tree. Oben: Ausgangs-Netz; Mitte: moraler, triangulierter Graph; Unten: Junction-Tree mit Separatoren. 81

5.10 Oben Bayes'sches Netz $\mathcal{B}1$ mit $X = \{A, B, C\}$: Durch Eingabe einer weichen Evidenz SE nach der Jeffrey-Regel wird die Verbindung zwischen A und dem Elternknoten C und damit CPT $P(A|C)$ aufgelöst. Unten Bayes'sches Netz $\mathcal{B}2$ mit $X = \{D, E, F\}$: Durch Eingabe der weichen Evidenz über die virtuelle Variable $V \notin X$ bleibt die Verbindung zwischen D und Elternknoten F unverändert. 86

5.11 *Bayes-Beispiel 2* mit der konfliktauslösenden Evidenzmenge{`Zeichen`=*wahr*, `Hoehe`=*anders*} 108

5.12 Übersicht einiger graphischer Modelle. Als Rechtecke dargestellt sind die allgemeinen Graphengrundmodelle, Elternknoten besitzen jeweils eine höhere Universalität in der Abbildung von Informationen. Die ovalen Elemente repräsentieren Unterklassen des jeweiligen graphischen Grundmodells. Linien mit einem Rauten-Ende weisen auf Erweiterungen einer Unterklasse. Fein gestrichelte Linien zwischen einem ovalen Element X und einem gestrichelt ovalen Element Y bezeichnen eine *Y kann mit X implementiert werden*-Beziehung. 116

5.13 Beziehung zwischen der Menge $\mathcal{Q}$ aller Wahrscheinlichkeitsverteilungen und ihren Untermengen $\mathcal{U}$ und $\mathcal{D}$, deren Wahrscheinlichkeitsverteilungen in Form eines UG und/oder DG als Perfect Map beschrieben werden können. 117

5.14 Oben ein Bayes'sches Netz bestehend aus den drei Variablen A, B und C. Unten zwei dazu passende Faktorgraphen mit den Faktoren $F1$ sowie $F2$, $F3$ und $F4$. 121

6.1 Architektur des Designverifikationssystems. 128

7.1 Exemplarische Struktur eines Bayes'schen Netzes für die Verifikation von lokalen Designmerkmalen eines Elements *Element1* mit den drei Schichten Eingabe (gelb), Vergleich (blau) und Ausgabe (grün). 135

7.2 Exemplarische Bayes'sche Netze für die Verifikation von globalen Merkmalen. Die Vergleichsschicht ist hier zugleich die Ausgabeschicht. Die gelb-grünen MVs sind zugleich VVs lokaler Verifikationen. 136

7.3 Ein Wizardmenü im Media-Kontext vom MMI 3G+. 140

7.4 Die fünf Zustände des Icons *IconPlay* und die Koexistenz mit dem Tooltip-Widget. 140

7.5 Bayes'sches Netz für die Verifikation eines Teils des Wizardmenüs aus Abb. 7.3. Das orangefarben umrahmte Teilnetz dient der Verifikation lokaler Merkmale des Icons *IconPlay*, das blau umrahmte Teilnetz der Verifikation lokaler Merkmale des dazugehörigen Textelements *TooltipTextPlay* und das grün umrahmte Teilnetz der Verifikation des globalen Merkmals *Koexistenz*. Die gelb markierten Variablen sind MVs und die grün markierten VVs. Die gezeigten CPTs sind in ihrem Initialzustand, da keine Merkmale bzw. Evidenzen gesetzt sind. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010)) 141

7.6 Teil des Soll-Designs eines MMI-Wizardmenüs. 143

7.7 Bayes'sches Netz für die Verifikation des Wizardmenü-Ausschnitts in Abb. 7.6. Wahrscheinlichkeiten der einzelnen Variablen bei Propagation der Soll-Merkmale und sicheren Erkennersicherheiten. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010)) 144

7.8 Bayes'sches Netz für die Verifikation des Wizardmenü-Ausschnitts in Abb. 7.6. Wahrscheinlichkeiten der einzelnen Variablen bei Propagation der Soll-Merkmale und unsicherer Erkennersicherheiten. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010)) 145

7.9 Bayes'sches Netz für die Verifikation des Wizardmenü-Ausschnitts in Abb. 7.6. Wahrscheinlichkeiten der einzelnen Variablen bei Propagation von Merkmalen, die schwach von den Sollwerten abweichen. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010) . . 147

7.10 Bayes'sches Netz für die Verifikation des Wizardmenü-Ausschnitts in Abb. 7.6. Wahrscheinlichkeiten der einzelnen Variablen bei Propagation von Merkmalen, die teilweise sehr stark von den Sollwerten abweichen. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010)) 148

7.11 Teil des Ist-Designs eines MMI-Wizardmenüs. 149

7.12 Bayes'sches Netz für die Verifikation des Wizardmenü-Ausschnitts in Abb. 7.6. Wahrscheinlichkeiten der einzelnen Variablen bei Propagation der Soll-Merkmale. Die Kombination des erkannten Iconzustands und der Existenz des Textelements führt zu einer falschen Koexistenz. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010)) 150

7.13 Bayes'sches Netz für die Verifikation des Wizardmenü-Ausschnitts in Abb. 7.6. Wahrscheinlichkeiten der einzelnen Variablen bei Propagation der Soll-Merkmale für das Symbolelement. Im Ist-Design existiert kein Textelement, somit ist die Koexistenz bei Iconzustand *disabled* korrekt. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010)) 151

7.14 Bayes'sches Netz für die Verifikation des Merkmals *Font* des Elements *TooltipTextPlay*. Oben: der Initialzustand; Unten links: die Wahrscheinlichkeitsverteilungen bei Eingabe der Sollwerte; Unten rechts: fehlerhafte Eingaben für die Merkmalskomponenten *Serifen* und *Proportional*. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010)) 152

7.15 Bayes'sches Netz zur Verifikation der lokalen und globalen Merkmale zweier Icons und ihrer dazugehörigen zwei Tooltips eines MMI-Wizardmenüs. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001)) 157

7.16 Dasselbe Bayes'sche Netz aus Abbildung 7.15 ohne Beschriftung der Variablen. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001)) 157

7.17 Dasselbe Bayes'sche Netz aus Abbildung 7.15 in dem nur die Variablen der Ausgabeschicht und deren direkten Elternknoten eingeblendet sind. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001)) 158

7.18 Bayes'sches Netz *A* zur Verdeutlichung der Kombination von Wahrscheinlichkeiten. Die Wahrscheinlichkeitsverteilung der Ausgabevariablen `VV` stimmt hier exakt mit den Wahrscheinlichkeitsverteilungen der Eingabevariablen `MV1` bis `MV11` überein. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010)) 162

7.19 Bayes'sches Netz *B* zur Verdeutlichung der Kombination von Wahrscheinlichkeiten. Bei dieser Art der Kombination weist die Wahrscheinlichkeit des Zustands *wahr* der Ausgabevariablen VV eine Abschwächung gegenüber der Wahrscheinlichkeiten der Eingabevariablen MV1 bis MV10 auf. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010)) 162

7.20 Bayes'sches Netz *A*: Mit Evidenz für Zustand *falsch* bei Variable MV1 verringert sich die Wahrscheinlichkeit des Zustands *wahr* bei Variable VV nur minimal. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010)) 163

7.21 Bayes'sches Netz *B*: Mit Evidenz für Zustand *falsch* bei Variable MV5 verringert sich die Wahrscheinlichkeit des Zustands *wahr* bei Variable VV auf 90%, was der gewünschten Gewichtung der Merkmale entspricht. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010)) 163

7.22 Bayes'sches Netz *B*: Mit Evidenz für Zustand *falsch* bei Variable MV9 verringert sich die Wahrscheinlichkeit des Zustands *wahr* bei Variable VV auf 82%. Das ist eine stärkere Verringerung, als der gewünschten Gewichtung der Merkmale entspricht. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010)) 164

7.23 Bayes'sches Netz *A*: Mit Evidenz für Zustand *falsch* bei Variable MV10 verringert sich die Wahrscheinlichkeit des Zustands *wahr* bei Variable VV auf 67%, was der gewünschten Gewichtung der Merkmale entspricht. (Netz erstellt mit [*BayesBuilder* 2007-2010]) 164

7.24 Bayes'sches Netz *B*: Mit Evidenz für Zustand *falsch* bei den Variable MV1 bis MV8 verringert sich die Wahrscheinlichkeit des Zustands *wahr* bei Variable VV auf 34%. Das widerspricht der gewünschten Gewichtung der Merkmale, die eine Verringerung auf 20% definiert. (Netz erstellt mit [*BayesBuilder* 2007-2010]) 165

7.25 Bayes'sches Netz *B*: Mit Evidenz für Zustand *falsch* bei den Variable MV1 bis MV8 verringert sich die Wahrscheinlichkeit des Zustands *wahr* bei Variable VV auf 18%. Das widerspricht der gewünschten Gewichtung der Merkmale, die eine Verringerung auf 20% definiert. (Netz erstellt mit [*BayesBuilder* 2007-2010]) 165

8.1 Reihenfolge der Parameter in den Arrays `CombinationSingle` (CPT oben links), `CombinationDuad` (CPT unten links) und `CombinationTriple` (CPT rechts) des Zusatzwissens in JSON. 187

8.2 Generierung des Bayes'schen Netzes aus einer formalen Spezifikation, die aus drei Wissensquellen besteht. 188

9.1 Mit den Metriken AIC und MDL gelerntes Bayes'sches Netz zum Beispiel A. Es wurde die erste Variablenreihenfolge verwendet, sodass die Variable `Korrekt` der Wurzelknoten ist. Oben: Netz mit initialen Parametern. Unten: Parameter bei gesetzten Evidenzen (blaue Balken). (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010)) 211

9.2 Mit den Metriken AIC und Log-K2 gelerntes Bayes'sches Netz und CPT der Variablen `Korrekt` zum Beispiel A. Es wurde die zweite Variablenreihenfolge verwendet, sodass die Variable `Korrekt` ein Kindknoten ist. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001)) 212

9.3 Mit der MDL-Metrik gelerntes Bayes'sches Netz und CPT der Variablen `Korrekt` zum Beispiel A. Es wurde die zweite Variablenreihenfolge verwendet, sodass die Variable `Korrekt` ein Kindknoten ist. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001)) 213

9.4 Mit den Metriken AIC und Log-K2 aus den erweiterten Lerndaten A gelerntes Bayes'sches Netz und CPTs. Es wurde die erste Variablenreihenfolge verwendet, sodass die Variable `Korrekt` der Wurzelknoten ist. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001)) 214

9.5 Inferenzberechnungen im Bayes'schen Netz, das mit den Metriken AIC und Log-K2 aus den erweiterten Lerndaten A gelernt wurde. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010)) 214

9.6 Mit der AIC-Metrik aus den erweiterten Lerndaten A gelerntes Bayes'sches Netz und CPTs. Es wurde die zweite Variablenreihenfolge verwendet, sodass die Variable `Korrekt` ein Kindknoten ist. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001)) 215

9.7 Mit den Metriken Log-K2 und MDL aus den erweiterten Lerndaten A gelerntes Bayes'sches Netz und CPTs. Es wurde die zweite Variablenreihenfolge verwendet, sodass die Variable `Korrekt` ein Kindknoten ist. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001)) . 216
9.8 Mit der (R)HC-Suche und der MDL-Metrik aus den erweiterten Lerndaten A gelerntes Bayes'sches Netz. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001)) 216
9.9 Mit den Metriken AIC und Log-K2 gelerntes Bayes'sches Netz zum Beispiel B. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001)) . 217
9.10 Mit der MDL-Metrik gelerntes Bayes'sches Netz zum Beispiel B. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001)) 217
9.11 Mit der (R)HC-Suche und der MDL-Metrik gelerntes Bayes'sches Netz zum Beispiel B. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001)) 218
9.12 Mit der MDL-Metrik gelerntes Bayes'sches Netz zur Verifikation der Koexistenz der Zustände von sieben Wizardicons. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001)) 218
9.13 Mit der AIC-Metrik gelerntes Bayes'sches Netz zur Verifikation der Koexistenz der Zustände von sieben Wizardicons. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001)) 219
9.14 Mit der MDL-Metrik gelerntes Bayes'sches Netz zur Repräsentation der lokalen Merkmale von sieben Icons und zwei Textelementen sowie der Koexistenz zwischen jeweils einem Icon und einem Textelement. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001)) . . 220
9.15 Mit WEKA basierend auf den einfachen Lerndaten A gelerntes Bayes'sches Netz. Durch Verwendung der Bayes'schen Schätzung beinhaltet das Netz trotz spärlichen Lerndaten keine spärlichen CPTs. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001)) 223
9.16 Auswahl an Verfahren für das Parameter- und das Strukturlernen Bayes'scher Netze. Die gelbmarkierten Verfahren wurden in dieser Arbeit angewendet. 225
9.17 Kombiniertes Bayes'sches Netz aus der Spezifikation generierten (gelb und orangefarben) und gelernten (rot) Teilnetzen. 228

10.1 Implementierung des Designverifikationssystems für das MMI. . . . 234

10.2 Links: Verifizierter Screenshot-Ausschnitt eines Wizardmenüs ohne Designfehler. Rechts: Sicherheiten der Bildverarbeitung und der Verifikation der einzelnen Elemente. 236
10.3 Oben: Das Icon *skipforward* weicht mit seiner x-Position etwas stärker vom Soll ab. Unten: Das obligatorische Icon *skipbackward* wurde von der Bildverarbeitung nicht gefunden und fehlt daher. 237
10.4 Oben: Verifizierter Screenshot-Ausschnitt eines Wizardmenüs, in dem nur die korrekten Elemente markiert sind. Mitte: Derselbe Screenshot-Ausschnitt mit Markierung der Position des fehlenden Icons. Unten: Derselbe Screenshot-Ausschnitt mit Markierung des von der Bildverarbeitung nicht klassifizierbaren Elements. 238
10.5 Ein mögliches GUI-Tool zur Verwaltung und Visualisierung von MMI-Designfehlern. (Die GUI-Skizze wurde mit der Software *Evolus Pencil* (EVOLUS 2010) erstellt.) 240

Tabellenverzeichnis

2.1 Lokale Merkmale von Designelementen. 16
2.2 Globale Merkmale von Designelementen. 17

5.1 CPT für das *Bayes-Beispiel 1* zur Definition der Abhängigkeit zwischen den Variablen `Farbe` und `Zeichen`. 71
5.2 CPT für das *Bayes-Beispiel 1* zur Definition der Abhängigkeit zwischen den Variablen `Hoehe` und `Zeichen`. 71
5.3 CPT für das *Bayes-Beispiel 2* zur Definition der Abhängigkeitsbeziehungen zwischen den Merkmalsvariablen. 72
5.4 CPT ohne Nulleinträge für den naiven Bayes-Klassifikator zur Definition der Abhängigkeit zwischen den Variablen `Farbe` und `Zeichen`. (*Bayes-Beispiel 1b*) 77
5.5 CPT ohne Nulleinträge für den naiven Bayes-Klassifikator zur Definition der Abhängigkeit zwischen den Variablen `Hoehe` und `Zeichen`. (*Bayes-Beispiel 1b*) 77

7.1 CPT, der die obligatorische Existenz des Icons *IconPlay* definiert. . 142
7.2 CPT, der den Sollwert der oberen linken x-Position des Icons *IconPlay* definiert und Abweichungen entsprechend gewichtet. 142
7.3 CPT, der die Gesamtkorrektheit des Merkmals *Farbe* definiert bestehend aus den drei Korrektheitsaussagen der einzelnen Farbkanäle *Rot*, *Grün* und *Blau* mit `CC=Korrekt_TooltipTextPlay_Farbe` und `CR/CG/CB=Korrekt_TooltipTextPlay_FarbkanalRot/_FarbkanalGruen/_FarbkanalBlau`. 143
7.4 CPT, der die Kombination dreier binären Variablen definiert. 161
7.5 CPT, der die Kombination zweier binären Variablen definiert. 161

9.1 Lokale Merkmale des Wizard-Icons *IconPlay*. 207

9.2 Lerndaten zur Verifikation des Wizard-Icons *IconPlay* bestehend aus korrektem und fehlerhaften Design. 208
9.3 CPT der Variablen `IconPlay_PositionY`, der für das Beispiel A mit der Log-K2-Metrik gelernt wurde. 211
9.4 CPT der Variablen `Korrekt`, der für das Beispiel A und den erweiterten Lerndaten mit den Metriken AIC, Log-K2 und MDL gelernt wurde. 216

B.1 Verbundwahrscheinlichkeitstabelle $P(F, Z) = P(F|Z) \cdot P(Z)$. 261
B.2 Verbundwahrscheinlichkeitstabelle $P(H, Z) = P(H|Z) \cdot P(Z)$. 262
B.3 Verbundwahrscheinlichkeitstabelle $P(F, Z) = P(F|Z) \cdot P(Z)$. 263
B.4 Verbundwahrscheinlichkeitstabelle $P(H, Z) = P(H|Z) \cdot P(Z)$. 263

Kapitel 1

Einleitung

Dieses Kapitel stellt einleitend das Thema der Dissertation vor. Abschnitt 1.1 liefert einen kurzen Einblick in die Testautomatisierung von Audi Infotainmentsystemen als Anwendungsumfeld. In Abschnitt 1.2 wird die Entwicklung eines Systems zur Verifikation des Designs von grafischen Benutzeroberflächen motiviert. Dabei werden die zentralen Fragestellungen und die zu erreichenden Ziele formuliert. Ebenfalls erfolgt eine Abgrenzung des Themas. Da ein Bayes'sches Netz ein Hauptbestandteil des Verifikationssystems ist, werden in Abschnitt 1.3 aktuelle praktische Anwendungen Bayes'scher Netze vorgestellt. Abschließend erläutert Abschnitt 1.4 die Gliederung der Dissertation.

1.1 Anwendungsumfeld

Heutzutage verfügen immer mehr Automobile über ein Infotainmentsystem, das dem Anwender mittels einer grafischen Benutzeroberfläche (GUI) Funktionen aus den Bereichen Information, Sicherheit und Unterhaltung bietet. Der Funktionsumfang und die grafische Darstellung variieren je nach Automarke und Systemversion. Da sich Infotainmentsysteme zu einem wichtigen, wettberwebsrelevanten Faktor in der Automobilbranche entwickelt haben, ist es erstrebenswert, darin die neusten technischen Innovationen zu integrieren. Auch das Infotainmentsystem *MMI* von Audi ist seit der Erstveröffentlichung im Jahr 2001 stetig komplexer geworden. Bspw. bietet es neben dem Display und dem Bedienteil mit einem Touchpad und einem Sprachdialogsystem mehrere Schnittstellen für die Benutzerinteraktion an. Daneben nimmt auch das Angebot an Fahrerassistenzsystemen stetig zu. Der

Funktionsumfang spiegelt sich direkt in der GUI wider, die aktuell rund 3000 verschiedene Ansichten (engl. Views) umfasst.
Wie jedes technische System muss auch ein Infotainmentsystem auf Fehler getestet werden, bevor es zum Verkauf angeboten wird. Zumal ein Infotainmentsystem in der Regel während des Fahrens vom Fahrer bedient wird, ist eine korrekte Funktionsweise umso wichtiger. Da das manuelle Testen des komplexen Funktionsumfangs zu aufwendig ist, hat sich in diesem Bereich sehr stark die Testautomatisierung etabliert. Die Testautomatisierung für das MMI ist modellbasiert. Ein Referenzmodell bildet die Verhaltenslogik des MMI ab, um automatisiert durch die Menüstruktur zu navigieren. Die zu testende GUI liegt in Form von Screenshots als Bilddateien vor, die jeweils eine Momentaufnahme des Displayinhalts repräsentieren. Mittels Bildverarbeitung wird der Displayinhalt erkannt. Auf diese Weise können bspw. Buttons gefunden und Views identifiziert werden, was für die Navigation und den Abgleich mit dem Referenzmodell erforderlich ist. Neben der Verhaltenslogik wird auch der Inhalt von Texten basierend auf Sollwerten verifiziert.

1.2 Motivation und Zielsetzung

Im Gegensatz zur Verhaltenslogik und den Textinhalten wird das GUI-Design von der aktuellen MMI-Testautomatisierung nicht getestet. Allerdings basiert auch das Design der MMI-GUI auf zahlreichen Vorgaben, deren Einhaltung gewährleistet werden sollte. U.a. wird im Designentwurf die Position und Farbe von Texten sowie die semantische Bedeutung von Elementzuständen festgelegt. Das Design der MMI-GUI entsteht aus psychologischen Überlegungen heraus und wird so konzipiert, dass eine einfache, intuitive und ablenkungsarme Bedienung der komplexen Funktionalität ermöglicht wird. Ein fehlerhaftes Design kann zu einer erschwerten oder sogar falschen Bedienung des Infotainmentsystems führen. Neben Fehlern in der Funktionalität führen auch gerade Fehler in der Darstellung zu Ablenkung und können zur Gefahr im Straßenverkehr werden. Da die Komplexität des MMI und damit das der GUI stetig steigt, wird auch das Design zukünftig an Bedeutung gewinnen. In der Vergangenheit konnten immer wieder designspezifische Fehler beobachtet werden. Die Fehlerquote für Designfehler in einer frühen Testphase einer MMI-GUI belief sich dabei auf etwa 15%[1]. Designfehler können zudem die Ur-

[1] interne Quelle der AUDI AG

sache für das Fehlschlagen bestehender MMI-Tests sein. Wenn bspw. die Position eines Textes nicht mit dem Soll übereinstimmt, kann die Texterkennung an der erwarteten Position keinen Text finden. Nicht zuletzt besteht auch für das MMI der Anspruch auf eine 0-Fehler-Software, die Qualität und Stabilität gewährleistet. Aus all diesen Gründen ist die Erweiterung der bisherigen MMI-Testautomatisierung um eine Designverifikation erstrebenswert.
Aktuell ist kein Verfahren bekannt, um Designfehler in GUIs automatisiert zu identifizieren. Unter der Annahme, dass für eine GUI alle möglichen optischen Darstellungen in Form von Screenshots zur Verfügung stehen, kann eine Designverifikation als einfacher Bildvergleich realisiert werden. Diese Annahme gilt normalerweise jedoch nicht, da GUIs meist sehr dynamische Inhalte mit einer Vielzahl an Kombinationsmöglichkeiten aufweisen. Um für jeden möglichen Inhalt einen Screenshots zu erstellen, wäre ein sehr großer Aufwand nötig. Auch würde dies schnell zu einer enorm großen Menge an Screenshots führen. Eine Designverifikation kann deshalb nicht als Bildvergleich implementiert werden, sondern soll direkt das gegebene Design mit den Sollvorgaben des Designentwurfs vergleichen. Im Gegensatz zum modellbasierten Testen, das bislang in der MMI-Testautomatisierung vorherrscht, wird das Designverifikationssystem keinen Modellvergleich durchführen.
Diese Arbeit liefert ein grundlegendes Konzept eines Verifikationssystems für GUI-Design. Das System wird am Beispiel der MMI-GUI evaluiert und soll die Lücke im aktuellen MMI-Testautomatisierungsprozess schließen. Für die Produktivität sind Aspekte wie Erweiterbarkeit, Effizienz und die Vollständigkeit der Verifikation entscheidend:

- **Erweiterbarkeit**: Das Designverifikationssystem wird als wissensbasiertes System konzipiert und gliedert sich auf diese Weise in einzelne, voneinander unabhängige Komponenten, die erweiterbar sind.

- **Effizienz**: Die Verifikation von GUI-Design stellt keine Echtzeitanforderungen. In Anbetracht der stetig wachsenden Komplexität der GUI von Audi Infotainmentsystemen ist die Effizienz des Verfahrens dennoch vorteilhaft und entscheidend für die Benutzerakzeptanz. Jede Systemkomponente soll daher möglichst effizient ihre Aufgaben bewerkstelligen.

- **Vollständigkeit**: Die Designverifikation soll möglichst alle Designfehler in einer GUI finden und damit eine hohe Fehlerabdeckung erreichen. Die Voll-

ständigkeit der Verifikation kann nur gewährleistet werden, wenn die Wissensbasis das vollständige Ist- und Soll-Design bereitstellt und der Systemkern alle Informationen verarbeiten kann.

Das Soll-Design wird im Designentwurf von den Designern vorgegeben. Das Ist-Design liegt in Form von Screenhots vor und muss mittels Bildverarbeitungsmethoden ausgewertet werden. Da Bildinformationen naturgemäß mit Unsicherheit behaftet sind, handelt es sich bei der Designverifikation um einen Vergleich zwischen sicherem Soll- und unsicherem Ist-Design. Ein harter Vergleich würde zu vielen falsch-positiven Ausgaben führen, da eine Abweichung zum Soll kein echter Designfehler sein muss, sondern sich auch in der unsicheren Bildinformation begründen kann. Aus diesem Grund wird als Systemkern ein Bayes'sches Netz verwendet, mit dem die Unsicherheiten als Wahrscheinlichkeiten interpretiert und geeignet verarbeitet werden. Für eine automatische Verarbeitung der Informationen im Sinne der Testautomatisierung muss das Soll-Design maschinenlesbar vorliegen.
Alle oben genannten Punkte wurden in dieser Dissertation berücksichtigt und untersucht. Die Anwendung von Bayes'schen Netzen ist bereits sehr vielfältig, wie die Literatur zeigt (s. Abschnitt 1.3). Das Anwendungsumfeld Testautomatisierung ist allerdings neu. Die Dissertation stellt damit aus Sicht der Testautomatisierung einen neuen Ansatz vor, der etablierte Konzepte wie das modellbasierte Testen verlässt. Für den Formalismus Bayes'sches Netz ist es eine weitere Prüfung, ob er mächtig genug ist sehr umfangreiche Informationen, in diesem Fall GUI-Design, zu beschreiben und zu verarbeiten. Da das Soll-Design aktuell nicht in einem maschinenlesbaren Format vorliegt, wurde außerdem eine formale Spezifikation des Soll-Designs entworfen, die auch für weitere Aufgaben in der MMI-Testautomatisierung eine wertvolle Wissensbasis darstellt. Daneben wurden auch für die Benutzerakzeptanz wichtige Aspekte wie die Erklärbarkeit der Ergebnisse und die konkrete Integration im MMI-Testautomatisierungsprozess thematisiert. Die Entwicklung eines Verifikationssystems für GUI-Design hat mit dem MMI einen direkten Praxisbezug. Neben der Konzeptualisierung des Designverifikationssystems als neuer Anwendungsbereich Bayes'scher Netze, werden auch die Möglichkeiten und Grenzen in der praktischen Umsetzung aufgezeigt. Zu guter letzt leistet die Entwicklung eines Verfahrens zur automatisierten GUI-Designverifikation auch einen Beitrag für die Softwareentwicklung bzw. Softwarequalitätssicherung. Für einen umfangreichen produktiven Nutzen soll das Verfahren darüber hinaus universell eingesetzt

werden können und sich nicht auf das Audi Infotainmentsystem einschränken.

1.3 Bayes'sche Netze in der Praxis

Der Einsatz eines Bayes'schen Netzes für die Verifikation von GUI-Design ist neu. Im Gegensatz dazu ist die Anwendung Bayes'scher Netze im Bereich der medizinischen Diagnose und für unterschiedliche Klassifikationsaufgaben weitverbreitet. Auch für Spam-Filter, Unwettervorhersagen, die Erstellung individueller Kundenprofile und sogar für die Interpretation von literarischer Prosa wurden bereits Bayes'sche Netze entwickelt (s. Beierle und Kern-Isberner 2006, Kap. 12.4). Bei der Entwicklung lernfähiger oder dynamischer Systeme erweisen sich Bayes'sche Netze als Inferenzkomponente ebenfalls als geeignet (Korb und Nicholson 2004, Teil II und Kap. 4.5).
Die in der Literatur beschriebenen Bayes'schen Netze sind aus Gründen der Verständlichkeit meist recht klein, bestehend aus maximal ein paar Duzend Variablen mit einer geringen Anzahl an Zuständen und einer spärlichen Vernetzung. Nur in wenigen Arbeiten werden komplexe reale Systeme umfangreich diskutiert und die Mächtigkeit Bayes'scher Netze verdeutlicht. Z.B. stellt [Wiegerinck, Kappen und Burgers 2010] das medizinische Diagnosesystem *Promedas* vor, das auf einem Bayes'schen Netz basiert. Es wird seit 1996 stetig weiterentwickelt, umfasst aktuell mehrere tausend Variablen und bildet über 50000 Relationen ab. Damit ist es in der Lage über 3500 Diagnosen zu stellen. Die zwei weiteren in Wiegerinck, Kappen und Burgers 2010 beschriebenen Netze, zum einen zur Klassifikation von Gesteinsproben für Ölbohrungen und zum anderen für die Personenidentifizierung anhand DNA, sind nicht sehr komplex. Sie weisen jedoch weitere Schwierigkeiten wie kontinuierliche Variablenzustände und eine dynamische, datenbasierte Generierung auf. Auch im Bereich der Bildverarbeitung gibt es Arbeiten über die Verwendung Bayes'scher Netze. In [Luo, Savakis und Singhal 2005] wird z.B. ein Bayes'sches Netz für die semantische Klassifikation von Fotos vorgestellt, dessen Eingabedaten Bildinformationen sind. Bayes'sche Netze können offensichtlich in einer Vielzahl von Anwendungsgebieten eingesetzt werden und sehr umfangreiche Informationen verarbeiten. Ein Flaschenhals bei sehr komplexen Bayes'schen Netzen ist die Rechenzeit, die sich jedoch mit den existierenden effizienten Inferenzalgorithmen meist in den Griff bekommen lässt.

Im Bereich des modellbasierten Testens und der Testautomatisierung ist der Einsatz von Bayes'schen Netzen jedoch weitestgehend unerforscht. Lediglich in [Borth und Hasseln 2002] wird ein Bayes'sches Netz für die automatische Diagnose eines technischen Systems im Automobilkontext beschrieben. Als Hinderungsgrund für einen derartigen Einsatz wird dort die Schwierigkeit angesehen, die Netze automatisch auf Basis von formal beschriebenem Wissen zu generieren. Gerade in der Industrie ist das Vorhandensein einer vollständigen formalen Spezifikation oftmals nicht gegeben und der produktive Einsatz scheitert aufgrund der fehlenden automatischen Erstellung der Netze.
In dieser Arbeit werden neben dem Verifikationsverfahren auch der problematische Prozess der Netzakquise genauer beleuchtet und Wege für den produktiven Einsatz aufgezeigt, wenn keine (vollständige) formale Spezifikation vorhanden ist.

1.4 Gliederung der Dissertation

In den Kapiteln 2 bis 4 werden die Grundlagen des Forschungsbereichs vermittelt und auf relevante Literatur Bezug genommen. In Kap. 2 werden Infotainmentsysteme und ihre Testautomatisierung vorgestellt. Insbesondere wird auf den aktuellen Testautomatisierungsprozess vom MMI eingegangen und erläutert, in welcher Form dabei Bildverarbeitungsverfahren zum Einsatz kommen. In Kap. 3 wird die Notwendigkeit von formalen Spezifikationssprachen für automatisierte Prozesse diskutiert. Die Auszeichnungssprache XML und das Datenaustauschformat JSON werden vergleichend aufgeführt. Daneben werden sechs GUI-Spezifikationssprachen kurz vorgestellt und ihr Unvermögen GUI-Design zu beschreiben dargelegt. Kap. 4 liefert einen Überblick zu wissensbasierten Systemen und ihre Einsatzgebiete. Es werden Ansätze erläutert, in einem wissensbasierten System unsichere Informationen zu verarbeiten.
Das Kap. 5 gibt eine detaillierte Einführung in die Theorie Bayes'scher Netze. Neben allgemeinen Definitionen und Beispielen werden auch Erweiterungen wie approximative Inferenzverfahren, unsichere Evidenzen und dynamische Bayes'sche Netze erläutert. Es wird auf die Möglichkeiten der automatischen Generierung aus einer formalen Spezifikation und durch datenbasiertes Lernen eingegangen. Die Erklärbarkeit von Bayes'schen Netzen ist ebenfalls ein Thema. Abschließend wird eine Einordnung des Formalismus' Bayes'scher Netze in weitere probabilistische

Graphenmodelle gegeben.
Die Kap. 6 bis 9 stellen die zentralen Aspekte der Forschungsarbeit dar. In Kap. 6 wird die Architektur des Designverifikationssystems mit den einzelnen Komponenten dargelegt und der allgemeine Verifikationsprozess skizziert. In Kap. 7 wird ein Bayes'sches Netz für die Designverifikation konzipiert. Der abstrakte Entwurf wird am Beispiel eines GUI-Ausschnitts des MMI konkretisiert. Die einzelnen Entscheidungen werden ausführlich bewertet und der Einsatz eines Bayes'schen Netzes im Vergleich zu anderen Inferenzverfahren, insbesondere der Fuzzy-Logik, begründet. Daran anschließend wird in Kap. 8 ein formales Spezifikationsformat in JSON für die Beschreibung von GUI-Design vorgestellt und gezeigt, wie daraus das Bayes'sche Netz automatisch generiert werden kann. Hier wird auch begründet, weshalb kein existierendes Spezifikationsformat zum Einsatz kommt und die Vorteile der JSON-Spezifikation im direkten Vergleich mit einer XML-Spezifikation aufgezeigt. Kap. 9 zeigt mit dem Lernen von Bayes'schen Netzen eine Alternative für die automatische Netzgenerierung auf.
In Kap. 10 wird kurz gezeigt, wie das Designverifikationssystem in den aktuellen Prozess der MMI-Testautomatisierung integriert werden kann und welche zukünftige Arbeiten für den umfangreichen produktiven Einsatz notwendig sind.
Den Abschluss bildet Kap. 11 mit der Zusammenfassung der Arbeit. Die gewonnenen Erkenntnisse werden evaluiert und im Hinblick auf die Erfüllung der geforderten Ziele bewertet. Die Entwicklung des Designverifikationssystems hat zudem neue Fragestellungen hervorgebracht, die in einem kurzen Ausblick für weiterführende Arbeiten zusammengestellt sind.

Kapitel 2

Infotainmentsysteme und Testautomatisierung

In Abschnitt 2.1 werden Infotainmentsysteme im Allgemeinen und in Abschnitt 2.2 das Audi Infotainmentsystem MMI vorgestellt. Daran anschließend erläutert Abschnitt 2.3 den Prozess der Testautomatisierung von Infotainmentsystemen und Abschnitt 2.4 gibt einen Einblick in die aktuelle MMI-Testautomatisierung.

2.1 Infotainmentsysteme

Im 19. Jahrhundert wurde das erste Automobil entwickelt. Die angebotenen Funktionen dienten damals lediglich dem Zweck Informationen über den Zustand des Automobils zu liefern und wurden dem Fahrer dezentral mittels verschiedener analoger Anzeigen übermittelt. Erst in den 1920er Jahren kamen die ersten Autoradios auf den Markt und boten dem Fahrer neben Information auch Unterhaltung an (Stehle 2010). Ab diesem Zeitpunkt wurden neben technisch notwendigen Funktionen auch Komfort-Funktionen als weitere, separate Geräte in das Automobil integriert, wie bspw. 1940 die Autoklimaanlage und 1993 das Navigationssystem (Stehle 2010). Mit Erfindung der Kassette und der CD erhielten auch diese Medien Einzug in das Automobil (Stehle 2010). Aufgrund des stetig wachsenden Funktionsumfangs wurde schließlich die zentrale Steuerung aller Funktionen wünschenswert. 1993 veröffentlichte der Automobilhersteller Mercedes Benz das *Communications- und Navigationssystem*, das alle Funktionen in einem Gerät kombinierte und damit den Vorläufer der heutigen Infotainmentsysteme darstellt (Amores 2011).

2.1.1 Funktionen

Ein Infotainmentsystem besitzt stets eine *Headunit*, die die verschiedenen Soft- und Hardwarekomponenten miteinander vernetzt und eine zentrale Steuerung ermöglicht. Aktuelle Infotainmentsysteme bieten Funktionen aus den folgenden Bereichen an (s. Wietzke und Tran 2005, Kap. 2):

- Information (z.B. Navigation, Verkehrsmeldungen, Online-Dienste)
- Unterhaltung (z.B. Radio, DVD, mp3, Fernsehen, Internet, Rear-Seat-Entertainment)
- Telekommunikation (z.B. Telefon, SMS, E-Mail, Anbindung mobiler Geräte)
- Sicherheit (z.B. Fahrerassistenzsysteme wie Spurerkennung und Parkassistent)
- Autoeinstellungen (z.B. Klimaanlage, Sitzposition)
- Systemverwaltung (z.B. Ressourcenverwaltung, Download einer neuen Software-Version)

Der tatsächliche Funktionsumfang eines Infotainmentsystems variiert je nach Automarke und Modell und ist oftmals auch länderspezifisch.

2.1.2 HMI

Die Interaktion mit dem Benutzer erfolgt über verschiedene Mensch-Maschine-Schnittstellen (HMI, engl. Human Machine Interface). Ein wichtiger Bestandteil des HMI ist die grafische Benutzeroberfläche, im Folgenden als GUI bezeichnet (engl. graphical user interface). Über die GUI werden dem Benutzer die vielfältigen Informationen zentral auf einem Display dargestellt. Über weitere HMIs kann der Benutzer Einstellungen vornehmen und so mit dem Infotainmentsystem interagieren. Herkömmliche HMIs sind Dreh-Drück-Steller und Tasten. Für mehr Komfort und Sicherheit während des Fahrens werden zudem vermehrt Sprachdialogsysteme angeboten. Vorrangig in den Premiummodellen kann die Bedienung mittlerweile auch über Touchpads oder Touchscreens erfolgen.

2.2 Audi Infotainmentsystem

Das Audi Infotainmentsystem wird als *MultiMediaInterface* (MMI) bezeichnet. Im Jahr 2001 wurde es im Rahmen der Konzeptstudie *Avantissimo* veröffentlicht (*Konzeptstudie Audi Avantissimo* 2012). Seitdem weißt es, trotz der stetig wachsenden Funktionsvielfalt, ein einheitliches Bedienkonzept auf, bei der sich die Tastenanordnung des Bedienteils im Aufbau der GUI widerspiegelt. Abb. 2.1 zeigt dies exemplarisch an der MMI Version 2G von 2002. In Abb. 2.2 ist die Veränderung der GUI anhand des Telefon- und des Navigationskontextes über die Jahre zu erkennen. Vorerst nur in den Premiummodellen A6, A7 und A8 integriert, ist es mittlerweile auch in den Mittelklassewagen erhältlich. Je nach Modell und MMI-Version umfasst es die im vorherigen Abschnitt genannten Funktionen und HMIs. Daneben wird das hinter dem Lenkrad befindliche Kombinationsinstrument immer stärker in die Darstellung der Funktionen einbezogen. Auf einem kleinerem Display können die Informationen dort direkt im Blickfeld des Fahrers angezeigt werden. Auch das Tachometer und weitere Zeigeranzeigen sind zunehmend digital und mit Zusatzinformationen ausgestattet.

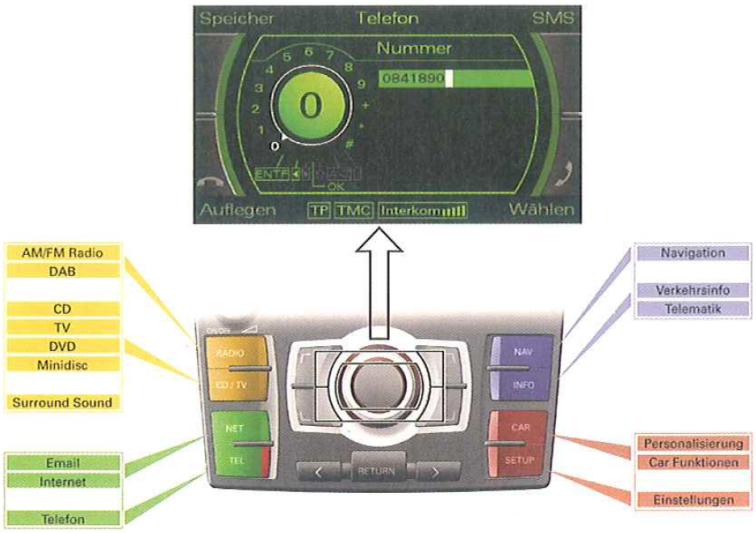


Abbildung 2.1: Einheitliche Bedienlogik des Audi MMI am Beispiel der Version 2G von 2002. Oben: Screenshots der MMI-GUI; Unten: Bedienteil mit zum Screendesign passender Funktionstastenanordnung (Tièschky u. a. 2002, S. 13)

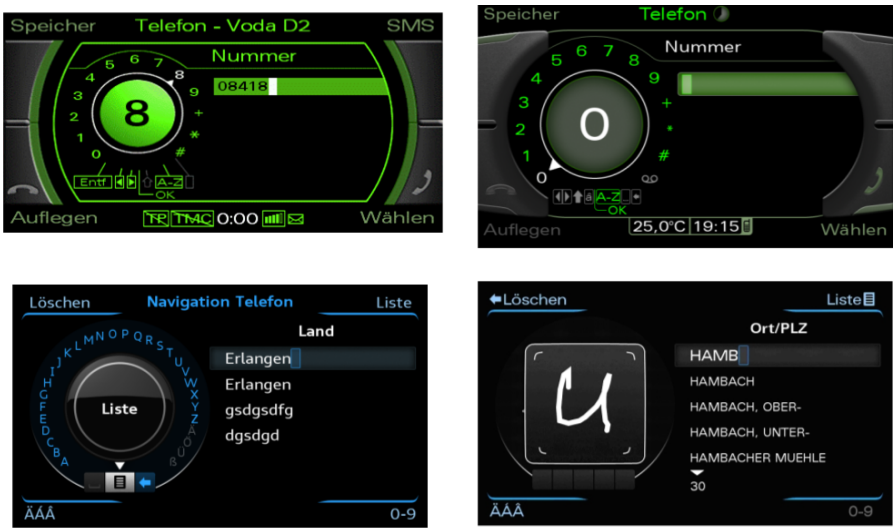


Abbildung 2.2: GUI von vier Versionen des MMI. Oben: Telefonmenü beim 2G- und 3G-System. Unten: Zieleingabe im Navigationsmenü beim 3G+-System mit Touchpadeingabe und dem MIB-System.

2.2.1 MMI GUI

Die verschiedenen MMI-Versionen variieren in ihrem Funktionsumfang und auch in ihrer grafischen Darstellung. Bspw. gibt es sogenannte Basic MMI-Systeme, deren Displays eine Auflösung von 400 x 240 Pixel besitzen. High MMI-Systeme haben hingegen eine Auflösung von 800 x 480 Pixel.
Die GUI besteht aus fünf Hauptmenüs bzw. Kontexten, die sich in ihrer Grundfarbe voneinander unterscheiden: der orange Medien- und Radio-Kontext, der rote Car-Kontext, der grüne Telefon-Kontext, der blaue Navigations-Kontext, und das weiße Setup. Alle Kontexte weisen dieselbe Grundstruktur mit Unterteilung in Screenbereiche auf, die in Abb. 2.3 exemplarisch dargestellt ist. Die Screenbereiche Titel, Untertitel, Status und Softkey sind in jedem Kontext stets vorhanden und ihr Aufbau ist bis auf wenige, vorrangig farbliche und textuelle, Unterschiede identisch. Der Inhalt des Hauptbereichs ist stattdessen sehr unterschiedlich und abhängig von den jeweils darzustellenden Informationen, die in Listen, Speller (s. Abb. 2.2) und weiteren Darstellungsformen repräsentiert werden. Ein konkreter Inhalt einer MMI-GUI wird als *View* bezeichnet. Abb. 2.4 zeigt exemplarisch verschiedene Ausprägungen der View *Setup-Wizardmenü*, die im Hauptbereich die Informationen in einem sogenannten *Wizard* darstellt.

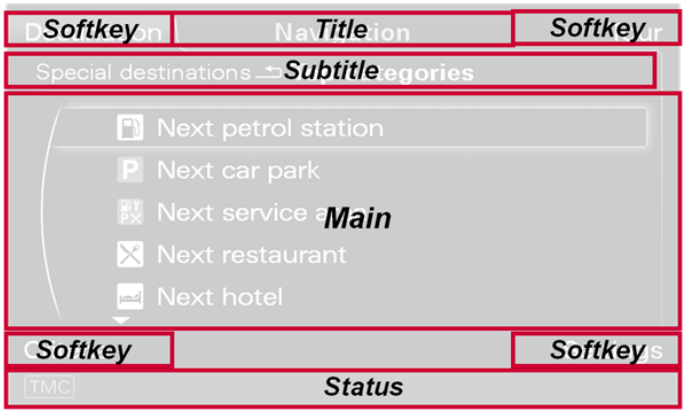


Abbildung 2.3: Grundlegende Screenbereiche einer MMI-GUI.

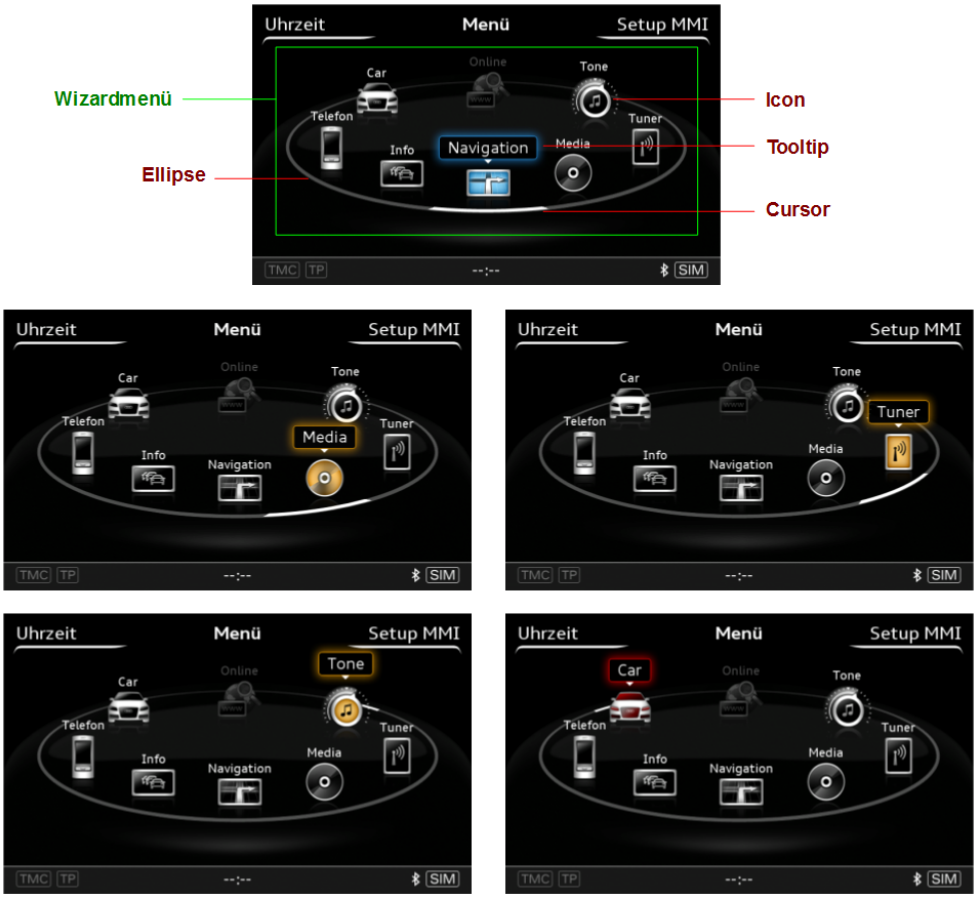


Abbildung 2.4: . View *Setup-Wizardmenü* des MIB High. Die Screenshots zeigen dieselbe View mit einem jeweils anderen selektierten Icon und verschiedenen Cursorpositionen. Im oberen Screenshot sind die einzelnen Elemente des Wizardmenüs gekennzeichnet.

Basierend auf dieser Definition besteht eine GUI aus einer Menge von Views, die durch eine Menge von Screenshots abgebildet werden können. Die Größe der Screenshots-Menge kann je nach View sehr unterschiedlich sein und ist abhängig von der Funktionalität der View und wie sich diese auf die grafische Darstellung auswirkt. Abbildung 2.4 zeigt nur einen kleinen Teil aller möglichen Screenshots der *Setup-Wizardmenü*-View: Es sind noch einige weitere Cursorpositionen möglich und auch die Iconzustände können in einer Vielzahl von Kombinationen auftreten. Die Anzahl an Views pro GUI ist abhängig vom Funktionsumfang. Das MMI der aktuellen Version MIB umfasst bspw. über 3000 Views.
Die Elemente einer GUI können textueller oder symbolischer Art sein. Elemente mit textuellen Inhalten werden im Folgenden als *Textelemente* bezeichnet; ein Beispiel ist eine Textzeile in einer Liste. Elemente mit symbolischen Inhalten werden im Folgenden als *Symbolelemente* bezeichnet; z.B. Cursor, Fortschrittsbalken. Besondere Symbolelemente sind Piktogramme, die im Folgenden als *Icon* bezeichnet werden. In Abb. 2.5 sind die Icons des *Setupmenü-Wizards* (s. Abb. 2.4) in ihren jeweils drei visuellen Ausprägungen aufgeführt. Zu bemerken ist die unterschiedliche Farbe der selektierten Icons, die der Farbe des zugehörigen Kontextes entspricht.

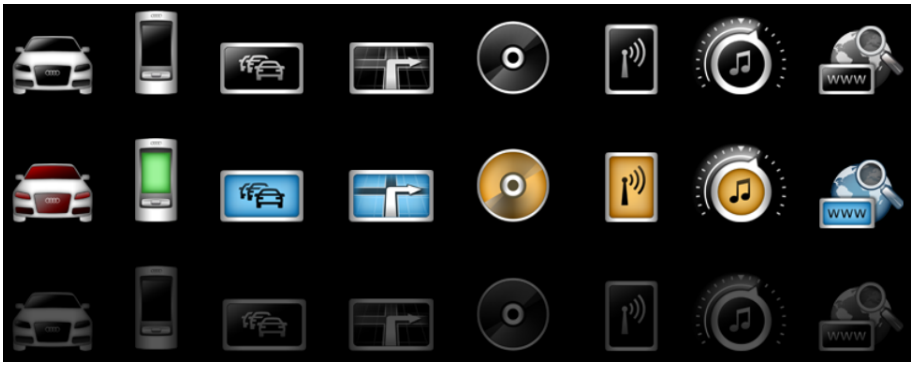


Abbildung 2.5: . Die acht Icons der View *Setup-Wizardmenü* des MIB High mit ihren zustandsabhängigen Ausprägungen von oben nach unten: *aktiv, selektiert, inaktiv*.

Einige Elemente sind genau genommen *Widgets* (dt. Steuerelemente), die neben einer grafischen Darstellung auch eine Funktionalität besitzen. Eine Schaltfläche, z.B. dargestellt durch ein Icon, die bei Betätigung durch den Benutzer den Wechsel in ein anderes Menü auslöst oder ein Scrollbalken zur Navigation in einer Liste, sind z.B. Widgets. Viele dieser Widgets bestehen dabei aus mehreren atomaren gra-

fischen Elementen. Bspw. ist bei der MMI-GUI ein Softkey eine Schaltfläche, die aus einem Text, einer Hintergrundfläche und einem Rahmen besteht. Die atomaren Elemente und die Widgets sind demnach hierarchisch in einer GUI geordnet. Beim GUI-Design werden die einzelnen, atomaren grafischen Elemente betrachtet, weswegen der Begriff „Element“ anstelle von „Widget “ innerhalb dieser Arbeit verwendet wird.

2.2.2 GUI-Design

Für jedes MMI ist das Layout und Design der GUI klar spezifiziert (s. Abschnitt 2.2.2.3) und lässt sich in lokale und globale Designmerkmale unterteilen. Ein Merkmal bezieht sich dabei stets auf Elemente einer View zu einem bestimmten Zeitpunkt. Globale Merkmale zwischen Elementen verschiedener Views sowie der zeitliche Verlauf ist für das hier betrachtete Layout und Design nicht relevant.
Im Folgenden werden die Merkmale genauer erläutert. Die Abb. 2.6 veranschaulicht einige Designmerkmale exemplarisch an einem konkreten Screenshot des MMI.

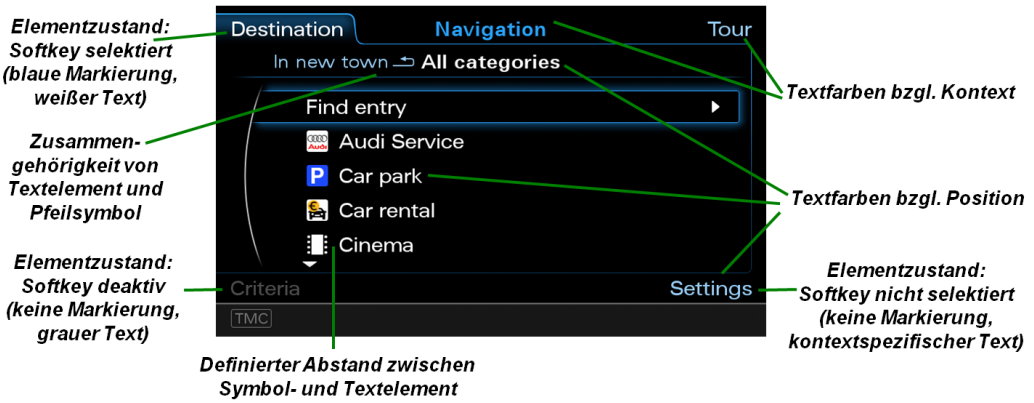


Abbildung 2.6: Designmerkmale eines MMI-Screenshots.

2.2.2.1 Lokale Merkmale

In Tabelle 2.1 sind die lokalen Merkmale mit Beispielen angegeben. Die durchschnittlichen RGB-Werte ergeben sich als Quotient der aufsummierten RGB-Werte durch die Gesamtanzahl. Neben RGB können auch andere Farbräume verwendet werden. Neben den Merkmalen, die für alle Elemente gelten, werden zusätzlich für Symbolelemente das Referenzbild und für Textelemente der Font mit dem Soll

verglichen. Ein Font lässt sich anhand verschiedener Einzelmerkmale wie Serifen, Buchstabenabstand und Proportionalität[1] klassifizieren. Daneben können Elemente als Teil von Widgets verschiedene Zustände annehmen, die eine grafisch dargestellte *Semantik* besitzen (vgl. Abb. 2.5). Ein Softkey im aktiven Zustand ist z.B. mit einem weißen Text beschriftet und im inaktiven Zustand mit einem grauen. Die Farbe der Hintergrundflächen und die Art des Rahmens hängt ebenfalls vom Zustand ab.

Elementtyp	**Merkmal**	**Erläuterung/Beispiel**
Text, Symbol	(Erkenner-)Sicherheit	Symbolelement s von Symbolerkenner mit 0.95 Wahrscheinlichkeit identifiziert
	Existenz	obligatorisch oder optional
	Position	absolute x- und y-Koordinate im Screenshot; z.B. obere linke Ecke oder Mittelpunkt
	Elementzustand	lokale Semantik; z.B. selektiert, inaktiv
	Farbe	Durchschnittliche RGB-Werte
	Dimension	Breite und Höhe in Pixel; bei dynamischen Textinhalten minimale/maximale Dimension
Text	Font	Fontdatei
Symbol	Referenz(-bild)	Bilddatei

Tabelle 2.1: Lokale Merkmale von Designelementen.

2.2.2.2 Globale Merkmale

Des Weiteren existieren Beziehungen zwischen Elementen, d.h. *globale* Merkmale. Globale Merkmale können Elemente eines Screenbereichs betreffen oder auch bereichsübergreifend definiert sein. In Tabelle 2.2 sind alle globalen Merkmale mit Beispielen aufgelistet. Einige Beispiele beziehen sich auf das Setup-Wizardmenü in Abb. 2.4.

[1] Bei einem proportionalen Font haben verschiedene Buchstaben eine individuelle Breite, z.B. Times New Roman. Bei einem nicht-proportionalen Font besitzen hingegen alle Buchstaben dieselbe Breite, z.B. `Courier`.

Merkmal	**Erläuterung/Beispiel**
Abstand	Abstand zwischen zwei Elementen in x- oder y-Richtung in Pixeln
Ausrichtung	Ausrichtung eines Elements bzgl. seines Screensbereichs: links, zentriert, rechts
Koexistenz	Wenn Symbolelement *S1* existiert, existiert auch Symbolelement *S2*: Ist ein Icon im Setup-Wizardmenü im Zustand *selektiert*, ändert sich das darüberliegende Textelement in einen *Tooltip* bestehend aus Text, Rahmen und Pfeil; Im Setup-Wizardmenü ist der Cursor stets an der Position des selektierten Icons.
gegenseitiger Ausschluss	Wenn Symbolelement *S1* existiert, existiert Symbolelement *S2* nicht: Im Setup-Wizardmenü ist maximal ein Icon im Zustand *selektiert*.
Semantik	Keine semantischen Widersprüche zwischen der optischen Darstellung von Elementen; Das Design von Text und Fläche eines Softkeys muss beides auf den gleichen Elementzustand hindeuten

Tabelle 2.2: Globale Merkmale von Designelementen.

2.2.2.3 Spezifikation

Das Design und Layout des MMI wird von Grafikdesignern in Photoshopdateien (PSD) (*Adobe Photoshop* 2012) entworfen. Eine PSD-Datei beinhaltet meist verschiedene Views zu einem Kontext. Die Abb. 2.7 zeigt eine PSD-Datei zusammen mit ihrer Layerstruktur. Je nachdem, welche Layer als sichtbar markiert sind, können verschiedene Views und Inhalte angezeigt bzw. zusammengestellt werden; z.B. können Iconzustände geändert und Tooltip-Elemente ein- und ausgeblendet werden.

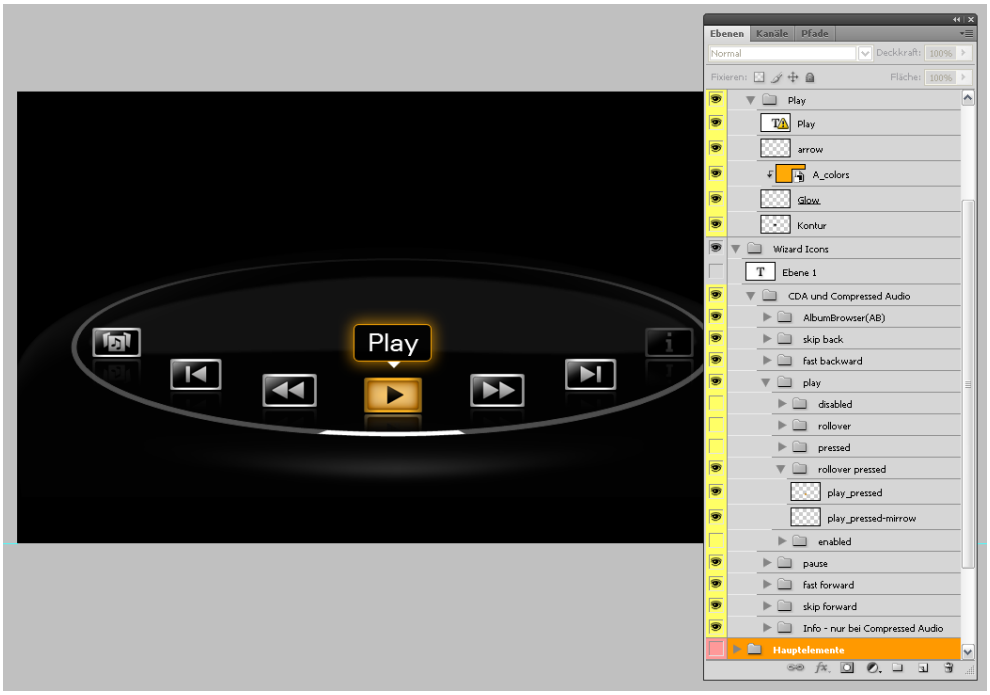


Abbildung 2.7: Eine PSD-Datei mit ihrer Layerstruktur. Die aktuelle Kombination sichtbarer Layer stellt ein Wizardmenü des Medien-Kontextes dar.

Weitere Informationen, insbesondere Designlogiken, werden in PDF-Dokumenten in textueller und visueller Form spezifiziert, bspw. der Aufbau und die Bedienlogik für Wizardmenüs (s. Abb. 2.8) sowie die verschiedenen Zustände eines Icons (s. Abb. 2.9 unten). Positionierungen von Elementen und Abstände sowie Fonts sind ebenfalls definiert (s. Abb. 2.10). Die letztgenannten drei Merkmale sind auch Inhalt der PSD-Dateien und damit redundant beschrieben. In den PDF-Dokumenten wird darauf hingewiesen, dass bei Unstimmigkeiten die Angaben in den PSD-Dateien gelten. Die Positionierungen in den PSD-Dateien sind zudem pixelverbindlich. Die PSD-Dateien bilden damit die Grundlage der Designspezifikation. Weitere Eigenschaften der PSD-Spezifikation sind:

- Die PSD-Dateien repräsentieren das Design und Layout einer MMI-Version anhand von Beispielen. D.h. die PSD-Dateien bilden nicht jeden möglichen Screenshot einer View ab. Insbesondere werden Texte in nur einer Sprache (normalerweise Deutsch oder Englisch) dargestellt. Dynamische Textinhalte, z.B. Listen von Radiosendern, sind durch Beispieltexte veranschaulicht. Die Dynamik mancher Elemente, z.B. die Abhängigkeit der Breite eines Tooltip-Elements vom konkreten Textinhalt, ist damit nicht in den PSD-Dateien ver-

merkt. Sind für ein Element verschiedene Positionen möglich, ist oft meist nur eine Untermenge davon visualisiert; z.B. werden nicht alle möglichen Cursorpositionen für ein Wizardmenü visualisiert.

- Das Grundlayout (s. Abb. 2.3) ist oftmals redundant in verschiedenen PSD-Dateien enthalten.

- Die PSD-Datei an sich besitzt keine Logik, wie Layer zusammengestellt werden. Damit können auch Elemente miteinander kombiniert werden, die keine korrekte GUI darstellen.

- Die PSD-Dateien weisen mitunter keine einheitliche Struktur und Bezeichnung der Layer auf.

- Im PSD-Designentwurf können Positionen auf Subpixelebene angegeben werden, d.h. bspw. die Position $x = 5.5$.

Damit beinhalten die PSD-Dateien lediglich Beispiele für das Design und Layout einer MMI-Version und insbesondere keine logischen Zusammenhänge. Alle Kombinationsmöglichkeiten und Designlogiken sind abstrakt in den PDF-Dokumenten definiert, die die Spezifikation vervollständigen.
Das Design der MMI-GUI ist damit informell spezifiziert und gelangt in dieser Form auch zu den Entwicklern, die die Vorgaben manuell umsetzen.

Elements generally used in wizards:

Background	Background graphic for wizards (various depending on wizard type)
Buttons	Graphic
Label	One- or two-line Font
Cursor	Element in contact with the circle of the background graphic, moves along the elliptical path in animated form
Cursor box	Bitmap Cursor box size dynamically depending on text length
Information box (depending on the wizard)	

Wizards may occur both in the central menu (no softkey menu open) and within an open softkey menu.

Abbildung 2.8: Ausschnitt aus dem Spezifikationsdokument [Betz 2010] zur Definition des Aufbaus und der Funktionsweise eines Wizardmenüs im 3G+.



Abbildung 2.9: Definition von Positionierungen und Iconzuständen für das 3G+ im Spezifikationsdokument [Promoli 2011].

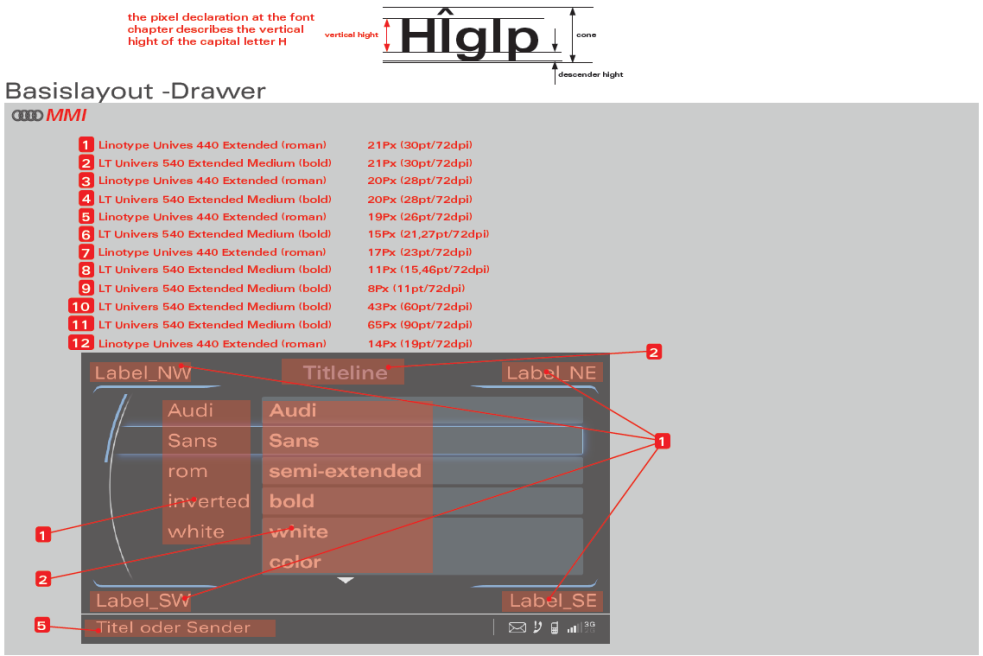


Abbildung 2.10: Definition der Fonts für das 3G+ im Spezifikationsdokument [Promoli 2011].

2.2.2.4 Navigationskarten

Der Vollständigkeit halber sei noch erwähnt, dass die GUIs auf dem Display des Kombinationsinstruments und für die Darstellung einer Navigationskarte nicht mehr die typische Einteilung in die vorgestellten Screenbereiche aufweisen. Es gelten aber auch für diese GUIs spezifizierte Designvorgaben [2]. Im Rahmen der Arbeit wird auf die Designverifikation dieser Inhalte allerdings nicht weiter Bezug genommen.

2.3 HMI-Testautomatisierung

Das Testen ist ein wichtiger Teil des Qualitätsmanagements bei der Entwicklung von Hard- und Software. Vereinfacht besteht ein Test immer darin, das beobachtete Ist mit dem erwarteten Soll zu vergleichen; stimmen die Werte nicht überein, wurde ein Fehler im System gefunden. Ein Testwert kann ein einzelnes Ausgabeergebnis einer Hard- oder Softwareschnittstelle oder auch komplexes Systemverhalten darstellen.
Da heutige Systeme sehr umfangreich sind, stellt manuelles Testen einen zu hohen personellen und zeitlichen Aufwand dar. Aus diesem Grund ist seit einigen Jahren die Testautomatisierung ein wichtiges Thema. Die Testautomatisierung ist ein komplexes Gebiet, das im Rahmen dieser Arbeit nur kurz vorgestellt wird. Ziel ist sowohl die automatisierte Testdurchführung als auch die automatisierte Testgenerierung basierend auf den gegebenen Testanforderungen (Stolle u. a. 2005).

2.3.1 Verifikation und Validierung

Die Testinhalte lassen sich mit den Begriffen *Verifikation* und *Validierung* in zwei Bereiche gliedern. Bei der Verifikation wird das System basierend auf der Spezifikation getestet und damit ein formaler Nachweis erbracht, dass das System der Spezifikation entspricht (vgl. Hoffmann 2008, Kap. 6.1). Verifikationstechniken basieren häufig auf logischen und mathematischen Formalismen bzw. Regeln. Die Validierung gibt hingegen eine Aussage darüber, ob das System die Anforderungen

[2] Die Navigationskarten unterliegen nur Audi-Designspezifikationen, sofern es sich um die von Audi entwickelte Kartendarstellungen handelt. Daneben kann das MMI, je nach Funktionsumfang, auch Karten von Drittanbietern, z.B. die Satellitenansicht von Google, anzeigen.

der Benutzer oder des Anwendungsbereichs erfüllt. Daneben kann die Validierung von Eingabedaten auch bedeuten, diese auf Plausibilität zu prüfen, bspw. ob sie im geforderten Wertebereich liegen.
Bei der Verifikation werden Fehler in der Implementierung gefunden, indem Widersprüche zur Spezifikation entdeckt werden. Ist die Spezifikation bereits fehlerhaft, können diese Fehler nicht gefunden werden. Höchstens kann dann durch die Validierung festgestellt werden, dass das System nicht das gewünschte Verhalten aufweist. Das Auffinden von Spezifikationsfehlern ist sehr problematisch und es existieren keine formalen Testverfahren (Hoffmann 2008, Kap. 2.8). Es ist außerdem möglich, dass ein System basierend auf der Spezifikation korrekte Ergebnisse liefert, aber die Benutzeranforderungen völlig verfehlt.[3]

2.3.2 Fehlervermeidung

Die Fehlerkosten steigen nicht-linear im Laufe des Entwicklungsprozesses, was erstmals von [Juran u. a. 1999] festgestellt wurde. Eine spätere Erkennung der Fehler führt damit zu einem immensen Anstieg der Fehlerbehebungskosten, weswegen Tests bereits in frühen Entwicklungsphasen durchgeführt werden sollten.
In der Regel kann nicht sichergestellt werden, dass ein System fehlerfrei ist. Durch eine geeignete Definition und Durchführung von Testfällen kann lediglich eine hohe Testabdeckung erreicht und damit die Fehleranzahl wesentlich verringert werden. Aus Sicht des Testaufwands ist dabei das Ziel, mit möglichst wenig Testfällen eine sehr hohe Testabdeckung zu erreichen. Darüber hinaus ergeben sich viele weitere Fragestellungen; bspw. die automatisierte Feststellung der Testabdeckung. Tests können außerdem basierend auf typischen Benutzerszenarien generiert werden, um die oftmals immense Menge der möglichen auf sinnvolle Testfälle einzuschränken.

Da die Bedienung eines Infotainmentsystems meist vom Fahrer während des Fahrens erfolgt, müssen für das automobile HMI besondere Sicherheitsregeln gelten (Kern und Schmidt 2009). Das 1998 erstmals veröffentlichte und seitdem stetig aktualisierte „European Statement of principles on HMI “ (European Communities 2006) liefert für die automobile HMI-Entwicklung Vorschläge, um eine Ablenkung des Fahrers zu vermeiden. Auch in [Stevens u. a. 2002] werden Richtlinien für

[3] Es gibt auch andere, teilweise konträre Definitionen für die Begriffe *Verifikation* und *Validierung* (vgl. Grady 2007, Kap. 1.2.6).

den sicheren Umgang mit Infotainmentsystemen diskutiert und [Kern und Schmidt 2009] befasst sich mit der geeigneten, ablenkungsarmen räumlichen Anordnung der Funktionen im Automobil.

2.3.3 Modellbasiertes Testen

Für das automatisierte Testen der Verhaltenslogik von HMI ist das modellbasierte Testen eine gängige Methode. Dabei bildet ein Systemmodell das Systemverhalten oder die Benutzerinteraktionen ab. Ein Systemmodell kann um zusätzliche testspezifische Parameter, z.B. Reaktionszeiten, zu einem Testmodell erweitert werden. So kann z.B. die Menüstruktur einer GUI oder die Abfolge von Sprachein- und -ausgaben eines Sprachdialogsystems automatisiert getestet werden. Für die Modellierung liefert UML (engl. Unified Modeling Language) mit seinen Zustandsdiagrammen ein mächtiges Werkzeug (Reich 2008). Eine hohe Testabdeckung wird erreicht, wenn die Modelle möglichst alle Interaktionspfade abdecken und jede Funktion mindestens einmal durchgeführt wird, gegebenenfalls mit verschiedenen Eingabeparametern. Ein weiteres Ziel des modellbasierten Testens ist die automatisierte oder zumindest teilautomatisierte Erstellung der Testmodelle selbst. Auch hier gibt es aktuelle Forschungsprojekte, z.B. [Duan, Höfer und Hussmann 2010]. Für die automatisierte Erstellung von Testmodellen ist eine essentielle Voraussetzung die Existenz einer formalen, maschinenlesbaren Spezifikation, die das HMI-Verhalten beschreibt. [Bringmann und Krämer 2008] gibt eine gute Einführung in das modellbasierte Testen im Automobilbereich und [Huang u. a. 2010a] befasst sich speziell mit dem Testen von Infotainmentsystemen.

2.3.4 Ansätze aus der Künstlichen Intelligenz

Anstatt das HMI-Verhalten und die Benutzerinteraktionen wie beim modellbasierten Testen üblich in Form von Zustandsdiagrammen abzubilden, gibt es auch andere Möglichkeiten der Modellierung. In der *Künstlichen Intelligenz* werden Verfahren entwickelt, um menschliches Verhalten mit intelligenten Computersystemen zu simulieren und damit verschiedene Aufgaben zu automatisieren. Das ist auch gerade das Ziel der Testautomatisierung. Ein Teilgebiet der Künstlichen Intelligenz sind die *wissensbasierten Systeme* (s. Abschnitt 4.1), mit denen Wissen auf unterschiedlichste Weise modelliert und verarbeitet werden kann. Welche Modellierung kon-

kret verwendet wird, ist abhängig von der jeweiligen Problemstellung. Bayes'sche Netze (s. Kap. 5) eignen sich z.B. für die Auswertung von unsicheren Informationen.
Die automatische Erstellung des Modells spielt auch hier eine wichtige Rolle und ist für den praktischen Einsatz umso entscheidender, je komplexer das zu modellierende Wissen ist. In [Borth und Hasseln 2002] wird z.B. für die Diagnose eines technischen Systems ein Bayes'sches Netz direkt aus einer formalen Systemspezifikation generiert.

2.3.5 Bildverarbeitung in der Testautomatisierung

Um beim automatisierten Testen von GUI-basierten Systemen die GUI-Inhalte zu erkennen, ist der Einsatz von Bildverarbeitungsmethoden notwendig. Die erkannten Inhalte geben dann Rückschluss über das angezeigte Menü und den aktuellen Systemzustand. Im Testverfahren von [Chang, Yeh und Miller 2010] werden z.B. Buttons erkannt, um eine automatisierte Menünavigation zu ermöglichen.
Auch die GUI-Inhalte selbst können verifiziert werden. [Huang u. a. 2010b] illustriert den Aufbau eines Testsystems für Infotainmentsysteme und geht dabei auch auf den Einsatz von Bildverarbeitungsmethoden ein. Mittels Symbol- und Texterkennung können die Inhalte der GUI erkannt und mit Sollwerten verglichen werden. Darüber hinaus werden Farben identifiziert, um auf den Zustand von Buttons zu schließen.

2.3.5.1 Bildverarbeitung allgemein

Im Folgenden werden die einzelnen Schritte einer typischen Bildverarbeitung erläutert, bei der die *Bilderkennung* der letzte Schritt ist (vgl. Erhardt 2008). Andere Definitionen sind möglich, vor allem in Abhängigkeit des jeweiligen Anwendungsbereichs der Bildverarbeitung (Erhardt 2008).

1. **Bildbearbeitung oder - vorverarbeitung**: Die Bildbearbeitung dient der Bildverbesserung, um die darauf folgende Analyse zu vereinfachen und die Ergebnisse der Erkennung zu optimieren. Hier werden oftmals *Bildfilter* angewendet, um z.B. das Rauschen zu mindern, den Kontrast zu verstärken oder die Konturen zu schärfen. Auch die Umwandlung eines Farbbildes in

ein Grauwert- oder Binärbild oder der Wechsel in einen anderen Farbraum kann ein sinnvoller Vorverarbeitungsschritt sein.

2. **Bildanalyse**: Das Ergebnis der Bildanalyse ist eine parametrisierte Beschreibung des gesamten Bildes oder bestimmter Bildbereiche. Mittels eines Histogramms kann z.B. die Häufigkeit der Farbwerte bestimmt werden. Weitere Merkmale sind Größe und Form von bestimmten Bildbereichen.

3. **Bilderkennung**: Auf Basis der analysierten Merkmale werden die Bildinhalte klassifiziert, d.h. ihnen eine Semantik zugeordnet. Z.B. identifizieren Texterkennungsverfahren Bildbereiche als Schriftzeichen.
   Eine Bilderkennung kann relativ einfach als Bildvergleich durchgeführt werden. Das *Template Matching* ist bspw. ein Verfahren, mit dem gegebene Referenzbilder in einem Bild gesucht werden. Der Bildvergleich kann als einfacher Vergleich zwischen den Farbwerten der Pixel realisiert sein, oder auch komplexere Distanzmaße (s. auch Kap. 5.4.1) anwenden. Der Vergleich kann zudem skalierungs- und rotationsinvariant sein.
   Daneben bietet das Gebiet der *Mustererkennung* umfangreiche Möglichkeiten zur Klassifikation nicht nur grafischer Merkmale an. Es gibt bspw. eine Vielzahl linearer Klassifikatoren (s. z.B. Bishop 2006, Kap. 4) und oftmals werden auch Konzepte aus der Künstlichen Intelligenz, z.B. Neuronale Netze (s. z.B. Bishop 2006, Kap. 5), eingesetzt.

2.3.5.2 Voraussetzungen

Um Bildverarbeitung für ein GUI-basiertes System durchführen zu können, muss die auf einem Display angezeigte Momentaufnahme der GUI (Screenshot) als Bilddatei vorliegen. In der Regel kann dies mit einer Kamera, die den Displayinhalt fotografiert, vorgenommen werden. Darüber hinaus bieten manche Geräte eine Framegrabber-Schnittstelle an, die das direkte Abgreifen des digitalen Bildsignals ermöglicht. Das mit einem Framegrabber aufgenommene Bild hat gegenüber einem Kamerabild den Vorteil, das es störungsfrei ist. Bei einer Kameraaufnahme muss hingegen präzise auf eine geeignete Ausrichtung und die Vermeidung von Störquellen wie einfallendes Licht geachtet werden. Eine Kamera stellt jedoch umfangreichere Möglichkeiten bereit um bspw. den Bildkontrast positiv für die spätere Auswertung zu verändern. Solche Bildbearbeitungen können aber auch noch nachträglich mittels

verschiedener Filter vorgenommen werden (s. oben). Unabhängig von der Aufnahmetechnik kann eine geringe Bildauflösung oder verlustbehaftete Kompression zu Problemen bei der Auswertung und fehlerhaften Ergebnissen führen.

2.3.5.3 Unsicherheit von Bildinformationen

Informationen aus Bildern sind naturgemäß mit Unsicherheiten behaftet. Bspw. kann bei genauerer Betrachtung eines MMI-Screenshots festgestellt werden, dass der Hintergrund der MMI-Menüs oft nicht, wie spezifiziert, rein schwarz ist (RGB=(0,0,0)), sondern die Farbwerte der Pixel schwanken. Derartige Farbabweichungen können durch das Rendering entstehen. Daneben wird häufig Kantenglättung (engl. anti-aliasing) vorgenommen, um Farbübergänge zu glätten und damit das Pixelraster zu maskieren. Andere optische Verbesserungsmaßnahmen, z.B. Subpixel-Rendering (s. z.B. Dixon, Leventhal und Fogarty 2011), können ebenfalls zu Farbabweichungen und auch zu Positionsabweichungen führen. Die Konturen einzelner Elemente sind in diesen Fällen auch nicht mehr eindeutig begrenzt, wie das Beispiel in Abbildung 2.11 zeigt.

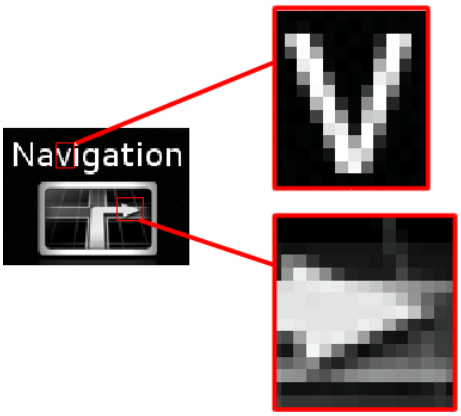


Abbildung 2.11: Auswirkung der Kantenglättung auf GUI-Elemente am Beispiel eines Ausschnitts eines MMI-Screenshots.

Die häufig erwünschten Abweichungen von den Sollwerten sind vom Betrachter oftmals nicht wahrnehmbar, für ein Computersystem sind sie jedoch sehr problematisch. Mit einer Kamera erzeugte Screenshots können zudem weitere Störungen, wie Helligkeitsunterschiede und Verzerrungen, beinhalten, durch die noch größere Abweichungen zu den spezifizierten Werten entstehen.

Aus diesen Gründen muss die Bilderkennung in einem gewissen Maße tolerant gegenüber Abweichungen sein. Für einen pixelbasierten Bildvergleich heißt das, dass nicht nur exakte Farbübereinstimmungen gelten, sondern auch Farbwertabweichungen in einem definierten Intervall zulässig sind. Das führt allerdings auch zu unsicheren Erkennerergebnissen. D.h. oft kann nicht mit Sicherheit entschieden werden, um welchen Buchstaben oder welches Symbol es sich tatsächlich handelt, da bereits die auszuwertenden Merkmale von den spezifizierten Werten abweichen.

2.3.5.4 Erkennung von GUI-Inhalten

Es gibt zahlreiche pixelbasierte Verfahren, um die Elemente bzw. Widgets einer GUI in Screenshots zu identifizieren. Wenn zusätzlich Wissen über die Struktur der GUI gegeben ist, kann mit den erkannten Elementen anschließend die GUI-Hierarchie rekonstruiert werden. Liegen alle Elemente der GUI als Referenzbilder vor, kann ein einfaches Template Matching für die Erkennung angewendet werden. Bei einer hohen Auflösung der Screenshots und einer großen Menge an Referenzbildern ist dies allerdings sehr rechenintensiv. [Dixon, Leventhal und Fogarty 2011] stellt ein echtzeitfähiges pixelbasiertes Verfahren vor: Für zur Laufzeit nicht veränderliche Widgets können bestimmte Pixelanordnungen als Merkmale verwendet werden, um das jeweilige Widget zu erkennen. Dabei wird in der Regel nicht ein vollständiges Bild eines Widgets zum Vergleich verwendet, sondern nur Regionen aus diesem. Je weniger und je kleinere Regionen zur Beschreibung eins Elements notwendig sind, umso effizienter ist der Erkenneralgorithmus.
In [Dixon, Leventhal und Fogarty 2011] werden die GUI-Inhalte zum Zweck der Adaption einer bestehenden GUI in ein anderes Look&Feel oder für (benutzerspezifische) Änderungen (z.B. Übersetzung der Texte) zur Laufzeit erkannt. Deshalb spielt hier die Performanz der Erkennung eine entscheidende Rolle. Mit pixelbasierten Verfahren kann also eine GUI verändert werden, ohne dass dazu Zugang zur eigentlichen Applikation bzw. Änderungen in deren Quellcode vorgenommen werden müssen. Stark von Kantenglättung und weiteren optischen Verbesserungsmaßnahmen betroffene Widgetinhalte wie bspw. Schriftzeichen können laut [Dixon, Leventhal und Fogarty 2011] mittels dieses pixelbasierten Verfahrens allerdings nicht geeignet identifiziert werden.
Im Umfeld der Dissertation ist im Rahmen einer Masterarbeit (Schad 2011) eine Bildverarbeitung entstanden, die basierend auf Fontinformationen und Referenzbil-

dern alle Text- und Symbolelemente einer MMI-GUI identifizieren kann. Das Verfahren basiert auf einer Kantendetektion und der anschließenden Vektorisierung der Kanten. Sowohl der Font und die Referenzbilder, als auch der Screenshot werden in diese Vektorendarstellung transferiert. Danach können durch den Vergleich der Ist- mit den Soll-Vektoren die Elemente erkannt werden. Durch die Vektorendarstellung ist das Verfahren skalierungsinvariant; lediglich die Proportionen zwischen den einzelnen Vektoren müssen in einem gewissen Toleranzbereich den Sollwerten entsprechen. Kantenglättung stört damit dieses Verfahren in der Regel nicht. Die Bildverarbeitung liefert für ein klassifiziertes Element die Erkennersicherheit in Prozent sowie die lokalen Merkmale. Die Performanz der Bildverarbeitung ist gut (s. Schad 2011): Die Vektorisierung ist parallelisierbar und einmal vektorisierte Fonts und Referenzbilder können als Dateien gespeichert werden, sodass sie bei der Erkennung mehrerer Screenshots derselben GUI-Version nicht nochmals erzeugt werden müssen.

2.4 MMI-Testautomatisierung

Der Funktionsumfang des MMI nimmt stetig zu und erfordert automatisierte Tests. Die MMI-GUI bildet in der Regel alle Funktionalitäten visuell ab und ermöglicht eine umfangreiche Benutzerinteraktion. Deshalb wird die GUI bei den MMI-Tests als wesentlicher Bestandteil miteinbezogen.

2.4.1 Testmethoden

Das Testen des MMI geschieht auf zweierlei Art. Zum einen existiert eine große Zahl von *Testwürfeln* und zum anderen werden *Testfahrten* durchgeführt. Bei den Testfahrten wird das bereits ins Fahrzeuge integrierte MMI manuell während der Fahrt getestet und gegebenenfalls Aufzeichnungen von Fehlern gemacht. Mit den Testwürfeln werden manuelle und automatisierte Tests durchgeführt. Dafür umfasst ein Testwürfel in der Regel ein vollständiges MMI mit allen notwendigen Hard- und Softwarkomponenten, so wie es auch im fertigen Auto verbaut ist. Daneben können bestimmte Komponenten auch durch Software oder einen Roboter simuliert werden, bspw. die Erzeugung von Verkehrsmeldungen, eine Spracheingabe, ein Tastendruck und das Einlegen einer CD. Mit den Würfelaufbauten können auf

relativ einfache Weise alle existierenden versions- und länderspezifischen Varianten des MMI eingerichtet und getestet werden.

2.4.2 Aktueller Umfang

Die aktuelle MMI-Testautomatisierung ist modellbasiert (s. Abschnitt 2.3.3). Basierend auf einer System- und Testspezifikation werden die Tests als Zustandsdiagramme modelliert und so die automatische Navigation durch die Menüstruktur des MMI ermöglicht. Die dazugehörige grafische Benutzeroberfläche wird auf dem Display angezeigt und in Form von Screenshots (s. Abschnitt 2.3.5) mittels Bildverarbeitung (s. Abschnitt 2.4.2.1) erkannt und mit Sollwerten verglichen. Für das MMI existiert die Möglichkeit Screenshots mittels eines Framegrabbers zu erstellen.
Bei den MMI-Tests handelt es sich um Online-Tests, d.h. während der Navigation durch das MMI und der Erstellung von Screenshots, werden die Informationen direkt ausgewertet.

2.4.2.1 Bildverarbeitung

Für die Bilderkennung stehen der MMI-Testautomatisierung aktuell eine Texterkennung, eine Symbolerkennung mittels Template Matching (s. Abschnitt 2.3.5) sowie die Identifizierung von Farben und bestimmten Menüstrukturen (z.B. Wizard und Speller) zur Verfügung. Neben der Verifikation der GUI-Inhalte, ist die Bilderkennung auch für die modellbasierte Menünavigation sinnvoll und macht sie flexibler. Es kann z.B. sein, dass bei einer nachfolgenden MMI-Version die Position von Softkeys aufgrund von Änderungen in der Spezifikation vertauscht wurden. Falls im Modell die Button-Positionen hart kodiert sind, wird bei der neuen MMI-Version der falsche Button aktiviert und die Menünavigation schlägt letztendlich fehl bzw. es wird nicht das gewünschte Menü erreicht. Wenn stattdessen durch Bildverarbeitung der richtige Button identifiziert wird, tritt dieses Problem nicht auf und der Test kann ohne Änderungen für beide MMI-Versionen eingesetzt werden. D.h. durch die Ergebnisse der Bildverarbeitung muss die Menünavigation nicht blind erfolgen. Aus diesem Grund ist die Bildverarbeitung ein essentieller Bestandteil der MMI-Testautomatisierung.

