Automatisierte Verifikation des Designs der grafischen Benutzeroberfläche von Infotainmentsystemen mit einem Bayes'schen Netzwerk

Miriam Heilemann
I/EE-56
AUDI AG Ingolstadt, Deutschland
miriam.heilemann@t-online.de

Günther Palm
Institut für Neuroinformatik
Universität Ulm, 89081 Ulm, Deutschland

Abstract: Die Bereitstellung von elektronischen Komponenten im Automobil in Form von Fahrerassistenzsystemen, Nachrichten- und Unterhaltungsmedien gewinnt zunehmend an Bedeutung und ist längst ein wettbewerbsrelevanter Faktor. Ein Infotainmentsystem ermöglicht durch eine grafische Benutzeroberfläche (GUI) die Interaktion mit diesen Funktionalitäten. Da die Komplexität moderner Infotainmentsysteme aufgrund der wachsenden Funktionsvielfalt stetig zunimmt, ist ein umfangreiches automatisiertes Testen erforderlich. Die Verifikation der GUI ist dabei eine besondere Herausforderung. Dieser Artikel stellt ein neues Verfahren vor um das durch Bildverarbeitungsmethoden klassifizierte GUI-Design zu verifizieren. Unter Anwendung eines Bayes'schen Netzwerks werden die unsicheren Informationen der Bildverarbeitung mit der Designspezifikation verglichen. Am Beispiel eines Infotainmentsystems der AUDI AG werden der Aufbau und die Leistungsfähigkeit des entwickelten Verifikationsverfahrens erläutert.

1 Einleitung

In den vergangenen Jahren haben sich Infotainmentsysteme von Automobilen zu einem sehr wettbewerbsrelevanten Faktor entwickelt. Ein Infotainmentsystem ist ein komplexes System bestehend aus Hard- und Softwarekomponenten und stellt dem Anwender mittels einer grafischen Benutzeroberfläche (GUI) technische Informationen über das Fahrzeug, Nachrichten- und Unterhaltungsmedien bereit, oft sind auch ein Navigationssystem und Fahrerassistenzsysteme integriert. Der Funktionsumfang und die grafische Darstellung variieren je nach Automarke und Systemversion.
Wie alle übrigen Komponenten eines Automobils muss auch das Infotainmentsystem verifiziert und validiert[1] werden bevor es an Kunden verkauft wird. Die Komplexität moderner Infotainmentsysteme nimmt aufgrund der wachsenden Funktionsvielfalt stetig zu und erfordert ein umfangreiches automatisiertes Testen des gesamten Leistungsumfangs. Das Audi-Infotainmentsystem, im Folgenden kurz als MMI (MultiMediaInterface) bezeichnet, bietet mit dem Display, dem Bedienteil und dem Sprachdialogsystem mehrere Schnittstellen für die Interaktion zwischen Infotainmentsystem und Anwender an und stellt damit

[1] Verifikation: System genügt der Spezifikation und liefert darauf basierend korrekte Ergebnisse; Validierung: System erfüllt Anforderungen der Benutzer bzgl. des Einsatzzwecks

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4
weitere Artikel online:
http://informatik2011.de/519.html