Performanz Durch die steigende Komplexität des MMI wird auch der Testumfang stetig größer. Die Tests müssen daher möglichst effizient sein und wenig Zeit beanspruchen. Bei der Bildverarbeitung kann durch Einschränkung des Bildbereichs, in dem Bildverarbeitung durchgeführt wird, eine Zeitersparnis erzielt werden. Wie in Abschnitt 2.2.2 ausgeführt, lässt sich die MMI-GUI in definierte Screenbereiche gliedern. Für die MMI-Tests ist normalerweise bekannt, in welchem Screenbereich sich das Element des Interesses befindet. Darüber hinaus kann der jeweilige Screenbereich evtl. durch weiteres Wissen noch weiter eingeschränkt werden. Dafür werden manuell rechteckige Masken definiert, die zentral verwaltet und automatisiert eingelesen werden.

Behandlung von Unsicherheit Bei der aktuellen MMI-Testautomatisierung werden unsichere Bildinformationen (s. Abschnitt 2.3.5.3) behandelt, indem eine harte Entscheidung mittels eines Schwellwerts vorgenommen wird. Z.B. wird ein Symbol als erkannt gewertet, wenn die Sicherheit des Symbolerkenners mindestens 80% beträgt (vgl. Heilemann und Palm 2011, Kap. 2.4). Diese Vorgehensweise hat sich bei den existierenden MMI-Tests bewährt und ist bei diesen bislang ausreichend.

Kapitel 3

Formale Spezifikationssprachen

Nach allgemeinen Bemerkungen zu formalen Sprachen und Spezifikationen, werden in den Abschnitten 3.2 und 3.3 die Auszeichnungssprache *XML* und das Datenaustauschformat *JSON* detailliert vorgestellt und miteinander verglichen. Anschließend liefert Abschnitt 3.4 einen Überblick über einige existierende GUI-Spezifikationssprachen; auch die Spezifikation von GUI-Design wird diskutiert. Des Weiteren wird erläutert, wie Entwicklung und Spezifikation aufeinander abgestimmt werden können.

3.1 Formale Sprachen und Spezifikationen

Formale Sprachen werden für computergesteuerte Abläufe eingesetzt und haben dort den Anspruch maschinenlesbar zu sein. Programmiersprachen sind z.B. formale Sprachen. Daneben werden formale Sprachen auch für Dokumente verwendet, die für den Menschen lesbar sein sollen, um eine klare, einheitliche Struktur zu erzwingen. Die erste formale Sprache wurde unter dem Namen „Begriffsschrift“ in [Frege 1879] veröffentlicht und ermöglichte erstmals die Formalisierung der klassischen Prädikatenlogik, mit der mathematische sowie Konstrukte der natürlichen Sprache abgebildet werden können.
Im Bereich der Softwareentwicklung bezeichnet eine Spezifikation die Beschreibung eines Softwaresystems oder -moduls bzgl. seiner Funktionalität oder seines Aufbaus. Bei wissensbasierten Systemen geht es darum, das Wissen in einer geeigneten Weise zu beschreiben, d.h. eine Wissensspezifikation zu definieren. Eine Spezifikation kann informell oder formal beschrieben sein. Für die allgemeine Softwa-

reentwicklung gibt es seit etwa den 1970er Jahren formale Spezifikationssprachen und seit den 1980er Jahren auch für wissensbasierte Systeme (Fensel 1994).

3.1.1 Vorteile formaler Spezifikationen

Eine formale Spezifikation hat gegenüber einer informellen Spezifikation den Vorteil, dass sie Informationen explizit mit einer eindeutigen Semantik formuliert. Je umfangreicher die Informationen sind, umso wichtiger ist eine strukturierte, formale Beschreibung. Erst diese ermöglicht eine automatische Verarbeitung und macht sie wartbar und wiederverwendbar (vgl. Fensel 1994). Außerdem muss es möglich sein eine Spezifikation auf Vollständigkeit und Widerspruchsfreiheit zu prüfen, was automatisiert nur für eine formale Spezifikation realisierbar ist (Marwedel 2007). Darüber hinaus zwingt eine informelle Spezifikation "mit ihrer prinzipiellen Unvollständigkeit [...] den Programmierer geradezu, Lücken durch seine Phantasien zu überbrücken. Und nicht immer trifft er dabei die Erwartungen der späteren Benutzer bzw. das fehlende Expertenwissen" (Fensel 1994, S. 8).

3.1.2 Nachteile formaler Spezifikationen

Die Formalisierung hat allerdings auch Nachteile. In [Fensel 1994] wird die Sinnhaftigkeit des Einsatzes von formalen Spezifikationssprachen zur Wissensbeschreibung für wissensbasierte Systeme diskutiert. U.a. haben, basierend auf [Fensel 1994], formale Spezifikationssprachen die folgenden Nachteile:

- **Sprachflut**: Da es nicht möglich ist, eine Sprache zu entwickeln, die alle Probleme löst, werden immer wieder neue Sprachen entwickelt. Gute Sprachen sind stets anwendungsspezifisch und nicht für den universellen Einsatz geeignet. Bereits 1994, im Veröffentlichungsjahr von [Fensel 1994], konnte diese Entwicklung auch im Bereich der wissensbasierten Systeme festgestellt werden.

- **Verständlichkeit**: Die Semantik formaler Sprachen ist nur auf den ersten Blick exakt und in sich geschlossen definiert. Im Normalfall muss die Semantik einer formalen Sprache mittels einer zweiten als bekannt vorausgesetzten formalen Sprache definiert werden. Die Verständlichkeit ist je nach Anwendungsfall auch unterschiedlich zu definieren, je nachdem ob die Spezifikation

maschinen- oder menschenlesbar sein soll.

- **Hohe Komplexität und komplizierte Erstellung**: Formale Sprachen basieren auf einer definierten Grammatik und Syntax. Je komplexer die zu beschreibenden Informationen sind, umso komplexer muss auch die Sprache sein. Mit steigender Komplexität verkompliziert sich auch die Erstellung der Sprache.

- **Hoher Aufwand und Kosten**: Bestehende Prozesse müssen angepasst werden, um ab sofort mit einer formalen Spezifikation zu arbeiten. Die Entwicklung von unterstützenden Werkzeugen ist erforderlich sowie die Schulung der beteiligten Personen.

Für jeden Anwendungsfall muss daher entschieden werden, ob die Verwendung einer formalen Spezifikation tatsächlich zu einem großen Nutzen führt. Bei umfangreichen Systemen ist das in der Regel der Fall. Der Aufwand und die Kosten zur Erstellung und Inbetriebnahme einer Spezifikation sind einmalig, wohingegen der Einsatz einer formalen Spezifikation eine früher Fehlervermeidung und -erkennung unterstützt und damit zu einer Kostenreduktion bei der Softwareentwicklung führt (vgl. Abschnitt 2.3). Auch die bessere Wart- und Wiederverwendbarkeit rechtfertigt den Aufwand eine formale Spezifikation einzuführen. Durch die einmalige Entwicklung von Werkzeugen, die die Arbeit mit der Spezifikation erleichtern, wird zudem das Verständlichkeitsproblem gelöst. [Memmel, Vanderdonckt und Reiterer 2008] diskutiert sehr ausführlich die Bedeutung einer bedachten Entwicklung von HMI. Die Verwendung von formalen Spezifikationen spielt dabei eine entscheidende Rolle.
Auch die Wahl der Spezifikationssprache wirkt sich entscheidend auf die oben genannten Aufwände aus sowie auch auf die Benutzerakzeptanz und Praktikabilität. Im Folgenden werden die formale Auszeichnungssprache *XML* und das formale Datenaustauschformat *JSON* genauer betrachtet und ihre Vor- und Nachteile gegenübergestellt.

3.2 Extensible Markup Language (XML)

XML wurde 1998 vom W3C veröffentlicht. Wie der Name bereits aussagt, ist XML eine *erweiterbare Auszeichnungssprache* und bietet damit vielseitige Möglichkei-

ten für die Beschreibung von Dokumenteninhalten (Steyer 2010). XML ist eine sogenannte *Metasprache*, die die Grundlage für die Definition weiterer, konkreterer formaler Sprachen liefert. Das auf XML basierende XHTML wird bspw. zur Beschreibung von Webseitenlayout verwendet. In Abschnitt 3.4 wird deutlich werden, dass auch sehr viele formale GUI-Spezifikationssprachen auf XML basieren.

3.2.1 XML-Syntax

Ein XML-Dokument beginnt stets mit dem Prolog, der die Version der verwendeten XML-Syntax und optional die Kodierung definiert:

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

XML hat eine auf Tags basierende Syntax. Daten werden in Form von Elementen mit einem Anfangs- und einem End-Tag definiert:

```
<Element>Daten</Element>
```

Elemente können Attribute und auch weitere Elemente beinhalten:

```
<Element attribut1="Attribut" attribut2=10>
  <KindElement attribut3=false>KindDaten</KindElement>
</Element>
```

Auf diese Weise können die Daten hierarchisch in einer Baumstruktur angeordnet werden; ein XML-Dokument muss dabei immer genau einen Wurzelknoten besitzen. Für die Attributwerte unterstützt XML eine Vielzahl von einfachen Datentypen.

3.2.2 XML-Schema

Mittels einer Schemasprache können eigene einfache und komplexe Datentypen sowie weitere Konventionen für ein gültiges XML-Dokument definiert werden. Die *XML Schema Defintion* (XSD) ist eine Schemasprache, die selbst auf XML basiert. Ein XSD-Dokument definiert Elemente und die hierarchische Struktur des XML-Dokuments. Des Weiteren können in einem XSD-Dokument Einschränkungen festgelegt werden, z.B. für ein integer-Element ein Wertebereich oder die Eindeutigkeit für Elementnamen. Die Definition von neuen einfachen und komplexen Datentypen und Namensräumen ist ebenfalls möglich. Neben XSD gibt es auch noch andere Schemasprachen, wie bspw. die *Dokumenttypdefinition* (DTD). Mit Verwendung eines Schemas besitzt ein XML-Dokument bereits eine interne Validierungsprüfung.

3.2.3 Einsatz und Nachteile

Die Metasprache XML ist sehr bekannt und bildet zusammen mit einer Schemasprache häufig die Basis bei der Entwicklung einer neuen Spezifikationssprache. Sehr viele Programmiersprachen bieten APIs an, um XML-Dateien zu lesen und zu erstellen. Für die Entwicklung von XML-Dokumenten existieren eine Vielzahl von Editoren. Die Editoren unterstützen dabei oft auch die Einbindung von Schemadateien für die direkte Validierung.
Trotz des weitverbreiteten Einsatzes weist XML als Spezifikationssprache auch einige Nachteile auf: Die Tag-Syntax ist sehr komplex und führt bereits bei der Beschreibung weniger Daten zu einem hohen Speicherverbrauch. Wegen der umfangreichen Syntax erfordert XML zudem komplexe Parser. Viele Programmiersprachen bieten zwar APIs an, diese sind aber meist recht aufwendig implementiert, da XML mit der Tag-Syntax normalerweise nicht zum Datenmodell der jeweiligen Programmiersprache passt. Ebenfalls aufgrund der komplexen Struktur kann der Zugriff auf einzelne XML-Elemente zeitintensiv sein.
D.h. XML ist tatsächlich relativ ineffizient bzgl. des Speicherverbrauchs, des Programmieraufwands und der Rechenzeit. XML ist eine sinnvolle Unterstützung für die Beschreibung kompletter Dokumente und die Erstellung eigener formaler Sprachen, die wiederum komplexe Sachverhalte definieren. Für die bloße Übertragung von Daten ist der Einsatz von XML als Spezifikationssprache allerdings oft überdimensioniert.

3.3 JavaScript Object Notation (JSON)

Aufgrund der Nachteile von XML wurde in den vergangenen Jahren JSON (*JSON.org - Introducing JSON*) für die Beschreibung von Daten entwickelt. Bei JSON handelt es sich um eine Untermenge der Skriptsprache *JavaScript*. JSON beschränkt sich auf die Verwendung von Objekten und Arrays, um Daten hierarchisch zu strukturieren. Im Gegensatz zu XML ist JSON ein bloßes Datenaustauschformat und keine Metasprache. Es bietet keine weiterführenden Konzepte wie Typerweiterungen oder Namensräume.

3.3.1 JSON-Syntax

Arrays und Objekte sind in JSON als Zeichenketten beschrieben. Ein Array wird mit eckigen Klammern definiert, z.B.:

```
[WertA, WertB, WertC]
```

Ein Objekt wird mit geschweiften Klammern definiert und besitzt Schlüssel-Wert-Paare als Attribute, z.B.:

```
{"Schlüssel1":WertD,"Schlüssel2":WertE}
```

Die Werte sind entweder von einem einfachen Datentypen, ein weiteres Objekt oder ein Array. Ein Wert vom Typ `string` steht, wie der Schlüssel, stets in Hochkommata. Ein Array kann nur Objekte desselben Typs beinhalten. Damit ist JSON im Wesentlichen beschrieben, weiterführende Beispiele finden sich in [Steyer 2010].

3.3.2 JSON-Schema

Seit 2010 gibt es auch für JSON eine Schemasprache (Zyp 2010), durch die eine interne Validierung gewährleistet werden kann. Das Schema ist ebenfalls in JSON formuliert und verfügt über eine Reihe von Schlüsselwörtern, um Objekte und Arrays mit spezifischen Eigenschaften zu definieren. Neben der Festlegung von Objekt- und Arraynamen, können Einschränkungen angegeben werden. Der folgende Beispielcode zeigt exemplarisch die Grundstruktur eines JSON-Schemas:

```
1  {
     "description": "An example",
     "type":"object",
     "properties": {
       "EinfachesElement": {
6        "type": "string"
         },
       "EinArray": {
         "type": "array",
         "items": {
11         "type": "number"
         }
       }
     }
   }
```

Mit dem Schlüsselwort `type` kann für jedes Element der Datentyp festgelegt werden. Es stehen die komplexen Typen `object` und `array` sowie einfache Typen wie `integer`, `string` und `boolean` zur Verfügung. Die Schlüssel-Wert-Paare eines Elements vom Typ `object` werden innerhalb einer `properties`-Deklaration angegeben. Der Datentyp für ein Array wird in `items` deklariert. Bei einem Element

einfachen Typs erfolgt lediglich die Typangabe.
Für jedes Element kann zusätzlich mit dem Schlüsselwort `description` eine Beschreibung angelegt werden. Eine Beschreibung ist stets vom Typ `string`.
Der Wertebereich eines Elements eines einfachen Typs kann auf verschiedene Arten eingeschränkt werden. U.a. kann mit den Schlüsselwörtern *minimum* und `maximum` für numerische Werte ein Minimal- und ein Maximalwert definiert werden und mit dem Schlüsselwort `enum` ist die Definition von Enumerationen möglich. Mit `"uniqueItems": true` kann die Eindeutigkeit von Elementen innerhalb eines Arrays gefordert werden. Die Identität von Elementen eines Arrays ist in [Zyp 2010] wie folgt definiert:

- Elemente eines einfachen Datentyps sind identisch, wenn sie denselben Wert besitzen.
- Objekte sind identisch, wenn sie denselben Namen besitzen und jede weitere Eigenschaft ebenfalls identisch ist.
- Arrays sind identisch, wenn sie dieselbe Anzahl an Elementen beinhalten und die Elemente ansich identisch sind.

Die Definition der Eindeutigkeit für Elemente, die sich nicht innerhalb eines Arrays befinden, ist nicht möglich.
Mit dem Schlüsselwort `minItems` kann festgelegt werden, wie viele Elemente ein Array mindestens enthalten muss. Des Weiteren können Objekte und Arrays mit `"optional": true` als optional gekennzeichnet werden. Wird keine Optionalität angegeben, ist das definierte Elemente obligatorisch und muss in jeder JSON-Datei enthalten sein, die das Schema verwendet.
Die Referenzierung auf Elemente desselben oder eines anderen Schemas wird mit dem Schlüsselwort `$ref` realisiert. Im folgenden Beispiel wird die Definition des Elements `RefElement` aus der Schemadatei *referenz.json* verwendet, die sich im selben Verzeichnis wie die aktuelle Schemadatei befindet:

```
{
  "type": "object",
  "properties": {
    "RefElement" : {
      "$ref" : "referenz.json"
    }
  }
}
```

Die Referenz kann auch eine absolute Pfadangabe sein und bspw. auf eine Webseite verweisen. Mit dem Präfix # wird auf Elemente innerhalb einer Schemadatei referenziert:

```
{
  "type": "object",
  "properties": {
    "Element1": {
      "type": "string"
    },
    "Element2" : {
      "$ref" : "#/properties/Element1"
    }
  }
}
```

Die Gesamtheit aller Schlüsselwörter ist in [Zyp 2010] angegeben.
Bei dem in [Zyp 2010] vorgestellten JSON-Schema handelt es sich allerdings lediglich um einen Entwurf in dritter Version, der laut eigenen Angaben nur bis zum 26. Mai 2011 Gültigkeit besaß. Aktuell gibt es damit keine offizielle Dokumentation der JSON-Schemasprache.

3.3.3 Einsatz

In JSON können Daten in Form von Arrays und Objekten auf einfache, kompakte Weise strukturiert, serialisiert und übertragen werden. Es ist für den Benutzer einfach zu lesen und schreiben und ebenso einfach maschinell zu verarbeiten. Dadurch ist die Verwendung von JSON weniger fehleranfällig als komplexere Spezifikationssprachen (Steyer 2010, Kap. 3.10). Da JSON ein Teil von JavaScript ist, sind Informationen über JSON in der Fachliteratur meist im Zusammenhang mit JavaScript zu finden, bspw. in [Steyer 2010]. JSON stellt gültiges JavaScript dar und kann so direkt in JavaScript-Programme eingebunden werden. Für eine Vielzahl weiterer Programmiersprachen existieren APIs (s. Auflistung in [*JSON.org - Introducing JSON*]). Da Arrays und Objekte in vielen Programmiersprachen als Konstrukte vorkommen, ist die Erstellung von APIs einfach. Aufgrund der schlanken Struktur von JSON sind auch die APIs schlank. Daneben gibt es kostenlose Editoren (s. *JSON.org - Introducing JSON*), die die JSON-Struktur übersichtlich als Baum darstellen und Fehler in der Syntax erkennen.
Schon seit einigen Jahren wird JSON erfolgreich in der Webentwicklung von Google und Yahoo eingesetzt (Steyer 2010, Kap. 3.10) und findet aufgrund der vielen Vorteile immer mehr Zuspruch. Obgleich es keine offizielle Dokumentation über die JSON-Schemasprache gibt, kommt diese zum Einsatz und wird in vielen Fo-

ren diskutiert (bspw. in der [*JSON Schema Google Group*]). Bei Verwendung eines JSON-Schemas steht für die Validierung der *JSON Schema Validator* [Court 2010] zur Verfügung.

3.3.4 JSON vs. XML

Da JSON ein einfaches Datenaustauschformat und XML eine erweiterbare Metasprache ist, sind die beiden Konzepte an sich grundlegend verschieden. Indem Arrays in Arrays geschachtelt werden, kann, analog zu XML, mit JSON auch eine hierarchische Datenstruktur erzeugt werden. In dieser Hinsicht besitzt JSON dieselbe Aussagekraft wie XML und eine Konvertierung zwischen diesen beiden Sprachen ist möglich. Für die Konvertierung werden auch Parser angeboten, z.B. [Briganti 2012]. Die Definition von Schemadateien ist auch für beide Sprachen möglich. Während für JSON nur eine Schemasprache existiert, stehen für XML allerdings mehrere Schemasprachen zur Wahl.

3.3.4.1 Vorteile von JSON gegenüber XML

JSON weist gegenüber XML einige entscheidende Vorteile für den Austausch von Daten auf:

- **Geeignetes Datenmodell**: Die Einbindung von JSON ist für viele Programmiersprachen einfach, da diese bereits Objekte und Arrays unterstützen.

- **Lesbarkeit**: Die hierarchische Strukturierung der Daten ist für den Menschen intuitiv verständlich. Die kompakte Struktur ermöglicht auch eine effiziente Maschinenlesbarkeit.

- **Stark reduzierte Syntax**: JSON beinhaltet im Wesentlichen nur die eigentlichen Daten und keine Tags. Gerade bei der Beschreibung sehr vieler Daten ist die kompakte Struktur von JSON gegenüber der komplexen XML-Struktur ressourcensparender. Allein durch die zwingenden Anfang- und End-Tags ist eine XML-Datei größer, als eine JSON-Datei desselben Inhalts (Steyer 2010, Kap. 3.10.3).

3.3.4.2 Nachteile von JSON gegenüber XML

Basierend auf der Metasprache XML können neue, umfangreiche Spezifikationssprachen entworfen werden. In einer XML-Schemadatei können für jede Spezifikationssprache neue Datentypen und Namensräume definiert werden. Daneben bieten die bestehenden Konstrukte zahlreiche Möglichkeiten Konventionen festzulegen. JSON und das JSON-Schema bieten nicht so umfangreiche Möglichkeiten an. Bspw. ist es in JSON im Gegensatz zu XML nicht möglich die Namenseindeutigkeit für Elemente innerhalb einer Datei zu fordern.

3.4 Formale Spezifikation von GUIs

3.4.1 GUI-Spezifikationssprachen

Im Folgenden werden ein paar bedeutende GUI-Spezifikationssprachen aufgeführt. Mit Ausnahme von CAP sind alle Sprachen XML-basiert. Ein Grund dafür kann sein, dass XML eine sehr umfangreiche Metasprache ist auf deren Basis neue Sprachen für verschiedenste Anwendungsbereiche definiert werden können. Mit einer XML-Schemasprache wird direkt eine interne Validierungsprüfung mitgeliefert. Als Standard ist es zudem ausführlich dokumentiert und wird von zahlreichen Programmiersprachen unterstützt.
Neben den vorgestellten existieren noch zahlreiche weitere Spezifikationssprachen, die aber keine nennenswerten größeren Umfang als die hier vorgestellten aufweisen. Einen umfangreichen Überblick bieten u.a. auch [Luyten 2004] und [Reich 2008].

3.4.1.1 User Interface Markup Language (UIML)

Die *UIML* (dt. Auszeichnungssprache für Benutzerschnittstellen) (s. Abrams u. a. 1999; Helms u. a. 2008) ist eine geräteunabhängige Spezifikationssprache für Benutzerschnittstellen.

Syntax UIML basiert auf XML. Ein UIML-Dokument mit den wichtigsten Komponenten ist folgendermaßen aufgebaut (s. Helms u. a. 2008):

```
<?xml version="1.0"?>
<!DOCTYPE uiml PUBLIC "-//OASIS//DTD UIML 4.0 Draft//EN"
   http://uiml.org/dtds/UIML4_0a.dtd">
<uiml xmlns='http://uiml.org/dtds/UIML4_0a.dtd'>
  <interface>
    <structure> ... </structure>
    <style> ... </style>
    <behavior> ... </behavior>
    <layout> ... </layout>
  </interface>
  <peers> ... </peers>
</uiml>
```

Nach dem Prolog (s. Abschnitt 3.2.1) beginnt das UIML-Dokument mit dem Tag <uiml>. In einem interface-Element wird eine Benutzerschnittstelle, z.B. eine GUI oder eine Spracheingabe, beschrieben. Es besteht aus den vier Teilen structure, style, behavior und layout. Im structure-Element werden die einzelnen Widgets und ihre hierarchische Anordnung definiert. Jedem Element wird dabei ein eindeutiger Name und eine Klasse als Datentyp zugewiesen, die für die folgende Implementierung als Widget vorliegen muss. Unter style wird die grafische Darstellung der Widgets und ihr Inhalt beschrieben. Dabei wird auf die zuvor definierten Widgets per Namen referenziert. Im Abschnitt, behavior, wird den Widgets Verhaltenslogik zugewiesen. Das Layout kann zusätzlich zu style auch mit umfangreicheren Regeln in einem separaten Abschnitt layout formuliert werden. Unter peers wird die Abbildung zwischen den im UIML-Dokument verwendeten Klassen und Namen zur Applikationslogik definiert.

Einsatz Der Zweck von UIML ist die Spezifikation der Funktionalität ohne Kenntnisse über die spätere konkrete Darstellung. Die meisten Elemente sind optional und können gegebenenfalls auch mehrfach definiert werden, z.B. um verschiedene Benutzerschnittstellen und Stile zu beschreiben. Die modulare Struktur von UIML gewährt eine klare Trennung zwischen den einzelnen Inhalten. UIML wird vor allem für die Spezifikation von Applikationen für mobile Geräte eingesetzt, da sich hier je nach Hersteller die Darstellungsfähigkeit stark unterscheiden kann.
Aus UIML können automatisch Benutzerschnittstellen generiert werden. Dafür kann UIML compiliert oder in eine Programmiersprache transferiert werden. UIML-Dokumente stellen auch einen sicheren Transfer für Internetanwendungen dar, da sie keinen ausführbaren Code beinhalten.
Die aktuelle Version 4.0 wurde 2008 veröffentlicht und ist in [Helms u. a. 2008]

ausführlich dokumentiert.

3.4.1.2 User Interface Extensible Markup Language (UsiXML)

Auch *UsiXML* (*UsiXML.org* 2009-2011) ist eine Spezifikationssprache zur geräteunabhängigen Beschreibung von Benutzerschnittstellen und basiert auf XML.

Syntax UsiXML bietet die folgenden Modelle für die Beschreibung einzelner Aspekte einer Anwendung an: *Domain Model*, *Task Model*, *Abstract User Interface Model*, *Concrete User Interface Model*, *Context Model*, *Mapping Model* und *Transformation Model*. Es ist ein Schichtenmodell von abstrakter zur konkreter Beschreibung zu erkennen. Alle Modelle können unabhängig voneinander verwendet werden, je nachdem wie detailliert eine Anwendung beschrieben werden soll. Das Schichtenmodell von UsiXML ist deutlich umfangreicher als UIML. U.a. unterstützt UsiXML die Defintion von Einschränkungen. Mit dem im UsiXML-Schema definierten komplexen Typ `mutualEmphasis` (s. *UsiXML-Schema, MutualEmphasis* 2010) kann bspw. ein gegenseitiger Ausschluss formuliert werden):

```
<ipo:mutualEmphasis
id=" xs:ID [0..1]"
name=" xs:string [0..1]">
<ipo:source> ... </ipo:source> [0..*]
<ipo:target> ... </ipo:target> [0..*]
</ipo:mutualEmphasis>
```

Des Weiteren können Designinformationen wie Positionen und Textfarben konkret angegeben werden.

Einsatz Die in [Luyten 2004] getroffene Aussage, dass UsiXML bislang die am besten entwickelte GUI-Beschreibungssprache ist, trifft auch bei aktueller Betrachtung der existierenden Sprachen noch immer zu. Im Vergleich zu UIML liefert UsiXML ein komplettes Framework für den Engineering und Reengineering-Prozess (Luyten 2004) und ist vom W3C standardisiert. Die aktuellste Version ist 1.8.0 aus dem Jahr 2007 (*UsiXML.org* 2009-2011).
Für die Entwicklung und die Visualisierung von UsiXML-Dokumenten existieren zahlreiche Softwareprogramme; *GrafiXML* ist bspw. ein grafischer Editor für die Gestaltung verschiedener Benutzerschnittstellen. Die Softwareprogramme schränken die Möglichkeiten in der Nutzung des Formats allerdings oft auf Desktop-Systeme ein, in denen viele Informationen, u.a. das Design, nicht konkret angegeben werden.

3.4.1.3 Extensible Application Markup Language (XAML)

XAML (dt. Erweiterbare Auszeichnungssprache für Anwendungen) wird seit 2008 von Microsoft für die Spezifikation von Windows-Systemen unter der .Net-Umgebung entwickelt und ist in [Microsoft 2012] dokumentiert.

Syntax Auch XAML basiert auf XML. XAML-Dokumente können auf einfache Weise in HTML eingebunden und so auf einer Webseite angezeigt werden. Die folgende XAML-Spezifikation führt zu der in Abb. 3.1 dargestellten GUI innerhalb einer HTML-Webseite.

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Page.Background>
    <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
      <GradientStop Color="#FF80FFFF" Offset="0"/>
      <GradientStop Color="#FF042CFF" Offset="1"/>
    </LinearGradientBrush>
  </Page.Background>
  <Canvas>
    <Button Canvas.Left="10" Canvas.Top="20" Width="200" Height="20">
      Hello World Button</Button>
    <TextBox Canvas.Left="10" Canvas.Top="50" Width="200" Height="100">
      Hello World!</TextBox>
  </Canvas>
</Page>
```

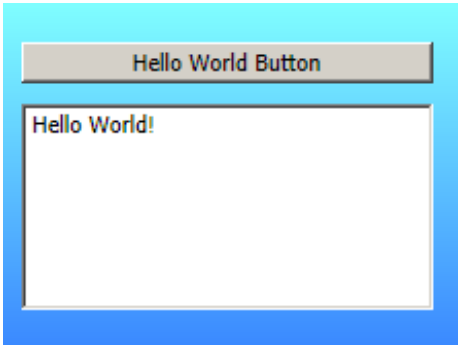


Abbildung 3.1: Mit XAML spezifizierte GUI.

Der allgemeine Aufbau eines XAML-Dokuments besteht aus einem Wurzelelement, im Beispiel `Page`, das einen Container für weitere Elemente darstellt. Neben `Page` können auch verschiedene *Panels*, z.B. `Canvas` und `GridPanel`, einen Container bilden. Jeder Containertyp legt dabei auch ein Layout für die zugewiesenen Elemente fest. Innerhalb eines Containers können Steuerelemente wie z.B. `Button` und `ListItem` definiert werden.

Es ist möglich eigene Namensräume und Elementklassen zu definieren, z.B. innerhalb eines C#-Projekts. Zur Einbindung wird im Prolog auf die Namensräume verwiesen.

Einsatz Im Gegensatz zu UIML und UsiXML hat XAML lediglich das Ziel, Windows-Systeme zu spezifizieren. Dadurch ist die Beschreibung in XAML wesentlich konkreter als in UIML oder UsiXML.
XAML wird von der Microsoft .Net-Umgebung unterstützt; die in XAML spezifizierte Benutzerschnittstellen können deshalb z.B. in C++ und C# implementiert werden. XAML ist offen zugänglich, aufgrund der starken Verknüpfung mit .Net wird es aktuell aber kaum von anderen Programmiersprachen eingebunden. Die Möglichkeit XAML in HTML einzubinden, macht einen Internet-Browser zu einem komfortablen Viewer. Da XAML eine Microsoft-Entwicklung ist, wird XAML neben dem Internet Explorer allerdings nur von wenigen weiteren Browsern unterstützt. Damit bietet XAML für den breiten Einsatz deutlich weniger Möglichkeiten an als UIML und usiXML.

3.4.1.4 Original Equipment Manufacturer XML (OEM XML)

Der Zweck der bisher vorgestellten Spezifikationssprachen ist eine möglichst universelle Beschreibung von Anwendungen. Im Gegensatz dazu wurde *OEM XML* (Brunhorn 2007) speziell für die Spezifikation von Infotainmentsystemen entwickelt. OEM XML entstand als Kooperationsprojekt der Automobilhersteller AUDI, BMW, Daimler, Porsche und VW.

Syntax OEM XML liegen die folgenden Schemata zugrunde (Brunhorn 2007):

- **Common.xsd**: Allgemeine Schemadefinitionen
- **Structure.xsd**: Struktur eines Widgets
- **Behaviour.xsd**: Verhaltenslogik eines Widgets
- **Resources.xsd**: Pfaddefinitionen zu Ressourcen (z.B. Fonts und Bilder)
- **Texts.xsd**: Definition von Texten
- **API.xsd**: Schnittstellenbeschreibung

Für jedes Widget kann sowohl eine *Structure*- als auch eine *Behaviour*-XML-Datei erstellt werden, die die Struktur bzw. das Verhalten beschreibt. Die Struktur umfasst die Positionierung und das grafische Aussehen des Widgets. Das Verhalten wird in Form von Zustandsdiagrammen in XML-Notation formuliert. Über Referenzierungen anhand des Namens werden die Widgets hierarchisch geordnet. So entsteht eine umfangreiche Baumstruktur, die ein gesamtes Infotainmentsystem beschreiben kann. Für eine bessere Übersicht können die einzelnen Dateien in Verzeichnissen gegliedert werden, z.B. sortiert nach MMI-Kontexten. Die *Structure*- und die *Behaviour*-XML-Datei eines Widgets müssen sich allerdings stets im selben Verzeichnis befinden.
Das Beispiel zeigt die Definition des Hinter- und des Vordergrunds eines Buttons mit dem Zustand *Down* innerhalb einer Behaviour-Datei.

```
<States>
 <State Name="Down">
  <Element Name="Background">
   <Widget Name="BaseImage">
    <WidgetStructure>
     <BaseWidget Name="Image">
      <BaseWidgetProperties>
       <Property Name="ImageProperties">
        <Property Name="FilePath">
         <Property Name="File">
          <Value Value="DefaultButton_DOWN.png"/>
         </Property>
         <Property Name="Path">
          <Value Value="IMAGES_PATH"/>
         </Property>
        </Property>
        </Property>
       </BaseWidgetProperties>
      </BaseWidget>
     </WidgetStructure>
    </Widget>
   </Element>
   <Element Name="Foreground">
    <Widget Name="BaseTextField">
     <WidgetStructure>
      <BaseWidget Name="TextField">
       <BaseWidgetProperties>
        <Property Name="TextFieldProperties">
         <Property Name="TextContent">
          <Property Name="Text">
           <Value Value="Button_DOWN"/>
          </Property>
         </Property>
        </Property>
       </BaseWidgetProperties>
      </BaseWidget>
     </WidgetStructure>
    </Widget>
   </Element>
  </State>
</States>
```

Einsatz Mit OEM XML kann ein Infotainmentsystem in Form seiner Widget-Hierarchie und Zustandsdiagrammen umfangreich formal spezifiziert werden. Die hohe Anzahl verschiedener XML-Dokumente und die zahlreichen Referenzierungen erschweren allerdings den Umgang mit OEM XML. An gewünschte Informationen zu gelangen bedeutet oftmals einen hohen Rechenaufwand, da verschiedene Dateien gelesen werden müssen. Darüber hinaus ist OEM XML aufgrund der kryptischen IDs und Querverweise schlecht menschenlesbar, was einen umfangreichen Editor zwingend erforderlich macht. Über den tatsächlichen produktiven Nutzen und die Existenz unterstützender Werkzeuge kann keine Aussage gemacht werden (vgl. Reich 2008).

3.4.1.5 AbstraktHMI

Mit der Sprache *AbstraktHMI* (Reich 2008) wurde in Kooperation mit Daimler der Versuch unternommen, aus bestehenden Sprachen, insbesondere OEM XML, eine Spezifikationssprache für Infotainmentsysteme zu schaffen, die einfacher zu handhaben ist.

Syntax Widgets werden mit einer eindeutigen ID und einem Klassentyp definiert. Bei der Definition wird bereits auch die hierarchische Gliederung der Widgets festgelegt (Reich 2008, Kap. 4.4.1):

```
<hmi:widget id="Functions" class="Container">
 <hmi:widget id="whiteBar" class="Image"/>
 <hmi:widget id="ButtonNavi" class="ImageButton"/>
</hmi:widget>
```

Für jedes Widget, auf das mit dem Schlüsselwort `element-name` und der ID referenziert wird, können strukturelle Eigenschaften definiert werden (Reich 2008, Kap. 4.4.1):