ein umfangreiches HMI (engl. Human Machine Interface) zur Verfügung. Das Aussehen der auf dem Display dargestellten GUI ist für jede Version des MMI klar spezifiziert und konzipiert um eine intuitive, ablenkungsarme Bedienung während des Fahrens zu ermöglichen. In diesem Artikel wird ein Verfahren vorgestellt um das grafische Design von GUIs, im Besonderen von GUIs des MMI, automatisiert zu testen. D.h. es gilt zu verifizieren ob die grafische Darstellung auf dem Display der Designspezifikation entspricht. Unstimmigkeiten weisen auf Fehler in der Implementierung der GUI hin. Während die Spezifikation das *Soll* klar definiert, kann das zu testende *Ist* in Form des Displayinhalts nur mittels Bildverarbeitungsmethoden gewonnen werden. Die Ergebnisse von optischen Bildverarbeitungsmethoden unterliegen allerdings naturgemäß Unsicherheiten (siehe Details in 2.4). Eine Abweichung von der Spezifikation muss daher nicht zwangsläufig ein Fehler sein, sondern kann sich auch auf unsicheren Bildinformationen begründen. Das vorgestellte Verifikationsverfahren berücksichtigt diese Unsicherheiten geeignet mittels eines Bayes'schen Netzwerks.
Im folgenden Kapitel wird der Anwendungsbereich des Verifikationsverfahrens mit Bezug auf themenverwandte Arbeiten genauer ausgeführt. Das Gebiet des modellbasierten Testens von HMI wird kurz vorgestellt und es wird darauf eingegangen, wie Bildverarbeitungsmethoden bereits aktuell bei der Testautomatisierung von MMI zum Einsatz kommen. Anhand konkreter Beispiele wird die GUI des MMI beschrieben und verdeutlicht, welche Fehler das Verifikationsverfahren identifizieren soll. Ansätze um aus unsicheren (Bild-)Informationen Schlussfolgerungen zu ziehen werden diskutiert. In Kapitel 3 werden die Anforderungen an das Verfahren als Kern eines Verifikationssystems konkretisiert und die Ziele definiert. Das Verifikationsverfahren und der Aufbau des Verifikationssystems werden in Kapitel 4 ausführlich erläutert. An konkreten Beispielen wird die Leistungsfähigkeit des Verfahrens untersucht. Das Kapitel 5 bildet mit einer Bewertung der Ergebnisse, die im laufenden Projekt bereits entstanden sind, und zukünftig geplanten weiterführenden Themen den Abschluss.

2 Problemstellung und Terminologie

2.1 Modellbasiertes Testen von HMI

Das modellbasierte Testen ist bei Systemtests eine gängige und geeignete Methode um Tests zu automatisieren. Ein Testmodell bildet in Form von Zustandsdiagrammen das Systemverhalten, oft bzgl. Benutzerinteraktionen, ab. In [BK08] wird eine gute Einführung in das modellbasierte Testen im Automobilbereich gegeben. [HMA10] befasst sich speziell mit Infotainmentsystemen. Neben der Testdurchführung ist ein weiteres Ziel beim modellbasierten Testen die automatisierte oder zumindest teilautomatisierte Erstellung der Testmodelle selbst. Auch hier gibt es zahlreiche aktuelle Forschungsprojekte, z.B. [DHH10]. Für die automatisierte Erstellung von Testmodellen ist die Existenz einer formalen, maschinenlesbaren Spezifikation, die das HMI-Verhalten beschreibt, essentiell. In [Fle07] wird mit Hinblick auf die Testmodellerstellung eine Spezifikation zur Beschreibung des HMI von Infotainmentsystemen vorgestellt.

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4
weitere Artikel online:
http://informatik2011.de/519.html

2.2 Bildverarbeitung in der MMI-Testautomatisierung

Im Vergleich zur umfangreichen Literatur über modellbasiertes Testen von HMI, lassen sich nur vereinzelt Arbeiten wie bspw. [CYM10] finden, die sich mit der Verwendung von Bildverarbeitungsmethoden zum Testen der GUI befassen. Eine GUI ist meist ein wesentlicher Teil eines HMIs und Bildverarbeitungsmethoden dienen beim automatisierten Testen als Ersatz des menschlichen Auges. Bei dem zu testenden Objekt handelt es sich um einen Screenshot, eine Momentaufnahme des Displayinhalts vorliegend als Bilddatei. Ein Screenshot wird entweder durch eine Kamera oder durch das direkte Abgreifen des digitalen Bildsignals mittels eines Framegrabbers erzeugt. Ein durch einen Framegrabber aufgenommenes Bild hat gegenüber einem Kamerabild den Vorteil, das es größtenteils störungsfrei ist. Während bei einer Kameraaufnahme präzise auf eine geeignete Ausrichtung und die Vermeidung von Störquellen wie einfallendes Licht geachtet werden muss, kann durch einen Framegrabber das originale Bildsignal empfangen werden. Hingegen stellt eine Kamera umfangreiche Möglichkeiten bereit den Bildkontrast positiv für die spätere Auswertung zu beeinflussen, was ein Framegrabber nicht ermöglicht. Unabhängig von der Aufnahmetechnik kann eine geringe Bildauflösung zu Problemen bei der Auswertung führen. Für das MMI existiert die Möglichkeit störungsfreie Screenshots mittels eines Framegrabbers zu erstellen. Je nach MMI-Generation ist die Auflösung unterschiedlich, bspw. 400x240 (Basic) oder 800x480 (High) Pixel.
Die Erkennung der Bildinhalte dient aktuell dem Zweck festzustellen in welchem Menü sich das System befindet um Schlussfolgerungen auf den Systemzustand zu ziehen und mit dem erwarteten Zustand zu vergleichen. Das erforderliche Wissen über die erwarteten Bildinhalte ist mangels einer zentral vorliegenden Wissensbasis in jedem Test hartkodiert. Das daraus resultierende Problem ist offensichtlich: Die Inhalte und Darstellung der GUI variieren je nach konkretem MMI und können sich zudem im laufenden Entwicklungsprozess einer Systemgeneration ändern. Die Änderungen müssen dann an vielen Stellen aktualisiert werden um wieder korrekte Tests durchführen zu können. Die Merkmale einer MMI-GUI sollten daher formal und maschinenlesbar spezifiziert und für jeden Test zugänglich sein. In der Fachliteratur gibt es vereinzelt Ansätze hierfür. Die verwendeten Formate basieren dabei meist auf XML, z.B. [usiXML], und Erweiterungen davon. Auch im Automobilbereich wurden verschiedene Versuche unternommen eine HMI-Spezifikationssprache zu entwickeln, die auch die grafischen Merkmale einer GUI beschreiben kann. Ein produktiver Einsatz derartiger Formate für GUI-Tests ist jedoch nicht bekannt. Ein Grund für das Scheitern der Ansätze kann sein, dass die Formate oft noch zu fern von ihrem späteren Anwendungsbereich und den dort bereits etablierten Werkzeugen erstellt worden sind, selbstverständlich mit der Forderung möglichst universell einsetzbar zu sein.