```
<hmi:style>
 <hmi:properties element-name="System.AppLine.Functions.MainScreen">
  <hmi:property name="enabled" public="false">true</hmi:property>
  <hmi:property name=" System.Image.height">316</hmi:property>
  <hmi:property name="visible" public="false">true</hmi:property>
  <hmi:property name="y" public="false">111</hmi:property>
 </hmi:properties>
</hmi:style>
```

Einsatz AbstraktHMI unterstützt die Definition mehrerer Baureihen und allgemeiner Aktionen, die für alle Zustandsdiagramme gelten. Damit kann besser als in OEM XML auf die Variantenvielfalt von Infotainmentsystemen eingegangen und

Redundanzen vermieden werden. Auch die Vielfalt an Dokumenten wurde eingeschränkt, indem normalerweise die Struktur und das Verhalten aller Widgets innerhalb jeweils eines XML-Dokuments beschrieben wird.
In [Reich 2008] wurden Überlegungen unternommen, die entwickelte Sprache in den automobilen HMI-Entwicklungsprozess bei Daimler mit unterstützenden Werkzeugen zu integrieren, was das Konzept abrundet. Inwieweit AbstraktHMI allerdings tatsächlich Einzug in die Entwicklung bei den einzelnen Automobilherstellern erhalten hat, ist nicht bekannt.

3.4.1.6 Canonical Abstract Prototypes (CAP)

CAP wurde in [Constantine u. a. 2000] vorgestellt und in [Constantine 2003] weiter vertieft. Das Ziel von CAP ist eine leicht verständliche formale Spezifikation von Benutzerschnittstellen. Vor allem soll CAP für die Erstellung von Prototypen in den frühen Entwicklungsphasen eingesetzt werden.

Syntax Im Gegensatz zu den bisher vorgestellten GUI-Spezifikationssprachen basiert CAP nicht auf XML. Es handelt sich um eine eigene Notation, die sich aus abstrakten Komponenten, den *Canonical Abstract Components* zusammensetzt. Eine abstrakte Komponente kann ein GUI-Element (z.B. Button, Textfeld), aber auch eine Aktion (z.B. Kopieren, Löschen) sein. Mit jeder Komponente ist auch eine bestimmte Funktionalität verbunden. Das Design und Layout der mit abstrakten Komponenten definierten GUI-Elemente kann ebenfalls spezifiziert werden. Die CAP-Notation beinhaltet zudem eine grafische Darstellung aller abstrakten Komponenten in Form von einfachen schwarz-weiß Symbolen, mit der sich die entworfenen Benutzerschnittstellen skizzieren lassen.

Einsatz CAP wurde entwickelt, um die Erstellung von Prototypen formal zu gestalten. Die Definition der Benutzerschnittstellen geschieht abstrakt. Erst bei der Implementierung werden die abstrakten Modelle der Verhaltenslogik und des Designs auf konkrete Werte abgebildet: „[...] canonical abstract prototypes provide designers with a standardized vocabulary for expressing visual and interaction designs independent of details of appearance and behavior. “ (Constantine 2003, S. 1).
Das Tool *CanonSketch* (Campos und Nunes 2004) bietet umfangreiche Möglichkeiten, GUIs in CAP zu spezifizieren. Neben CAP wird die *Wisdom*-Erweiterung von

UML (Nunes und Cunha 2001) eingesetzt, um auch komplexe Verhaltenslogik formal und verständlich abbilden zu können. Damit ermöglicht CanonSketch schließlich eine modellbasierte Benutzerschittstellenentwicklung. CanonSketch wurde mit dem Ziel entwickelt, umfangreichere Möglichkeiten bei der GUI-Entwicklung zu geben. U.a. können mit CanonSketch Webseiten in HTML aus der Spezifikation erstellt werden. Frühere Tools sind hingegen laut [Campos und Nunes 2004] zu eingeschränkt und kompliziert.

3.4.2 GUI-Spezifikationssprachen und GUI-Design

Für die formale Spezifikation von GUIs existieren sehr viele Spezifikationssprachen. Die meisten Sprachen wurden zur Beschreibung von Webseiten oder Desktop-GUIs entwickelt. Dort ist das Ziel, die jeweilige GUI vor allem in ihrer Funktionalität und Verhaltenslogik zu spezifizieren. Für das Design ergibt sich stattdessen durch den Anwendungsbereich meist, dass exakte Angaben nicht erwünscht sind. Bei Webseiten ist das letztendlich dargestellte Layout bspw. abhängig von der gewählten Fenstergröße des Internet-Browsers sowie möglicher weiterer Einstellung wie. z.B. die Schriftgröße. Auch bei der Programmierung von Desktop-GUIs werden oft anstelle absoluter Positionierungen nur allgemeingültige Layoutanordnungen für die einzelnen Elemente angegeben, z.B. *BorderLayout* oder *GridLayout* in Java. Das Design soll erst durch eine konkrete Implementierung definiert werden. D.h. derartige GUI-Spezifikationen werden in einem Entwicklungsstand erstellt, wenn das tatsächliche Design noch unbekannt ist und/oder um eine klare Trennung zwischen der Spezifikation und der Implementierung zu erreichen. Deshalb weisen viele der existierenden Sprachen auch keine Möglichkeit für die Spezifikation von GUI-Design auf. Dies wird auch in [Luyten 2004, Kap. 3.4] bemängelt: „Most XML-based languages have a firm focus on reusability and scalability: making one design for multiple devices is the main goal. This is something that has been quite successful for form-based interfaces, but more multi-modal or even graphically challenging interfaces that are usable are not supported very well yet.“
Z.B. kann im MMI-Design ein Elementmerkmal abhängig vom Zustand des jeweiligen Elements sein. In usiXML können jedoch nur für Elemente vom Typ *Button* Zustände definiert werden. Von diesem Zustand abhängige andere Merkmale lassen sich gar nicht modellieren, da die Definition des konkreten Aussehens erst durch die Implementierung erfolgen soll und damit nicht Aufgabe von usiXML ist.

Ähnlich verhält es sich mit UIML. XAML bietet konkretere Beschreibungsmöglichkeiten. Die Beschränkung liegt hier aber darin, dass XAML auf die Spezifikation von Windows-Systemen ausgelegt ist. Zudem wird XAML nur von den .Net-Programmiersprachen offiziell unterstützt.
Obgleich OEM XML für die Spezifikation von Infotainmentsystemen entwickelt wurde, liefert auch diese Sprache keine ausreichenden Möglichkeiten für die Definition von Design. Bei der Entwicklung von AbstraktHMI wurden einige Nachteile von OEM XML vermieden, mit dem Ziel eine übersichtlichere, auch menschenlesbare Spezifikation zu erhalten. Allerdings liegt auch bei AbstraktHMI der Schwerpunkt auf der Beschreibung von Verhaltenslogik. Mit keiner Spezifikationssprache kann vollständig das konkrete GUI-Design definiert werden, da dies stets erst durch die konkrete Implementierung geschehen soll.

3.4.2.1 Erweiterbarkeit

Insbesondere die XML-basierten Spezifikationssprachen sind erweiterbar. Damit ist es theoretisch möglich, jede bestehende GUI-Spezifikationssprache um konkrete Designinformationen zu erweitern. Das bedeutet aber eine aufwendige Weiterentwicklung von ohnehin schon recht komplexen Formaten, von denen ein großer Teil gar keine Verwendung bei der Spezifikation von GUI-Design findet. Eine Weiterentwicklung erfordert auch umfangreiche Kenntnisse über die bestehende Spezifikationssprache. Zudem sind bei einer Spracherweiterung normalerweise auch die bestehenden APIs für die Integration in eine Programmiersprache zu erweitern. Der Zusatzaufwand ist damit enorm und da von vornherein nicht sichergestellt werden kann, ob das erweiterte Format dann tatsächlich effizient und vollständig GUI-Design beschreiben kann, keinesfalls gerechtfertigt. Sicher bestünde auch die Möglichkeit, ein bestehendes Format zu erweitern und gleichzeitig zu reduzieren, sodass letztendlich nur noch eine Designbeschreibung enthalten ist. Auch das setzt aber ein hohes Verständnis für das Format voraus.

3.4.2.2 Wahl der Basissprache

Auch ist zu überlegen, welche Sprache sich am besten für die Spezifikation von GUI-Design eignet. Da die Beschreibung von GUI-Design relativ simpel im Vergleich zur Beschreibung von vollständigen Benutzerschnittstellen und komplexer Verhaltenslogik ist, bietet sich anstatt XML die Verwendung einer kompakteren,

ressourcensparenderen Sprache, wie bspw. JSON, an. Daneben hat die Verwendung von XML im Bereich GUI-Spezifikationssprachen durchaus auch ihre Berechtigung, denn für die Spezifikation einer gesamten GUI einschließlich Verhaltenslogik und Systemzuständen wäre JSON sicher zu schlicht.

3.4.3 Spezifikation von GUI-Design

3.4.3.1 Photoshop und Plugins

Bei GUI-Designern ist die Software *Photoshop* von Adobe ein weitverbreitetes Werkzeug, das sehr viele Möglichkeiten für die Gestaltung von GUI-Design bietet. Als Bildbearbeitungs-Software unterstützt Photoshop direkt nicht die Generierung einer formalen Spezifikation. Über eine Plugin-Schnittstelle können allerdings weitere Funktionalitäten integriert und Skripte zur Automatisierung bestimmter Aufgaben ausgeführt werden. *PhotoProto* von Altia ist ein Photopshop-Plugin, das es ermöglicht eine in Photoshop entworfene GUI mit Verhaltenslogik zu annotieren (s. auch Reich 2008). Es können auch Video- und Audiodateien eingebunden werden. Der Designentwurf wird formal in einer XML-Datei gespeichert. Die XML-basierte Spezifikationssprache ist eine Eigenentwicklung von Altia. Sie basiert auf SVG (Scalable Vector Graphics (dt. skalierbare Vektorgrafik)) (s. W3C 2011). Die erstellten XML-Dokumente sind offen zugänglich, eine Dokumentation zu dieser Spezifikationssprache existiert allerdings nicht. Das zusätzliche Tool *Altia DeepScreen* ermöglicht den Export von Quellcode aus mit PhotoProto erstellten PSD-Dateien, um einen lauffähigen Prototypen zu erhalten. Für komplexere, in Zustandsdiagrammen formulierte, Verhaltenslogik, bietet das Tool *Altia Connections* eine Schnittstelle zu der Software *Telelogic Rhapsody*. Die Tools von Altia bieten damit umfangreiche Möglichkeiten einen Designentwurf kombiniert mit Verhaltenslogik zu erstellen und diesen in einem maschinenlesbaren Format zu speichern. Das von Altia selbst genannte Ziel ist allerdings die Erstellung funktionsfähiger Prototypen. Ein gravierender Nachteil von PhotoProto ist, dass es sich nicht erweitern lässt. Bspw. ist für Buttons fest definiert, dass sie zwei, drei, vier oder sechs Zustände annehmen können, die optisch verschieden dargestellt werden. Das schränkt die Gestaltungsfreiheiten entscheidend ein. Die Verwendung von PhotoProto ist damit nicht für jede GUI möglich. Des Weiteren ist eine konsistente Modellierung über mehrere Tools hinweg schwierig zu erreichen und fehleranfällig.

3.4.3.2 INSPECTOR

[Memmel und Reiterer 2008] stellt mit *INSPECTOR* ein weiteres Tool für die Entwicklung von GUIs vor, das eine formale Spezifikation liefert. Die letzte Veröffentlichung zu INSPECTOR erschien 2010 in [Geyer, Memmel und Reiterer 2010]. Der Ansatz von INSPECTOR unterscheidet sich stark von dem von PhotoProto. Die Spezifikationsgrundlage bildet eine reduzierte und veränderte Form von UML-Diagrammen. UML an sich wurde als zu komplex und unverständlich für verschiedene Personengruppen angesehen (vgl. Geyer, Memmel und Reiterer 2010, Kap. 3). Mit Use-Case- und Aktivitäts-Diagrammen liegt der Schwerpunkt hier in der Modellierung der Verhaltenslogik. Ein weiterer wesentlicher Aspekt ist die für alle Personengruppen geeignete Visualisierung.
Für die Spezifikation des GUI-Designs bietet INSPECTOR eine breite Palette von einfachen Zeichnungen bis hin zu komplexen Widgets und der Einbindung von Bildern und weiteren Medien an. Für die Visualisierung des GUI-Designs unterstützt INSPECTOR den Export in das XAML-Format (s. Abschnitt 3.4.1.3), wodurch die Darstellung in einem Internet-Browser ermöglicht wird (Geyer, Memmel und Reiterer 2010).
Inspector ist ein umfangreiches Tool, das mit dem Ziel entwickelt wurde, die Erstellung von Prototypen bereits in den frühen Phasen des Designentwurfs zu unterstützen und die Spezifikation für alle Personengruppen geeignet zu visualisieren. Es können abstrakte und konkrete GUIs modelliert werden, wobei die Rückverfolgbarkeit und Transparenz zwischen den Modellen stets gewährleistet wird. Damit bildet INSPECTOR eine sehr gute Grundlage, bereits sehr früh mit einer formalen Spezifikation zu beginnen.
Eine komplexe Design-Software wie Photoshop kann und will INSPECTOR nicht ersetzen. Je nach Komplexität der zu entwickelten GUI ist daher anschließend noch der Einsatz eines leistungsstärkeren Produkts wie Photoshop notwendig. Der Tool-Wechsel stellt dann allerdings wieder einen Bruch in der Rückverfolgbarkeit der Spezifikation dar. Ein Tool, das den gesamten Entwicklungsprozess abdeckt, ist meist nicht realisierbar und würde die kreative Arbeit deutlich einschränken.

3.4.4 Integration in den GUI-Entwicklungsprozess

Bezüglich einer formalen Beschreibung einer GUI ist eine sehr wichtige Frage die nach der Realisierbarkeit. Bereits in [Fensel 1994] wurde kritisiert, dass im Automobilbereich den Benutzerschnittstellen noch immer zu wenig Aufmerksamkeit und Zeit im Entwicklungsprozess zugesprochen wird. Die Erstellung von formalen Spezifikationen muss als ein wichtiger Prozess angesehen werden, um die Qualität der Software zu verbessern (vgl. Fensel 1994, S. 9).
Der Prozess der GUI-Entwicklung weist allerdings Schritte auf, bei denen es nicht möglich ist zu einer formalisierten Beschreibung zu gelangen. Insbesondere beim Designentwurf sind nicht Techniker oder Informatiker, sondern Designer und Psychologen beteiligt, die eine kreative, informelle Arbeitsweise haben. Der Bruch zwischen informeller und formaler Spezifikation im Entwicklungsprozess ist nach wie vor aktuell, wie die informelle Spezifikation des MMI-Designs zeigt (s. Abschnitt 2.2.2.3). Um bereits in frühen Entwicklungsphasen eine maschinenlesbare Spezifikation zu erhalten, müssen geeignete Tools vorhanden sein, die den Umgang mit formalen Sprachen unterstützen und auch für Nicht-Experten gangbar ist. In [Geyer, Memmel und Reiterer 2010], ebenfalls bezugnehmend auf den Automobilbereich, wird sehr detailliert auf die Problematik eingegangen, für verschiedene Prozessschritte und Personengruppen Tools bereitzustellen, die eine einheitliche, formale Spezifikation ermöglichen und für jeden einfach zu bedienen sind. Die Rückverfolgbarkeit und Transparenz von Anforderungen und Entscheidungen spielt in diesem Zusammenhang auch eine große Rolle (vgl. Geyer, Memmel und Reiterer 2010, Kap. 1.1). In der Realität verwendet meist jede Personengruppe das Tool, mit dem sie vertraut ist und eine bereichsübergreifende, einheitliche Spezifikation kann nicht realisiert werden (vgl. Geyer, Memmel und Reiterer 2010, Kap. 1.1). Das Fehlen dieser Voraussetzungen bzw. des Willens seit Jahren etablierte Entwicklungsprozesse zu ändern ist ein Grund, weswegen viele Spezifikationssprachen nie umfangreich produktiv eingesetzt werden.

Kapitel 4

Wissensverarbeitung

Nach der Vorstellung des Aufbaus und der Arbeitsweise von wissensbasierten Systemen in Abschnitt 4.1, werden in Abschnitt 4.2 verschiedene Verfahren für die Verarbeitung von unsicherem Wissen gegenübergestellt.

4.1 Wissensbasierte Systeme

Hard- und Software werden in einem immer rasanteren Tempo entwickelt. Seit in etwa den 1960er Jahren werden sie nicht mehr ausschließlich für die einfache Datenverarbeitung verwendet, sondern sind in der Lage aus einer umfangreichen Datenbasis Wissen zu verarbeiten und dadurch zu neuen Informationen zu gelangen. Diese so genannten *wissensbasierten Systeme* kommen mittlerweile in sehr vielen verschiedenen Anwendungsbereichen zum Einsatz. Im Gegensatz zu übrigen Softwaresystemen werden bei wissensbasierten Systemen Lösungen nicht durch Algorithmen berechnet, sondern auf Basis von Wissen mittels intelligenter Verfahren generiert (Beierle und Kern-Isberner 2006, Kap. 2.4).

4.1.1 Aufbau

Trotz der Vielfältigkeit ihrer Einsatzgebiete besteht ein wissensbasiertes System im Wesentlichen aus den drei Komponenten *Wissensbasis*, *Inferenzkomponente* und *Erklärungskomponente*:

- **Wissensbasis**: In der Wissensbasis ist das Wissen in einer für die jeweilige Anwendung geeigneten Form formuliert. Es ist *deklarativ* als Faktenwissen

aber auch mit weiteren Formalismen wie *Wenn-Dann-Regeln* definierbar. In vielen Anwendungsbereichen reicht die *klassische Logik* oftmals nicht aus um das Wissen exakt zu beschreiben (Beierle und Kern-Isberner 2006, Kap. 2). In diesen Fällen ist die Entwicklung einer für den Anwendungsfall passenden komplexeren Wissensbeschreibung erforderlich.

- **Inferenzkomponente**: In der Inferenzkomponente wird die Verarbeitung des Wissens vorgenommen. Durch Inferenz des gegebenen Wissens werden neue Schlussfolgerungen gebildet. Um die Verarbeitung zu vereinfachen, sollte die Art der Wissensrepräsentation auf die Inferenzberechnungen abgestimmt sein. Durch Eingabe des Wissens wird die Inferenzkomponente konfiguriert und kann anschließend mittels verschiedenster Verfahren neue Informationen bzw. Lösungen generieren. Die Inferenz kann durch die klassische Logik erfolgen, aber auch durch komplexere Formalismen wie bspw. *Markovketten* oder *Bayes'sche Netze*. Die Wahl des Inferenzverfahrens wird maßgeblich durch die Art des zu verarbeitenden Wissens beeinflusst (Beierle und Kern-Isberner 2006, Kapitel 2.6.1). In Kapitel 4.2 wird genauer auf den Umgang mit unsicherem Wissen eingegangen und in Kapitel 5 werden Bayes'sche Netze als mögliche Inferenzkomponente detailliert vorgestellt.

- **Erklärungskomponente**: Die Erklärungskomponente gibt die Ergebnisse der Wissensverarbeitung aus und kann ausführliche Erklärungen zur Lösungsfindung liefern (Beierle und Kern-Isberner 2006, Kapitel 2.6.1).

Die Komponenten stellen einzelne Module des Systems dar, die miteinander interagieren, aber unabhängig voneinander austauschbar sein sollten. Gegeben einer geeigneten Schnittstelle sollte z.B. für verschiedene Wissensbasen dieselbe Inferenzkomponente verwendet werden können. Problembeschreibung und Problemlösung sind damit klar voneinander getrennt und das Wissen liegt zentral vor (Beierle und Kern-Isberner 2006, Kap. 2.2). Die Möglichkeiten der konkreten Umsetzung sind wiederum sehr vielfältig und müssen je nach Anwendungsbereich geeignet ausgewählt werden.
Neben den drei zentralen Komponenten eines wissensbasierten Systems, gibt es des Weiteren den Wissenserwerb, der ebenfalls als eigenständiges Modul angesehen werden kann. Durch die *Wissenserwerbskomponente* wird die Wissensbasis aufgebaut und unterstützt die geeignete Bereitstellung des Wissens (Beierle und Kern-

Isberner 2006, Kap. 2.6.1). Zwischen den Komponenten sind darüber hinaus Kommunikationsschnittstellen definiert, die einen automatischen Transfer der Ergebnisse ermöglichen.

4.1.2 Kategorisierung

[Beierle und Kern-Isberner 2006] behandelt das Thema wissensbasierte Systeme sehr detailliert. Es werden der allgemeine Aufbau beschrieben sowie die Unterschiede zwischen *regelbasierten Systemen*, *Expertensystemen* und *Agentensystemen*, die jeweils eine Spezialisierung von wissensbasierten Systemen darstellen (siehe unten), diskutiert. Die gängigsten Verfahren für den Wissenserwerb, die Wissensrepräsentation und die Inferenz werden vorgestellt. Dabei werden neben der klassischen Logik und dem regelbasierten Ansatz auch komplexere Verfahren zum Umgang mit unsicherem Wissen ausführlich erläutert.
Bei einer Unterscheidung nach der Art der Wissensrepräsentation ist das regelbasierte System zu nennen. Das Wissen ist hier in Form von Wenn-Dann-Regeln formuliert. Unter Anwendung der klassischen Logik lassen sich daraus auf einfache Weise Schlussfolgerungen ableiten. Regelbasierte Systeme gehören zu den ältesten wissensbasierten Systemen. Sie werden auch heutzutage in Bereichen eingesetzt, in denen klar strukturierte Schlussfolgerungen mittels der klassischen Logik ausreichen.
Eine andere Kategorisierung kann nach der Wissensherkunft vorgenommen werden. Bei Expertensystemen handelt es sich um menschliches Expertenwissen. Dieses besitzt besondere Eigenschaften und Schwierigkeiten, bspw. liegt Expertenwissen oft nur implizit vor, lässt sich dadurch schwer fassen und nicht immer vollständig in Formalismen beschreiben. Expertensysteme wurden erstmals im Bereich der Medizin für die Diagnostik entwickelt. Einen guten Überblick hierzu gibt [Spreckelsen und Spitzer 2008].
Ein Agentensystem ist ein Softwareprogramm, das in einem bestimmten Anwendungsgebiet *autonomes Verhalten* zeigt. Die Wissensverarbeitung eines Agentensystems weist damit starke Bezüge zu Verfahren der *Künstlichen Intelligenz* auf und ist folglich meist komplexer als andere wissensbasierte Systeme. Roboter, die selbständig ohne Benutzerinteraktion bestimmte Aufgaben erledigen und auf ihre Umwelt reagieren, sind ein Beispiel für Agentensysteme. Die Kategorisierung schließt sich nicht gegenseitig aus. Ein wissensbasiertes System kann gleichzeitig

ein regelbasiertes System, Expertensystem und Agentensystem sein, wenn die oben genannten Eigenschaften zutreffen. Im Rahmen dieser Arbeit wird der allgemeine Begriff wissensbasiertes System verwendet.

4.2 Schlussfolgern mit unsicherem Wissen

Die im vorherigen Abschnitt genannten Wissensrepräsentationen in Form klassischer Logik und Regeln gehören zu den *symbolischen Methoden*. Sie liefern klare Aussagen, z.B. ob ein Zustand wahr oder falsch ist. D.h. mit diesen Verfahren können sichere Informationen ausgewertet werden. Bei der Verifikation des GUI-Designs müssen Bildinformationen verarbeitet werden. Wie in Kap. 2.3.5.3 erläutert, handelt es sich dabei um unsicheres Wissen. Beim Umgang mit unsicherem Wissen stoßen die symbolischen Methoden an ihre Grenzen, da sie Unsicherheiten nicht abbilden können (Korb und Nicholson 2004, Kap. 1.2).

4.2.1 Arten von Unsicherheit

Tatsächlich spielt das Schlussfolgern mit Unsicherheit für wissensbasierte Systeme eine immer größere Rolle. Hauptsächlich im Bereich der Medizin beschäftigen sich Forschungsgruppen seit Jahren mit dieser Herausforderung. Hier sind Symptome Indizien für Krankheiten. Durch die Beobachtung von Krankheitsfällen lassen sich Statistiken über die Kausalität zwischen einer Krankheit und ihren Symptomen feststellen. Je nach Krankheit muss das Vorhandensein eines Symptoms aber nicht zwingend sein, d.h. das Auftreten ist nicht zu 100% sicher. Außerdem können Tests zur Symptomfeststellung in einem gewissen Maße falsch-positive oder falsch-negative Ergebnisse liefern. Unsicherheit kann somit zum einen bedeuten, dass der Einfluss eines Zustands keine 100% Auswirkung auf das Ergebnis hat und zum anderen, dass der beobachtete Zustand nur zu einer bestimmten Sicherheit wirklich existiert. Daneben gibt es noch eine dritte Form von Unsicherheit, die *Unschärfe*, bei der die Grenzen zwischen den Zuständen verschwimmen. Ein Beispiel aus dem medizinischen Bereich ist das Symptom *Fieber*, dass anhand der Körpertemperatur festgestellt wird. Strenggenommen kann keine harte Grenze angegeben werden, ab wann eine gemessene Temperatur Fieber bedeutet oder so gar konkreter, wann es sich um leichtes, mittleres oder hohes Fieber handelt. Derartige Zustände sind nicht

hart voneinander abgrenzbar und es müssen Übergangsbereiche definiert werden, in denen mehrere Zustände möglich sind. Bei der Entwicklung von Systemen, die unsichere Informationen verarbeiten sollen, muss daher auch darauf geachtet werden um welche Form der Unsicherheit es sich handelt.

4.2.2 Schlussfolgern unter Unsicherheit mit quantitativen Methoden

Für das Schlussfolgern mit Unsicherheit wurden die sogenannten *quantitativen Methoden* entwickelt, die auf heuristischen oder numerischen Verfahren basieren. Aussagen werden gewichtet, um ihre Sicherheit, die Stärke ihrer Einflussnahme oder ihre Zugehörigkeit zu beschreiben.

4.2.2.1 Gewissheitsfaktoren

In den 1970er Jahren wurde erstmals mit den sogenannten *Gewissheitsfaktoren* (engl. *Certainty Factors*) ein heuristisches Konzept entwickelt. Das sehr bekannte und in [Buchanan und Shortliffe 1984] ausführlich beschriebene medizinische Diagnosesystem MYCIN basiert auf Wenn-Dann-Regeln mit Gewissheitsfaktoren. Jeder Regel wird ein Gewissheitsfaktor $\gamma \in [-1, 1], \gamma \in \mathbb{R}$ zugewiesen, der die Stärke des Eintritts des Dann-Teils festlegt, wenn der Wenn-Teil zutrifft. Ein positiver Faktor unterstützt dabei die Hypothese, ein negativer Faktor lehnt sie ab. Daneben können auch die ins System eingegebenen Fakten mit Gewissheitsfaktoren justiert werden. Die Kombination der Regeln ist kommutativ, d.h. die einzelnen Informationen werden als voneinander unabhängig angesehen. Das ist in der Realität aber oftmals nicht der Fall. Es ist dem Anwender überlassen, welche Werte konkret verwendet werden. Die Gewissheitsfaktoren sind damit sehr interpretativ. Sie besitzen außerdem keine mathematisch fundierte Grundlage und unterliegen auch sonst keiner Konsistenzprüfung.

4.2.2.2 Probabilistische Graphenmodelle

Der 1985 in [Heckerman 1985] vorgenommene Versuch das Konzept der Gewissheitsfaktoren mit der Wahrscheinlichkeitstheorie zu begründen, zeigt deutlich, dass das nicht vollständig möglich ist. Aus diesen Gründen ist nach [Spreckelsen und

Spitzer 2008] mittlerweile bei medizinischen Diagnosesystemen sogar von der Verwendung der Gewissheitsfaktoren abzuraten und durch Formalismen abzulösen, die vollständig mit der Wahrscheinlichkeitstheorie begründbar sind. Die mathematische Korrektheit und Konsistenz der Inferenzberechnungen ist dann gewährleistet (vgl. Wiegerinck, Kappen und Burgers 2010, S. 2). Probabilistische graphische Modelle, in denen die Knoten Zufallsvariablen sind, stellen einen solchen Formalismus dar. Sicherheiten werden als Wahrscheinlichkeiten betrachtet, wobei sich die Wahrscheinlichkeiten der Zustände einer Zufallsvariablen stets zu 1 addieren und sich gegenseitig ausschließen. Die Zustandsmengen können diskret oder kontinuierlich sein. Mit *Markov Netzen* können ungerichtete Graphen und damit ungerichtete Beziehungen zwischen Zufallsvariablen formuliert werden. *Bayes'sche Netze* hingegen sind gerichtete azyklische Graphen und können auf diese Weise gerichtete Abhängigkeiten zwischen Zufallsvariablen abbilden, d.h. *konditionale Wahrscheinlichkeiten*. Bayes'sche Netze sowie ihre Beziehung zu anderen probabilistischen Graphenmodellen werden im folgenden Kap. 5 genauer erläutert.

4.2.2.3 Dempster-Shafer-Evidenztheorie

Durch die Wahrscheinlichkeitstheorie ist für probabilistische Logiken die Kombination von Informationen klar definiert. Jedoch können auch probabilistische Logiken nicht alle realen Probleme lösen. Wie auch in [Beierle und Kern-Isberner 2006, Kap. 13] ausführlich mit Beispielen erläutert, gibt es in der Wahrscheinlichkeitstheorie keine Unterscheidung zwischen *echter Unsicherheit* und *Unwissen*. Wahrscheinlichkeitswerte können aber genau eine dieser beiden Ursachen haben. Unwissenheit im wahrscheinlichkeitstheoretischen Ansatz kann durch eine Gleichverteilung dargestellt werden, z.B. *0.5 wahr* und *0.5 falsch*. Das hat aber dieselbe Bedeutung wie *gleichwahrscheinlich*. Die Wahrscheinlichkeitstheorie fordert zudem, dass sich Zustände einer Zufallsvariablen gegenseitig ausschließen und die Zustandsmenge vollständig ist. In realen Anwendungsbereichen kann das oft nicht gewährleistet werden. Bspw. ist in der Regel nicht jede mögliche medizinische Diagnose bekannt. Eine Lösung dieser Problematik gibt die 1976 veröffentlichte *Dempster-Shafer-Evidenztheorie*. Sie erweitert die probabilistische Logik um die sogenannten *Glaubens-* bzw. *Vertrauensfunktionen*, mit denen Unwissenheit explizit formuliert und so von der probabilistischen Unsicherheit getrennt betrachtet werden kann. In der Evidenztheorie haben Variablenzustände nicht die Bedeutung von Elementarer-

eignissen. Stattdessen wird die Potenzmenge der Zustandsmenge einer Variablen betrachtet und der sichere Glauben auf die Teilmengen verteilt. Zweifel kann dann durch das Komplement ausgedrückt werden. Die Differenz zwischen Glauben und Zweifel wird als *Plausibilität* bezeichnet und ist ein Maß für die Ungewissheit. Die Evidenztheorie definiert ebenfalls, wie Plausibilitäten kombiniert werden. Nur wenn der Glauben und die Plausibilität sich zu 100% summieren, kann der Glauben als Wahrscheinlichkeit betrachtet werden. Die Glaubensfunktion wird in diesem Fall auch als *Bayes'sche Glaubensfunktion* bezeichnet. Anwendung findet die Evidenztheorie vor allem in der Auswertung von Sensordaten (Beierle und Kern-Isberner 2006, Kap. 13.2).

4.2.2.4 Fuzzy-Logik

Zu Beginn des Abschnitts wurde bemerkt, dass es noch eine dritte Art der Unsicherheit gibt, die Unschärfe. In diesem Fall muss das sogenannte *approximative Schließen* mit unscharfen bzw. vagen Informationen angewendet werden. Bereits in der Antike formulierte der Philosoph Platon, dass zwischen den Zuständen wahr und falsch ein dritter Bereich liege. Jedoch erst 1965 veröffentlichte L. A. Zadeh in [Zadeh 1965] die Theorie der *Fuzzy-Mengen*. Mittels Fuzzy-Mengen lassen sich Übergänge zwischen Zuständen definieren und damit Unschärfe hinzufügen. Mit diesem Formalismus können z.B. die oben genannten Fieber-Zustände abgebildet werden. Ein Wert kann mehreren Fuzzy-Mengen zu einem bestimmten Grad $g \in [0, 1], g \in \mathbb{R}$ zugeteilt werden. $g = 1$ bedeutet eine sichere Zugehörigkeit und $g = 0$ einen sicheren Ausschluss. Auf dieselbe Weise werden auch Beziehungen (Fuzzy-Relationen) dargestellt. Die Kombination von Fuzzy-Mengen ist über das kartesische Produkt definiert. Die *Fuzzy-Logik* basiert auf diesem Ansatz (Spreckelsen und Spitzer 2008).

4.2.3 Gegenüberstellung und weiterführende Ansätze

Wie ersichtlich ist, gibt es mittlerweile zahlreiche Formalismen für den Umgang mit unsicherem Wissen. Mittels quantitativer Methoden lassen sich im Gegensatz zu den symbolischen Methoden Unsicherheiten gewichten, kombinieren und daraus neue Informationen ableiten. Die umfangreichen Möglichkeiten der quantitativen Methoden bringen aber auch Nachteile mit sich. Die probabilistische Logik

entspricht nicht der klassischen Logik (Beierle und Kern-Isberner 2006, Kap. 12, S. 365 und Beierle und Kern-Isberner 2006, Kap. 3.3.1, S. 24). Die klassische Logik definiert z.B., dass *A AND B* nur dann *wahr* ist, wenn *A* und *B* beide *wahr* sind. In der probabilistischen Logik muss das nicht zutreffen und *falsch AND wahr* kann auch *wahr* sein. Je nach Wahl der Gewichtungen sind bei der Kombination von Informationen sehr unterschiedliche Ergebnisse möglich. Dempster-Shafer und Fuzzy-Logik bieten noch mehr Möglichkeiten, aber auch erheblich mehr Fehlerquellen und eine höhere (Rechen-)Komplexität. Die Verwendung von Potenzmengen bei der Evidenztheorie führt ohne Optimierungsmaßnahmen zu einem exponentiellen Rechenaufwand. Die Fuzzy-Logik entfernt sich stark von dem mathematischen Fundament der Wahrscheinlichkeitstheorie und kann daher auch zu einem indeterministischen, inkonsistenten Verhalten führen. Laut Beierle und Kern-Isberner 2006 (Kap. 12, S. 365) ist der probabilistische Ansatz daher noch immer die beste Lösung. Mit der ebenfalls zunehmenden Rechenleistung moderner Computer lässt sich die Komplexität meist gut bewältigen. Daneben sind reale Probleme meist selbst sehr komplex. Glücklicherweise gibt es einige Vereinfachungen und approximative Verfahren, um die Rechenzeiten zu optimieren. In Abschnitt 5.2 werden ein paar dieser Optimierungsmöglichkeiten für Inferenzberechnungen in Bayes'schen Netzen aufgeführt.

Letztendlich hängt es in vielen Anwendungsbereichen von der Betrachtungsweise der zu verarbeitenden Information ab, welche Methode sich am besten eignet. Oft ist es auch Definitionssache, ob es sich um echte Unsicherheit, Unwissen oder um Unschärfe handelt. Die Einsatzgebiete von z.B. Bayes'schen Netzen und der Fuzzy-Logik sind sehr vielfältig und erstrecken sich u.a. für verschiedenste Klassifikationsaufgaben, Bildverarbeitung und die Diagnose technischer Systeme (Holmes und Jain 2008; Luo, Savakis und Singhal 2005; Korb und Nicholson 2004; Kruse, Gebhardt und Klawonn 1993). Das eigentliche Haupteinsatzgebiet von Fuzzy-Systemen ist allerdings die Regelungstechnik (Kruse, Gebhardt und Klawonn 1993). Im Gegensatz zu probabilistischen Graphenmodellen bieten Ansätze wie die Fuzzy-Logik jedoch wenig Transparenz bzgl. Wissensbeschreibung und Systemverhalten. Das wird in [Spreckelsen und Spitzer 2008, Kap. 8] für den Einsatz bei medizinischen Diagnosesystemen kritisiert. Nach [Kruse, Gebhardt und Klawonn 1993] erweisen sich kombinierte Ansätze (z.B. Fuzzy-Logik und Bayes'sche Netze) allerdings als sehr geeignet, um die Vorteile verschiedener Ansätze nutzen zu können. Für die

Realisierung des Designverifikationssystems wird ein Bayes'sches Netz verwendet. In Kapitel 6.3.1 wird diese Entscheidung insbesondere in Gegenüberstellung mit der Fuzzy-Logik begründet.

Das Schließen mit symbolischen und quantitativen Methoden ist normalerweise monoton, d.h. bei Erweiterung des Wissens wird angenommen, dass die bisherigen Erkenntnisse unverändert bestehen bleiben. In der Realität können oftmals nie alle möglichen Ereignisse von vornherein in einem wissensbasierten System vorgesehen werden, d.h. das gegebene Wissen ist oft unvollständig. Das hat zur Folge, dass durch neues Wissens bereits gezogene Schlussfolgerungen möglicherweise widerrufen werden müssen. Diese Art des Schließens wird als nicht-monoton bezeichnet (Beierle und Kern-Isberner 2006, Kap. 7). Für regelbasierte Systeme bedeutet die Beachtung von nicht-monotonen Schließen die Hinzunahme weiterer Regeln, die die Konsistenz wiederherstellen, wenn diese durch unvorhergesehene neue Informationen zerstört wird. Ein solches regelbasiertes System wird in der KI als *Truth Maintenance System* bezeichnet und ist z.B. in Beierle und Kern-Isberner 2006] ausführlich dargestellt. Auch für Bayes'sche Netze gibt es dahingehende Erweiterungen, die das *asymmetrische Schließen* realisieren (siehe Kap. 5.7).

Kapitel 5

Bayes'sche Netze

Das Designverifikationssystem besitzt als Inferenzkomponente ein Bayes'sches Netz. Nachdem in Kapitel 1 bereits ein Überblick über die Anwendungsbereiche von Bayes'schen Netzen gegeben wurde, liefert dieses Kapitel ein grundlegendes Verständnis über den Formalismus. In Abschnitt 5.1 wird der Bayes'sche Wahrscheinlichkeitsbegriff in Abgrenzung zum physikalischen Wahrscheinlichkeitsbegriff definiert und die Eigenschaften Bayes'scher Netze sowie das probabilistische Schließen in Bayes'schen Netzen werden anhand von Beispielen erläutert. Darauf aufbauend beschreibt Abschnitt 5.2 die Berechnung von konditionalen Wahrscheinlichkeiten mit exakten und mit approximativen Verfahren. Neben dem Umgang mit sicheren Evidenzen wird auch die Behandlung von unsicheren Evidenzen thematisiert. In Abschnitt 5.3 wird auf die automatische Generierung von Bayes'schen Netzen aus einer formalen Wissensbeschreibung und durch datenbasiertes Lernen eingegangen. Daneben werden praktische Anwendungen aus der Literatur vorgestellt. Da die Modellierung von Bayes'schen Netzen und die Eingabeinformationen fehlerbehaftet sein können, werden in Abschnitt 5.4 Methoden vorgestellt, um die Güte von Bayes'schen Netzen und Generierungsverfahren zu bestimmen und Konflikte bei der Eingabe von Informationen zu erkennen. Die Abschnitte 5.5 und 5.6 befassen sich dann mit der Effizienz und der Erklärbarkeit von Bayes'schen Netzen, die für den produktiven Einsatz eine entscheidende Rolle spielen. Das Kapitel schließt in Abschnitt 5.7 mit weiterführenden Formalismen und Erweiterungen. Es wird ein Überblick über andere probabilistische Graphenmodelle, insbesondere der Faktorgraphen gegeben. Als Ergänzung liefert Anhang D eine kurze Zusammenfassung zu verschiedenen kommerziellen und freiverfügbaren Softwareprogrammen

für die Arbeit mit Bayes'schen Netzen. Alle in diesem Kapitel verwendeten Formelbezeichungen sind in Anhang A definiert.

5.1 Terminologie

5.1.1 Definition des Wahrscheinlichkeitsbegriffs

Im Grundlagenteil wurde bereits erläutert wie mit quantitativen Methoden unsicheres Wissen verarbeitet wird. Die Unsicherheiten können dabei als Wahrscheinlichkeiten interpretiert werden. Eine Wahrscheinlichkeitsfunktion $P(x_i) = p_i \in [0, 1]$ ist eine Abbildung zwischen einem Zustand x_i einer Variablen X und einem Wahrscheinlichkeitswert p_i. Der Wahrscheinlichkeitswert $p_i = 1$ bezeichnet das sichere Eintreten des Zustands x_i und $p_i = 0$ das unmögliche Eintreten.
Zur Definition des Wahrscheinlichkeitsbegriffs gibt es den *physikalischen* und den *Bayes'schen Ansatz*. Der erstgenannte Ansatz definiert Wahrscheinlichkeiten als *Ergebnisse von physikalischen Zufallsexperimenten*. Die Wahrscheinlichkeitsverteilung über die Zustände einer Zufallsvariablen wird durch Wiederholen eines Experiments ermittelt, wie z.B. bei einem Münzwurf über die zwei Zustände *Kopf* und *Zahl*. Um Verteilungen zu erhalten, die korrekt die realen Verhältnisse abbilden, muss das Experiment sehr häufig durchgeführt werden. Es handelt sich dann um objektive, statistische Beobachtungen. In vielen realen Anwendungsbereichen ist die Durchführung von Experimenten jedoch nicht möglich. Z.B. kann zur Bestimmung der Eintrittswahrscheinlichkeit der Aussage „Morgen scheint die Sonne" kein Experiment durchgeführt werden. Die Festlegung einer Wahrscheinlichkeit muss hier subjektiv erfolgen und kann damit auch von verschiedenen Personen unterschiedlich gewählt werden. Bereits im 18. Jahrhundert entwickelte der Mathematiker Thomas Bayes den *Bayes'schen Wahrscheinlichkeitsbegriff* (Korb und Nicholson 2004, Kap. 1.4). Bayes'sche Wahrscheinlichkeiten werden basierend auf subjektiven Einschätzungen, z.B. von Experten, definiert. Damit erweitert der Bayes'sche Ansatz den physikalisch basierten Wahrscheinlichkeitsbegriff und ermöglicht die Festlegung von Wahrscheinlichkeiten in Bereichen, in denen keine physikalischen Experimente möglich sind. Die subjektiv ermittelten Wahrscheinlichkeiten können sich jedoch auch auf Beobachtungen begründen. Bspw. kann der Wahrheitsgehalt der obigen Aussage sinken, wenn die vergangenen Tage Regenwetter war. Daneben

können subjektive Aussagen mit physikalischen Experimenten in Beziehung gesetzt werden. Die subjektive Einschätzung obiger Aussage kann bspw. damit verglichen werden, eine weiße Kugel aus einer Urne mit 100 Kugeln, n weiße und $100 - n$ blaue, zu ziehen. Für ein bestimmtes n entspricht $\frac{n}{100}$ genau der subjektiven Wahrscheinlichkeit der Aussage (Jensen und Nielsen 2007, Kap. 1.1). In der praktischen Anwendung hat sich zudem gezeigt, dass die Verwendung von subjektiven Wahrscheinlichkeiten oftmals zu korrekten Ergebnissen führt (vgl. Anwendungsbeispiele in Korb und Nicholson 2004, Kap. 1.5.5).

5.1.2 Bayestheorem

Das ebenfalls von Thomas Bayes formulierte Bayestheorem bedient sich der Bayes'schen Wahrscheinlichkeitsdefinition und definiert, wie mit *konditionalen Wahrscheinlichkeiten* gerechnet werden kann. Die Berechnungen werden als Inferenzberechnungen bezeichnet. In [Wiegerinck, Kappen und Burgers 2010, S. 2] wird dazu geschrieben: "Probabilistic inference is the problem of computing the posterior probabilities of unobserved model variables given the observations of other model variables."(dt. „Die Aufgabe der probabilistischen Inferenz ist die Berechnung der posteriori Wahrscheinlichkeiten zu den unbeobachteten Modellvariablen gegeben beobachteter Modellvariablen.“) Beobachtete Variablen sind diejenigen, deren Zustand bekannt ist, d.h. eine normalerweise sichere Evidenz für einen Zustand vorliegt[1]. Eine Kombination der gesetzten Zustände von beobachteten Modellvariablen wird auch als *Evidenzmenge* $\mathbf{E}$ bezeichnet. In Formel 5.1 ist das Bayestheorem am Beispiel der Beziehung zwischen einer Hypothese H und einer beobachteten Evidenz E dargestellt. Die Wahrscheinlichkeit der Hypothese H bedingt durch Evidenz E ($P(H|E)$), kann durch die Wahrscheinlichkeit der Evidenz E bedingt durch H ($P(E|H)$) berechnet werden, wenn zudem die Basiswahrscheinlichkeiten $P(H)$ und $P(E)$ gegeben sind. Die konditionale Wahrscheinlichkeitsfunktion $P(E|H)$ wird auch als *Likelihood* und die Basiswahrscheinlichkeiten als *A-priori-Wahrscheinlichkeiten* bezeichnet. Das Ergebnis $P(H|E)$ ist die *Posteriori-Wahrscheinlichkeit* für H (Duda und Hart 1973; Jensen und Nielsen 2007).

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)} \tag{5.1}$$

[1]Evidenzen können auch unsicher sein, d.h. ihre Wahrscheinlichkeit ist kleiner 1. In Abschnitt 5.2.3 wird darauf eingegangen.

Das bedeutet $posteriori = \frac{likelihood \cdot prior}{evidence}$.

5.1.3 Eigenschaften Bayes'scher Netze

Ein Bayes'sches Netz $\mathcal{B} = (\mathbf{X}, \mathbf{E}, \mathbf{\Theta})$ ist ein komplexes statistisches Modell, in dem konditionale Abhängigkeiten zwischen Zufallsvariablen explizit abgebildet werden können. Es lässt sich sehr anschaulich als gerichteter, azyklischer Graph (DAG, engl. directed acyclic graph) darstellen bestehend aus einer Menge von Knoten $\mathbf{X}$ und Kanten $\mathbf{E}$. Die Knoten repräsentieren die Zufallsvariablen X_1 bis X_n, die eine beliebige Anzahl m verschiedener Zustände x_{ik} ($i \in [1, ..., n]$ und $k \in [1, ..., m]$) annehmen können; z.B. im Fall einer binären Variablen die Zustände *wahr* und *falsch*[2]. Die gerichteten Kanten repräsentieren die konditionalen Abhängigkeiten zwischen Variablen. Wenn eine gerichtete Kante von einem Knoten A zu einem Knoten B führt, wird A als *Elternknoten* von B bezeichnet und dementsprechend ist B ein *Kindknoten* von A. Für alle Variablen sind Wahrscheinlichkeitsverteilungen $\mathbf{\Theta}$ definiert. Die konditionale Abhängigkeitsbeziehung kann in Wahrscheinlichkeitstabellen angegeben werden, im Folgenden auch als *CPT* (engl. Conditional Probability Table, dt. Tabelle der konditionalen Wahrscheinlichkeiten bzw. Übergangswahrscheinlichkeiten) bezeichnet. Für Variablen, die keine Elternknoten besitzen, werden statt Übergangswahrscheinlichkeiten A-priori-Wahrscheinlichkeiten angegeben, d.h. eine Wahrscheinlichkeitsverteilung über die Variablenzustände. Zustandswahrscheinlichkeiten einer Variablen summieren sich stets zu 1. Daraus folgt, dass sich die Zustände einer Variablen gegenseitig stets ausschließen und die Zustandsmenge vollständig sein muss.

5.1.3.1 Unabhängigkeit und d-Separierung

In einem Bayes'schen Netz ist eine Variable X stets unabhängig von ihren Nicht-Kindknoten, wenn für alle ihre Elternknoten $Pa(X)$ Evidenzen gegeben sind. Dieses Verhalten ist Teil der *d-Separierungseigenschaften* (engl. d-separation bzw. direction-dependent separation, dt. richtungsabhängige Separierung) (Pearl 1988, Kap. 3.3) und gilt für serielle und divergierende Verbindungen in einem Bayes'schen Netz. In Abb. 5.1 ist mit dem obigen und dem mittigen Bayes'schen Netz dafür jeweils

[2] In dieser Arbeit werden die Begriffe *Knoten* und *Variable* zur Bezeichnung einer Zufallsvariablen eines Bayes'schen Netzes synonym verwendet.

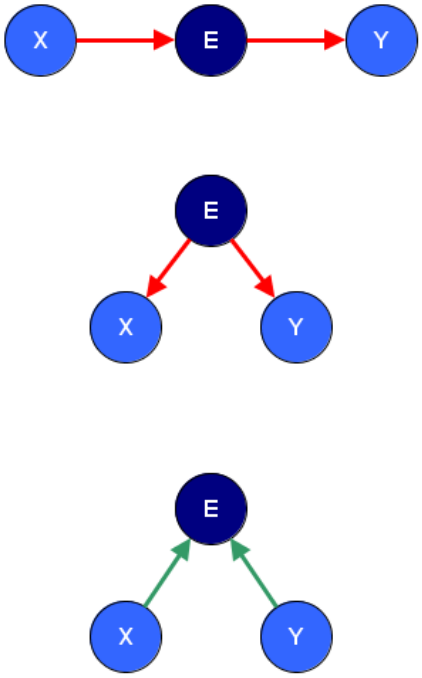


Abbildung 5.1: Darstellung der d-Separierungseigenschaften für eine serielle, eine divergierende und eine konvergierende Verbindung (vgl. Jensen und Nielsen 2007, Kap. 2.2). Für die Variable E ist jeweils Evidenz gesetzt. Die Variablen X und Y werden durch die gesetzte Evidenz entweder unabhängig voneinander (oben und mittig) oder abhängig (unten).

ein Beispiel dargestellt. Durch Eingabe einer Evidenz für Variable E wird die Kommunikation zwischen den Variablen X und Y blockiert. In einer konvergierenden Verbindung (s. unteres Netz in Abb. 5.1) wird stattdessen erst durch Eingabe von Evidenz die Kommunikation ermöglicht.
Für unabhängige Variablen A und B gilt $P(A|B) = P(A)$ und $P(B|A) = P(B)$. Daraus folgt, dass unabhängige Variablen keinerlei gegenseitige Informationen beinhalten. Die Wahrscheinlichkeitstheorie definiert zudem, dass Wahrscheinlichkeiten von unabhängigen Variablen multipliziert werden können.

5.1.3.2 Verteilung von Verbundwahrscheinlichkeiten

Ein Bayes'sches Netz $\mathbf{B}$ besteht aus den zwei Teilen Graphenstruktur $\mathcal{S} = (\mathbf{X}, \mathbf{E})$ und Wahrscheinlichkeiten bzw. Parametern $\boldsymbol{\Theta}$. Damit definiert ein Bayes'sches Netz die *Verteilung von Verbundwahrscheinlichkeiten* $\mathcal{P}$ (JPD, engl. Joint Probability

Distribution) über die Menge aller Variablen **X** (Jensen und Nielsen 2007, Kap. 2.3.2). Unter Verwendung der Kettenregel lässt sich die JPD als Produkt der Übergangswahrscheinlichkeiten darstellen, wie in Formel 5.2 angegeben.

$$\mathcal{P}(\mathbf{X}) = \prod_{i}^{n} P(X_i|Pa(X_i)) \tag{5.2}$$

Die JPD ist eine eindeutige, kompakte Beschreibung der in einem Bayes'schen Netz enthaltenen Information, indem sie die Wahrscheinlichkeiten der Zustandskombinationen aller Variablen **X** beschreibt. $P(X_i|Pa(X_i))$ bezeichnet die Wahrscheinlichkeitsverteilung der konditionalen Abhängigkeiten zwischen den Variablen X_i und ihren Elternknoten $Pa(X_i)$.
Für zwei Variablen A und B resultiert die JPD direkt aus der Rechenvorschrift 5.3, die in [Jensen und Nielsen 2007] als *Fundamental rule for variables* (dt. Grundregel für Variablen) bezeichnet wird.

$$\mathcal{P}(A, B) = P(A|B)P(B) \tag{5.3}$$

Aus der JPD lässt sich herleiten, dass Bayes'sche Netze unterschiedlicher Struktur $\mathcal{S}$ dieselbe JPD $\mathcal{P}$ repräsentieren können: Die Richtung der Kante zwischen zwei Variablen A und B ist irrelevant, wenn die A-priori-Wahrscheinlichkeiten $P(A)$ und $P(B)$ sowie die CPTs $P(A|B)$ und $P(B|A)$ zu derselben JPD $\mathcal{P}(A, B)$ führen. Die Kantenrichtung spielt allerdings für die Definition der konditionalen Wahrscheinlichkeiten und die Interpretierbarkeit des Netzes eine wichtige Rolle. Oft werden die Kanten von der Ursache in Richtung der Wirkung gezogen (Kausalität). Diese Richtung ist intuitiver und die Wahrscheinlichkeitsverteilungen können leichter angegeben werden. Eine JPD ist damit also nicht eindeutig genau einem Bayes'schen Netz zugeordnet. Bayes'sche Netze, die dieselben Variablen beinhalten und genau dieselbe JPD darstellen, werden als *Markov-äquivalent* bezeichnet (Korb und Nicholson 2004, Kap. 6.3.1). Markov-Äquivalenz ist gleich bedeutend damit, dass dieselben Unabhängigkeitsbedingungen gelten (Flesch und Lucas 2007). Markov-äquivalente Bayes'sche Netze bilden eine eindeutige Äquivalenzklasse und werden auch als *essentielle Graphen* (engl. essential graph) bezeichnet. Essentielle Graphen besitzen neben gewöhnlichen Kanten auch essentielle Kanten, d.h. die Richtung einer essentiellen Kante kann nicht umgekehrt werden ohne dabei die Ab- bzw. Unabhängigkeitsbeziehungen zu verändern. Das wiederum bedeutet, dass sich die Strukturen Bayes'scher Netze einer Äquivalenzklasse nur in ihren gewöhnlichen Kanten voneinander unterscheiden.

5.1.3.3 I-Map, D-Map und Perfect Map

Ein Bayes'sches Netz $\mathcal{B}$ mit der JPD $\mathcal{P}$ soll eine gegebene Wahrscheinlichkeitsverteilung Q möglichst exakt repräsentieren, d.h. im Idealfall $\mathcal{P} \equiv Q$. Die Graphenstruktur $\mathcal{S}$ besitzt dann die *Markov-Eigenschaft*, die für $\mathcal{B}$ besagt, dass alle direkten Abhängigkeitsbeziehungen aus Q auch mit Kanten explizit dargestellt sind (Korb und Nicholson 2004, Kap. 2.2.4).[3] Das Bayes'sche Netz ist dann eine *I-Map* (engl. Independency-Map) und es gilt: Wenn die Variablen A und B in $\mathcal{P}$ unabhängig sind, dann sind sie es auch in Q. Q kann noch weitere Unabhängigkeiten definieren, die nicht in $\mathcal{P}$ enthalten sein müssen. Deshalb ist ein vollständig verknüpftes Bayes'sches Netz eine I-Map, da $\mathcal{P}$ dann explizit gar keine Unabhängigkeiten definiert und dies somit auch nicht von Q fordert. Für die Abbildung von Unabhängigkeiten ist ein vollständiges Bayes'sches Netz jedoch sehr ineffizient, da dafür CPTs definiert werden müssen, die viele Nulleinträge enthalten und damit an sich redundant sind. $\mathcal{B}$ besitzt dann nicht die minimal notwendige Kantenanzahl und stellt keine kompakte Beschreibung von Q dar. Der Sinn eines Bayes'schen Netzes, mit einem möglichst schwach verknüpften Graphen effizient eine Wahrscheinlichkeitsverteilung abzubilden, ist verfehlt.
Die Markov-Eigenschaft allein genügt nicht, um $\mathcal{P} \equiv Q$ zu erreichen, da Q Unabhängigkeiten beinhalten kann, die nicht Teil von $\mathcal{P}$ sind. Erst wenn $\mathcal{B}$ auch eine *D-Map* (engl. Dependency-Map) ist, ist $\mathcal{P} \equiv Q$. Für eine D-Map gilt: Wenn die Variablen A und B in $\mathcal{P}$ abhängig sind, dann sind sie es auch in Q. Q kann noch weitere Abhängigkeiten beinhalten. Ein unverknüpftes Bayes'sches Netz ist eine D-Map, da dieses keine einzige Abhängigkeitsbeziehung repräsentiert.
Wenn $\mathcal{B}$ sowohl eine I-Map als auch eine D-Map ist, ist es eine *Perfect-Map* und es gilt $\mathcal{P} \equiv Q$. $\mathcal{B}$ repräsentiert Q dann auf optimale Weise und beinhaltet nur Kanten, die minimal notwendig sind. Für eine Perfect-Map ist die JPD $\mathcal{P}$ maximal reduziert (Bishop 2006, Kap. 8.3.4).

[3] Eine ausführliche Beschreibung der Markov-Eigenschaften für gerichtete und ungerichtete Graphen liefert [Flesch und Lucas 2007].

5.1.4 Bayes'sche Netze zur Klassifikation

5.1.4.1 Naiver Bayes-Klassifikator

Die einfachste Form eines Bayes'schen Netzes ist in Abb. 5.2 zu sehen und wird als *naiver Bayes-Klassifikator* bezeichnet (erstmals vorgestellt in [Duda und Hart 1973]). Ein naiver Bayes-Klassifikator besitzt genau einen Elternknoten, auch Wurzelknoten genannt. Alle übrigen Knoten sind direkte Kinder des Elternknotens. Untereinander sind die Kindknoten nicht verknüpft und daher bei gegebener Evidenz für den Wurzelknoten unabhängig voneinander (s. divergierende Verbindung in Abschnitt 5.1.3.1). Der naive Bayes-Klassifikator bildet damit vor allem starke Unabhängigkeitsbeziehungen ab.

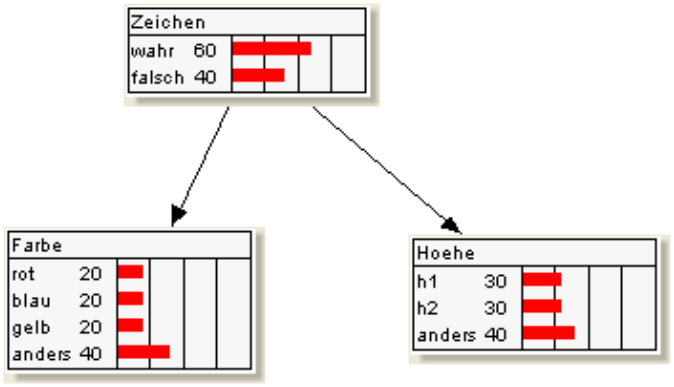


Abbildung 5.2: Naiver Bayes-Klassifikator zur Klassifikation von einfarbigen Regionen als Schriftzeichen basierend auf den Merkmalen *Farbe* und *Höhe*. (*Bayes-Beispiel 1*)

Mit dem in Abb. 5.2 dargestellten Bayes-Klassifikator, der im Folgenden als *Bayes-Beispiel 1* bezeichnet wird, soll auf Basis der Merkmale *Farbe* und *Höhe* entschieden werden, ob es sich bei einer einfarbigen Region in einem Bild um ein Schriftzeichen handelt oder nicht. Die zwei möglichen Klassen *Schriftzeichen* und *kein Schriftzeichen* werden im Wurzelknoten `Zeichen (Z)` mit den Zuständen *wahr* und *falsch* repräsentiert. Der Zustand dieser *Klassenvariablen* wird durch die Zustände der beobachtbaren *Merkmalsvariablen* `Farbe (F)` und `Hoehe (H)` beeinflusst. Die A-priori-Wahrscheinlichkeit für die Variable `Zeichen` sei mit *wahr*= 0.6 definiert. Die CPTs der Variablen `Farbe` und `Hoehe` beschreiben die Abhängigkeit der Merkmale von der Variable `Zeichen`. Im Beispiel ist mit den in den Tabel-

len 5.1 und 5.2 aufgeführten CPTs definiert, dass Schriftzeichen *rot*, *blau* oder *gelb* sind und entweder eine Höhe von *h1* oder *h2* haben. Für die Repräsentation aller andersartigen Regionen besitzen die Merkmalsvariablen den Zustand *anders (a)*, der die Zustandsmengen vervollständigt. In Abb. 5.2 sind für die Merkmalsvariablen bereits die Zustandswahrscheinlichkeiten in Abhängigkeit der A-priori-Verteilung der Variablen `Zeichen` dargestellt. Die Merkmalsvariablen selbst sind d-separiert bei gegebener Evidenz für die Variable `Zeichen`.

F|Z	wahr	falsch
rot	$\frac{1}{3}$	0.0
blau	$\frac{1}{3}$	0.0
gelb	$\frac{1}{3}$	0.0
a	0.0	1.0

Tabelle 5.1: CPT für das *Bayes-Beispiel 1* zur Definition der Abhängigkeit zwischen den Variablen `Farbe` und `Zeichen`.

H|Z	wahr	falsch
h1	0.5	0.0
h2	0.5	0.0
a	0.0	1.0

Tabelle 5.2: CPT für das *Bayes-Beispiel 1* zur Definition der Abhängigkeit zwischen den Variablen `Hoehe` und `Zeichen`.

5.1.4.2 Erweiterter Bayes-Klassifikator

Unabhängigkeiten führen unter Anwendung des Bayes-Theorems zwar zu sehr kompakten, effizienten Berechnungen, kommen jedoch in realen Problemstellungen oftmals nicht vor. Auch im Fall der Schriftzeichenklassifikation führt es zu besseren Ergebnissen, wenn zwischen den Merkmalen *Farbe* und *Höhe* eine konditionale Abhängigkeit definiert wird. Bspw. kann die Wissensbasis für die Schriftzeichenklassifikation enthalten, dass Schriftzeichen mit Höhe *h1* nur die Farben *Rot* oder *Blau* haben können. Schriftzeichen der Höhe *h2* hingegen kommen nur in der Farbe

Gelb vor. Die Hinzunahme dieser Informationen führt zu dem in Abb. 5.3 dargestellten Bayes'schen Netz, im Folgenden als *Bayes-Beispiel 2* bezeichnet. Durch Umkehr der Kantenrichtungen kann die in Tabelle 5.3 abgebildete CPT definiert werden, die die gewünschte Abhängigkeitsbeziehung darstellt.

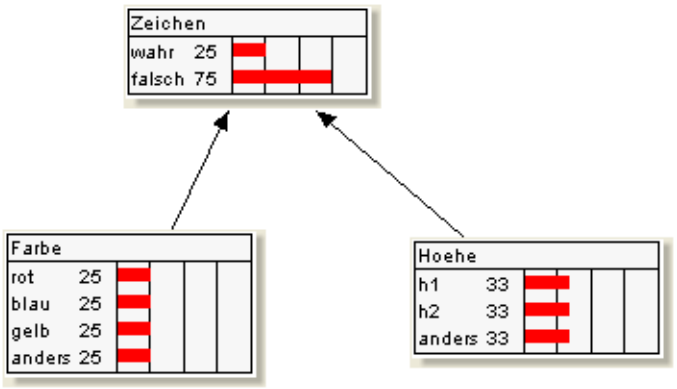


Abbildung 5.3: Bayes'sches Netz zur Klassifikation von einfarbigen Regionen als Schriftzeichen, basierend auf den Merkmalen *Farbe* und *Höhe* mit definierter Abhängigkeit zwischen den Merkmalsvariablen. (*Bayes-Beispiel 2*)

Z | F, H	**rot**			**blau**			**gelb**			**a**		
	h1	**h2**	**a**	**h1**	**h2**	**a**	**h1**	**h2**	**a**	**h1**	**h2**	**a**
wahr	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
falsch	0.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0

Tabelle 5.3: CPT für das *Bayes-Beispiel 2* zur Definition der Abhängigkeitsbeziehungen zwischen den Merkmalsvariablen.

In Abb. 5.4 sind zwei Beispiele für Merkmalskombinationen angegeben. Die Ergebnisse der Inferenzberechnungen spiegeln genau die Wissensbasis wider. Z.B. schließt die definierte CPT $P(Zeichen|Farbe, Hoehe)$ aus, dass blaue Regionen mit Höhe *h2* als Schriftzeichen klassifiziert werden. Mit einem naiven Bayes-Klassifikator können solche Informationen nicht repräsentiert werden, was ihren Anwendungsbereich entscheidend einschränkt. Dass zwischen den Merkmalsvariablen nun eine direkte Abhängigkeit existiert, lässt sich bei gegebenem Zustand für `Zeichen` leicht prüfen: Bei Änderung des Zustands der einen Merkmalsvariablen, verändert sich auch der Zustand der anderen Merkmalsvariablen. D.h. die Merkmalsvariablen

sind nun nicht mehr d-separiert, wenn eine Evidenz für die Variable `Zeichen` gegeben ist. In Abb. 5.5 ist, bei gegebenen Zustand *wahr* für die Variable `Zeichen`, die Abhängigkeit der Variablen `Farbe` bzgl. der Änderung des Zustands der Variablen `Hoehe` verdeutlicht. Für `Hoehe`=*anders* sind die Wahrscheinlichkeiten aller Zustände der Variablen `Farbe` 0. Das ist offensichtlich ein Fehler, da sich die Wahrscheinlichkeiten der Zustände einer Variablen stets zu 1 summieren sollen. Der Grund dafür ist, dass die Zustandskombination `Zeichen`=*wahr* und `Hoehe`=*anders* in der verwendeten CPT nicht definiert ist. D.h. im Beispiel gilt für diese Kombination die Annahme, dass sie niemals eintritt. Wird dennoch so eine Zustandskombination als Evidenz eingegeben, führt das zu inkonsistenten Ergebnissen. In Abschnitt 5.4.3 wird dieses Verhalten erläutert und aufgezeigt, welche Maßnahmen zur Feststellung und Vermeidung derartiger Konflikte vorgenommen werden können.

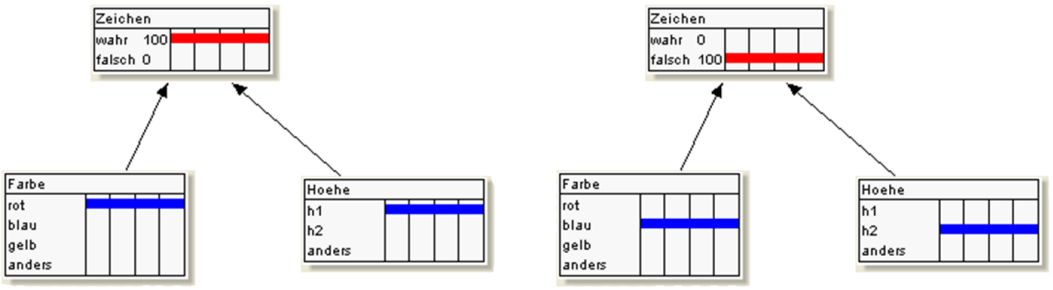


Abbildung 5.4: Klassifikationsergebnisse in Abhängigkeit der gegebenen Merkmale *Farbe* und *Hoehe* (*Bayes-Beispiel 2*).

Wie im vorherigen Abschnitt erläutert gibt es für die Darstellung einer Wahrscheinlichkeitsverteilung in der Regel mehrere Bayes'sche Netze und die Wahl der Kantenrichtungen schränkt die Berechnungsmöglichkeiten nicht ein. Auch im Beispiel kann sowohl bei gesetztem Zustand für die Merkmals- bzw. Ursachenvariablen die Zustandsverteilung für die Wirkungsvariable *Zeichen* berechnet werden (Prognose) als auch umgekehrt (Diagnose). Bspw. erhält man bei Festlegung des Zustands *wahr* für Variable `Zeichen` Ergebnisse für die Variablen `Farbe` und `Hoehe`; nämlich genau die Merkmalszustände, die für Schriftzeichen zutreffen (s. Abb. 5.6). *Bayes-Beispiel 1* und *2* zeigen wie Wissen, in diesem Fall über die Merkmale von Schriftzeichen, in einem Bayes'schen Netz in Struktur und CPTs abgebildet werden kann. Die Beispiel-Netze bestehen aus sehr wenigen Variablen. Der Formalismus

Bayes'scher Netze ermöglicht allerdings auch die Informationsverarbeitung in sehr komplexen Netzen (vgl. Kap. 1.3). Das Bayes'sche Netz bildet dabei stets einen effizienten Kompromiss zwischen der vollständigen Unabhängigkeitsforderung eines naiven Bayes-Klassifikators und der Notwendigkeit, für jede mögliche Kombination konditionale Wahrscheinlichkeiten anzugeben.

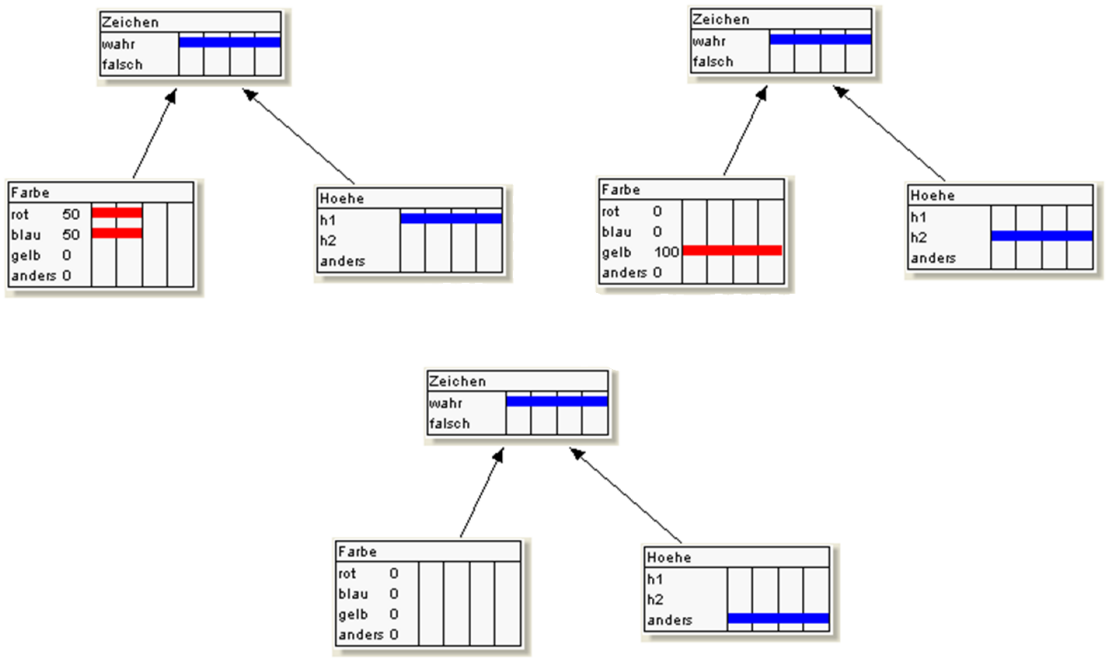


Abbildung 5.5: Abhängigkeit der Variablen `Farbe` von der Variablen `Hoehe` bei gegebener Evidenz für die Variablen `Zeichen` und `Hoehe` (*Bayes-Beispiel 2*).

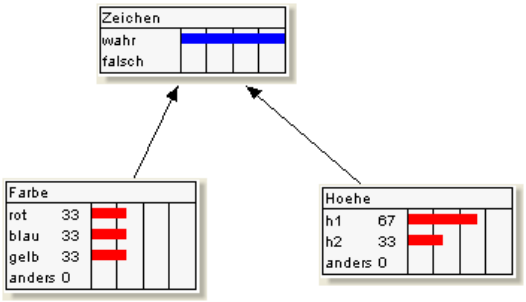


Abbildung 5.6: Zustandswahrscheinlichkeiten der Variablen `Farbe` und `Hoehe` bei gegebener Evidenz der Variablen `Zeichen` (*Bayes-Beispiel 2*).

5.2 Inferenzberechnungen

In einem Bayes'schen Netz wird Wissen in Form von konditionalen Abhängigkeits- und Unabhängigkeitsbeziehungen repräsentiert. Liegen Evidenzen vor, können diese durch das Netz propagiert und Informationen über abhängige Variablen abgeleitet, d.h. Schlussfolgerungen gezogen werden. Je nach Komplexität des Bayes'schen Netzes können hierfür exakte oder approximative Inferenzverfahren angewendet werden. Einige dieser Verfahren werden in den folgenden zwei Abschnitten vorgestellt. In Abschnitt 5.2.3 wird des Weiteren erläutert, wie die Inferenzberechnung auch für unsichere Evidenzen angewendet werden kann.
In jedem Fall gelten für das Rechnen mit konditionalen Wahrscheinlichkeiten die folgenden Eigenschaften (vgl. Jensen und Nielsen 2007, Kap. 1.4); P_1, P_2 und P_3 bezeichnen konditionalen Wahrscheinlichkeiten und *Dom* bezeichnet den Definitionsbereich (engl. domain):

1. $Dom(P_1P_2) = Dom(P_1) \cup Dom(P_2)$
2. Kommutativgesetz: $P_1 \cdot P_2 = P_2 \cdot P_1$
3. Assoziativgesetz: $(P_1 \cdot P_2) \cdot P_3 = P_1 \cdot (P_2 \cdot P_3)$
4. Neutrales Element: $\mathbf{1} \cdot P_1 = P_1$

Diese Eigenschaften können allgemein aufgefasst werden und gelten nicht nur für konditionale Wahrscheinlichkeiten. Sie basieren auf den von [Shafer und Shenoy 1990] aufgestellten sieben Axiomen für die Propagierung von *Potentialen* in einem *Junction-Tree* (s. Abschnitt 5.2.1.4).

5.2.1 Exakte Inferenzverfahren

5.2.1.1 Marginalisierung

Wenn das Bayes'sche Netz $\mathcal{B} = (\mathbf{X}, \mathbf{E}, \mathbf{\Theta})$ mit $\mathbf{X} = \{C, F_1, ...F_n\}$ als Klassifikator fungiert, besteht die Aufgabe darin, die Klasse c zu finden, die die konditionale Wahrscheinlichkeit $P(c|f_1, ..., f_n)$ maximiert. Mittels der CPTs kann zudem für jedes Merkmal eine unterschiedliche Gewichtung definiert werden, die die Stärke der Merkmalsbeeinflussung definiert. Ist die Evidenz einiger Variablen $\mathbf{Y} \in \mathbf{X}$ gegeben, können die Zustandswahrscheinlichkeiten unbeobachteter Variablen $\mathbf{Z} \in \mathbf{X} \backslash \mathbf{Y}$

berechnet werden. Hierzu werden die Wahrscheinlichkeiten aller Zustandskombinationen, in denen die Variablen $\mathbf{Y}$ jeweils den gefragten Zustand annehmen, aufsummiert. Diese Vorgehensweise wird als *Marginalisierung* bezeichnet (vgl. Jensen und Nielsen 2007 Kap. 2.3.2 bis 2.3.4). Mittels der Marginalisierung können aus der JPD sukzessive Variablen entfernt werden. Z.B. ergibt sich für die Marginalisierung der Variablen X_j aus der JPD Formel 5.4.

$$\mathcal{P}(\mathbf{X} \backslash X_j) = \sum_{X_j} \mathcal{P}(\mathbf{X}) \tag{5.4}$$

Auf diese Weise kann, gegeben $\mathcal{P}(A, B)$, die Wahrscheinlichkeitsverteilung $P(A)$ für die Zustände von A in Kombination mit den Zuständen von B berechnet werden, indem die Variable B marginalisiert wird:

$$P(A) = \sum_{B} \mathcal{P}(A, B)$$

Die Berechnung von $P(B)$ ist ebenso möglich.
Für die Marginalisierung gelten ähnliche Eigenschaften wie für die Multiplikation von konditionalen Wahrscheinlichkeiten (vgl. Jensen und Nielsen 2007, Kap. 1.4):

1. Kommutativität:
$$\sum_{A} \sum_{B} \mathcal{P}_1 = \sum_{B} \sum_{A} \mathcal{P}_1$$

2. Distributivität: Wenn $A \notin Dom(\mathcal{P}_1)$, dann
$$\sum_{A} Dom(\mathcal{P}_1) Dom(\mathcal{P}_2) = Dom(\mathcal{P}_1) \sum_{A} Dom(\mathcal{P}_2)$$

3. Neutrales Element:
$$\sum_{A} P(A|\mathbf{X}') = \mathbf{1}, mit A \notin \mathbf{X}'$$

Beispiel für die Marginalisierung: Für den naiven Bayes-Klassifikator aus Kap. 5.1.4 (Abb. 5.2) kann durch Marginalisierung z.B. die Wahrscheinlichkeit des Zustands *wahr* für die Variable `Zeichen` berechnet werden. Die Zustandswahrscheinlichkeiten der Variablen `Farbe` und `Hoehe` müssen dafür gegeben sein. Dann ergibt sich Rechenvorschrift 5.5.

$$P(Z = wahr) = \sum_{f \in F, h \in H} P(F = f, H = h, Z = wahr) \tag{5.5}$$

f und h bezeichnen die Zustände der Variablen `Farbe` und `Hoehe`, und $P(F = f, H = h, Z = wahr)$ die Wahrscheinlichkeit, dass die Variable `Zeichen` den Zustand `wahr` annimmt, wenn für die Variablen `Farbe` und `Hoehe` Evidenz für jeweils einen bestimmten Zustand f und h gesetzt ist.
Dem naiven Bayes-Klassifikator (*Bayes-Beispiel 1*, Abb. 5.2) liegen nun die in den Tabellen 5.4 und 5.5 dargestellten zwei CPTs zugrunde. Im Folgenden wird dieser veränderte naive Bayes-Klassifikator auch als *Bayes-Beispiel 1b* bezeichnet. Sind

F|Z	**wahr**	**falsch**
rot	0.333	0.001
blau	0.333	0.001
gelb	0.333	0.001
a	0.001	0.997

Tabelle 5.4: CPT ohne Nulleinträge für den naiven Bayes-Klassifikator zur Definition der Abhängigkeit zwischen den Variablen `Farbe` und `Zeichen`. (*Bayes-Beispiel 1b*)

H|Z	**wahr**	**falsch**
h1	0.4995	0.001
h2	0.4995	0.001
a	0.001	0.998

Tabelle 5.5: CPT ohne Nulleinträge für den naiven Bayes-Klassifikator zur Definition der Abhängigkeit zwischen den Variablen `Hoehe` und `Zeichen`. (*Bayes-Beispiel 1b*)

die Evidenzen `Farbe`=*anders* und `Hoehe`=*h2* gegeben, ergibt sich für die Berechnung der Posteriori-Wahrscheinlichkeiten der Variablen `Zeichen`

$$
\begin{aligned}
&P(Z = wahr | F = a, H = h2) \\
&\quad = P(Z = wahr) \cdot P(F = a | Z = wahr) \cdot P(H = h2 | Z = wahr) \\
&\quad = 0.6 \cdot 0.001 \cdot 0.4995 \\
&\quad = 0.0002997
\end{aligned}
$$

und

$$\begin{aligned} & P(Z = falsch|F = a, H = h2) \\ & = P(Z = falsch) \cdot P(F = a|Z = falsch) \cdot P(H = h2|Z = falsch) \\ & = 0.4 \cdot 0.997 \cdot 0.001 \\ & = 0.0003988. \end{aligned}$$

Die Summe der Wahrscheinlichkeiten aller Zustände muss auf 1 normiert werden, deshalb muss noch jeder Wert durch die Summe 0.0002997 + 0.0003988 = 0.0006985 dividiert werden. Das Ergebnis ist $P(Zeichen = wahr) = 0.43$ und $P(Zeichen = falsch) = 0.57$.

Bei der Marginalisierung handelt es sich um ein exaktes Inferenzverfahren. Es wird auch als *Variableneliminierung* bezeichnet, da hier die CPTs durch Eliminierung von Variablen verkleinert werden bis nur noch die gesuchte Verteilung übrig ist (vgl. Jensen und Nielsen 2007, Kap. 2.3.4, S. 42). Für naive Bayes-Klassifikatoren und allgemein Bayes'sche Netze mit einer geringen Anzahl Variablen ist die Marginalisierung sehr effizient. Die Anzahl der Summanden steigt jedoch exponentiell mit der Variablenanzahl, d.h. für große Netze ist die Marginalisierung sehr rechenintensiv Wiegerinck, Kappen und Burgers 2010.

5.2.1.2 Message-Passing-Algorithmus

Für schwach verknüpfte Bayes'sche Netze mit der Struktur eines *Polytree*[4] wurde in [Pearl 1986] der *Message-Passing-Algorithmus* als exaktes und effizientes Inferenzverfahren vorgestellt. Ausgehend von Knoten mit gegebenen Evidenzen werden sukzessiv die Eltern- und Kindknoten über Zustandsänderungen informiert und deren Zustände neu berechnet. D.h. die Aktualisierung einer Variablen geschieht durch Änderung der Parameter von benachbarten Variablen bzw. die Einflüsse von Eltern- und Kindknoten werden fusioniert.
Der Algorithmus ist sehr effizient, da für die Berechnung der Zustandswahrscheinlichkeiten eines Knotens nur dessen direkte Nachbarn betrachtet werden. Die Zeit für die Propagierung der Informationen ist proportional zum längsten Pfad im Polytree. Für einen m-nären Baum mit n Zuständen pro Variable werden $2n^2 + mn + 2n$

[4] Ein Graph ist ein Polytree genau dann, wenn es zwischen zwei Knoten höchstens einen ungerichteten Pfad gibt (Rebane und Pearl 1987).

Multiplikationen pro Aktualisierung benötigt. Eine weitere Optimierung der Rechenzeit ist möglich, indem Mehrfachberechnungen von Multiplikationen vermieden werden. Der Message-Passing-Algorithmus funktioniert nur für Bayes'sche Netze mit einer Polytree-Struktur, da nicht definiert ist wie mit Zyklen umgegangen werden soll. Erfreulicherweise kann ein komplex verknüpftes Bayes'sches Netz durch Zusammenfassen von Variablen (engl. Clustering) stets in einen Polytree transferiert werden (Korb und Nicholson 2004, Kap. 3.5.1). Im Folgenden werden ein einfaches und ein noch effizienters Clustering-Verfahren vorgestellt.

5.2.1.3 Einfaches Clustering

Als Beispiel sei das in Abb. 5.7 dargestellte Bayes'sche Netz gegeben. Es stellt eine Erweiterung von *Bayes-Beispiel 2* dar und wird im Folgenden als *Bayes-Beispiel 3* bezeichnet. Das Teilnetz bestehend aus den Variablen `Farbe`, `Hoehe` und `Farbe_Hoehe_Zeichen (FH)` entspricht *Bayes-Beispiel 2* (vgl. Abb. 5.3). Neben den Merkmalen *Farbe* und *Höhe* wirkt sich nun auch die *Position (P)* der gefundenen Region auf das Klassifikationsergebnis aus. Die Position ist ebenfalls abhängig von der *Farbe*, was in der Variablen `Position_Farbe_Zeichen (PF)` kodiert ist. Die Variable `Zeichen (Z)` repräsentiert das Gesamtergebnis der Klassifikation. Das Bayes'sche Netz hat keine Polytree-Struktur, da es einen ungerichteten Zyklus zwischen den Variablen `F`, `PF`, `FH` und `Z` aufweist.

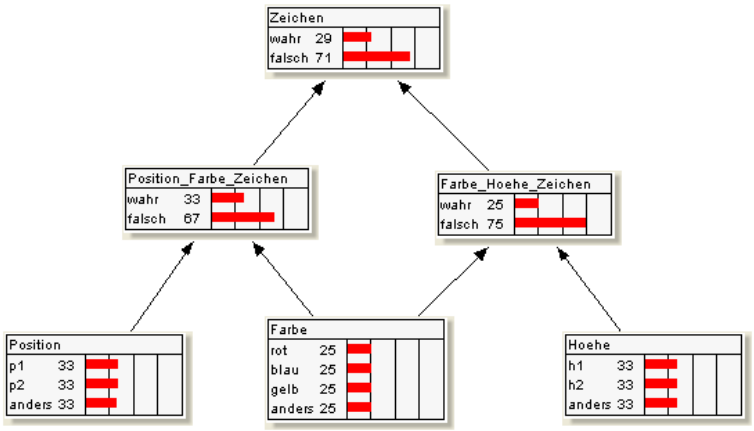


Abbildung 5.7: Erweitertes Bayes'sches Netz für die Schriftzeichenklassifikation mit den Merkmalen *Position*, *Farbe* und *Hoehe*. (*Bayes-Beispiel 3*)

Durch Zusammenfassen der Variablen `PF` und `FH` zu einer Variablen `PFH` entsteht

die in Abb. 5.8 dargestellte Polytree-Struktur (*Bayes-Beispiel 4*). Die neuen CPTs ergeben sich aus den ursprünglichen, wie in den Formeln 5.6 und 5.7 angegeben. Die Multiplikation in Formel 5.7 ist möglich, da die Variablen `PF` und `FH` voneinander unabhängig sind, wenn Evidenz für `P`, `F` und `H` gegeben ist. *Bayes-Beispiel 4* bildet genau dieselbe Wahrscheinlichkeitsverteilung ab wie *Bayes-Beispiel 3*, d.h. die JPDs sind identisch. Im Gegensatz zu *Bayes-Beispiel 3* kann für *Bayes-Beispiel 4* allerdings Inferenz mittels des Message-Passing-Algorithmus berechnet werden.

$$P(Z|PFH) = P(Z|PF, FH) \tag{5.6}$$

$$P(Z|PFH) = P(PF|P, F) \cdot P(FH|F, H) \tag{5.7}$$

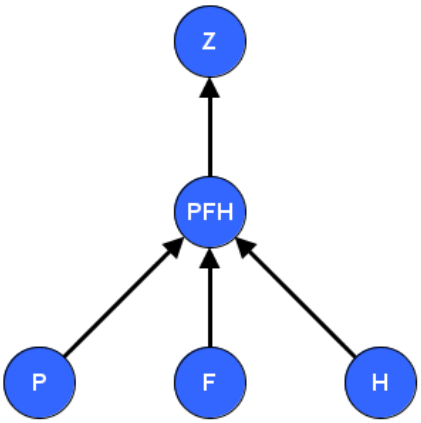


Abbildung 5.8: *Bayes-Beispiel 4* mit Polytree-Struktur, das durch Zusammenfassen von Variablen aus *Bayes-Beispiel 3* entstanden ist.

5.2.1.4 Junction-Tree-Algorithmus

Ein methodisches und effizientes Clustering-Verfahren für die Umwandlung in einen Polytree bietet der *Junction-Tree-Algorithmus* (Korb und Nicholson 2004, Kap. 3.5.2). Im Folgenden werden die einzelnen Schritte des Algorithmus erläutert.[5] In Abb. 5.9 sind die Umwandlungsschritte für das *Bayes-Beispiel 3* grafisch dargestellt.

[5]Die hier aufgeführte Variante entspricht dem ursprünglich von [Lauritzen und Spiegelhalter 1988] vorgestellten Algorithmus. Von [Shafer und Shenoy 1990] wurde unter der Bezeichnung *Join-Tree-Algorithmus* ein ähnliches Inferenzverfahren veröffentlicht.

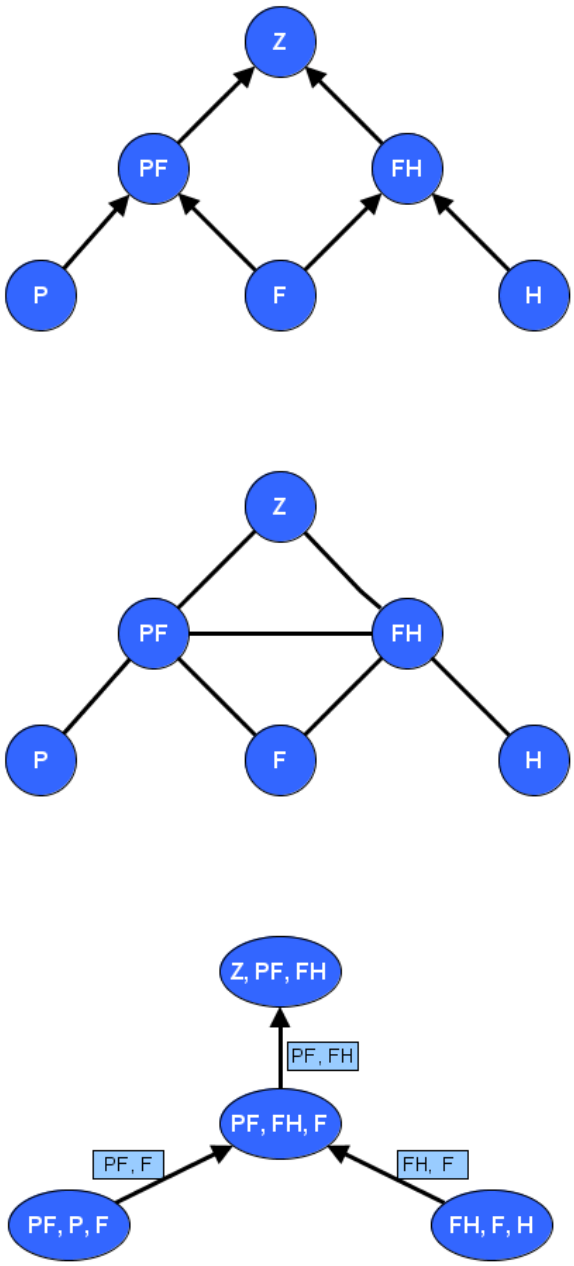


Abbildung 5.9: Umwandlung von einem Bayes'schen Netz zu einem Junction-Tree. Oben: Ausgangs-Netz; Mitte: moraler, triangulierter Graph; Unten: Junction-Tree mit Separatoren.

1. **Umwandlung in den moralen Graphen**: Der DAG wird durch Wegfall der Kantenrichtungen zu einem ungerichteten Graphen. Knoten, die ein gemeinsames Kind haben, werden durch eine neue Kante miteinander verbunden. Der entstandene morale Graph ist eine Perfect-Map (s. Kap. 5.1.3), da nur so

viele Kanten hinzugefügt wurden, wie maximal notwendig sind.

2. **Triangulierung**: Dem moralen Graphen werden Kanten hinzugefügt, sodass jeder Zyklus, der mehr als drei Variablen umfasst, einen Teilzyklus bestehend aus genau drei Variablen erhält. In [Jensen und Nielsen 2007] wird die Triangulierung mit dem folgenden Theorem noch genauer definiert: Ein ungerichteter Graph ist trianguliert genau dann, wenn alle Knoten durch sukzessives Entfernen eines *simplizialen* Knotens X entfernt werden können.
   Simpliziale Knoten sind diejenigen Knoten, die eine komplette Nachbarmenge besitzen; d.h. wenn die Nachbarmenge einschließlich des Knotens X eine Clique bildet[6]. Das bedeutet nicht, dass in einem triangulierten Graphen alle Knoten simplizial sind. Es muss durch das Vorhandensein von simplizialen Knoten lediglich eine *perfekte Eliminierungsfolge* geben, um letztendlich alle Knoten zu entfernen.
   Der morale Graph zu *Bayes-Beispiel 3* ist bereits trianguliert.

3. **Junction-Tree und Separatoren**: In dem triangulierten Graphen werden die maximalen Cliquen bestimmt. Eine Clique ist maximal, wenn durch das Hinzufügen eines weiteren Knotens die Cliqueneigenschaft verloren geht. Jede maximale Clique bildet einen zusammengesetzten Knoten (Cluster) im Junction-Tree, die entsprechend den Kantenrichtungen des ursprünglichen Graphenmodells miteinander verbunden werden. Jede Kante besitzt einen *Separator*, der die Schnittmengenknoten der benachbarten Cluster beinhaltet.

4. **Parameter für die Cluster-CPTs**: Die Cluster-CPTs des Junction Tree ergeben sich aus den CPTs des ursprünglichen Bayes'schen Netzes. Die Anzahl der Einträge eines Cluster-CPTs ist das Produkt der Zustandsanzahlen der zusammengefassten Variablen. Begonnen wird mit Cluster-CPTs, die nur aus Einsen bestehen. Dann wird für jeden Knoten X im originalen Netz ein Knoten Y im Junction-Tree gewählt, der X und alle Elternknoten von X enthält. Der Cluster-CPT zu Knoten Y wird angepasst, indem er mit dem originalen CPT $P(X|Pa(X))$ multipliziert wird.

Der Junction-Tree ist ein Polytree. Folglich ist hier eine effiziente Inferenzberechnung mit dem Message-Passing-Algorithmus möglich. Die Rechenkomplexität ist

[6]Eine Clique definiert in einem ungerichteten Graphen eine Teilmenge von Knoten, in der alle Knoten paarweise miteinander verbunden sind (vgl. Korb und Nicholson 2004, Kap. 3.5.2).

proportional zur Tiefe des Junction-Tree.
Der Nachteil bei der Umwandlung in einen Polytree ist, dass größere CPTs entstehen, die zu einer höheren Rechen- und Speicherkomplexität führen (s. Abschnitt 5.5). Die Effizienzsteigerung durch Verwendung einer Polytree-Struktur kann deshalb verringert sein (Korb und Nicholson 2004, Kap. 3.5.4). Zudem existieren für einen Graphen in der Regel mehrere Triangulierungen und zu einem trianguliert en Graphen können wiederum verschiedene Junction-Trees aufgebaut werden. Das Finden einer optimalen Triangulierung ist ein NP-hartes Problem. In [Jensen und Jensen 1994] wird eine Methode zum Finden eines optimalen Junction-Tree vorgestellt. Daneben wird auch gezeigt, dass das NP-Problem der Triangulierung nicht lösbar ist und der Junction-Tree-Algorithmus das effizienteste Clustering-Verfahren ist.

5.2.2 Approximative Inferenzverfahren

Der Junction-Tree-Algorithmus ist ein exaktes und zudem effizientes Berechnungsverfahren, das Nutzen aus der graphischen Struktur eines Bayes'schen Netzes zieht. Dennoch ist er nicht für jedes Bayes'sche Netz beliebiger Größe effizient anwendbar. Der Algorithmus eignet sich vor allem für schwach verknüpfte Netze. Dort können auf einfache Weise ein Junction-Tree aufgebaut und die Inferenzen effizient mittels Massage-Passing berechnet werden. Die Anzahl der Kombinationen und damit der Berechnungen steigt jedoch auch hier exponentiell mit der Variablenanzahl (Wiegerinck, Kappen und Burgers 2010 und Abschnitt 5.5). Bei sehr komplexen Netzen mit tausenden Variablen und Beziehungen ist daher nur der Einsatz von approximativen Verfahren praktikabel. Dafür existieren einige stochastische Methoden wie bspw. das *Gibbs-Sampling* (Details z.B. in Korb und Nicholson 2004, Kap. 3.6). In [Draper und Hanks 1994] wird ein lokales Inferenzverfahren vorgestellt, bei dem weit entfernte Variablen, deren Einfluss auf Ausgabevariablen deshalb sehr gering ist, ignoriert werden. Daneben ermöglicht die Verwendung bestimmter Wahrscheinlichkeitsverteilungen in den CPTs wie bspw. *noisy-Or* und *noisy-MAX* effizientere Berechnungen (s. Wiegerinck, Kappen und Burgers 2010, S. 10-12). D.h. ob Inferenz effizient in einem Bayes'schen Netz berechnet werden kann, wird nicht nur maßgeblich durch die Netzgröße und den Vernetzungsgrad bestimmt, sondern kann auch durch die Art der Wahrscheinlichkeitsverteilungen beeinflusst werden.

5.2.3 Inferenz mit unsicheren Evidenzen

Bisher wurde davon ausgegangen, dass Evidenzen sicher sind, d.h. sich jede beobachtete Variable genau in einem sicheren Zustand befindet und alle anderen Zustände Wahrscheinlichkeit 0 haben. Diese Art von Evidenz wird als *harte Evidenz* (Bilmes 2004) oder in [Korb und Nicholson 2004] auch als *spezifische Evidenz* bezeichnet. In vielen Anwendungsbereichen weisen Beobachtungen jedoch Unsicherheiten auf, wie z.B. in der Bildverarbeitung. In [Pearl 1988] wurde erstmals konkret an Beispielen erläutert, wie unsichere Evidenzen unter der Bezeichnung *virtuelle Evidenzen* (engl. virtual evidence) in einem Bayes'schen Netz verarbeitet werden können. Seitdem werden in der Literatur die Eigenschaften von unsicheren Evidenzen und ihre Handhabung recht konträr beschrieben (s. Abschnitt 5.2.3.3). Im Folgenden wird eine Unterscheidung in *weiche Evidenz* (engl. soft evidence) und *virtuelle Evidenz* vorgenommen, die mit der in [Bilmes 2004] und [Koski und Noble 2011] konform ist.

5.2.3.1 Weiche Evidenz

Eine weiche Evidenz ist eine Wahrscheinlichkeitsverteilung über die Zustände einer Variablen. Wird für eine Variable X_i eine weiche Evidenz gesetzt, verändert sich $P(X_i)$ in $Q(X_i)$. Ist die A-priori-Verteilung $P(X_i)$ gleichverteilt, ist zudem $Q(X_i)$ gleich der weichen Evidenz. Die weiche Evidenz wird dann an alle Variablen X_j mit $X_i \in Pa(X_j)$ und $j \neq i$ propagiert. Das bedeutet, die Funktionen der Übergangswahrscheinlichkeiten $P(X_j|X_i)$ werden durch eine neue ersetzt. Mit der in [Jeffrey 1965] formulierten *Wahrscheinlichkeits-Kinematik*, vor allem bekannt als *Jeffrey-Konditionierung* bzw. *Jeffrey-Regel*, kann die CPT aktualisiert werden:

$$Q(X_j) = P(X_j|X_i)Q(X_i) \tag{5.8}$$

Die weiche Evidenz SE beeinflusst die einzelnen Zustandswahrscheinlichkeiten der Variablen X_i und X_j folgendermaßen:

$$P(x_{ik}|se_k) = Q(x_{ik}) \tag{5.9}$$

und

$$Q(x_{jl}) = P(x_{jl}|se_k) = \sum_{x_{jl}} P(x_{jl}|x_{ik})Q(x_{ik}) \tag{5.10}$$

Im Originalbeispiel von R. Jeffrey ist die Hypothese eine Annahme über die Farbe eines Kleidungsstücks. Bei guten Lichtbedingungen kann die Farbe und damit die Evidenz E klar festgestellt werden. Bei schlechten Lichtbedingungen ist nur noch eine unsichere Vermutung SE möglich.
Die Jeffrey-Regel ist nicht kommutativ, d.h. die Reihenfolge der gegebenen weichen Evidenzen bestimmt das Endergebnis. Letztendlich wirkt sich nur die zuletzt eingegebene weiche Evidenz aus und löscht alles Vorherige (Chan und Darwiche 2005). Die weiche Evidenz ist in diesem Fall ähnlich zur harten Evidenz, einmal gesetzt kann sie nicht mehr durch Inferenz beeinflusst werden (vgl. Korb und Nicholson 2004, Kap. 3.4). Das bedeutet auch, dass alle $Pa(X_i)$ keinen Einfluss mehr auf X_i haben. Je nach Anwendungsfall kann diese Eigenschaft ein Vor- oder auch Nachteil sein.
Als Alternative kann die Jeffrey-Regel auch im Sinne von [Pearl 1988] als *Likelihood-Verhältnis* dargestellt werden. Die weiche Evidenz wird dann mittels einer sogenannten *virtuellen Variable V* für Variable X_i gesetzt. V besitzt als einzigen Elternknoten die zu beeinflussende Variable X_i. Wenn X_i binär ist, berechnet sich das Likelihood-Verhältnis wie folgt, mit $X_i = true$ und $\overline{X}_i = false$:

$$L(X_i) = \frac{P(V|X_i)}{P(V|\overline{X}_i)} = \frac{P(X_i)Q(\overline{X}_i)}{Q(X_i)P(\overline{X}_i)} \tag{5.11}$$

Die virtuelle Evidenz ist in $P(V|X_i)$ kodiert und wird gesetzt, indem eine harte Evidenz für V gesetzt wird. Die Evidenz ist fest definiert, daher sind die übrigen CPT-Einträge von $P(V|X_i)$ irrelevant und beliebig. Zudem gibt es nur eine Propagationsrichtung; von der virtuellen Variable V zum Elternknoten X_i. Das sind auch die Gründe, warum dieser Variablentyp als virtuell bezeichnet wird. Eine Variable X_i kann nun über mehrere virtuelle Variablen V_j durch verschiedene weiche Evidenzen beeinflusst werden. Die virtuellen Variablen werden dabei, ähnlich wie in einem naiven Bayes-Klassifikator, als unabhängig voneinander betrachtet.
Das Konzept der virtuellen Variable fügt sich problemlos in den Formalismus Bayes'scher Netze ein und wird daher normalerweise auch verwendet, um weiche Evidenzen zu setzen. Zudem können auf diese Weise mehrere weiche Evidenzen eine Variable beeinflussen und es müssen nicht zur Laufzeit CPTs von „echten" Variablen geändert werden. Das bedeutet außerdem, dass beim Setzen einer virtuellen Evidenz für Variable X_i mittels einer virtuellen Variable V, Verbindungen zu $Pa(X_i)$ erhalten bleiben. In Abbildung 5.10 ist das im Vergleich mit der bloßen Anwendung

der Jeffrey-Regel anhand zweier Bayes'scher Netze veranschaulicht. Besitzt X_i keine Elternknoten, ist zudem die Verwendung einer virtuellen Variable dasselbe, wie die direkte Anpassung der A-priori-Wahrscheinlichkeit $P(X_i)$ in $Q(X_i)$.

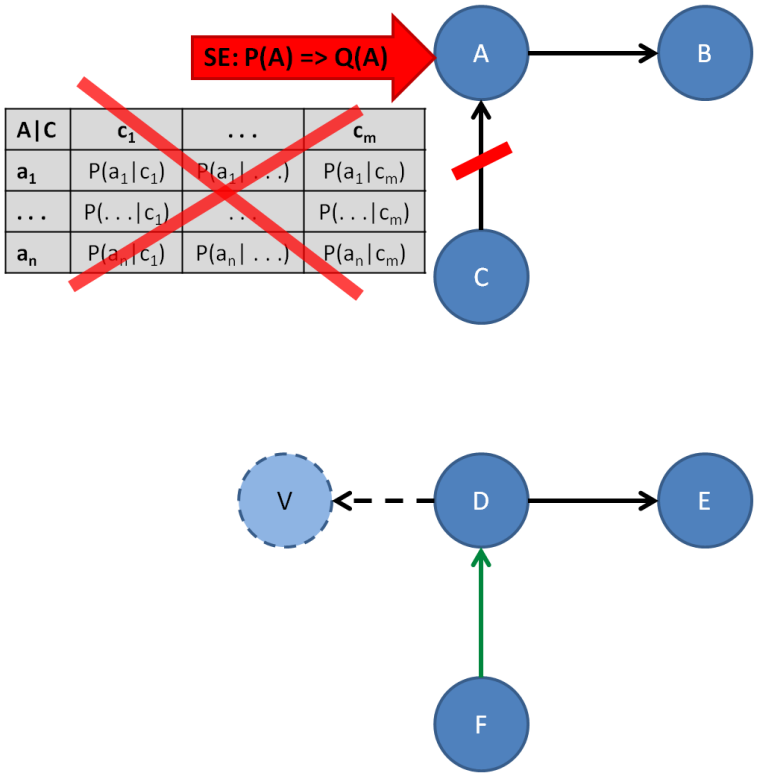


Abbildung 5.10: Oben Bayes'sches Netz $\mathcal{B}1$ mit $X = \{A, B, C\}$: Durch Eingabe einer weichen Evidenz SE nach der Jeffrey-Regel wird die Verbindung zwischen A und dem Elternknoten C und damit CPT $P(A|C)$ aufgelöst. Unten Bayes'sches Netz $\mathcal{B}2$ mit $X = \{D, E, F\}$: Durch Eingabe der weichen Evidenz über die virtuelle Variable $V \notin X$ bleibt die Verbindung zwischen D und Elternknoten F unverändert.

5.2.3.2 Virtuelle Evidenz

In realen Anwendungen sind konkrete Likelihoods oftmals schwer zu ermitteln, die „ungenauen" *Likelihood-Verhältnisse* hingegen einfacher festzustellen. Z.B. „Morgen scheint mit Wahrscheinlichkeit 0.8 die Sonne." vs. „Es ist viermal wahrscheinlicher, dass morgen die Sonne scheint, als dass es regnet." bedeutet, dass sich der Beobachter nicht sicher ist, ob es sich bei der Angabe „80% " tatsächlich um eine richtige Wahrscheinlichkeit handelt, oder nur um den relativen Glauben. Es ist dann notwendig, die Behauptung „Für eine Variable ist der Zustand a zweimal so

wahrscheinlich wie der Zustand *b*." auszudrücken. Die Angabe eines Likelihood-Verhältnisses ist auch erforderlich, wenn die unsichere Evidenz von externen Quellen stammt. Oft ist dann nicht bekannt, mit welcher konkreten Wahrscheinlichkeit sich die unsichere Evidenz auf eine Variable des Bayes'schen Netzes auswirken soll und ob eine solche Information evtl. schon im Netz enthalten ist (vgl. Bilmes 2004, Kap. 5.2).

Unsichere Evidenzen, die ein Verhältnis zwischen Likelihoods bzw. relative Häufigkeiten abbilden, werden als *virtuelle Evidenzen* bezeichnet. Es ist lediglich das Verhältnis entscheidend, d.h. die Angabe „Der Zustand *a* verhält sich zu Zustand *b* wie 4 zu 2". Das hat dieselbe Bedeutung wie „Der Zustand *a* verhält sich zu Zustand *b* wie 2 zu 1". Die konkreten Werte kürzen sich bei der Inferenzberechnung weg (vgl. Bilmes 2004, Kap. 4). Das Likelihood-Verhältnis beschreibt die Stärke, die eine virtuelle Evidenz auf die Posteriori-Wahrscheinlichkeit einer Variablen hat. Daraus folgt auch, dass sich die Likelihood-Verhältnisse nicht zu 1 summieren müssen und daher nicht als Wahrscheinlichkeiten interpretierbar sind (Pearl 1986, S. 254).

Für ein Bayes'sches Netz $\mathcal{B} = (\mathbf{X}, \mathbf{E}, \mathbf{\Theta})$ und eine virtuelle Variable V gilt daher $V \notin \mathbf{X}$. Die Verwendung von virtuellen Variablen dient lediglich dazu eine virtuelle Evidenz in mathematisch konsistenter Weise in das Modell $\mathcal{B}$ einzugeben (vgl. Bilmes 2004, Kap. 4).

Die Likelihood-Verhältnisse können kombiniert werden. Das führt aber zu der Problematik, wie gewährleistet wird, dass kombinierte Likelihood-Verhältnisse die gewünschte unsichere Evidenz ausdrücken. Das Ergebnis ist oftmals nicht intuitiv, vor allem wenn die A-priori-Verteilung der Variablen X_i nicht gleichverteilt ist und/oder X_i Elternknoten besitzt. Es finden sich in der Literatur zahlreiche Arbeiten über verschiedene Kombinationsmöglichkeiten von virtuellen Evidenzen. In [Halpern und Pucella 2009] basiert die Kombination z.B. auf der Evidenztheorie von Shafer.

5.2.3.3 Unsichere Evidenz in der Literatur

In der Literatur ist die Beschreibung von unsicheren Evidenzen und ihrer Berechnung in einem Bayes'schen Netz sehr unterschiedlich und damit auf recht verwirrende Weise dargestellt. Während in [Bilmes 2004] und [Koski und Noble 2011] die auch hier verwendete Unterscheidung in virtuelle Evidenz und weiche Evidenz vorgenommen wird, werden in [Korb und Nicholson 2004] unsichere Evidenzen

nur kurz unter dem Begriff virtuelle Evidenz erläutert. [Jensen und Nielsen 2007] geht ebenfalls nur kurz auf diese Thematik ein und meint unter der Bezeichnung *Likelihood-Evidenz* offensichtlich die virtuellen Evidenzen von [Bilmes 2004]. In anderen Quellen werden virtuelle Evidenz und weiche Evidenz als Synonyme aufgefasst. Letztendlich kommt es, wie auch in [Chan und Darwiche 2005] erläutert, auf die Interpretation der unsicheren Evidenz an und wie diese andere Variablen beeinflussen soll. Die im Rahmen dieser Arbeit als weiche Evidenz bezeichnete unsichere Evidenz beschreibt eine Wahrscheinlichkeitsverteilung und verhindert in ihrer ursprünglichen Form die Einflussnahme von Elternknoten, wie auch die Inferenzberechnung mit der Jeffrey-Regel zeigt. Virtuelle Evidenz hingegen beschreibt ein Likelihood-Verhältnis und beeinflusst die Wahrscheinlichkeitsverteilung einer Variablen relativ. Da diese Verhältnisse nicht zwingend als Wahrscheinlichkeit interpretierbar sind, wird virtuelle Evidenz mittels einer virtuellen Variable in Bayes'sche Netze eingegeben. Die Ansätze von Jeffrey und Pearl sind jedoch nicht gegensätzlich, sondern lassen sich ineinander überführen, wie bereits oben für die Jeffrey-Regel gezeigt. In [Chan und Darwiche 2005] wird das recht ausführlich diskutiert.
Schließlich kann auch sichere Evidenz von einer virtuellen Variablen kodiert werden. Der Unterschied ist, dass dann der CPT an der zu setzenden Stelle eine 1 enthält und ansonsten 0. Damit können alle Arten von Evidenz mittels einer virtuellen Variable in einem Bayes'schen Netz gesetzt werden und es ist stets die Kombination mehrerer Evidenzen möglich. Die Ergebnisse sind je nach gewähltem Ansatz selbstverständlich unterschiedlich zu interpretieren. In [Pearl 1988] wird virtuelle Evidenz daher auch als Verallgemeinerung des Evidenzbegriffs in Bayes'schen Netzen definiert; durch virtuelle Evidenz wird die Mächtigkeit Bayes'scher Netze erhöht.

5.3 Generierung von Bayes'schen Netzen

Bayes'sche Netze beschreiben in Struktur und Parametern Wahrscheinlichkeitsverteilungen und können damit verschiedenste Informationen repräsentieren. Die Frage ist, wie zu einer Problemstellung ein geeignetes Bayes'sches Netz gefunden und erstellt werden kann.
Kleine Bayes'sche Netze, d.h. bestehend aus wenigen Dutzend Variablen, werden

oft von Experten manuell erstellt. Für den manuellen Aufbau stehen zahlreiche kommerzielle und auch kostenlose Softwarelösungen zur Verfügung (s. Anhang D). Neben einer grafischen Benutzeroberfläche für das komfortable Zusammenstellen der Netzstruktur, können hier auch Variablen mit Evidenz belegt und Inferenzen berechnet werden.
Ab einer gewissen Größe ist die manuelle Erstellung von Bayes'schen Netzen jedoch nicht mehr handhabbar. Während außerdem die Netzstruktur durch vorhandenes Expertenwissen meist relativ einfach festgestellt werden kann, gestaltet sich das Finden der Parameter oftmals als sehr anspruchsvoll und fehleranfällig. Selbst Experten fällt es oft schwer in einem Anwendungsbereich die korrekten probabilistischen Zusammenhänge zu erkennen. [Beierle und Kern-Isberner 2006, Kap. 12.6, S. 403] postulieren, dass die Feststellung geeigneter Parameter meist mit einem erheblichen statistischen Aufwand verbunden ist. Daneben kann es sich je nach Anwendung auch um dynamisch veränderliche Netze handeln (s. Abschnitt 5.7). Dynamische veränderliche Netze müssen zwangsläufig zur Programmlaufzeit automatisch generiert werden. Für die automatische Generierung gibt es zwei Möglichkeiten; ein Bayes'sches Netz kann automatisch basierend auf einer formalen Beschreibung des darzustellenden Wissens aufgebaut oder basierend auf Daten gelernt werden. Im Folgenden werden diese beiden Ansätze näher beschrieben.

5.3.1 Netzgenerierung aus formaler Wissensbeschreibung

Für die Realisierung der ersten Möglichkeit ist das Vorhandensein des Wissens in maschinenlesbarer Form erforderlich. Bei der Generierung eines Bayes'schen Netzes auf Basis formal beschriebenen Wissens handelt es sich im Wesentlichen um einen Transfer des Wissens von dem einen in das andere Format. Der Transfer ist umso effizienter, je mehr das Format der Wissensbeschreibung bereits der Struktur des Bayes'schen Netzes entspricht. Zudem sollten alle Informationen explizit beschrieben sein; vor allem die konditionalen Abhängigkeiten. Die Verwendung von Namenskonventionen in der Wissensbeschreibung versteht sich für ein maschinenlesbares Format von selbst. Bei Existenz einer geeigneten formalen Wissensbeschreibung kann die Netzstruktur auf eine relativ einfache Weise automatisch generiert werden.
Weitaus schwieriger gestaltet sich der Erwerb der Parameter. Es kann gefordert werden, dass zu jeder Variablenabhängigkeit auch die Parameter für diese in der

Wissensbeschreibung angegeben sind. In der Realität ist das aber, z.B. bei einer Systemspezifikation, oftmals nicht der Fall und kann auch nicht auf einfache Weise hinzugefügt werden. Hier sind Zusatzinformationen erforderlich, die die Abhängigkeitsbeziehungen mit konkreten Parametern definieren. Dabei kann auf verschiedene Modelle zurückgegriffen werden, die zu sehr effizienten Inferenzberechnungen führen (s. Kap. 5.2).
In [Borth und Hasseln 2002] ist beschrieben, wie mittels eines regelbasierten Expertensystems aus einer formalen Systemspezifikation ein Bayes'sches Netz aufgebaut werden kann.[7] Nach der Generierung werden einige Optimierungen vorgenommen. Das soll das Bayes'sche Netz vereinfachen und damit verbessern. So können redundante Informationen beseitigt werden und eine performantere Netzstruktur entstehen. Die Optimierungen werden dort ebenfalls durch ein regelbasiertes System gesteuert. In [Wiegerinck, Kappen und Burgers 2010] wird eine Anwendung beschrieben, bei der das Bayes'sche Netz an sich klein ist, aber dynamisch zur Laufzeit erstellt werden muss. Auch die dynamische Erstellung wird durch die Bereitstellung einer geeigneten Wissensbeschreibung ermöglicht.

5.3.2 Netzgenerierung durch datenbasiertes Lernen

Es gibt eine Vielzahl von Verfahren um die Netzstruktur oder die Parameter auf Basis von Daten $\mathcal{D}$ zu lernen. Bei den Lerndaten handelt es sich um eine Menge von Datensätzen d_i, die jeweils eine mögliche Zustandskombination aller Variablen des Netzes darstellen. Im optimalen Fall sind $\mathcal{D}$ vollständige, statistische Daten, d.h. jeder Zustand jeder Variablen ist in mindestens einem Datensatz vertreten und es gibt keine unbekannten bzw. fehlenden Zustandsbelegungen oder Variablen. Die statistische Verteilung sollte das Vorkommen der Zustandskombinationen in der Realität möglichst exakt wiedergeben. Basierend auf den Daten ist ein Bayes'sches Netz $\mathcal{B} = (\mathbf{X}, \mathbf{E}, \mathbf{\Theta})$ zu lernen, das die Daten $\mathcal{D}$ am besten repräsentiert.

5.3.2.1 Ausgangslagen

In realen Anwendungsbereichen existieren vier verschiedene Ausgangslagen für das Lernen von Bayes'schen Netzen, die auch in [Wittig 2003] erläutert werden:

[7] Das in [Borth und Hasseln 2002] vorgestellte Bayes'sche Netz ist in der Lage das Verhalten von technischen Systemen zu diagnostizieren.

Wenn die Daten vollständig sind und die Netzstruktur bereits bekannt ist, ist das Lernen der Parameter unproblematisch. Die Übergangswahrscheinlichkeiten können dann direkt aus der heuristischen Verteilung der Daten abgeleitet werden. Eine unbekannte Struktur lässt sich ebenfalls relativ einfach aus vollständigen Daten herleiten. Bei unvollständigen Daten müssen hingegen kompliziertere Verfahren eingesetzt werden, die das Fehlen von Informationen geeignet kompensieren. Der schwierigste Fall ist daher, wenn sowohl die Struktur unbekannt als auch die Daten unvollständig sind. Es kann dann nur mit aufwendigen Verfahren ein optimales Bayes'sches Netz angenähert werden.

5.3.2.2 Optimierungsproblem

Im Wesentlichen handelt es sich stets um eine Suche nach dem am besten passenden Bayes'schen Netz. Die Menge der möglichen Bayes'schen Netze steigt mit der Anzahl an Variablen und Zuständen rapide an, weswegen das Suchproblem NP-hart ist. Alle möglichen Bayes'schen Netze auf Eignung zu prüfen ist bereits bei einer geringen Anzahl von Variablen nicht mehr praktikabel. Für drei Variablen gibt es 25 mögliche Netzstrukturen, bei zehn Variablen sind es bereits $4,2 \cdot 10^{18}$ und es kann gezeigt werden, dass die Anzahl möglicher Strukturen mehr als exponentiell mit der Anzahl an Variablen steigt (vgl. Jensen und Nielsen 2007, Kap. 7). Daher sind Vorkehrungen zu treffen, die die Suchmenge einschränken, dabei aber so weit wie möglich optimale Netzstrukturen beibehalten. In [Heckerman 1985] wird das Lernen von Bayes'schen Netzen treffend als Kombination von Wissen und statistischen Daten charakterisiert.

5.3.2.3 Unter- und Überparametrisierung

Ein generelles Problem beim Lernen ist die *Unter-* und *Überparametrisierung*. Unterparametrisierung bedeutet, dass das gelernte Bayes'sche Netz zu weit vom tatsächlichen Optimum entfernt ist und wichtige Abhängigkeitsbeziehungen nicht abgebildet werden. Bei der Überparametrisierung hingegen entspricht das gelernte Netz sehr exakt den Lerndaten, was zu Problemen führt, wenn die späteren Eingabedaten davon abweichen. D.h. das Bayes'sche Netz ist nicht flexibel genug und gewichtet in der Regel Abhängigkeiten zu stark. Für die Struktur bedeutet das im ersten Fall zu wenige und im zweiten Fall zu viele Kanten.

5.3.2.4 Strukturlernen

Für das Strukturlernen gibt es nach [Jensen und Nielsen 2007] im Wesentlichen zwei Arten von Algorithmen, die *einschränkenden* (engl. constraint based) und die *bewertenden* bzw. *Optimierungsverfahren* (engl. score based). Einschränkende Lernverfahren bedienen sich dem Konzept der konditionalen Unabhängigkeit, um beginnend mit einem vollständig verbundenen ungerichteten Graphen das Bayes'-sche Netz zu finden, das am besten die in den Daten enthaltene Information repräsentiert. Konditionale Unabhängigkeiten können durch statistische Tests, z.B. dem Chi-Quadrat-Test (χ^2) (Korb und Nicholson 2004, Kap. 8.10), ermittelt werden. Bei den Optimierungsverfahren wird mittels eines Suchalgorithmus und einer Bewertungsmetrik in einer Menge möglicher Bayes'scher Netze dasjenige gefunden, das die Bewertung maximiert. Daneben wird bei einigen Verfahren die Menge möglicher Bayes'scher Netze eingeschränkt, indem mit Angabe einer festen Variablenreihenfolge für jede Variable die potentiellen Elternknoten vorgegeben werden. Je umfangreicher die Bayes'schen Netze sind, umso aufwendiger sind die Lernverfahren. Für die Optimierungsverfahren können verschiedene effiziente Suchalgorithmen wie bspw. Greedy angewendet werden. Einschränkende Lernverfahren sind ebenfalls effizient und vor allem leicht zu implementieren, jedoch häufig fehleranfälliger. In der Praxis werden deshalb meistens die vielversprechenderen Optimierungsverfahren angewendet (Friedman 1999; Korb und Nicholson 2004, Kap. 8). Im Rahmen dieser Arbeit werden im Weiteren auch nur diese betrachtet.
Für Optimierungsverfahren gibt es eine Vielzahl von Bewertungsmetriken und Suchalgorithmen. Zudem kann in globale und lokale Verfahren unterschieden werden. Während globale Verfahren in jedem Schritt stets das gesamte Bayes'sche Netz ansehen und bewerten, wird bei lokalen Verfahren jeweils nur ein Knoten betrachtet. Lokale Algorithmen können parallelisiert werden, was bzgl. der Rechenzeit einen entscheidenden Vorteil darstellt. Sie können jedoch nur ein lokales Optimum finden, das nicht unbedingt das globale sein muss. Dennoch zeigt die Praxis, dass mit lokalen Algorithmen sehr gute Ergebnisse erzielt werden können. Einige lokale Optimierungsverfahren werden im Folgenden genauer erläutert und gegenübergestellt.

Lokale Bewertungsmetriken

- **K2-Metrik**
  Die *K2-Metrik* (Cooper und Herskovits 1992) definiert die Wahrscheinlich-

keit der Datenrepräsentation unter der Annahme, dass der Knoten X_i die Eltern $Pa(X_i)$ besitzt.

$$Q_{K2}(X_i, Pa(X_i)) = \prod_{j=1}^{M_i} \frac{(|X_i| - 1)!}{(N_{ij} + |X_i| - 1)!} \prod_{k=1}^{|X_i|} N_{ijk}!$$

Mit $|X_i|$ Anzahl der Zustände von X_i, M_i Anzahl der Zustandskombinationen von $Pa(X_i)$, N_{ij} Anzahl der Datensätze in denen $Pa(X_i)$ die Zustandskombination j haben und N_{ijk} Anzahl der Datensätze in denen X_i Zustand k besitzt und die j-te Zustandskombination von $Pa(X_i)$ vorliegt.

- **Log-Likelihood (LL)**
  Die *Log-Likelihood-Metrik* gibt an, wie viel Information benötigt wird, damit das Bayes'sche Netz $\mathcal{B}$ durch die Daten $\mathcal{D}$ repräsentiert wird. Bei Verwendung des Logarithmus' zur Basis 2 wird der Informationsgehalt in Bit ermittelt.
  $$Q_{LL}(\mathcal{B}, \mathcal{D}) = \sum_{i=1}^{n} \sum_{j=1}^{M_i} \sum_{k=1}^{|X_i|} N_{ijk} log \frac{N_{ijk}}{N_{ij}}$$
  Die Metrik basiert auf der Entropie (Korb und Nicholson 2004, Kap. 8.3).

- **Akaike-Informationskriterium (AIC)**
  Beim Akaike-Informationskriterium (Akaike 1973) wird LL um einen Bestrafungsfaktor erweitert. Diese Maßnahme verhindert Überparametrisierung.
  $$Q_{AIC}(\mathcal{B}, \mathcal{D}) = Q_{LL}(\mathcal{B}, \mathcal{D}) - K$$
  Mit K die Netzkomplexität:
  $$K = \sum_{i=1}^{n} (|X_i| - 1) \cdot M_i$$

- **MDL**
  Bei der *Minimum-Description-Length* (Rissanen 1978) wird LL eingeschränkt, indem die maximale Anzahl an Elternknoten festgelegt wird. Es soll damit das kleinstmögliche Bayes'sche Netz zur Datenrepräsentation gefunden werden. Nach [Rissanen 1978] wird auf diese Weise noch stärker als bei AIC Überparametrisierung verhindert, insbesondere bei sehr großen Datenmengen.
  $$Q_{MDL}(\mathcal{B}, \mathcal{D}) = Q_{LL}(\mathcal{B}, \mathcal{D}) - \frac{K}{2} log N$$

Die K2-Metrik gehört zu den Bayes'schen Bewertungsmetriken, während LL, AIC und MDL auf der Entropie basieren. In der Praxis besteht der wesentliche Unterschied darin, dass bei sehr kleinen Datenmengen bei LL, MDL und AIC oftmals Unterparametrisierung zu beobachten ist, wohingegen K2 bereits hier korrekte Abhängigkeiten erkennt. Allerdings tritt bei K2 häufig eine Überparametrisierung auf, was bei den auf der Entropie basierenden Bewertungsmetriken seltener der Fall ist und bei AIC und MDL zusätzlich durch einen Bestrafungsfaktor abgeschwächt wird. In Kap. 9 wird dieses Verhalten an Beispielen verdeutlicht.
Die Berechnung aller vorgestellten Bewertungsmetriken ist zeitintensiv. In [Herskovits 1991] wird bewiesen, dass die Komplexität der Entropieberechnung NP-hart ist und im schlechtesten Fall exponentiell mit der Variablenanzahl steigt. [Herskovits 1991] stellt aber auch einen Algorithmus für eine effizientere Entropieberechnung für Bayes'sche Netze vor. Die hohe Rechenkomplexität der K2-Metrik begründet sich in der Fakultätenberechnung und der Division durch sehr große Zahlen. Etwas Abhilfe schafft hier die Logarithmierung:

$$Q_{LOG-K2}(X_i, Pa(X_i)) = \prod_{j=1}^{M_i} (log((|X_i| - 1)!) - log((N_{ij} + |X_i| - 1)!) + \prod_{k=1}^{|X_i|} log(N_{ijk}!)$$

Unter Anwendung des Logarithmus wird die Division zur Substraktion, was den Rechenprozess beschleunigt. Jedoch entstehen weiterhin sehr große Zahlen und so ist auch Log-K2 nur für kleinere Netze und Datenmengen unter ca. 30.000 Datensätzen praktikabel. In Cooper und Herskovits 1992 wird die Komplexität einer effizienten Implementierung des K2-Algorithmus mittels Indexbäumen mit $O(nm)$ angegeben. n ist die Anzahl der Variablen und m die Anzahl der Datensätze, wobei angenommen wird dass m deutlich kleiner ist als die Anzahl aller möglichen Zustandskombinationen. Offensichtlich, steigt die Komplexität mit steigender Datenmenge um das Vielfache der Variablenanzahl.
Die Bewertungsmetriken stellen Produkte oder Summen über dem Beitrag jeder einzelnen Variablen X_i dar. D.h. jeder Faktor oder Summand bezieht sich nur auf genau eine Variable und deren Elternknoten. Die Bewertungen werden lokal für jede Variable vorgenommen, wodurch auch eine lokale Suche verwendet werden kann.

Lokale Suchalgorithmen

- **K2-Suche**
  Die *K2-Suche* wurde zusammen mit der K2-Metrik in [Cooper und Herskovits 1992] vorgestellt. Sie ist ein heuristischer Greedy-Algorithmus bei dem die Variablen in einer vorgegebenen Reihenfolge durchlaufen werden. Für jede Variable X_i werden Elternknoten aus der Menge $X_1, X_2, ...X_i - 1$ hinzugefügt, die die Bewertung des Netzes verbessern. Es entsteht ein Netz bei dem die Variable X_1 auf jeden Fall ein Wurzelknoten ist.

- **Hill Climbing (HC)**
  Beim *Hill Climbing* (Russell u. a. 1996) sind der Ausgangspunkt zur Suche eines optimalen Netzes unverknüpfte Variablen. In jedem Iterationsschritt wird zufällig eine der Operationen *Kante hinzufügen*, *Kante löschen* oder *Kantenrichtung ändern* ausgewählt. Die Operation wird dann für das Knotenpaar durchgeführt, für das die Bewertung maximal ist. D.h. zur Variablen X_i werden alle übrigen Variablen als potentielle Elternknoten in Betracht gezogen. Der Algorithmus endet, wenn keine weitere Maximierung mehr erreicht wird.

- **Repeated Hill Climbing (RHC)**
  Die Ausgangslage für das *Repeated Hill Climbing* ist eine Menge zufällig erzeugter Netzstrukturen. Für jede Netzstruktur werden jeweils die Operationen des Hill Climbing durchgeführt, bis ihre Bewertung maximal ist. Die erhaltene Netzstruktur mit der höchsten Bewertung repräsentiert die gegebenen Informationen am besten.

Bei der K2-Suche wird die Suchmenge eingeschränkt, indem die Variablenreihenfolge die potentiellen Elternknoten bestimmt. Durch Festlegung einer maximal zulässigen Elternanzahl kann die Suche weiter eingeschränkt werden. Ist derartiges Wissen im Anwendungsbereich gegeben, sind das akzeptable Einschränkungen. Liegt Wissen darüber allerdings nicht ursprünglich vor, führt diese Annahme oft zu fehlerhaften Netzstrukturen, da zu wenige Möglichkeiten betrachtet werden. In diesem Fall sollte z.B. HC oder RHC angewendet werden. Die Rechenzeit von (R)HC ist in der Regel höher als die der K2-Suche, da der Suchraum größer ist. Nach Friedman 1999 ist jedoch die Verwendung von Vorwissen bzgl. der Komplexität des Suchraums unerheblich, wenn es sich um ein lokales Optimierungsverfahren

handelt. Dann beträgt die Komplexität stets $O(n^2)$ mit n Anzahl der Variablen. RHC ist zudem zufallsbedingt und damit nicht deterministisch; bei mehrmaliger Durchführung kann es vorkommen, dass die gelernten Netze verschieden sind.
Ähnlich zu einem Gradientenverfahren wird mit den vorgestellten Suchalgorithmen jeweils ein lokales Optimum gefunden, was jedoch nicht zwingend dem globalen entspricht. Ebenfalls stellen Plateaus ein massives Problem dar, da der Wert dort an vielen Punkten konstant ist und eine konstante Bewertung normalerweise ein Abbruchkriterium für die Suche ist. Nur eine globale Suche löst dieses Problem vollständig. Liefern K2 und HC keine guten Ergebnisse, kann allerdings bereits durch RHC eine akzeptable Verbesserung erzielt werden. Indem bei RHC wiederholte Suchläufe mit einer jeweils anderen Start-Netzstruktur durchgeführt werden, ist dieses Verfahren weniger anfällig dafür das globale Optimum nicht zu finden. Daneben gibt es für die lokalen Suchalgorithmen noch zahlreiche weitere Verbesserungen und Erweiterungen. Beim *Simulated Annealing* sind z.B. zusätzliche Schritte im Suchraum erlaubt, um etwas breiter zu suchen und so falschen (lokalen) Optima zu entkommen (Russell u. a. 1996). Weitere Suchalgorithmen sowie auch die Anwendungsmöglichkeiten von Zusatzwissen sind z.B. in [Russell u. a. 1996] zu finden.

5.3.2.5 Parameterlernen

Auch für das Parameterlernen eines Bayes'schen Netzes gibt es verschiedene Algorithmen. Bei gegebener Struktur ist das Ziel hier die am besten passenden Wahrscheinlichkeitsverteilungen zu ermitteln. Zum einen wird nach [Jensen und Nielsen 2007] unterschieden, ob es sich um ein einmaliges Lernen handelt (*Batch*), oder ob die gelernten CPTs zur Laufzeit des Programms angepasst werden, wenn neue Daten hinzukommen (*Adaption*). Zum anderen können die gegebenen Daten vollständig oder unvollständig sein. Existieren vollständige Daten, bietet sich eine *Maximum-Likelihood-Schätzung* oder eine *Bayes'sche Schätzung* an (s. Jensen und Nielsen 2007). Der Unterschied besteht in der Interpretation des Wahrscheinlichkeitsbegriffs. Bevor diese zwei Schätzungen erläutert werden, werden folgende Annahmen getroffen: Das Parameterlernen erfolgt aus einer gegebenen Datenmenge $\mathcal{D}$, bestehend aus Datensätzen d, die jeweils eine mögliche Kombination der Variablenzustände enthalten. $\mathcal{D}$ ist vollständig, d.h. jedes d beinhaltet alle Variablen und es fehlen keine Zustände. Kombinationen von Elternknoten sind unabhängig von-

einander (lokale Unabhängigkeit). Z.B. ist $P(A|b_1, c_1)$ unabhängig von $P(A|b_2, c_2)$ mit $Pa(A) = \{B, C\}$ und (b_1, c_1), (b_2, c_2) sind zwei verschiedene Kombinationen der Variablenzustände von B und C. Zudem sind die Parameter bzw. CPTs verschiedener Variablen voneinander unabhängig (globale Unabhängigkeit).

Maximum-Likelihood-Schätzung $P(d|\mathcal{B})$ ist die Likelihood für das Bayes'sche Netz $\mathcal{B}$ gegeben dem Datensatz $d \in \mathcal{D}$. Daraus folgt die in Formel 5.12 dargestellte Likelihood von $\mathcal{B}$ gegeben $\mathcal{D}$. Die Multiplikation der Likelihoods ist möglich, da die Unabhängigkeitsannahme gilt.

$$L(\mathcal{B}|\mathcal{D}) = \prod_{d \in \mathcal{D}} P(d|\mathcal{B}) \tag{5.12}$$

Für effizientere Berechnungen wird oft die logarithmierte Likelihood (Log-Likelihood) verwendet:

$$LL(\mathcal{B}|\mathcal{D}) = \sum_{d \in \mathcal{D}} log_2 P(d|\mathcal{B}) \tag{5.13}$$

Es muss nun ein Parametersatz Θ_{max} für $\mathcal{B}$ gefunden werden, der die (Log-)Likelihood maximiert:

$$\Theta_{max} = \max L(\mathcal{B}_i|\mathcal{D}) = \max LL(\mathcal{B}_i|\mathcal{D}) \tag{5.14}$$

mit $\mathcal{B}_i$ ein strukturgleiches Bayes'sches Netz $\mathcal{B}$ mit Parametersatz Θ_i.
In der Praxis können die maximalen Likelihoods auf einfache Weise heuristisch bestimmt werden. Für jede Verteilung konditionaler Wahrscheinlichkeiten, z.B. $P(A = a_1|B = b_1)$, werden die Datensätze gezählt, die diese Kombination aufweisen. Anschließend wird durch die Gesamtanzahl der jeweiligen Kombination der Elternknoten dividiert: $\frac{N(A=a_1, B=b_1)}{N(B=b_1)}$. Folglich werden bei der Maximum-Likelihood-Schätzung die Datensätze als Ergebnisse von Zufallsexperimenten angesehen und dementsprechend eine Wahrscheinlichkeitsverteilung ermittelt.
Offensichtlich funktioniert diese heuristische Vorgehensweise nur gut für große $\mathcal{D}$. Je kleiner bzw. spärlicher $\mathcal{D}$ ist, umso häufiger ist es der Fall, dass bestimmte Kombinationen nicht vorkommen. Das führt zu einer Wahrscheinlichkeit von 0.0 und ist problematisch in realen Anwendungen (vgl. Abschnitt 5.4.3).

Bayes'sche Schätzung Im Fall von spärlichen Datenmengen $\mathcal{D}$ kann die Bayes'sche Schätzung zur Parameterbestimmung verwendet werden. Hier wird der Wahrscheinlichkeitsbegriff im Bayes'schen Sinn interpretiert. Anschaulich kann jede Parameterschätzung als ein Knoten in einem Bayes'schen Netz betrachtet werden, für

den eine A-priori-Verteilung angegeben wird; z.B. eine Gleichverteilung, falls keine konkreteren Angaben gemacht werden können. Dann wird das Bayes-Theorem angewendet, um die Posteriori-Verteilungen zu bestimmen. Die Bayes'sche Schätzung wird auch Maximum-a-posteriori-Schätzung genannt.
In der Praxis entspricht die Annahme von gleichverteilten A-priori-Verteilungen dem Hinzufügen eines virtuellen Experiments für die jeweilige Kombination. Durch die A-priori-Verteilungen werden die Daten deshalb um Rauschen erweitert und Nullwahrscheinlichkeiten sind nicht mehr möglich. Verfahren der Bayes'schen Statistik eignen sich ebenfalls gut für bereits verrauschte Daten; d.h. Daten in denen mitunter die Variablenwerte nicht exakt gemessen wurden.

5.3.2.6 Umgang mit unvollständigen Daten

In diesem Kapitel wurde bereits mehrmals auf die Möglichkeit von Nullwahrscheinlichkeiten für alle Zustände einer Variablen hingewiesen, wenn die CPTs Nulleinträge aufweisen. Beim Lernen kann es sehr leicht zu dieser Inkonsistenz kommen, da die Lerndaten meist nicht alle möglichen Zustandskombinationen beinhalten. Das ist oftmals auch gar nicht praktikabel, da die Anzahl der möglichen Zustandskombinationen exponentiell steigt. D.h. die Daten sind spärlich. Die Bayes'sche Schätzung fügt Rauschen hinzu, was diese Eigenschaft abschwächt. Eine andere sehr einfache Möglichkeit ist es die Übergangswahrscheinlichkeiten direkt anzupassen, sodass keine Wahrscheinlichkeit exakt 0 beträgt (vgl. Kap. 5.2).
In vielen realen Anwendungsbereichen sind die statistischen Daten nicht nur spärlich, sondern auch unvollständig. D.h. die darin vorkommenden Variablen repräsentieren nicht alle möglichen Zustände oder es fehlen sogar Variablen. Problematisch sind unvollständige Daten vor allem für das Parameterlernen. Wenn die gegebenen Daten keine vollständigen Verteilungen darstellen, müssen fehlende Informationen zwangsläufig aus einer anderen Quelle hinzukommen. Ist die Struktur des Netzes bekannt, stellen unvollständige Daten für das Parameterlernen zudem nur ein verhältnismäßig geringes Problem dar (vgl. Wittig 2003 Kap. 4.2.4). Wenn die Struktur nicht bekannt ist, kann durch fehlende statistische Daten jedoch auch das Strukturlernen negativ beeinflusst werden. Manche Abhängigkeiten oder Unabhängigkeiten können in diesem Fall nicht korrekt erkannt werden. Das gelernte Bayes'sche Netz bildet dann nicht die realen Verhältnissen ab und ist fehlerhaft.
Unter der Annahme, dass die fehlenden Werte unabhängig von beobachteten Werten

sind, kann der deterministische *EM-Algorithmus* (engl. Expectation-Maximization) (z.B. beschrieben in [Jensen und Nielsen 2007]) angewendet werden, der von der Maximum-Likelihood-Schätzung Gebrauch macht. Allerdings muss beachtet werden, dass es sich in diesem Fall nur um lokale Maximum-Likelihood-Schätzungen handelt, die vom globalen Optimum abweichen können. Dieselbe Unabhängigkeitsannahme muss auch bei approximativen Parameterschätzungen wie dem Gibbs-Sampling gelten (s. Korb und Nicholson 2004 Kap. 7 für Details). Da für praktische Anwendungen aber oftmals unvollständige Lerndaten vorliegen, für die die Annahme der Unabhängigkeit nicht gilt, wurden einige Verfahren entwickelt, um auch mit diesen Daten sinnvolle Netze zu lernen. In [Beierle und Kern-Isberner 2006] ist in Kap. 12.6 eine Möglichkeit hierfür dargestellt. Die gegebenen Übergangswahrscheinlichkeiten können als Regeln aufgefasst werden. P(A|B)=x entspricht der Regel *Wenn A wahr ist, dann ist B wahr mit Wahrscheinlichkeit x*. Die Wissensbasis wird somit in Form einer Regelmenge beschrieben. Oft existiert eine Vielzahl von Wahrscheinlichkeitsverteilungen, die diese Regelmenge erfüllen. Die Aufgabe ist es, die beste Verteilung zu finden. Die beste Verteilung ist diejenige, die gerade die Regelmenge beschreibt und keine weiteren Informationen hinzufügt. Mittels Maximierung der Entropie kann die beste Verteilung gefunden werden. Ähnliche und weitere Möglichkeiten zum Lernen mit unvollständigen Daten werden auch in [Wiegerinck und Heskes 2001] und [Korb und Nicholson 2004] beschrieben. In [Cooper und Herskovits 1992] werden im Zusammenhang mit der K2-Suche Ansätze hierzu vorgestellt.

Daneben können auch direkt die Daten verändert werden. Aus unvollständigen Daten kann durch zufälliges Ziehen mit Zurücklegen eine neue Datenmenge gewonnen werden, die anstatt der Originaldaten zum Lernen verwendet wird. Durch die zufällige Ziehung von Stichproben wird die Unvollständigkeit durch Rauschen überdeckt, was zu weniger Fehlern führt. Hierfür können bekannte Sampling-Methoden wie das *Monte Carlo Sampling* verwendet werden.

Mit derartigen Verfahren können auch bei unvollständigen Daten Bayes'sche Netze gelernt werden, die die Problemstellungen geeignet beschreiben. Wäre das nicht möglich, würde der produktive Einsatz vieler wissensbasierter Systeme an den unvollständigen Daten scheitern.

5.3.2.7 Umgang mit sehr großen Datenmengen

Problematisch und sehr ineffizient ist stets eine sehr große Datenmenge, die in der Realität durchaus vorkommt. Das *UCI Machine Learning Repository* [Frank und Asuncion 2010] bietet im Internet zahlreiche Daten an, die größtenteils von realen Anwendungen stammen. Die kleinste Datenmenge umfasst dabei 10 Datensätze und 31 Variablen, die größte 8 Mio. Datensätze und 100.000 Variablen. Mit steigender Anzahl an Variablen sollte auch die Größe der Daten zunehmen, um die Informationen geeignet repräsentieren zu können. Nimmt man an, dass die Daten sowohl bzgl. der Variablenzustände, als auch der Kombinationen vollständig sind, entstehen bereits bei wenigen Variablen sehr viele Datensätze. Z.B. gibt es bei drei binären Variablen $2^3 = 8$ Zustandskombinationen. Für das in Abschnitt 5.1 vorgestellte erweiterte Bayes'sche Netz zur Textklassifikation (Abb. 5.7) existieren $2^3 \cdot 3^2 \cdot 4 = 288$ Kombinationen. Diese Netze sind allerdings so klein, dass sie in der Regel manuell generiert werden können. Für größere Netze steigt die Kombinationsanzahl exponentiell an. Selbstverständlich ist der Sinn eines Bayes'schen Netzes, dass eben nicht alle Kombinationen bei der Inferenzberechnung betrachtet werden müssen. Durch die Netzstruktur wird die Menge an möglichen Kombinationen bereits auf die gewichtigen Abhängigkeitsbeziehungen stark eingeschränkt. Wenn nur Daten vorliegen, aus denen die Struktur gelernt werden soll, müssen aber genau genommen alle Kombinationen gegeben sein, damit in dieser vollständigen Menge die relevanten Kombinationen erkannt werden können. Wissen über die Struktur ist hier von Vorteil, um die Datenmenge möglichst klein zu halten bzw. zu reduzieren. Es ist auch möglich eine sehr umfangreiche Datenmenge durch geeignete Filter zu reduzieren. Dabei werden bspw. Datensätze entfernt, die Kombinationen repräsentieren, die nur mit einer verhältnismäßig minimalen Häufigkeit in der Menge enthalten sind, oder es werden nur die x häufigsten Kombinationen betrachtet.
Eine weitere Möglichkeit ist es, den Suchraum für das Strukturlernen durch paarweise Unabhängigkeitstests zu verkleinern. Das Maß der Abhängigkeit kann die *Mutual Information* (Cover und Thomas 1991) basierend auf der Entropie sein (s. Abschnitt 5.4.1). Mit Unabhängigkeitstests können z.B. die potentiellen Elternknoten einer Variablen identifiziert werden. Bei der Suche des besten Netzes müssen dann nur noch k Eltern der Variablen betrachtet werden und nicht mehr $n - 1$ (n Variablenanzahl, k meist deutlich kleiner als n). Die Anwendung der Unabhängigkeitstests muss stets im Hinblick auf die Verbesserung der Bewertung geschehen,

ansonsten werden falsche Netze gelernt (Friedman 1999). Des Weiteren wird bei der Mutual Information die Häufigkeit der Zustandskombinationen ermittelt und nicht beachtet, dass die betrachteten Variablen eine unterschiedliche Anzahl von Zuständen besitzen können. So wird stets die Verbindung stärker gewichtet, die eine höhere Anzahl an Zuständen aufweist. Das führt zu komplexeren Netzen, die wiederum fehleranfälliger sein können. Daher ist es sinnvoll, auch den Grad der Verbindung zu gewichten und auf diese Weise komplexere Verbindungen zu bestrafen.

5.3.3 Generierung von Bayes'schen Netzen in der Praxis

Das Finden einer geeigneten Netzstruktur birgt einen nicht unerheblichen rechentechnischen Aufwand. Da eine manuelle Netzerstellung nur für Netze bis zu einer bestimmten Größe möglich ist (vgl. Wiegerinck, Kappen und Burgers 2010, S. 2 f.), ist die automatische Generierung der Bayes'schen Netze für viele reale wissensbasierte Systeme notwendig. Die Generierung eines Bayes'schen Netzes aus einer formalen Spezifikation ist dabei viel geradliniger, einfacher und weniger fehleranfällig als das datenbasierte Lernen. Wenn eine geeignete formale Beschreibung des abzubildenden Wissens existiert, sollte daher stets die erst genannte Vorgehensweise eingesetzt werden. Während sich die Struktur des Bayes'schen Netzes oftmals direkt aus der Spezifikation herleiten lässt, sind die Parameter zur Beschreibung der konkreten Abhängigkeitsbeziehungen meist nicht explizit enthalten. Die Parameter müssen dem Netz entweder durch konkrete Zusatzinformationen oder durch die in Abschnitt 5.3.2 beschriebenen Datenmengen hinzugefügt werden. Das Parameterlernen aus einer gegebenen Datenmenge gestaltet sich am einfachsten, wenn die Struktur bereits bekannt ist und die Daten vollständig sind (Wittig 2003, Kap. 4.2.4). In der Praxis sind die Lerndaten aber sehr oft unvollständig. Wenn keine geeignete Spezifikation als Wissensbeschreibung vorliegt, muss zudem auch die Netzstruktur aus den Daten gelernt werden. Ein optimales Lernverfahren generiert daher sowohl die Struktur als auch die Parameter.
Die in [Wiegerinck, Kappen und Burgers 2010] vorgestellten Bayes'schen Netze suggerieren, basierend auf Daten gelernt werden zu können. Für das medizinische Diagnosesystem *Promedas*, das über 3500 Diagnosen stellen kann und in dessen Bayes'schen Netz über 50000 Beziehungen abgebildet sind, stammen die Informationen aus Patientendaten. Allerdings werden auf diesen Daten keine Lernalgorithmen angewendet, weswegen die Generierung des Netzes hier nur unter Anwendung

von Zusatzwissen automatisch erfolgen kann. Z.B. werden die Informationen über die Beziehungen zwischen Variablen und die Stärke ihrer Einflusskraft aufeinander in CPTs übertragen. Durch Wissen über die Variablenbeziehungen ist bereits die Netzstruktur bekannt und muss nicht mehr gelernt werden (Kappen 2005). Das komplexe Bayes'sche Netz von Promedas wird somit automatisch aus einer formalen Wissensbasis generiert.
In der Literatur konnte kein komplexes Bayes'sches Netz gefunden werden, das vollständig durch Lernen erzeugt wird. Zwar gibt es eine Vielfalt von Lernalgorithmen und unterstützende Software, deren tatsächlicher praktischer Einsatz für reale Problemstellungen ist jedoch unklar. Ebenso unklar ist bei Literaturbeispielen in der Regel die Abstammung der verwendeten Lerndaten, wie umfangreich diese sind und wie sie aussehen. Der Idealfall ist ein wissensbasiertes System, das nicht nur automatisch zu Schlussfolgerungen gelangt, sondern auch die Inferenzkomponente aus gegebenen Wissen automatisch aufgebaut wird. In realen Anwendungsbereichen ist dies allerdings oft nie vollständig möglich.

5.4 Korrektheit

Die Konsistenz einer Inferenzkomponente, die mittels der klassischen Logik Schlussfolgerungen zieht, kann durch Prüfung der Korrektheit der logischen Kalküle sichergestellt werden (s. z.B. Beierle und Kern-Isberner 2006, Kap. 3). Auf Basis der logischen Kalküle kann auch die Identität der Abbildung zwischen der Wissensbasis und der Inferenzkomponente validiert werden. Meist enthält die Wissensbasis in diesem Fall auch direkt die logikbasierten Regeln. Für komplexe probabilistische Modelle ist eine Korrektheitsprüfung hingegen weitaus schwieriger.
Die Grundlage für ein korrektes Bayes'sches Netz, das für ein bestimmtes Anwendungsgebiet konsistente Ergebnisse liefert, bilden die definierten Variablen und ihre Zustände (s. auch Korb und Nicholson 2004, Kap. 10.2). Für diese ist sicherzustellen, dass sie alle relevanten Informationen in korrekten Abhängigkeitsbeziehungen abbilden und klar formulierbare Aussagen repräsentieren. Die Zustandsmenge jeder Variablen muss vollständig sein und aus sich gegenseitig ausschließenden Zuständen bestehen. Bei der manuellen Erstellung ist es Aufgabe des Modellierers basierend auf Expertenwissen diese Eigenschaften zu garantieren. Bei der automatischen Generierung müssen die formale Wissensbeschreibung bzw. die Lerndaten

ein konsistentes Bayes'sches Netz herbeiführen. Da das nicht garantierbar ist, ist eine anschließende Korrektheitsprüfung erforderlich. Je komplexer ein Bayes'sches Netz ist, umso weniger praktikabel ist eine manuelle Prüfung. Zudem ist es auch für Experten mitunter schwer, Fehler zu erkennen.
Werden Bayes'sche Netze automatisch generiert, ist als erstes sicherzustellen, dass die Generierung fehlerfrei ist. Bei der Generierung aus einer formalen Spezifikation muss der Parser und beim Lernen das Lernverfahren geprüft werden.
In Abhängigkeit von der Beschaffenheit der formalen Spezifikation ist ein Parser relativ einfach zu realisieren. Da dabei, wie überall in der Softwareentwicklung, Fehler auftreten können, ist eine Prüfung des Parserergebnisses sinnvoll. Eine Möglichkeit hierfür ist die in [Marwedel 2007, Kap. 2.1] beschriebene Plausibilitätsprüfung, aus der formalen GUI-Designbeschreibung wieder die ursprüngliche GUI zu erhalten.
Wie aus Abschnitt 5.3.2 hervorgeht, existiert eine Vielzahl von Lernalgorithmen. Bereits die Größe und Beschaffenheit der Daten kann einen entscheidenden Einfluss auf das Lernergebnis haben und zusammen mit einer ungünstigen Wahl des Lernalgorithmus leicht zu Unter- oder Überparametrisierung führen. Die eingesetzten Suchalgorithmen schränken zwecks praktikabler Rechenzeit mit deterministischen Regeln oder zufallsbedingt die Menge betrachteter möglicher Netze ein, was ein weiterer Einflussfaktor auf das Lernergebnis ist. Zudem wird bei lokalen Algorithmen nur nach lokalen Optima gesucht. Unvollständige Daten und approximative Verfahren führen noch häufiger zu Fehlern. D.h. das maschinelle Lernen garantiert nicht, dass das gelernte Netz tatsächlich die realen Zustände korrekt abbildet und eine Gütebestimmung für gelernte Bayes'sche Netze ist daher äußerst ratsam. Das maschinelle Lernen stellt damit einen viel komplexeren und fehleranfälligeren Bereich in der Informatik dar, als die Entwicklung eines korrekt funktionierenden Parsers. Aus diesem Grund werden im Folgenden vorrangig Verfahren zur Gütebestimmung von Lernverfahren und gelernten Netzen diskutiert und für die Prüfung von Parser-Software sei auf die einschlägige Fachliteratur verwiesen.
Neben einer fehlerhaften Netzstruktur oder Wahl der Parameter, können auch ungünstige oder fehlerhafte Evidenzen zu inkonsistenten Ergebnissen führen. Deshalb muss das Bayes'sche Netz auch in Kombination mit den möglichen Evidenzeingaben geprüft werden. Die Feststellung und Vermeidung von Evidenz-Konflikten wird in Abschnitt 5.4.3 beschrieben. Sie kann als automatische Prüfung in das wissens-

basierte System integriert und bei jeder Evidenzeingabe durchgeführt werden.

5.4.1 Gütebestimmung für Generierungsverfahren

Eine Gütebestimmung für Generierungsverfahren kann als Vergleich zwischen einem bekannten, korrekten Bayes'schen Netz $\mathcal{B}^k = (S^k = X, E^k, \Theta^k)$ und einem generierten Netz $\mathcal{B}^g = (S^g = (X, E^g), \Theta^g)$ realisiert werden. Die Daten für die Generierung von $\mathcal{B}^g$ werden direkt aus $\mathcal{B}^k$ erzeugt, d.h. aus $\mathcal{B}^k$ wird eine formale Spezifikation oder Lerndaten gewonnen.[8] Die Variablen $\mathbf{X}$ sind für beide Netze gleich. Da es zu einer Wahrscheinlichkeitsverteilung mehrere Bayes'sche Netze geben kann (s. Kap. 5.1.3), macht ein Vergleich der Netzstrukturen S^k und S^g keinen Sinn. Stattdessen ist es erforderlich, die Parameter Θ^k und Θ^g bzw. die JPDs $\mathcal{P}^k(\mathbf{X})$ und $\mathcal{P}^g(\mathbf{X})$ auf Übereinstimmung zu prüfen. Als Vergleichskriterium können verschiedene Distanzmaße D verwendet werden. Je größer jeweils die Distanz $D(\mathcal{P}^k, \mathcal{P}^g)$ ist, umso stärker unterscheiden sich die Wahrscheinlichkeitsverteilungen $\mathcal{P}^k$ und $\mathcal{P}^g$ voneinander. Mittels eines Distanzmaßes kann die Güte verschiedener Generierungsverfahren bestimmt und miteinander verglichen werden. Bei der großen Anzahl von möglichen Lernverfahren ist das ein geeignetes Auswahlkriterium.
In [Jensen und Nielsen 2007] wird die *euklidische Distanz* herangezogen:

$$D_E(\mathcal{P}^k, \mathcal{P}^g) = \sum_{i=1}^{n} (P^k(X_i) - P^g(X_i))^2$$

Für die euklidische Distanz gilt $D(\mathcal{P}^k, \mathcal{P}^g) = 0 \iff \mathcal{P}^k = \mathcal{P}^g$.
[Heckerman, Geiger und Chickering 1994] verwendet stattdessen die *Kullback-Leibler-Distanz* bzw. *Kreuz-Entropie* (Kullback und Leibler 1951):

$$D_K(\mathcal{P}^k, \mathcal{P}^g) = \sum_{i=1}^{n} P^k(X_i) \cdot log \frac{P^k(X_i)}{P^g(X_i)}$$

mit $0 \cdot log(0/P^g(X_i)) = 0$ und $P^k(X_i) \cdot log(P^k(X_i)/0) = \infty$. Während die euklidische Distanz symmetrisch ist, d.h. $D_E(\mathcal{P}^k, \mathcal{P}^g) = D_E(\mathcal{P}^g, \mathcal{P}^k)$, trifft das auf die Kullback-Leibler-Distanz allerdings nicht zu. Neben dem Vergleich zwischen zwei JPDs kann die Kullback-Leibler-Distanz auch für den Test auf Abhängigkeit zwischen zwei Zufallsvariablen verwendet werden:

$$I(X, Y) = \sum_{i=1}^{n} \sum_{j=1}^{m} \mathcal{P}(X_i, Y_j) \cdot log \frac{\mathcal{P}(X_i, Y_j)}{P(X_i) \cdot P(Y_j)}$$

[8] Die Datenerzeugung muss fehlerfrei sein.

Diese *Mutual Information* beschreibt den gegenseitigen Informationsgehalt zweier Variablen X und Y und ist genau dann 0, wenn die zwei Variablen voneinander unabhängig sind (Cover und Thomas 1991).

5.4.2 Gütebestimmung für Bayes'sche Netze

Ist die erwartete Verbundwahrscheinlichkeitsverteilung bekannt, kann wie im vorherigen Abschnitt die Güte eines Bayes'schen Netzes ermittelt werden. Im Normalfall ist die Verbundwahrscheinlichkeitsverteilung aber eine unbekannte Größe. Ein einfacher Vergleich der JPDs ist daher nur zum Test eines neuen Generierungsverfahrens sinnvoll und nicht zur Gütebestimmung von gelernten Netzen im Allgemeinen.

5.4.2.1 Bootstrapping-Verfahren

Eine praktikable Möglichkeit ist die, bereits in Abschnitt 5.3.2.6 genannte, Erzeugung neuer Daten aus den gegebenen Daten mittels zufälligen Ziehens mit Zurücklegen. Dieser Vorgang kann mehrmals wiederholt werden. Die neuen Datenmengen stellen dann verrauschte Kopien der Ursprungsdaten dar und können zum Testen des gelernten Netzes verwendet werden. Dieses Verfahren wird auch als *Bootstrapping-Verfahren* bezeichnet (Efron und Tibshirani 1993). Ist die Datenmenge ausreichend groß, reicht es evtl. aus, sie lediglich fest oder zufällig in Lern- und Testdatenmengen zu zerlegen.

5.4.2.2 Kreuz-Validierung

Eine weitere Methode zur Erzeugung von Testdaten ist die k-fache Kreuz-Validierung (s. z.B. Bishop 2006, Kap. 1.3). Die gegebenen Daten werden in k gleichgroße Teilmengen zerlegt, von denen $k - 1k$ die Lerndaten und eine die Testdaten bildet. Auf diese Weise werden k Zerlegungen erstellt, wobei die Teilmenge zum Lernen jeweils eine andere ist. Für jede Zerlegung wird ein Bayes'sches Netz gelernt und getestet. Das beste Netz kann dann unter Anwendung von Gütekriterien (siehe unten) ausgewählt werden. Der Nachteil der k-fachen Kreuzvalidierung ist, dass mehrere Lern- und Testdurchläufe erforderlich sind.

5.4.2.3 Test im produktiven Einsatz

Die Korrektheit eines Bayes'schen Netzes kann auch im produktiven Einsatz geprüft werden. Die Testdaten sind dann reale Daten aus dem Anwendungsbereich, für die die erwartenden Ergebnisse, d.h. die Ausgaben des Bayes'schen Netzes, bereits bekannt sind.

5.4.2.4 Gütekriterien

In beiden Fällen kann der Vergleich zwischen tatsächlicher Ausgabe und Erwartungswert automatisiert erfolgen. Zur Bestimmung der Güte können statistische Gütekriterien, z.B. die unten angegebenen, verwendet werden. Die Güte kann für jede Variable des Bayes'schen Netzes bestimmt werden, soweit die Erwartungswerte bekannt sind. Bei Klassifikationsaufgaben sind bspw. die Ausgaben der Klassenvariablen zu prüfen.

Richtig-Positiv- und Falsch-Negativ-Rate Die *Richtig-Positiv-Rate* (auch *Sensitivität*) bezeichnet den prozentualen Anteil der richtig klassifizierten positiven Fälle bezogen auf die Gesamtanzahl von tatsächlich positiven Fällen:

$$P(\textit{positiv klassifiziert}|\textit{tatsächlich positiv}) = \frac{N(\textit{richtig positiv klassifiziert})}{N(\textit{tatsächlich positiv})}$$

Die *Falsch-Negativ-Rate* ist das Gegenteil der Richtig-Positiv-Rate. Sie bezeichnet den prozentualen Anteil der fälschlicherweise negativ klassifizierten positiven Fälle bezogen auf die Gesamtanzahl von tatsächlich positiven Fällen:

$$P(\textit{negativ klassifiziert}|\textit{tatsächlich positiv}) = \frac{N(\textit{falsch negativ klassifiziert})}{N(\textit{tatsächlich positiv})}$$

Richtig-Positiv- und Falsch-Negativ-Rate summieren sich zu 1.

Richtig-Negativ- und Falsch-Positiv-Rate Die *Richtig-Negativ-Rate* (auch *Spezifität*) bezeichnet den prozentualen Anteil der richtig erkannten negativen Fälle bezogen auf die Gesamtanzahl von tatsächlich negativen Fällen:

$$P(\textit{negativ klassifiziert}|\textit{tatsächlich negativ}) = \frac{N(\textit{richtig negativ klassifiziert})}{N(\textit{tatsächlich negativ})}$$

Die *Falsch-Positiv-Rate* (auch *Spezifität*) bezeichnet den prozentualen Anteil der fälschlicherweise positiv klassifizierten negativen Fälle bezogen auf die Gesamtanzahl von tatsächlich negativen Fällen:

$$P(\textit{positiv klassifiziert}|\textit{tatsächlich negativ}) = \frac{N(\textit{falsch positiv klassifiziert})}{N(\textit{tatsächlich negativ})}$$

Richtig-Negativ- und Falsch-Positiv-Rate summieren sich zu 1.

Korrekt- und Falschklassifikationsrate Die *Korrektklassifikationsrate* bezeichnet den prozentualen Anteil der richtig klassifizierten Fälle bezogen auf die Gesamtanzahl von Fällen:

$$P(\textit{richtig klassifiziert}) = \frac{N(\textit{richtig klassifiziert})}{N(\textit{Fälle})}$$

Die *Falschklassifikationsrate* bezeichnet den prozentualen Anteil der falsch klassifizierten Fälle bezogen auf die Gesamtanzahl von Fällen:

$$P(\textit{falsch klassifiziert}) = \frac{N(\textit{falsch klassifiziert})}{N(\textit{Fälle})}$$

Korrekt- und Falschklassifikationsrate summieren sich zu 1.

5.4.3 Feststellung und Vermeidung von Konflikten

Bei dem *Bayes-Beispiel 1b* (s. Abschnitt 5.2.1.1) wurden zur Erläuterung der Marginalisierung bewusst CPTs verwendet, in denen kein Eintrag exakt 0 ist. Werden stattdessen die ursprünglichen CPTs von *Bayes-Beispiel 1* (s. Abschnitt 5.1.4.1) verwendet, kann für die Evidenzen $Farbe = anders$ und $Hoehe = h2$ keine korrekte Zustandsverteilung für *Zeichen* berechnet werden, wie die folgenden zwei Gleichungen aufzeigen.

$$\begin{aligned}
&P(Z = wahr|F = anders, H = h2)\\
&\quad = P(Z = wahr) \cdot P(F = anders|Z = wahr) \cdot P(H = h2|Z = wahr)\\
&\quad = 0,6 \cdot 0,0 \cdot 0,5\\
&\quad = 0,0
\end{aligned}$$

und

$$\begin{aligned}
&P(Z = falsch|F = anders, H = h2)\\
&\quad = P(Z = falsch) \cdot P(F = anders|Z = falsch) \cdot P(H = h2|Z = falsch)\\
&\quad = 0,4 \cdot 1,0 \cdot 0,0\\
&\quad = 0,0
\end{aligned}$$

Beide Ergebnisse sind 0. Daraus folgt, dass die Wahrscheinlichkeit sowohl für *Zeichen=wahr* als auch *Zeichen=falsch* 0 ist. Das widerspricht der Axiomatik der bayes'schen Wahrscheinlichkeitsrechnung.

5.4.3.1 Konfliktursachen

Die Verwendung von exakt 0 in CPTs bedeutet, dass das Auftreten der jeweiligen Zustandskombination unmöglich ist, also niemals vorkommt. Wenn solche Zustandskombinationen auch in der Realität niemals vorkommen, kann das Netz bedenkenlos verwendet werden. Andernfalls bildet das Bayes'sche Netz die realen Bedingungen nicht korrekt ab und ist somit fehlerhaft entworfen worden. In Konflikt stehende Evidenzen entstehen auch durch fehlerhafte Evidenzen. Das kann in realen Anwendungen häufig durch ungenaue Messungen passieren (Korb und Nicholson 2004, Kap. 3.7.2).
Abhilfe schafft offensichtlich die Vermeidung von Nulleinträgen (vgl. Jensen und Nielsen 2007, Kap. 8.1) wie *Bayes-Beispiel 1b* zeigt. CPTs, die Nulleinträge enthalten, werden auch als *spärlich* bezeichnet und CPTs, in denen jeder Wert >0 ist, als *dicht* (vgl. Bilmes 2004). In dichten CPTs ist jede Zustandskombination zu einer bestimmten Wahrscheinlichkeit >0 möglich.

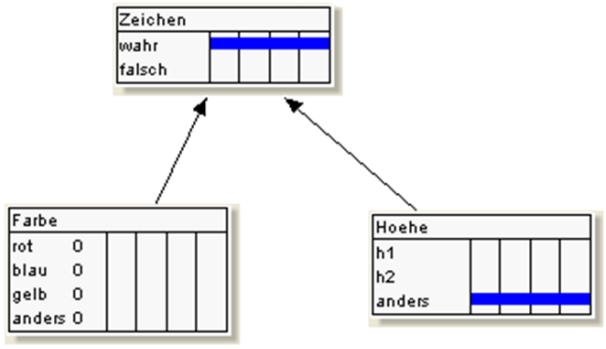


Abbildung 5.11: *Bayes-Beispiel 2* mit der konfliktauslösenden Evidenzmenge{`Zeichen`=*wahr*, `Hoehe`=*anders*}

Die Vermeidung von Nulleinträgen verhindert allerdings nur die mathematische Inkonsistenz und löst nicht den Konflikt, wie im folgenden Abschnitt auch gezeigt wird. Zudem kann es in einer Anwendung durchaus gerechtfertigt sein für bestimmte Zustandskombinationen die Unmöglichkeit anzunehmen. Durch fehlerhafte Verwendung eines Bayes'schen Netzes lassen sich oft Inkonsistenzen erzwingen, da der Sinn eines Bayes'schen Netzes die effiziente Abbildung von relevanten Informationen mit spärlichen CPTs ist (vgl. 5.1.3.3). Die in *Bayes-Beispiel 2* verwendete Evidenzmenge {`Zeichen`=*wahr*, `Hoehe`=*anders*} (s. Abschnitt 5.1.4.2 und Abb. 5.11) ist nicht realistisch, da bei einer korrekten Verwendung des Netzes davon aus-

gegangen werden kann, dass die Variable `Zeichen` keine Eingabevariable ist.

5.4.3.2 Konfliktmaß

Als erstes ist festzustellen, ob für die gegebenen Evidenzen ein Konflikt vorliegt. In [Jensen u. a. 1991] werden Vorgehensweisen dafür aufgezeigt. Der einfachste Fall ist für eine Posteriori-Wahrscheinlichkeit 0 gegeben. Dann stehen, wie im *Bayes-Beispiel 1*, die gegebenen Evidenzen im Konflikt. Nicht immer ist ein Konflikt aber so leicht zu erkennen.
Für die Feststellung von Konflikten können Zustände oder auch Variablen eingeführt werden, die das Unbekannte zusammenfassen (Jensen u. a. 1991). Eine hohe Wahrscheinlichkeit dafür weist auf einen Konflikt hin. Im Beispiel wurde bei den Merkmalsvariablen mit dem Zustand *anders* bereits die Vollständigkeit der Zustandsmengen sichergestellt. Um Konflikte aufzuzeigen, kann auch der Variablen `Zeichen` ein Zustand *anders* hinzugefügt werden. Allerdings ist diese Vorgehensweise irreführend, da mit den Zuständen *wahr* und *falsch* die Zustandsmenge aus logischer Sicht bereits vollständig ist. Damit ist es auch problematisch korrekte konditionale Wahrscheinlichkeiten für den Zustand *anders* zu finden. Möglich wäre hier, bewusst in Konflikt stehende Evidenzen in das Netz einzugeben und die konditionalen Wahrscheinlichkeiten auf diese Weise zu konfigurieren. Diese Vorgehensweise wurde bei dem medizinischen Diagnosesystem *MUNIN* (Olesen u. a. 1989) erfolgreich durchgeführt. Ein einzelnes *anders* gibt jedoch nur lokal für eine Variable eine Konfliktaussage. Um einen Konflikt global zu erkennen, müssen die lokalen Aussagen bis zu den Ausgabevariablen propagiert werden. Das vergrößert die CPTs immens und führt zu aufwendigeren Inferenzberechnungen. Diese Möglichkeit ist damit nicht praktikabel und [Jensen und Nielsen 2007] schlagen deshalb die folgende Formel als *Konfliktmaß* vor:

$$conf(e_1, ..., e_m) = log_2 \frac{P(e_1) \cdot \cdots P(e_m)}{\mathcal{P}(e_1, ..., e_m)}$$

Ist $conf > 0$, so ist das ein Indikator für einen möglichen Konflikt. Das Konfliktmaß basiert auf der Annahme, dass bei konfliktfreien Evidenzen die Wahrscheinlichkeit von $\mathcal{P}(e_1, ..., e_m)$ größer ist als das Produkt der Wahrscheinlichkeiten der einzelnen als unabhängig angenommenen Evidenzen. $conf > 0$ zeigt damit eine Diskrepanz zwischen dem Modell und den einzelnen Evidenzen an (Jensen und Nielsen 2007, Kap. 3.4.3).

Bei $P(e) = 0$, wie im *Bayes-Beispiel 1* mit den Evidenzen `Farbe`=*anders*, `Hoehe`=*h2* und im *Bayes-Beispiel 2* mit den Evidenzen `Zeichen`=*wahr*, `Hoehe`=*anders*, ist $conf = \infty$ und zeigt damit eindeutig einen Konflikt an. Auch *Bayes-Beispiel 1b* weist mit den gegebenen Evidenzen einen möglichen Konflikt auf:

$$\begin{aligned} conf(F = a, H = h2) &= log_2 \frac{P(F = a) \cdot P(H = h2)}{P(F = a, H = h2)} \\ &= log_2 \frac{0,3994 \cdot 0,3001}{0,0006985} \\ &\approx 7,4 \end{aligned}$$

Die Zwischenschritte zur Berechnung des Konfliktmaßes für *Bayes-Beispiel 1* und *Bayes-Beispiel 1b* sind in Anhang B aufgeführt.
Derart niedrige Posteriori-Wahrscheinlichkeiten, wie sie im letzten Beispiel auftreten ($P(Z = w, F = a, H = h2) = 0.0002997$ und $P(Z = f, F = a, H = h2) = 0.0003988$), können auch eine seltene Zustandskombination bedeuten. D.h. ob es sich bei $conf > 0$ tatsächlich um einen Konflikt handelt, kann nicht festgestellt werden. Es kann allerdings der Ursprung des Konflikts im Bayes'schen Netz nachverfolgt und lokalisiert werden. Dafür wird eine Evidenzmenge in Teilmengen zerlegt, die jeweils genau die direkten Kindknoten eines Elternknotens betreffen (vgl. Jensen und Nielsen 2007, Kap. 5.5.4), und für jede dieser Teilmengen das Konfliktmaß bestimmt.

5.5 Effizienz

Neben der Korrektheit ist auch die Effizienz eines Bayes'schen Netzes entscheidend für den produktiven Einsatz. Eine Möglichkeit, die Geschwindigkeit der Inferenzberechnungen zu erhöhen, ist die Verwendung von approximativen Inferenzverfahren (s. Abschnitt 5.2.2). Die Effizienzsteigerung geschieht hier allerdings zu Lasten der Rechengenauigkeit. Da sich die Netzstruktur direkt auf den Zeit- und Speicherverbrauch der Inferenzberechnungen auswirkt, sollte bereits bei der Generierung darauf geachtet werden, eine effiziente Struktur zu erstellen. Dann kann oftmals auch ein exaktes Inferenzverfahren effizient verwendet werden.
Ein erstes Kriterium für die Effizienz ist die Größe der CPTs, d.h. die Anzahl der CPT-Einträge. Für eine CPT $P(X_i|Pa(X_i))$ berechnet sich die Größe G aus dem Produkt der Zustandsanzahl der Variablen X_i und dem Produkt der Zustandsanzahlen

der Elternknoten (s. Formel 5.15). Damit steigt die CPT-Größe exponentiell mit der Anzahl der Elternknoten und den Zuständen aller beteiligten Variablen.

$$G(P(X_i|Pa(X_i))) = |x_i| \cdot \prod_j |Pa(X_i)_j| \tag{5.15}$$

Aus diesem Grund ist ein vollständig verknüpftes Netz sehr ineffizient. Z.B. ist die Komplexität der CPT-Einträge eines vollständig verknüpften Netzes, bestehend aus n binären Variablen, $O(2^n)$. In den meisten Fällen reicht ein weniger stark verknüpftes Netz aus, um den Problembereich abzubilden, und das Optimum ist, wenn möglich, eine Perfect-Map (s. Kap. 5.1.3). Die Komplexität verringert sich hier auf $O(n \cdot 2^k)$, mit k der Maximalanzahl der Elternknoten, und steigt bei $k \ll n$ nur noch linear mit der Anzahl der Variablen. Folglich ist ein schwach verknüpftes Netz mit einer größeren Anzahl an Variablen stets einem sehr stark verknüpften Netz mit weniger Variablen vorzuziehen. In [Korb und Nicholson 2004, Kap. 3.5.2] ist eine Funktion zur Berechnung der Rechenkomplexität eines Junction Trees angegeben.
Da es zu einem Bayes'schen Netz aufgrund unterschiedlicher Triangulierungen verschiedene Junction Trees geben kann, kann durch die Berechnung der Rechenkomplexität die effizienteste Struktur gefunden werden. Um die Umwandlung in einen Junction Tree zu beschleunigen, ist bei der Netzgenerierung, soweit möglich, die Verwendung von Polytree-Strukturen sinnvoll. Zusätzlich können spärliche CPTs komprimiert werden (Korb und Nicholson 2004, Kap. 3.5.2).
Damit führt u.a. die Einhaltung der folgenden Kriterien zu effizienten Bayes'schen Netzen:

- Variablen mit einer geringen Anzahl Zustände; gegebenenfalls Zustände zusammenfassen
- kleine CPTs; direkte Abhängigkeiten durch hinzufügen von vermittelnden Variablen (s. Abschnitt 5.6) reduzieren
- effiziente Parametermodelle (s. Abschnitt 5.2.2)
- Polytree-Strukturen

Die Optimierungen können direkt bei der Erstellung oder nachträglich erfolgen. In [Borth und Hasseln 2002] wird z.B. ein regelbasiertes System verwendet, um bestimmte Netzeigenschaften zu prüfen und gegebenenfalls anzupassen.

5.6 Erklärbarkeit Bayes'scher Netze

Wie in Kap. 4.1 aufgeführt, gehört zu einem wissensbasierten System meist eine Erklärungskomponente, die die Ergebnisse für den Anwender geeignet aufbereitet und Informationen darüber liefert, wie das System zu den Ergebnissen gelangt ist. Die Erklärbarkeit der Ergebnisse ist ein wichtiger Aspekt bei der Verarbeitung von Informationen, da sie ein entscheidendes Kriterium für die Akzeptanz des Gesamtsystems ist (vgl. Wittig 2003, Kap. 1.2).
Erklärbarkeit in einem Bayes'schen Netz bedeutet im Wesentlichen, dass für die Variablen und ihre Zustände eine Semantik hinterlegt ist, die gegebenenfalls abgefragt werden kann. Mit ihrem graphischen Modell stellen Bayes'sche Netze eine explizite und leicht verständliche Wissensbeschreibung dar. Die Abbildung kausaler Beziehungen eignet sich besonders gut, um die Schlussfolgerungen transparent zu machen. Oft ist es auch für Nicht-Experten anhand des Graphenmodells relativ einfach kausale Zusammenhänge in einem Anwendungsbereich zu verstehen.
Über die Möglichkeiten der Erklärbarkeit Bayes'scher Netze gibt [Lacave und Díez 2002] einen ausführlichen Überblick. Im folgenden Abschnitt 5.6.1 werden u.a. daraus einige Ideen vorgestellt. In Abschnitt 5.6.2 wird eine für die Erklärbarkeit Bayes'scher Netze förderliche Netzstruktur vorgeschlagen und Abschnitt 5.6.3 diskutiert die Problematik der Erklärbarkeit gelernter Bayes'scher Netze.

5.6.1 Arten von Erklärbarkeit und ihre Darstellung

Generell können Erklärungen in einer textuellen oder einer grafischen Form erfolgen. Um verständliche Erklärungen zu liefern, sollte der Wissensstand des Benutzers berücksichtigt werden. Zudem kann die Erklärungskomponente so konzipiert sein, dass die Erklärungen nicht nur passiv angezeigt werden, sondern der Benutzer das wissensbasierte System interaktiv erforschen und in einer gewissen Form Fragen stellen kann.
Die Arten von Erklärungen können nach [Lacave und Díez 2002] in die folgenden drei Kategorien eingeteilt werden.

5.6.1.1 Abduktion

Die *Abduktion* geht auf den Mathematiker und Philosophen C. S. Peirce zurück und ist neben der Induktion und der Deduktion eine Art der Schlussfolgerung (Bei-

erle und Kern-Isberner 2006). Beim abduktiven Schließen wird anhand eines bekannten Ergebnisses auf die mögliche Ursache geschlossen. Für ein Bayes'sches Netz bedeutet das, die wahrscheinlichste Evidenzmenge zu finden, die zu dem in den Ausgabevariablen beobachteten Ergebnis führt. Die Abduktion zeigt damit eine Möglichkeit auf, wie ein Ergebnis zustande gekommen sein kann.
Die Erklärbarkeit der Abduktion liegt darin, dem Benutzer zu belegen, dass es für das beobachtete Ergebnis eine Ursache gibt und somit das wissensbasierte System eine realistische Aussage getroffen hat. Es können allerdings mehrere verschiedene Ursachen existieren und damit muss die eine abgeleitete Ursache nicht unbedingt der tatsächlichen entsprechen (vgl. Beierle und Kern-Isberner 2006, Kap. 3.1.2).

5.6.1.2 Modellerklärungen

Ein zentraler Punkt bei der Erklärung von Bayes'schen Netzen ist die Erläuterung des zugrundeliegenden Modells bzw. der damit abgebildeten Wissensbasis. Da Bayes'sche Netze mit ihrer graphischen Struktur bereits anschaulich die Abhängig- und Unabhängigkeitsbeziehungen darstellen, kann direkt dieses Modell grafisch angezeigt werden. Das setzt beim Benutzer allerdings ein Grundverständnis über Bayes'sche Netze voraus und eignet sich nur für kleinere, überschaubare Netze. Geeigneter für die meisten Anwendungen ist daher eine textuelle Erklärung der Beziehungen.
Um Erklärungen dynamisch zur Laufzeit generieren zu können, ist es sinnvoll die Variablen zuvor nach Art zu kategorisieren und die Erklärungen aus einzelnen Mustern zusammenzusetzen. Daneben lässt sich in einem Bayes'schen Netz leicht auf die Eltern- und Kindknoten einer Variablen zugreifen und somit die direkte Abhängigkeitsbeziehung herleiten. Bspw. stellen in einem kausalen Netz die Elternknoten Ursachen dar.
Auch die Art der Beziehung kann in vier Kategorien unterschieden werden: positiv, negativ, Nullverbindung und unbekannt (Lacave und Díez 2002). Auch für diese Kategorien können Mustererklärungen bereitgestellt werden. Für diskrete ordinale Variablen A und C mit $A \in Pa(C)$ bedeutet eine positive Beziehung, dass, unabhängig von den Zuständen anderer Elternknoten $Pa(C)\backslash A$, höhere Zustandswerte von C auch zu höheren Zustandswerten bei A führen. Eine negative Verbindung ist analog definiert und eine Nullverbindung bedeutet die Unabhängigkeit zwischen A und C für jede Zustandskombination von $Pa(C)\backslash A$. Trifft keine der ersten drei Kategorien

zu, ist die Art der Beziehung unbekannt.

5.6.1.3 Erklärung der Schlussfolgerungen

Mit dieser Art der Erklärung ist die Erläuterung der Ergebniswahrscheinlichkeiten gemeint. Normalerweise ist dabei jeweils eine Variable im Fokus der Erklärung. Die aktuelle Wahrscheinlichkeitsverteilung kann numerisch oder auch linguistisch angegeben werden. Wie Studien gezeigt haben, sind linguistische Formulierungen von Wahrscheinlichkeiten für Menschen meist verständlicher (s. Lacave und Díez 2002, Kap. 4.3.1). Ende der 1980er Jahre entwickelte C. Elsaesser einen Katalog, wie numerische Wahrscheinlichkeitswerte, z.B. im Intervall $[0.08, 0.01]$, auf linguistische Aussagen, z.B. „sehr unwahrscheinlich", abgebildet werden können. Darauf basierend können auch Aussagen über die Abhängigkeitsbeziehungen bis hin zur Einflussstärke gemacht werden. Ein Maß für die Stärke des Einflusses einer Variablen auf eine andere ist z.B. die Kullback-Leibler-Distanz (s. Abschnitt 5.4.1). Im Zusammenhang mit dieser Art der Erklärbarkeit kann auch die Sensitivität von Evidenzen ermittelt werden. Diese gibt Aufschluss darüber, welche Evidenzen maßgeblich eine Aussage beeinflussen und wird in [Jensen und Nielsen 2007] ausführlich beschrieben.
Ein noch größeres Verständnis für den Schlussfolgerungsprozess erhält der Benutzer durch die Möglichkeit der direkten Interaktion. Dafür bietet die Erklärungskomponente dem Benutzer an die Evidenzeingabe zu variieren und die resultierenden Änderungen der Ergebnisse zu beobachten.

5.6.2 Vermittelnde Variablen

Eine geeignete Strukturierung des Bayes'schen Netzes kann für die Erklärbarkeit förderlich sein. Es können sogenannte *vermittelnde Variablen* (engl. mediating variable) (Jensen und Nielsen 2007) zur Repräsentation von Zwischenergebnissen hinzugefügt werden. Im *Bayes-Beispiel 3* (Abb. 5.7) sind die Variablen `Position_Farbe_Zeichen` und `Farbe_Hoehe_Zeichen` für die Repräsentation der gewünschten Informationen nicht notwendig. Sie beinhalten aber Zwischenergebnisse der Klassifikation, die ebenso ablesbar sind wie das Gesamtergebnis, das die Variable `Zeichen` repräsentiert. Z.B. kann so das Ergebnis $P(Zeichen = falsch) = 1.0$ mit der semantischen Bedeutung „Die Region stellt sicher kein Schriftzeichen dar."

durch Abfrage der Ergebnisse der vermittelnden Variablen weiter detailliert werden. D.h. welche Merkmalskombination (*Position* und *Farbe* und/oder *Farbe* und *Höhe*) dazu führt, dass es sich um kein Schriftzeichen handelt.
Ein Zwischenschritt, der in einem Bayes'schen Netz nicht explizit durch eine Variable definiert ist, kann nicht separat zur Bildung einer Erklärung ausgewertet werden. Da die Verwendung von vermittelnden Variablen auch die Komplexität der CPTs und damit die Rechenzeit verringert (s. Abschnitt 5.5), stellen sie zudem ein sehr praktikables Konzept dar.

5.6.3 Erklärbarkeit gelernter Netze

Bei der manuellen Erstellung sowie auch bei der Generierung auf Basis einer formalen Wissensbeschreibung kann die endgültige Struktur des Netzes stark beeinflusst werden. Bspw. können vermittelnde Variablen zur Repräsentation von Zwischenergebnissen hinzugefügt, oder eine intuitiv verständliche kausale Struktur aufgebaut werden. Die nachvollziehbare Struktur geht jedoch meist verloren, wenn Bayes'sche Netze gelernt werden. Beim Lernen bekommt das Netz in der Regel nur die Variablen, die explizit in den Daten enthalten sind; vermittelnde Variablen werden nicht automatisch hinzugefügt. Die Beziehungen sind dann sehr kompakt in den JPDs definiert und für detaillierte Erklärungen nicht mehr zugänglich.
Bereits 1992 wurden in [Cooper und Herskovits 1992] Möglichkeiten vorgestellt während des Lernprozesses vermittelnde Variablen hinzuzufügen. In [Wittig 2003] wurden verschiedene Lernalgorithmen genau unter dem Aspekt der Erklärbarkeit der entstehenden Netzstruktur untersucht und verbessert. Das Lernen von Bayes'schen Netzen und ihre Erklärbarkeit lassen sich also vereinbaren.

5.7 Einordnung und Erweiterungen

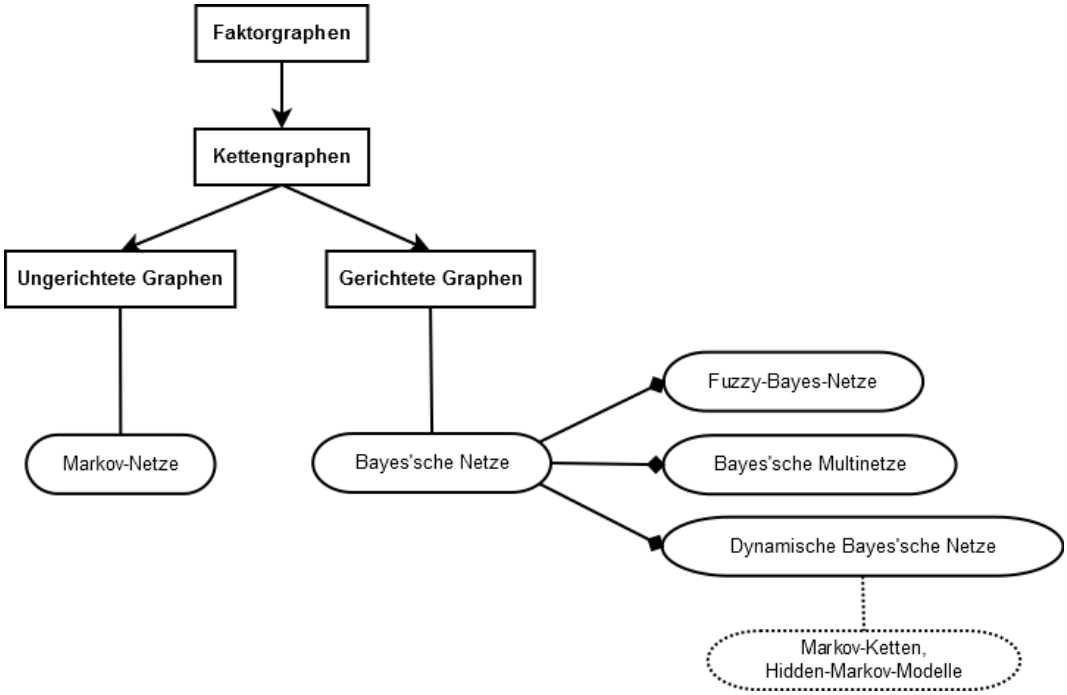


Abbildung 5.12: Übersicht einiger graphischer Modelle. Als Rechtecke dargestellt sind die allgemeinen Graphengrundmodelle, Elternknoten besitzen jeweils eine höhere Universalität in der Abbildung von Informationen. Die ovalen Elemente repräsentieren Unterklassen des jeweiligen graphischen Grundmodells. Linien mit einem Rauten-Ende weisen auf Erweiterungen einer Unterklasse. Fein gestrichelte Linien zwischen einem ovalen Element X und einem gestrichelt ovalen Element Y bezeichnen eine *Y kann mit X implementiert werden*-Beziehung.

Allgemeine Vorteile probabilistischer Graphenmodelle sind, dass die Information sehr anschaulich dargestellt wird und Inferenz- sowie Lernverfahren auf relativ einfache Weise mittels graphischer Manipulationen durchgeführt werden können. In Abb. 5.12 sind einige Graphenmodelle, geordnet nach ihrer Universalität Informationen abzubilden, aufgeführt. Ein Bayes'sches Netz ist offensichtlich nicht das universellste Graphenmodell. Allerdings kann ein Bayes'sches Netz bis hin zur *Perfect Map* (Kap. 5.1.3) eine sehr kompakte Beschreibung einer Wahrscheinlichkeitsverteilung liefern. Insbesondere konditionale Ab- und Unabhängigkeiten zwischen Variablen lassen sich direkt formulieren und führen zu effizienten Berechnungen. Je

komplexer die Beziehungen zwischen den Variablen jedoch sind, umso mehr nähert sich ein Bayes'sches Netz an einen vollständig verknüpften Graphen an. Auch derart komplexe Informationen können in einem Bayes'schen Netz repräsentiert werden, allerdings weniger effizient. Ein Bayes'sches Netz ist dann evtl. nicht mehr das geeignetste Modell.

Mit ihrer Darstellbarkeit als DAG sind Bayes'sche Netze ein Spezialfall von den *ungerichteten Graphen* (UG, engl. undirected graph), zu denen die *Markov-Netze* (engl. *Markov Random Fields*) gehören (Bishop 2006). Bayes'sche Netze bilden kausale Variablenbeziehungen mittels lokaler konditionaler Wahrscheinlichkeiten ab, Markov-Netze stellen weiche Variablenbedingungen in Form einer Wahrscheinlichkeitsverteilung dar. Die durch ein Bayes'sches Netz repräsentierte Wahrscheinlichkeitsverteilung kann auch in einem UG dargestellt werden, da ein vollständig verbundener UG stets die Information eines DAG abbilden kann[9]. In so einem Fall sind dann viele Kanten überflüssig. Besser ist es nur so viele Kanten hinzuzufügen, wie maximal notwendig sind. Genau das geschieht bei der in Kap. 5.2 beschriebenen Moralisierung. Im optimalen Fall entspricht der erzeugte UG einer *Perfect Map*. Der Zusammenhang zwischen einem *gerichteten Graph* (DG, engl. directed graph)), von dem DAG ein Spezialfall ist, UG und Perfect Map ist, analog zu der Erläuterung in [Bishop 2006, Kap. 8.3.4], in Abb. 5.13 dargestellt. Aus der Men-

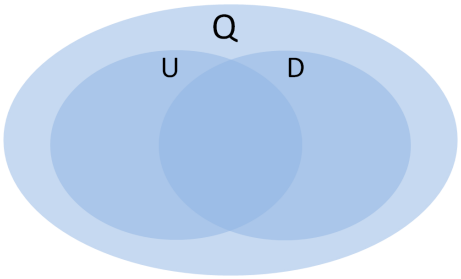


Abbildung 5.13: Beziehung zwischen der Menge $\mathcal{Q}$ aller Wahrscheinlichkeitsverteilungen und ihren Untermengen $\mathcal{U}$ und $\mathcal{D}$, deren Wahrscheinlichkeitsverteilungen in Form eines UG und/oder DG als Perfect Map beschrieben werden können.

ge aller möglichen Wahrscheinlichkeitsverteilungen $\mathcal{Q}$ gibt es nur eine Teilmenge $\mathcal{D}$, die von DGs exakt als Perfect Map abgebildet werden kann. Die weitere Teilmenge $\mathcal{U}$ beinhaltet alle Wahrscheinlichkeitsverteilungen, für die UGs als Perfect

[9]Ein vollständig verbundener UG kann jede beliebige Wahrscheinlichkeitsverteilung beschreiben. Konditionale Unabhängigkeiten sind dann nur implizit enthalten (vgl. Bishop 2006, Kap. 8.2).

Map existieren. Für die Wahrscheinlichkeitsverteilungen $\mathcal{D} \cap \mathcal{U}$ gibt es sowohl DGs als auch UGs, die eine Perfect Map sind. Daneben gibt es auch Wahrscheinlichkeitsverteilungen, für die weder ein DG noch ein UG eine Perfect Map darstellen können. Damit sind sowohl Bayes'sche als auch Markov-Netze nicht in der Lage, bestimmte Wahrscheinlichkeitsverteilungen optimal zu beschreiben. $\mathcal{D} \not\equiv \mathcal{U}$ erklärt außerdem, dass je nach Anwendungsfall ein DG oder ein UG das geeignetere Graphenmodell ist. So gehen z.B. d-Separierungseigenschaften eines Bayes'schen Netzes verloren, wenn es in ein Markov-Netz umgewandelt wird. Außerdem können in einem Markov-Netz bedingte Unabhängigkeiten nicht mehr explizit durch das Fehlen von Kanten repräsentiert werden.

Wie oben erläutert können Bayes'sche Netze viele Informationen sehr effizient repräsentieren, besitzen aber auch Grenzen. In Kap. 5.2.3 wurden die unsicheren Evidenzen vorgestellt, die die Ausdrucksmächtigkeit Bayes'scher Netze vergrößern. Eine andere Verallgemeinerung der Bayes'schen Netze ist die Erweiterung zu *Bayes'schen Multinetzen* (Geiger und Heckerman 1996). In einem Bayes'schen Netz ist die Abhängigkeit zwischen zwei Variablen A und B mit $P(A|B)$ fest definiert. Bei bestimmten Zuständen von B kann es jedoch notwendig sein, A als unabhängig von B anzusehen. Das ist in einem Bayes'schen Netz nur suboptimal abbildbar, da für die Definition einer teilweisen Unabhängigkeit eine CPT sehr viele Nulleinträge enthalten muss. Ein Bayes'sches Multinetz hingegen ermöglicht eine effiziente Definition dieser asymmetrischen Unabhängigkeitsbeziehungen. Es besteht aus mehreren Bayes'schen Netzen, wobei jeweils nur bestimmte Zustandskombinationen betrachtet werden. Alle Bayes'schen Netze umfassen dieselben Variablen. In [Geiger und Heckerman 1996] wird daher mit den *Ähnlichkeitsnetzen* (engl. Similarity Networks) ein performanteres Konzept vorgestellt. Die einzelnen Netze von einem Ähnlichkeitsnetz müssen nicht dieselben Variablen beinhalten, sondern lediglich diejenigen, die notwendig sind, um die Informationen korrekt abzubilden. Ähnlichkeitsnetze definieren damit gegenüber einem äquivalenten Multinetz weniger konditionale Abhängigkeiten, was zu effizienteren Inferenzberechnungen und einem geringeren Speicherverbrauch führt. Zudem wird vermieden, dass bedingte Wahrscheinlichkeiten für Zustandskombinationen definiert werden müssen, die nicht realistisch sind. Bei den Netzen eines Ähnlichkeitsnetzes handelt es sich

um Hypergraphen[10], womit sich dieses Konzept von den einfachen DG-Modellen entfernt. In [Geiger und Heckerman 1996] werden noch weitere optimierte Konzepte vorgestellt und auch gezeigt, dass es möglich ist ein Ähnlichkeitsnetz in ein Bayes'sches Netz zu transferieren, um dieselben Inferenzalgorithmen anwenden zu können. Die Menge der abbildbaren Wahrscheinlichkeitsverteilungen ist stets identisch.
Daneben sind statt diskreten auch kontinuierliche Variablenzustände in einem Bayes'schen Netz möglich. Die diskreten Einträge der CPTs müssen in diesem Fall durch Dichtefunktionen, z.B. die Gauß-Verteilungen, ersetzt werden. Da im Rahmen dieser Arbeit nur Bayes'sche Netze mit diskreten Variablenzuständen betrachtet werden, sei für Details zum Umgang mit kontinuierlichen Variablenzuständen auf Fachliteratur, wie bspw. Jensen und Nielsen 2007 und Korb und Nicholson 2004 verwiesen. Bayes'sche Netze, die sowohl diskrete als auch kontinuierliche Variablen besitzen, werden als *Hybride Bayes'sche Netze* bezeichnet. Hierzu gehören die *Fuzzy-Bayes-Netze*, die durch Verwendung von kontinuierlichen Variablenzuständen mittels Gauß-Verteilungen Fuzzy-Mengen abbilden können und damit der Fuzzy-Logik bzw. dem unscharfen Schließen nahekommen.
Für die zukünftige Weiterentwicklung des Designverifikationsverfahrens (siehe Kap. 11.3) stellen vor allem die *Faktorgraphen* als Oberklasse von UG und DG sowie die *dynamischen Bayes'schen Netze* mit zusätzlicher Zeitdimension interessante Konzepte dar und werden aus diesem Grund in den folgenden Abschnitten ausführlicher vorgestellt.

5.7.1 Ketten- und Faktorgraphen

Eine Oberklasse von UGs und DGs sind die *Kettengraphen* (s. z.B. Lauritzen 1996), in denen sowohl ungerichtete als auch gerichtete Kanten enthalten sein können. Eine Äquivalenzklasse $[\mathcal{G}]$, die von Markov-äquivalenten Bayes'sche Netzen $\mathcal{B} \in [\mathcal{G}]$ (s. 5.1.3) gebildet wird, kann durch einen sogenannten *essentiellen Graphen* (engl. essential Graph) $\mathcal{G}*$ dargestellt werden, der ein Kettengraph ist (Flesch und Lucas 2007). Essentielle Graphen besitzen neben gewöhnlichen Kanten essentielle Kanten, d.h. die Richtung einer essentiellen Kante kann nicht umgekehrt werden, ohne dabei die Ab- bzw. Unabhängigkeitsbeziehungen zu verändern. Damit unterschei-

[10] In einem Hypergraphen kann eine Kante (Hyperkante) mehr als zwei Knoten gleichzeitig miteinander verbinden (Beierle und Kern-Isberner 2006, Anhang B.4).

den sich die Strukturen Bayes'scher Netze einer Äquivalenzklasse nur in ihren gewöhnlichen Kanten voneinander. Der essentielle Graph $\mathcal{G}*$ zu einer Äquivalenzklasse $[\mathcal{G}]$ besteht aus allen essentiellen Kanten sowie aus allen gewöhnlichen Kanten aller $\mathcal{B} \in [\mathcal{G}]$, die zu ungerichteten Kanten umgewandelt werden (Flesch und Lucas 2007).[11]

Die *Faktorgraphen* stellen mit einem bipartiten Graphenmodell einen noch vielseitigeren Formalismus dar. Wie in 5.2 als Marginalisierung vorgestellt, kann die von einem Bayes'schen Netz abgebildete JPD als Produkt von Faktoren dargestellt werden, mit $X_s \subseteq X$.

$$P(X) = \prod_s F_s(X_s) \tag{5.16}$$

Dasselbe gilt auch für Markov-Netze, wie z.B. [Bishop 2006, Kap. 8] detaillierter ausführt. Durch die Faktorisierung lassen sich komplexe globale Berechnungen in lokale Berechnungen zerlegen. Im Fall eines Bayes'schen Netzes sind $F_s(X_s)$ lokale konditionale Wahrscheinlichkeitsverteilungen[12]. Ein Faktorgraph macht die Faktorisierung explizit, indem die Faktoren durch zusätzliche Knoten repräsentiert werden. Zudem kann die Faktorisierung viel detaillierter und damit universeller beschrieben werden als in einem anderen Graphenmodell. F_s kann beliebig zerlegt werden, um den Einfluss verschiedener Faktoren getrennt darzustellen. Daraus folgt, dass zur Repräsentation eines Bayes'schen Netzes verschiedene Faktorgraphen möglich sind. Je nach Anwendungsbereich kann so eine geeignete Faktorisierung gewählt werden (Bishop 2006, Kap. 8.4.3).

Das in Abb. 5.14 dargestellte Beispiel ist [Bishop 2006] entnommen und zeigt ein Bayes'sches Netz sowie zwei mögliche Faktorgraphen. Die Faktorisierung des unteren linken Faktorgraphens ist $F1 = P(A)P(B)P(C|A,B)$ und beschreibt das Bayes'sche Netz vollständig. Im unteren rechten Faktorgraphen ist Faktor $F1$ weiter zerlegt in $F2 = P(A)$, $F3 = P(B)$ und $F4 = P(C|A,B)$. Wie ersichtlich ist, handelt es sich bei Faktorgraphen um ungerichtete Graphen, deren Knotenmenge bipartit ist bestehend aus Variablen und Faktoren. Im Beispiel haben die Faktorgraphen zudem eine Baumstruktur, d.h. zwischen zwei Knoten gibt es genau einen Pfad und der Graph ist zyklenfrei. Eine Baumstruktur ist kein zwingendes Kriterium für einen Faktorgraphen, kann aber aus einer geeigneten Faktorisierung resultieren. Wie be-

[11] Wenn der Kettengraph $\mathcal{G}*$ kein DAG ist, gilt $\mathcal{G}* \notin [\mathcal{G}]$ (Flesch und Lucas 2007).

[12] [Shafer und Shenoy 1990] erläutert, dass eine Faktorisierung essentiell für lokale Berechnungen ist, die Faktoren aber nicht zwingend konditionale Wahrscheinlichkeiten beschreiben müssen.

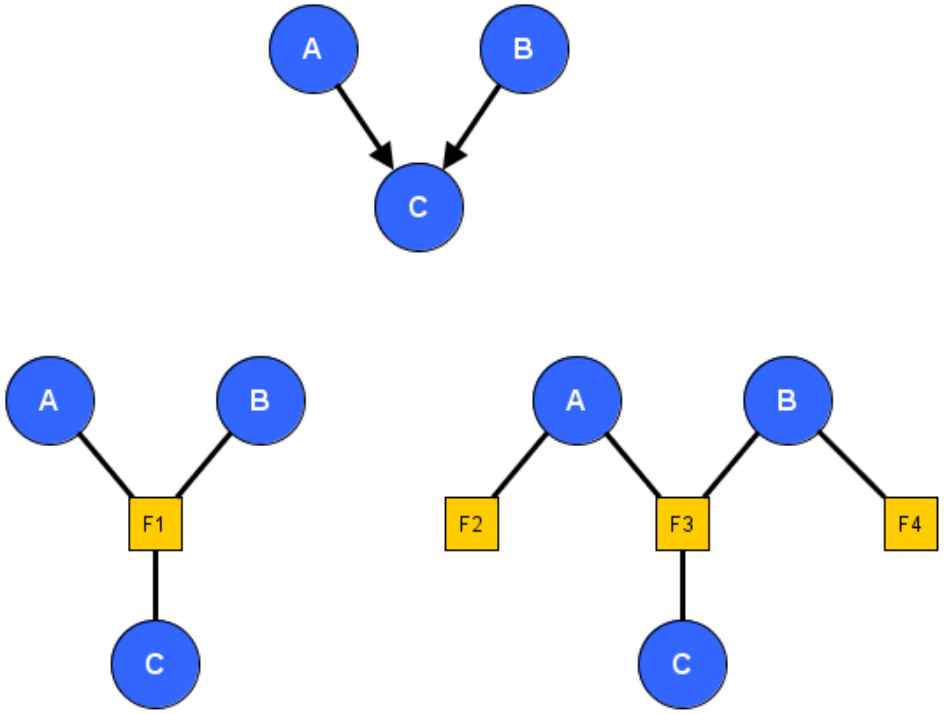


Abbildung 5.14: Oben ein Bayes'sches Netz bestehend aus den drei Variablen A, B und C. Unten zwei dazu passende Faktorgraphen mit den Faktoren $F1$ sowie $F2$, $F3$ und $F4$.

reits in Kap. 5.2 erläutert existieren vor allem für Baumstrukturen sehr effiziente exakte Inferenzverfahren. Bei einem Faktorgraphen in Baumstruktur können diese direkt angewendet werden. Daneben können mittels Integralprodukt auch für kontinuierlichen Variablen exakte effiziente Inferenzberechnungen durchgeführt werden. In einem Faktorgraphen ist stets eine geeignete Faktorisierung darstellbar, um ein effizientes graphisches Modell zu erhalten.

5.7.2 Dynamische Bayes'sche Netze

Graphische Modelle sind zunächst einmal statisch, d.h. die Zeitdimension ist nicht relevant. Es kann jedoch erforderlich sein, zu verschiedenen Zeitpunkten unterschiedliche Informationen zu verarbeiten bzw. die Informationsverarbeitung in einem zeitlichen Verlauf abzubilden. 1992 wurde in [Kjærulff 1992] das Konzept der dynamischen probabilistischen Netze vorgestellt, zu denen die *dynamischen Bayes'schen Netze* gehören.

Ein dynamisches Bayes'sches Netz setzt sich aus Wiederholungen eines statischen Bayes'schen Netzes $\mathcal{B}$ zusammen, die über die Zeit miteinander verknüpft sind. $\mathcal{B}$ bestehend aus den Variablen $X = X_1^t, ..., X_n^t$ bildet dabei den Zustand zu einem diskreten Zeitpunkt t ab und wird als *Zeitscheibe* (engl. time slice) bezeichnet. Aus dieser Definition folgt, dass jede Zeitscheibe dieselbe Struktur und CPTs aufweist, während die eingegebenen Informationen abhängig vom Zeitpunkt variieren (Jensen und Nielsen 2007, Kap. 3.3.7).
Die Eingabevariablen einer Zeitscheibe erhalten ihre Informationen aus Elternknoten vorhergehender Zeitscheiben und die Ausgabevariablen besitzen Kindknoten in nachfolgenden Zeitscheiben. Für dynamische Bayes'sche Netze gilt in der Regel die Markov-Voraussetzung, dass wenn eine Variable X_i zu einem Zeitpunkt t von einer Variablen X_j einer vorhergehenden Zeitscheibe abhängig ist, die Variable X_j nur in der unmittelbar vorhergehenden Zeitscheibe mit Zeitpunkt $t - 1$ liegen darf (d.h. das Netz hat kein Gedächtnis). Wenn die Markov-Voraussetzung gilt und pro Zeitscheibe nur eine Variable existiert, entspricht das dynamische Bayes'sche Netz einer Markovkette (Markov-Prozess erster Ordnung). Darüber hinaus kann eine Variable zusätzlich von anderen Variablen derselben Zeitscheibe abhängig sein, womit es sich dann um ein Hidden Markov Modell handelt (Markov-Prozess n-ter Ordnung, mit n maximale Anzahl an Variablenabhängigkeiten) (Korb und Nicholson 2004, Kap. 4.5; Pavlovic 1999). Abhängigkeiten zu m vorhergehenden Zeitpunkten $t - 1, ..., t - m$ sind möglich, gestalten die Inferenzberechnungen aber aufwendiger. Die Beziehungen zwischen Variablen verschiedener Zeitscheiben werden genauso wie Variablen innerhalb einer Zeitscheibe in CPTs definiert. Ein dynamisches Bayes'sches Netz kennt den bisherigen zeitlichen Verlauf der Information und kann Prognosen über den Zukünftigen geben. Damit ist die Richtung der Propagation stets von vergangenen zu zukünftigen Zeitscheiben und eine Umkehrung offensichtlich unsinnig. Die feste Kantenrichtung verallgemeinert den Formalismus der Bayes'schen Netze. Die Inferenzberechnungen können sehr ähnlich wie für ein statisches Bayes'sches Netz durchgeführt werden. In [Jensen und Nielsen 2007] wird in Kap. 4.6.2 die Triangulierung von dynamischen Bayes'schen Netzen für die Anwendung des Junction-Tree-Algorithmus beschrieben. Der Rechenaufwand und der Speicherverbrauch sind offensichtlich höher als bei statischen Netzen und erfordern effiziente Implementierungen. Neben der Abbildung eines zeitlichen Informationsflusses können durch die Verwendung dynamischer Bayes'scher Netze zyklische

Abhängigkeiten, die ein Bayes'sches Netz nicht darstellen kann, vermieden werden (Wittig 2003). Neben Inferenzberechnung ist auch das Lernen dynamischer Bayes'scher Netze möglich, wie bspw. in [Pavlovic 1999] erläutert wird.

5.7.3 Graphenmodelle in der Praxis

Vorgestellt wurden Faktorgraphen erstmals in [Kschischang, Frey und Loeliger 1998] und werden bis heute zusammen mit dem Summenprodukt-Algorithmus erfolgreich für die Dekodierung von Daten eingesetzt. [Beierle und Kern-Isberner 2006] erläutert sehr ausführlich den Zusammenhang zwischen Faktorgraphen, DGs und UGs und erklärt die Inferenzberechnung für Bayes'sche Netze in der verallgemeinerten Form des *Summenprodukt-Algorithmus* für Faktorgraphen. In [Bilmes 2004] finden Faktorgraphen eine kurze Erwähnung. Ansonsten wird ihnen in der Literatur über Bayes'sche Netze bislang kaum Beachtung geschenkt. Ein Grund dafür kann sein, dass in vielen Anwendungsbereichen Bayes'sche Netze bereits ein sehr geeignetes Modell für die Wissensbeschreibung sind. Faktorgraphen sind ein geglückter Versuch UGs und DGs in einem Formalismus zusammenzufassen und auf diese Weise eine höhere Universalität der Beschreibungsmöglichkeit zu erhalten. Allerdings erfordert die Verwendung der bipartiten Faktorgraphen auch eine andere Denkweise, um die Faktoren zu bestimmen und bringt aufwendigere Implementierungen mit sich.

Wenngleich die historischen Ursprünge von HMMs und Bayes'schen Netzen vollkommen verschieden sind - HMMs resultieren aus den Markov-Netzen, die unter theoretischen Aspekten der statistischen Physik entwickelt wurden um Wechselwirkungen zwischen Elementen bzw. Atomen abzubilden; Bayes'sche Netze basieren auf der Stochastik und Mathematik für die Abbildung subjektiver Wahrscheinlichkeiten - können beide Konzepte als dynamische Bayes'sche Netze aufgefasst werden. Aufgrund der gerichteten Graphenstruktur weist ein dynamisches Bayes'sches Netz gegenüber einem HMM explizite Rechenvorschriften auf und ermöglicht effizientere Inferenzverfahren. Auch ist ein höherer Grad an Interpretierbarkeit erreichbar. Die Verwendung von dynamischen Bayes'schen Netzen anstelle von HMMs kann also sehr vorteilhaft sein. In [Pavlovic 1999] wird ein auf dynamischen Bayes'schen Netzen basierendes Verfahren zur Auswertung von Sensordaten für Mensch-Maschine-Interaktionen, z.B. Spracherkennung und Gestikerkennung, vorgestellt. In diesem Zusammenhang wird unter dem Begriff *Mixed-State Dynamic*

Bayesian Networks eine Kombination von HMMs und Kalman Filtern vorgestellt, die als dynamisches Bayes'sches Netz umgesetzt ist. Daneben sind weitere Kombinationen wie z.B. dynamische Bayes'sche Multinetze (Bilmes 2000) realisierbar. So gesehen können dynamische Bayes'sche Netze in einem breiten Anwendungsumfeld eingesetzt werden.
Die Erweiterung bzw. Verallgemeinerung von graphischen Modellen scheint kaum Grenzen zu haben. Die Kombination einzelner Formalismen sind denkbar, vor allem aus theoretischer Sicht. In der Praxis bedeutet die Verwendung einiger umfangreicherer Formalismen, wie kontinuierliche Variablenzustände und dynamische Bayes'sche Netze, offensichtlich einen höheren Rechenaufwand und Speicherverbrauch. Auch die Validier- und Kontrollierbarkeit wird mit steigender Komplexität aufwendiger. Aus diesen Gründen sollte je nach Anwendungsbereich genau entschieden werden, welches minimale Modell ausreichend gute Ergebnisse liefern kann. Z.B. reicht in vielen Fällen, in denen die ursprünglichen Zustände kontinuierlicher Art sind, eine Diskretisierung dieser aus. Die Anforderung sehr komplexe Informationen möglichst exakt abzubilden und zu verarbeiten, kann im Einzelfall jedoch den Einsatz derart komplexer Konzepte rechtfertigen. Die Verallgemeinerung zu universelleren bzw. auch vereinfachten Modellen (z.B. Baumstrukturen) zeigt außerdem, dass solange man sich im Umfeld graphischer Modelle bewegt, stets effiziente Berechnungen und intuitive Erklärbarkeit gegeben sind.

Kapitel 6

Designverifikationssystem

Wie in Kap. 2.4 ersichtlich, ist die MMI-Testautomatisierung bereits sehr umfangreich und vielschichtig. Die aktuellen Tests decken allerdings nur die Verhaltenslogik des MMI ab; das Design der GUI wird nicht getestet. Auch in der Literatur wird im Bereich von GUI-Tests meist nur die Verhaltenslogik betrachtet, z.B. in [Li und Wu 2004] und [Kervinen u. a. 2006]. Systemverhalten und Interaktionslogik lassen sich leicht in Zustandsdiagrammen abbilden und es bietet sich das modellbasierte Testen an. Für die Verifikation von GUI-Design ist dieser Ansatz jedoch nicht geeignet, da es sich hierbei nicht um Prozessabläufe handelt. Ein System für eine umfassende Verifikation von GUI-Design ist aus der Literatur nicht bekannt.
In diesem Kapitel wird die Architektur des entwickelten Designverifikationssystems vorgestellt und der Verifikationsprozess als Ganzes erläutert. Den Anfang bildet in Abschnitt 6.1 die Formulierung der allgemeinen Systemanforderungen und ihre Realisierbarkeit. In Abschnitt 6.3 werden die einzelnen Systemkomponenten erläutert. Dabei wird konkret auf die jeweiligen Anforderungen eingegangen. In Abschnitt 6.4 folgt eine kurze Zusammenfassung.

6.1 Anforderungen

Das Ziel ist die Entwicklung eines wissensbasierten Systems, das die korrekte Umsetzung des Soll-Designs einer GUI prüft. Für das Designverifikationssystem im Gesamten ergeben sich die folgenden Anforderungen, die eine Erweiterung der in Kap. 1.2 zusammengefassten Ziele darstellen:

1. **Vollständigkeit**: Das Designverifikationssystem soll die Identifizierung aller

in Kap. 2.2.2 aufgeführten Designmerkmale gewährleisten.

2. **Unsichere Information**: Beim Ist-Design handelt es sich um unsichere Bildinformationen. Die Inferenzkomponente muss in der Lage sein, unsichere Informationen zu verarbeiten.

3. **Automatisierung**: Die Generierung der Wissensbasis sowie der gesamte Verifikationsprozess muss automatisiert erfolgen. Aufgrund des Umfangs einer MMI-GUI und der Vielzahl an MMI-Versionen ist es nicht praktikabel diese Aufgaben manuell durchzuführen.

4. **Maschinenlesbarkeit**: Für die automatisierte Eingabe und Verarbeitung müssen das Ist- und das Soll-Design in einem maschinenlesbaren Format vorliegen.

5. **Effizienz**: Die aktuellen MMI Versionen sind in ihrer GUI bereits sehr komplex und für zukünftige MMI-Generationen ist ein weiteres Wachstum der Komplexität zu erwarten. Alle Systemkomponenten müssen in der Lage sein, eine große Menge an Informationen effizient, d.h. zeit- und ressourcensparend, zu verarbeiten.

6. **Erweiterbarkeit**: Das Verifikationssystem soll einen modularen Aufbau besitzen, um zukünftige Änderungen auf einfache Weise zu ermöglichen. Da neue MMI-Generationen meist auf Basis von alten entwickelt werden, sollte auch die formale Soll-Design-Beschreibung iterativ erweiterbar sein, um für neue MMI-Generationen nicht komplett neu erzeugt werden zu müssen. Dasselbe ist auch sinnvoll für bereits generierte Bayes'sche Netze.

7. **Universalität**: Das Verifikationssystem soll universell für verschiedene MMI-Generationen mit unterschiedlichem Design eingesetzt werden können sowie auch für GUI-Design anderer Systeme.

8. **Korrektheit**: Das Verifikationssystem liefert korrekte Ergebnisse. Eine automatisierte Korrektheitsprüfung für die einzelnen Komponenten ist aufgrund der Komplexität des Systems sinnvoll.

In dieser Arbeit wird vor allem auf die Realisierung der Forderungen 1 bis 5 eingegangen. Auch die übrigen Forderungen sind für den produktiven Einsatz in der

MMI-Testautomatisierung entscheidend, konnten im Rahmen der Dissertation aber nicht erschöpfend evaluiert werden.

6.2 Realisierbarkeit

Im Gegensatz zu Klassifikationsaufgaben im Allgemeinen sind die Merkmale für die Designverifikation bereits durch das Soll-Design gegeben. Bei anderen Klassifikationsaufgaben, z.B. bei der Musterklassifikation, müssen oftmals erst geeignete Merkmale gefunden werden, was meist ein schwieriger und langwieriger Prozess ist. Zudem geht es bei der Designverifikation lediglich darum, bereits klassifizierte grafische Elemente mit dem Soll zu vergleichen und in Kombination miteinander zu verifizieren. D.h. die Designverifikation soll nicht die eigentliche Klassifikation, die Bilderkennung ersetzen, sondern wertet nur die Klassifikationsergebnisse aus.
Im Idealfall werden alle im Soll-Design enthaltenen Designmerkmale verifiziert. Allerdings kann nur das verifiziert werden, was im Ist-Design mittels Bildverarbeitung erkannt werden kann. Dies betrifft z.B. die Verifikation des Fonts. Für die Designverifikation wird die vektorbasierte Bildverarbeitung von [Schad 2011] eingesetzt, die keine Identifizierung des Fonts unterstützt. Angemerkt sei jedoch, dass Algorithmen zur Fonterkennung existieren, bspw. der Web-Service *WhatTheFont WhatTheFont*.
Nicht gefunden werden können und sollen Fehler im Textinhalt und in der Verhaltenslogik, da dies nicht Teil der Designspezifikation oder der daraus unmittelbar resultierenden Semantik sind. Der konkrete Inhalt eines Textelements ist irrelevant für die Designverifikation. Der zweite Punkt betrifft das Wissen über den aktuellen Systemzustand, der für die Designverifikation nicht vorliegt. So kann z.B. der folgende Sachverhalt nicht als Fehler erkannt werden: „Ein Softkey ist im Zustand *inaktiv*, der aktuelle Systemzustand erfordert aber einen Softkey im Zustand *aktiv*“.
Des Weiteren ist die Durchschnittsfarbe bei symbolischen Elementen nicht relevant, wenn mit Angabe des Referenzbildes dieses Merkmal bereits abgesichert ist. Dasselbe gilt für die Dimension, sofern es sich nicht um eine skalierungsinvariante Bilderkennung handelt.

6.3 Systemarchitektur

Das Designverifikationssystem ist ein wissensbasiertes System, das sich aus den in Kap. 4.1.1 aufgeführten Komponenten zusammensetzt. Die Abb. 6.1 zeigt die einzelnen Systemkomponenten, Eingabe- und Ausgabedaten sowie die Kommunikationsschnittstellen.

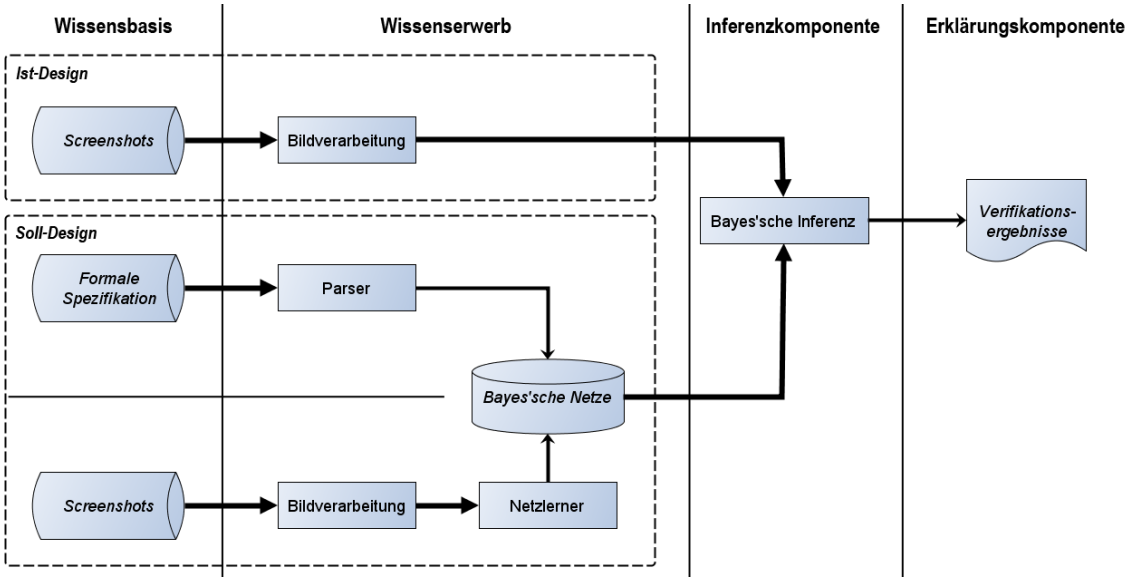


Abbildung 6.1: Architektur des Designverifikationssystems.

6.3.1 Inferenzkomponente

Für die Realisierung der Inferenzkomponente wird ein Bayes'sches Netz verwendet. In Kap. 5 wurden verschiedene Bayes'sche Netze für die Klassifikation von Schriftzeichen vorgestellt. Für die Verifikation des GUI-Designs soll nun mittels eines Bayes'schen Netzes entschieden werden, ob das Ist-Design dem spezifizierten Soll entspricht oder Fehler aufweist. Dafür muss das Bayes'sche Netz die Merkmale des Soll-Designs abbilden und für das Ist Eingabevariablen anbieten. In Kapitel 7 wird die Verwendung eines Bayes'schen Netzes für den Vergleich zwischen Soll- und Ist-Design ausführlich beschrieben.

6.3.2 Wissensbasis

Die Wissensbasis ist zweigeteilt in das Ist- und das Soll-Design. Das Ist-Design ist in Form von Screenshots gegeben, die das zu verifizierende GUI-Design darstellen. Für das Soll-Design gibt es zwei verschiedene Wissensquellen, eine formale Spezifikation oder ebenfalls Screenshots. Die Screenshots müssen in diesem Fall das korrekte Design abbilden, d.h. von einer korrekten GUI stammen.
Während die Screenshots auf einfache Weise mittels eines Framegrabbers oder einer Kamera erzeugt werden können, steht eine formale Soll-Design-Beschreibung bislang nicht zur Verfügung. D.h. es muss eine geeignete Spezifikationssprache gefunden werden, was umfangreichere Überlegungen bedarf. In Kap. 3 wurde XML als gängige Auszeichnungssprache vorgestellt. Obgleich sehr viele Spezifikationssprachen auf XML basieren, besitzt XML mit seiner komplexen Tag-Struktur auch einige Nachteile. Bei der Spezifikation von GUI-Design weisen die existierenden Beschreibungssprachen zudem gravierende Mängel auf. Deshalb wurde im Rahmen dieser Arbeit für die Spezifikation von GUI-Design eine neue Sprache entwickelt. Die Neuentwicklung wird in Kap. 8 detailliert vorgestellt.

6.3.3 Wissenserwerbskomponente

Das in Form von Screenshots und formaler Spezifikation vorliegende Ist- und Soll-Design muss in einer Wissenserwerbskomponente noch weiter aufbereitet werden, damit es geeignet von der Inferenzkomponente verarbeitet werden kann. Wissen, das als Screenshots vorliegt, muss mittels Bildverarbeitung extrahiert werden, damit es wie die formale Spezifikation in einem maschinenlesbaren Format vorliegt. Aus dem erworbenen Wissen muss dann ein Bayes'sches Netz generiert werden.

6.3.3.1 Erwerb des Ist-Designs mittels Bildverarbeitungsmethoden

Für die Erkennung des Inhaltes von Bildern steht eine Vielzahl von Techniken bereit (vgl. Kap. 2.3.5 und 2.4.2.1). Für die Designverifikation ist es wichtig, jedes Element zusammen mit den zu verifizierenden Merkmalen in einem Screenshot zu erkennen. Als erstes ist die Unterscheidung in Text- und Symbolelemente notwendig. Sind die Elemente klassifiziert, lassen sich weitere Merkmale wie Position, Dimension, Farbe oder Referenzbild leicht bestimmen.
Bei der Wahl einer Bildverarbeitung, die die Eingabedaten für eine Designveri-

fikation liefert, gilt zu beachten, dass für die Klassifikation möglichst kein designspezifisches Wissen verwendet wird. Andernfalls kann der Grund für ein nichtklassifizierbares Element bereits ein Designfehler sein, der dann nicht weiter spezifiziert werden kann. Für eine MMI-GUI bedeutet das, dass stets alle Elemente in jedem Screen gefunden werden müssen und nicht aus Zwecken der Effizienz die Anzahl der möglichen Elemente durch Wissen über den Kontext oder den Screenbereich eingeschränkt wird. Des Weiteren muss die Bildverarbeitung auch auf Inhalte hinweisen, die unbekannt sind und daher nicht klar als Text oder Symbol klassifiziert werden können.
Für die Designverifikation kommt das in Kap. 2.3.5.4 vorgestellte vektorbasierte Verfahren von [Schad 2011] zum Einsatz. Das Verfahren wurde speziell für diesen Anwendungsfall entwickelt und liefert die geforderten Ergebnisse.

6.3.3.2 Erwerb des Soll-Designs aus einer formalen Spezifikation

In Kap. 5.3.1 wurde erläutert, dass Bayes'sche Netze automatisch aus einer formalen Spezifikation transferiert werden können. Damit der Transfer effizient ist, muss die formale Beschreibung zu der Struktur eines Bayes'schen Netzes passen. Daneben sind Zusatzinformationen über die Parameter notwendig. Neben der Konzeption einer formalen Soll-Designspezifikation, wird in Kap. 8 auch die automatische Generierung eines Bayes'schen Netzes aus dieser detailliert beschrieben.

6.3.3.3 Erwerb des Soll-Designs durch Lernen

In den Kap. 8 und 10 wird deutlich werden, dass theoretisch zwar jedes GUI-Design formal beschrieben werden kann, in vielen realen Anwendungsbereichen einige Informationen aber niemals in dieser Form zur Verfügung stehen werden. Aus diesem Grund zeigt Abb. 6.1 noch eine zweite Möglichkeit, das Bayes'sche Netz aus dem Soll-Design zu generieren: Wenn die Soll-Informationen in Form von korrekten Screenshots vorliegen und eine Bildverarbeitung diese Informationen extrahiert, werden Daten erzeugt, die zum Lernen Bayes'scher Netze verwendet werden können.
Wie bereits in Kap. 1.2 erläutert, ist es nicht praktikabel für jede mögliche optische Darstellung einer GUI einen Screenshot zu erzeugen. Damit kann die Menge korrekter Screenshots stets nur einen Teil der gesamten GUI abbilden. Das Lernen dient deshalb nur als Erweiterung der formalen Designspezifikation und kann

sie nicht komplett ersetzen. In Kap. 9 werden die Möglichkeiten und Grenzen des Lernens von Designmerkmalen ausführlich diskutiert.

6.3.4 Erklärungskomponente

In der Inferenzkomponente wird das sichere Soll- und das unsichere Ist-Design in Form eines Bayes'schen Netzes repräsentiert und ausgewertet. Das Ergebnis ist die lokale und globale Korrektheit jedes gefundenen Designelements als Wahrscheinlichkeit. Da die Benutzer in der Regel Tester der GUI und keine Experten im Gebiet wissensbasierter Systeme oder Bayes'sche Netze sind, müssen die Wahrscheinlichkeitsaussagen benutzerfreundlich präsentiert werden.

6.3.4.1 Inhalt

In Kap. 5.6 wurden bereits die verschiedenen Arten und Möglichkeiten der Erklärbarkeit Bayes'scher Netze ausgeführt. Für das Designverifikationssystem ist vor allem die Modellerklärung von Bedeutung; die Erklärungskomponente soll nicht nur allgemein auf fehlerhafte Screenshots hinweisen, sondern eine detaillierte Begründung liefern, welche lokalen und/oder globalen Merkmale nicht dem Soll entsprechen. Um die Fehlerursachen auf einfache Weise identifizieren zu können, muss das Bayes'sche Netz eine geeignete Struktur mit vermittelnden Variablen (s. Kap. 5.6.2) besitzen.

6.3.4.2 Darstellung

Bei der Verifikation von GUI-Design bietet sich vor allem eine visuelle Ausgabe an, bei der im verifizierten Screenshot die Fehlerbereiche markiert sind. Daneben sollte eine textuelle Ausgabe gegebenenfalls zusätzliche Informationen liefern. Damit bei einer großen Fehleranzahl die visuelle Anzeige nicht zu unübersichtlich wird, sollte dem Benutzer das aktive Ein- und Ausblenden der Markierungen ermöglicht werden. Für ein noch besseres Verständnis kann auch in Erwägung gezogen werden, die Wahrscheinlichkeiten nicht numerisch, sondern linguistisch zu formulieren (vgl. Kap. 5.6.1.3). In den Kap. 10.2 und 10.3.4 sind ein paar Vorschläge für die visuelle Ergebnisausgabe abgebildet.

6.4 Schlussfolgerung

Als wissensbasiertes System strukturiert, gewährt das Designverifikationssystem eine klare Trennung zwischen den einzelnen Komponenten. Damit ist eine einfache Erweiterbarkeit gewährleistet. Die unsicheren Ist-Designinformationen werden als Wahrscheinlichkeiten interpretiert und in einem Bayes'schen Netz verarbeitet. Mit Bereitstellung maschinenlesbarer Eingabedaten durch die Wissenserwerbskomponente, kann die gesamte Designverifikation automatisiert erfolgen. Beim Erwerb des Ist-Designs ist die Bildverarbeitung der beschränkende Faktor. Mit [Schad 2011] steht aber eine umfangreiche Bildverarbeitung zur Verfügung, mit der alle grafischen Elemente erkannt und ihre Designmerkmale, mit Ausnahme detaillierter Informationen über Fonts, identifiziert werden können. Die Effizienz des Designverifikationssystems wird maßgeblich durch die Art der Wissenbasis und den Wissenserwerb sowie die Inferenzkomponente bestimmt. In den folgenden Kap. 7 bis Kap. 9 wird gezeigt, wie die in Abschnitt 6.1 formulierten Anforderungen konkret für diese Systemkomponenten umgesetzt werden können.

Kapitel 7

Bayes'sches Netz zur Designverifikation

Dieses Kapitel beginnt in Abschnitt 7.1 mit der abstrakten Beschreibung der Struktur und der Parameter des Bayes'schen Netzes für die Designverifikation. Dabei wird auch auf die Kombination und Interpretation von Korrektheitsaussagen eingegangen. Abschnitt 7.2 stellt den allgemeinen Verifikationsprozess anhand des konzipierten Netzes dar. In Abschnitt 7.3 wird basierend auf dem abstrakten Konzept ein konkretes Bayes'sches Netz für die Verifikation eines Teils der MMI-GUI modelliert und die Verifikation durchgeführt. Abschnitt 7.4 liefert ausführliche Begründungen für das konzipierte Bayes'sche Netz und Abschnitt 7.5 schließt das Kapitel mit einer kurzen zusammenfassenden Bewertung ab.
Im Rahmen der Designverifikation werden die Wahrscheinlichkeiten als Prozentangaben zwischen 0 und 100 interpretiert.

7.1 Allgemeiner Aufbau

Durch die Funktionsweise ergibt sich ein dreigeteiltes Bayes'sche Netz, bestehend aus *Eingabe-*, *Vergleichs-* und *Ausgabeschicht*. Über die *Merkmalsvariablen* in der Eingabeschicht werden die Merkmale des Ist-Designs in das Bayes'sche Netz eingegeben. Die *Erklärungsvariablen* der Vergleichsschicht repräsentieren das Soll-Design und ermöglichen den Vergleich mit dem Ist. Die Ergebnisse des Vergleichs, d.h. die Korrektheit, liefern die *Verifikationsvariablen* der Ausgabeschicht. Für eine geeignete Handhabung werden Konventionen für die Namensgebung der Variablen

definiert.

7.1.1 Variablen

7.1.1.1 Merkmalsvariablen (MV)

In der Eingabeschicht wird ein lokales oder globales Merkmal (s. Kap. 2.2.2) jeweils durch eine MV abgebildet. Der Name einer MV setzt sich aus dem Merkmal und dem/den Namen des/der betreffenden Elemente zusammen: (`Element_`)$^+$`Merkmal`. Die Zustandsmenge der MVs ergibt sich aus dem Soll-Design: Jede MV hat eine Zustandsmenge $Z = \{s_1, ..., s_n, a_1, ..., a_m, anders\}$ mit $s_1, ..., s_n$ die n Sollwerte und $a_1, ..., a_m$ die m zu gewichtenden Soll-Abweichungen. Zusätzlich gibt es stets den Zustand $anders$, der die Zustandsmenge vervollständigt. Ein Merkmal wird als harte oder weiche Evidenz (s. Kap. 5.2.1) in der zugehörigen MV gesetzt.

7.1.1.2 Erklärungsvariablen (EV)

In der Vergleichsschicht gibt es pro MV eine EV, wobei die EV ein Kindknoten der MV ist. Eine EV besitzt normalerweise die zwei Zustände *wahr* und *falsch* und bildet damit Korrektheitsaussagen zu allen möglichen Kombinationen ab. Evtl. ist noch ein dritter Zustand *unbekannt* notwendig (siehe Beispiel in Abschnitt 7.3).
Der Name einer EV setzt sich aus dem Namen der zugehörigen MV und dem Präfix `Korrekt` zusammen: `Korrekt_`(`Element_`) + `Merkmal`.
Weitere EVs sind möglich, um bestimmte Kombinationen der Eingabeinformationen zu repräsentieren und im Fehlerfall detaillierte Aussagen treffen zu können. Aus Gründen der Effizienz werden die EVs so in vermittelnden Variablen zusammengefasst, dass jede vermittelnde Variable maximal drei Elternknoten besitzt und maximal drei Variablen als Blattknoten für die Weiterleitung in die Ausgabeschicht übrig bleiben.

7.1.1.3 Verifikationsvariablen (VV)

Eine VV repräsentiert mit den Zuständen *wahr*, *falsch* und ggf. *unbekannt* die Korrektheit eines Elements in der Gesamtheit seiner lokalen Merkmale, oder die Korrektheit eines globalen Merkmals. Eine Menge von EVs sind die Elternknoten der VV. Darüber hinaus kann eine VV einer lokalen Verifikation einen weiteren Kind-

knoten besitzen, der die Existenz verifiziert: Die Korrektheit der Merkmale eines Elements bedeutet auch, dass dieses Element existiert.

7.1.2 Netzstruktur

Die Abbildungen 7.1 und 7.2 zeigen exemplarisch verschiedene Bayes'sche Netze für die Designverifikation. Je nach konkretem Design sind auch andere, ähnliche Strukturen möglich. In Abb. 7.1 sind die beschriebenen Schichten des Bayes'schen Netzes und ihre Variablen für die Verifikation von lokalen Merkmalen zur Verifikation eines Elements *Element1* dargestellt. Darin ist auch die Namensgebung für vermittelnde Variablen zu sehen, z.B.
`Korrekt_MerkmalA_MerkmalB_MerkmalC`. Wie in Kap. 2.2.2 erläutert, kann ein

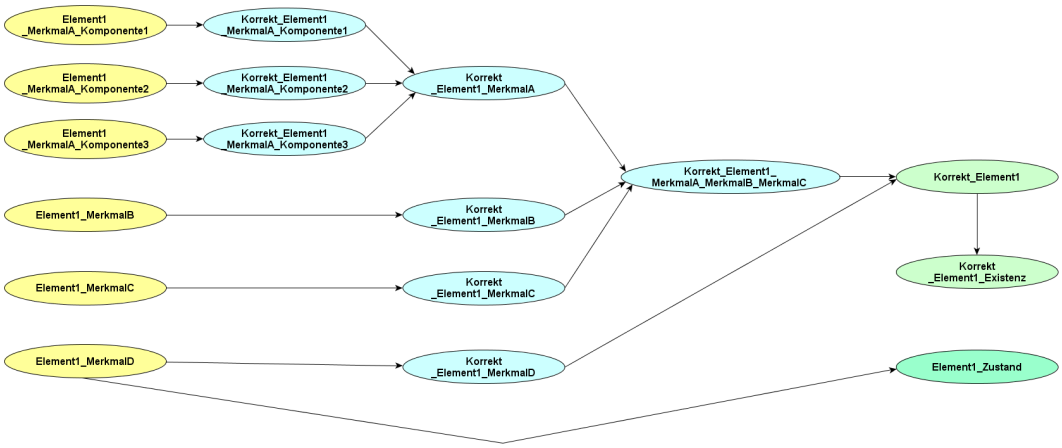


Abbildung 7.1: Exemplarische Struktur eines Bayes'schen Netzes für die Verifikation von lokalen Designmerkmalen eines Elements *Element1* mit den drei Schichten Eingabe (gelb), Vergleich (blau) und Ausgabe (grün).

Merkmal aus mehreren Einzelkomponenten bestehen, bspw. die Position aus der x- und der y-Koordinate. Die Variablen `Element1_MerkmalA_Komponente1` und `Element1_MerkmalA_Komponente2` repräsentieren damit als Komponenten von *MerkmalA* ein solches Merkmal. Für eine detaillierte Erklärung der Ergebnisse ist es sinnvoll, semantisch zusammengehörige Merkmalskomponenten auch in der Vergleichsschicht in einer EV zusammenzufassen. Im Beispiel geschieht dies in der Variablen `Korrekt_Element1_MerkmalA`. Den Merkmalen *MerkmalB*, *MerkmalC* und *MerkmalD* wird ebenso jeweils eine eigene EV zugeteilt.

Die Ausgabeschicht besteht hier aus drei Variablen. Die VV `Korrekt_Element1` liefert das Ergebnis über die lokale Verifikation von Element *Element1* und die VV bzw. Existenzvariable `Korrekt_Element1_Existenz` über die Existenz. Die dritte Variable `Element1_Zustand` gibt Auskunft über den Zustand des Elements, sofern verschiedene Zustände möglich sind. Im Beispiel ist der Elementzustand von genau einem Elementmerkmal konditional abhängig, er kann aber auch von mehreren Merkmalen abhängig sein. In Abb. 7.2 sind Beispiele für globale Verifikationen abgebildet.

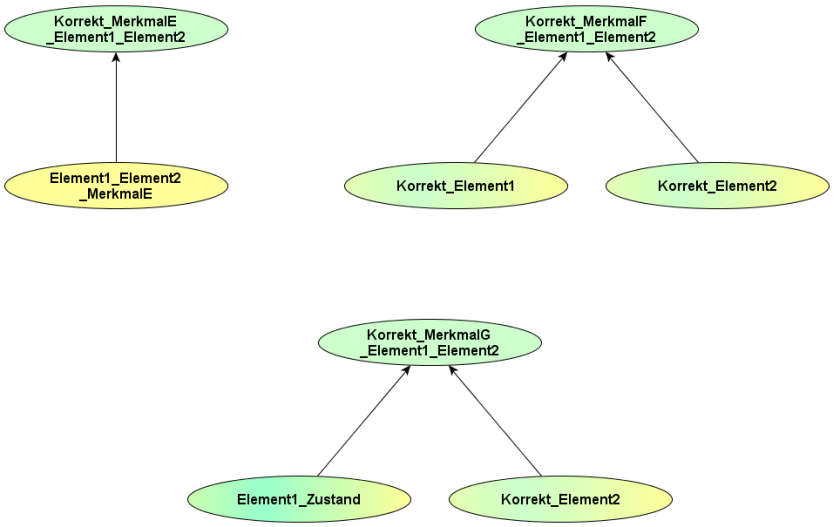


Abbildung 7.2: Exemplarische Bayes'sche Netze für die Verifikation von globalen Merkmalen. Die Vergleichsschicht ist hier zugleich die Ausgabeschicht. Die gelbgrünen MVs sind zugleich VVs lokaler Verifikationen.

Zustandsvariablen können hier auch als MVs agieren. Für die Koexistenz oder den gegenseitigen Ausschluss können ebenso auch die Existenzvariablen MVs sein. Merkmal *MerkmalE* repräsentiert ein weiteres mögliches globales Merkmal, z.B. den Abstand. Bei den globalen Verifikationen handelt es sich bei einer EV meist bereits um eine VV der Ausgabeschicht.
Die Kantenrichtungen beschreiben durchgängig kausale Beziehungen, d.h. die Merkmale sind die Ursachen, die zu den Korrektheitsaussagen in der Inferenz- und der Ausgabeschicht führen. Daneben kann auf diese Weise sehr einfach unsichere Evidenz in die MVs eingegeben werden (s. Kap. 5.2.3).
Zudem zeigt sich, dass alle Bayes'schen Netze eine Baumstruktur aufweisen (Definition Baumstruktur s. z.B. [Shafer und Shenoy 1990]). Netze für lokale Verifika-

tionen sind dabei evtl. mit Netzen globaler Verifikationen verbunden.
Unter der Voraussetzung, dass die Elementnamen eindeutig sind, sind auch die Namen der MVs eindeutig.

7.1.3 Parameter

In den CPTs der EVs ist das Soll des jeweiligen Merkmals kodiert sowie die Gewichtung der Abweichungen über den gesamten Wertebereich. Die Parameter der CPTs von vermittelnden Variablen und VVs definieren die Kombination der Korrektheitsaussagen.

7.1.3.1 Gewichtung von Soll-Abweichungen

Die Übergangswahrscheinlichkeiten in den CPTs der EVs definieren die Beziehung zwischen dem eingegebenen Merkmal und ihrer Übereinstimmung mit dem Soll. Durch Angabe entsprechender Wahrscheinlichkeiten in den CPTs können Abweichungen geeignet gewichtet werden. Die Definition der Übergangswahrscheinlichkeiten gibt damit an, wie stark Abweichungen als Fehler gewertet werden und ab welchem Grad der Abweichung das jeweilige Merkmal als sicher falsch betrachtet werden soll. Es kann nicht nur bestimmt werden, wie stark Soll-Abweichungen als Fehler zählen, sondern auch die Bedeutung von verschiedenen Merkmalen für die Gesamtkorrektheit eines Elements kann unterschiedlich gewichtet werden.

7.1.3.2 Kombination von Korrektheitsaussagen

Die vermittelnden Variablen der Vergleichsschicht und die VVs der Ausgabeschicht stellen meist Kombinationen von Korrektheitsaussagen dar. Die Kombination der Korrektheitsaussagen ist auf verschiedene Weise möglich, je nachdem wie stark sich fehlerhafte Merkmale auf die Gesamtkorrektheit auswirken sollen:

- **Noisy-OR**: Die Gesamtkorretheit ist zu 100% korrekt, wenn wenigstens ein Merkmal zu 100% korrekt ist.

- **Noisy-AND**: Wenn ein Merkmal zu 100% falsch ist, ist auch die Gesamtkorrektheit zu 100% falsch.

- **Anteilige Kombination**: Die Wahrscheinlichkeit der kombinierten Korrektheit gibt Rückschluss auf die Anzahl der korrekten bzw. der fehlerhaften Eingabeinformationen.

Noisy-OR eignet sich nicht für die Designverifikation, da das Design eines Elements nicht als 100% korrekt gewertet werden soll, wenn mindestens ein Merkmal falsch ist. Im Gegensatz dazu gewichtet Noisy-AND sehr streng. Beim Bayes'schen Netz in 7.3 wird die anteilige Kombination verwendet. Diese hat den Vorteil, anhand der Gesamtkorrektheit auf die Anzahl der falschen Merkmale rückzuschließen. Ein weiterer Vorteil ist eine gute Performanz: Es ist eine effizientere Abfrage der Einzelmerkmale möglich, da abgebrochen werden kann, wenn so viele wie durch die Gesamtwahrscheinlichkeit repräsentierte inkorrekte Merkmale gefunden wurden.
Andere Kombinationen sind ebenso möglich. Der Einfluss jedes Merkmals auf die Gesamtkorrektheit kann individuell gewichtet werden; z.B. wenn Fehler in der Farbe als gravierender eingestuft werden sollen, als Fehler in der Position.

7.1.4 Verifikationsergebnisse

Die direkte Ausgabe der Designverifikation sind Wahrscheinlichkeitsverteilungen über die Zustände *wahr*, *falsch* und ggf. *unbekannt* der VVs. Die Wahrscheinlichkeit des Zustands *wahr* liefert eine Aussage darüber, wie sicher es ist, dass das betrachtete Ist-Element oder globale Ist-Merkmal dem Soll entspricht. Die Wahrscheinlichkeit $P(wahr) = x$ ist abhängig von der Gewichtung der Soll-Abweichungen und der Kombination der Korrektheitsaussagen und muss daher nicht zwangsläufig bedeuten, dass vom betrachteten Element bzgl. der Merkmale genau $x\%$ korrekt sind. Die Wahrscheinlichkeitsverteilungen der EVs liefern detaillierte Begründungen für fehlerhaftes lokales oder globales Design. Eine hohe Sicherheit für den Zustand *unbekannt* bedeutet darüber hinaus, dass die Verifikation aufgrund fehlender Merkmale oder Unstimmigkeiten nicht durchgeführt werden konnte.

7.2 Verifikationsprozess

Für die lokale Verifikation eines Elements aus dem Ist-Design müssen die lokalen Merkmale in die MVs jedes Bayes'schen Netzes eingegeben werden. Mittels des effizienten Junction-Tree-Algorithmus (s. Kap. 5.2.1) können dann die Wahr-

scheinlichkeitsverteilungen der VVs berechnet werden. Das Ergebnis ist entweder die eindeutige Zuweisung zu einem Soll-Element, eine mehrfache Zuweisung oder gar keine:

- **Eindeutige Zuweisung**: Das Ist-Element entspricht genau einem Soll-Element und wird damit korrekt dargestellt.
- **Mehrfachzuweisung**: Ein Ist-Element wird mehreren Soll-Elementen zugewiesen und es ist zu klären, ob die gegebenen Merkmale des Ist-Elements evtl. nicht ausreichen, um eine eindeutige Zuweisung zu ermöglichen. In diesem Fall ist eine Ausgabe sinnvoll, die auf den Mangel an Merkmalen hinweist.
- **Keine Zuweisung**: Ein Ist-Element stimmt mit keinem Soll-Element überein und weist damit Designfehler auf. In diesem Fall ist durch Abfrage der EVs und vermittelnden Variablen herauszufinden, ob es Soll-Elemente gibt, die nur in wenigen Merkmalen vom Ist-Element abweichen und so evtl. in bestimmten Merkmalen des Ist-Elements Fehler vorliegen.

Daneben kann auch eine fehlerhafte oder unvollständige Spezifikation zu Mehrfachzuweisung oder keiner Zuweisung führen. Für fehlerhafte Elemente sollten keine globalen Verifikationen durchgeführt werden, da das zu Folgefehlern führen kann. Die Erklärungskomponente des Designverifikationssystems sollte auch in diesen Fällen aussagekräftige Erklärungen für die Ergebnisse liefern.

7.3 Verifikation eines Wizardmenüs

Als konkretes Beispiel dient das in Abb. 7.3 dargestellte Wizardmenü aus dem Medien-Kontext eines MMI 3G+. Es umfasst acht Icons, von denen jeweils nur sieben gleichzeitig zu sehen sind, da die Icons *IconPause* und *IconPlay* an derselben Position stehen und sich gegenseitig ausschließen. Im Gegensatz zu den Icons im Setup-Wizardmenü aus Kap. 2.2.1 mit drei Zuständen, kann hier jedes Icon fünf verschiedene Zustände annehmen. Abbildung 7.4 zeigt exemplarisch die fünf Zustände für das Icon *IconPlay*. Wie zu sehen ist, existiert in Abhängigkeit vom Iconzustand ein *Tooltip* bestehend aus dem Textelement *TooltipText* und den zwei Symbolelementen *TooltipRahmen* und *TooltipPfeil*. In Abbildung 7.5 ist ein Bayes'sches Netz zur Verifikation der lokalen Merkmale des Icons und des zugehörigen Textelements

TooltipTextPlay sowie des globalen Merkmals *Koexistenz* zwischen dem Zustand von *IconPlay* und der Existenz von *TooltipTextPlay* dargestellt.

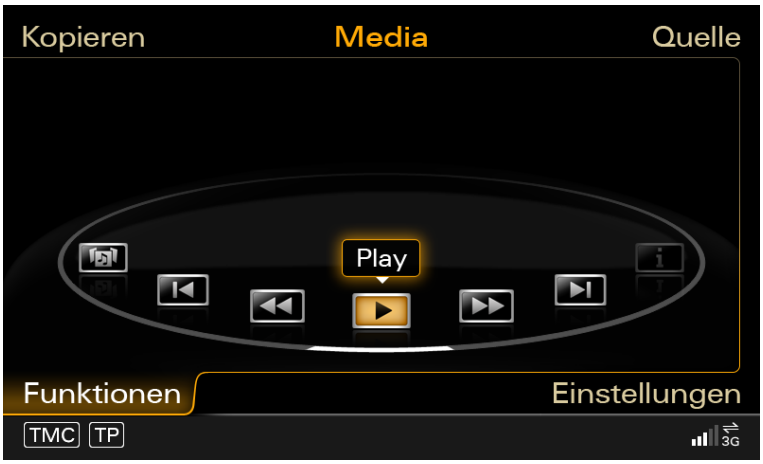


Abbildung 7.3: Ein Wizardmenü im Media-Kontext vom MMI 3G+.



Abbildung 7.4: Die fünf Zustände des Icons *IconPlay* und die Koexistenz mit dem Tooltip-Widget.

Die MVs sind gelb und die VVs grün markiert. Für das Textelement gibt es keine Existenzvariable, da die Existenz als Koexistenz mit dem Iconzustand definiert ist. Die Korrektheit dieses globalen Merkmals wird von der Variablen `Korrekt_Koexistenz_IconPlay_TooltipTextPlay` repräsentiert. Der Zustand *unbekannt* ist notwendig, da im Fall von `IconPlay_Referenz`=*anders* keine Aussage über die Korrektheit der Koexistenz gemacht werden kann.

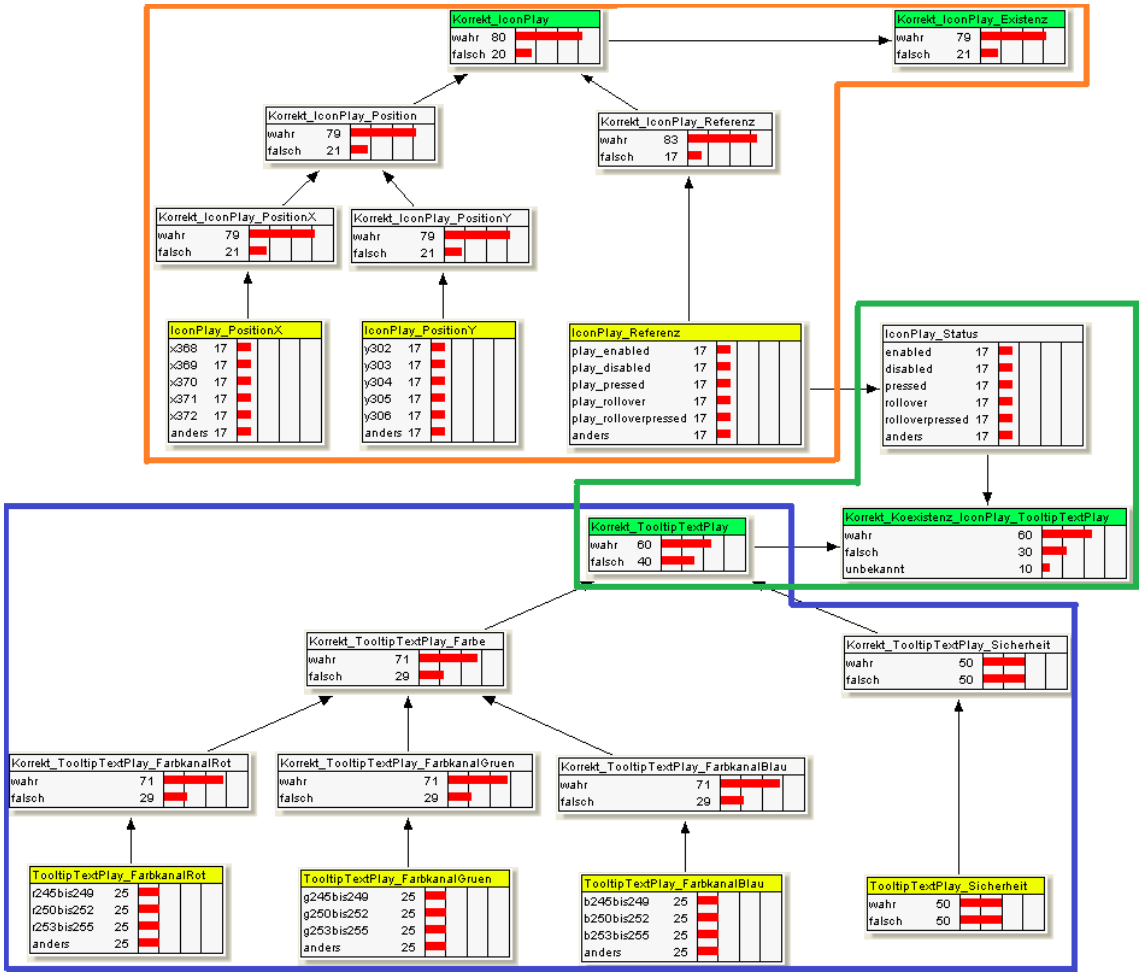


Abbildung 7.5: Bayes'sches Netz für die Verifikation eines Teils des Wizardmenüs aus Abb. 7.3. Das orangefarben umrahmte Teilnetz dient der Verifikation lokaler Merkmale des Icons *IconPlay*, das blau umrahmte Teilnetz der Verifikation lokaler Merkmale des dazugehörigen Textelements *TooltipTextPlay* und das grün umrahmte Teilnetz der Verifikation des globalen Merkmals *Koexistenz*. Die gelb markierten Variablen sind MVs und die grün markierten VVs. Die gezeigten CPTs sind in ihrem Initialzustand, da keine Merkmale bzw. Evidenzen gesetzt sind. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010))

7.3.1 Parameter

Alle Übergangswahrscheinlichkeiten sind ungleich 1 und 0, um mathematische Inkonsistenzen zu vermeiden (vgl. Kap. 5.4.3). Der CPT in Tabelle 7.1 definiert die

obligatorische Existenz des Icons *IconPlay*. Tabelle 7.2 zeigt am Beispiel des Merkmals *PositionX* vom Element *IconPlay*, wie in einem CPT die Soll-Abweichungen gewichtet werden können. Position $x = 370$ entspricht dem Soll und führt zu einer Korrektheit von 99%. Weicht die Position um ein Pixel vom Sollwert ab, wird die Korrektheit noch mit 98% definiert. Eine Abweichung um zwei Pixel führt zu einer Korrektheit von 90% und größere Abweichungen, die in Zustand *anders* zusammengefasst sind, zu 1%. Die Korrektheiten werden anteilig kombiniert. Der CPT

Korrekt_IconPlay_Existenz|Korrekt_IconPlay	**wahr**	**falsch**
wahr	0.99	0.01
falsch	0.01	0.99

Tabelle 7.1: CPT, der die obligatorische Existenz des Icons *IconPlay* definiert.

der VV `Korrekt_TooltipTextPlay_Farbe` in Tabelle 7.3 gibt dafür ein Beispiel. Merkmale wie *PositionX* und *FarbkanalRot* werden durch Bildverarbeitung genau identifiziert und können so als harte Evidenzen in den zugehörigen MVs gesetzt werden. Eine Ausnahme bilden die Merkmale *Referenz* und *Sicherheit*, die im Beispiel von den MVs `IconPlay_Referenz` und `TooltipTextPlay_Sicherheit` repräsentiert werden. Bei diesen Merkmalen handelt es sich um die Sicherheit des Erkenners, d.h. mit welcher Sicherheit dem Ist-Element ein Referenzbild zugewiesen werden kann bzw. wie sicher es ist, dass es sich um ein Textelement handelt. Ergebnisse der Bildverarbeitung sind in diesem Fall Wahrscheinlichkeiten, die als weiche Evidenzen in die MVs eingegeben werden.

Korrekt_IconPlay_PositionX|IconPlay_PositionX	**wahr**	**falsch**
x368	0.9	0.1
x369	0.98	0.02
x370	0.99	0.01
x371	0.98	0.02
x372	0.9	0.1
anders	0.01	0.99

Tabelle 7.2: CPT, der den Sollwert der oberen linken x-Position des Icons *IconPlay* definiert und Abweichungen entsprechend gewichtet.

CC|CR, CG, CB	wahr	falsch
wahr, wahr, wahr	0.99	0.01
wahr, wahr, falsch	0.66	0.34
wahr, falsch, wahr	0.66	0.34
wahr, falsch, falsch	0.33	0.67
falsch, wahr, wahr,	0.66	0.34
falsch, wahr, falsch	0.33	0.67
falsch, falsch, wahr	0.33	0.67
falsch, falsch, falsch	0.01	0.99

Tabelle 7.3: CPT, der die Gesamtkorrektheit des Merkmals *Farbe* definiert bestehend aus den drei Korrektheitsaussagen der einzelnen Farbkanäle *Rot*, *Grün* und *Blau* mit `CC=Korrekt_TooltipTextPlay_Farbe` und `CR/CG/CB=Korrekt_TooltipTextPlay_FarbkanalRot/_FarbkanalGruen/_FarbkanalBlau`.

7.3.2 Verifikation

Im Folgenden wird an sechs Beispielen gezeigt, wie sich verschiedene Ist-Merkmale auf das Verifikationsergebnis auswirken. Abb. 7.6 zeigt das Soll-Design. Das Ist-Design besteht bei den ersten fünf Beispielen stets aus zwei Elementen, von denen das eine als Symbolelement und das andere als Textelement von der Bildverarbeitung klassifiziert wurde. Im sechsten Beispiel konnte im Ist-Design lediglich ein Symbolelement gefunden werden.



Abbildung 7.6: Teil des Soll-Designs eines MMI-Wizardmenüs.

7.3.2.1 Soll-Merkmale

Das in Abb. 7.7 dargestellte Bayes'sche Netz zeigt die Wahrscheinlichkeiten nach Propagation der Sollwerte.

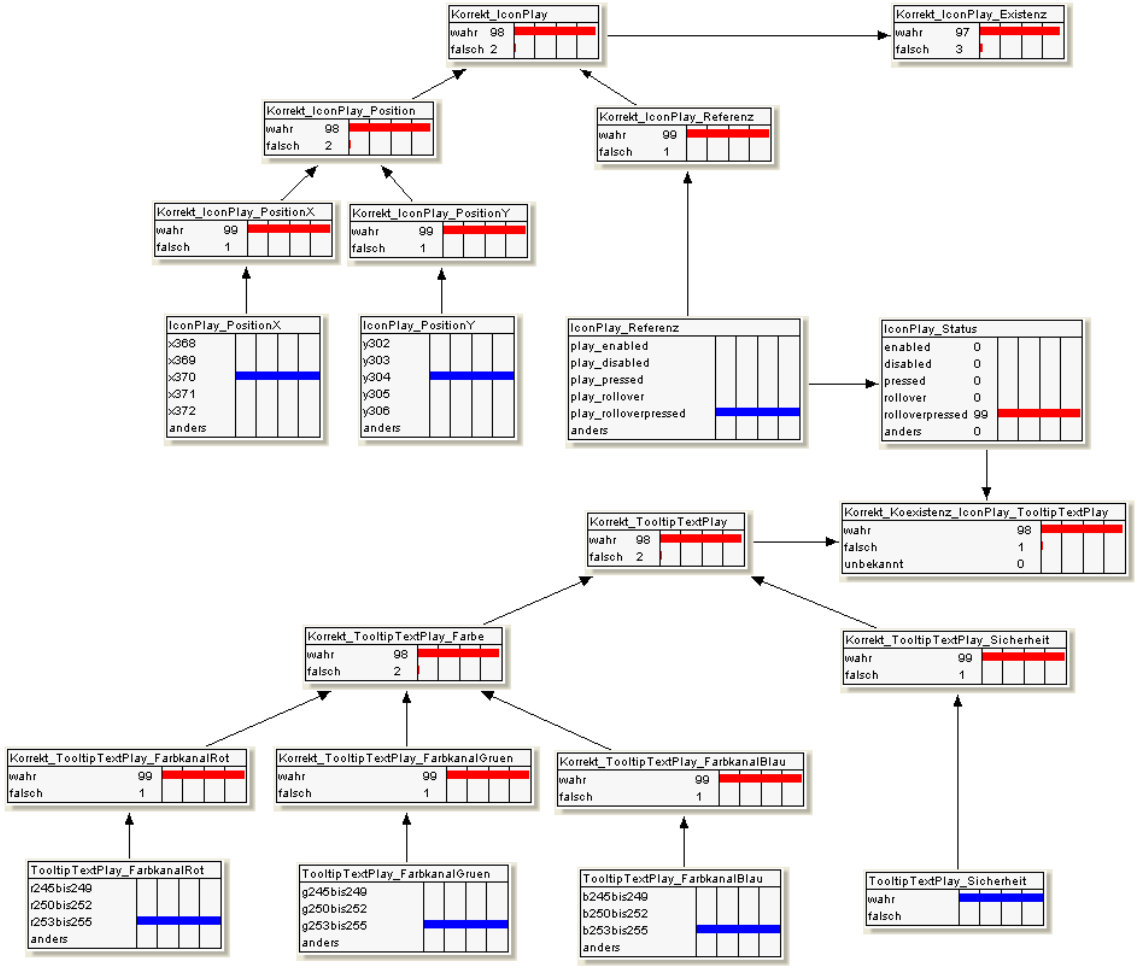


Abbildung 7.7: Bayes'sches Netz für die Verifikation des Wizardmenü-Ausschnitts in Abb. 7.6. Wahrscheinlichkeiten der einzelnen Variablen bei Propagation der Soll-Merkmale und sicheren Erkennersicherheiten. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010))

Auch die Erkennersicherheit wurde für die beiden Elemente *IconPlay* und *TooltipTextPlay* mit 100% eingegeben. Wie zu sehen ist, ergibt sich daraus eine Korrektheit von 99% für die einzelnen Merkmale (EVs `Korrekt_IconPlay_PositionX`, `Korrekt_IconPlay_PositionY`, `Korrekt_IconPlay_Referenz`, `Korrekt_TooltipTextPlay_FarbkanalRot`, `Korrekt_TooltipTextPlay_FarbkanalGruen`, `Korrekt_TooltipTextPlay_FarbkanalBlau`, `Korrekt_TooltipTextPlay_Sicherheit`). An den VVs kann die Korrektheit der lokalen Merkmalen jedes Elements mit jeweils 98% abgelesen werden. Ebenfalls beträgt auch die

Korrektheit der Koexistenz 98%. Die Korrektheit der Existenz des Icons *IconPlay* beträgt 97%. Die Abnahme der Wahrscheinlichkeiten ist auf die Kombination der Wahrscheinlichkeiten zurückzuführen (siehe auch Abschnitt 7.4.4).

7.3.2.2 Unsichere Erkennersicherheit

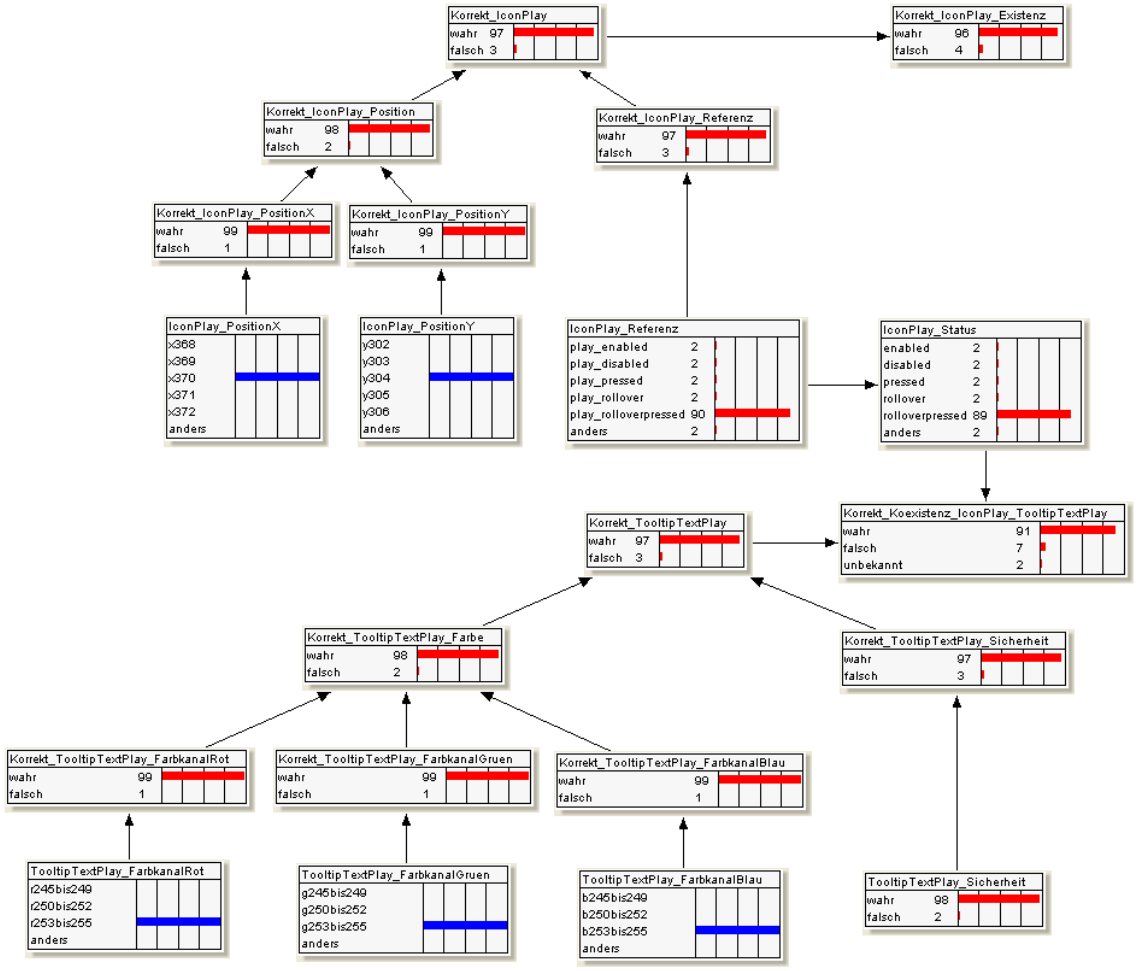


Abbildung 7.8: Bayes'sches Netz für die Verifikation des Wizardmenü-Ausschnitts in Abb. 7.6. Wahrscheinlichkeiten der einzelnen Variablen bei Propagation der Soll-Merkmale und unsicherer Erkennersicherheiten. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010))

Beim in Abb. 7.8 dargestellten Bayes'schen Netz wurden die Erkennersicherheiten reduziert. Das Element *TooltipTextPlay* wurde nur mit 98% Sicherheit als Textelement und die Referenz *play_rolloverpressed* wurde nur als 90% sicher klassifiziert.

Die sonstigen Ist-Merkmale entsprechen exakt dem Soll.
Die Gesamtkorrektheit der lokalen Merkmale beider Elemente beträgt nun 97%. Allerdings ist sie im Vergleich zum 100% korrektem Ist-Design im vorhergehendem Beispiel nur um jeweils 1% gesunken. Der Unterschied bei der Verifikation der Koexistenz ist größer; die erhöhte Unsicherheit bzgl. des Iconzustands wirkt sich stärker aus und führt zu einer Korrektheit der Koexistenz von 91%.

7.3.2.3 Schwache Soll-Abweichungen

Im Bayes'schen Netz in Abb. 7.9 ist zu erkennen, dass die lokalen und globalen Korrektheiten nun zwischen 94% und 98% liegen. Die Ursache dafür sind die Abweichungen vom Soll bei den meisten Merkmalen beider Elemente. Die Abweichungen wirken sich entsprechend der in den CPTs definierten Gewichtungen auf die Verifikationsergebnisse aus und führen, ebenso wie niedrige Erkennersicherheiten, zu geringeren Korrektheiten.

7.3.2.4 Designfehler

Für das Symbolelement werden nun die Merkmale *PositionX* und *PositionY* mit dem Zustand *anders* in das Bayes'sche Netz eingegeben. Für das Textelement weichen die Ist-Merkmale *FarbkanalRot* und *FarbkanalBlau* ebenso stark von den Sollwerten ab. Der grüne Farbkanal weicht weniger stark vom Soll ab und wird noch mit 90% als korrekt gewertet. Abb. 7.10 zeigt das Ergebnis.
Wie oben beschrieben, wurden die CPTs so definiert, dass die Korrektheiten der einzelnen Merkmale zu gleichen Anteilen die kombinierte Korrektheit beeinflussen. Für das Textelement ergibt sich daher eine etwas geringere als ein Drittel Wahrscheinlichkeit für ein korrektes Design. Das Symbolelement ist zu 50% korrekt, da das Merkmal *Position*, bestehend aus den Merkmalskomponenten *PositionX* und *PositionY*, gegenüber dem 100% korrekten Merkmal *Referenz* gleichwertig behandelt wird.
In beiden Fällen sollten die Wahrscheinlichkeiten der lokalen Verifikationen zu dem Ergebnis führen, dass das Ist-Design fehlerhaft ist. Die globale Verifikation weist eine Korrektheit von 98% auf, d.h. nicht fehlerhaft. Ihre Auswertung macht aber keinen Sinn, da zuerst die Korrektheit der Elemente an sich sichergestellt werden sollte.

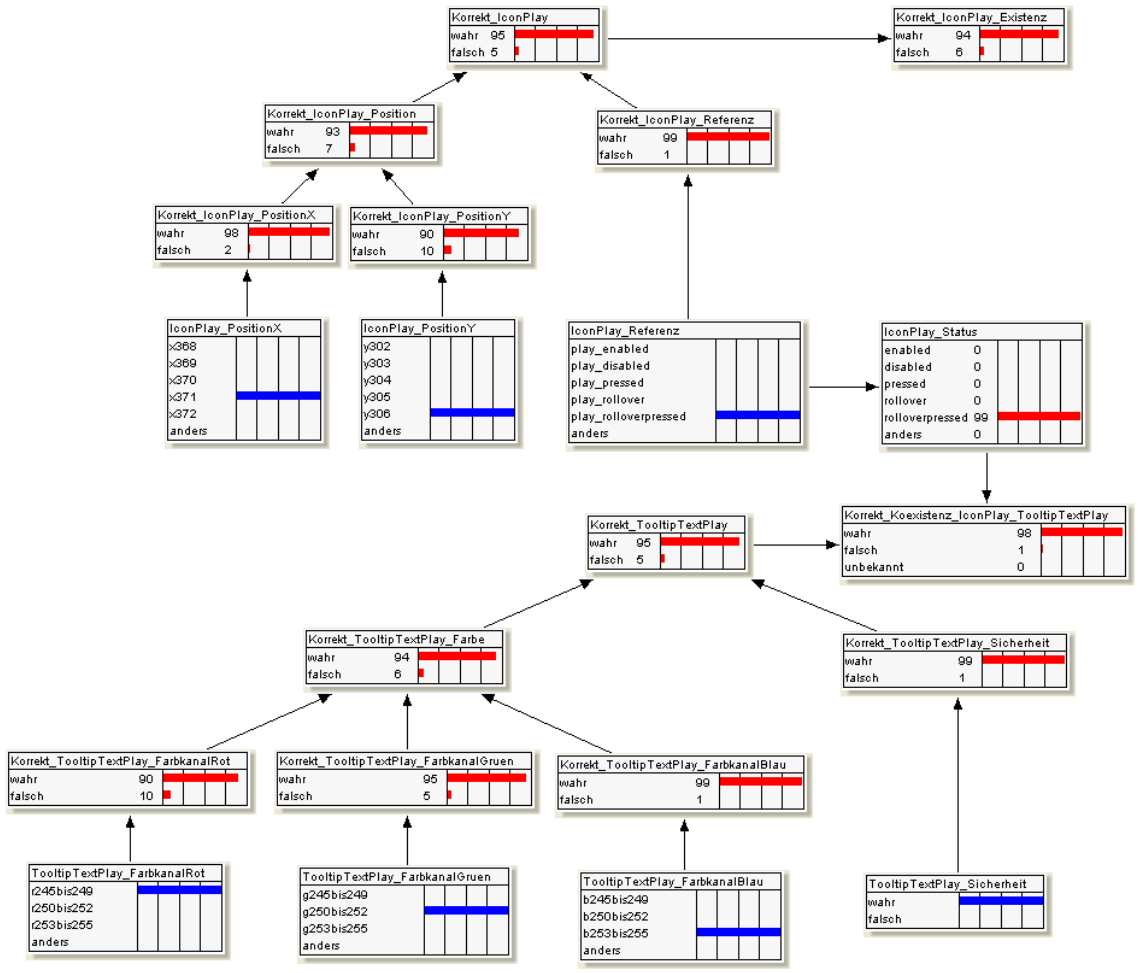


Abbildung 7.9: Bayes'sches Netz für die Verifikation des Wizardmenü-Ausschnitts in Abb. 7.6. Wahrscheinlichkeiten der einzelnen Variablen bei Propagation von Merkmalen, die schwach von den Sollwerten abweichen. (Netz erstellt mit Bayes-Builder (*BayesBuilder* 2007-2010)

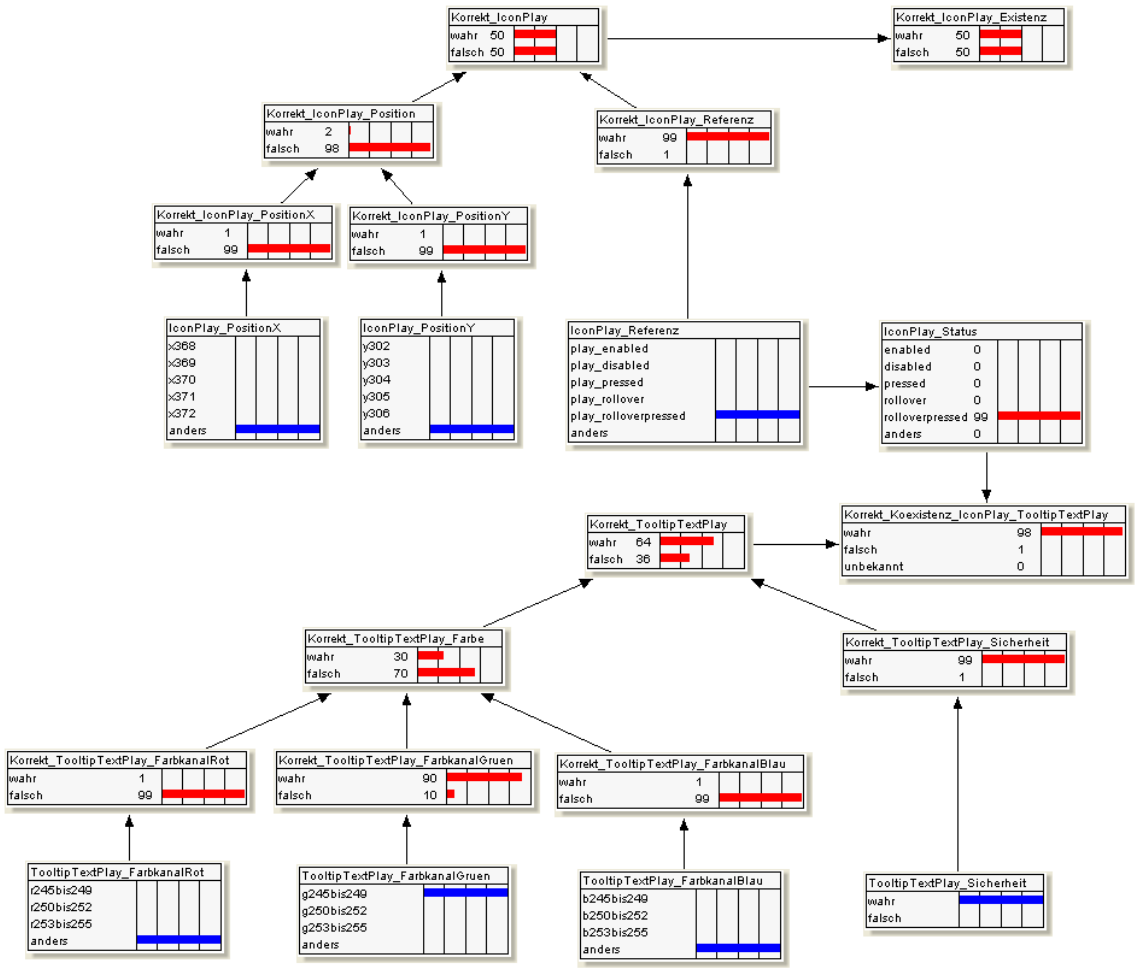


Abbildung 7.10: Bayes'sches Netz für die Verifikation des Wizardmenü-Ausschnitts in Abb. 7.6. Wahrscheinlichkeiten der einzelnen Variablen bei Propagation von Merkmalen, die teilweise sehr stark von den Sollwerten abweichen. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010))

7.3.2.5 Koexistenz falsch

Verifiziert wird das in Abb. 7.11 dargestellte Ist-Design. Im Gegensatz zu den vorangegangenen Beispielen ist der Zustand vom Merkmal *Referenz* des Symbolelements nun *play_disabled*. Daraus ergibt sich der Iconzustand *disabled*, bei dem laut Spezifikation kein Tooltip angezeigt werden darf (s. Abb. 7.4). Für ein Textelement existieren aber Ist-Merkmale und das Element *TooltipTextPlay* wird korrekt verifiziert. D.h. im Ist-Design existiert sowohl das Icon *play_disabled* als auch der Tooltip-Text. In Abb. 7.12 ist das Ergebnis mit einer Koexistenz-Korrektheit von 3% zu sehen.



Abbildung 7.11: Teil des Ist-Designs eines MMI-Wizardmenüs.

7.3.2.6 Koexistenz korrekt

Beim in Abb. 7.13 dargestellten Beispiel entspricht das erkannte Symbolelement erneut der Referenz *play_disabled*. Nun wird aber kein Textelement gefunden, das den geforderten Sollwerten für *TooltipTextPlay* entspricht. D.h., alle zu diesem Element gehörenden MVs werden auf den Zustand *anders* bzw. *falsch* gesetzt. Die Koexistenz ist in diesem Fall zu 97% korrekt, da wie spezifiziert nun kein Tooltip-Text vorhanden ist.

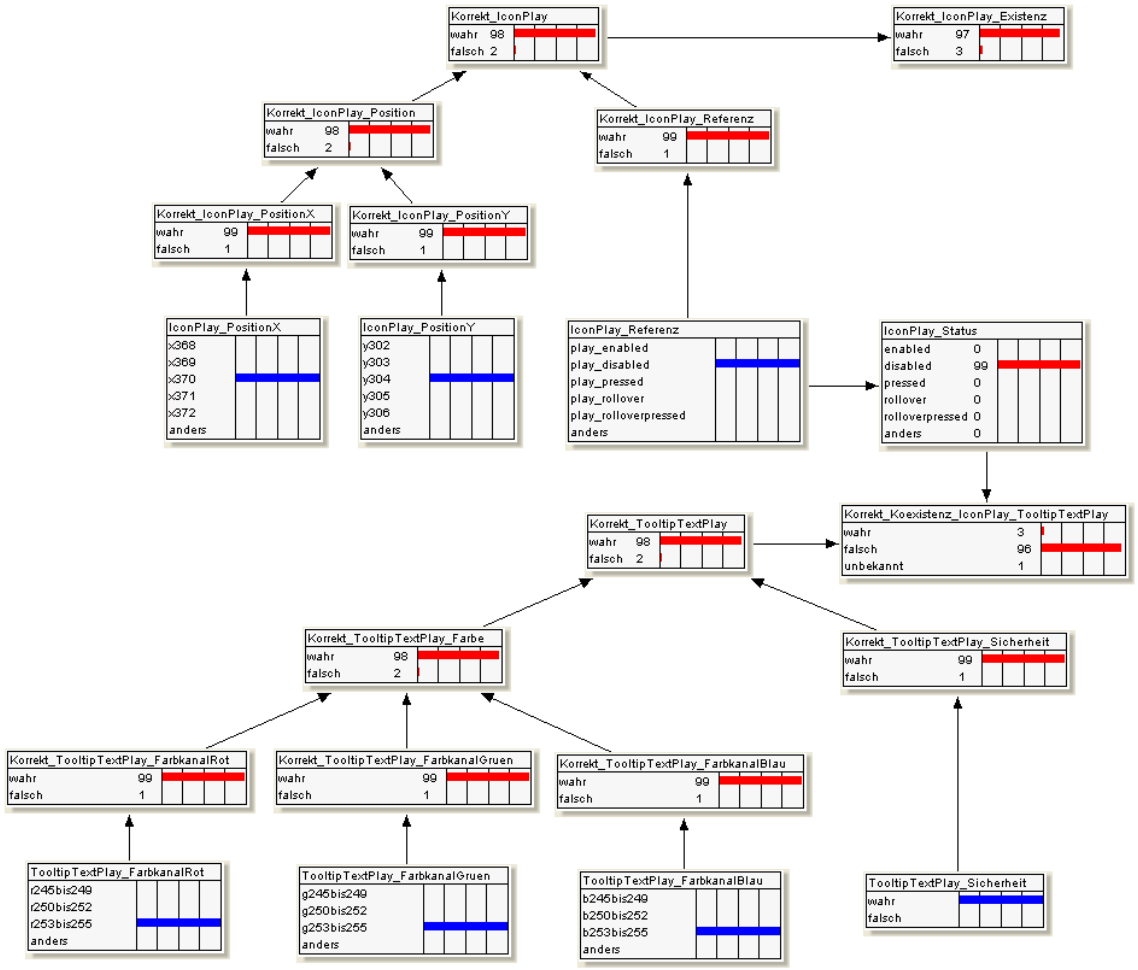


Abbildung 7.12: Bayes'sches Netz für die Verifikation des Wizardmenü-Ausschnitts in Abb. 7.6. Wahrscheinlichkeiten der einzelnen Variablen bei Propagation der Soll-Merkmale. Die Kombination des erkannten Iconzustands und der Existenz des Textelements führt zu einer falschen Koexistenz. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010))

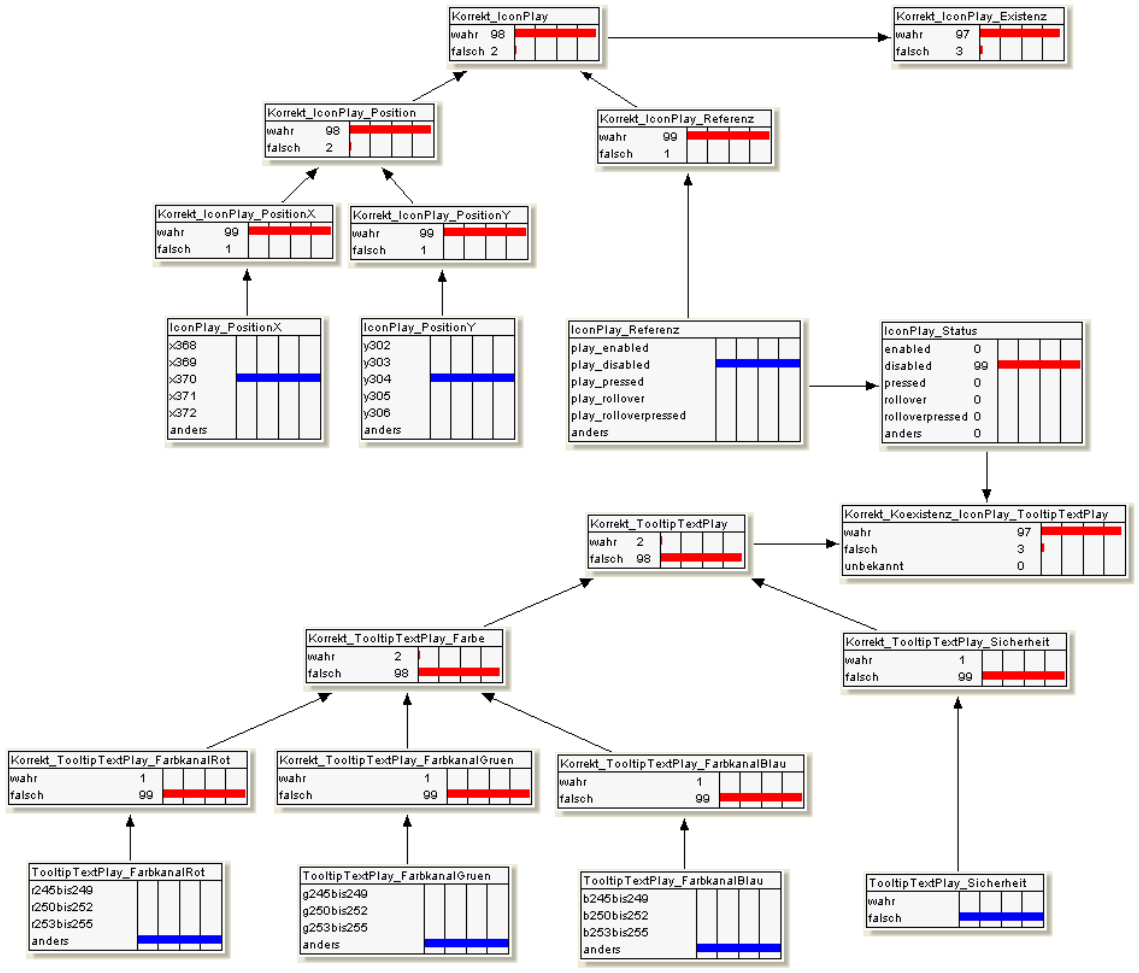


Abbildung 7.13: Bayes'sches Netz für die Verifikation des Wizardmenü-Ausschnitts in Abb. 7.6. Wahrscheinlichkeiten der einzelnen Variablen bei Propagation der Soll-Merkmale für das Symbolelement. Im Ist-Design existiert kein Textelement, somit ist die Koexistenz bei Iconzustand *disabled* korrekt. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010))

7.3.3 Erweiterungen

7.3.3.1 Font

In Kap. 6.2 wurde darauf hingewiesen, dass die verwendete Bildverarbeitung von Schad 2011 nicht den Font von Textelementen identifizieren kann. Der zukünftige Einsatz bestehender Fonterkenner (z.B. *WhatTheFont*) ist hier möglich. Ebenfalls können Fontinformationen auf einfache Weise in das Bayes'sche Netz integriert werden. Zum einen kann der Font als Ganzes in das Netz eingegeben werden, d.h. als eine MV. Zum anderen kann er in Komponenten zerlegt werden (vgl. Kap. 2.2.2), wie in Abb. 7.14 dargestellt. Die Art der Verarbeitung ist abhängig davon, welche Informationen die Bildverarbeitung liefert und wie detailliert die Erklärungen für Designfehler sein sollen. Im Fall eines Designfehlers im Font wird bei der ersten Variante lediglich ausgegeben werden können, dass der Font falsch ist, während bei der zweiten Variante detailliert auf die Korrektheit der einzelnen Komponenten bezuggenommen werden kann. Wie bei allen anderen MVs können hier auch weiche anstelle von harten Evidenzen eingegeben werden.

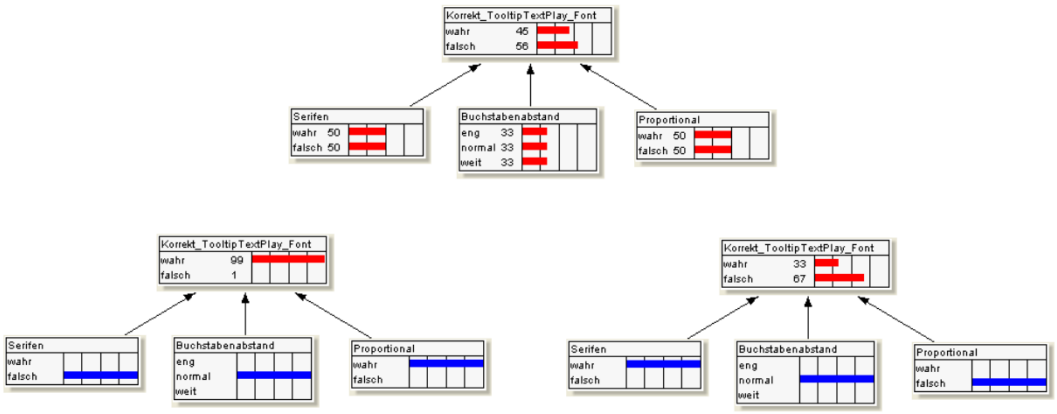


Abbildung 7.14: Bayes'sches Netz für die Verifikation des Merkmals *Font* des Elements *TooltipTextPlay*. Oben: der Initialzustand; Unten links: die Wahrscheinlichkeitsverteilungen bei Eingabe der Sollwerte; Unten rechts: fehlerhafte Eingaben für die Merkmalskomponenten *Serifen* und *Proportional*. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010))

Im Beispiel entspricht ein proportionaler Font ohne Serifen und mit normalem Buchstabenabstand dem Soll. Diese drei Merkmale können auf mehrere Fonts zutreffen und geben damit keine Garantie, dass es sich tatsächlich um den spezifizierten Font handelt. Die Grenze liegt in der Menge der Einzelmerkmale, in die ein Font aufgesplittet werden kann. Dem Bayes'schen Netz können stets weitere Merkmale zur Differenzierung hinzugefügt werden.
Die Kombination der Merkmalskomponenten ist im CPT des Beispiels wieder so definiert, dass sie sich gleichsam auf das Ergebnis auswirken. Evtl. sind gerade bei der Font-Verifikation aber andere Gewichtungen sinnvoller. Im Beispiel ist der Font noch zu 33% korrekt, obgleich nur der Buchstabenabstand dem Soll entspricht. Die Merkmale *Serifen* und *Proportional* stellen in der Regel aber gravierendere Entscheidungsmerkmale für einen Font dar, als das Merkmal *Buchstabenabstand*.

7.3.3.2 Erweiterung um Zustand *unbekannt*

Wenn nicht alle Merkmale des Ist-Designs vorliegen (z.B. aufgrund fehlender Bildverarbeitung), das Bayes'sche Netz diese aber fordert, müssen die jeweiligen MVs einen weiteren Zustand *unbekannt* anbieten. Diese Information muss dann bis zu den VVs propagiert werden, d.h. alle direkten und indirekten Kindknoten der MVs müssen auch um diesen Zustand erweitert werden.
Ein anderer Fall tritt ein, wenn ein im Soll enthaltenes Element im Ist-Design gar nicht existiert. Beim bereits in Abschnitt 7.3.2.6 vorgestellten Beispiel aus Abb. 7.13 wird dies gehandhabt, indem für das im Ist-Design fehlende Textelement `TooltipTextPlay` im Bayes'schen Netz alle zugehörigen MVs auf den Zustand *anders* gesetzt werden. Das ist allerdings gleichbedeutend mit „Das erkannte Textelement entspricht überhaupt nicht den Sollwerten des Elements *TooltipTextPlay*". D.h. im System muss bekannt sein, ob ein Ist-Element existiert oder nicht, um die Ausgabe der VV `Korrekt_TooltipTextPlay` richtig zu werten. Besser ist auch hier die Hinzunahme eines weiteren Zustands. Für beide Fälle reicht ein Zustand *unbekannt* aus: Wenn sämtliche Merkmale eines Elements unbekannt sind, kann dies so gewertet werden, als würde das Element nicht existieren. Das Verifikationsergebnis ist in diesem Fall völlig unbekannt.

7.4 Begründung und Einordnung

Im Folgenden werden die grundlegenden Überlegungen bei der Konzeption des Bayes'schen Netzes diskutiert und begründet. Dabei werden auch Alternativen zu den verwendeten Vorgehensweisen besprochen.

7.4.1 Verwendung eines Bayes'schen Netzes

Wie in Kap. 4.2 erläutert, eignet sich ein Bayes'sches Netz für den Umgang mit unsicheren Informationen, so wie sie bei dem durch Bildverarbeitung gewonnenen Ist-Design vorkommen. Auch die Zustandsmengen können im Bereich der Designverifikation vollständig identifiziert werden, womit auch diesbzgl. die Verwendung eines wahrscheinlichkeitstheoretischen Ansatzes unproblematisch ist.
In Kap. 4.2 wurden noch weitere Formalismen für das Schlussfolgern mit Unsicherheit betrachtet. Dies wirft die Frage auf, warum ein Bayes'sches Netz und kein anderer Formalismus für die Designverifikation zum Einsatz kommt.

7.4.1.1 Vorteile Bayes'scher Netze

Ein ausschlaggebendes Kriterium bei der Entscheidung für ein Bayes'sches Netz war, dass im Gegensatz zu vielen anderen Ansätzen Bayes'sche Netze auf einer mathematischen Grundlage basieren. Durch die klare mathematische Definition ist das Schlussfolgern in Bayes'schen Netzen kontrollierbar, deterministisch und validierbar, was einen problemlosen produktiven Einsatz maßgeblich unterstützt. Da die Unsicherheiten auf Wahrscheinlichkeiten abgebildet werden, sind die Ergebnisse auch weniger subjektiv, was Fehlinterpretationen vermeidet. Durch Einschränkungen können diese Eigenschaften auch für manche anderen Formalismen gelten; z.B bei der Fuzzy-Logik mit der Forderung, dass die Unschärfe als Wahrscheinlichkeit interpretierbar ist. Das erfordert allerdings die Definition zusätzlicher Regeln und gestaltet den Umgang um einiges komplexer.
Daneben ist ein weiterer Vorteil Bayes'scher Netze ihre grafische Repräsentation, die effiziente Berechnungen und Transparenz ermöglicht. Der Aufbau einer Erklärungskomponente ist für grafische Modelle einfacher. Zudem stehen für Bayes'sche Netze bei Bedarf zahlreiche Erweiterungen bereit, die meist den Anspruch haben mathematisch konsistent zu bleiben (vgl. 5.2.3 und 5.7).

7.4.1.2 Bayes'sche Netze versus Fuzzy-Logik

Vor allem für die Abgrenzung zur Fuzzy-Logik ist zu definieren, um welche Art Unsicherheit es sich bei der Designverifikation handelt.
Bildverarbeitungsmethoden liefern an sich exakte Werte, bspw. Pixelposition $x = 10$. Der gemessene Wert kann aber vom erwarteten abweichen, ohne dass es sich um einen tatsächlichen Fehler handelt, da z.B. durch das Rendering Schwankungen auftreten können. Zudem sind manche Bildverarbeitungsmethoden wie Symbolerkenner sehr komplex und betrachten mehrere Einzelmerkmale, die wiederum nicht als sicher angenommen werden können. D.h. die Werte an sich werden genau angegeben, aber ihre Sicherheit hängt von der Güte des Bildes und der Bildverarbeitungsmethoden ab. Die Sicherheit kann als Wahrscheinlichkeit interpretiert werden. Die bei der MMI-Testautomatisierung eingesetzten Symbol- und Texterkenner liefern bereits Wahrscheinlichkeiten zurück. Für die Designverifikation bedeutet das, dass leichte Abweichungen von erwarteten Werten nicht als sicher falsch gewertet werden sollten. Die Behandlung der Unsicherheit als Wahrscheinlichkeit, z.B. „Die Beobachtung A gilt mit Wahrscheinlichkeit 0.8", führt zum probabilistischen Schließen, für das ein Bayes'sche Netz einen geeigneten Formalismus darstellt. Wäre hingegen das unscharfe Schließen und damit die Fuzzy-Logik ein geeignetes Verfahren, müssten die Beobachtungen an sich als unscharf betrachtet werden bzw. sich nur verbal und nicht mit exakten Werten formulieren lassen, z.B. „A ist ziemlich sicher". Für Bilderkenner wird Fuzzy-Logik erfolgreich eingesetzt, indem die Bildinhalte an sich als unscharf angesehen werden, bspw. ab wann eine Farbabstufung eine Kante darstellt (s. z.B. Nischwitz, Fischer und Haberäcker 2007, Kap. 33.3 und Tizhoosh 1998). Allerdings entstehen dabei vage Ergebnisaussagen wie „A ist nahezu kreisrund", was eine weitere Interpretation bzw. die Verwendung von Schwellwerten bedarf. In Bayes'schen Netzen hingegen wird Unsicherheit in Form von Wahrscheinlichkeiten bis zum Endergebnis kombiniert und propagiert und das vorzeitige Verwenden von Schwellen ist nicht erforderlich.
Auch bei der Designverifikation könnte eine Korrektheitsaussage als Unschärfe realisiert werden. Eine Verifikation sollte jedoch eine konkrete Aussage über die Korrektheit liefern und nicht eine unscharfe, z.B. „Element A ist ziemlich falsch". Die Interpretation ist dabei sehr subjektiv von jeweiligem Anwender abhängig, was einen weiteren indeterministischen Faktor darstellt. Genau das erschwert die Kontrollierbarkeit von Fuzzy-Systemen. In Anwendungsbereichen, in denen die Aus-

sagen tatsächlich ungenau oder sogar widersprüchlich sind, ist der Einsatz von Fuzzy-Logik jedoch gerechtfertigt (s. Nischwitz, Fischer und Haberäcker 2007, Kap. 33.2.1). Das trifft für die Designverifikation aber nicht zu.
Daneben kann die verwendete Bildverarbeitung als irrelevant für die Auswertung ihrer Ergebnisse betrachtet werden; es sollte keine Rolle spielen, ob durch z.B. Fuzzy-Logik oder Neuronale Netze ein Symbol erkannt wurde. Bei der Designverifikation ist nur die Weiterverarbeitung und Kombination dieser Informationen entscheidend. Eine leichte Abweichung vom Soll kann als unsichere Beobachtung angesehen werden und sollte nicht als Vagheit interpretiert werden.

7.4.2 Effizienz

Obgleich hier immer im Singular als „Bayes'sches Netz“ bezeichnet, besteht das Netz aus mehreren Teilnetzen, die nur teilweise miteinander verknüpft sind. Die einzelnen Teilnetze für die Verifikation von lokalen und globalen Merkmalen bilden jeweils eine Baumstruktur. Für eine Baumstruktur sind, wie in Kap. 5.2.1 erläutert, sehr effiziente exakte Inferenzberechnungen möglich. Die Kantenrichtungen wurden bewusst nach Ursache-Wirkung-Beziehung gewählt. Auf diese Weise ist ein einfaches Setzen von weichen Evidenzen in die MVs möglich. Der Aufbau eines kausalen Bayes'schen Netzes ist zudem eine typische Herangehensweise und entspricht damit auch der Anleitung aus Kap. 8.8.2 in [Spreckelsen und Spitzer 2008] zum Aufbau eines konsistenten Bayes'schen Netzes.
Die Abbildungen 7.15, 7.16 und 7.16 veranschaulichen diese Eigenschaften an einem Ausschnitt des Bayes'schen Netzes zur Verifikation eines MMI-Wizardmenüs. Der Ausschnitt umfasst die Verifikation der lokalen und globalen Merkmale von zwei Icons und ihres jeweiligen Tooltips. Als Erweiterung zum Beispiel in Abschnitt 7.3 sind hier auch die Teilnetze zur Verifikation des Abstands zwischen einem Icon und dem dazugehörigen Tooltip sowie die Verifikation der Koexistenz der Iconzustände dargestellt. Insbesondere in Abbildung 7.16 sind die einzelnen Teilnetze, ihre Baumstruktur und die Kantenrichtungen gut zu erkennen. Die Abbildung 7.17 zeigt nur die Variablen der Ausgabeschicht sowie ihre direkten Elternknoten.

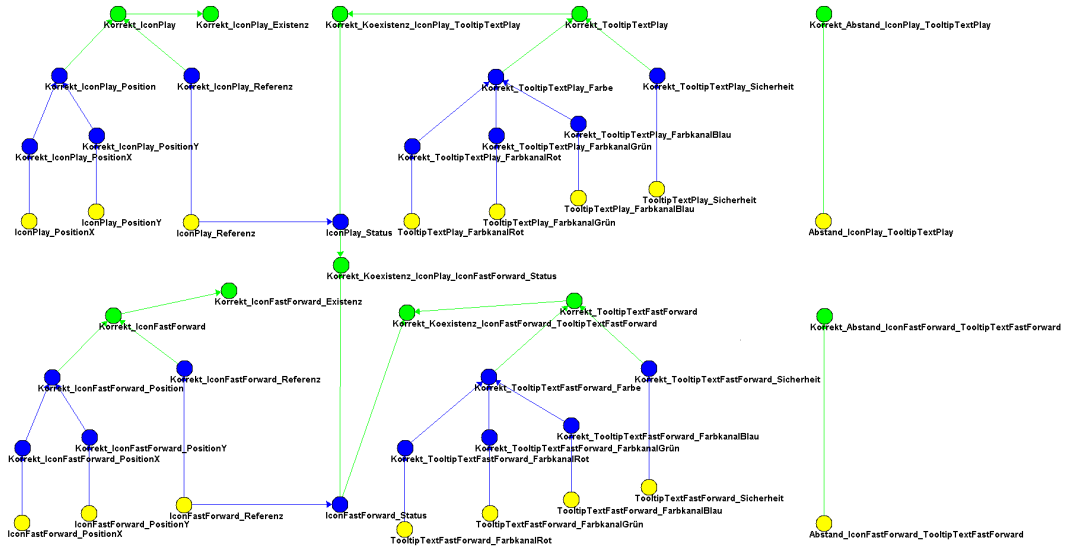


Abbildung 7.15: Bayes'sches Netz zur Verifikation der lokalen und globalen Merkmale zweier Icons und ihrer dazugehörigen zwei Tooltips eines MMI-Wizardmenüs. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001))

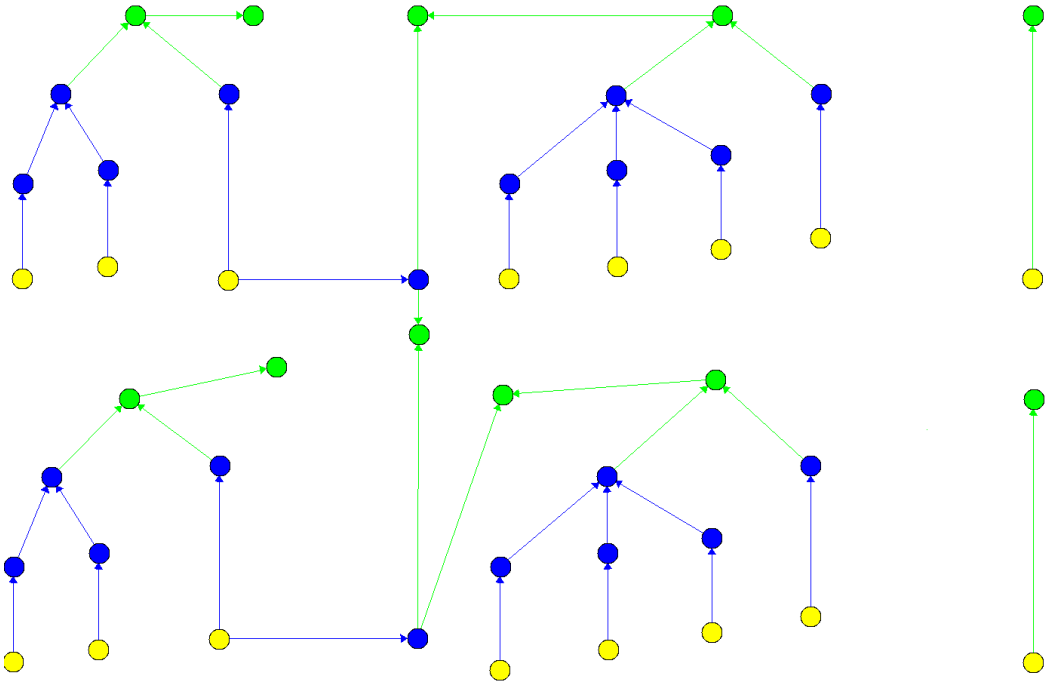


Abbildung 7.16: Dasselbe Bayes'sche Netz aus Abbildung 7.15 ohne Beschriftung der Variablen. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001))

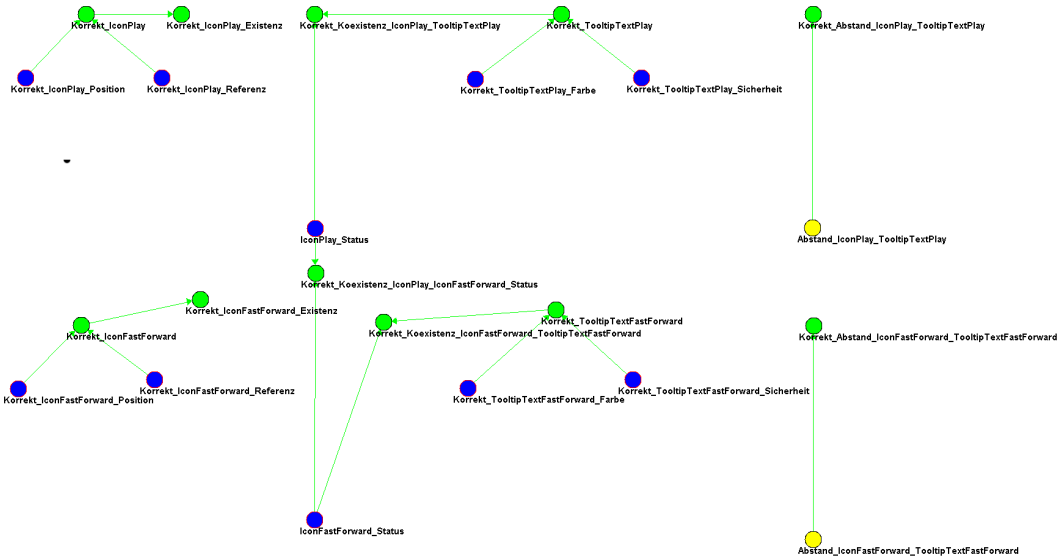


Abbildung 7.17: Dasselbe Bayes'sche Netz aus Abbildung 7.15 in dem nur die Variablen der Ausgabeschicht und deren direkten Elternknoten eingeblendet sind. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001))

7.4.3 Behandlung von Unsicherheit

7.4.3.1 Harte und weiche Evidenz

Bei der Designverifikation existieren zwei verschiedene Formen von Unsicherheit, die unterschiedlich behandelt werden müssen. In der Regel handelt es sich bei den lokalen und globalen Merkmalen eines Ist-Elements um harte Evidenzen. Für die Eingabe dieser bieten die MVs eine vollständige Zustandsmenge an. Daneben liefern Bilderkenner meist Sicherheiten zu ihren Ergebnissen, die als Wahrscheinlichkeiten interpretiert werden können. Diese Form von Unsicherheit wird als weiche Evidenz in die MVs eingegeben, die das Merkmal *Sicherheit* bzw. *Referenz* repräsentieren. Da die MVs generell keine Elternknoten besitzen, können die weichen Evidenzen stets auf einfache Weise eingegeben werden, indem die Apriori-Wahrscheinlichkeiten entsprechend angepasst werden (vgl. Kap. 5.2.3). Virtuelle Variablen, um Likelihood-Verhältnisse einzugeben und zu kombinieren, sind nicht notwendig.

7.4.3.2 Aufteilung der Wahrscheinlichkeiten

Bei Eingabe einer harten Evidenz gibt es nur eine Möglichkeit, wie die Wahrscheinlichkeiten auf die Zustände verteilt werden: Der Zustand, der die harte Evidenz repräsentiert, erhält die Sicherheit 1.0 und alle übrigen Zustände erhalten die Sicherheit 0.0. Bei Eingabe von weichen Evidenzen gibt es mitunter mehrere Möglichkeiten. Im obigen Beispiel wird die negative Wahrscheinlichkeit ($P(negativ) = 1-P(Soll)$) gleichverteilt über die übrigen Zustände. Wenn es nur die zwei Zustände *wahr* und *falsch* gibt, so wie im Beispiel beim Merkmal *Sicherheit*, ist die Verteilung $P(wahr) = P(Soll)$, $P(falsch) = P(negativ)$ die einzig mögliche. Beim Merkmal *Referenz* gibt es aber in der Regel mehr als zwei Zustände und die Gleichverteilung der negativen Wahrscheinlichkeit ist nicht die einzige Möglichkeit. Z.B. können die Referenzen stärker gewichtet werden als der Zustand *anders*. Die umgekehrte Möglichkeit macht auch Sinn, da normalerweise dem Zustand *anders* viel mehr mögliche Referenzen zugeordnet sind und sich damit die Wahrscheinlichkeit dieser summiert. Verschiedene Sichtweisen sind hier möglich und eine komplett richtige Aufteilung gibt es nicht. Für die aktuelle Implementierung wurde die Gleichverteilung gewählt, da strenggenommen alle anderen Referenzen *anders* sind und es nicht zwangsläufig der Fall sein muss, dass sich Referenzen eines Icons stärker ähneln als Referenzen verschiedener Icons. Existiert zudem der Zustand *unbekannt*, muss erneut überlegt werden, wie die negativen Wahrscheinlichkeiten verteilt werden.
Darüber hinaus wird aktuell nur das Erkennerergebnis mit der höchsten Sicherheit in das Bayes'sche Netz eingegeben. Möglich wäre aber auch, weitere, weniger sichere Ergebnisse einzugeben. Falls die Bildverarbeitung fehlerhaft ist oder die Bildqualität zu schlecht, könnte es vorkommen, dass das tatsächliche Element nicht die höchste Sicherheit besitzt. In diesem Fall müsste pro Referenz eine separate MV existieren mit den Zuständen *wahr* und *falsch*. Außerdem schließen sich die in einer MV definierten Referenzen gegenseitig aus (vgl. Axiomatik vom Bayes'schen Netz in Kap. 5.1.3), was jedoch bei Erkennersicherheiten in der Regel nicht zutrifft[1]. Die Normierung der Erkennersicherheiten würde hier auf einfache Weise Abhilfe schaffen. Bislang ist die Betrachtung der höchsten Erkennersicherheit allerdings völlig ausreichend. Falls dies zukünftig nicht mehr der Fall sein sollte, kann das Bayes'sche Netz aber mit den eben genannten Möglichkeiten verändert werden.

[1] Ein Symbolerkenner kann z.B. mit 99% Sicherheit das Symbol *A* erkennen und zu 96% Sicherheit das Symbol *B*.

7.4.3.3 Gewichtung von Soll-Abweichungen

Das Ist-Design kann durch Bildstörungen mit Unsicherheit behaftet sein, weswegen Abweichungen vom Soll nicht immer komplett als Designfehler gewertet werden dürfen. Die Wahl der Zustände für die MVs orientiert sich an dem Sollwert des jeweiligen Merkmals und der gewünschten Gewichtung der Abweichungen. Durch den Zustand *anders* wird, wenn nicht bereits gegeben, die Vollständigkeit der Zustandsmenge gewährleistet. EVs und VVs besitzen in der Regel die zwei Zustände *wahr* und *falsch*, um Auskunft über die Übereinstimmung zwischen dem Soll und dem Ist eines Merkmals oder eines Elements zu geben. Falls erforderlich, kann noch ein Zustand *unbekannt* hinzugefügt werden, um fehlende oder nicht-auswertbare Informationen zu kennzeichnen. Damit ist die Wahl der Zustände sehr flexibel und kann individuell gewählt werden.
Auch die Parameter der CPTs können flexibel gewählt werden. Auf diese Weise ist es möglich, die Gewichtung von Abweichungen für jedes Merkmal individuell einzustellen.

7.4.4 Kombination der Wahrscheinlichkeiten

Im Bayes'schen Netz in Abschnitt 7.3 konnte festgestellt werden, dass es bei den Inferenzberechnungen zur Abschwächung der Wahrscheinlichkeiten kommen kann. Der Grund hierfür ist vor allem die Verwendung von nicht exakt 0 und 1 in den CPTs. Für effiziente Berechnungen sollen zudem große CPTs vermieden werden, weswegen vermittelnde Variablen eingesetzt werden. Dadurch wird die Tiefe des Netzes erhöht und es kommt vermehrt zu Kombinationen der Wahrscheinlichkeiten. Die Einflussnahme zwischen zwei Variablen ist dabei umso schwächer, je länger der Pfad zwischen den Variablen ist. Die folgenden Beispiele in den Abbildungen 7.18 und 7.25 sollen dies veranschaulichen. Die in diesen Netzen verwendeten Parameter sind in den Tabellen 7.4 und 7.5 aufgeführt. Damit sind die CPTs wie bisher definiert, sodass die Evidenzen der MVs die Ergebnisse in den direkten Kindknoten zu gleichen Teilen beeinflussen. Diese Einflussnahme sollte möglichst auch bis zur VV propagiert werden. Z.B. sollen ein falsches Merkmal und 11 korrekte Merkmale zu der Wahrscheinlichkeit $P(wahr) = \frac{1}{11}$ für die VV führen. Da nicht exakt 0 und 1 verwendet wird, kann dies allerdings nur annähernd erreicht werden.

Kombination|Variable1, Variable2, Variable3	**wahr**	**falsch**
wahr, wahr, wahr	0.99	0.01
wahr, wahr, falsch	0.66	0.34
wahr, falsch, wahr	0.66	0.34
wahr, falsch, falsch	0.33	0.67
falsch, wahr, wahr,	0.66	0.34
falsch, wahr, falsch	0.33	0.67
falsch, falsch, wahr	0.33	0.67
falsch, falsch, falsch	0.01	0.99

Tabelle 7.4: CPT, der die Kombination dreier binären Variablen definiert.

Kombination|Variable1, Variable2	**wahr**	**falsch**
wahr, wahr	0.99	0.01
wahr, falsch	0.5	0.5
falsch, wahr	0.5	0.5
falsch, falsch	0.01	0.99

Tabelle 7.5: CPT, der die Kombination zweier binären Variablen definiert.

Die Abbildungen 7.18 und 7.19 zeigen zwei Bayes'sche Netze *A* und *B*, die eine hohe Tiefe und damit einen hohen Grad an Kombinationen aufweisen. Wie an der Wahrscheinlichkeitsverteilung der Variablen `VV` zu sehen ist, tritt nur bei dem Netz in Abbildung 7.19 eine Abschwächung auf. Offensichtlich spielt auch die Art der Kombination eine entscheidende Rolle.

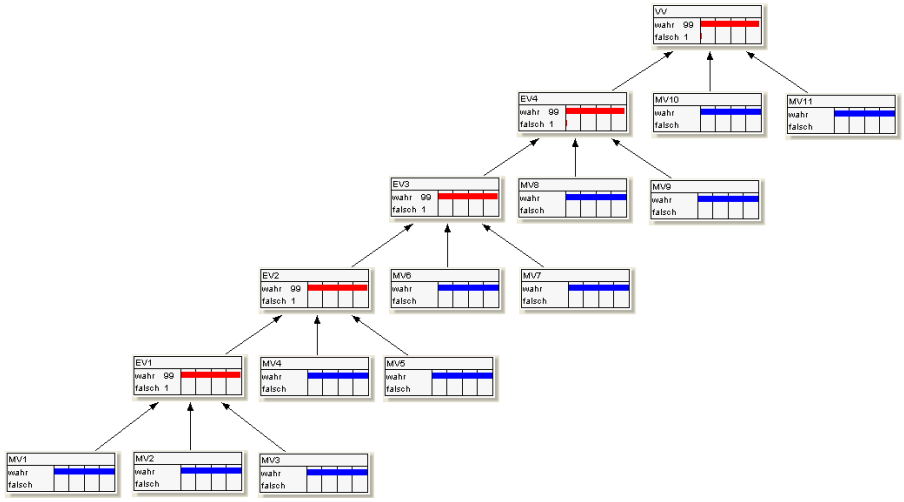


Abbildung 7.18: Bayes'sches Netz *A* zur Verdeutlichung der Kombination von Wahrscheinlichkeiten. Die Wahrscheinlichkeitsverteilung der Ausgabevariablen `VV` stimmt hier exakt mit den Wahrscheinlichkeitsverteilungen der Eingabevariablen `MV1` bis `MV11` überein. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010))

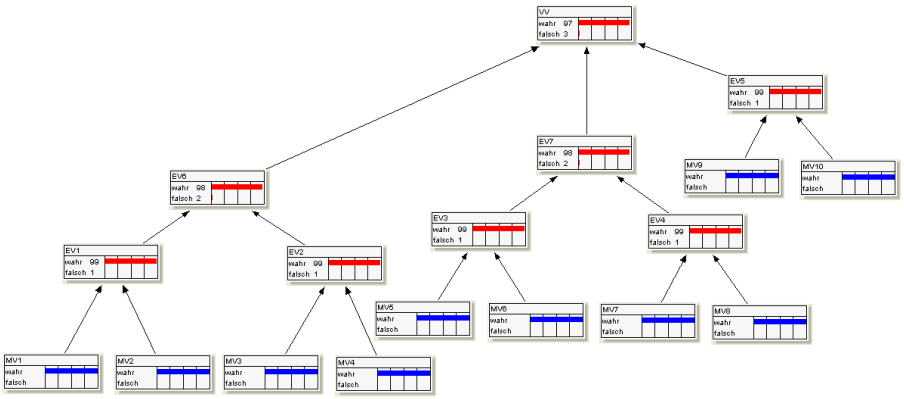


Abbildung 7.19: Bayes'sches Netz *B* zur Verdeutlichung der Kombination von Wahrscheinlichkeiten. Bei dieser Art der Kombination weist die Wahrscheinlichkeit des Zustands *wahr* der Ausgabevariablen `VV` eine Abschwächung gegenüber der Wahrscheinlichkeiten der Eingabevariablen `MV1` bis `MV10` auf. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010))

Beim Setzen der Eingabevariable `MV1` auf *falsch* im Netz *A*, dargestellt in Abb. 7.20, zeigt sich, dass sich dies nur zu 1% auf das Ergebnis in `VV` auswirkt. Bei Netz *B* hingegen beträgt die Wahrscheinlichkeit für den Zustand *falsch* in der Variablen

VV genau 10%, wenn MV5 auf *falsch* gesetzt wird (s. Abb. 7.21). Das entspricht der gewünschten anteiligen Kombination.

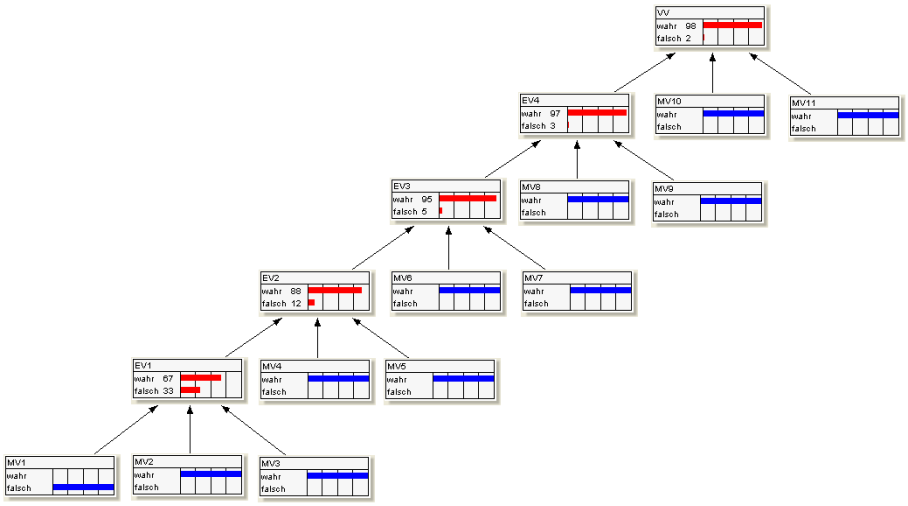


Abbildung 7.20: Bayes'sches Netz *A*: Mit Evidenz für Zustand *falsch* bei Variable MV1 verringert sich die Wahrscheinlichkeit des Zustands *wahr* bei Variable VV nur minimal. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010))

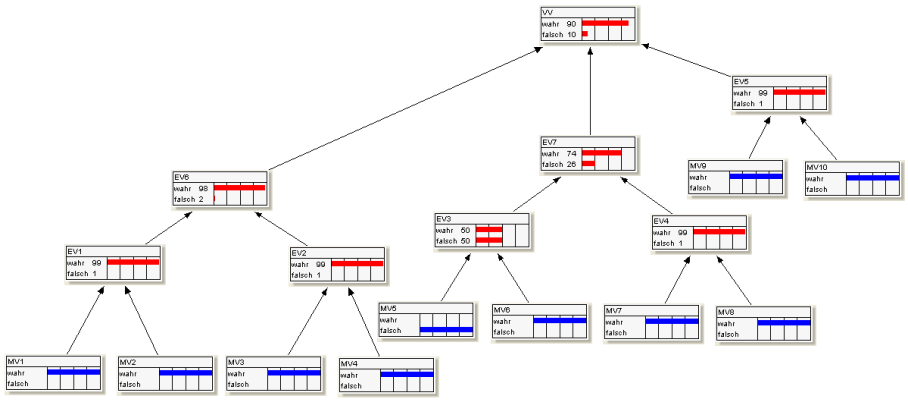


Abbildung 7.21: Bayes'sches Netz *B*: Mit Evidenz für Zustand *falsch* bei Variable MV5 verringert sich die Wahrscheinlichkeit des Zustands *wahr* bei Variable VV auf 90%, was der gewünschten Gewichtung der Merkmale entspricht. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010))

Die Ergebnisse sind allerdings abhängig davon in welcher Tiefe sich die MVs im Netz befinden. Bei der Konstellation in Abb. 7.22 entspricht das Ergebnis für Netz *B* nicht mehr der gewünschten Gewichtung, während das bei Abb. 7.23 für

Netz *A* nun der Fall ist. Das liegt daran, dass bei der Inferenzberechnung in einem Bayes'schen Netz nur die direkten Elternknoten betrachtet werden; wie viele Vorgänger-Variablen bereits zum Ergebnis beigetragen haben, fehlt als Information.

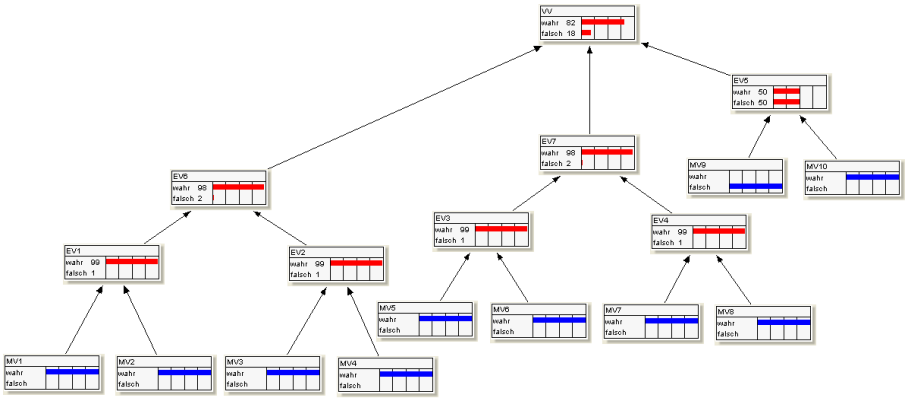


Abbildung 7.22: Bayes'sches Netz *B*: Mit Evidenz für Zustand *falsch* bei Variable `MV9` verringert sich die Wahrscheinlichkeit des Zustands *wahr* bei Variable `VV` auf 82%. Das ist eine stärkere Verringerung, als der gewünschten Gewichtung der Merkmale entspricht. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010))

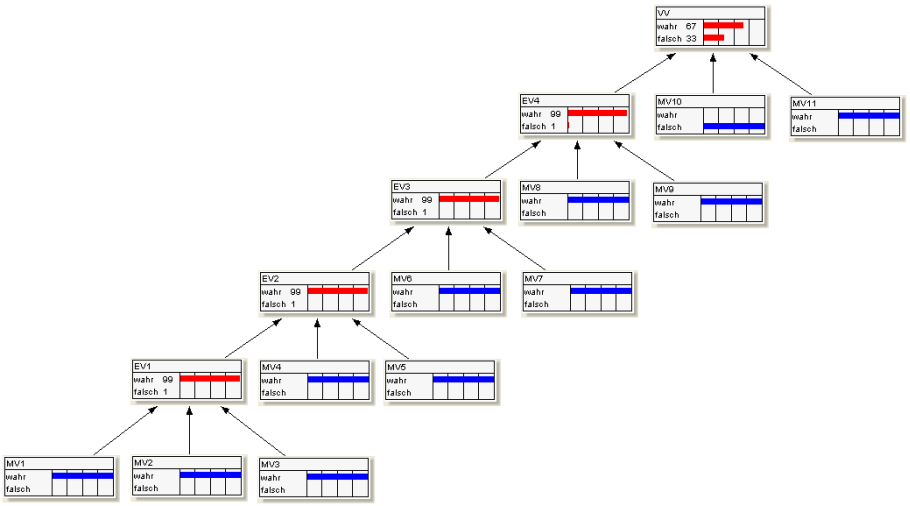


Abbildung 7.23: Bayes'sches Netz *A*: Mit Evidenz für Zustand *falsch* bei Variable `MV10` verringert sich die Wahrscheinlichkeit des Zustands *wahr* bei Variable `VV` auf 67%, was der gewünschten Gewichtung der Merkmale entspricht. (Netz erstellt mit [*BayesBuilder* 2007-2010])

So kann aus den Wahrscheinlichkeitsverteilungen der VVs auch nicht zwingend

auf die eingegebenen Evidenzen geschlussfolgert werden, was auch die Abbildungen 7.24 und 7.25 zeigen. Der Formalismus Bayes'scher Netze sieht die Weiterleitung solcher Informationen nicht vor.

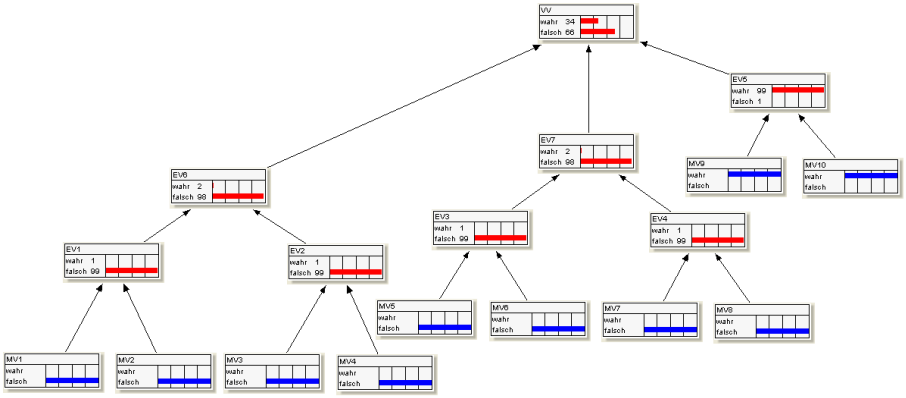


Abbildung 7.24: Bayes'sches Netz *B*: Mit Evidenz für Zustand *falsch* bei den Variable `MV1` bis `MV8` verringert sich die Wahrscheinlichkeit des Zustands *wahr* bei Variable `VV` auf 34%. Das widerspricht der gewünschten Gewichtung der Merkmale, die eine Verringerung auf 20% definiert. (Netz erstellt mit [*BayesBuilder* 2007-2010])

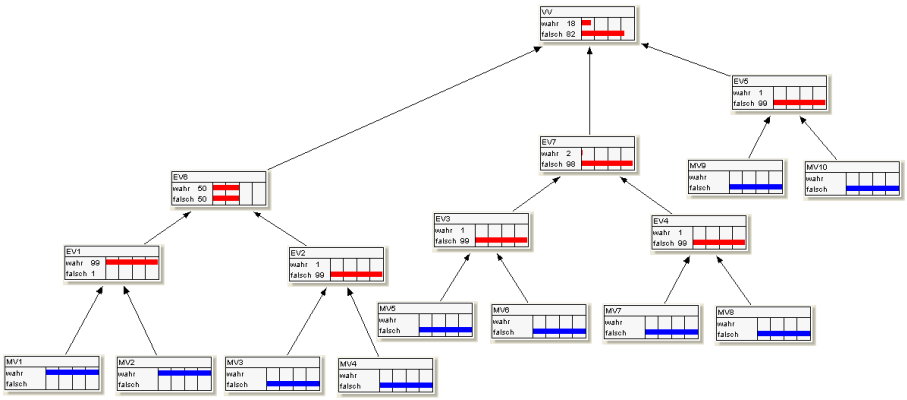


Abbildung 7.25: Bayes'sches Netz *B*: Mit Evidenz für Zustand *falsch* bei den Variable `MV1` bis `MV8` verringert sich die Wahrscheinlichkeit des Zustands *wahr* bei Variable `VV` auf 18%. Das widerspricht der gewünschten Gewichtung der Merkmale, die eine Verringerung auf 20% definiert. (Netz erstellt mit [*BayesBuilder* 2007-2010])

Die relativ groben Werte 0.99 und 0.01 wurden gewählt, um den Einfluss bei Kombinationen zu verdeutlichen. Die Verwendung von 0.999 und 0.001 oder noch feineren Werten ist geeigneter; dann führt die Kombination nicht ganz so schnell zu

einer Abschwächung. Je mehr Nachkommastellen die Werte besitzen, umso mehr Rechenungenauigkeiten können aber auftreten.
Neben der Kombination der Korrektheitsaussagen für die lokale Verifikation eines Elements oder einer globalen Verifikation für Elementgruppen, ist auch die Kombination dieser VVs denkbar um schließlich eine VV zu erhalten, die Auskunft über die Korrektheit des gesamten Ist-Designs gibt. Diese Gesamtkorrektheit ist aber für den praktischen Einsatz nicht relevant und würde zudem eine hohe Rechenzeit beanspruchen. Bei der Verifikation von GUI-Design muss das Verfahren vor allem detaillierte Aussagen über die Korrektheit einzelner Elemente liefern können.

7.4.5 Merkmalseingabe

Für jedes erkannte Element müssen die lokalen Merkmale in die MVs jedes Netzes für die lokale Verifikation eingegeben werden, um Übereinstimmungen mit dem Soll festzustellen. Im Beispiel in Abschnitt 7.3 gibt es pro Elementtyp (Text oder Symbol) nur jeweils ein solches Netz. Ein Bayes'sches Netz, dass die lokale Verifikation aller acht Icons des betrachteten Wizardmenüs umfasst, beinhaltet auch acht Teilnetze für die lokale Verifikation jedes Icons. Die Merkmale eines Ist-Elements müssen dann in alle acht Teilnetze eingegeben werden. Das bedeutet einen erheblichen mehrfachen Aufwand.
Eine Alternative wäre, nur eine Schicht MVs bereitzustellen, die Kindknoten in die Erklärungsschichten aller relevanten Teilnetze beinhalten. In diesem Fall müssen die Merkmale pro Ist-Element nur einmal gesetzt werden. Trotzdem ist diese Möglichkeit nicht praktikabel. Für die Zustandsmengen der MVs bedeutet das nämlich einen enormen Zuwachs, da die für alle Soll-Elemente relevanten Merkmale abgedeckt werden müssen. Das wiederum resultiert in viel größeren CPTs. Wie eigene Experimente gezeigt haben, nehmen Inferenzberechnungen mit sehr großen CPTs deutlich mehr Zeit in Anspruch, als das Setzen von Evidenzen für eine größere Anzahl von Variablen. Das Setzen der Evidenzen kann zudem auf einfache Weise parallelisiert werden.

7.4.6 Verifikation globaler Merkmale

Bislang werden globale Verifikationen nur vorgenommen, wenn die lokalen Merkmale der beteiligten Elemente ein korrektes Design aufweisen. Diese Vorgehens-

weise ist effizient, da bei fehlerhaften lokalen Design die Inferenzberechnungen für die globalen Verifikationen nicht mehr durchgeführt werden. Des Weiteren macht sie auch Sinn, da globale Verifikationen oft zu unsinnigen oder falschen Ergebnissen führen, wenn die lokalen Merkmale nicht korrekt sind. Bspw. wird der Abstand zwischen zwei Elementen meist nicht richtig sein, wenn die Position des einen Elements falsch ist.
Die obige Vorgehensweise erfordert eine zusätzliche Logik im System. Vor jeder globalen Verifikation muss geprüft werden, ob die notwendigen lokalen Elemente korrekt sind. Alternativ ist es auch möglich, alles im Bayes'schen Netz zu ermitteln. Dafür könnte bei jeder globalen Verifikation das Ergebnis der Existenz-Variablen der geforderten Elemente einfließen. Wenn ein Element eine sehr niedrige Existenz-Wahrscheinlichkeit hat, führt das bei der Korrektheit des globalen Merkmals auch zu niedrigen Werten. Daraus kann dann ebenfalls geschlussfolgert werden, dass die globale Verifikation falsch ist aufgrund falscher lokaler Merkmale.
Die Einbeziehung der Existenz-Variablen führt zu größeren CPTs bei den globalen Verifikationen und damit zu komplexeren Inferenzberechnungen. Theoretisch ist in einem Bayes'schen Netz alles kombinierbar, was aber nicht unbedingt Sinn macht. Eine zusätzliche Logik, die bei fehlerhaften lokalen Elementen von vornherein die globale Verifikation verhindert, ist die effizientere Vorgehensweise.

7.4.7 Interpretation der Ergebnisse

Die Wahrscheinlichkeitsangaben alleine reichen nicht aus, um zu entscheiden, ob z.B. eine Übereinstimmung zum Soll mit Sicherheit 90% ein korrektes Design bedeutet oder nicht. Für die Interpretation der Wahrscheinlichkeitsverteilungen der VVs muss eine Schwelle gesetzt werden. Bspw. kann festgelegt werden, dass alle Merkmale, deren Korrektheit mindestens 90% beträgt, als korrekt eingestuft werden. Die Höhe der Schwelle muss für den jeweiligen Anwendungsbereich oder evtl. auch speziell für das jeweilige Merkmal geeignet gewählt werden. Bei der Verifikation von GUI-Design, das durch störungsbehaftete Kameraaufnahmen dargestellt wird, sollte die Schwelle niedriger sein als bei Framegrabber-Aufnahmen. Daneben beeinflusst die gewählte Gewichtung der Soll-Abweichungen das Ergebnis und damit die Wahl der Schwellwerte.
Die Interpretation der Wahrscheinlichkeiten ist zudem auch abhängig von der Art der Kombinationen (s. Abschnitt 7.4.4. Für ein intuitives Verständnis der Wahr-

scheinlichkeiten sollte eine möglichst flache Kombinationshierarchie verwendet werden, sodass es zu möglichst wenig Abschwächung kommt. Ist der Aufbau des Bayes'schen Netzes und damit die Art der Kombinationen bekannt, können mit Hilfe dieses Wissens allerdings auch die Ergebnisse geeignet interpretiert werden. Diese Vorgehensweise kann jedoch recht komplex werden, wie man auch in den vorgestellten Beispielen in Abschnitt 7.4.4 sieht. Nicht nur die Art der Kombination, sondern auch die Verteilung der Evidenz hat bei Bayes'schen Netzen mit einer großen Tiefe einen entscheidenden Einfluss auf das Ergebnis.

7.4.8 Korrektheit

Für den produktiven Einsatz des Designverifikationssystems ist die Korrektheit ein wesentlicher Punkt. Die Arten der Verifikationsfehler können in zwei Fragestellungen formuliert werden:

- Wird ein Merkmal oder ein Element als richtig verifiziert, obwohl es in Wirklichkeit falsch ist?
- Wird ein Merkmal oder ein Element als falsch verifiziert, obwohl es in Wirklichkeit richtig ist?

Die beiden Situationen entsprechen den in Kap. 5.4.2.4 vorgestellten Gütekriterien *Falsch-Positiv-Rate* und *Falsch-Negativ-Rate* für die Gütebestimmung von Bayes'schen Netzen.
Die Grundlage für eine korrekte Designverifikation bilden die Fehlerfreiheit von Ist und Soll-Design sowie die korrekte Modellierung des Bayes'schen Netzes. Falls die Bildverarbeitung nicht jedes geforderte Ist-Merkmal liefert oder die Kombination eingegebener Ist-Merkmale keine Korrektheitsaussage ermöglicht, können zudem durch die Verwendung des Zustands *unbekannt* fehlerhafte oder unsinnige Ergebnisse vermieden werden.
Bezugnehmend auf Abschnitt 7.4.4 können auch durch eine hohe Anzahl an Kombinationen der Korrektheitsaussagen beide oben genannten Fehlersituationen eintreten. Durch die Abschwächung der Wahrscheinlichkeiten kann die Gesamtwahrscheinlichkeit für die Korrektheit eines Elements sinken und dann evtl. als Designfehler interpretiert werden. Im Gegensatz dazu kann ein Fehler in einem Merkmal auch einen zu geringen Einfluss auf die Gesamtkorrektheit haben, sodass er

nicht bemerkt wird. Damit ist die Art der Kombination entscheidend für die Güte des Designverifikationsverfahrens und die Kombinationsanzahl sollte möglichst gering sein.
Eine weitere Ursache für Verifikationsfehler ist die Verwendung ungeeigneter Parameter. Wie bereits erläutert, weist Ist-Design, das durch einen Framegrabber erzeugt wurde, wesentlich weniger Störungen auf als Kamerabilder. Bei der Verifikation von störungsarmen Bildern müssen die Soll-Abweichungen stärker und bei störungsreichen Bildern schwächer als Designfehler gewichtet werden. Wird ein Bayes'sches Netz, das für störungsreiches Ist-Design konzipiert ist, für störungsarme Bilder verwendet, werden einige tatsächliche Designfehler abgeschwächt und als durch Störungen verursachte Abweichungen gewertet. Im umgekehrten Fall werden viele störungsbedingte Soll-Abweichungen als Designfehler gewertet. Je nach Anwendungsbereich müssen daher die verwendeten Parameter geeignet sein, um korrekte Verifikationsergebnisse zu erhalten.

7.4.9 Erweiterbarkeit

Je nach Inhalt der Designspezifikation können zur Abbildung anderer Merkmale weitere Merkmalsvariablen modelliert werden. Die Erweiterung des Bayes'schen Netzes um neue Merkmale ist dabei auf einfache Weise möglich (s. z.B. Hinzufügung des Fonts in Abschnitt 7.3). Ebenso können den Variablen weitere Zustände hinzugefügt werden, bspw. der Zustand *unbekannt*.
Die in diesem Kapitel dargestellte Verifikation innerhalb eines Screenbereichs bedeutet im Übrigen keine Einschränkung für die Verifikation globaler Merkmale. Globale Merkmale können auch zwischen Elementen verschiedener Screenbereiche verifiziert werden. Die Vorgehensweise ist dann wie folgt: Zuerst werden pro Screenbereich alle lokalen Merkmale verifiziert und somit die Zuweisungen zwischen Ist- und Soll-Elementen vorgenommen. Für korrekt Elemente können dann die globalen Merkmale des jeweiligen Screenbereichs verifiziert werden. Wenn die Merkmale aller Screenbereiche verifiziert sind, können die bereichsübergreifenden globalen Merkmale verifiziert werden.

7.5 Schlussfolgerung

In Abschnitt 7.1 wurde der Aufbau eines Bayes'schen Netzes zur Designverifikation und in Abschnitt 7.2 der Verifikationsprozess mit diesem abstrakt erläutert. Die Netzstruktur spiegelt die Zusammenhänge zwischen den Merkmalen der Elemente des Ist- und des Soll-Designs wider. Das Bayes'sche Netz weist dafür drei verschiedene Arten von Variablen auf, die die Eingabe-, Vergleichs- und Ausgabeschicht bilden. Schließlich besteht die Inferenzkomponente, die das Design einer gesamten GUI verifiziert, aus einer Vielzahl an Bayes'schen Netzen, die nur vereinzelt über Kanten miteinander verbunden sind. Alle Teilnetze für eine lokale Verifikation eines Elements oder die globale Verifikation eines Merkmals bilden eine Baumstruktur, für die effiziente Inferenzberechnungen möglich sind.
Das konkrete Beispiel in Abschnitt 7.3 zeigt, dass eine Designverifikation mit dem entworfenem Bayes'schen Netz funktioniert. Dabei wurde eine einheitliche Gewichtung der Soll-Abweichungen und eine anteilige Kombination der Korrektheitsaussagen verwendet, die Rückschluss auf fehlerhafte Merkmale liefert. Abschnitt 7.4 liefert Begründungen für das konzipierte Bayes'sche Netz und alternative Möglichkeiten. Es wurde erläutert, wie die unsicheren Informationen des Ist-Designs im Bayes'schen Netz als harte und weiche Evidenzen geeignet gehandhabt werden. Da die MVs Wurzelknoten sind, ist für die Eingabe von weichen Evidenzen keine virtuelle Variable notwendig. Das konzipierte Bayes'sche Netz ist zudem sehr flexibel und erweiterbar; sowohl die Gewichtung von Abweichungen als auch die Kombination von Wahrscheinlichkeiten kann individuell für jedes Element und jedes Merkmal eingestellt werden. Insbesondere aufgrund der großen Flexibilität muss auch die Korrektheit des Bayes'schen Netzes sichergestellt werden, was in Abschnitt 7.4.8 erläutert wurde. Um spärliche CPTs zu vermeiden, werden keine Einträge exakt 0 und 1 verwendet. Abschnitt 7.4.4 diskutierte die damit verbundene Abschwächung von Wahrscheinlichkeiten bei der Kombination. Die bei der Designverifikation aktuelle Anzahl an Merkmalen und die damit verbundene notwendige Tiefe des Bayes'schen Netzes ist akzeptabel und führt nur zu schwachen Abschwächungen. Der Einsatz eines Bayes'schen Netzes wurde in Abschnitt 7.4.1.2 insbesondere gegenüber der Fuzzy-Logik begründet. Bayes'sche Netze haben auch ihre Stärken und Schwächen. Da sie im Vergleich zu anderen Formalismen aber einige entscheidende Vorteile aufweisen, die auch gerade für den produktiven Einsatz in der Testautoma-

tisierung eine Rolle spielen, stellen sie ein geeignetes Inferenzverfahren dar.
Die Entwicklung des Bayes'schen Netzes für das vorgestellte Verifikationsverfahren spiegelt die in Kap. 5.3 geschilderte typische Herangehensweise wider: In diesem Kapitel wurde das Bayes'sche Netz manuell mittels der Software *BayesBuilder* (*BayesBuilder* 2007-2010) entwickelt. Die Parameter sind ebenfalls aus Überlegungen heraus entstanden und sind „educated guesses"(dt. fundierte Vermutungen; s. Wiegerinck, Kappen und Burgers 2010, S. 6). Das Netz ist relativ klein und umfasst nur einen geringen Teil des Designs einer vollständigen MMI-GUI. Um die Verifikation einer kompletten GUI zu ermöglichen, wird in Kap. 8 darauf eingegangen, wie auf Basis einer formalen Designspezifikation eine automatische Netzgenerierung möglich ist. In Kap. 9 wird anschließend die Alternative, das datenbasierte Lernen der Netzstruktur, diskutiert.

Kapitel 8

Generierung des Bayes'schen Netzes aus einer Spezifikation

Im vorhergehenden Kapitel wurde ein Bayes'sches Netz vorgestellt und gezeigt, dass mit diesem die geforderte Verifikation von GUI-Design realisierbar ist. Für den produktiven Einsatz ist eine automatische Generierung des Bayes'schen Netzes erforderlich. Eine Möglichkeit ist die automatische Generierung aus einer formalen Spezifikation, so wie sie in Kap. 5.3.1 theoretisch erläutert wurde. In diesem Kapitel wird ein konkretes Spezifikationsformat in JSON (s. Kap. 3.3) entwickelt. Hierzu werden im ersten Abschnitt grundlegende Überlegungen formuliert. Daran anschließend wird in den Abschnitten 8.2 und 8.3 das JSON-Spezifikationsformat vorgestellt und gezeigt, wie daraus ein Bayes'sches Netz generiert werden kann. Am Beispiel der Beschreibung des GUI-Designs eines MMI-Wizardmenüs wird in Abschnitt 8.4 die Praxistauglichkeit des entwickelten Spezifikationsformats nachgewiesen. Das Kapitel schließt mit einer Begründung der gewählten Vorgehensweise und einer kurzen Zusammenfassung.

8.1 Grundlegende Anforderungen und Überlegungen

Das Design grafischer Benutzeroberflächen wird von Designern normalerweise mit grafischen Programmen wie Adobe Photoshop spezifiziert (vgl. Kap. 3.4.3). Informationen, die mit diesen Programmen nicht definierbar sind, werden in textueller Form beschrieben. Beides ist nicht formal und kaum maschinenlesbar. Das Design der MMI-GUI ist bislang in dieser Form spezifiziert (s. Kap. 2.2.2.3). Darüber hin-

aus unterstützen alle existierenden formalen GUI-Spezifikationssprachen die Definition von GUI-Design nicht ausreichend (vgl. Kap. 3.4). Für die formale Beschreibung des MMI-Soll-Designs muss daher ein neues Spezifikationsformat entwickelt werden.

8.1.1 Allgemeine Anforderungen

Es ergeben sich die folgenden Anforderungen, die eine formale Spezifikation des Soll-Designs für die Designverifikation erfüllen muss (vgl. Kap. 6.1):

1. **Lesbarkeit und Verwendbarkeit**: In erster Linie soll durch die Spezifikation eine Automatisierung ermöglicht werden, d.h. die Spezifikation muss maschinenlesbar sein. Für die Wart- und Wiederverwendbarkeit bietet es sich zudem an, dass die Spezifikation auch für den Menschen einfach lesbar ist. Da eine GUI einen hierarchischen Aufbau besitzt, soll die Spezifikationssprache eine hierarchische Struktur unterstützen. Eine hierarchische Gliederung ist auch im Sinne von [Marwedel 2007, Kap. 2.1], um eine komplexe Beschreibung für den Menschen verständlich zu gestalten.

2. **Effizienz**: Heutige MMI-Systeme sind in ihrer GUI sehr umfangreich und für zukünftige Entwicklungen ist eine noch höhere Komplexität zu erwarten. Daher muss die Spezifikation eine kompakte Beschreibung des Soll-Designs ermöglichen, die wenig Speicher benötigt. Ebenfalls muss ein effizienter Transfer des Wissens in die Inferenzkomponente gewährleistet sein. Dabei ist es förderlich, wenn die Struktur der Spezifikation zu der eines Bayes'schen Netzes passt und der Zugriff auf einzelne Designinformationen effizient durchgeführt werden kann.

3. **Vollständigkeit**: Um der Forderung gerecht zu werden, alle designspezifischen Fehler zu finden, muss die Spezifikation das in Kap. 2.2.2 beschriebene MMI-GUI-Design im vollen Umfang repräsentieren.

4. **Flexibilität und Erweiterbarkeit**: Im Rahmen dieser Arbeit liegt der Fokus auf der GUI eines MMI. Die entwickelte Spezifikation sollte dennoch universell für verschiedene GUIs eingesetzt werden können. Damit muss ein Spezifikationsformat gewählt werden, das GUI-Design in seiner Gesamtheit

repräsentieren und Informationen aus verschiedenen Wissensquellen vereinheitlichen kann. Auch aus Sicht der MMI-Designverifikation ist diese Anforderung sinnvoll, da das Aussehen von GUIs zukünftiger MMI-Systeme unbekannt ist.

5. **Korrektheit**: Die Spezifikation muss das Soll-Design korrekt abbilden; andernfalls wird auf Basis einer falschen Vorgabe verifiziert.

Alle genannten Punkte sind entscheidend für den erfolgreichen produktiven Einsatz in der MMI-Testautomatisierung. Insbesondere die vier erstgenannten Anforderungen werden in dieser Arbeit berücksichtigt. Dass die Spezifikation zudem korrekt sein muss, liegt auf der Hand. Wie die Korrektheit der Spezifikation geprüft und gewährleistet werden kann, wird hier nicht im Detail evaluiert.

8.1.2 Struktur der formalen Soll-Designspezifikation

Eine GUI weist einen hierarchischen Aufbau auf; die MMI-GUI gliedert sich bspw. in mehrere Views, die wiederum in Bereiche unterteilt sein können (vgl. Kap. 2.2.1). Jede View bzw. jeder Bereich kann eine Menge von grafischen Elementen beinhalten, jedes Element besitzt lokale Merkmale und zwischen Elementen können globale Merkmale existieren. Daher ist es vorteilhaft auch die Spezifikation hierarchisch zu gliedern.
Die MMI-GUI besitzt ein grundlegendes Layout, das nahezu in jeder View enthalten ist (s. Kap. 2.2.1). Aus diesem Grund soll die Spezifikation *Masterviews* unterstützen. Inhalte, die in mehreren Views enthalten sind, werden nur einmalig in einer Masterview definiert und Redundanzen werden vermieden. Auch die Referenzierung einmalig definierter Bereiche und Elemente verhindert Redundanzen. Für die Definition globaler Merkmale sollte bspw. auf bereits für lokale Merkmale definierte Elemente referenziert werden können. Die Referenzierung erfordert eine eindeutige Namensgebung für Masterviews, Bereiche und Elemente. Im Folgenden ist die Grundstruktur einer GUI-Designspezifikation aufgeführt, die auf diesen Überlegungen basiert.