2.3 Das grafische Design der MMI-GUI

In Abbildung 1 sind Screenshots der GUI eines MMI der 3. Generation High und der 3. Generation+ High abgebildet. Daneben ist das allgemeine Layout dargestellt, bestehend aus

ISBN 978-3-88579-286-4

Abbildung 1: Die GUI von Wizardmenüs im *Media*- und *Car*-Kontext des deutschen MMI 3G+ High (links), eine *Media*-Liste im chinesischem MMI 3G High und das allgemeingültige Layout (rechts).

den Bereichen *Title*, *Subtitle*, *Softkey*, *Main* und *Status*. Die vorherrschenden Farben sind zudem abhängig vom Kontext, bspw. ist das *Media*-Menü in Orange- und das *Car*-Menü in Rottönen. Auch die Komplexität der Funktionalität des MMI ist in der GUI sichtbar. Die Informationen selbst sind in verschiedenen Menüstrukturen dargestellt, z.B. in Listen oder Wizards. Neben verschiedenen Generationen und Versionen gibt es Länder- und Sprachvarianten, die auch Unterschiede in der GUI aufweisen. Im Rahmen dieses Artikels sind die grafischen Inhalte wie folgt definiert: Elemente mit textuellen Inhalten werden im Folgenden als Textelemente bezeichnet (z.B. eine Textzeile in einem Listenmenü) und Elemente mit symbolischen Inhalten als Symbolelemente (z.B. Icons, Cursor, Schaltflächen, Fortschrittsbalken).[2] Einige Elemente nehmen als Teil von GUI-Steuerelementen verschiedene Zustände an und repräsentieren sie visuell. Bspw. ist beim MMI 3G+ ein sogenannter Softkey (eine Schaltfläche) im aktiven Zustand mit einem weißen Text beschriftet und im inaktiven Zustand mit einem grauen. Die Farbe der Softkey-Fläche ist ebenfalls zustandsabhängig. In Abbildung 2 ist eine Auswahl designspezifischer Merkmale anhand eines konkreten Screenshots veranschaulicht. Lokale, d.h. ein Element betreffende Merkmale wie *Position* und *Farbe* sollen von dem zu entwickelnden Verfahren verifiziert werden, sowie globale Merkmale wie der *Abstand* zwischen Elementen und *Abhängigkeiten* bzgl. der Elementzustände und der Existenz. Wenn z.B. die Softkey-Fläche eine Farbe besitzt, die auf den Zustand „selektiert“ hinweist, muss auch der Softkey-Text eine für diesen Zustand entsprechende Farbe besitzen. Existenzabhängigkeiten weisen z.B. die Elemente des Screenbereichs *Subtitle* auf. Hier ist es sowohl korrekt, dass dieser Bereich zwei Textelemente getrennt durch ein Symbolelement beinhaltet (siehe Abbildung 2), als auch nur ein weißes Textelement. D.h. das Nichtvorhandensein des Symbolelements ist nur im ersten Fall ein Fehler und ist abhängig von der Existenz der Textelemente.