```
GUI
  Bereiche
    Bereich1
      EindeutigerName
    ...
  Masterviews
    Master1
```

```
      EindeutigerName
      BereichsReferenz
      Element1
        EindeutigerName
        lokalesMerkmal1
        lokalesMerkmal2
        ...
      ...
    ...
  Views
    View1
      MasterReferenz
      BereichsReferenz
      Element1
        EindeutigerName
        lokalesMerkmal1
        lokalesMerkmal2
        ...
      ...
    ...
  GlobaleMerkmale
    GlobalesMerkmal1
      ElementReferenz1
      ...
    ...
```

Im Gegensatz zu allen übrigen lokalen und globalen Merkmalen, die in einer Spezifikation direkt als Wert definiert werden können (z.B. ein Abstand in Pixel und eine Farbe in RGB-Werten), müssen Referenzbilder zu Symbolelementen als Verweis auf Bilddateien angegeben werden. D.h. neben dem Spezifikationsdokument müssen alle im GUI-Design enthaltenen Symbolelemente in Form von Bildern an einem definierten Speicherort zugänglich sein. Der Dateipfad definiert die eindeutige Zuweisung zwischen Referenzbild und Symbolelement. Das Wissen über die Referenzbilder ist auch für die Bildverarbeitung zur Erkennung des Ist-Designs erforderlich. Nur wenn sowohl die Soll-Spezifikation, als auch die Bildverarbeitung dieselbe Menge von Referenzbildern nutzen, kann ein Vergleich zwischen Soll- und Ist-Symbolen korrekt erfolgen.

8.1.3 Netzgenerierung

Allein aus einer Spezifikation des Soll-Designs kann noch kein Bayes'sches Netz generiert werden. Als weitere Wissensquelle muss eine grundlegende Netzstruktur gegeben sein, in die die Soll-Designinformationen eingefügt werden. Die Grundstruktur gibt die Variablen, die Variablenbeziehungen und das Aussehen der Zustandsmengen vor. In Kap. 7.1 wurde eine solche Grundstruktur konzipiert. Die Zustandsmengen aller MVs sind mit *{<Soll>, anders}* definiert, wobei *<Soll>* einen Platzhalter für den in der Spezifikation definierten Sollwert für das jeweilige Merkmal darstellt. EVs und VVs werden mit der Zustandsmenge *{wahr, falsch}* angelegt.

Alle CPTs von EVs und VVs beinhalten initial nur die Werte 0 und 1. Die Apriori-Wahrscheinlichkeiten aller MVs sind gleichverteilt.
Jede Elementdefinition in der Soll-Designspezifikation führt zu einem Teilnetz für die lokale Verifikation und jede globale Merkmalsdefinition zu einem Teilnetz für eine globale Verifikation. Die Soll-Abweichungen und ihre Gewichtung fehlen in dem entstandenen Bayes'schen Netz, da weder die Soll-Designspezifikation noch die Grundstruktur diese Informationen liefert. Darüber hinaus müssen auch die CPTs für die Kombination von Variablen und damit die Gewichtung von Korrektheitsaussagen angepasst werden, sofern nicht die bestehenden CPTs, die nur Nullen und Einsen aufweisen, verwendet werden sollen. D.h. es ist noch weiteres *Zusatzwissen* erforderlich, um ein funktionierendes Bayes'sches Netz zu erhalten.
Um das Bayes'sche Netz automatisch zu generieren, müssen die Grundstruktur und das Zusatzwissen ebenfalls formal spezifiziert sein. Es ist ein Parser erforderlich, der die Informationen aus den Wissensquellen kombiniert und daraus ein vollständiges Bayes'sches Netz erstellt.

8.2 Soll-Designspezifikation in JSON

Die unter den Punkten 1. und 2. aufgeführten Anforderungen *Lesbarkeit, Verwendbarkeit* und *Effizienz* weist die weit verbreitete Auszeichnungssprache XML als Eigenschaften auf (s. Kap. 3.2). Noch besser erfüllt diese Anforderungen das Datenaustauschformat JSON (s. Kap. 3.3 und 3.3.4), das aus diesem Grund für die Beschreibung des Soll-Designs verwendet wird. Das im Folgenden vorgestellte JSON-Schema weist die in Abschnitt 8.1.2 vorgeschlagene hierarchische Struktur und Inhalte auf.

8.2.1 Lokales Design

Das JSON-Schema zur Beschreibung des Aufbaus einer GUI und ihrer lokalen Designmerkmale ist in Quellcode 8.1 aufgeführt.

Quellcode 8.1: JSON-Schema für die Spezifikation lokaler Designmerkmale

```
{
  "description":"Design of a GUI",
  "type":"object",
  "properties": {
    "GUIName": {
      "type": "string",
```

```
      "description": "Name of the GUI"
  },
  "Dimension": {
    "type": "object",
    "description": "Width and height of the GUI",
    "properties": {
      "Width": {
        "type": "integer" },
      "Height": {
        "type": "integer" }
    }
  },
  "Areas": {
    "type": "array",
    "description": "Defined areas within the GUI",
    "optional": true,
    "minItems": 1,
    "items": {
      "type": "object",
      "properties": {
        "Name": {
          "type": "string" },
        "Dimension": {
          "$ref" : "#/properties/Dimension",
          "optional": true
        },
        "UpperLeftPosition": {
          "type": "object",
          "description": "Upper left position of the area",
          "optional": true,
          "properties": {
            "X": {
              "type": "integer" },
            "Y": {
              "type": "integer" }
          }
        },
        "CenterPosition": {
          "type": "object",
          "description": "Center position of the area",
          "optional": true,
          "properties": {
            "X": {
              "type": "integer" },
            "Y": {
              "type": "integer" }
          } } } }
  },
  "Masters": {
    "type": "array",
    "description": "Defined master views within the GUI",
    "optional": true,
    "minItems": 1,
    "items": {
      "type": "object",
      "properties": {
        "Name": {
          "type": "string" },
        "AreaReference": {
          "type": "string",
          "optional": true },
        "Symbols": {
          "type": "array",
          "optional": true,
          "minItems": 1,
          "items": {
            "type": "object",
            "properties": {
              "Name": {
                "type": "string" },
```

```
            "Optional": {
            "Type": {
              "type": "string"
              "optional": true },
            "Dimension": {
              "$ref": "#/properties/Dimension",
              "optional": true
            },
            "UpperLeftPosition": {
              "$ref": "#/properties/UpperLeftPosition",
              "optional": true
            },
            "CenterPosition": {
              "$ref": "#/properties/CenterPosition",
              "optional": true
            },
            "Path": {
              "type": "string",
              "description": "Path of the reference symbol on disk",
              "optional": true
            },
            "Color": {
              "type": "object",
              "description": "RGB color of the element",
              "optional": true,
              "properties": {
                "Red": {
                  "type": "integer",
                  "minimum": 0,
                  "maximum": 255
                },
                "Green": {
                  "type": "integer",
                  "minimum": 0,
                  "maximum": 255
                },
                "Blue": {
                  "type": "integer",
                  "minimum": 0,
                  "maximum": 255
                } } } } }
        },
        "Texts": {
          "type": "array",
          "optional": true,
          "minItems": 1,
          "items": {
            "type": "object",
            "properties": {
              "Name": {
                "type": "string" },
              "Optional": {
                "type": "boolean",
                "optional": true },
              "Dimension": {
                "$ref" : "#/properties/Dimension",
                "optional": true
              },
              "UpperLeftPosition": {
                "$ref": "#/properties/UpperLeftPosition",
                "optional": true
              },
              "CenterPosition": {
                "$ref": "#/properties/CenterPosition",
                "optional": true
              },
              "Font": {
                "type": "object",
                "description": "Font of the text element",
                "optional": true,
```

```
                "properties": {
148               "Name": {
                    "type": "string" },
                  "Style": {
                    "type": "string" },
                  "Size": {
153                 "type": "integer" },
                  "Bold": {
                    "type": "boolean" },
                  "Italic":  {
                    "type": "boolean" }
158             }
              },
              "Color": {
                "$ref": "#/Masters/Symbols/Color"
              } } } } } }
163 },
  "Views": {
    "type": "array",
    "description": "Defined views within the GUI",
    "minItems": 1,
168 "items": {
      "type": "object",
      "properties": {
        "Name": {
          "type": "string" },
173     "AreaReference": {
          "type": "string",
          "optional": true },
        "MasterReference": {
          "type": "string",
178       "optional": true },
        "Symbols": {
          "$ref": "#/Masters/Symbols"
        },
        "Texts": {
183       "$ref": "#/Masters/Texts"
        } } } }
}
```

In den Zeilen 5 bis 17 werden mit einem Namen und der Angabe der Dimension allgemeingültige Eigenschaften einer GUI definiert. Da für diese Elemente keine Optionalität angegeben ist, sind sie per Default obligatorisch. In dem Array `Areas` (ab Zeile 19) können für die GUI verschiedene Screenbereiche definiert werden (vgl. Kap. 2.2.1). Die Definition von Screenbereichen ist optional (s. Zeile 22). Auch die Definition von Master-Views im Array `Masters` (ab Zeile 55) ist optional. Für die Angabe der Position gibt es die Schlüsselwörter `UpperLeftPosition` und `CenterPosition`, die die obere linke Ecke und den Mittelpunkt der Bounding-Box[1] des Elements beschreiben. Die Position bezieht sich dabei stets auf das Koordinatensystem höchster Ebene, d.h. das der GUI mit Ursprung in der oberen, linken Ecke.

[1]Eine Bounding-Box ist ein Rechteck, das ein zweidimensionales grafisches Element minimal umschließt. Die Achsen der hier verwendeten Bounding-Box sind parallel zu den Koordinatenachsen. Die Beschreibung einer Bounding-Box beinhaltet die vier Werte *obere, linke x-Koordinate*, *obere, linke y-Koordinate*, *Breite* und *Höhe*.

Wird ein Array zur Spezifikation in der JSON-Datei definiert, so gibt das Schema vor, dass es auch mindestens ein Element beinhalten muss (s. `"minItems": 1` in z.B. Zeile 167).
Die vorgestellte Schemadatei stellt lediglich ein Beispiel dar, um GUI-Design zu definieren. Je nach Art und Komplexität der GUI sind evtl. weitere Elemente oder auch weitere Einschränkungen notwendig. Z.B. sind die lokalen Merkmale innerhalb der Arrays `Symbols` und `Texts` bislang nur als einfache Objekte vorgesehen. Stattdessen kann ein lokales Merkmal aber auch als ein Array repräsentiert werden, um mehrere Möglichkeiten definieren zu können. Bspw. kann so jeweils ein Pfad in Abhängigkeit des Elementzustands (Schlüsselwort `State`) angegeben werden:

```
"Paths": {
  "type": "array",
  "description": "Paths of the reference templates representing different states of the symbol",
  "optional": true,
  "minItems": 1,
  "items": {
    "type": "object",
    "properties": {
      "Path": {
        "type": "string" },
      "State": {
        "type": "string" }
    } }
}
```

8.2.2 Globales Design

Globale Merkmale werden innerhalb einer View oder eines Master mit weiteren Arrays definiert, der Quellcode 8.2 zeigt das JSON-Schema. Die betreffenden Elemente müssen zuvor mit ihren lokalen Merkmalen definiert worden sein, damit der Verweis auf diese möglich ist.

Quellcode 8.2: JSON-Schema für die Spezifikation globaler Designmerkmale

```
"Alignments": {
  "type": "array",
  "description": "Alignments of the referenced elements",
  "optional": true,
  "minItems": 1,
  "items": {
    "type": "object"
    "properties": {
      "ElementReference": {
      "type": "string" },
      "Alignment": {
        "enum": ["left", "center", "right"] }
    } }
},
"Distances": {
  "type": "array",
  "description": "Distances between two elements",
  "optional": true,
```

```
  "minItems": 1,
  "items": {
    "type": "object"
    "properties": {
      "ElementReference1": {
        "type": "string" },
      "ElementReference2": {
        "type": "string" },
      "Axis": {
        "enum": ["X", "Y"] },
      "Relation": {
        "enum": ["sideBySide", "enclosing"] },
      "ReferencePoint": {
        "enum": ["border", "border_top", "border_bottom", "border_left", "border_right", "center"] },
      "Distance": {
        "type": "integer" }
    } }
},
"Coexistences": {
  "type": "array",
  "description": "Coexistences of 2 elements at minimum",
  "optional": true,
  "minItems": 1,
  "items": {
    "type": "object"
    "properties": {
      "ElementReferences": {
        "type": "array",
        "minItems": 2,
        "items": {
          "type": "string" }
      } } }
},
"Mutualexclusions": {
  "type": "array",
  "description": "Mutualexclusions of 2 elements at minimum",
  "optional": true,
  "minItems": 1,
  "items": {
    "type": "object"
    "properties": {
      "ElementReferences": {
        "type": "array",
        "minItems": 2,
        "items": {
          "type": "string" }
      } } }
}
```

Im Array `Alignments` wird pro Objekt die Ausrichtung eines Elements bzgl. des zugehörigen Screenbereichs definiert; die Attribute sind die Elementreferenz (`ElementReference`) und eine der drei Ausrichtungen (`Alignment`) *left*, *center*, *right*. Mit einem Objekt im Array `Distances` wird der Abstand zwischen zwei Elementen definiert; die Attribute sind die Elementreferenzen (`ElementReference1`, `ElementReference2`), die betrachtete Koordinatenachse (`Axis`), die Relation (`Relation`), der Referenzpunkt (`ReferencePoint`) und der Abstand in Pixel (`Distance`). Die Relation beschreibt die Anordnung der Elemente zueinander: Bei *sideBySide* liegen die Elemente nebeneinander, bei *enclosing* wird das zweite Element vom ersten Element umschlossen. Mit Angabe des Referenzpunkts wird fest-

gelegt, zwischen welchen Punkten der Abstand gemessen wird: *border* definiert den kürzesten Abstand zwischen den zwei Elementen und *center* zwischen den Mittelpunkten. Für die Relation *enclosing* können daneben noch *border_top*, *border_bottom*, *border_left* und *boder_right* definiert werden, die den Abstand zwischen den zwei obersten, untersten, am weitesten linken oder am weitesten rechten Punkten beschreiben.

Ein Objekt im Array `Coexistences` definiert zwei oder mehr Elemente, die nur im Zusammenhang existieren. Im Gegensatz dazu definiert ein Objekt im Array `Mutualexclusions` zwei oder mehr Elemente, die sich gegenseitig ausschließen.

Alle Arrays sind optional. Wird allerdings ein Array zur Spezifikation in der JSON-Datei definiert, so gibt das Schema vor, dass es mindestens ein Element beinhalten muss.

Auch hier zeigt das Schema lediglich ein Beispiel. Neben den Namen der betreffenden Elemente können weitere Objekte oder Arrays hinzugefügt werden, um z.B. die betreffenden Elementzustände zu spezifizieren:

```
"Mutualexclusions": {
  "type": "array",
  "description": "Mutualexclusions of 2 elements at minimum",
  "optional": true,
  "minItems": 1,
  "items": {
    "type": "object",
    "properties": {
      "ElementReferences": {
        "type": "array",
        "minItems": 2,
        "items": {
          "type": "object",
          "properties": {
            "ElementReference": "string",
            "States": {
              "type": "array",
              "items": {
                "type": "string" }
            } } } } } }
}
```

8.3 Generierung des Bayes'sche Netzes

Dieser Abschnitt stellt abstrakt die Generierung eines Bayes'schen Netzes aus einer formalen Soll-Designspezifikation mithilfe von Zusatzwissen dar (s. Abschnitt 8.1.3); Abschnitt 8.4 liefert dann ein konkretes Beispiel.

8.3.1 Bayes'sches Netz aus JSON-Spezifikation

Ein Bayes'sches Netz kann formal in dem *Bayesian Interchange Format XML* (BIFXML) (Gagliardi Cozman 1998) beschrieben werden. Als Basis liegt die aus den Abbildungen 7.1 und 7.2 abstrakte Grundstruktur des Bayes'schen Netzes im BIFXML-Format vor. Die in JSON definierten Informationen müssen nun auf diese Struktur abgebildet werden.

8.3.1.1 Lokales Design

In der JSON-Spezifikation werden Elemente sowohl in Masters als auch in Views definiert. Jede Elementdefinition führt zu einem Teilnetz für die lokale Verifikation. Die MVs werden dabei basierend auf den für das jeweilige Element definierten Merkmalen entwickelt. Z. B. ergeben sich für das in JSON definierte Merkmal *Dimension* mit den Merkmalskomponenten *Width* und *Height* die MVs `<Name>_Dimension_Breite` und `<Name>_Dimension_Hoehe`. `<Name>` ist ein Platzhalter für den Namen des in JSON definierten Elements (Schlüsselwort `Name`). Die Zustandsmengen aller Variablen entsprechen den in der Grundstruktur vorgegebenen Verteilungen mit Nullen und Einsen (s. Abschnitt 8.1.3). Wenn für das Element das Schlüsselwort `Optional` in JSON mit dem Wert *false* definiert ist, wird des Weiteren eine Existenz-Variable generiert, ebenfalls mit der vorgegebenen Zustandsmenge.

8.3.1.2 Globales Design

Analog zu den lokalen Merkmalen werden aus den spezifizierten globalen Merkmalen, basierend auf der Grundstruktur, Bayes'sche Netze generiert. Bei den globalen Merkmalen *Koexistenz* und *Gegenseitiger Ausschluss* sind die MVs die bereits generierten lokalen VVs der jeweiligen Elemente (vgl. Kap. 7.1.2). Für alle übrigen globalen Merkmale werden neue MVs angelegt.

8.3.2 Vervollständigung mit Zusatzwissen

Um die Generierung des Bayes'schen Netzes einheitlich zu gestalten, bietet es sich an, auch das Zusatzwissen in JSON zu formulieren. Da das Soll-Design und das Zusatzwissen in der Regel aus verschiedenen Quellen stammen werden (s. Kap.

10), sollte das Zusatzwissen in einer separaten JSON-Datei vorliegen. Das JSON-Schema in Quellcode 8.3 stellt eine mögliche Struktur dar.

Quellcode 8.3: Definition des Zusatzwissens in JSON

```
{
  "name": "Zusatzwissen",
  "Mappings": {
    "type": "array",
    "description": "Converting probabilities",
    "optional": true,
    "minItems": 1,
    "items": {
      "type": "object",
      "properties": {
        "old": "number",
        "new": "number" }
    } },
  "Variances": {
    "type": "array",
    "description": "variances to target design",
    "optional": true,
    "minItems": 1,
    "items": {
      "Elementreferences": {
        "type": "array",
        "minItems": 1,
        "items": {
          "type": "string" }
      }
      "Features": {
        "type": array,
        "description": "Features for which this variance is valid",
        "items": {
          "type": "string" }
      },
      "Tolerances": {
        "type": array,
        "description": "Number of variances and size of intervals",
        "items": {
          "type": "integer" }
      },
      "Probabilities": {
        "type": array,
        "description": "Probabilities of variances",
        "items": {
          "type": "number" }
      } } } } }
  },
  "CombinationSingle": {
    "type": "array",
    "optional": true,
    "items": {
      "type": "number" }
  },
  "CombinationDuad": {
    "type": "array",
    "optional": true,
    "items": {
      "type": "number" }
  },
  "CombinationTriple": {
    "type": "array",
    "optional": true,
    "items": {
      "type": "number" }
  },
  "Existences": {
    "type": "object",
```

```
    "optional": true,
    "properties": {
      "Optional": {
        "type": "array",
        "items": {
          "type": "number" }
      },
      "Obligatory": {
        "type": "array",
        "items": {
          "type": "number" },
      } } }
}
```

Im Array `Mappings` werden allgemeine Umwandlungen für die existierenden Parameter der Grundstruktur definiert. So können z.B. die Einsen in 0.99 und die Nullen in 0.01 konvertiert werden. Das liefert bei binären Variablen konsistente CPTs, deren Werte sich pro Zeile zu 1 summieren. Handelt es sich nicht um binäre Variablen, müssen die neuen Werte von der Software, die das Zusatzwissen anwendet, geeignet verteilt werden.

Im Array `Variances` wird die Gewichtung der Soll-Abweichungen definiert. Pro Element (lokale Verifikation) oder Elementen (globale Verifikation) kann ein Objekt angelegt werden, das für die jeweiligen Merkmale, angegeben im Array `Features`, die Abweichungen festlegt. Dabei beschreibt das Array `Tolerances` mit seiner Länge, wie viele aufeinanderfolgende einzelne Werte oder Intervalle inklusive dem Sollwert separat gewichtet werden sollen. Der Eintrag mit Index i bezieht sich auf das $+/-i$. angrenzende Intervall. Der erste Eintrag vom Typ `integer` ($i = 0$) bezieht sich auf das Intervall, das den Sollwert beinhaltet. Die einzelnen Einträge geben an wie groß die Intervalle sind; z.B. bezeichnet der Eintrag 0 einen einzelnen Wert und der Eintrag 1 ein Intervall, das aus zwei Werten besteht. Das Array `Probabilities` hat dieselbe Länge und definiert die Wahrscheinlichkeiten für den jeweiligen `Tolerances`-Eintrag. Auf diese Weise entsteht für die Gewichtung der Soll-Abweichungen eine Glockenkurve mit Maximum im Sollwert. Die Wahrscheinlichkeit für den obligatorischen Zustand *false* wurde bereits durch die allgemeinen Umwandlungen gesetzt.

In den Zeilen 45 bis 62 werden die Wahrscheinlichkeitsverteilungen für die CPTs definiert, die die Kombination der Korrektheitsaussagen beschreiben. Im Array `CombinationSingle` wird ein CPT für die Abhängigkeit zu einem Elternknoten, in `CombinationDuad` zu zwei und in `CombinationTriple` zu drei Elternknoten definiert. Die Parameter werden in den Arrays in der Reihenfolge angegeben, wie sie auch das BIFXML-Format für die CPTs verlangt. In Abb. 8.1 sind diese Reihen-

V | E1	wahr	falsch
wahr	1	3
falsch	2	4

V | E1, E2	wahr	falsch
wahr, wahr	1	5
wahr, falsch	2	6
falsch, wahr	3	7
falsch, falsch	4	8

V | E1, E2, E3	wahr	falsch
wahr, wahr, wahr	1	9
wahr, wahr, falsch	2	10
wahr, falsch, wahr	3	11
wahr, falsch, falsch	4	12
falsch, wahr, wahr	5	13
falsch, wahr, falsch	6	14
falsch, falsch, wahr	7	15
falsch, falsch, falsch	8	16

Abbildung 8.1: Reihenfolge der Parameter in den Arrays `CombinationSingle` (CPT oben links), `CombinationDuad` (CPT unten links) und `CombinationTriple` (CPT rechts) des Zusatzwissens in JSON.

folgen für die drei Größen der CPTs veranschaulicht.
Im Array `Existences` (ab Zeile 63) werden auf dieselbe Weise Parameter für die CPTs der Existenz-Variablen, sowohl für die optionale als auch für die obligatorische Existenz, definiert.

8.3.3 Generierungsprozess

Abb. 8.2 veranschaulicht die Generierung des Bayes'schen Netzes. Es ist vorteilhaft, die gesamte Generierung innerhalb eines Parsers vorzunehmen. Im Gegensatz zu der obigen Beschreibung sollte kein unvollständiges Bayes'sches Netz als Zwischenschritt entstehen, sondern die Informationen aller drei Wissensquellen sofort verwertet werden. In diesem Fall ist auch die Referenzierung auf Elemente und Parameter vom Zusatzwissen zur Soll-Designspezifikation einfacher realisierbar. Die obige Separierung wurde lediglich zur besseren Verständlichkeit des Prozesses vorgenommen. Das Beispiel im folgenden Abschnitt wird aus diesem Grund auf die gleiche Weise präsentiert.

8.4 Spezifikation eines Wizardmenüs

Für das in Kap. 7.3 in Abb. 7.5 dargestellte Bayes'sche Netz werden nun die Soll-Designspezifikation und das Zusatzwissen in JSON angegeben.

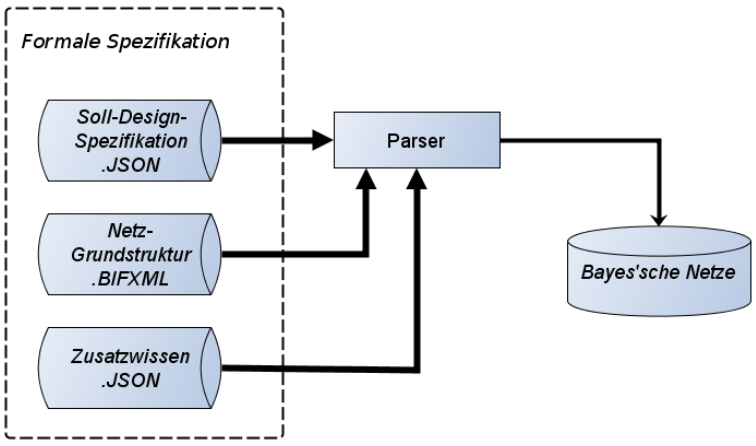


Abbildung 8.2: Generierung des Bayes'schen Netzes aus einer formalen Spezifikation, die aus drei Wissensquellen besteht.

8.4.1 Soll-Designspezifikation

Der Quellcode 8.4 stellt die Spezifikation der lokalen und globalen Merkmale für das Icon *IconPlay* und dem dazugehörigen Tooltip des Wizardmenüs dar. Zugrunde liegt das in Anhang C.1 aufgeführte Schema, das sich aus den in Abschnitt 8.2 vorgestellten Teilen zusammensetzt.

Quellcode 8.4: Definition der Soll-Designspezifikation in JSON für das Wizardmenü-Beispiel aus Kap. 7.3

```
{
  "GUIName": "MMI Test",
  "Dimension":
    {
      "Width": 800,
      "Height": 480,
    },
  ... // Area- und Master-Definitionen
  "Views": [
    {
      "Name": "CDA Wizard Test",
      "MasterReference": "Wizard1",
      "AreaReference": "Main",
      "Symbols": [
        {
          "Name": "IconPlay",
          "Optional": false,
          "UpperLeftPosition":
            {
              "X": 369,
              "Y": 204
            },
          "CenterPosition":
            {
              "X": 300,
              "Y": 322.5
            },
          "Paths": [
```

```
            {
              "Path": "./symbols/wizards/cda/play_pressed.png",
              "State": "pressed"
            },
            {
              "Path": "./symbols/wizards/cda/play_disabled.png",
              "State": "disabled"
            },
            {
              "Path": "./symbols/wizards/cda/play_rollover.png",
              "State": "rollover"
            },
            {
              "Path": "./symbols/wizards/cda/play_rolloverpressed.png",
              "State": "rollover pressed"
            },
            {
              "Path": "./symbols/wizards/cda/play_enabled.png",
              "State": "enabled"
            }
          ]
        },
        {
          "Name": "TooltipBorderPlay",
          "Type": "WizardTooltipBorder",
          "Color":
            {
              "Red": 255,
              "Green": 170,
              "Blue": 6
            }
        },
        {
          "Name": "TooltipArrowPlay",
          "Path": "./symbols/wizards/arrow.png"
        }
      ],
      "Texts": [
        {
          "Name": "TooltipTextPlay",
          "Font": {
            "Name": "LinotypeUnivers",
            "Style": "440Extended",
            "Size": 26,
            "Bold": false,
            "Italic": false
          },
          "Color": {
            "Red": 255,
            "Green": 255,
            "Blue": 255
          }
        }
      ],
      "Distances":[
        {
          "ElementReference1": "TooltipBorderPlay",
          "ElementReference2": "TooltipTextPlay",
          "Axis": "X",
          "Relation": "enclosing",
          "ReferencePoint": "border_left",
          "Distance": 15
        },
        {
          "ElementReference1": "TooltipBorderPlay",
          "ElementReference2": "TooltipTextPlay",
          "Axis": "X",
          "Relation": "enclosing",
          "ReferencePoint": "border_right",
          "Distance": 15
```

```
        },
        {
          "ElementReference1": "TooltipBorderPlay",
          "ElementReference2": "TooltipTextPlay",
          "Axis": "Y",
          "Relation": "enclosing",
          "ReferencePoint": "border_top",
          "Distance": 9
        },
        {
          "ElementReference1": "TooltipBorderPlay",
          "ElementReference2": "TooltipTextPlay",
          "Axis": "Y",
          "Relation": "enclosing",
          "ReferencePoint": "border_bottom",
          "Distance": 10
        },
        {
          "ElementReference1": "TooltipArrowPlay",
          "ElementReference2": "IconPlay",
          "Axis": "Y",
          "Relation": "sideBySide",
          "ReferencePoint": "border",
          "Distance": 10
        }
      ],
      "Coexistences":[
        {
          "ElementReferences":[
            "TooltipTextPlay",
            "TooltipArrowPlay",
            "TooltipBorderPlay"
          ]
        }
      ],
      "Mutualexclusions":[
        {
          "ElementReferences":[
            {
              "ElementReference": "IconPlay",
              "States":[
                "pressed",
                "enabled",
                "disabled"
              ]
            },
            {
              "ElementReference": "TooltipTextPlay"
            },
            {
              "ElementReference": "TooltipArrowPlay",
            },
            {
              "ElementReference": "TooltipBorderPlay"
            }
          ]
        }
      ]
  } ]
}
```

Die hier dargestellte Definition der Position mit `UpperLeftPosition` und `CenterPosition` ist eine Möglichkeit die Positionierung und Dimension eines Elements vollständig zu beschreiben. Dieselbe Aussagekraft kann auch mit `UpperLeftPosition` und `Dimension` erreicht werden.

Für das Symbolelement *TooltipBorderPlay* wird anstelle einer Datei eine Farbe angegeben. Der Rahmen eines Tooltip kann nicht als Bilddatei hinterlegt werden, da sich die Größe dem Textinhalt anpasst. Da der Rahmen einfarbig ist, kann die Farbe verifiziert werden. Der Typ wird als weiteres Attribut angegeben, um Bilderkenner-Ergebnisse zuzuweisen.

8.4.2 Zusatzwissen

Das Zusatzwissen für die Generierung des Bayes'schen Netzes in Abb. 7.5 ist in Quellcode 8.5 aufgeführt. Die JSON-Struktur basiert auf dem in Abschnitt 8.3.2 vorgestellten Schema.

Quellcode 8.5: Definition des Zusatzwissens in JSON für das Wizardmenü-Beispiel aus Kap. 7.3

```
{
  "Mappings": [
    {
      "old": 1.0,
      "new": 0.99
    },
    {
      "old": 0.0,
      "new": 0.01
    }
  ],
  "Variances": [
    {
      "Elementreferences": ["TooltipTextPlay"],
      "Features": ["FarbkanalRot", "FarbkanalGruen", "FarbkanalBlau"],
      "Tolerances": [3, 3, 5],
      "Probabilities": [0.99, 0.95, 0.9]
    },
    {
      "Elementreferences": ["IconPlay"],
      "Features": ["PositionX", "PositionY"],
      "Tolerances": [0, 0, 0],
      "Probabilities": [0.99, 0.98, 0.9]
    }
  ],
  "CombinationDuad": [0.99 0.01 0.5 0.5 0.5 0.5 0.01 0.99],
  "CombinationTriple": [0.99 0.01 0.66 0.34 0.66 0.34 0.33 0.67 0.66 0.34 0.33 0.67 0.33 0.67 0.01 0.99],
  "Coexistences": [
    {
      "Elementreferences": ["IconPlay", "TooltipTextPlay"],
      "Features": [Koexistenz],
      "Probabilities": [0.005, 0.99, 0.005, 0.005, 0.99, 0.005, 0.005, 0.99, 0.005, 0.99, 0.009, 0.001, 0.99,
          0.009, 0.001, 0.005, 0.005, 0.99, 0.99, 0.009, 0.001, 0.99, 0.009, 0.001, 0.99, 0.009, 0.001, 0.99,
          0.009, 0.001, 0.99, 0.009, 0.001, 0.99, 0.009, 0.001]
    }
  ]
}
```

Mit den definierten Parameterumwandlungen im Array `Mappings` bilden die CPTs der Variablen `Korrekt_TooltipTextPlay_Sicherheit`, `Korrekt_IconPlay_Referenz`, `Korrekt_IconPlay_Existenz` und `IconPlay_Status` die gewünsch-

ten Wahrscheinlichkeitsverteilungen ab.
Die Objekte im Array `Variances` definieren die Gewichtung der Merkmale *FarbkanalRot*, *FarbkanalGruen* und *FarbkanalBlau* für das Element *TooltipTextPlay* sowie die Gewichtung für die Merkmale *PositionX* und *PositionY* für das Element *IconPlay*. Basierend auf den in der Spezifikation definierten Sollwerten entstehen die Intervalle bzw. Einzelwerte *253bis255*, *250bis252* und *246bis250* für die Farbkanäle sowie die Positionen *x368*, *x369*, *x370*, *x371*, *x372*, *y302*, *y303*, *y304*, *y305* und *y306*.
Die CPTs der EVs und VVs werden mit den in den Arrays `CombinationDuad` und `CombinationTriple` definierten Parametern vervollständigt. Das Array `CombinationSingle` ist nicht notwendig, da durch die Parameterumwandlung bereits alle CPTs, die nur von einem Elternknoten abhängig sind, passend definiert werden konnten. In den Zeilen 29 bis 35 werden die Parameter für die Variable `Korrekt_Koexistenz_IconPlay_TooltipTextPlay` definiert.

8.5 Begründung und Einordnung

Im Folgenden werden die entwickelten Konzepte bewertet. Insbesondere wird auf die Vor- und Nachteile von JSON als Spezifikationsformat eingegangen.

8.5.1 JSON als Spezifikationsformat

Beim Einsatz einer formalen Spezifikationssprache muss stets zwischen zwei Möglichkeiten abgewogen werden: Entweder wird ein existierendes Format verwendet, das meist auf den eigenen Anwendungsfall angepasst werden muss, oder es wird ein neues Konzept entwickelt, das exakt auf den Anwendungsfall zugeschnitten ist. Beide Vorgehensweisen sind mit Aufwand verbunden. Welche Möglichkeit adäquater ist, ist abhängig vom Anwendungsfall und der Eignung der existierenden Formate. Da für die Beschreibung von GUI-Design die existierenden Spezifikationssprachen nicht allen Anforderungen genügen, wurde ein neues Format konzipiert. In den folgenden Abschnitten wird diese Entscheidung detailliert begründet.

8.5.1.1 Existierende Spezifikationssprachen

Alle in Kap. 3.4 vorgestellten Spezifikationssprachen ermöglichen durchaus eine detaillierte Beschreibung einer GUI und weiteren Komponenten einer Software. Die Struktur der Sprachen ist zudem sehr modular, was eine Gliederung in einzelne Bereiche unterstützt und den Zugriff auf bestimmte Inhalte erleichtert. Der Fokus liegt allerdings stets auf der Spezifikation von Verhaltenslogik; bzgl. der Beschreibung von GUI-Design sind die Sprachen (gewollt) unzureichend. Obgleich auch im automobilen Umfeld Spezifikationssprachen entwickelt wurden, deckt keine das vollständige automobile HMI ab.
Die existierenden GUI-Spezifikationssprachen haben alle das Ziel, Schnittstellen formal zu beschreiben, um sie dann, insbesondere für verschiedene Applikationen, automatisch in ausführbaren Code umzuwandeln. Das Ziel bei der Designverifikation ist hingegen, lediglich Informationen des GUI-Designs formal abzubilden und somit eine geeignete Wissensbasis für ein Designverifikationssystem zu erhalten. Ein anderer Anwendungsbereich und die erforderlichen umfangreichen Kenntnisse über die Spezifikationssprache stellen die Erweiterung einer existierenden Spezifikationssprache um die fehlenden Designinhalte nicht als praktikable Lösung dar. Sprachen wie OEM XML können zudem aufgrund ihrer sehr komplizierten, nichtmenschenlesbaren Struktur zu keiner effizienten Lösung für die Beschreibung von GUI-Design führen. Die Komplexität eines Infotainmentsystems ist zudem enorm; das Tag-System von XML vergrößert diese Datenmenge noch einmal um ein Vielfaches. Aus diesem Grund ist fraglich, ob die Verwendung eines XML-basierten Formats für die Spezifikation von GUI-Design überhaupt sinnvoll ist. Die zugrunde liegende Basissprache sollte dennoch möglichst bekannt und von verschiedenen Programmiersprachen bereits unterstützt werden, was den Aufwand der Systemerstellung deutlich verringert. Wie die vorhergehenden Abschnitte gezeigt haben, ist trotz der großen Anzahl verschiedener Informationen die Spezifikation in JSON sehr kompakt und einfach. D.h. die Erweiterung und Reduzierung eines existierenden Formats stellt in der Regel ein um einiges höheres Risiko und Aufwand dar. Nach [Fensel 1994] zeichnet sich eine gute Spezifikationssprache zudem durch ihre Anwendungsspezifität aus.

8.5.1.2 Eigenschaften der JSON-Spezifikation

Bezugnehmend auf die in Kap. 8.1.1 formulierten Anforderungen werden im Folgenden die Eigenschaften der entwickelten JSON-Spezifikation bewertet.

Lesbarkeit und Verwendbarkeit Wie die Beispiele in Abschnitt 8.4 zeigen, können die hierarchische Struktur einer GUI, ihr Design und die daraus resultierenden Semantiken und Zusammenhänge in der Spezifikation beschrieben werden. Komplexe Typen, wie bspw. `CenterPosition` und `Views`, werden dabei geeignet in Form eines Objekts oder Arrays definiert. Die Spezifikation ist formal und damit maschinenlesbar. Sie unterstützt zudem eine hierarchische Gliederung der Information, die für den Menschen intuitiv verständlich ist. Die kompakte Darstellung und das Vorhandensein von Editoren, die die hierarchische Struktur als Baum visualisieren, gewährleisten eine noch bessere Lesbarkeit für den Menschen. Bei der Entwicklung der Struktur wurde auch auf Einheitlichkeit geachtet. Weitere Vorgaben sollen dies unterstützen. Bspw. erfolgt die Definition von Designelementen immer in dem Teil, in dem die lokalen Merkmale definiert werden. Erst dann kann es für die Definition von globalen Merkmalen verwendet werden. Die Definition geschieht unabhängig davon, ob für ein Element tatsächlich lokale Merkmale bekannt sind.

Effizienz JSON beschränkt sich mit Objekten und Arrays auf die Beschreibung der Informationen und weist keine zusätzlichen Konstrukte wie bspw. Tags auf, die zu einer unnötigen Vergrößerung der Datenmenge führen. Redundanzen werden durch die Verwendung von Master-Views und die Referenzierung auf Master-Views, Screenbereiche und Elemente vermieden. Das entwickelte JSON-Schema gibt zudem vor, dass keine leeren Arrays angelegt werden dürfen und dadurch unnötig Speicherplatz verbraucht wird. Auf diese Weise wird eine kompakte, platzsparende Wissensbeschreibung und ein effizienter Zugriff auf das Soll-Design ermöglicht. Auch der Transfer in ein Bayes'sches Netz kann effizient durchgeführt werden, da die Spezifikation auf sehr direkte Weise die Informationen definiert und so eine einfache Konvertierung in das BIFXML-Format gewährleistet.

Vollständigkeit Im Rahmen der Beispiele in Abschnitt 8.4 kann das entwickelte Spezifikationsformat alle erforderlichen Designinhalte abbilden. Eine umfangreiche Prüfung auf Tauglichkeit kann allerdings nur im praktischem Einsatz erfolgen

und lässt sich deshalb nicht für zukünftige Designs voraussagen. Flexibilität und Erweiterbarkeit kann die Fähigkeit andersartiges GUI-Design abzubilden unterstützen.

Flexibilität und Erweiterbarkeit Die vorgestellte JSON-Struktur bietet flexible Möglichkeiten für die Definition von Designinhalten. Bspw. kann auf verschiedene Weise die Position eines Elements beschrieben werden. Bis auf die vorgegebene Hierarchie und die Forderung nach eindeutigen Namen, sind viele Elemente optional. Damit ist es z.B. nicht zwingend, einen Master zu definieren oder für ein Symbolelement einen Pfad anzugeben. Diese Flexibilität ist wichtig, da Designvorgaben sehr unterschiedlich sein können. Für das MMI-Design sind z.B. Positionen oftmals nicht einheitlich angegeben und evtl. nur über Abstände zu anderen Elementen definiert.[2]
Die formale Spezifikation des Soll-Designs lässt sich aufgrund der hierarchischen Struktur auf einfache Weise erweitern. Anstelle eines Objekts kann stets auch ein Array verwendet werden, um mehrere Möglichkeiten definieren zu können (s. Abschnitt 8.2). Im Ausblick, Kap. 11.3, wird kurz darauf eingegangen, wie bereits generierte Bayes'sche Netze oder vorhandene formale Spezifikationen effizient um neue Informationen erweitert werden können.

Korrektheit Die Korrektheitsprüfung steht im engen Zusammenhang mit den Konventionen, an die sich eine Spezifikation halten soll. Je mehr Konventionen bereits durch das Schema vorgegeben sind, umso weniger muss extern validiert werden. Bspw. fordert das Schema für die Definition des Merkmals *Abstand* die Referenz auf genau zwei Elemente und für die Definition einer Koexistenz und eines gegenseitigen Ausschlusses die Referenz auf mindestens zwei Elemente. Damit wird bereits an dieser Stelle die Definition einer unsinnigen Anzahl von Elementen verhindert[3]. Auch die Verwendung von Enumerationen, Minimum- und Maximumangaben verhindern Fehler in der Spezifikation.

[2] Alternativ könnte auch gefordert werden, dass die Designvorgaben einheitlich sind. Bestehende, meist sehr komplexe, Entwicklungsprozesse zu ändern, ist aber mit einem hohem Aufwand verbunden oder gar nicht möglich. Zudem ist gerade die Entstehung von GUI-Design eine sehr kreative Arbeit. Das Spezifikationsformat sollte daher Freiheiten lassen.

[3] Ob es sich bei den referenzierten Elementen tatsächlich um existierende Elemente oder gar die richtigen Elemente handelt, ist wieder eine ganz andere Frage.

Die Ausdrucksmächtigkeit eines JSON-Schemas hat allerdings Grenzen und es können nicht alle Konventionen definiert werden. Z.B. kann die Eindeutigkeit der Namen von Areas, Master-Views, Views und Elementen nicht arrayübergreifend definiert werden. Und beim Zusatzwissen muss sichergestellt werden, dass bei der Generierung der Soll-Abweichungen keine Werte entstehen, die außerhalb des Wertebereichs des jeweiligen Merkmals liegen. Falls die Spezifikation maschinell erstellt wird, muss der Parser die Einhaltung dieser Konventionen gewährleisten. Darüber hinaus steht der einfachen Validierung mittels eines Schemas die Forderung nach Flexibilität entgegen. Je universeller die JSON-Struktur für verschiedene GUI-Designs eingesetzt werden soll, umso weniger Einschränkungen dürfen im Schema definiert werden. Dann sind aber mitunter umso mehr externe Prüfungen erforderlich. Für die Formulierung eines Spezifikationsformats bedeutet das ein stetiges Abwägen zwischen Flexibilität und Einschränkung. Es ist ratsam, pro zu spezifizierender GUI eine separate Schemadatei zu erstellen, die umfangreiche Konventionen beinhalten kann.

8.5.1.3 JSON im Vergleich zu XML

Um die Vor- und Nachteile von JSON gegenüber XML zu verdeutlichen (s. Kap. 3.3.4), wird die JSON-Datei mit zwei XML-Dateien gleichen Inhalts[4] verglichen. Die erste XML-Datei wurde mittels des Konvertierungs-Tools [Briganti 2012] direkt aus der in Abschnitt 8.4.1 vorgestellten JSON-Datei erzeugt und ist in Quellcode 8.6 aufgelistet. Die zweite XML-Datei ist in Quellcode 8.7 zu sehen und wurde manuell auf Basis einer ebenfalls manuell erstellten XML-Schemadatei entwickelt. Die Schemadatei ist in Anhang C.2 aufgeführt.

Quellcode 8.6: XML-Datei 1 direkt aus JSON-Datei generiert

```
<?xml version="1.0" encoding="UTF-8" ?>
  <GUIName>MMI Test</GUIName>
  <Dimension>
    <Width>800</Width>
    <Height>480</Height>
  </Dimension>
  <Views>
    <Name>CDA Wizard Test</Name>
    <MasterReference>Wizard1</MasterReference>
    <AreaReference>Main</AreaReference>
    <Symbols>
      <Name>IconPlay</Name>
      <Optional>false</Optional>
```

[4] Derselbe Inhalt bedeutet, dass die Dateien dasselbe GUI-Design repräsentieren und die Schemata dieselbe GUI-Struktur und denselben Umfang an Konventionen.

```
      <UpperLeftPosition>
        <X>369</X>
        <Y>204</Y>
      </UpperLeftPosition>
      <CenterPosition>
        <X>300</X>
        <Y>322.5</Y>
      </CenterPosition>
      <Paths>
        <Path>./symbols/wizards/cda/play_pressed.png</Path>
        <State>pressed</State>
      </Paths>
      <Paths>
        <Path>./symbols/wizards/cda/play_disabled.png</Path>
        <State>disabled</State>
      </Paths>
      <Paths>
        <Path>./symbols/wizards/cda/play_rollover.png</Path>
        <State>rollover</State>
      </Paths>
      <Paths>
        <Path>./symbols/wizards/cda/play_rolloverpressed.png</Path>
        <State>rollover pressed</State>
      </Paths>
      <Paths>
        <Path>./symbols/wizards/cda/play_enabled.png</Path>
        <State>enabled</State>
      </Paths>
    </Symbols>
    <Symbols>
      <Name>TooltipBorderPlay</Name>
      <Type>WizardTooltipBorder</Type>
      <Color>
        <Red>255</Red>
        <Green>170</Green>
        <Blue>6</Blue>
      </Color>
    </Symbols>
    <Symbols>
      <Name>TooltipArrowPlay</Name>
      <Path>./symbols/wizards/arrow.png</Path>
    </Symbols>
    <Texts>
      <Name>TooltipTextPlay</Name>
      <Font>
        <Name>LinotypeUnivers</Name>
        <Style>440Extended</Style>
        <Size>26</Size>
        <Bold>false</Bold>
        <Italic>false</Italic>
      </Font>
      <Color>
        <Red>255</Red>
        <Green>255</Green>
        <Blue>255</Blue>
      </Color>
    </Texts>
    <Distances>
      <ElementReference1>TooltipBorderPlay</ElementReference1>
      <ElementReference2>TooltipTextPlay</ElementReference2>
      <Axis>X</Axis>
      <Relation>enclosing</Relation>
      <ReferencePoint>border_left</ReferencePoint>
      <Distance>15</Distance>
    </Distances>
    <Distances>
      <ElementReference1>TooltipBorderPlay</ElementReference1>
      <ElementReference2>TooltipTextPlay</ElementReference2>
      <Axis>X</Axis>
      <Relation>enclosing</Relation>
```

```
      <ReferencePoint>border_right</ReferencePoint>
      <Distance>15</Distance>
    </Distances>
    <Distances>
      <ElementReference1>TooltipBorderPlay</ElementReference1>
      <ElementReference2>TooltipTextPlay</ElementReference2>
      <Axis>Y</Axis>
      <Relation>enclosing</Relation>
      <ReferencePoint>border_top</ReferencePoint>
      <Distance>9</Distance>
    </Distances>
    <Distances>
      <ElementReference1>TooltipBorderPlay</ElementReference1>
      <ElementReference2>TooltipTextPlay</ElementReference2>
      <Axis>Y</Axis>
      <Relation>enclosing</Relation>
      <ReferencePoint>border_bottom</ReferencePoint>
      <Distance>10</Distance>
    </Distances>
    <Distances>
      <ElementReference1>TooltipArrowPlay</ElementReference1>
      <ElementReference2>IconPlay</ElementReference2>
      <Axis>Y</Axis>
      <Relation>sideBySide</Relation>
      <ReferencePoint>border</ReferencePoint>
      <Distance>10</Distance>
    </Distances>
    <Coexistences>
      <ElementReferences>TooltipTextPlay</ElementReferences>
      <ElementReferences>TooltipArrowPlay</ElementReferences>
      <ElementReferences>TooltipBorderPlay</ElementReferences>
    </Coexistences>
    <Mutualexclusions>
      <ElementReferences>
        <ElementReference>IconPlay</ElementReference>
        <States>pressed</States>
        <States>enabled</States>
        <States>disabled</States>
      </ElementReferences>
      <ElementReferences>
        <ElementReference>TooltipTextPlay</ElementReference>
      </ElementReferences>
      <ElementReferences>
        <ElementReference>TooltipArrowPlay</ElementReference>
      </ElementReferences>
      <ElementReferences>
        <ElementReference>TooltipBorderPlay</ElementReference>
      </ElementReferences>
    </Mutualexclusions>
  </Views>
```

Quellcode 8.7: XML-Datei 2 manuell erstellt auf Basis einer XML-Schemadatei

```
<?xml version="1.0" encoding="utf-8"?>
<GUI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="
    ScreenSYMBOLXMLSchema.xsd" Name="MMI Test">
  <DIMENSION>
    <Width>800</Width>
    <Height>480</Height>
  </DIMENSION>
  ... // Area und Master Definitionen
  <VIEW Name="CDA Wizard Test">
    <MASTERREF Name="Wizard"/>
    <AREAREF Name="Main">
      <TEXT Name="TooltipTextPlay">
        <FONT Name="LinotypeUnivers" Style="440Extended" Size="26" Bold="false" Italic="false" />
        <COLOR R="255" G="255" B="255"/>
      </TEXT>
      <SYMBOL Name="TooltipBorderPlay" Type="WizardTooltipBorder">
```