[2] Ein GUI-Steuerelement kann nach dieser Definition aus mehreren Elementen bestehen, z.B. eine Schaltfläche aus der symbolischen Fläche und dem textuellen Schaltflächentext.

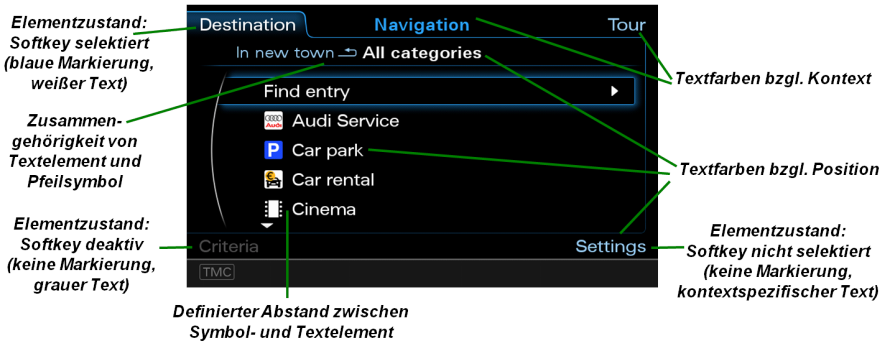

Abbildung 2: Beispiele für designspezifische Merkmale einer GUI.

Im Folgenden wird mit dem Begriff „Design“ das grafische Design und mit „Element“ ein grafisches Element einer MMI-GUI bezeichnet. Mit „Screenshot“ ist stets ein MMI-Screenshot gemeint.

2.4 Unsicherheit von Bildinformationen

Ein wichtiger Punkt ist die geeignete Verarbeitung der durch Bildverarbeitungsmethoden gewonnenen Bildinformationen. Allerdings sind Ergebnisse von Bildverarbeitungsmethoden naturgemäß mit Unsicherheiten behaftet. Z.B. besteht ein Screenshot in der Regel aus sehr vielen Farben. Selbst wenn für den Betrachter die feinen Farbabstufungen kaum erkennbar und im Fall von Kantenglättung (Anti-Aliasing) sogar erwünscht sind, sind sie für ein Computersystem sehr problematisch. Bei Bildern die mittels einer Kamera erzeugt wurden ist auf korrekte Farbwerte, so wie sie in der Designspezifikation definiert wurden, noch weniger Verlass. Aus diesem Grund müssen Systeme, die optische Informationen auswerten in einem gewissen Maße tolerant gegenüber diesen Schwankungen sein. Das jedoch führt zwangsläufig zu unsicheren Ergebnissen. Im aktuellen MMI-Testprozess wird eine harte Entscheidung basierend auf den unsicheren Ergebnissen vorgenommen indem ein Schwellwert eingesetzt wird. Z.B. kann ein Symbol als erkannt gewertet werden, wenn der Symbolerkenner eine Sicherheit von mindestens 80% angibt. Ein Verfahren, das die Umsetzung der Designspezifikation verifiziert, könnte ebenso vorgehen. Sinnvoller ist es jedoch die unsicheren Ergebnisse zu beachten und beim Vergleich mit dem *Soll* geeignet einzubeziehen.
Das Schlussfolgern mit Unsicherheit spielt in verschiedensten Bereichen eine Rolle. Vor allem im medizinischen Umfeld beschäftigen sich seit einigen Jahrzehnten Forschungsgruppen mit dieser Thematik um u. a. Diagnosesysteme zu entwickeln. Krankheiten lassen sich anhand von Symptomen diagnostizieren und aus einer Vielzahl beobachteter Krankheitsfälle ergeben sich Statistiken über den kausalen Zusammenhang zwischen einer Krankheit und ihren Symptomen. Im Fall einer bestimmten Krankheit ist das Vorhanden-