```
        <COLOR R="255" G="170" B="6"/>
      </SYMBOL>
      <SYMBOL Name="TooltipArrowPlay">
        <PATH Path="D:\symbols\wizards\arrow.png"/>
      </SYMBOL>
       <SYMBOL Name="Play" Optional="false">
        <POSITION>
          <XPosition>369</XPosition>
          <YPosition>204</YPosition>
          <XCenterPosition>300</XCenterPosition>
          <YCenterPosition>322.5</YCenterPosition>
        </POSITION>
        <PATH Path=".\symbols\wizards\cda\play_pressed.png" State="pressed" />
        <PATH Path=".\symbols\wizards\cda\play_disabled.png" State="disabled" />
        <PATH Path=".\symbols\wizards\cda\play_rollover.png" State="rollover" />
        <PATH Path=".\symbols\wizards\cda\play_rolloverpressed.png" State="rollover pressed" />
        <PATH Path=".\symbols\wizards\cda\play_enabled.png" State="enabled" />
      </SYMBOL>
      <DISTANCE Axis="x" Distance="15" ReferencePoint="border_left" Relation="enclosing">
        <SYMBOLREF Name="TooltipBorderPlay"/>
        <TEXTREF Name="TooltipTextPlay"/>
      </DISTANCE>
      <DISTANCE Axis="x" Distance="15" ReferencePoint="border_right" Relation="enclosing">
        <SYMBOLREF Name="TooltipBorderPlay"/>
        <TEXTREF Name="TooltipTextPlay"/>
      </DISTANCE>
      <DISTANCE Axis="y" Distance="9" ReferencePoint="border_top" Relation="enclosing">
        <SYMBOLREF Name="TooltipBorderPlay"/>
        <TEXTREF Name="TooltipTextPlay"/>
      </DISTANCE>
      <DISTANCE Axis="y" Distance="10" ReferencePoint="border_bottom" Relation="enclosing">
        <SYMBOLREF Name="TooltipBorderPlay"/>
        <TEXTREF Name="TooltipTextPlay"/>
      </DISTANCE>
      <DISTANCE Axis="y" Distance="10" ReferencePoint="border" Relation="sideBySide">
        <SYMBOLREF Name="TooltipArrowPlay"/>
        <SYMBOLREF Name="IconPlay"/>
      </DISTANCE>
      <COEXISTENCE>
        <TEXTREF Name="TooltipTextPlay"/>
        <SYMBOLREF Name="TooltipArrowPlay"/>
        <SYMBOLREF Name="TooltipBorderPlay"/>
      </COEXISTENCE>
      <MUTALEXCLUSION>
        <PROPERTY PropertyName="State" Value="rollover"/>
        <SYMBOLREF Name="SYMBOL_Play" />
        <TEXTREF Name="TooltipTextPlay"/>
        <SYMBOLREF Name="TooltipArrowPlay"/>
        <SYMBOLREF Name="TooltipBorderPlay"/>
      </MUTALEXCLUSION>
      <MUTALEXCLUSION>
        <PROPERTY PropertyName="State" Value="rollover pressed"/>
        <SYMBOLREF Name="IconPlay" />
        <TEXTREF Name="TooltipTextPlay"/>
        <SYMBOLREF Name="TooltipArrowPlay"/>
        <SYMBOLREF Name="TooltipBorderPlay"/>
      </MUTALEXCLUSION>
      <MUTALEXCLUSION>
        <PROPERTY PropertyName="State" Value="pressed"/>
        <SYMBOLREF Name="IconPlay" />
        <TEXTREF Name="TooltipTextPlay"/>
        <SYMBOLREF Name="TooltipArrowPlay"/>
        <SYMBOLREF Name="TooltipBorderPlay"/>
      </MUTALEXCLUSION>
    </AREAREF>
  </VIEW>
</GUI>
```

Es sei angemerkt, dass es noch zahlreiche weitere Möglichkeiten gibt, den gewünschten Inhalt in JSON und XML zu beschreiben. Der Vergleich bezieht sich damit nur auf das vorgestellte Beispiel. Ebenfalls kann es je nach spezifizierten Informationen zu unterschiedlichen Ergebnissen kommen. Die Eignung eines Spezifikationsformats ist, wie bereits oben erwähnt, anwendungsspezifisch.

Vorteile der JSON-Spezifikation Die direkte Konvertierung von JSON in XML ohne Verwendung eines Schemas mit dem Konvertierungstool [Briganti 2012] führt zu einer XML-Datei mit 3215 Zeichen, die 3,13KB Speicher einnimmt. Die zweite XML-Datei beinhaltet 2746 Zeichen und nimmt 2,68KB Speicher ein, zuzüglich 9682 Zeichen und 9,45KB für die Schemadatei. Im Gegensatz dazu umfasst die JSON-Datei, die denselben Inhalt repräsentiert, nur 2118 Zeichen und benötigt 2,06KB Speicher. Die JSON-Schemadatei (s. Anhang C.1) beinhaltet 3296 Zeichen und benötigt 3,21KB Speicher.
Bereits anhand dieses kleinen Beispiels ist zu erkennnen, dass Informationen mit JSON tatsächlich platzsparender repräsentiert werden als mit XML. Die bessere Eignung von JSON für die Beschreibung von GUI-Design gegenüber XML lässt sich durch den ursprünglichen Anwendungsbereich der beiden Formate erklären. Während es sich bei JSON lediglich um ein Datenaustauschformat handelt, handelt es sich bei XML um eine erweiterbare Auszeichnungssprache. Diese Eigenschaft ist für die Beschreibung von GUI-Design nicht erforderlich und der Einsatz von XML daher überdimensioniert.

Nachteile der JSON-Spezifikation In der entwickelten JSON-Spezifikation wurde kein Gebrauch des Schlüsselworts `uniqueItems` gemacht, um die Eindeutigkeit von Elementen innerhalb eines Arrays zu fordern. Genau genommen wird eine Eindeutigkeitskonvention benötigt, die sich auf den Namen aller spezifizierten Designelemente bezieht. Im Gegensatz zu der XML Schema Definition bietet das JSON-Schema nicht diese Möglichkeit. D.h. hier ist eine externe Validierung erforderlich.

8.5.2 Zusatzwissen

Da die Soll-Designspezifikation in JSON formuliert ist und JSON eine einfache, kompakte Notation bereitstellt, wurde auch für die Beschreibung des Zusatzwissens

eine JSON-Struktur konzipiert.Analog zur Soll-Designspezifikation, kann auch die Beschreibung des Zusatzwissens erweitert werden. U.a. sind die folgenden Erweiterungen angebracht:

- Aktuell wird im Zusatzwissen auf die Element- uns Merkmalsnamen in der Soll-Designspezifikation referenziert. Alternativ wäre auch die Angabe der Variablennamen möglich, so wie sie in dem bereits aus der Spezifikation generierten Bayes'schen Netz vorkommen. Welche Vorgehensweise geeigneter ist, ist Frage der Implementierung, die bestimmt, wie das Bayes'sche Netz aus den verschiedenen Wissensquellen aufgebaut wird.

- Im Zusatzwissen können auch Apriori-Verteilungen definiert werden, um die bisher verwendete Gleichverteilung zu ersetzen.

- In allen bisher betrachteten Beispielen sind Soll-Abweichungen nur für numerische Werte erforderlich. Für verbale Werte müsste das Array `Tolerances` anders definiert oder die verbalen Werte in numerische abgebildet werden, was im Rahmen dieser Arbeit aber keine weitere Rolle spielt.

- Bislang wurde nur eine Glockenkurve für die Definition der Soll-Abweichungen verwendet werden. Andere Gewichtungen sind auch realisierbar.

8.5.3 Generierungsprozess

Die Soll-Designspezifikation beschreibt das Soll-Design einer GUI und beinhaltet damit nicht alle Informationen, die für die Generierung eines funktionsfähigen Bayes'schen Netzes erforderlich sind. Zusätzlich müssen auch eine Grundstruktur des Bayes'schen Netzes vorliegen, die die Variablen und ihre Abhängigkeiten definiert sowie Zusatzwissen, das die Soll-Abweichungen und weitere Parameter liefert. Die Kombination dieser Wissensquellen kann als *strukturierte Informationen* bezeichnet werden. In [Wiegerinck, Kappen und Burgers 2010], Kap. 3.1. wird eine ähnlich strukturierte Vorgehensweise bzw. generische Struktur vorgestellt, aus der ein Bayes'sches Netz generiert wird. Der Grund dort sind die zum automatischen Lernen nicht in ausreichendem Maße vorhandenen Daten.
Der Parser führt die Kombination der in der Soll-Designspezifikation, der Grundstruktur und dem Zusatzwissen enthaltenen Informationen durch und fordert aus diesem Grund bestimmte Konventionen für die Wissensquellen:

- Eindeutigkeit der Designelementnamen in der Spezifikation
- Abbildung zwischen den Variablen der Grundstruktur und den in der Spezifikation definierten Elementen und Merkmalen
- Abbildung zwischen den in der Spezifikation und dem Zusatzwissen definierten Elementmerkmalen

Unter Einhaltung der Konventionen sind die Wissensquellen unabhängig voneinander austauschbar. Z.B. kann unterschiedliches Zusatzwissen verwendet werden, um aus demselben Soll-Design und derselben Grundstruktur Bayes'sche Netze zu generieren, die die Merkmale unterschiedlich gewichten.

8.5.4 Vergleich mit manuell erzeugtem Bayes'schen Netz

Es ist die Generierung eines Netzes möglich, das identisch zu dem in Kap. 7 manuell erstellten Netz ist. Diese Eigenschaft begründet sich vor allem darin, dass das Bayes'sche Netz auf Basis der manuell überlegten Netzstruktur generiert wird. Daneben bilden die entwickelte JSON-Spezifikation, die Grundstruktur im BIFXML-Format und das Zusatzwissen alle geforderten Informationen vollständig ab und führen zu keinerlei Einschränkungen oder Änderungen.

8.6 Schlussfolgerung

In diesem Kapitel wurde eine Spezifikation in dem Datenaustauschformat JSON konzipiert und konkret am Beispiel eines MMI-Wizardmenüs gezeigt, wie daraus ein Bayes'sches Netz generiert werden kann. Die in Abschnitt 8.1.1 formulierten Anforderungen werden durch die entwickelte JSON-Struktur in hohem Maße erfüllt. Mit der JSON-Struktur wird genau ein Format bereitgestellt, um das GUI-Design vom MMI vollständig zu beschreiben und dennoch Spielraum für andersartige GUIs zu lassen. Diese wesentlichen Vorteile resultieren u.a. daraus, dass die JSON-Spezifikation genau für die Beschreibung von GUI-Design konzipiert wurde und damit exakt auf diesen Anwendungsbereich zugeschnitten ist. Das Format ist zudem intuitiv und platzsparend, was sowohl die Entwicklung, als auch die Verwendung einfach und effizient ermöglicht. Diese Vorteile konnten im direkten Vergleich mit XML (Abschnitt 8.5.1) noch verdeutlicht werden. Der einzige Nach-

teil von JSON gegenüber XML ist die geringere Mächtigkeit des Schemas, sodass manche Konventionen extern validiert werden müssen. In Kap. 10 wird sich allerdings zeigen, dass eine XML-Spezifikation für den produktiven Einsatz in der MMI-Testautomatisierung noch aus einem weiteren Grund hinderlich wäre, wohingegen JSON sich direkt in die bestehenden Gegebenheiten integrieren lässt. Damit ist die Spezifikation von GUI-Design auf einfache und kompakte Weise gelungen.
In diesem Kapitel wurde nicht darauf eingegangen, woher die Wissensquellen stammen. Die Soll-Designspezifikation hat dabei den größten Umfang und für komplexe GUIs ist eine manuelle Erstellung nicht praktikabel. Das Kap. 10 zeigt, wie das GUI-Design vom MMI zumindest teilweise aus dem Designentwurf automatisch gewonnen werden kann. In Kap. 9 wird erläutert, wie sowohl das Soll-Design als auch die Grundstruktur durch Lernen gewonnen werden können. Übrig bleibt das Zusatzwissen, das nur bedingt durch das Lernen entsteht. Das Zusatzwissen kann als eine Art Konfiguration betrachtet werden und Generierungsmöglichkeiten werden in Kap. 10 angesprochen.

Kapitel 9

Generierung des Bayes'schen Netzes durch Lernen

In Kap. 5.3.2 wurden einige Verfahren zum Lernen Bayes'scher Netze vorgestellt. In diesem Kapitel wird gezeigt, wie Bayes'sche Netze für die Designverifikation gelernt werden können. Die Lerndaten werden dazu mittels Bildverarbeitung aus Screenshots gewonnen, die das korrekte Design darstellen. Das Ziel ist hier nicht, Lernverfahren zu vergleichen, sondern zu zeigen, inwieweit das Lernen von GUI-Designinformationen möglich ist.

9.1 Motivation und Ziel

GUI-Design ist meist sehr umfangreich, sodass eine manuelle Erstellung der Bayes'schen Netze für die Designverifikation nicht praktikabel ist. Bei Vorhandensein einer formalen Wissensbeschreibung ist aus Gründen der Einfachheit die Generierung aus dieser dem Lernen vorzuziehen (vgl. Kap. 5.3). Für das MMI liegt aktuell jedoch keine formale Designspezifikation vor. Die im vorangegangenen Kap. vorgestellte JSON-Spezifikation kann ein komplettes GUI-Design beschreiben, ihr praktische Einsatz für das MMI ist aber schwierig. Wie in Kap. 2.2.2.3 erläutert, ist das MMI-Design in PSD-Dateien und PDF-Dokumenten definiert. Während Photoshop Schnittstellen für den Zugriff auf Dateiinformationen anbietet (s. Kap. 10 für Details), können die Inhalte der PDF-Dokumente nicht maschinell ausgewertet werden. Damit fehlt insbesondere der Zugriff auf die globalen Merkmale, die nur in PDF-Dokumenten spezifiziert sind. Photoshop ist außerdem kein geeignetes For-

mat, um zukünftig eine dahingehende Erweiterung vorzunehmen. Die PSD-Dateien bilden darüber hinaus lokale Merkmale nur beispielhaft ab und stellen daher keine vollständige Spezifikation des lokalen Designs dar. Letztlich soll das Designverifikationssystem auch in anderen Bereichen anwendbar sein und ein in PSD-Dateien beschriebener Designentwurf keine zwingende Voraussetzung darstellen (vgl. Anforderungen in Kap. 6.1).
Weniger einschränkend ist die Forderung, dass der Designentwurf in irgendeiner automatisch verwertbaren Form vorliegt, z.B. ganz allgemein in digitalen Bildern. Unter Anwendung von Bildverarbeitungsmethoden, bspw. denselben wie zur Gewinnung des Ist-Designs, können die einzelnen GUI-Inhalte ermittelt werden. Im Unterschied zu PSD-Dateien ist in einfachen Bildern die Hierarchie der GUI nicht enthalten. Um die GUI-Hierarchie herauszufinden, sind daher zusätzliche Verfahren notwendig (s. Kap. 2.3.5.4). Diese Verfahren benötigen allerdings wiederum Zusatzwissen über die Struktur der GUI. Zudem sind in bloßen Bildern wiederum keine globalen Merkmale definierbar. Das Problem der Unvollständigkeit bleibt somit bestehen.
Existieren nun Screenshots, die ausschnittsweise einen bestimmten Sachverhalt des Designs einer GUI vollständig darstellen, könnten daraus Bayes'sche Netze für die Verifikation von lokalen und globalen Merkmalen gelernt werden. Die Generierung von Teilen des Bayes'schen Netzes aus einer formalen Spezifikation wäre dann nicht mehr notwendig und der Generierungsprozess würde sich vereinfachen, da lediglich ein Generierungsverfahren verwendet werden müsste. Das Ziel ist es, einen Lernalgorithmus zu entwickeln, der minimal die globalen Zusammenhänge von GUI-Elementen lernt und im besten Fall das gesamte GUI-Design in einem Bayes'schen Netz abbildet. Das bisher verwendete Bayes'sche Netz ist aus gezielten Überlegungen heraus entstanden und bei der Generierung aus einer formalen Spezifikation existiert nur korrektes Design als Eingabe. Es ist dann Sache der Konfiguration, wie stark Abweichungen vom Soll als Fehler ins Gewicht fallen. Wenn stattdessen für das Lernen nicht nur korrekte, sondern auch Screenshots mit echten Designfehlern als Lerndaten gegeben sind, enthält das Bayes'sche Netz reale Wahrscheinlichkeitsverteilungen, die dem Anwendungsbereich entsprechen. Das Lernen kann dann zu neuen, interessanten Erkenntnissen führen. Ebenfalls können durch das Lernen globale Merkmale entdeckt werden, die bislang nicht explizit spezifiziert wurden, da sie gar nicht bekannt waren. Um dies zu erreichen, sollte mög-

lichst kein Zusatzwissen über strukturelle Eigenschaften des Bayes'schen Netzes beim Lernen verwendet werden.

9.2 Voraussetzungen

Bezogen auf den Anwendungsbereich Designverifikation sind für das Lernen von Bayes'schen Netzen zwei Schritte notwendig: das eigentliche Lernen und die Gewinnung der Lerndaten. Für die Gewinnung der Lerndaten sind das Vorhandensein von Screenshots, die das korrekte GUI-Design repräsentieren, und einer Bildverarbeitung essentielle Voraussetzungen.

9.2.1 Lerndaten

Gelernt werden soll stets ein bestimmter Sachverhalt, der innerhalb einer View definiert ist und durch eine Menge von Screenshots repräsentiert wird. Die Unterteilung in Views entspricht der allgemeinen Sichtweise von MMI-GUIs (s. Kap. 2.2.1). Eine wichtige Frage ist, wie die Screenshots bzw. Daten konkret aussehen müssen, damit das Lernen ein funktionsfähiges Bayes'sches Netz liefert.

9.2.1.1 Beispiel A

Referenz	PositionX	PositionY
play_enabled	370.0	304.0
play_pressed	370.0	304.0
play_rollover	370.0	304.0
play_rolloverpressed	370.0	304.0
play_disabled	370.0	304.0

Tabelle 9.1: Lokale Merkmale des Wizard-Icons *IconPlay*.

Aus fünf Screenshot-Ausschnitten der View *CDA Wizard Test* (vgl. Kap. 8.4), die die fünf Zustände des Wizard-Icons *IconPlay* darstellen (im Folgenden auch als *Beispiel* bzw. *Lerndaten A* bezeichnet), kann mittels Bildverarbeitung das lokale Design des Wizard-Icons extrahiert werden. Mit den in Tabelle 9.1 aufgelisteten

Merkmalen ist das lokale Soll-Design vollständig beschrieben und kann zum Lernen verwendet werden. Neben den Screenshots ist allerdings Zusatzwissen erforderlich, das eine Beziehung zwischen den in verschiedenen Screenshots erkannten Icons herstellt und besagt, dass es sich um dasselbe Icon in verschiedenen Zuständen handelt.

Korrekt	**Referenz_IconPlay**	**PositionX_IconPlay**	**PositionY_IconPlay**
wahr	play_enabled	370.0	304.0
wahr	play_pressed	370.0	304.0
wahr	play_rollover	370.0	304.0
wahr	play_rolloverpressed	370.0	304.0
wahr	play_disabled	370.0	304.0
falsch	anders	370.0	304.0
falsch	play_enabled	anders	304.0
falsch	play_pressed	anders	304.0
falsch	play_rollover	anders	304.0
falsch	play_rolloverpressed	anders	304.0
falsch	play_disabled	anders	304.0
falsch	anders	anders	304.0
falsch	play_enabled	370.0	anders
falsch	play_pressed	370.0	anders
falsch	play_rollover	370.0	anders
falsch	play_rolloverpressed	370.0	anders
falsch	play_disabled	370.0	anders
falsch	anders	370.0	anders
falsch	play_enabled	anders	anders
falsch	play_pressed	anders	anders
falsch	play_rollover	anders	anders
falsch	play_rolloverpressed	anders	anders
falsch	play_disabled	anders	anders
falsch	anders	anders	anders

Tabelle 9.2: Lerndaten zur Verifikation des Wizard-Icons *IconPlay* bestehend aus korrektem und fehlerhaften Design.

Wenn die Soll-Kombinationen als Lerndaten betrachtet werden, bildet jede Spalte die Zustandsmenge einer Variablen des zukünftigen Bayes'schen Netzes. Zusätzlich wird eine weitere Spalte hinzugefügt, die die Korrektheitsaussage repräsentiert. Hier wird deutlich, dass die Soll-Kombinationen unvollständige Lerndaten bilden, da sie nur korrektes Design darstellen und nicht alle möglichen Variablenzustände beinhalten. Für die Verwendung relativ einfacher Lernalgorithmen ist das Vorhandensein aller notwendigen Variablen und Zustände wichtig. Andernfalls handelt es sich um Lernen mit unvollständigen Daten, was weitaus problematischer ist (s. Kap. 5.3.2.6). Deshalb müssen die Daten, wie in Tabelle 9.2 dargestellt, um fehlerhaftes Design erweitert werden. Fehlerhaftes Design stellen genau die Kombinationen der Merkmalszustände dar, die nicht bereits als korrektes Design enthalten sind. In Tabelle 9.2 sind sie ab Zeile 7 aufgeführt und an dem Eintrag *falsch* in der Spalte *Korrekt* zu erkennen. Die Lerndaten A umfassen damit 24 Datensätze von denen die ersten fünf korrektes Design und die weiteren 19 fehlerhaftes Design repräsentieren.

9.2.1.2 Beispiel B

Für die Beschreibung des globalen Merkmals *Koexistenz* zwischen dem Wizard-Icon *IconPlay* und dem dazugehörigen Textelemet *TooltipTextPlay* in derselben View (im Folgenden auch als *Beispiel* bzw. *Lerndaten B* bezeichnet), sind sieben Screenshots erforderlich (s. Abbildung 7.4 in Kap. 7.3). Auch hier muss eine Beziehung zwischen den Informationen der einzelnen Screenshots durch Zusatzwissen hergestellt werden.
Die Lerndaten B repräsentieren ein Icon mit den lokalen Merkmalen *Referenz*, *PositionX* und *PositionY*. Als globales Merkmal ist die vom Iconzustand abhängige Existenz des Textelements *TooltipText* abgebildet. Das ergibt $6^1 \cdot 2^1 \cdot 2^1 \cdot 2^1 = 48$ Datensätze.

9.2.2 Lernverfahren

Zum Lernen kommen das in [Nürenberg 2012] beschriebene *BNSF-Tool* sowie WEKA (Bouckaert 2006-2007) zum Einsatz. Das BNSF-Tool stellt die K2- und die (R)HC-Suche, die Metriken MDL, AIC und Log-K2, und die LL-Parameterschätzung zur Verfügung. Die K2-Suche wird zudem parallelisiert durchgeführt. Als Ergän-

zung hierzu liefert WEKA die Bayes'sche Parameterschätzung. Alle hier verwendeten Algorithmen wurden in Kap. 5.3.2 beschrieben.

9.3 Beispiel

Am Beispiel der in Abschnitt 9.2.1 vorgestellten Lerndaten werden in den Abschnitten 9.3.1 und 9.3.2 die Metriken MDL, AIC und Log-K2 untersucht. Soweit nicht anders angegeben wird dabei stets die LL-Parameterschätzung und die K2-Suche verwendet. Für die K2-Suche ist die Angabe einer Variablenreihenfolge und der Maximalanzahl von Elternknoten erforderlich. In Abschnitt 9.3.3 werden aus weiteren Designinformationen Bayes'sche Netze gelernt.

9.3.1 Beispiel A - Lokale Merkmale eines Icons

Die erste Variablenreihenfolge ist: `Korrekt`, `IconPlay_Referenz`, `IconPlay_PositionX`, `IconPlay_PositionY`. D.h. die VV `Korrekt` ist der Startknoten und damit Wurzelknoten. Die zweite Variablenreihenfolge ist: `IconPlay_Referenz`, `IconPlay_PositionX`, `IconPlay_PositionY` und `Korrekt`.
Die Maximalanzahl an potentiellen Elternknoten ist drei, damit kann jeweils der letzte Knoten alle Vorgänger als Elternknoten besitzen.

9.3.1.1 Lerndaten A

Wenn die obigen 24 Datensätze aus Tabelle 9.2 und die erste Variablenreihenfolge verwendet werden, wird mit den zwei Metriken MDL und AIC das in Abbildung 9.1 dargestellte Netz gelernt. Für die Variable `IconPlay_Referenz` wurde keine Abhängigkeit zu einer anderen Variablen festgestellt. Das ist ein Fehler, da so die Eingabe des Merkmals `Referenz` keinerlei Auswirkung auf die Korrektheit hat. Zudem liefern diese Netze bei Eingabe der Soll-Position eine Korrektheitsaussage von lediglich 85% (s. Abbildung 9.1 unten rechts). Ein mit der Log-K2-Metrik gelerntes Netz unterscheidet sich von den anderen zwei in einer zusätzlichen Kante, die aus der Variablen `IconPlay_PositionY` einen direkten Kindknoten der Variable `IconPlay_PositionX` macht. Während die mit MDL und AIC gelernten Parameter konsistente CPTs ergeben, weist das mit der Log-K2-Metrik gelernte Netz hier einen weiteren Fehler auf: die Summe der Zeilenein-

träge für $\{Korrekt = wahr, IconPlay_PositionX = anders\}$ im CPT der Variablen `IconPlay_PositionY` beträgt 0.0 (s. Tabelle 9.3).

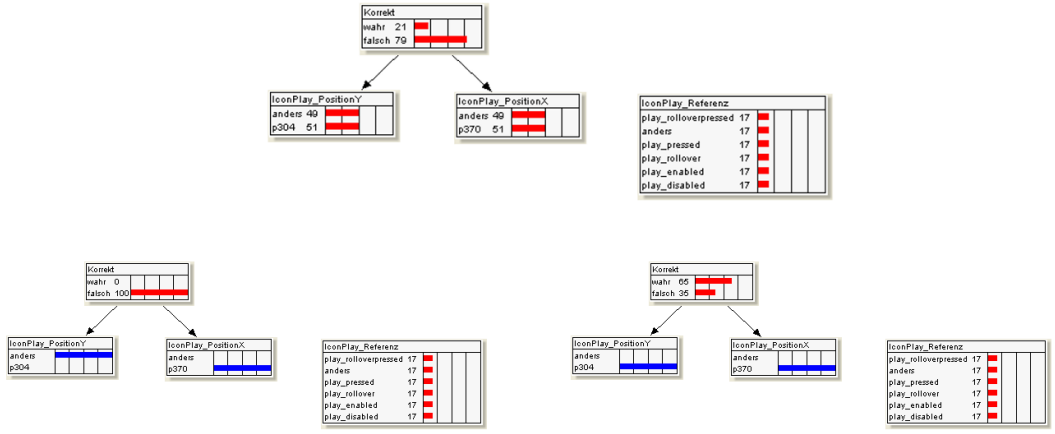


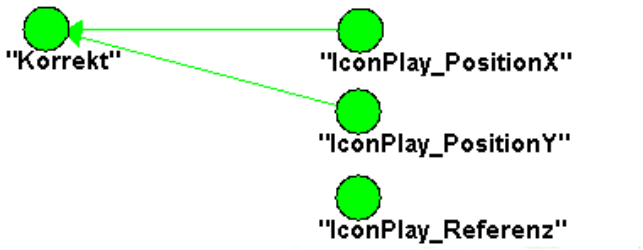
Abbildung 9.1: Mit den Metriken AIC und MDL gelerntes Bayes'sches Netz zum Beispiel A. Es wurde die erste Variablenreihenfolge verwendet, sodass die Variable `Korrekt` der Wurzelknoten ist. Oben: Netz mit initialen Parametern. Unten: Parameter bei gesetzten Evidenzen (blaue Balken). (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010))

IconPlay_PositionY|Korrekt, IconPlay_PositionX	**anders**	**304.0**
wahr, anders	0.0	0.0
falsch, anders	0.5	0.5
wahr, 370.0	0.0	1.0
falsch, 370.0	0.875	0.125

Tabelle 9.3: CPT der Variablen `IconPlay_PositionY`, der für das Beispiel A mit der Log-K2-Metrik gelernt wurde.

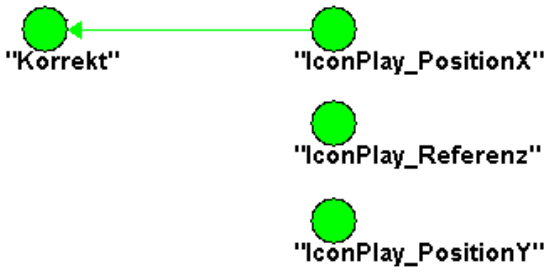
Bei Betrachtung der Daten wird deutlich, dass die Nulleinträge genau die Kombinationen betreffen, die in den Lerndaten nicht enthalten sind. Die übrigen Parameter spiegeln ebenfalls exakt die Häufigkeiten der gegebenen korrekten und fehlerhaften Merkmalskombinationen wider. Durch Wahl der VV als Wurzelknoten entstanden mit den Metriken AIC und MDL Netze, deren Variablen maximal paarweise miteinander verbunden sind. Diese gelernten Abhängigkeiten sind in den Lerndaten alle

vorhanden und so werden konsistente CPTs gelernt. Damit ist neben den Lerndaten auch die Variablenreihenfolge ein entscheidender Faktor für da Lernergebnis.
Bei Verwendung der zweiten Variablenreihenfolge entstehen mit allen drei Metriken konsistente Bayes'sche Netze. In Abbildung 9.2 sind das Bayes'sche Netz und die CPT-Einträge der VV dargestellt, die mit der AIC- und der Log-K2-Metrik gelernt werden. Bei Verwendung der MDL-Metrik entsteht das in Abbildung 9.3 gezeigte Bayes'sche Netz, das nur noch eine Kante besitzt. Die Fehler in den gelernten Netzen zeigen, dass die gegebenen Lerndaten nicht ausreichen, mit den verwendeten Lernverfahren alle gewünschten Beziehungen abzubilden und ein korrektes Bayes'sches Netz zu erhalten.
Die Rechenzeit beträgt mit jeder der drei Metriken rund 0,01 Sekunden.



Korrekt| PositionX, PositionY	wahr	falsch
anders, anders	0.0	1.0
anders, 370.0	0.0	1.0
304.0, anders	0.0	1.0
304., 370.0	0.833333...	0.166666...

Abbildung 9.2: Mit den Metriken AIC und Log-K2 gelerntes Bayes'sches Netz und CPT der Variablen `Korrekt` zum Beispiel A. Es wurde die zweite Variablenreihenfolge verwendet, sodass die Variable `Korrekt` ein Kindknoten ist. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001))

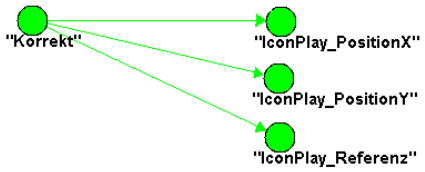


Korrekt| PositionX	wahr	falsch
anders	0.0	1.0
370.0	0.4166666...	0.5833333...

Abbildung 9.3: Mit der MDL-Metrik gelerntes Bayes'sches Netz und CPT der Variablen `Korrekt` zum Beispiel A. Es wurde die zweite Variablenreihenfolge verwendet, sodass die Variable `Korrekt` ein Kindknoten ist. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001))

9.3.1.2 Erweiterte Lerndaten A

Die Lerndaten werden erweitert, indem alle Kombinationen, die korrektes Design zeigen (die ersten fünf Datensätze in den ursprünglichen Daten), 100-fach eingetragen werden. D.h. die neuen Lerndaten umfassen 519 Datensätze. Die Rechenzeit verlängert sich dadurch nur unmerklich auf maximal 0,03 Sekunden. Für die K2-Suche wird erneut zuerst die erste Variablenreihenfolge verwendet. Abb. 9.4 zeigt das von den Metriken AIC und Log-K2 gelernte Netz. Für die Variable `IconPlay_Referenz` wurde nun auch eine Abhängigkeit zur Variable `Korrekt` gefunden. Alle CPTs sind konsistent. Zudem führen die Parameter nun zu eindeutigeren Ergebnissen, wenn Merkmale eingegeben werden (s. Abbildung 9.5). Durch die 100-fach korrekten Designeinträge in den Lerndaten ist der Unterschied zwischen korrektem und fehlerhaften Design vergrößert worden. Lediglich die MDL-Metrik funktioniert auch bei den erweiterten Lerndaten nicht zufriedenstellend und liefert erneut ein Netz ohne Verbindung zur Variable `IconPlay_Referenz`, die übrigen Parameter sind identisch mit den von AIC und LogK2 gelernten.



IconPlay_PositionX| Korrekt	anders	370.0
wahr	0.0	1.0
falsch	0.6319845...	0.3680154...

IconPlay_PositionY| Korrekt	anders	304.0
wahr	0.0	1.0
falsch	0.6319845...	0.3680154...

IconPlay _Referenz| Korrekt	play_ rollover pressed	anders	play_ pressed	play_ rollover	play_ enabled	play_ disabled
wahr	0.2	0.0	0.2	0.2	0.2	0.2
falsch	0.1579961...	0.2100192...	0.1579961...	0.1579961...	0.1579961...	0.1579961...

Abbildung 9.4: Mit den Metriken AIC und Log-K2 aus den erweiterten Lerndaten A gelerntes Bayes'sches Netz und CPTs. Es wurde die erste Variablenreihenfolge verwendet, sodass die Variable `Korrekt` der Wurzelknoten ist. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001))

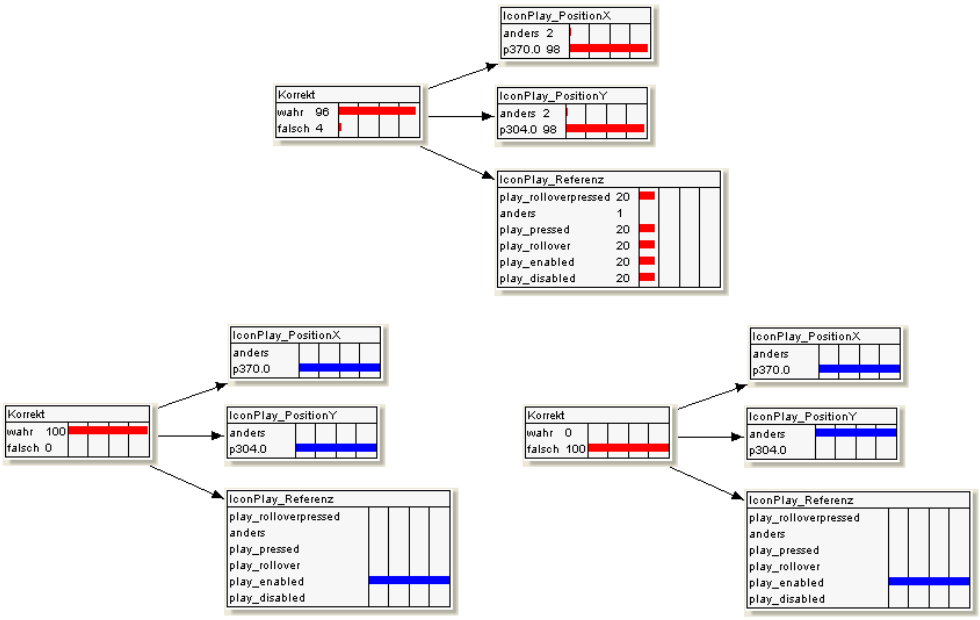
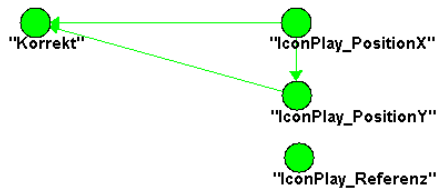


Abbildung 9.5: Inferenzberechnungen im Bayes'schen Netz, das mit den Metriken AIC und Log-K2 aus den erweiterten Lerndaten A gelernt wurde. (Netz erstellt mit BayesBuilder (*BayesBuilder* 2007-2010))

Mit der zweiten Variablenreihenfolge werden die in den Abbildungen 9.6 und 9.7 dargestellten Bayes'schen Netze gelernt. In diesem Fall liefern die Log-K2- und die MDL-Metrik ein identisches Netz. Die Parameter des CPT der VV sind in Tabelle 9.4 aufgeführt. Bei den erweiterten Daten führt die umgekehrte Variablenreihenfolge zu schlechteren Ergebnissen; alle Metriken liefern ein Bayes'sches Netz, in dem keine Abhängigkeitsbeziehung zur Variable `IconPlay_Referenz` besteht.
Da mit der K2-Suche und der MDL-Metrik kein korrektes Netz gelernt werden konnte, wird statt K2 nun (R)HC eingesetzt. Die (R)HC-Suche besitzt nicht die Einschränkung, dass eine feste Variablenreihenfolge betrachtet wird und der Suchraum ist daher größer. Da (R)HC zufallsbasiert ist, können in mehreren Durchläufen allerdings verschiedene Netze gelernt werden. Bei Beispiel A entspricht das gelernte Netz meist dem der K2-Suche. Bei etwa durchschnittlich jedem sechsten Durchlauf wird das in Abbildung 9.8 dargestellte Netz gelernt. Es besitzt eine weitere Kante, allerdings ist auch hier die Variable `IconPlay_Referenz` unverknüpft. (R)HC führt hier zu keinem besseren Ergebnis. Die Rechenzeit beträgt rund 0,25 Sekunden.

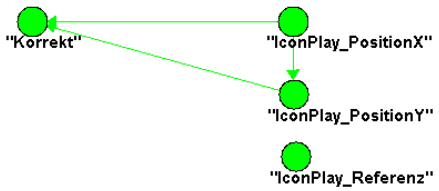


IconPlay_PositionX	anders	370.0
	0.0231213...	0.9768786...

IconPlay_PositionY| IconPlay_PositionX	anders	304.0
anders	0.5	0.5
370.0	0.01	0.99

IconPlay _Referenz	play_ rollover pressed	anders	play_ pressed	play_ rollover	play_ enabled	play_ disabled
	0.1984585...	0.0077071...	0.1984585...	0.1984585...	0.1984585...	0.1984585...

Abbildung 9.6: Mit der AIC-Metrik aus den erweiterten Lerndaten A gelerntes Bayes'sches Netz und CPTs. Es wurde die zweite Variablenreihenfolge verwendet, sodass die Variable `Korrekt` ein Kindknoten ist. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001))



IconPlay_PositionX	anders	370.0
	0.0231213...	0.9768786...

IconPlay_PositionY| IconPlay_PositionX	anders	304.0
anders	0.5	0.5
370.0	0.0115606...	0.9884393...

IconPlay _Referenz	play_ rollover pressed	anders	play_ pressed	play_ rollover	play_ enabled	play_ disabled
	0.1984585...	0.0077071...	0.1984585...	0.1984585...	0.1984585...	0.1984585...

Abbildung 9.7: Mit den Metriken Log-K2 und MDL aus den erweiterten Lerndaten A gelerntes Bayes'sches Netz und CPTs. Es wurde die zweite Variablenreihenfolge verwendet, sodass die Variable `Korrekt` ein Kindknoten ist. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001))

Korrekt|IconPlay_PositionY, IconPlay_PositionX	**wahr**	**falsch**
anders, anders	0.0	1.0
anders, 370.0	0.0	1.0
304.0, anders	0.0	1.0
304.0, 370.0	≈ 0.999	≈ 0.001

Tabelle 9.4: CPT der Variablen `Korrekt`, der für das Beispiel A und den erweiterten Lerndaten mit den Metriken AIC, Log-K2 und MDL gelernt wurde.

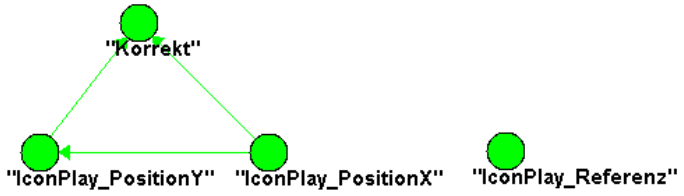


Abbildung 9.8: Mit der (R)HC-Suche und der MDL-Metrik aus den erweiterten Lerndaten A gelerntes Bayes'sches Netz. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001))

9.3.2 Beispiel B - Globale Merkmale

Die oben angegeben Lerndaten B werden ebenfalls erweitert, indem die Kombinationen, die korrektes Design repräsentieren, 100-fach eingetragen werden. Das führt zu 741 Datensätzen.
Das Lernen mit der AIC-Metrik benötigt rund 0,3 Sekunden, das Lernen mit der Log-K2- und der MDL-Metrik ist etwas schneller mit rund 0,1 Sekunden. Die Abbildungen 9.9 und 9.10 zeigen die gelernten Bayes'schen Netze. Mit der AIC und der Log-K2 wurde dasselbe Bayes'sche Netz gelernt. Die Abhängigkeit zwischen der Existenz des Textelements, der Icon-Referenz und der VV ist darin korrekt abgebildet. Die Position des Icons hat, ebenfalls den Lerndaten entsprechend, nur einen Einfluss auf die VV und ist unabhängig von der Icon-Referenz oder der Existenz des Textelements. Alle Parameter wurden ebenfalls geeignet gelernt. Wenn Soll-Merkmale eingegeben werden, führt das zu einer Korrektheit von über 99%. Wenn ein falsches Merkmal eingegeben wird, beträgt die Sicherheit für korrektes Design exakt 0%. Das mit der MDL-Metrik gelernte Netz besitzt zwei Kanten weniger und es fehlt die Beziehung für die globale Verifikation. D.h. dieses Netz bildet die Daten nicht wie gewünscht ab.

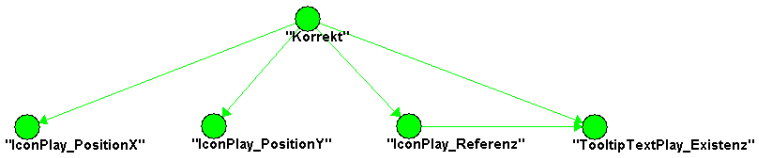


Abbildung 9.9: Mit den Metriken AIC und Log-K2 gelerntes Bayes'sches Netz zum Beispiel B. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001))

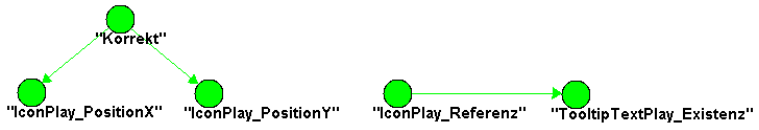


Abbildung 9.10: Mit der MDL-Metrik gelerntes Bayes'sches Netz zum Beispiel B. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001))

Da das gelernte Netz mit der MDL-Metrik nicht alle erforderlichen Kanten aufweist, wird nochmals die (R)HC-Suche angewendet. Meist resultiert das Bayes'sche

Netz, das bereits mit der K2-Suche gefunden wurde, etwa durchschnittlich bei jedem 20. Durchlauf ist allerdings das in Abbildung 9.11 dargestellte Netz das Ergebnis. Es besitzt eine weitere Kante, jedoch nicht die geforderte. Die Rechenzeit mit der (R)HC-Metrik beträgt rund 0,25 Sekunden.

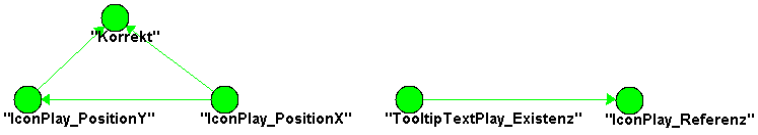


Abbildung 9.11: Mit der (R)HC-Suche und der MDL-Metrik gelerntes Bayes'sches Netz zum Beispiel B. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001))

9.3.3 Weitere gelernte Bayes'sche Netze

9.3.3.1 Koexistenz von Wizardicons

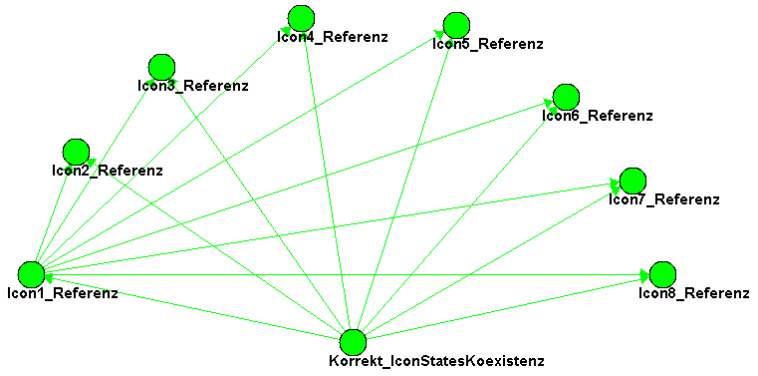


Abbildung 9.12: Mit der MDL-Metrik gelerntes Bayes'sches Netz zur Verifikation der Koexistenz der Zustände von sieben Wizardicons. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001))

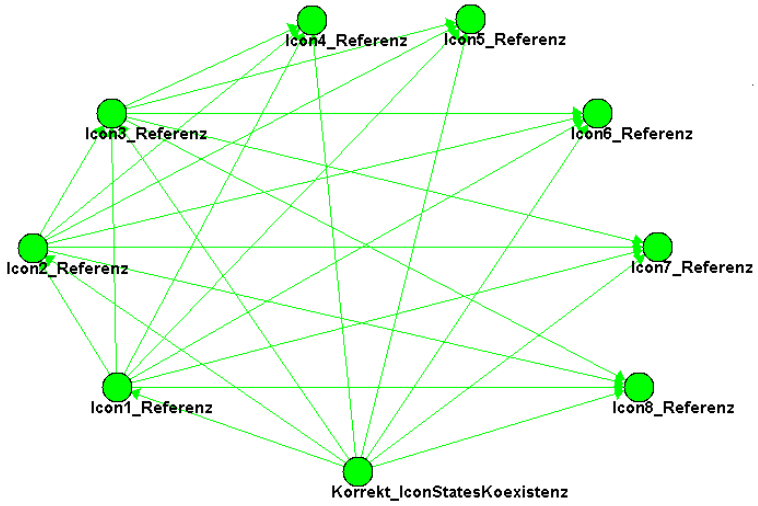


Abbildung 9.13: Mit der AIC-Metrik gelerntes Bayes'sches Netz zur Verifikation der Koexistenz der Zustände von sieben Wizardicons. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001))

Die Abbildungen 9.12 und 9.13 zeigen die Struktur zwei gelernter Bayes'scher Netze für die Verifikation der Koexistenz der Zustände von sieben Wizardicons[1]. Die Lerndaten enthalten alle möglichen Kombinationen jeweils einfach und umfassen 1.679.616 Datensätze. Das in Abbildung 9.12 dargestellte Netz wurde mit der MDL-Metrik gelernt, die Lernzeit lag bei rund 9,5 Minuten. In Abbildung 9.13 ist das mit der AIC-Metrik gelernte Netz dargestellt, das Lernen dauerte hier rund 2,8 Stunden. Beide Netze enthalten konsistente Parameter und bilden die Informationen wie gewünscht ab. Die Log-K2-Metrik konnte nicht verwendet werden, da die Rechenzeit bei einer derart großen Datenmenge immens hoch ist. Der Grund ist, dass ab einer Anzahl von rund 30.000 Datensätzen bei der LogK2-Metrik die Fakultäten gigantisch groß werden.

9.3.3.2 Icons und Textelemente

Abb. 9.14 zeigt ein Bayes'sches Netz, das aus Daten gelernt wurde, die die lokalen Merkmale von sieben Icons und zwei Textelementen (Tooltip) sowie zweimal die globale Beziehung zwischen einem Iconzustand und einem Textelement beschreiben. Es handelt sich um erweiterte Lerndaten, die zusätzlich auch noch alle

[1] In einem MMI-Wizardmenü darf sich nur maximal ein Icon in einem der Zustände *pressed*, *rollover* oder *rolloverpressed* befinden.

nicht möglichen Kombinationen beinhalten. Die Anzahl der Datensätze beträgt über 78.000. Es wurde die MDL-Metrik verwendet, mit der das Lernen ca. 89 Sekunden dauert.

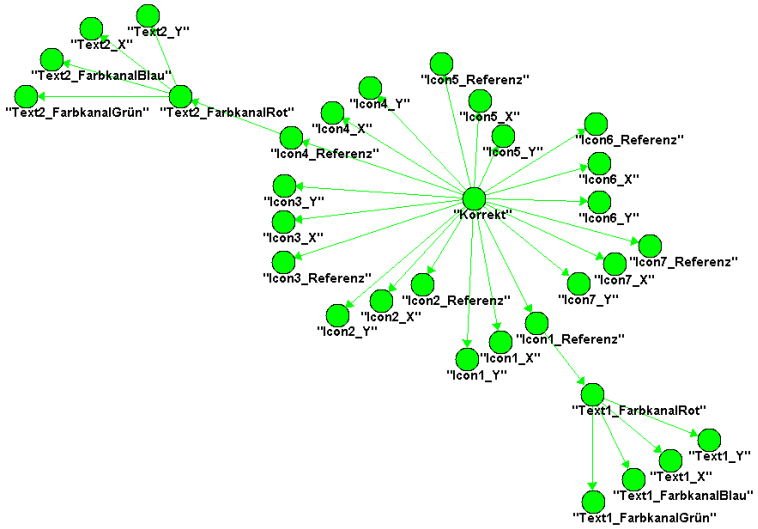


Abbildung 9.14: Mit der MDL-Metrik gelerntes Bayes'sches Netz zur Repräsentation der lokalen Merkmale von sieben Icons und zwei Textelementen sowie der Koexistenz zwischen jeweils einem Icon und einem Textelement. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001))

9.4 Begründung und Einordnung

Wichtige Kriterien für einen erfolgreichen produktiven Einsatz des Lernens ist die Qualität und Performanz der Lernverfahren sowie die Erklärbarkeit, die Korrektheit und die Fehlerabdeckung der gelernten Bayes'schen Netze. Bezugnehmend auf die vorgestellten Beispiele werden diese Punkte nun detailliert diskutiert.

9.4.1 Erzeugung von Lerndaten

Direkt aus korrekten Screenhots erworbene Lerndaten enthalten nur das Soll-Design. Wie die obigen Beispiele zeigen, ist das in der Regel zu wenig, um funktionierende Bayes'sche Netze zu lernen. Aus diesem Grund wurden die Lerndaten um fehlerhaftes Design erweitert. Die Erweiterung kann auf einfache Weise automatisiert werden. Um korrekte Verifikationsergebnisse zu erhalten, muss zudem die Häufigkeit von korrektem Design gegenüber dem fehlerhaften Design überwiegen.

Die Lernergebnisse in diesem Kapitel zeigen, dass mit diesen künstlich erweiterten Lerndaten funktionierende Bayes'sche Netze gelernt werden, die die geforderten Informationen wie gewünscht repräsentieren. Die Vorgehensweise ist dennoch fragwürdig. Durch die künstliche Erzeugung der Lerndaten wird eine Annahme über das Aussehen der Lerndaten getroffen, die evtl. von der Realität stark abweichen kann. Damit ist die Erzeugung der Lerndaten sehr fehleranfällig. Das Ideal wäre, Screenshots von sowohl korrektem als auch fehlerhaften Design als Basis zu haben. Auch bei der Designverifikation ist, wie so oft im Bereich des maschinellen Lernens, das Fehlen realer Lerndaten, die bereits ausreichend sind, ein Hauptproblem.

9.4.2 Qualität und Performanz der Lernverfahren

In Kap. 5.3.2 wurden verschiedene Lernverfahren für Bayes'sche Netze erläutert, die sich jeweils aus einer Metrik, einem Suchverfahren und einer Parameterschätzung zusammensetzen.

9.4.2.1 Metriken

Wie an den Beispielen zu erkennen ist, liefert die MDL-Metrik im Vergleich zu der AIC und der Log-K2-Metrik oftmals schwächer verknüpfte Netze. Die MDL-Metrik prüft bedingte Abhängigkeiten zwischen Variablen härter als die anderen Metriken. Sie hat zum Ziel, das kleinstmögliche Bayes'sche Netz zu finden und eine Überparametrisierung zu verhindern (vgl. Kap. 5.3.2.4). Auf diese Weise kann allerdings schneller eine Unterparametrisierung auftreten, was insbesondere bei den oben verwendeten nicht-erweiterten Lerndaten der Fall ist.

Performanz Bezüglich der Rechenzeit ist das Lernen für kleine Lerndaten sehr praktikabel und bewegt sich im Millisekundenbereich. Bei derart kleinen Lerndaten lassen sich zudem kaum merkliche Unterschiede in der Rechenzeit bei den untersuchten Metriken feststellen. Bei großen Datenmengen steigt die Rechenzeit mit der LogK2-Metrik aufgrund der großen Fakultäten enorm an. Die LogK2-Metrik ist deshalb, im Gegensatz zu MDL und AIC, nur für Datenmengen bis rund 30.000 Datensätze praktikabel anwendbar. Auch die AIC-Metrik benötigt bei großen Lerndaten deutlich mehr Rechenzeit als MDL. Beim Beispiel in Abschnitt 9.3.3.1 dauerte das Lernen basierend auf 1.679.616 Datensätzen mit der AIC-Metrik rund 2,8

Stunden und mit MDL nur rund 9,5 Minuten. Während bei kleinen Datenmengen meist mit der AIC- und der Log-K2-Metrik die besten Ergebnisse erzielt wurden, ist bei großen Datenmengen die MDL-Metrik vorteilhafter. Die Unterparametrisierung als Nachteil der MDL-Metrik konnte bei großen Datenmengen für die Beispiele zudem nicht beobachtet werden.

9.4.2.2 Suchverfahren

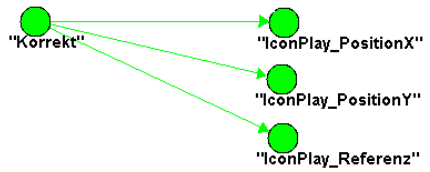
Die K2-Suche arbeitet durch Vorgabe der Variablenreihenfolge auf einem eingeschränkten Suchraum. Das führt zu einer effizienten Suche, kann aber auch zu suboptimalen Netzen führen. Da im Gegensatz dazu bei der (R)HC-Suche diese Einschränkung nicht verwendet wird, kann (R)HC zu besseren Ergebnissen führen. Für die obigen Beispiele konnte diese Eigenschaft allerdings nicht beobachtet werden. Ein Nachteil von (R)HC ist zudem, dass diese Verfahren zufallsbasiert sind und pro Durchlauf ein anderes Ergebnis möglich ist.

9.4.2.3 Parameterschätzung

Für alle Beispiele wurde die LL-Parameterschätzung verwendet. Wie in Kap. 5.3.2.5 erläutert, bildet diese die Häufigkeiten der Lerndaten exakt ab. Wenn die Lerndaten spärlich sind, berechnet die LL-Parameterschätzung daher auch spärliche CPTs. Fehlende Abhängigkeiten, wie bspw. zwischen den Variablen `Korrekt` und `Referenz` mit den einfachen Lerndaten A, begründen sich ebenfalls darin. Die Häufigkeiten aller Abhängigkeiten sind in diesem Fall sehr ähnlich zueinander, weswegen keine ausreichend starke Korrelation gefunden werden kann. Erst mit Verwendung erweiterter Lerndaten, in denen das Verhältnis zwischen korrektem und fehlerhaften Design vergrößert ist, kann bei Beispiel A wenigstens mit zwei der drei Metriken, AIC und LogK2, eine Abhängigkeit gefunden werden. Korrekterweise ist die Wahrscheinlichkeit für die Kombination `Korrekt`=*wahr* und `Referenz`=*anders* in diesen CPTs 0.0, da sie nicht in den Lerndaten enthalten ist.

Vermeidung von spärlichen CPTs Da spärliche Lerndaten nicht nur zu spärlichen CPTs, sondern auch zu inkonsistenten CPTs führen können (s. z.B. Tabelle 9.3), sollten Nulleinträge vermieden werden.
Eine Möglichkeit ist es, die Lerndaten mit den fehlenden Kombinationen zu vervollständigen. Um geeignete Wahrscheinlichkeitsverteilungen zu erhalten, muss al-

lerdings auch das Verhältnis zwischen den echten Kombinationen und den übrigen Kombinationen[2] angepasst werden. Bspw. führt die Verwendung von 100-mal echten Kombinationen und einmal übrigen Kombinationen zu den CPT-Einträgen 0.99 und 0.01. Die auf diese Weise erweiterten Lerndaten A umfassen dann 2828 Datensätze. Bei umfangreicheren Soll-Informationen führt diese Technik schnell zu sehr großen Datenmengen und ist daher nicht praktikabel.
Neben der LL-Schätzung wurde in Kap. 5.3.2.5 die Bayes'sche Parameterschätzung vorgestellt. Diese fügt den Lerndaten Rauschen hinzu und verhindert damit spärliche CPTs. Das WEKA-Tool (Bouckaert 2006-2007) bietet diese Art von Parameterschätzung an. Exemplarisch zeigt Abbildung 9.15 ein Bayes'sches Netz, das mit WEKA aus den Lerndaten A gelernt wurde.



IconPlay_PositionX| Korrekt	370.0	anders
wahr	0.9166666...	0.0833333...
falsch	0.375	0.625

IconPlay_PositionY| Korrekt	304.0	anders
wahr	0.9166666...	0.0833333...
falsch	0.375	0.625

IconPlay _Referenz| Korrekt	play_ enabled	play_ disabeld	play_ rollover	play_ rollover pressed	play_ pressed	anders
wahr	0.1875	0.1875	0.1875	0.1875	0.1875	0.0625
falsch	0.1590909...	0.1590909...	0.1590909...	0.1590909...	0.1590909...	0.2045454...

Abbildung 9.15: Mit WEKA basierend auf den einfachen Lerndaten A gelerntes Bayes'sches Netz. Durch Verwendung der Bayes'schen Schätzung beinhaltet das Netz trotz spärlichen Lerndaten keine spärlichen CPTs. (Netz erstellt mit JavaBayes (Gagliardi Cozman 2001))

Obwohl diese Lerndaten spärlich sind, wird ein Bayes'sches Netz gelernt, das

[2]Echte Kombinationen sind diejenigen Datensätze, die tatsächlich korrektes oder fehlerhaftes Design repräsentieren. Die übrigen Kombinationen stellen unsinnige Datensätze dar, die an sich gar nicht möglich sind. Die übrigen Kombinationen sind bei den Lerndaten genau die Komplementärmenge, die durch Negierung der Korrektheitsaussage erhalten wird.

keine spärlichen CPTs beinhaltet und alle Informationen korrekt abbildet. Bspw. wurde für die in den Lerndaten nicht enthaltene Kombination (`Korrekt`=*wahr*, `Referenz`=*anders*) durch Rauschen eine Wahrscheinlichkeit von 0.0625 hinzugefügt. Die Spalteneinträge für `Korrekt`=*falsch* sind hingegen identisch zu den mit LL gelernten Netzen, da hier kein Rauschen hinzugefügt werden musste, um fehlende Kombinationen zu ersetzen. Für die Suche wurde K2 verwendet und als Metrik MDL[3]. Wie zu erkennen ist, kann unter Verwendung der Bayes'schen Schätzung mit der MDL-Metrik, die leicht zu unterparametrisierten Netzen führt, ein korrektes Netz gelernt werden. Analog lassen sich mit der Bayes'schen Schätzung auch für die weiteren obigen Beispiele Netze mit nicht-spärlichen CPTs lernen. WEKA funktioniert aufgrund begrenztem Arbeitsspeichers allerdings nur bis etwa einer Datenmenge von 20.000 Datensätzen und ist damit nicht für sehr große Lerndaten einsetzbar.

Gewichtung von Informationen Die mit der LL-Schätzung gelernten CPTs spiegeln die Häufigkeiten der Kombinationen in den Lerndaten wider. Deshalb führt ein einziges falsches Merkmal bereits zu 100% falschem Gesamtdesign. D.h. die Kombination der Korrektheitsaussagen ist eine andere als die in Kap. 7.1.3.2 verwendete und gibt keinen Rückschluss auf die Anzahl falscher Merkmal. Durch das hinzugefügte Rauschen bei der Bayes'schen Schätzung wird dieser Effekt abgeschwächt. Eine weitere Möglichkeit ist die Änderung der Lerndaten. Indem die Häufigkeit der falschen Korrektheiten variiert wird, kann z.B. eine anteilige Kombination erreicht werden (s. Kap. 7.1.3.2).
Gelernt wird nur das, was in den Lerndaten enthalten ist. Das gilt insbesondere auch für die Variablen und ihre Zustände. Die Gewichtung von Soll-Abweichungen ist in den Lerndaten nicht enthalten, wird nicht gelernt und ist so auch nicht im Bayes'schen Netz enthalten. Die gelernten CPTs können allerdings nachträglich um Soll-Abweichungen erweitert werden. Eine andere Möglichkeit wäre es, die Lerndaten um diese Informationen zu erweitern. Dadurch vergrößern sich die Lerndaten jedoch um ein Vielfaches, was die Rechenzeit enorm erhöht.

[3]Die AIC- und die Log-K2-Metrik konnten nicht getestet werden, da WEKA diese nicht unterstützt.

Performanz Sowohl LL als auch die Bayes'sche Schätzung führen mitunter zu Parametern, die unnötigerweise sehr viele Nachkommastellen aufweisen. Ein nachträgliches Runden auf bspw. drei Nachkommastellen spart Speicherplatz und reduziert die Zeit für die Inferenzberechnungen.

9.4.2.4 Komplexität

In 9.16 sind ein paar der existierenden Verfahren für das Parameter- und das Strukturlernen Bayes'scher Netze veranschaulicht. Die gelbmarkierten Methoden wurden in Kap. 2.2 vorgestellt und bei den obigen Beispielen angewendet. Diese decken nur einen geringen Teil der aufgeführten Methoden ab und auch die Abb. stellt keine vollständige Übersicht aller existierenden Methoden dar. Bereits die Themen Suchalgorithmen und Metriken sind ein weites Feld (vgl. Russell u. a. 1996). Letztendlich ist das Zusammenspiel von Lerndaten, Metrik, Suchverfahren und Parameterschätzung ausschlaggebend für das Lernergebnis. Bezugnehmend auf die betrachteten Beispiele führt z.B. die MDL-Metrik mit der LL-Schätzung oft zu unterparametrisierten Netzen, wohingegen unter Verwendung der Bayes'schen Schätzung geeignete Netze gelernt werden.

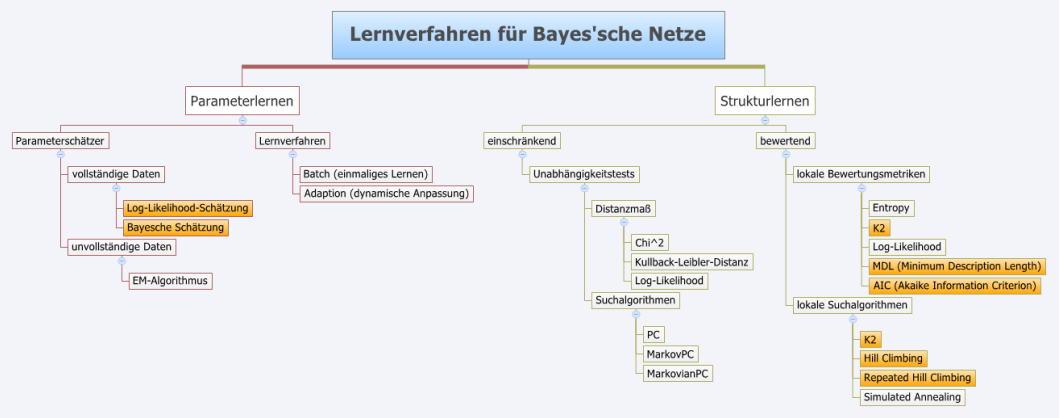


Abbildung 9.16: Auswahl an Verfahren für das Parameter- und das Strukturlernen Bayes'scher Netze. Die gelbmarkierten Verfahren wurden in dieser Arbeit angewendet.

9.4.3 Erklärbarkeit

Beim Vergleich mit dem in Kap. 7 vorgestellten manuell erzeugten Bayes'schen Netz fällt auf, dass die gelernten Netze stets nur eine VV besitzen und keine Vergleichsschicht mit EVs und vermittelnden Variablen. Das begründet sich darin, dass in den Lerndaten ebenfalls nur eine VV vorkommt, nämlich die, die die Gesamtkorrektheit repräsentiert. Mit diesen Netzen kann deshalb keine konkrete Angabe über Fehlerursachen gemacht werden. Das Hinzufügen weiterer Variablen zu den Lerndaten ist möglich, verkompliziert aber die Datenerstellung und führt vor allem zu einer immensen Vergrößerung der Datenmenge. Die Literatur, wie z.B. [Wittig 2003], zeigt, dass es auch auf andere Weise möglich ist gelernten Netzen einen höheren Grad an Erklärbarkeit bereits während des Lernens zu geben. Diese Ansätze sind allerdings noch weitestgehend wenig erforscht. Damit ist das Lernen von erklärbaren Bayes'schen Netzen problematisch und ein eigenes umfangreiches Gebiet.

9.4.4 Reduzierung des Lernaufwands

Die Formel 9.1 zeigt die Berechnung der Anzahl der Datensätze für das Beispiel A.

$$|I_r|^{|I|} \cdot |I_x|^{|I|} \cdot |I_y|^{|I|} = 6^1 \cdot 2^1 \cdot 2^1 = 24 \tag{9.1}$$

Mit $|I_r|$ Anzahl der Zustände von MV *Icon_Referenz*, $|I_x|$ und $|I_y|$ Anzahl der Zustände der MVs *Icon_XPosition* und *Icon_YPosition*. $|I|$ bezeichnet die Anzahl der betrachteten Icons, in diesem Fall 1.
Anhand der Formel wird klar, dass das für komplexere GUIs eine Unmenge von Datensätzen bedeuten muss. Für acht Wizard-Icons ergeben sich bereits $6^8 \cdot 2^8 \cdot 2^8 =$ 110.075.314.176 Soll-Kombinationen. Eine derart große Datenmenge kann nicht praktikabel verarbeitet werden. Daher ist es keinesfalls möglich direkt durch Lernen ein Bayes'sche Netz zu erzeugen, dass das GUI-Design eines vollständigen Wizardmenüs abbildet oder gar eines kompletten Infotainmentsystems.
Aufgrund der immensen Komplexität des Lernens und fehlender geeigneter Lerndaten werden nun Ansätze erläutert, um den Lernaufwand zun reduzieren.

9.4.4.1 Adaption

Eine einfache Möglichkeit ist es, ein Bayes'sches Netz einmalig für eine bestimmte Verifikation zu lernen und dann für ähnliche Verfikationen zu adaptieren. Dies bietet sich bspw. für die lokalen Merkmale von Wizardicons an. Die Struktur des Bayes'schen Netzes sowie die CPTs können für die Verifikation jedes Icons verwendet werden, lediglich die Zustände der Merkmalswerte (Position und Referenz) sind für jedes Icon unterschiedlich. In vielen Fällen sind daher sehr umfangreiche Lerndaten gar nicht erforderlich.

9.4.4.2 Kombination

Ein aus einer formalen Spezifikation generiertes Bayes'sches Netz kann anschließend durch Lernen erweitert werden, um bspw. globale Merkmale abzubilden. Durch die Bereitstellung von festen Netzteilen kann zum einen stärker die Netzstruktur beeinflusst werden; die Verwendung von vermittelnden Variablen zur Erhöhung der Interpretierbarkeit stellt hier kein Problem dar. Zum anderen werden die Lerndaten stark reduziert, da nicht mehr alle Kombinationen beachtet werden müssen. Wie die Literatur zeigt (z.B. Wiegerinck, Kappen und Burgers 2010, Borth und Hasseln 2002), sind in der Praxis oft Mischformen zwischen manueller Erstellung, der Generierung aus einer formalen Spezifikation und dem datenbasierten Lernen im Einsatz. Dabei gibt es zum einen die Möglichkeit, basierend auf Expertenwissen ein Bayes'sches Netz manuell oder teilautomatisch aufzubauen, oder zum anderen das Bayes'sche Netz zuerst aus Daten zu lernen und anschließend mit Expertenwissen zu vervollständigen bzw. zu korrigieren. In Anbetracht der eigenen Erkenntnisse und der Literatur ist das Lernen Bayes'scher Netze direkt aus gegeben Daten zwar theoretisch möglich, bei den meisten realen Gegebenheiten aber nicht exklusiv einsetzbar.
In Abbildung 9.17 ist ein Beispiel für ein kombiniertes Bayes'sches Netz dargestellt. Die gelben Variablen repräsentieren die MVs für die lokalen Merkmale. Zusammen mit den orangefarbenen Variablen (EVs und VVs für die lokale Verifikation) bilden sie das Bayes'sche Netz, das aus der Spezifikation generiert werden kann. Die globale Verifikation zwischen der Existenz eines Wizard-Icons und einem Tooltip-Textelement wird durch die rote VV repräsentiert. Die Parameter für die globale Beziehung wurden gelernt und dem Netz hinzugefügt. Ein weiterer Vorteil nur glo-

bale Merkmale lernen zu müssen ist, dass für die Beschreibung globaler Merkmale die Lerndaten meist kleiner sind als bei lokalen Merkmalen. Bei den globalen Merkmalen *Koexistenz* und *Gegenseitiger Ausschluss* sind insbesondere keine Soll-Abweichungen erforderlich.

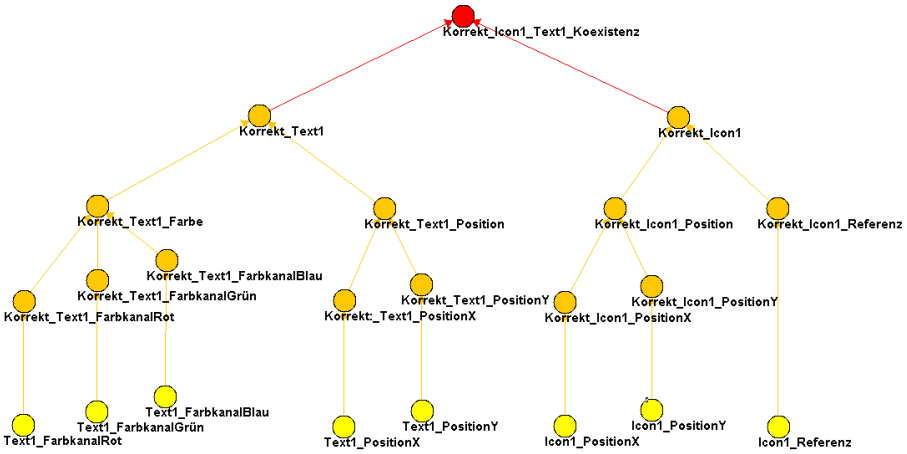


Abbildung 9.17: Kombiniertes Bayes'sches Netz aus der Spezifikation generierten (gelb und orangefarben) und gelernten (rot) Teilnetzen.

9.4.4.3 Abhängigkeiten

Wie im Beispiel aus Abschnitt 9.3.3.2 zu sehen, führen bereits die lokalen Merkmale von sieben Wizard-Icons und zwei Textelementen zu einer immensen Datengröße von über 78.000 Datensätzen. Die VV `Korrekt` soll in dem gelernten Netz sowohl die Korrektheitsaussage für die lokalen Merkmale, als auch die Koexistenz zwischen zwei Icons und ihrem dazugehörigen Textelement vereinen. Nicht nur für die Erklärbarkeit des Netzes ist diese Vorgehensweise problematisch, sondern sie führt auch, wie das Beispiel zeigt, zu einer ungeeigneten Netzstruktur. Die Trennung in einzelne Netze pro Icon und Textelement, um jeweils die lokalen Merkmale zu verifizieren, sowie zwei Netze für die Verifikation der zwei Koexistenzen löst das Problem. Die jeweils benötigten Lerndaten werden dabei deutlich reduziert, da pro Betrachtungsweise nicht mehr alle Variablen eine Rolle spielen.
Wenn nur mittels Bildverarbeitung gewonnene Informationen aus Screenshots vorliegen, fehlt normalerweise das Wissen, welche Kombinationen von Elementen und Merkmalen wichtig sind und welche nicht. Die Frage ist daher, ob eine Möglichkeit besteht, dieses Wissen lediglich aus den Informationen der Screenshots zu ermit-

teln. In Kap. 5.4.1 wurde die Kullback-Leibler-Distanz für den Test auf Abhängigkeit zweier Zufallsvariablen vorgestellt:

$$I(X,Y) = \sum_{1}^{n} \mathcal{P}(X,Y) \cdot log \frac{\mathcal{P}(X,Y)}{P(X) \cdot P(Y)}$$

$I(X,Y) = 0$ genau dann, wenn die Variablen X und Y unabhängig voneinander sind. Dieses Distanzmaß könnte paarweise auf die in einer Menge von Screenshots gefundenen Elemente angewendet werden. Nur wenn eine Abhängigkeit erkannt wird, ist die Erstellung von Lerndaten und das Lernen eines Bayes'schen Netzes erforderlich. D.h. durch Anwendung eines Distanzmaßes werden Gruppen von zueinander abhängigen Variablen ermittelt, deren Zustände eine Lerndatenmenge bilden.
Aus praktischer Sicht ist diese Vorgehensweise sehr aufwendig. Bei fehlendem Wissen ist es aber evtl. die einzige Möglichkeit, die Lerndaten und damit die Rechenzeit des Lernens stark zu reduzieren.

9.4.5 Korrektheit

Die Korrektheit der gelernten Bayes'schen Netze wurde in diesem Kapitel nur durch das direkte Testen der Netze geprüft. Die in Kap. 5.4.2 vorgestellten Methoden wie z.B. das Bootstrap-Verfahren und die Kreuz-Validierung wären umfangreichere, aber auch aufwendigere Möglichkeiten. Daneben ist auch eine Prüfung im produktiven Einsatz möglich. Die Gütekriterien aus Kap. 5.4.2.4 können explizite Aussagen über die Korrektheit eines Bayes'schen Netzes geben. Die gelernten Netze für die Designverifikation besitzen dafür eine vorteilhafte Struktur, da nur das Ergebnis der VV `Korrekt` interessiert und bewertet werden muss.
Ein generelles Problem beim Lernen Bayes'scher Netze ist die Tatsache, dass dem Lernverfahren in der Regel die Bedeutung der einzelnen Variablen nicht bekannt ist, z.B. bei welchen Variablen es sich um Ein- und Ausgabevariablen handelt. D.h. ein gelerntes Netz für die Designverifikation kann eine Struktur aufweisen, die nicht geeignet dafür ist alle Designmerkmale einzugeben und das Ergebnis an der Variablen `Korrekt` auszulesen. Manche MVs sind so verknüpft, dass ihre Zustandsänderung keinen Einfluss mehr auf `Korrekt` hat, wenn andere MVs bereits gesetzt sind (s. Beispiel in Abschnitt 9.3.3.2). Darüber hinaus können in Variablen, die Elternknoten besitzen, nicht auf einfache Weise unsichere Evidenzen eingegeben werden, sondern machen die Verwendung von virtuellen Variablen notwendig (vgl. Kap.

5.2.3). Die Beeinflussung der Netzstruktur durch Angabe der Variablenreihenfolge bei der K2-Suche kann hilfreich sein, löst aber oft nicht vollständig diese Problematik, wie das Beispiel in Abschnitt 9.3.3.2 verdeutlicht.
Die Komplexität des Lernens und die Vielzahl an Freiheitsgraden macht in jedem Fall eine Korrektheitsprüfung für die gelernten Bayes'schen Netze notwendig. Wie aus den Beispielen in diesem Kap. hervorgeht, können eine ungeeignete Wahl des Lernverfahrens und spärliche Lerndaten u.a. zu inkonsistenten CPTs und fehlenden Abhängigkeiten führen. Unterschiedliche und teilweise korrekte Ergebnisse verschiedener Lernverfahren basierend auf derselben Datenmenge zeigen außerdem, dass die Lerndaten an sich die Informationen korrekt repräsentieren. Manche Abhängigkeiten sind nur zu schwach, um von bestimmten Metriken identifiziert zu werden oder werden durch das Suchverfahren ausgeschlossen, da statt dem globalen Optimum nur ein lokales Optimum gefunden wird. Zufallsbasierte Suchverfahren können letzteres verhindern. Da zufallsbasierte Verfahren pro Durchlauf ein unterschiedliches Ergebnis liefern können, ist es zudem erforderlich in der Ergebnismenge wiederum das beste Netz zu ermitteln. Die Rechenzeit wird dadurch länger und eine optimale Anzahl der Wiederholungen ist nicht bekannt.

9.4.6 Fehlerabdeckung

Der Vorteil des Lernens gegenüber der Generierung aus einer formalen Spezifikation ist die Möglichkeit Information, die nur implizit in Daten beschrieben ist, zu finden. Daher kann die Fehlerabdeckung für die Designverifikation durch Lernen vergrößert werden. Genau wie bei der Generierung eines Bayes'schen Netzes aus einer formalen Spezifikation, kann ein durch Lernen erzeugtes Bayes'sches Netz allerdings auch nur die Informationen abbilden, die in den Daten enthalten sind. Das Problem, dass in der Praxis Bilder vom Designentwurf oftmals nur beispielhaft die GUI skizzieren bzw. kaum Möglichkeiten bestehen, Screenshots einer vollständigen GUI zu erwerben, die sowohl korrektes als auch fehlerhaftes Design darstellen, bleibt bestehen. Das Lernen vermeidet damit die Notwendigkeit von bestimmten Zusatzwissen nicht.

9.5 Schlussfolgerung

Der grundlegende Beweggrund, Bayes'sche Netze für die Designverifikation zu lernen, ist das Fehlen einer formalen Spezifikation des Soll-Designs für das MMI. Eine manuelle Erstellung der Bayes'schen Netze ist aufgrund der Komplexität der MMI-GUI zudem zu aufwendig. Die Lerndaten stammen aus Screenshots, die korrektes Design repräsentieren. Ohne weitere Vorkehrungen wird mit den gegebenen Verfahren allerdings ein Bayes'sches Netz gelernt, das nur die Informationen beinhalten, die die Lerndaten liefern. Es entstehen spärliche CPTs und eine andere Gewichtung der Korrektheitsaussagen oder gar Soll-Abweichungen ist in diesen gelernten Bayes'schen Netzen nicht enthalten. Darüber hinaus stellen gelernte Netze in der Regel die Informationen sehr kompakt dar. Das wirkt sich negativ auf die Effizienz aus und schränkt die Erklärbarkeit stark ein. Daher sind Verbesserungen notwendig. Wie gezeigt wurde, können spärliche CPTs relativ einfach durch den Einsatz einer geeigneten Parameterschätzung, die die Daten verrauscht, vermieden werden. Das Lernen einer höheren Erklärbarkeit bedarf umfangreicherer Techniken, die bislang kaum erforscht sind. Gängiger ist das Lernen basierend auf einer vorgegeben Grundstruktur oder das nachträgliche Hinzufügen weiterer Informationen. Ersteres wurde in Abschnitt 9.4.4.2 an einem konkreten Beispiel erläutert. Im ungünstigsten Fall können solche Vorgaben allerdings eine zu große Einschränkung bedeuten und das Lernen eines optimalen Netzes verhindern. Für eine automatisierte Erweiterung müssen die zusätzlichen Informationen, z.B. Soll-Abweichungen, zudem formal vorliegen. Wenn eine formale Spezifikation des GUI-Designs vorliegt, ist es daher stets vorteilhafter vollständig auf das Lernen zu verzichten und das Bayes'sche Netz komplett daraus zu generieren. Ein weiterer Grund, möglichst viele Informationen aus einer formalen Spezifikation zu gewinnen, ist die Komplexität des Lernens Bayes'scher Netze. Wie Kap. 5.3.2 und Abschnitt 9.4.2.4 zeigten, stehen für das Lernen eine Vielzahl von Metriken, Suchalgorithmen und Parameterschätzer zur Verfügung, die zudem auf verschiedene Weise miteinander kombiniert werden können. Je nach Auswahl der Verfahren und Umfang der Lerndaten, kann das Lernergebnis sehr unterschiedlich ausfallen. Bei großen Lerndaten beeinflusst darüber hinaus die Wahl des Lernverfahrens und der Software entscheidend die Efizienz.

Die vielen Freiheitsgrade machen das Lernen Bayes'scher Netze schwer kontrol-

lierbar. Im Rahmen der Dissertation konnte nur eine Teilmenge aller Möglichkeiten untersucht werden. Über die Tauglichkeit des Lernens eines Bayes'schen Netzes zur Repräsentation von GUI-Design kann deshalb hier keine endgültige und umfassende Aussage getroffen werden. Daneben werden auch in der Literatur bislang keine konkreten Vorgaben darüber gemacht, wann welche Verfahren zum Einsatz kommen und wie die Lerndaten aussehen.
Schließlich ist das grundlegende Problem nicht, eine formale Spezifikation aus Daten zu generieren oder ein Bayes'sches Netz direkt aus Daten zu lernen, sondern die Existenz geeigneter Daten. Dieses Problem wird auch durch Lernen nicht gelöst. Die Bereitstellung einer Menge an Screenshots, die jedes mögliche korrekte und fehlerhaftes Design einer GUI repräsentiert, wäre der Idealfall, ist aber nicht realistisch (vgl. Kap. 6.3.3.3). Aus diesem Grund kann das Lernen die Informationen über das Soll-Design lediglich erweitern und eine formale Spezifikation nicht vollständig ersetzen. Die Beispiele in diesem Kapitel zeigen damit, dass Lernen prinzipiell möglich ist und wie die Lerndaten dafür aussehen müssen. Für die Designverifikation kann es in eingeschränkten Bereichen eingesetzt werden, um die unvollständige Designspezifikation zu ergänzen.

Kapitel 10

Integration in die MMI-Testautomatisierung

Da das Verifikationssystem für die Verifikation der GUI des Audi Infotainmentsystems MMI entwickelt wurde, wird in diesem Kapitel kurz auf diese praktische Anwendung eingegangen. In Abschnitt 10.1 wird die Integration in den Prozess der MMI-Testautomatisierung beschrieben. Wichtige Ergebnisse des produktiven Einsatzes werden in Abschnitt 10.2 aufgeführt. In Abschnitt 10.3 werden zukünftige Aufgaben diskutiert. Abschnitt 10.4 gibt eine Zusammenfassung.

10.1 Aktueller Stand

Für den produktiven Einsatz ist eine Integration des Verifikationssystems in den bestehenden MMI-Testautomatisierungsprozess notwendig.

10.1.1 MMI-Designverifikationsprozess

Der Ablauf der für das MMI implementierten Designverifikation ist in Abbildung 10.1 veranschaulicht. Er entspricht im Wesentlichen dem in Kap. 6.3 konzipierten Designverifikationssystem. Bei der Netzgenerierung aus der Spezifikation, der Verifikation und der Bildverarbeitung handelt es sich jeweils um ein eigenständiges Tool. Alle drei Tools sind in Java implementiert. Die Inferenzberechnung erfolgt effizient mittels des Junction-Tree-Algorithmus (s. Kap. 5.2.1). Die Bildverarbeitung ist so in das Verifikationssystem integriert, dass die Ergebnisse direkt, ohne

Speicherung auf Festplatte, der Verifikation bereitgestellt werden (s. Schad 2011).

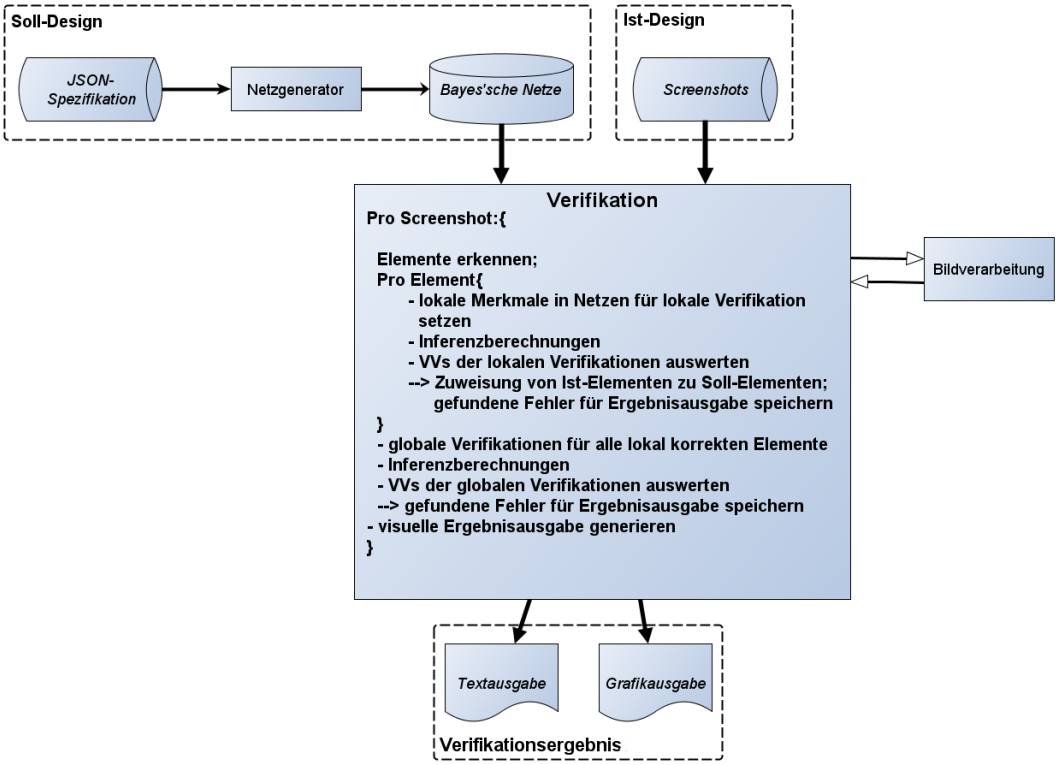


Abbildung 10.1: Implementierung des Designverifikationssystems für das MMI.

10.1.2 Ist-Design

Bei den herkömmlichen MMI-Tests entstehen Screenshots vom Ist-Design, die die Eingabe für die Designverifikation bilden. Für den Erwerb des Ist-Designs kommt die Bildverarbeitung von [Schad 2011] zum Einsatz. Die erforderlichen Informationen über Textfonts und Symbolelemente werden in Form von Font- und Bilddateien in einer Verzeichnisstruktur auf Festplatte bereitgestellt.

10.1.3 Soll-Design

Der Erwerb des Soll-Designs ist aufwendiger. Wie in Kap. 2.2.2.3 ausgeführt, ist das Design und Layout der MMI-GUI in PSD-Dateien und PDF-Dokumenten spezifiziert. Dieser Designentwurf ist nicht formal, kann deshalb nicht maschinell für Tests

verwendet werden und findet bislang auch keinen direkten Einsatz in der MMI-Testautomatisierung (vgl. Kap. 1.2). Adobe Photoshop bietet allerdings über eine API die Möglichkeit an, mittels der Skriptsprache JavaScript auf den Inhalt einer PSD-Datei zuzugreifen und diese auszulesen. Die gewonnen Informationen können in dem in Kap. 8 entwickelten JSON-Spezifikationsformat beschrieben werden. Da die JSON-Notation eine Untermenge der JavaScript-Syntax darstellt, ist eine Ausgabedatei im JSON-Format sehr einfach zu realisieren. Mittels JavaScript ist es auch möglich symbolische Inhalte aus PSD-Dateien als Bilder zu exportieren. Auf diese Weise kann die Menge der Referenzbilder ebenfalls automatisch bereitgestellt werden.
Die Erstellung der JSON-Spezifikation setzt eine klar definierte Struktur der PSD-Dateien mit Namenskonventionen voraus. In Zusammenarbeit mit den Designern konnten erste Anforderungen umgesetzt werden und ein JavaScript-Plugin existiert bereits.

10.1.4 Offline- und Online-Tests

Während die herkömmlichen MMI-Tests Online-Tests sind, wird die Designverifikation offline durchgeführt. Der Grund für diese Entscheidung ist, dass Online-MMI-Tests zeitkritisch sind. Alle Testalgorithmen, bspw. auch die Bildverarbeitungsmethoden, sollen möglichst durchgeführt worden sein, bevor im MMI-Menü weiternavigiert und ein neuer Screenshots erzeugt wird. Das Hinzufügen der Designverifikation würde diesen Zeitrahmen sprengen. Da die Designverifikation neben dem Soll-Design lediglich Screenshots des Ist-Designs benötigt und keine Informationen über den Systemzustand, stellt die Entscheidung für die Offline-Durchführung keinerlei Einschränkungen dar. Gefundene Fehler können anschließend in derselben zentralen Datenbank abgelegt werden.

10.2 Ergebnisse

Bis zum Abschluss der Dissertation konnte das Designverifikationssystem prototypisch für die Verifikation von Wizardmenüs erfolgreich eingesetzt werden. Die Bayes'schen Netze wurden dafür aus einer formalen Spezifikation generiert. Für die Spezifikation wurde die entwickelte JSON-Struktur verwendet. Die lokalen De-

signmerkmale wurden mittels JavaScript aus einer PSD-Datei extrahiert. Anschließend wurde die Spezifikation manuell um globale Merkmale ergänzt (vgl. JSON-Beispiel in Kap. 8.4). Das notwendige Zusatzwissen wurde ebenfalls manuell erstellt und entspricht dem aus Kap. 8.4.
Bezugnehmend auf das Wizardmenü-Beispiel aus Kap. 7.3, zeigen die Abbildungen 10.2 bis 10.4 ein paar Beispiele für die textuelle und visuelle Ausgabe von Verifikationsergebnissen. Um das Verifikationssystem zu testen, wurden die Screenshots mit Designfehlern präpariert. Alle Elemente, die korrektes lokales Design aufweisen, erhalten bei der visuellen Ausgabe eine grüne Markierung. In Abhängigkeit von den Gewichtungsstufen der Soll-Abweichungen variiert der Grünton von hell nach dunkel. Ein Element mit fehlerhaftem lokalen Design ist rot markiert. Fehlt ein Element, wird an der bestimmten Position eine orange Markierung gesetzt. Ein Element, das die Bildverarbeitung nicht klassifizieren kann, erhält eine hellblaue Markierung. Die textuelle Ausgabe liefert ggf. zusätzliche Informationen, z.B. bei fehlerhaftem Design die konkreten fehlerhaften Merkmale. In Abbildung 10.2 ist ein Verifikationsergebnis für ein vollständig korrektes Design zu sehen. Zusätzlich sind die einzelnen Bildverarbeitungs- und Verifikationssicherheiten angegeben.

Verification of Screenshot „MediaWizard1.png“:



	Recognition %	Correctness %
Tooltip text	**100**	**98,3**
Tooltip border	**99,2**	**98,2**
Tooltip arrow	**100**	**98,8**
Icon *skipbackward enabled*	**100**	**98,8**
Icon *fastbackward enabled*	**99,3**	**98,6**
Icon *play rolloverpressed*	**99,7**	**98,7**
Icon *fastforward enabled*	**99,3**	**98,6**
Icon *skipforward enabled*	**100**	**98,8**
Icon *info disabled*	**100**	**98,8**

Abbildung 10.2: Links: Verifizierter Screenshot-Ausschnitt eines Wizardmenüs ohne Designfehler. Rechts: Sicherheiten der Bildverarbeitung und der Verifikation der einzelnen Elemente.

Abb. 10.3 zeigt im obigen Beispiel eine dunkelgrüne Markierung für ein Element, dessen lokales Design stärker vom Soll abweicht. Die textuelle Beschreibung

gibt zusätzlich Auskunft darüber, dass die x-Position zu der Abweichung führt. Im unteren Beispiel fehlt ein obligatorisches Icon. Die visuelle Ausgabe für das in Abbildung 10.4 dargestellte vierte Beispiel ist dreigeteilt, um Überlagerungen der Markierungen und damit eine schlechte Lesbarkeit zu vermeiden. Das Icon *fast-backward* ist als fehlend markiert; im Screenshot ist es durch den Tooltip so stark verdeckt, dass die Bildverarbeitung es nicht klassifizieren kann. Aus diesem Grund wird der sichtbare Teil dieses Icons auch als unbekanntes Element bezeichnet. Der verschobene Tooltip wird als korrekt klassifiziert, da seine Position kein zu verifizierendes Merkmal ist (vgl. Kap. 7.3).

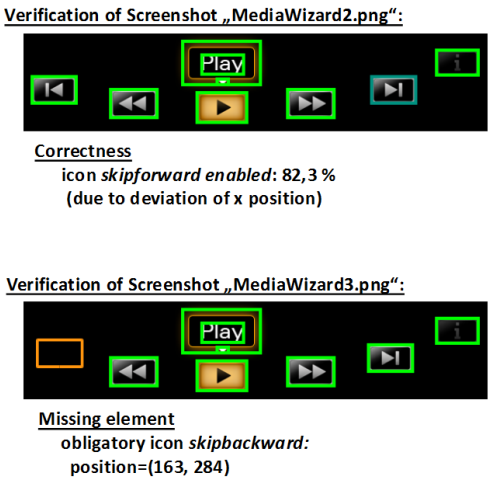


Abbildung 10.3: Oben: Das Icon *skipforward* weicht mit seiner x-Position etwas stärker vom Soll ab. Unten: Das obligatorische Icon *skipbackward* wurde von der Bildverarbeitung nicht gefunden und fehlt daher.

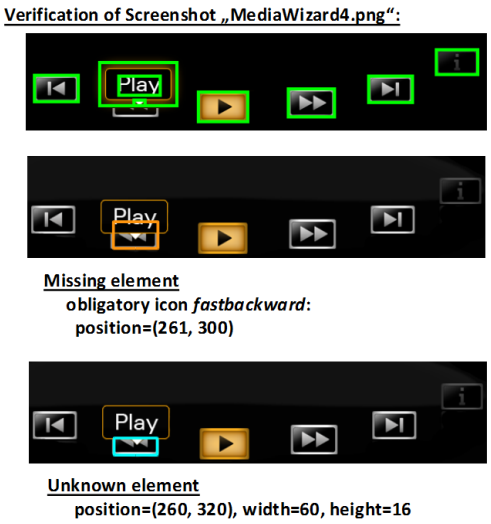


Abbildung 10.4: Oben: Verifizierter Screenshot-Ausschnitt eines Wizardmenüs, in dem nur die korrekten Elemente markiert sind. Mitte: Derselbe Screenshot-Ausschnitt mit Markierung der Position des fehlenden Icons. Unten: Derselbe Screenshot-Ausschnitt mit Markierung des von der Bildverarbeitung nicht klassifizierbaren Elements.

10.3 Bewertung und Ausblick

Das Designverifikationssystem ist bisher nur ein Prototyp. Für den umfangreichen produktiven Einsatz in der MMI-Testautomatisierung sind weitere Schritte notwendig, die nun diskutiert werden.

10.3.1 Vollständigkeit

Unklar ist bislang wie die, in den PSD-Dateien fehlenden, Designinformationen aus den PDF-Dokumenten verarbeitet werden sollen. Soweit sich an dem bestehenden Prozess des Designentwurfs nichts ändert, besteht keine Möglichkeit auf diese Informationen maschinell zuzugreifen. Ein Ansatz ist daher, ein Tool anzubieten, mit dem die in JSON formal beschriebenen Informationen aus den PSD-Dateien manuell ergänzt werden können. Das Lernen stellt dazu aktuell keine wirkliche Alternative dar; der Erwerb von Screenshots, die umfangreiches korrektes Design repräsentieren, ist bislang ebenfalls unklar.

10.3.2 Korrektheit

In den obigen Beispielen konnten alle Designfehler gefunden werden. U.a. bedeutet das, dass die Übergangswahrscheinlichkeiten geeignet für die verwendete Bildverarbeitung und die Qualität der Screenshots gewählt wurden.
Eine Ursache für falsche Designfehler kann eine fehlerhafte oder unzureichende Bildverarbeitung sein. Die Designverifikation weist allerdings auf Elemente hin, die nicht von der Bildverarbeitung klassifiziert werden können. In den obigen Beispielen handelt es sich dabei tatsächlich um Elemente, die fehlerhaft dargestellt sind. Liegt der Fehler hingegen bei der Bildverarbeitung, können die als *unbekannt* klassifizierten Elemente nicht weiter verifiziert werden. Die Verifikation wäre in diesem Fall unzureichend.
Das grafische Design des MMI ist in der Soll-Spezifikation pixelgenau angegeben. Damit ist genau genommen jede noch so minimale Abweichung ein Fehler, der von einem akzeptablen Verifikationsverfahren auch als solcher erkannt werden muss. Dieser Forderung widerspricht die Eigenschaft von Bildern, unsichere Informationen zu beinhalten, womit das Ist-Design nicht exakt ist. Das Verifikationsverfahren soll zum einen Abweichungen als Fehler feststellen, jedoch zum anderen Ungenauigkeiten, die durch unsichere Bildinformationen entstehen, tolerieren. Der Widerspruch dabei ist offensichtlich, da zu keinem Zeitpunkt in irgendeiner Weise festgestellt werden kann, ob sich eine erkannte Abweichung nur in unsicheren Bildinformationen begründet oder in einem echten Fehler. Um in den meisten Fällen akzeptable Verifikationsergebnisse zu liefern, ist eine adäquate Wahl der Parameter umso wichtiger.

10.3.3 Behandlung von Unsicherheit

In Kap. 2.2.2.3 wurde erwähnt, dass in PSD-Dateien Positionen in Subpixelgenauigkeit angegeben werden können. Eine Bildverarbeitung kann stattdessen in der Regel nur Positionen in Pixelgenauigkeit ermitteln. Damit kann es bereits hier zu leichten Abweichungen zwischen dem Soll- und dem Ist-Design kommen, was nicht als Designfehler gewertet werden darf. Auch diese Eigenschaft kann mit einer geeigneten Gewichtung der Soll-Abweichungen gehandhabt werden.

10.3.4 Anzeige der Verifikationsergebnisse

In Kap. 10.2 wurden die Verifikationsergebnisse sowohl textuell als auch visuell präsentiert. Abb. 10.5 zeigt exemplarisch ein mögliches GUI-Tool, um die Verifikationsergebnisse anzuzeigen und zu verwalten.

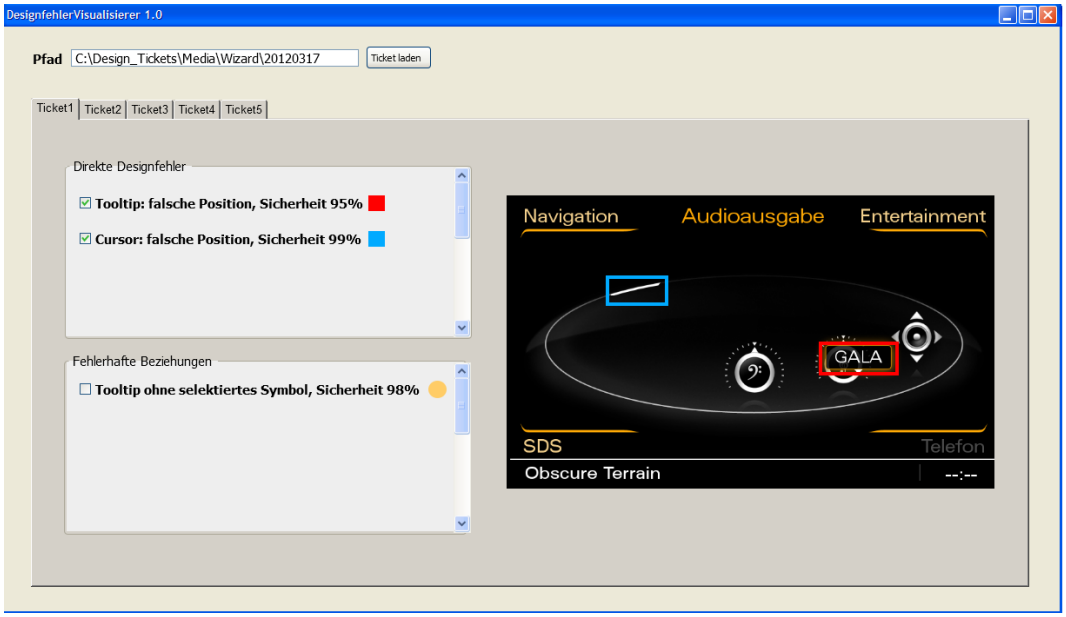


Abbildung 10.5: Ein mögliches GUI-Tool zur Verwaltung und Visualisierung von MMI-Designfehlern. (Die GUI-Skizze wurde mit der Software *Evolus Pencil* (EVOLUS 2010) erstellt.)

Das Tool orientiert sich an den in Kap. 6.3.4 formulierten Anforderungen für eine benutzerfreundliche Ergebnisausgabe. Anhand des Pfades kann ein Verzeichnis, das sogenannte Fehlertickets beinhaltet, geladen werden. Um dem Tool alle notwendigen Informationen geben zu können, muss ein Fehlerticket die gefundenen Designfehler und textuelle Erklärungen sowie einen Screenshot des jeweils verifizierten Designs umfassen. Auf Basis dieser Informationen können vom Tool die Designfehler im Screenshot markiert werden. Im Bereich „Direkte Designfehler“ werden alle Elemente mit fehlerhaften lokalen Merkmalen in textueller Form aufgelistet. Im Bereich „Fehlerhafte Beziehungen “ werden die gefundenen fehlerhaften globalen Merkmale aufgelistet. Über die Checkboxen kann jeder Fehler in der visuellen Anzeige eingeblendet oder ausgeblendet werden, um eine übersichtliche Darstellung zu ermöglichen (vgl. Kap. 6.3.4.2). Im abgebildeten Beispiel ist aus

diesem Grund die fehlerhafte Beziehung nicht visuell markiert.
Benutzer des GUI-Tools können neben den Testern auch die MMI-Entwickler sein, die die gefundenen Designfehler beheben sollen. Die konkrete Implementierung eines solchen Tools steht aktuell noch aus.

10.3.5 Online-Tests

Zukünftig sollte überlegt werden, wie die Designverifikation auch online realisiert werden könnte. Da ein fehlerhaftes Design Ursache für weitere Fehler sein kann, wäre eine vorangegangene Designverifikation vorteilhaft. Wenn bspw. ein Textbereich nicht an der vom Design vorgegebenen Position in der GUI vorkommt, wird die für die Texterkennung verwendete Maske nicht passen und es kommt zu fehlerhaften Erkennungen. Wenn jedoch durch die Designverifikation bereits bekannt ist, dass der Textbereich A anstatt an Position 1 an Position 2 zu finden ist, kann die Maske für die Texterkennung automatisch angepasst und die Texterkennung trotz fehlerhaftem Design korrekt durchgeführt werden.

10.3.6 Datenverwaltung

Es ist ein Verwaltungssystem notwendig, das die Bayes'schen Netze und die zu verifizierenden Screenshots bereitstellt. Um nicht die gesamte Menge an Bayes'schen Netzen bei der Verifikation jedes Screenshots ausführen zu müssen, ist es sinnvoll, die Screenshots nach Kontexten oder ggf. noch feiner zu gruppieren, was wiederum die Menge relevanter Bayes'scher Netze reduziert und die Verifikation beschleunigt. Des Weiteren fließen die identifizierten Fehler in den MMI-Fehlermanagementprozess ein, mit dem auch alle von anderen Tests gefundenen Fehler verwaltet werden.

10.4 Schlussfolgerung

Es konnte gezeigt werden, dass ein Praxis-Einsatz des entwickelten Verfahrens effizient realisierbar ist. Die Designverifikation wird aktuell als Offline-Test ausgeführt und die zu testenden Screenshots entstehen bereits bei der bisherigen Testautomatisierung als Ausgabedaten. Die gefundenen Designfehler eignen sich gut für die visuelle Darstellung und sollten aus diesem Grund für den Anwender auch visuell

ausgegeben werden. Eine gewisse Robustheit gegenüber den unsicheren Bildinformationen ist vorhanden und kann flexibel an die Beschaffenheit der Screenshots angepasst werden.
Ein umfangreicher produktiver Einsatz in der MMI-Testautomatisierung ist in Planung, stellt jedoch eine Herausforderung dar, weil hierfür Änderungen an dem seit Jahren existierenden und bewährten Designentwurf vorgenommen werden müssen. Erfolgsversprechend ist der Zugriff auf die Inhalte der PSD-Dateien und die Generierung einer Spezifikation in JSON mittels der Skriptsprache JavaScript. Die Struktur der PSD-Dateien muss klar definiert sein, damit die generierte JSON-Datei korrekt interpretierbar ist. Da eine klare Strukturierung auch Vorteile für die Arbeitsweise im Designentwurf mit sich bringt, ist die Forderung realisierbar. Es kann eine formale Spezifikation des GUI-Designs erzeugt werden, die nur einen geringfügigen Eingriff in den bisherigen Designentwurf darstellt. Da PSD-Dateien nicht das gesamte Soll-Design beschreiben können, muss eine Lösung gefunden werden, die noch fehlenden Informationen bereitzustellen. Daneben ist auch das erforderliche Verwaltungssystem in Planung. Erst wenn diese offenen Fragen geklärt sind, kann das Designverifikationssystem für ein vollständiges MMI durchgeführt werden und umfangreiche Ergebnisse zur Bewertung der Leistungsfähigkeit liefern.

Kapitel 11

Schlussbewertung

Zum Abschluss wird das entwickelte Designverifikationssystem im Ganzen betrachtet und bewertet. Abschnitt 11.1 fasst die Forschungsarbeit und ihre Ergebnisse zusammen. Abschnitt 11.2 geht auf die Erfüllung der gestellten Anforderungen und Ziele ein. In Abschnitt 11.3 wird in einem Ausblick kurz auf offene Fragestellung und zukünftige Arbeiten eingegangen. Das kurze Fazit in Abschnitt 11.4 fasst abschließend die wichtigsten Erkenntnisse zusammen.

11.1 Zusammenfassung und Ergebnisse

Die herkömmliche Testautomatisierung des Audi Infotainmentsystems MMI verifiziert explizit nur die Verhaltenslogik. So können nur indirekt Fehler gefunden werden, deren Ursache fehlerhaftes GUI-Design ist. Mit dem im Rahmen dieser Dissertation entwickelten Designverifikationssystem kann diese Lücke geschlossen werden.
Als wissensbasiertes System konzipiert, besteht das Designverifikationssystem aus den typischen Komponenten Wissensbasis, Wissenserwerb, Inferenzkomponente und Erklärungskomponente. Damit ist es sehr modular aufgebaut und unterstützt die unabhängige Erweiterbarkeit und Austauschbarkeit einzelner Komponenten. Die Wissensbasis ist zweigeteilt in das Soll- und das Ist-Design. Das Soll-Design ist für jedes MMI klar spezifiziert und umfasst Vorgaben über die lokalen Merkmale der einzelnen Designelemente (z.B. Farben und Positionen), die Semantik der optischen Darstellung (z.B. „ein inaktives Icon ist grau“) und globale Abhängigkeiten (z.B. Abstände und Koexistenzen). Das Ist-Design repräsentiert in Form von Screenshots

den aktuellen Displayinhalt und wird mit Bildverarbeitungsmethoden erworben. Je nach Art des Renderings und der Aufnahmequalität, weisen die Screenshots Störungen auf. Die Bildinformationen sind daher mit Unsicherheiten behaftet und eine Abweichung vom Soll muss nicht zwangsläufig ein Designfehler bedeuten. Den Systemkern, die Inferenzkomponente, bildet ein Bayes'sches Netz. Das Bayes'sche Netz definiert das Soll-Design und ermöglicht die Eingabe und Verarbeitung von unsicheren Ist-Designinformationen. Die Struktur des Bayes'schen Netzes repräsentiert dabei sehr intuitiv die Abhängigkeiten zwischen Designmerkmalen und -elementen. Auch die Eingaben sind explizit im Bayes'schen Netz dargestellt sowie auch die Ausgaben direkt nachvollzogen werden können. Über die Wahl der Zustandsmengen und der Parameter kann die Behandlung von Soll-Abweichungen flexibel konfiguriert werden. Die Verwendung eines Bayes'schen Netzes wurde gegenüber anderen Ansätzen, insbesondere der Fuzzy-Logik, kritisch hinterfragt. Es konnte festgestellt werden, dass der Formalismus' Bayes'scher Netze konsistenter und weniger interpretativ ist als viele andere Ansätze zur Behandlung von Unsicherheiten. Das grafische Modell macht ein Bayes'sches Netz zudem intuitiv verständlich und nachvollziehbar. Im produktiven Einsatz konnte gezeigt werden, dass ein Bayes'sches Netz in der Lage ist, das GUI-Design eines MMI vollständig zu verifizieren und geeignet auf Soll-Abweichungen einzugehen. Ebenfalls entscheidend für eine hohe Benutzerakzeptanz ist die Art der Ergebnisausgabe und die Erklärbarkeit der Schlussfolgerungen. Das entwickelte Bayes'sches Netz liefert mit seiner Struktur geeignete Ansatzpunkte für eine Erklärungskomponente. So kann bei einem Designfehler auf die Fehlerursache, d.h. das fehlerhafte Merkmal, rückgeschlossen werden. Daneben bietet sich die Visualisierung der Designfehler an, die exemplarisch vorgestellt wurde.

Wie auch in [Wiegerinck, Kappen und Burgers 2010] festgestellt[1], handelt es sich beim Bayes'schen Netz nur um einen Teil des gesamten Softwaresystems. Ursprünglich entstand das Bayes'sche Netz aus Überlegungen heraus und wurde manuell für die Verifikation bestimmter Bereiche der MMI-GUI erstellt. Für die Verifikation ei-

[1] „Finally, we would like to stress that the Bayesian network technology is only one side of the model. The other side is the domain knowledge, which is maybe even more important for the model. Therefore Bayesian network modeling always requires a close collaboration with domain experts. And even then, the model is of course only one of many ingredients of an application, such as user-interface, data-management, user-acceptance etc. which are all essential to make the application a success." (Wiegerinck, Kappen und Burgers 2010, S. 31/32)

nes umfangreichen GUI-Designs, wie das des MMI, ist eine manuelle Erstellung jedoch nicht praktikabel. Zur automatischen Verarbeitung muss das Soll-Design in einem formalen, maschinenlesbaren Format vorliegen. Für das MMI ist der Designentwurf in Adobe Photoshop-Dateien und PDF-Dokumenten spezifiziert; damit ist das Soll-Design nicht formal und direkt zugänglich. Das Problem ist daher nicht die eigentliche Designverifikation, sondern die automatisierte Generierung des Bayes'schen Netzes aus dem Soll-Design. Aus diesem Grund war ein weiterer Schwerpunkt dieser Arbeit, Lösungen für die automatische Generierung des Bayes'schen Netzes zu finden. Dafür wurde zum einen eine formale Spezifikation entwickelt und zum anderen untersucht, wie durch maschinelles Lernen Designinformationen aus Screenshots gewonnen werden können. Als Spezifikationssprache wurde das Datenaustauschformat JSON verwendet. Die konzipierte JSON-Struktur ermöglicht eine hierarchische Beschreibung von GUI-Design in einer kompakten und intuitiven Weise. Es wurde bewusst keine der bestehenden, meist XML-basierten GUI-Spezifikationssprachen verwendet, da diese auf die Beschreibung von Verhaltenslogik fokussiert sind und die Definition umfangreicher Designinformationen nicht vorsehen. Im direkten Vergleich mit XML erwies sich die Spezifikation von GUI-Design in JSON zudem als wesentlich ressourcensparender und damit effizienter. Da Adobe Photoshop eine Schnittstelle zu JavaScript anbietet und JSON eine Untermenge von JavaScript ist, ist die Erzeugung von JSON-Dateien aus dem PSD-Designentwurf zudem sehr einfach realisierbar. Das JavaScript-Plugin für Photoshop und der Parser für die Generierung des Bayes'schen Netzes sind bereits implementiert. Ein Problem bleibt die Unvollständigkeit des Photoshop-Designentwurfs, indem vor allem keine globalen Merkmale spezifiziert werden können. Das datenbasierte Lernen soll Abhilfe schaffen und einem Bayes'schen Netz fehlende Designinformationen hinzufügen. Die Ausgangsdaten sind hier Screenshots, die wenigstens das korrekte Design einer GUI repräsentieren. Für ausgewählte Bereiche der MMI-GUI konnten Bayes'sche Netze gelernt werden, die das gewünschte Soll-Design abbilden. In der Realität stehen jedoch meist keine Lerndaten zur Verfügung und der Idealfall, dass sowohl Screenshots von korrektem als auch von fehlerhaften Design gegeben sind, ist nicht erreichbar. Damit löst auch das Lernen nicht das eigentliche Problem der Unvollständigkeit der Wissensbasis. Zudem sind gelernte Bayes'sche Netze in der Regel sehr kompakt, wodurch ihre Erklärbarkeit sinkt. Erweiterungen der Lernverfahren oder nachträgliche Anpassungen der

gelernten Netze sind dann notwendig. Auch die Performanz ist ein beschränkender Faktor. Während zur Repräsentation einzelner lokaler oder globaler Merkmale, definiert durch einige 100 Datensätze, noch in wenigen Millisekunden ein Netz gelernt werden konnte, war für das Lernen eines umfangreicheren GUI-Bereichs eine Rechenzeit von mehreren Minuten bis einigen Stunden erforderlich. Das Lernen Bayes'scher Netze ist mit seiner Vielzahl an Suchverfahren und Parameterschätzern zudem ein sehr komplexes Thema. Die geeignete Kombination dieser Freiheitsgrade lässt sich nicht pauschalisieren. Damit ist sowohl die Generierung aus einer formalen Spezifikation als auch das Lernen bislang nicht in vollem Umfang für das MMI-GUI-Design zu bewerkstelligen und bis zum Ende der Dissertation konnte die Designverifikationssystem nur prototypisch getestet werden. Wie der folgende Abschnitt zeigt, konnten die geforderten Ziele dennoch erfüllt werden.

11.2 Erreichte Ziele

Bezugnehmend auf die in Kap. 6.1 formulierten Anforderungen, wird das entwickelte Designverifikationssystem abschließend bewertet.

11.2.1 Vollständigkeit

Ob die Designverifikation alle Designfehler findet, ist zuallererst abhängig von der Vollständigkeit des gegebenen Soll-Designs. Die entwickelte JSON-Spezifikation ist in der Lage, die hierarchische Struktur einer GUI und ihre Designinformationen abzubilden. Im Rahmen der Dissertation konnte exemplarisch gezeigt werden, das eine vollständige Spezifikation des MMI-GUI-Designs möglich ist. In der Praxis ist es für das MMI jedoch aktuell nicht realisierbar, alle Designinformationen als formale Spezifikation bereitzustellen. Die Möglichkeit, das Bayes'sche Netz auch aus Daten zu lernen, kann fehlende Informationen hinzufügen. Die dafür notwendigen Screenshots, die korrektes Design repräsentieren, sind mitunter allerdings ebenfalls schwer zu erwerben.
Verifiziert werden kann zudem nur, was im Ist-Design erkannt wird. Hier bilden die existierenden Bildverarbeitungsmethoden eine Grenze. Ein Beispiel ist die Verifikation des Textfonts. Aus Sicht der Inferenzkomponente ist die Verifikation des Fonts und seiner Einzelmerkmale unproblematisch (vgl. Kap. 7.3.3.1). Auch möglich wä-

re bei Eingabe des erkannten Fonts die Behandlung von diesem als Gesamtmerkmal. Eine Bildverarbeitung, die verschiedene Fonts oder deren Einzelmerkmale erkennt, stand jedoch im Rahmen dieser Arbeit nicht zur Verfügung. Daher liegen für diese Verifikation keine praktischen Ergebnisse vor.

11.2.2 Unsichere Information

Unsichere Bildinformationen können gehandhabt werden, indem mit Schwellwerten entschieden wird, ob bspw. die Erkennersicherheit eines Symbols ausreichend hoch ist. Bzgl. einzelner Werte ist dies eine geeignete Methode und die gängige Vorgehensweise in der bestehenden MMI-Testautomatisierung. Bei der Designverifikation müssen jedoch mehrere unsichere Informationen miteinander kombiniert werden. In diesem Fall ist es besser, Unsicherheiten zu beachten und bis zum Gesamtergebnis mitzuführen.
Für die Designverifikation wurde diese Anforderung mit einem Bayes'schen Netz als Inferenzkomponente realisiert. Mit dem probabilistischen Schließen ermöglicht es eine geeignete Verarbeitung und Kombination von Unsicherheiten, indem diese als Wahrscheinlichkeiten interpretiert werden. Durch die Wahl der Zustandsmengen und der Parameter kann die Gewichtung der Soll-Abweichungen sehr flexibel definiert werden. Für stark gestörte Kameraaufnahmen kann bspw. eine tolerantere Gewichtung eingesetzt werden, als für sehr störungsarme Framegrabber-Screenshots. Daneben kann auch für jedes Merkmal eine individuelle Gewichtung eingestellt werden.
Neben Bayes'schen Netzen existieren einige weitere Formalismen für die Verarbeitung von unsicheren Informationen. Die Verwendung eines Bayes'schen Netzes wurde daher ausführlich diskutiert und insbesondere mit der Fuzzy-Logik verglichen. Dabei wurden auch die zwei Arten der im Rahmen der Designverifikation anfallenden Unsicherheiten, die harte und die weiche Evidenz, spezifiziert. Die unsicheren Informationen sind nicht unscharf, weswegen ein Ansatz wie die Fuzzy-Logik nicht in Frage kommt. Der Formalismus' Bayes'scher Netze ist zudem konsistenter, intuitiver und weniger interpretativ als viele andere Ansätze. Jedoch sind auch hier einige Freiheitsgrade und Definitionsmöglichkeiten gegeben.Wie letztendlich Unsicherheiten als Wahrscheinlichkeiten aufgefasst werden lässt sich nicht einheitlich festlegen. Die vorgestellte Netzstruktur und die Parameter sind ein Vorschlag für ein Bayes'sches Netz zur Designverifikation. Beides entstand aus Über-

legungen heraus und wurde anhand von Beispielen auf Tauglichkeit überprüft. D.h. das entwickelte Bayes'sche Netz und seine Parameter basiert auf Expertenwissen und nicht auf realen, statistischen Daten. Wie auch in der Literatur immer wieder angemerkt wird (z.B. Beierle und Kern-Isberner 2006, Kap. 12, S. 365), liegen statistische Daten meist nicht vor. Auch für das MMI-GUI-Design ist der Idealfall, eine Menge von Screenshots, die sowohl korrektes als auch fehlerhaftes Design repräsentieren, nicht gegeben. Wie die Beispiele in dieser Arbeit zeigen, führen aber auch die „educated guesses" (vgl. Abschnitt 7.5 und Wiegerinck, Kappen und Burgers 2010, S. 6) zu geeigneten Ergebnissen bei der Designverifikation.

11.2.3 Automatisierung

Aufgrund der Komplexität des MMI-GUI-Designs ist eine manuelle Erstellung des Bayes'schen Netzes nicht praktikabel. Für die automatische Generierung wurden zwei Möglichkeiten, die Generierung aus einer formalen Spezifikation und durch datenbasiertes Lernen, untersucht. Wie die Ergebnisse aus dem produktiven Einsatz zeigen, ist JSON ein flexibles Austauschformat um Designinformationen automatisch aus PSD-Dateien zu extrahieren und daraus ein Bayes'sches Netz zu generieren. Durch das datenbasierte Lernen ist ebenfalls eine automatische Generierung eines Bayes'schen Netzes möglich.

11.2.4 Maschinenlesbarkeit

Das in Form von Screenshots gegebene Ist- und Soll-Design wird mittels Bildverarbeitungsmethoden ausgewertet und steht dann maschinenlesbar zur Verfügung. Die entwickelte formale Soll-Spezifikation in JSON ist ebenfalls maschinenlesbar. Zur Speicherung generierter Bayes'scher Netze wird das formale BIFXML-Format verwendet.

11.2.5 Effizienz

Explizit besteht keine Echtzeitforderung für das Designverifikationssystem. Die Designverifikation wird aktuell als Offline-Test durchgeführt und ist damit nicht an die Geschwindigkeit der übrigen MMI-Tests gebunden. In Anbetracht der stetig wachsenden Komplexität des MMI ist dennoch eine möglichst effiziente Ve-

rifikation wünschenswert. Die Effizienz wird dabei maßgeblich durch die Art der Wissenbasis und den Wissenserwerb sowie die Inferenzkomponente bestimmt. Das Bayes'sche Netz ermöglicht mit seiner Baumstruktur effiziente Inferenzberechnungen. Die JSON-Spezifikation ist so konzipiert, dass sie einen effizienten Zugriff auf die Informationen ermöglicht. Das Lernen ist stattdessen zeitintensiver in Abhängigkeit von der Datenmenge und bedarf weiterer Optimierungen, falls es umfangreicher zum Einsatz kommen soll.

11.2.6 Erweiterbarkeit

Das Designverifikationssystem basiert auf der für wissensbasierte Systeme vorgeschlagenen Architektur und weist damit bereits eine geeignete Strukturierung in einzelnen Komponenten auf. Die Komponenten können unabhängig voneinander ausgetauscht und erweitert werden, solange die geforderten Schnittstellen erhalten bleiben.
Die im Rahmen der Dissertation konzipierten JSON-Schemata für die formale Spezifikation von Soll-Design und dem Zusatzwissen stellen Beispiele dar. Zahlreiche Erweiterungen (vgl. u.a. Kap. 8.5.2) sind möglich. Über die Mächtigkeit auch zukünftiges Design darstellen zu können, lässt sich nur mutmaßen. Die Flexibilität von JSON deutet aber darauf hin, dass auch die Definition von zukünftigen GUI-Designs unproblematisch sein wird. Letztendlich können alle Informationen in Form von Objekten, Arrays und Referenzen formuliert werden. Auch das Bayes'sche Netz ist basierend auf den im Rahmen der Dissertation gewonnen Kenntnisse mächtig genug für die Repräsentation von GUI-Design und kann um neue Merkmale erweitert werden (s. Kap. 7.4.9).

11.2.7 Universalität

Im Rahmen der Dissertation wurde das entwickelte System für die Verifikation der GUI eines Audi Infotainmentsystems evaluiert, was aber keine Einschränkung für die Anwendung auf verschiedenste GUIs darstellt. Durch die Erweiterbarkeit des Systems ist auch der universelle Einsatz gewährleistet. Liegt das Designwissen in einer geeigneten formalen Form vor, kann stets ein Bayes'sches Netz generiert werden, das dieses Wissen abbildet. Während die für das MMI implementierte Generierung der JSON-Spezifikation einen Designentwurf im PSD-Format erfordert, kann

das Lernen universell auf Screenshots angewendet werden.

11.2.8 Korrektheit

Das entwickelte Designverifikationssystem ist neuartig und in der Literatur findet sich nichts vergleichbares. Damit ist auch kein Vergleich zu anderen Systemen zur Designverifikation möglich. Die Validierung des Verfahrens erfolgte basierend auf einer Menge von korrekten und fehlerhaften Screenshots des MMI sowie auch direkt im produktiven Einsatz. Wie die Ergebnisse in 10.2 zeigen, wird ein sehr großer Teil tatsächlicher Designfehler identifiziert und im Gegensatz dazu weist das Verfahren Robustheit gegenüber Fehlurteilen, die durch unsichere Bildinformationen verursacht werden, auf. Diese Flexibilität wird durch das Bayes'sche Netz erreicht. Das Designverifikationssystem ist dadurch allerdings auch in der Lage sehr leichte Abweichungen vom Soll zu identifizieren, die bei der Betrachtung nicht augenscheinlich sind. Hier liegt es an den Forderungen des Anwendungsbereichs, ob dies notwendig ist und als Fehler gewertet werden sollte. Auch kann es durch eine ungeeignete Parametrisierung sowie Kombination der Wahrscheinlichkeiten schnell zu falsch-positiven oder falsch-negativen Ergebnissen kommen (vgl. Kap. 7.4.8). Evtl. sind mehrere Testdurchläufe notwendig, bis das Bayes'sche Netz für den Anwendungsfall geeignet konfiguriert ist.
Daneben gibt es Grenzen in der Erklärbarkeit der Fehlerursache. Wenn bspw. die Bildverarbeitung ein Element nicht klassifizieren kann, ist dieses unbekannt und steht für eine weitere Verifikation nicht zur Verfügung. Auch muss ein erkannter Designfehler kein echter Fehler sein, sondern kann auch durch die Verwendung eines noch nicht aktualisierten Soll-Designs begründet sein. Hier ist es jedoch ausreichend, dass Fehler überhaupt entdeckt werden. Die MMI-Testautomatisierung sieht es ohnehin vor, dass jeder Fehler vor der Freigabe zuerst von einem Anwender begutachtet wird.

11.3 Ausblick

Das entwickelte Designverifikationssystem stellt einen neuartigen und sehr komplexen Anwendungsbereich für ein Bayes'sches Netz dar. Während der Entwicklung sind neue Erkenntnisse und Fragestellungen entstanden. In Kap. 10.3 wurden be-

reits einige zukünftige Aufgaben angesprochen, um den produktiven Einsatz in der MMI-Testautomatisierung zu ermöglichen. Im Folgenden sind weitere Ideen kurz zusammengefasst.

11.3.1 Aktualisierung bestehender Bayes'scher Netze und Wissensbasen

Meist weist eine neue MMI-Version nur geringfügige Änderungen im Vergleich zur Vorgängerversion auf. Zwecks enormer Zeitersparnis ist es in diesen Fällen sinnvoll nicht das komplette Bayes'sche Netz neu zu generieren, sondern nur die veränderten Teile zu aktualisieren. Dasselbe gilt für die Erstellung einer formalen Designspezifikation oder die Bereitstellung von Screenshots als Lerndaten.
In Kap. 9.4.4.2 wurde kurz erläutert, wie ein Bayes'sches Netz aus formal generierten und gelernten Teilen kombiniert werden kann. Auf ähnliche Weise können bestehenden Netzen neue Teile hinzugefügt oder Parameter verändert werden. Die JSON-Spezifikation bietet durch die hierarchische Struktur ebenfalls eine einfache Erweiterbarkeit der Soll-Design-Beschreibung an. Da für das Bayes'sche Netz und die JSON-Spezifikation Konventionen definiert sind, kann relativ einfach eine Software entwickelt werden, um Änderungen automatisiert einzufügen. Wenn die als Lerndaten verwendeten Screenshots geeignet verwaltet sind, können diesen ebenfalls neue Informationen hinzugefügt werden.

11.3.2 Unterstützung zukünftiger MMI-GUIs

Bereits heute ist der Funktionsumfang des MMI sehr umfangreich, was sich in der GUI direkt widerspiegelt. Zukünftig ist ein weiterer Anstieg der Funktionalität zu erwarten. 3D-Elemente, Animationen (insbesondere animierte Screenübergänge) und transparente Bereiche werden hier Einzug erhalten. Neben dem MMI-Display, wird zudem immer mehr das Kombinstrument als visuelle Anzeige eingesetzt. Der Inhalt des Kombi-Displays kann aktuell in der Regel nur mittels einer Kamera als Screenshot erworben werden, sodass die Behandlung von unsicheren Bildinformationen eine wichtige Rolle spielt.
Mit steigender Komplexität gewinnt auch die Verifikation des GUI-Designs an Bedeutung. Das Designverifikationssystem muss auch für die zukünftigen MMI-GUIs einsetzbar sein. Das betrifft nicht nur das Bayes'sche Netz als Inferenzkomponente,

sondern auch das JSON-Spezifikationsformat. Auch die Bildverarbeitungsmethoden müssen zukünftig nicht nur für die Designverifikation, sondern für die gesamte MMI-Testautomatisierung deutlich mehr leisten. Da 3D-Grafiken und Animationen nicht mehr in Adobe Photoshop entwickelt werden können, wird sich auch die Art des Designentwurfs ändern. Erneut wird dann ein Tool für die Erstellung der JSON-Spezifikation implementiert werden müssen.
Zum jetzigen Zeitpunkt kann keine Aussage getroffen werden, inwieweit das Designverifikationssystem mit den zukünftigen Veränderungen mithält. Es lässt sich lediglich erkennen, dass für das aktuelle MMI sowohl das Bayes'sche Netz als auch die JSON-Spezifikation mächtig genug sind, alle erforderlichen Designinformationen abzubilden. Die JSON-Spezifikation scheint flexibel und zugleich einfach genug, um weitere Informationen definieren zu können. Mit ihren zahlreiche Erweiterungen und verwandten Formalismen zeigen Bayes'sche Netze ebenfalls Zukunftsperspektive auf.

11.3.3 Lernen

Wie die bereits im Rahmen dieser Arbeit durchgeführten Experimente zeigen, ist eine Erweiterung des aus der formalen Designspezifikation generierten Bayes'schen Netzes durch anschließendes Lernen möglich. Die Problematik der spärlichen Lerndaten, die im Normalfall nur aus korrekten Screenshots gewonnen werden, konnte mit der Wahl der Bayes'schen Schätzung als geeigneter Parameterschätzer bereits gelöst werden. Daneben gibt es noch weitere Methoden für das Lernen mit spärlichen und unvollständigen Daten (s. Kap. 5.3.2.6). Auch für den effizienten Umgang mit sehr großen Datenmengen gibt es einige Ansätze (s. Kap. 5.3.2.7). Da die Erklärbarkeit der Bayes'schen Netze durch Lernen normalerweise abnimmt, gibt es auch hierfür speziell modifizierte Lernverfahren (s. Kap. 5.6.3). Lernverfahren, die mit unvollständigen Daten gute Ergebnisse erzielen und Zusatzwissen miteinbeziehen, können für die Designverifikation noch weiter erforscht werden. Solange allerdings keine geeigneten Lerndaten für das MMI-GUI-Design vorliegen, macht die Weiterführung des Lernansatzes aus produktiver Sicht keinen Sinn.

11.3.4 Globale Designtests

Bislang beschreibt ein Bayes'sches Netz das Design eines klar definierten Screenbereichs. Zusammenhänge zwischen grafischen Elementen, die über die Grenzen eines Screenbereichs hinausgehen, wurden in den Beispielen nicht verifiziert. Die Entscheidung, dass ein Netz nur einen Screenbereich abbildet wurde lediglich einfachheitshalber getroffen. Nach dem gleichen Konzept kann ein Bayes'sches Netz auch dazu verwendet werden das gesamte grafische Design eines Screenshots zu repräsentieren und zu verifizieren. Bspw. kann dann verifiziert wenn, dass sich von den in einer View befindlichen vier Softkeys jeweils nur genau ein Softkey im Zustand *selektiert* befindet.

11.3.5 Dynamische Designtests

Bisher kann das Designverifikationsverfahren ähnlich wie eine Markov-Kette betrachtet werden, bei dem nur gegenwärtige Merkmale das Verifikationsergebnis beeinflussen. D.h., es wird stets nur ein Screenshot verifiziert, vorhergehende oder zukünftige Screenshots sind nicht bekannt. Mit den in Kap. 5.7.2 vorgestellten dynamischen Bayes'schen Netzen könnte auch Wissen über das Aussehen des GUI-Designs vergangener und zukünftiger Zeitpunkte einbezogen werden. Auf diese Weise könnte z.B. der Ablauf eines animierten Screenübergangs (z.B. neuer Screen fährt von links ein) verifiziert werden. Darüber hinaus kann Wissen aus der Vergangenheit auch die Wahrscheinlichkeiten für bestimmte Merkmale oder Merkmalskombinationen beeinflussen.

11.3.6 Einsatzgebiet MMI-Testautomatisierung

Mit Implementierung des in dieser Arbeit beschriebenen Designverifikationssystems können auf Basis der ab sofort vorhandenen formalen Soll-Designspezifikation und der Bildverarbeitung Lösungen für weitere Anforderungen in der MMI-Testautomatisierung bereitgestellt werden. Mit dem Wissen über das Soll-Design lassen sich die für die Text- und Symbolerkennung notwendigen Maskenkoordinaten automatisch generieren. Eine Designänderung und damit notwendige Anpassung der Masken stellt dann keinen manuellen Aufwand mehr dar. Durch Generierung auf Basis des Soll-Designs können mehr und exaktere Masken verwendet werden.

Dadurch lassen sich Text- und Symbolbereiche genauer von anderen Bereichen abgrenzen und enthalten weniger zusätzliche Elemente, die zu fehlerhaften Erkennungen führen können. Das Vorhandensein des Soll-Designwissens in maschinenlesbarer Form führt damit zu einer Verbesserung der Erkennung, was wiederum zu aussagekräftigeren Tests führt. Darüber hinaus können auch mit der von [Schad 2011] entwickelten Bildverarbeitung Text- und Symbolbereiche genau bestimmt werden. Auch auf dieser Grundlage ist eine automatisierte Erstellung von Maskenkoordinaten möglich.
Wie in den vorangegangen Kapitel gezeigt worden ist, ist das auf einem Bayes'schen Netz basierende Designverifikationsverfahren nicht nur auf die bloße Verifikation des Ist-Designs beschränkt. Mittels des Wissens über das Soll-Design können außerdem Zustände von grafischen Elementen identifiziert werden, die wiederum als Eingabe für weitere Tests dienen.

11.3.7 Korrektheit

Aufgrund der fehlenden vollständigen formalen Designspezifikation und geeigneter Lerndaten konnte bis zum Abschluss der Dissertation das Designverifikationssystem nur anhand einiger Beispiele auf Tauglichkeit getestet werden. Wenn ein umfangreicher produktiver Einsatz möglich ist, muss auch die Korrektheitsprüfung in einem umfangreicheren Maße durchgeführt werden. Die Ergebnisse werden dann zu weiteren Erkenntnissen führen. U.a. kann eine automatische Konfiguration des Bayes'schen Netzes durch adaptive Anpassung während der Laufzeit erforderlich sein, um für den jeweiligen Anwendungsbereich eine geeignete Gewichtung der Soll-Abweichungen zu erhalten.

11.3.8 Optimierung

Um die bisher verwendeten „educated guesses“ (vgl. Abschnitt 7.5 Wiegerinck, Kappen und Burgers 2010, S. 6) durch reale Parameter zu ersetzen, müssen die Lerndaten sowohl korrektes als auch fehlerhaftes GUI-Design repräsentieren. Durch eine breite Sammlung solcher Screenshots für aktuelle MMI-Systeme, könnte dies zumindest teilweise ermöglicht werden. An dieser Stelle wäre auch interessant, wie das gelernte Netz reagiert, wenn Designfehler eingegeben werden, die nicht in den Lerndaten enthalten sind. Auch die Aktualisierung bereits gelernter Bayes'scher

Netze basierend auf neuen Informationen spielt dann eine Rolle (vgl. Abschnitt 11.3.1).
Obgleich die Designverifikation in der MMI-Testautomatisierung ohne Einschränkung als Offline-Test durchgeführt werden kann (vgl. 11.2.5), ist in Anbetracht der Komplexität des MMI-GUI-Designs eine Optimierung der Rechenzeit stets sinnvoll. Darüber hinaus würde eine online Durchführung der Designverifikation weitere Vorteile bringen (s. Kap. 10.3.5). Das konzipierte Bayes'sche Netz ist bereits durch seine Baumstruktur und die Verwendung vermittelnder Variablen recht effizient. Falls, insbesondere durch Lernen, weniger effiziente Bayes'sche Netze entstehen, sollte eine nachfolgende automatisierte Verbesserung durchgeführt werden. Dafür kann z.B., wie in [Borth und Hasseln 2002] vorgestellt, ein regelbasiertes System eingesetzt werden. Evtl. kann auch der Einsatz von Faktorgraphen anstelle von Bayes'schen Netzen zu effizienteren Verfahren führen.

11.4 Fazit

Im Rahmen der Dissertation konnte ein mit einem Bayes'schen Netz realisiertes Verifikationsverfahren entwickelt werden, das in der Lage ist, designspezifische Fehler in GUIs zu identifizieren. Um den Einsatz in der MMI-Testautomatisierung zu gewährleisten, wurde daneben auch eine vollständige Systemarchitektur konzipiert und gezeigt, wie das erforderliche Wissen über das Soll- und das Ist-Design in geeigneter Form erworben werden kann. Die Produktivität der Testautomatisierung ist eng verknüpft mit dem MMI-Entwicklungsprozess und wird idealerweise von diesem unterstützt. Da dies aktuell bzgl. einer formalen Designspezifikation und geeigneter Lerndaten nicht der Fall ist, konnte die Designverifikation zum Abschluss der Dissertation nicht im vollem Umfang produktiv eingesetzt werden. Das Designverifikationssystem ist aktuell im Prototypenstatus und konnte daher nur anhand von Beispielen auf Tauglichkeit geprüft werden. Das Fehlen einer formalen Spezifikation ist ein generelles Problem in der Softwareentwicklung. Um die Designverifikation in der MMI-Testautomatisierung einsetzen zu können, muss der Bruch zwischen informeller und formaler Spezifikation im Designentwurf beseitigt werden. Die Entwicklung der einfachen JSON-Spezifikation, die mächtig genug ist das MMI-GUI-Design abzubilden und des JavaScript-Plugins stellen dafür wichtige Schritte dar. Auch das Lernen Bayes'scher Netze kann hier zielführend sein,

wenn der Lernbereich geeignet einschränkt wird und Lerndaten existieren. Da das Lernen Bayes'scher Netze allerdings sehr viele Freiheitsgrade und damit Fehlerquellen besitzt, ist die Generierung aus einer formalen Spezifikation vorzuziehen. Auch in der Literatur wird die Generierung von Bayes'schen Netzen aus einer formalen Spezifikation gegenüber dem Lernen favorisiert. Meist wird die Generierung dabei zudem mit manuell hinzugefügten Expertenwissen gesteuert. In [Wiegerinck, Kappen und Burgers 2010] wurden drei real eingesetzte sehr umfangreiche Netze vorgestellt, um Informationen in den Bereichen Medizin, Petrophysik und Forensik zu verarbeiten. In [Borth und Hasseln 2002] wurde einer der wenigen Versuche unternommen ein Bayes'sches Netz für die Diagnose von technischen Systemen einzusetzen und automatisch aus einer System-Spezifikation zu generieren. Mit dem hier vorgestellten Verifikationsverfahren wurde erneut ein Bayes'sches Netz im Bereich der Testautomatisierung eingesetzt. Die Prototypenergebnisse der Dissertation zeigen die erfolgreiche Anwendung und das Potential des Designverifikationssystems für die MMI-Testautomatisierung. Die letztendliche Umsetzung in die Praxis bleibt den Anwendern überlassen und wird nicht ad hoc stattfinden. Am produktiven Einsatz für die vollständige MMI-Designverifikation wird aktuell gearbeitet. Unter anderem muss in einem der nächsten Schritte eine vollständige formale Designspezifikation erreicht werden. Ebenfalls muss eine Verwaltungskomponente eingerichtet werden, um die notwendigen Eingabedaten zentral bereitzustellen und die Verifikationsergebnisse geeignet zu speichern. Die Dissertation zeigt an dieser Stelle auch die Schwierigkeit wissenschaftliche Konzepte in der Praxis tatsächlich zu realisieren und die bewährten Prozesse anzupassen. Wenn das erforderliche Framework nicht geschaffen wird, bleibt auch das Designverifikationssystem und die JSON-Spezifikation nur ein Prototyp, wie viele Ansätze und Spezifikationssprachen zuvor.

Die Errungenschaften des Designverifikationssystems beschränken sich nicht nur auf die Designverifikation. Die formale Soll-Designspezifikation und die Bildverarbeitung können auch für weitere Zwecke, z.B. die für viele MMI-Tests notwendige Erstellung von Masken eingesetzt werden. Auch die Erkennung von Elementzuständen kann Eingabe für weitere MMI-Tests sein. Damit bietet das Designverifikationssystem letztendlich ein breites Spektrum neuer Möglichkeiten für die MMI-Testautomatisierung, die weit über das ursprüngliche Ziel, die Designverifikation, hinausgeht.

Anhang A

Symbol- und Abkürzungsverzeichnis

A.1 Konventionen

große fette Buchstaben	Mengen (z.B. $\mathbf{A}$)
große normale Buchstaben mit/ohne Index	Variablen (z.B. X_1, F_2, A)
kleine normale Buchstaben mit Index i	i. Zustand der jeweiligen Variablen (z.B. x_1, f_4)
Schreibmaschinenfont	Variablennamen (z.B. `Variable`)
kursive Schrift	Merkmalsnamen (z.B. *Merkmal*)
$(.)^+$	Inhalt in Klammern einmal oder beliebig mehr
$(.)^*$	Inhalt in Klammern nullmal oder beliebig mehr

A.2 Mathematische Symbole

i, j	Indizes, $i \neq j$
n, m, N	eine natürliche, positive Zahl
$P(X) \in [0, 1]$	Wahrscheinlichkeitsfunktion der Variablen X
$Pa(X)$	Menge der Elternknoten der Variablen X
$\mathcal{B} = (\mathcal{S}, \mathbf{\Theta}) = (\mathbf{X}, \mathbf{E}, \mathbf{\Theta})$	Bayes'sches Netz $\mathcal{B}$ mit Struktur

	$\mathcal{S}$, bestehend aus Variablen **X** und gerichteten Kanten **E**, und Parametern $\boldsymbol{\Theta}$
$\mathcal{P}$	Verteilung der Verbundwahrscheinlichkeiten eines Bayes'schen Netzes
$\mathcal{Q}$	Ziel-Wahrscheinlichkeitsverteilung, die ein Bayes'sches Netz abbilden soll
$\mathcal{P}^k, \mathcal{P}^g$	Verbundwahrscheinlichkeiten für Soll-Netz (korrekt) und gelerntes Netz
log	Logarithmus zur Basis 2
$\mathcal{D}$	Datenmenge
d_i	Datensatz i der Datenmenge $\mathcal{D}$
Q_{K2}	K2-Metrik
Q_{LOG-K2}	Log-K2-Metrik
Q_{LL}	Log-Likelihood-Metrik
Q_{AIC}	Akaike-Informationskriterium
Q_{MDL}	Minimum-Description-Length
K	Netzkomplexität
D_E	euklidische Distanz
D_K	Kullback-Leibler-Distanz
I	Mutual Information

A.3 Abkürzungen

Abb.	Abbildung
API	Application Programming Interface (dt. Programmierschnittstelle)
CPT	Conditional Probability Table (dt. Tabelle der konditionalen Wahrscheinlichkeiten)
DAG	Directed Acyclic Graph (dt. gerichteter azyklischer Graph)
DG	Directed Graph (dt. gerichteter Graph)

EV	Erklärungsvariable
GUI	Graphical User Interface (dt. Grafische Benutzeroberfläche)
HC	Hill Climbing
HMI	Human Machine Interface (dt. Mensch-Maschine-Schnittstelle)
JPD	Joint Probability Distribution (dt. Verteilung der Verbundwahrscheinlichkeiten)
Kap.	Kapitel
KI	Künstliche Intelligenz
MMI	Multi Media Interface; das Audi Infotainmentsystem
MV	Merkmalsvariable
PSD	Adobe Photoshopformat
RGB	RGB-Farbraum (Rot, Grün, Blau)
RHC	Repeated Hill Climbing
UG	ungerichteter Graph
VV	Verifikationsvariable
bzgl.	bezüglich
bspw.	beispielsweise
d.h.	das heißt
diesbzgl.	diesbezüglich
dt.	deutsch
engl.	englisch
evtl.	eventuell
ggf.	gegebenenfalls
s.	siehe
u.a.	unter anderem
vgl.	vergleiche
z.B.	zum Beispiel

Anhang B

Beispiele für die Berechnung des Konfliktmaßes

B.1 Naiver Bayes-Klassifikator mit spärlichen CPTs

B.1.1 Verbundwahrscheinlichkeitstabelle P(F,Z)

Farbe|Zeichen	wahr	falsch
rot	0,2	0,0
blau	0,2	0,0
gelb	0,2	0,0
anders	0,0	0,4

Tabelle B.1: Verbundwahrscheinlichkeitstabelle $P(F, Z) = P(F|Z) \cdot P(Z)$.

B.1.2 Wahrscheinlichkeit P(F=anders)

$$P(F = a) = P(Z = w, F = a) + P(Z = f, F = a) = 0,0 + 0,4 = 0,4$$

B.1.3 Verbundwahrscheinlichkeitstabelle P(H,Z)

Hoehe|Zeichen	wahr	falsch
h1	0,3	0,0
h2	0,3	0,0
anders	0,0	0,4

Tabelle B.2: Verbundwahrscheinlichkeitstabelle $P(H,Z) = P(H|Z) \cdot P(Z)$.

B.1.4 Wahrscheinlichkeit P(H=h2)

$$P(H = h2) = P(Z = w, H = h2) + P(Z = f, H = h2) = 0,3 + 0,0 = 0,3$$

B.1.5 Verbundwahrscheinlichkeit P(Z, F=anders, H=h2)

$$\begin{aligned} P(Z, F = a, H = h2) &= P(F|Z) \cdot P(H|Z) \cdot P(Z) \\ \Rightarrow P(Z = w, F = a, H = h2) &= 0,0 \cdot 0,5 \cdot 0,6 = 0,0 \\ P(Z = f, F = a, H = h2) &= 1,0 \cdot 0,0 \cdot 0,4 = 0,0 \end{aligned}$$

B.1.6 Konfliktmaß conf(F=anders, H=h2)

$$\begin{aligned} conf(F = a, H = h2) &= log_2 \frac{P(F = a) \cdot P(H = h2)}{P(F = a, H = h2)} \\ = log_2 \frac{P(F = a) \cdot P(H = h2)}{P(Z = w, F = a, H = h2) + P(Z = f, F = a, H = h2)} \\ &= log_2 \frac{0,4 \cdot 0,3}{0,0 + 0,0} \\ &= log_2 \frac{0,4 \cdot 0,3}{0,0} \\ &= \infty \end{aligned}$$

B.2 Naiver Bayes-Klassifikator mit dichten CPTs

B.2.1 Verbundwahrscheinlichkeitstabelle P(F,Z)

Farbe|Zeichen	wahr	falsch
rot	0,1998	0,0004
blau	0,1998	0,0004
gelb	0,1998	0,0004
anders	0,0006	0,3988

Tabelle B.3: Verbundwahrscheinlichkeitstabelle $P(F,Z) = P(F|Z) \cdot P(Z)$.

B.2.2 Wahrscheinlichkeit P(F=anders)

$$P(F = a) = P(Z = w, F = a) + P(Z = f, F = a) = 0,0006 + 0,3988 = 0,3994$$

B.2.3 Verbundwahrscheinlichkeitstabelle P(H,Z)

Hoehe|Zeichen	wahr	falsch
h1	0,2997	0,0004
h2	0,2997	0,0004
anders	0,0006	0,3992

Tabelle B.4: Verbundwahrscheinlichkeitstabelle $P(H,Z) = P(H|Z) \cdot P(Z)$.

B.2.4 WahrscheinlichkeitP (H=h2)

$$P(H = h2) = P(Z = w, H = h2) + P(Z = f, H = h2) = 0,2997 + 0,0004 = 0,3001$$

B.2.5 Verbundwahrscheinlichkeit P(Z, F=anders, H=h2)

$$P(Z, F = a, H = h2) = P(F|Z) \cdot P(H|Z) \cdot P(Z)$$

$$\Rightarrow P(Z = w, F = a, H = h2) = 0,001 \cdot 0,4995 \cdot 0,6 = 0,0002997$$

$$P(Z = f, F = a, H = h2) = 0,997 \cdot 0,0001 \cdot 0,4 = 0,0003988$$

B.2.6 Konfliktmaß conf(F=anders, H=h2)

$$
\begin{aligned}
conf(F = a, H = h2) &= log_2 \frac{P(F = a) \cdot P(H = h2)}{P(F = a, H = h2)} \\
&= log_2 \frac{P(F = a) \cdot P(H = h2)}{P(Z = w, F = a, H = h2) + P(Z = f, F = a, H = h2)} \\
&= log_2 \frac{0,3994 \cdot 0,3001}{0,0002997 + 0,0003988} \\
&= log_2 \frac{0,3994 \cdot 0,3001}{0,0006985} \\
&\approx 7,4
\end{aligned}
$$

Anhang C

GUI-Design-Schemata

C.1 JSON-Schema

```
{
  "type":"object",
  "properties": {
    "GUIName": {
      "type": "string",
  },
  "Dimension": {
    "type": "object",
    "properties": {
      "Width": {
        "type": "integer" },
      "Height": {
        "type": "integer" }
    }
  },
  "Areas": {
    "type": "array",
    "optional": true,
    "minItems": 1,
    "items": {
      "type": "object",
      "properties": {
        "Name": {
          "type": "string" },
        "Dimension": {
          "$ref" : "#/properties/Dimension",
          "optional": true
        },
        "UpperLeftPosition": {
          "type": "object",
          "optional": true,
          "properties": {
            "X": {
              "type": "integer" },
            "Y": {
              "type": "integer" }
          }
        },
        "CenterPosition": {
          "type": "object",
          "optional": true,
          "properties": {
            "X": {
```

```
            "type": "integer" },
          "Y": {
            "type": "integer" }
        } } } }
  },
  "Masters": {
    "type": "array",
    "optional": true,
    "minItems": 1,
    "items": {
      "type": "object",
      "properties": {
        "Name": {
          "type": "string" },
        "AreaReference": {
          "type": "string",
          "optional": true },
        "Symbols": {
          "type": "array",
          "optional": true,
          "minItems": 1,
          "items": {
            "type": "object",
            "properties": {
              "Name": {
                "type": "string" },
              "Optional": {
              "Type": {
                "type": "string"
                "optional": true },
              "Dimension": {
                "$ref": "#/properties/Dimension",
                "optional": true
              },
              "UpperLeftPosition": {
                "$ref": "#/properties/UpperLeftPosition",
                "optional": true
              },
              "CenterPosition": {
                "$ref": "#/properties/CenterPosition",
                "optional": true
              },
              "Paths": {
                "type": "array",
                "optional": true,
                "minItems": 1,
                "items": {
                  "type": "object",
                  "properties": {
                    "Path": {
                      "type": "string" },
                      "State": {
                        "type": "string" }
                } }
              },
              "Color": {
                "type": "object",
                "optional": true,
                "properties": {
                  "Red": {
                    "type": "integer",
                    "minimum": 0,
                    "maximum": 255
                  },
                  "Green": {
                    "type": "integer",
                    "minimum": 0,
                    "maximum": 255
                  },
                  "Blue": {
```

```
                    "type": "integer",
                    "minimum": 0,
                    "maximum": 255
                  } } } } }
      },
      "Texts": {
        "type": "array",
        "optional": true,
        "minItems": 1,
        "items": {
          "type": "object",
          "properties": {
            "Name": {
              "type": "string" },
            "Optional": {
              "type": "boolean",
              "optional": true },
            "Dimension": {
              "$ref" : "#/properties/Dimension",
              "optional": true
            },
            "UpperLeftPosition": {
              "$ref": "#/properties/UpperLeftPosition",
              "optional": true
            },
            "CenterPosition": {
              "$ref": "#/properties/CenterPosition",
              "optional": true
            },
            "Font": {
              "type": "object",
              "optional": true,
              "properties": {
                "Name": {
                  "type": "string" },
                "Style": {
                  "type": "string" },
                "Size": {
                  "type": "integer" },
                "Bold": {
                  "type": "boolean" },
                "Italic": {
                  "type": "boolean" }
              }
            },
            "Color": {
              "$ref": "#/Masters/Symbols/Color"
            } } } } } }
  },
  "Views": {
    "type": "array",
    "minItems": 1,
    "items": {
      "type": "object",
      "properties": {
        "Name": {
          "type": "string" },
        "AreaReference": {
          "type": "string",
          "optional": true },
        "MasterReference": {
          "type": "string",
          "optional": true },
        "Symbols": {
          "$ref": "#/Masters/Symbols"
        },
        "Texts": {
          "$ref": "#/Masters/Texts"
        } } } },
  "Alignments": {
```

```
    "type": "array",
    "optional": true,
    "minItems": 1,
    "items": {
      "type": "object"
      "properties": {
        "ElementReference": {
        "type": "string" },
        "Alignment": {
          "enum": ["left", "center", "right"] }
      } }
  },
  "Distances": {
    "type": "array",
    "optional": true,
    "minItems": 1,
    "items": {
      "type": "object"
      "properties": {
        "ElementReference1": {
          "type": "string" },
        "ElementReference2": {
          "type": "string" },
        "Axis": {
          "enum": ["X", "Y"] },
        "Relation": {
          "enum": ["sideBySide", "enclosing"] },
        "ReferencePoint": {
          "enum": ["border", "border_top", "border_bottom", "border_left", "border_right", "center"] },
        "Distance": {
          "type": "integer" }
      } }
  },
  "Coexistences": {
    "type": "array",
    "optional": true,
    "minItems": 1,
    "items": {
      "type": "object"
      "properties": {
        "ElementReferences": {
          "type": "array",
          "minItems": 2,
          "items": {
            "type": "string" }
        } } }
  },
  "Mutualexclusions": {
    "type": "array",
    "optional": true,
    "minItems": 1,
    "items": {
      "type": "object"
      "properties": {
        "ElementReferences": {
          "type": "array",
          "minItems": 2,
          "items": {
            "type": "string" }
        } } }
  }
}
```

C.2 XML-Schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="Feature">
    <xs:attribute name="State" type="State" default="none"/>
  </xs:complexType>
  <xs:complexType name="Position" >
    <xs:complexContent>
      <xs:extension base="Feature">
        <xs:sequence>
          <xs:element name="XPosition" type="FT" minOccurs="0"/>
          <xs:element name="YPosition" type="FT" minOccurs="0"/>
          <xs:element name="XCenterPosition" type="FT" minOccurs="0"/>
          <xs:element name="YCenterPosition" type="FT" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Dimension" >
    <xs:complexContent>
      <xs:extension base="Feature">
        <xs:sequence>
          <xs:element name="Width" type="FT" minOccurs="0"/>
          <xs:element name="Height" type="FT" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
     </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Color">
    <xs:complexContent>
      <xs:extension base="Feature">
         <xs:attribute name="Red" type="RGBT" use="required"/>
         <xs:attribute name="Green" type="RGBT" use="required"/>
         <xs:attribute name="Blue" type="RGBT" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Font">
    <xs:complexContent>
      <xs:extension base="Feature">
        <xs:attribute name="Name" type="xs:string" use="required" />
        <xs:attribute name="Style" type="xs:string" use="required" />
        <xs:attribute name="Size" type="xs:string" use="required" />
        <xs:attribute name="Bold" type="xs:boolean" use="required" />
        <xs:attribute name="Italic" type="xs:boolean" use="required" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Path">
    <xs:complexContent>
      <xs:extension base="Feature">
        <xs:attribute name="Path" type="xs:string" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Graphicelement"  abstract="true">
        <xs:sequence>
          <xs:element name="POSITION" type="Position" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="Name" type="xs:string" use="required"/>
        <xs:attribute name="Optional" type="xs:boolean"/>
  </xs:complexType>
  <xs:complexType name="TEXT">
    <xs:complexContent>
      <xs:extension base="Graphicelement">
        <xs:sequence>
          <xs:element name="FONT" type="Font" maxOccurs="unbounded"/>
          <xs:element name="COLOR" type="Color" maxOccurs="unbounded"/>
```

```
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="SYMBOL">
    <xs:complexContent>
      <xs:extension base="Graphicelement">
        <xs:attribute name="Type" type="xs:string"/>
        <xs:sequence>
          <xs:element name="PATH" type="Path" minOccurs="1" maxOccurs="unbounded"/
          <xs:element name="COLOR" type="Color" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="TextRef" >
    <xs:attribute name="Name" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="SYMBOLRef" >
    <xs:attribute name="Name" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="SYMBOLgroup" >
    <xs:sequence>
      <xs:element name="SYMBOLREF" type="SYMBOLRef" maxOccurs="unbounded"/>
      <xs:element name="PROPERTY" type="Property" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Textgroup" >
    <xs:sequence>
      <xs:element name="TEXTREF" type="TextRef" maxOccurs="unbounded"/>
      <xs:element name="PROPERTY" type="Property" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:group name="GRAPHICELEMENTS">
    <xs:choice>
      <xs:element name="TEXTREF" type="TextRef"/>
      <xs:element name="SYMBOLREF" type="SYMBOLRef"/>
      <xs:element name="SYMBOLGROUP" type="SYMBOLgroup"/>
      <xs:element name="TEXTGROUP" type="Textgroup"/>
    </xs:choice>
  </xs:group>
  <xs:complexType name="MasterRef" >
    <xs:attribute name="Name" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:simpleType name="Axis_Type">
    <xs:restriction base="xs:string">
      <xs:enumeration value="x"/>
      <xs:enumeration value="y"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="ReferencePoint_Type">
    <xs:restriction base="xs:string">
      <xs:enumeration value="border"/>
      <xs:enumeration value="border_top"/>
      <xs:enumeration value="border_bottom"/>
      <xs:enumeration value="border_left"/>
      <xs:enumeration value="border_right"/>
      <xs:enumeration value="center"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="Relation_Type">
    <xs:restriction base="xs:string">
      <xs:enumeration value="sideBySide"/>
      <xs:enumeration value="enclosing"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="Alignment_Type">
      <xs:restriction base="xs:string">
      <xs:enumeration value="left"/>
      <xs:enumeration value="center"/>
```

```
      <xs:enumeration value="right"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="GlobalFeature">
    <xs:complexContent>
      <xs:extension base="Feature">
        <xs:sequence>
          <xs:element name="PROPERTY" type="Property" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Distance" >
    <xs:complexContent>
      <xs:extension base="GlobalFeature">
        <xs:group ref="GRAPHICELEMENTS" minOccurs="2" maxOccurs="2"/>
        <xs:attribute name="Axis" type="Axis_Type" use="required"/>
        <xs:attribute name="ReferencePoint" type="ReferencePoint_Type" use="required"/>
        <xs:attribute name="Relation" type="Relation_Type" default="sideBySide"/>
        <xs:attribute name="Distance" type="IT" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Coexistence" >
    <xs:complexContent>
      <xs:extension base="GlobalFeature">
        <xs:group ref="GRAPHICELEMENTS" minOccurs="2" maxOccurs="unbounded"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="OptionalCoexistence" >
    <xs:complexContent>
      <xs:extension base="GlobalFeature">
        <xs:group ref="GRAPHICELEMENTS" minOccurs="2" maxOccurs="2"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Alignment" >
    <xs:complexContent>
      <xs:extension base="GlobalFeature">
        <xs:group ref="GRAPHICELEMENTS" maxOccurs="unbounded"/>
        <xs:attribute name="Align" type="Alignment_Type" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Mutualexclusion" >
    <xs:complexContent>
      <xs:extension base="GlobalFeature">
        <xs:group ref="GRAPHICELEMENTS" minOccurs="2" maxOccurs="unbounded"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Area" >
    <xs:sequence>
      <xs:element name="SIZE" type="Dimension"/>
      <xs:element name="POSITION" type="Position"/>
      <xs:group ref="GRAPHICELEMENTGROUP"/>
    </xs:sequence>
    <xs:attribute name="Name" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="AreaRef" >
    <xs:sequence>
      <xs:group ref="GRAPHICELEMENTGROUP"/>
    </xs:sequence>
    <xs:attribute name="Name" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:group name="GRAPHICELEMENTGROUP">
    <xs:sequence>
      <xs:element name="TEXT" type="Text" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="SYMBOL" type="SYMBOL" minOccurs="0" maxOccurs="unbounded">
```

```
      </xs:element>
      <xs:element name="ALIGNMENT" type="Alignment" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="DISTANCE" type="Distance" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="COEXISTENCE" type="Coexistence" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="MUTALEXCLUSION" type="Mutualexclusion" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:group>
  <xs:complexType name="Master" >
    <xs:sequence>
      <xs:group ref="GRAPHICELEMENTGROUP"/>
      <xs:element name="AREAREF" type="AreaRef" minOccurs="0" maxOccurs="unbounded">
      </xs:element>
    </xs:sequence>
    <xs:attribute name="Name" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="View" >
    <xs:sequence>
      <xs:group ref="GRAPHICELEMENTGROUP"/>
      <xs:element name="MASTERREF" type="MasterRef" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="AREAREF" type="AreaRef" minOccurs="0" maxOccurs="unbounded">
      </xs:element>
    </xs:sequence>
    <xs:attribute name="Name" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="GUI" >
    <xs:sequence>
      <xs:element name="DIMENSION" type="Dimension"/>
      <xs:element name="AREA" type="Area" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="MASTER" type="Master" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="VIEW" type="View" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Name" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:element name="GUI" type="GUI">
    <xs:key name="SYMBOLKey">
      <xs:selector xpath="VIEW/AREAREF/SYMBOL"/>
      <xs:field xpath="@Name"/>
    </xs:key>
    <xs:keyref name="SYMBOLKeyRef" refer="SYMBOLKey">
      <xs:selector xpath="VIEW/AREAREF/*/SYMBOLREF"/>
      <xs:field xpath="@Name"/>
    </xs:keyref>
    <xs:keyref name="SYMBOLGroupKeyRef" refer="SYMBOLKey">
      <xs:selector xpath="VIEW/AREAREF/*/SYMBOLGROUPREF/SYMBOLREF"/>
      <xs:field xpath="@Name"/>
    </xs:keyref>
    <xs:key name="TextKey">
      <xs:selector xpath="VIEW/AREAREF/TEXT"/>
      <xs:field xpath="@Name"/>
    </xs:key>
    <xs:keyref name="TextKeyRef" refer="TextKey">
      <xs:selector xpath="VIEW/AREAREF/*/TEXTREF"/>
      <xs:field xpath="@Name"/>
    </xs:keyref>
    <xs:key name="AreaKey">
      <xs:selector xpath="AREA"/>
      <xs:field xpath="@Name"/>
    </xs:key>
    <xs:keyref name="AreaKeyRefForMaster" refer="AreaKey">
      <xs:selector xpath="MASTER/AREAREF"/>
      <xs:field xpath="@Name"/>
    </xs:keyref>
    <xs:keyref name="AreaKeyRefForView" refer="AreaKey">
      <xs:selector xpath="VIEW/AREAREF"/>
      <xs:field xpath="@Name"/>
    </xs:keyref>
    <xs:key name="MasterKey">
      <xs:selector xpath="MASTER"/>
      <xs:field xpath="@Name"/>
    </xs:key>
```

```
      <xs:keyref name="MasterRef" refer="MasterKey">
        <xs:selector xpath="VIEW/MASTERREF"/>
        <xs:field xpath="@Name"/>
      </xs:keyref>
    </xs:element>
    <xs:complexType name="Property">
      <xs:attribute name="PropertyName" type="xs:string" use="required"/>
      <xs:attribute name="Value" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:simpleType name="State">
      <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:simpleType name="FT">
      <xs:restriction base="xs:float">
        <xs:minInclusive value="0.0"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="IT">
      <xs:restriction base="xs:integer">
        <xs:minInclusive value="1"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="RGBT">
      <xs:restriction base="xs:integer">
        <xs:minInclusive value="0"/>
        <xs:maxInclusive value="255"/>
      </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

Anhang D

Software für Bayes'sche Netze

Die kommerziellen Softwareprogramme *HUGIN* (*HUGIN EXPERT* 1989-2011) und *Netica* (*Netica* 1995-2011), erstmals veröffentlicht 1989 und 1995, haben einen sehr großen Leistungsumfang. Beide Programme stellen bspw. verschiedene Parametermodelle wie z.B. *Noisy-OR* für effiziente Inferenzberechnungen bereit und werden stetig um neue Funktionen erweitert. In der Software HUGIN ist zudem eine Rechenzeitoptimierung durch Vermeidung von Mehrfachberechnungen von Multiplikationen realisiert. HUGIN und Netica unterstützen auch die Verwendung unsicherer Evidenzen.
Zwei gute kostenlose Programme sind *BayesBuilder* (*BayesBuilder* 2007-2010) und *JavaBayes* (Gagliardi Cozman 2001). Sie unterstützen die wichtigsten Funktionen für den Aufbau Bayes'scher Netze mithilfe einer grafischen Benutzeroberfläche und Inferenzberechnungen mittels des Junction-Tree-Algorithmus[1]. BayesBuilder besitzt allerdings für Inferenzberechnungen den Nachteil, dass diese nur bis zu einer Cliquengröße von 30 möglich sind. BayesBuilder existiert seit dem Jahr 2007 und wird aktuell weiterentwickelt. Hingegen erschien für JavaBayes die bislang letzte Version 2001. Vorteil von JavaBayes ist jedoch, dass hier Inferenzberechnungen ohne Netzgrößenbeschränkung unterstützt werden. Zudem wird neben dem Junction-Tree-Algorithmus auch die Marginalisierung angeboten. In Korb und Nicholson 2004 werden diese vier Software-Programme sowie einige Weitere recht ausführlich vorgestellt und verglichen. Die bereits genannten kommerziellen Softwareprogramme HUGIN und Netica bieten beide die Möglichkeit, Parameter mittels des

[1] Der Junction-Tree-Algorithmus wird in JavaBayes irreführenderweise als *Bucket Tree Elimination* bezeichnet.

EM-Algorithmus zu lernen. Netica bietet darüber hinaus auch für unvollständige Daten geeignete Lernverfahren an. Daneben kann mit HUGIN auch die Struktur mittels eines einschränkenden Verfahrens, dem PC-Algorithmus (s. z.B. Korb und Nicholson 2004, Kap. 6.3.2), gelernt werden (Korb und Nicholson 2004, Anhang B.4.5 und B.4.8). BayesBuilder und JavaBayes unterstützen kein Lernen. Bzgl. dem Lernen von Bayes'schen Netzen ist des Weiteren die OpenSource-Software *WEKA* (Bouckaert 2006-2007) zu nennen. Sie bietet eine sehr umfangreiche Zusammenstellung lokaler und globaler Suchverfahren, einschränkender Verfahren und Optimierungsverfahren sowie verschiedene Schätzer für das Parameterlernen. Bei sehr kleinen Datenmengen kann die Bayes'sche Schätzung zum Parameterlernen angewendet werden, um gute Ergebnisse zu erzielen. Des Weiteren können die Daten auf Wunsch auf verschiedene Weise gefiltert werden, was große Datenmengen geeignet reduzieren kann. Gelernte Netze können zudem mittels Kreuz-Validierung auf Korrektheit geprüft werden. Nachteile von WEKA sind jedoch, dass Datenmengen größer als ca. 20.000 Datensätze nicht verarbeitet werden können und kein grafischer Editor zum manuellen Testen der gelernten Netze angeboten wird. Für den Export in z.B. JavaBayes wird aber das gängige BIFXML-Format (s. Gagliardi Cozman 1998) unterstützt.

Literaturverzeichnis

Abrams, M. u. a. (1999). „UIML: An Appliance-Independent XML User Interface Language.“ In: *Computer Networks* 31.11-16, S. 1695–1708.

Adobe Photoshop (2012). `http://www.adobe.com/de/products/photoshop.html`. Stand 31.07.2012.

Akaike, H. (1973). „Information theory and an extension of the maximum likelihood principle“. In: *Proc. of the 2nd Int. Symp. on Information Theory*. Hrsg. von Petrov B. N. und Csaki F., S. 267–281.

Amores, S. (2011). *Multimedia-Zukunft von Mercedes-Benz, 24.07.2011*. `http://www.prestigecars.de/luxus-magazin/2011/multimedia-zukunft-von-mercedes-benz/21580`. Pressemitteilung.

BayesBuilder (2007-2010). `http://www.snn.ru.nl`. Stand 08.09.2011.

Beierle, C. und G. Kern-Isberner (2006). *Methoden wissensbasierter Systeme: Grundlagen - Algorithmen - Anwendungen*. Vieweg + Teubner.

Betz, M. (Apr. 2010). *MMI 3G PLUS (D4) - GUI Catalogue*. Specification Document. AUDI AG.

Bilmes, J. (2000). „Dynamic Bayesian Multinets.“ In: *UAI'00*, S. 38–45.

– (2004). *On Soft Evidence in Bayesian Networks*. Techn. Ber. UWEETR-2004-0016. University of Washington, Dept. of Electrical Engineering. URL: `https://www.ee.washington.edu/techsite/papers/refer/UWEETR-2004-0016.html`.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New-York, Inc.

Borth, M. und H. von Hasseln (2002). „Systematic Generation of Bayesian Networks from Systems Specifications“. In: *Proceedings of the IFIP 17th World Computer Congress - TC12 Stream on Intelligent Information Processing*. Deventer, The Netherlands: Kluwer, B.V., S. 155–166.

Bouckaert, R. R. (2006-2007). *Bayesian Network Classifiers in Weka for Version 3-5-7*. http://www.cs.waikato.ac.nz/~remco/weka_bn.

Briganti, D. (2012). *JSON-XML-Converter*. http://jsontoxml.utilities-online.info. Stand 25.05.2012.

Bringmann, E. und A. Krämer (2008). „Model-Based Testing of Automotive Systems“. In: *2008 International Conference on Software Testing Verification and Validation*, S. 485–493.

Brunhorn, J. (2007). *XML-Sprache zur Beschreibung von HMIs für Infotainmentsysteme und Kombiinstrumente. Language Specification 1.0*. Techn. Ber. Carmeq GmbH / OEM Arbeitskreis HMI Methodik.

Buchanan, B. G. und E. H. Shortliffe (1984). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, S. 209–232.

Campos, P. F. und N. J. Nunes (2004). „CanonSketch: a User-Centered Tool for Canonical Abstract Prototyping“. In: *In Proceedings of DSV-IS'2004, 11th International Workshop on Design, Specification and Verification of Interactive Systems*. Springer-Verlag.

Chan, H. und A. Darwiche (März 2005). „On the revision of probabilistic beliefs using uncertain evidence“. In: *Artif. Intell.* 163 (1), S. 67–90.

Chang, T.-H., T. Yeh und R. C. Miller (2010). „GUI testing using computer vision“. In: *Proceedings of the 28th international conference on Human factors in computing systems CHI 10* Figure 1, S. 1535–1544.

Constantine, L. L. u. a. (Juli 2000). „From Abstraction to Realization in User Interface Design: Abstract Prototypes Based on Canonical Components“. In: Working Paper, The Convergence Colloquy. URL: www.foruse.com/articles/canonical.pdf.

Constantine, L. (2003). „Canonical Abstract Prototypes for abstract visual and interaction design“. In: *Proceedings of DSV-IS'2003, 10th International Conference on Design, Specification and Verification of Interactive Systems*. Hrsg. von J. Jorge, N. Nunes und J. Falcão e Cunha. Bd. 2844. Springer-Verlag, Berlin Heidelberg New York.

Cooper, G. F. und E. Herskovits (Nov. 1992). „A Bayesian Method for the Induction of Probabilistic Networks from Data“. In: *Mach. Learn.* 9 (4), S. 309–347.

Court, G. (2010). *JSON Schema Validator*. `https://github.com/garycourt/JSV`. Stand 21.08.2012.

Cover, T. M. und J. A. Thomas (1991). „Entropy, Relative Entropy and Mutual Information". In: *Elements of information theory*. New York, NY, USA: Wiley-Interscience. Kap. 2, S. 12–49.

Dixon, M., D. Leventhal und J. Fogarty (2011). „Content and hierarchy in pixel-based methods for reverse engineering interface structure". In: *Proceedings of the 2011 annual conference on Human factors in computing systems*. CHI '11. New York, NY, USA: ACM, S. 969–978.

Draper, D. und S. Hanks (1994). „Localized Partial Evaluation of Belief Networks". In: *Proceedings of the Tenth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*. San Francisco, CA: Morgan Kaufmann, S. 170–177.

Duan, L., A. Höfer und H. Hussmann (2010). „Model-Based Testing of Automotive HMIs based on a Test-Oriented HMI Specification Model". In: *Proceedings of the the 2nd International Conference on Advances in System Testing and Validation Lifecycle*. VALID 2010. IEEE.

Duda, R. O. und P. E. Hart (1973). *Pattern Classification and Scene Analysis*. John Willey & Sons.

EVOLUS (2010). *Evolus Pencil*. `http://pencil.evolus.vn/en-US/Home.aspx`.

Efron, B. und R. Tibshirani (1993). *An introduction to the bootstrap*. Monographs on statistics and applied probability. Chapman & Hall.

Erhardt, A. (2008). *Einführung in die Digitale Bildverarbeitung: Grundlagen, Systeme und Anwendungen*. Vieweg + Teubner.

European Communities, Commission of the (2006). *Commission Recommendation of 22 December 2006 on safe and efficient in-vehicle information and communication systems: Update of the European Statement of Principles on human machine interface*.

Fensel, D. (1994). „Sinn und Unsinn formaler Spezifikationssprachen für wissensbasierte Systeme". In: *Künstliche Intelligenz (KI)* 4.

Flesch, I. und P. Lucas (2007). „Markov Equivalence in Bayesian Networks". In: *Advances in Probabilistic Graphical Models*. Hrsg. von Peter Lucas, José Gá-

mez und Antonio Salmerón. Bd. 214. Studies in Fuzziness and Soft Computing. Springer Berlin / Heidelberg, S. 3–38.

Frank, A. und A. Asuncion (2010). *UCI Machine Learning Repository*. URL: http://archive.ics.uci.edu/ml.

Frege, G. (1879). *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Nebert.

Friedman, N. (1999). „Learning Bayesian Network Structure from Massive Datasets : The 'Sparse Candidate' Algorithm Background : Learning Structure". In: *Science*, S. 206–215.

Gagliardi Cozman, F. (1998). *BIFXML*. http://sites.poli.usp.br/p/fabio.cozman/Research/InterchangeFormat/index.html. Stand 08.09.2011.

– (2001). *JavaBayes*. http://www.cs.cmu.edu/~javabayes/Home. Stand 25.03.2011.

Geiger, D. und D. Heckerman (1996). „Knowledge representation and inference in similarity networks and Bayesian multinets". In: *Artificial Intelligence* 82.95, S. 45–74.

Geyer, F., T. Memmel und H. Reiterer (Aug. 2010). „Interactive User Interface Specifications - Supporting Collaboration and Creativity with Simulation-Driven Modelling Tools". In: *IADIS International Journal on WWW* 6.2, S. 63–79.

Grady, J. O. (2007). *System Verification: Proving the Design Solution Satisfies the Requirements*. Academic.

HUGIN EXPERT (1989-2011). http://www.hugin.com. Stand 08.09.2011.

Halpern, J. Y. und R. Pucella (2009). „Evidence with uncertain likelihoods." In: *Synthese* 171.1, S. 111–133.

Heckerman, D. (1985). „Probabilistic Interpretations for MYCIN's Certainty Factors". In: *Proceedings of the Proceedings of the First Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-85)*. New York, NY: Elsevier Science, S. 9–20.

Heckerman, D., D. Geiger und D. M. Chickering (1994). *Learning Bayesian networks: The combination of knowledge and statistical data*. Technical Report MSR-TR-94-09. Redmond, WA: Microsoft Research.

Heilemann, M. und G. Palm (Okt. 2011). „Automatisierte Verifikation des Designs der grafischen Benutzeroberfläche von Infotainmentsystemen mit einem Bayes'schen Netzwerk". In: *INFORMATIK 2011*. GI Gesellschaft für Informa-

tik e.V. Köllen Druck+Verlag GmbH. URL: http://www.user.tu-berlin.de/komm/CD/paper/060131.pdf.

Helms, J. u. a. (2008). *The User Interface Markup Language (UIML) version 4.0*. OASIS specification. Stand 06.06.2012. URL: https://www.oasis-open.org/committees/download.php/28457/uiml-4.0-cd01.pdf.

Herskovits, E. (1991). „Computer-based probabilistic-network construction". UMI Order No. GAX92-05646. Diss. Stanford, CA, USA.

Hoffmann, D. W. (2008). *Software-Qualität*. Springer.

Holmes, D. und L. Jain, Hrsg. (2008). *Innovations in Bayesian Networks - Theory and Applications. Series: Studies in Computational Intelligence*. Bd. 156. Springer Berlin- Heidelberg.

Huang, Y. u. a. (2010a). „Development of an automated testing system for vehicle infotainment system". In: *The International Journal of Advanced Manufacturing Technology*, S. 1–14.

– (2010b). „Development of an automated testing system for vehicle infotainment system". In: *The International Journal of Advanced Manufacturing Technology*.

JSON Schema Google Group. Stand 21.08.2012. URL: https://groups.google.com/forum/?fromgroups#!forum/json-schema.

JSON.org - Introducing JSON. http://json.org. Stand 25.05.2012.

Jeffrey, R. C. (1965). *The logic of decision*. McGraw-Hill series in probability and statistics. McGraw-Hill.

Jensen, F. V. und F. Jensen (1994). „Optimal Junction Trees". In: *In UAI*. Morgan Kaufmann, S. 360–366.

Jensen, F. V. und T. D. Nielsen (2007). *Bayesian networks and decision graphs*. Springer, S. 447.

Jensen, F. V. u. a. (1991). „Analysis in HUGIN of data conflict". In: *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*. UAI '90. New York, NY, USA: Elsevier Science Inc., S. 519–528.

Juran, J. M. u. a. (1999). „Juran's Quality Handbook". In: *Training for Quality* 1.2, S. 1699.

Kappen, H (2005). „A probabilistic decision support system for medical diagnosis SNN". In: *Neural Networks*, S. 26.

Kern, D. und A. Schmidt (2009). „Design space for driver-based automotive user interfaces". In: *Proceedings of the 1st International Conference on Automotive*

User Interfaces and Interactive Vehicular Applications. AutomotiveUI '09. New York, NY, USA: ACM, S. 3–10.

Kervinen, A. u. a. (2006). „Model-based testing through a GUI". In: *In Proceedings of the 5th International Workshop on Formal Approaches to Testing of Software (FATES 2005), number 3997 in Lecture Notes in Computer Science*. Springer, S. 16–31.

Kjærulff, U. (1992). *A Computational Scheme For Reasoning In Dynamic Probabilistic Networks*.

Konzeptstudie Audi Avantissimo (2012). `http://www.audi.de/de/brand/de/erlebniswelt/design___technologie/design-studien/avantissimo.html`. Stand 13.06.2012.

Korb, K. B. und A. E. Nicholson (2004). *Bayesian Artificial Intelligence*. Bd. 7. 2. Springer-Verlag.

Koski, T. und J. Noble (2011). *Bayesian Networks: An Introduction*. Wiley Series in Probability and Statistics. John Wiley & Sons.

Kruse, R., J. Gebhardt und F. Klawonn (1993). *Fuzzy-Systeme*. Leitfäden und Monographien der Informatik. Teubner Verlag, Stuttgart.

Kschischang, F. R., B. J. Frey und H.-A. Loeliger (1998). „Factor Graphs and the Sum-Product Algorithm". In: *IEEE TRANSACTIONS ON INFORMATION THEORY* 47, S. 498–519.

Kullback, S. und R. A. Leibler (1951). „On Information and Sufficiency". In: *Ann. Math. Statist.* 22.1, S. 79–86.

Lacave, C. und F. J. Díez (2002). „A review of explanation methods for Bayesian networks". In: *Knowledge Engineering Review* 17, S. 107–127.

Lauritzen, S. L. (1996). *Graphical models*. Oxford science publications. Clarendon Press.

Lauritzen, S. L. und D. J. Spiegelhalter (1988). „Local Computation with Probabilities on Graphical Structures and their Application to Expert Systems". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 50.2, S. 157–224.

Li, K. und M. Wu (2004). *Effective GUI Test Automation: Developing an Automated GUI Testing Tool*. John Wiley & Sons.

Luo, J., A. E. Savakis und A. Singhal (2005). „A Bayesian network-based framework for semantic image understanding". In: *Pattern Recognition* 38.6, S. 919 –934.

Luyten, K. (2004). „Dynamic User Interface Generation for Mobile and Embedded Systems with Model-Based User Interface Development". Diss. Doktorarbeit, Universit?t Limburg. URL: http://research.edm.uhasselt.be/kris/research/phd/v1.pdf.

Marwedel, P. (2007). „Spezifikationssprachen". In: *Eingebettete Systeme*. eXamen.press. Springer Berlin Heidelberg, S. 13–93.

Memmel, T. und H. Reiterer (März 2008). „Inspector: Method and Tool for Visual UI Specification". In: *Proceedings of the 3rd IASTED International Conference on Human Computer Interaction (IASTED-HCI 08)*. Acta Press, Canada, S. 170–179.

Memmel, Thomas, Jean Vanderdonckt und Harald Reiterer (2008). „Multi-fidelity User Interface Specifications". In: *Interactive Systems. Design, Specification, and Verification*. Hrsg. von T. C. Graham und P. Palanque. Berlin, Heidelberg: Springer-Verlag, S. 43–57.

Microsoft (2012). *XAML in WPF*. http://msdn.microsoft.com/en-us/library/ms747122. Stand 20.08.2012.

Netica (1995-2011). http://www.norsys.com. Stand 08.09.2011.

Nischwitz, A., M. Fischer und P. Haberäcker (2007). *Computergrafik und Bildverarbeitung: Alles für Studium und Praxis- Bildverarbeitungswerkzeuge, Beispiel-Software und interaktive Vorlesungen online verfügbar*. Vieweg.

Nürenberg, A. (2012). *Entwicklung und Implementierung eines multikernfähigen Plug-in basierten Bayes'schen Netzwerk Strukturfinder Tools*. Diplomarbeit, Hochschule Weihenstephan-Triesdorf und AUDI AG.

Nunes, N. J. und J. F. Cunha (2001). „Wisdom: a UML based architecture for interactive systems". In: *Proceedings of the 7th international conference on Design, specification, and verification of interactive systems*. DSV-IS'00. Limerick, Ireland: Springer-Verlag, Berlin, Heidelberg, S. 191–205.

Olesen, K. G. u. a. (Nov. 1989). „A munin network for the median nerve-a case study on loops". In: *Appl. Artif. Intell.* 3.2-3, S. 385–403.

Pavlovic, V. (Jan. 1999). „Dynamic Bayesian Networks for Information Fusion with Application to Human-Computer Interfaces". Diss. University of Illinois at

Urbana-Champaign. URL: http://www.cs.rutgers.edu/~vladimir/pub/phd.pdf.

Pearl, J. (1986). „Fusion, propagation, and structuring in belief networks“. In: *Artificial Intelligence* 29.3, S. 241–288.

– (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Promoli, B. (Jan. 2011). *Layout & Positioning Guidelines, 3G Plus - elements*. Techn. Ber. Version 5.5. Uedelhoven Studios.

Rebane, G. und J. Pearl (1987). „The Recovery of Causal Poly-Trees from Statistical Data“. In: *UAI 87: Proceedings of the Third Annual Conference on Uncertainty in Artificial Intelligence*. Hrsg. von Laveen N. Kanal, Tod S. Levitt und John F. Lemmer. Elsevier, S. 175–182.

Reich, B. (2008). *Abstrakte Beschreibung automobiler HMI-Systeme und deren Erweiterung für neue Dienste*. Diplomarbeit, Fakultät für Ingenieurswissenschaften und Informatik, Institut für Medieninformatik, Universität Ulm.

Rissanen, J. (1978). „Modeling by the shortest data description“. In: *Automation* 14, S. 465–471.

Russell, S. J. u. a. (1996). *Artificial intelligence: a modern approach*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Schad, T. (2011). *Entwurf und Entwicklung eines Bilderkennungssystems zum automatisierten Testen des AUDI MMI Displays*. Masterarbeit, FH Ingolstadt und AUDI AG.

Shafer, G. .R und P. P. Shenoy (1990). „Probability propagation“. In: *Annals of Mathematics and Artificial Intelligence* 2.1-4, S. 327–352.

Spreckelsen, C. und K. Spitzer (2008). *Wissensbasen und Expertensysteme in der Medizin: KI-Ansätze zwischen klinischer Entscheidungsunterstützung und medizinischem Wissensmanagement*. Teubner Studienbücher. Vieweg + Teubner.

Stehle, R. M. (2010). *CE im Auto - Vom ersten Autoradio zum mobilen Multimedia-Center*. http://www.gfu.de/home/historie/autoradio.xhtml. Stand 14.06.2012.

Stevens, A. u. a. (2002). *Design guidelines for safety of in-vehicle information systems*. Techn. Ber. Transport Research LaboratoryPA3721/01.

Steyer, R. (2010). *Das JavaScript-Handbuch: Einführung, Praxis und Referenz*. Programmer's Choice. Addison Wesley Verlag.

Stolle, R. u. a. (2005). „Challenges in Automated Model-Based HMI Testing“. In: *GI Jahrestagung (2)'05*, S. 186–190.

Tièschky, S. u. a. (2002). „Audi MMI - von der Idee zum Produkt“. In: *Elektronik Automotive - Sonderausgabe Audi A8*, S. 10–18. URL: http://www.elektroniknet.de.

Tizhoosh, H. R. (1998). *Fuzzy-Bildverarbeitung: Einführung in Theorie und Praxis*. Springer.

UsiXML-Schema, MutualEmphasis (2010). http://www.usixml.org/documentation/usixml1.8.0/UsiXML.xsd.html_h981898579.html. Stand 20.08.2012.

UsiXML.org (2009-2011). http://www.usixml.org. Stand 07.09.2012.

W3C (2011). *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. http://www.w3.org/TR/SVG. Stand 20.08.2012.

WhatTheFont. Stand 29.12.2012. URL: http://www.myfonts.com/WhatTheFont.

Wiegerinck, W. und T. Heskes (2001). „Probability Assessment with Maximum Entropy in Bayesian Networks“. In: *Computing Science and Statistics, volume 33. Interface*.

Wiegerinck, W., B. Kappen und E. Burgers (2010). „Bayesian Networks for Expert Systems: Theory and Practical Applications“. In: *Interactive Collaborative Information Systems*, S. 547–578.

Wietzke, J. und M.T. Tran (2005). *Automotive Embedded Systeme: Effizientes Framework - Vom Design Zur Implementierung*. Xpert. press Series. Springer.

Wittig, F. (2003). *Maschinelles Lernen Bayes'scher Netze für benutzeradaptive Systeme.* Bd. 267. DISKI. Infix Akademische Verlagsgesellschaft, S. I–XIII, 1–219.

Zadeh, L. A. (1965). „Fuzzy Sets“. In: *Information and Control* 8.3, S. 338–353.

Zyp, K. (Nov. 2010). *A JSON Media Type for Describing the Structure and Meaning of JSON Documents*. Hrsg. von K. Zyp. http://tools.ietf.org/html/draft-zyp-json-schema-03. Stand 21.08.2012.