sein der damit verbundenen Symptome jedoch nicht immer zwingend, d.h. ihr Auftreten nicht 100% sicher. Daneben kann die Feststellung eines Symptoms mit Unsicherheit behaftet sein, z.B. kann ein Test in einem gewissen Maße falsch-positive oder falsch-negative Ergebnisse liefern. In den 70er Jahren wurde erstmals der Begriff „Certainty Factors“ eingeführt und eine Möglichkeit geschaffen unsichere Informationen mit Sicherheit zu gewichten, mehrere unsichere Informationen zu kombinieren und auf diese Weise Schlussfolgerungen zu ziehen. Das Diagnosesystem MYCIN ist eine sehr bekannte praktische Anwendung die auf diesem Ansatz basiert [BS84]. Diese heuristische Methode wird jedoch immer mehr durch wahrscheinlichkeitsbasierte Lösungen ersetzt. Bereits im 18. Jahrhundert entwickelte der Mathematiker Thomas Bayes das Bayestheorem, das beschreibt, wie mit konditionalen Wahrscheinlichkeiten gerechnet werden kann. Dieser Formalismus kann grafisch als ein azyklischer Graph bestehend aus Knoten und Kanten dargestellt werden und wird als Bayes'sches Netzwerk bezeichnet. Die Knoten repräsentieren Zufallsvariablen, die verschiedene Zustände annehmen können. Die Kanten beschreiben zusammen mit Übergangswahrscheinlichkeiten die konditionalen Abhängigkeiten zwischen den Variablen. Wenn die Wahrscheinlichkeiten für die Zustände einiger Variablen, den Elternknoten, und die Übergangswahrscheinlichkeiten gegeben sind, können die Zustandswahrscheinlichkeiten von konditional abhängigen Variablen, den Kindknoten, berechnet werden. In Kapitel 4.2 werden die Inferenzberechnungen anhand eines Beispiels ausführlicher erläutert. Eine gute Einführung in Bayes'sche Netzwerke gibt zudem das Buch von F. Jensen [JN07].
Die Anwendung von Bayes'schen Netzwerken ist im Bereich der medizinischen Diagnose und für Klassifikationsaufgaben verschiedenster Art weit verbreitet. Eine umfangreiche Zusammenfassung über die Einsatzgebiete findet sich in [Hol08]. Auch für die Entwicklung lernfähiger und dynamischer Systeme bieten sich Bayes'sche Netzwerke an ([KN04] Teil II und Kapitel 4.5). Im Umfeld der Testautomatisierung ist die Anwendung von Bayes'schen Netzwerken noch ein weitestgehend unerforschtes Gebiet.

3 Motivation und Zielsetzung

Wie im vorangegangenen Kapitel erläutert beschränkt sich die Analyse von Screenshots bisher auf Einzeltests deren eigentliches Ziel nicht die Prüfung des Designs ist, sondern das Erkennen des Systemzustands (vgl. 2.2). Lediglich fehlerhafte Ergebnisse der Bildverarbeitung können Hinweise auf ein fehlerhaftes Design sein. Eine korrekte Umsetzung des spezifizierten GUI-Designs ist bedeutsam für eine intuitive Bedienung des MMI und Designfehler können zu einer erheblichen Ablenkung des Fahrers führen. Daher ist eine Erweiterung der bisherigen MMI-Testautomatisierung erstrebenswert um das Design im vollständigen Umfang auf Korrektheit zu prüfen. Die Anforderungen an das zu entwickelnde Verifikationssystem im Einzelnen sind wie folgt:

1. Das Verfahren soll die unsicheren und evt. auch unvollständigen Ergebnisse von Bildverarbeitungsmethoden berücksichtigen und dadurch Abweichungen von der Spezifikation als Unsicherheit oder echten Fehler werten.

2. Um alle in Kapitel 2.3 genannten Fehler und Zustände identifizieren zu können, muss die zugrunde liegende Wissensbasis diese Informationen beinhalten.
3. Im Hinblick auf die steigende Komplexität von Infotainmentsystemen ist ein automatisiertes Verifikationsverfahren mit Zugriff auf eine maschinenlesbare Designspezifikation erforderlich.
4. Das Spezifikationsformat und das Verfahren sollen für verschiedene MMI-Generationen mit unterschiedlichem Design eingesetzt werden können.
5. Für den produktiven Einsatz spielen die Rechenzeit und der Ressourcenverbrauch eine entscheidende Rolle. Aktuelle MMI sind in ihrem GUI sehr komplex, das MMI 3G+ umfasst bspw. über 2000 verschiedene Screenshots, die getestet werden müssen. Für zukünftige MMI-Generationen ist ein weiteres Wachstum der Komplexität zu erwarten. Eine Optimierung bzgl. der Verifikationszeit und dem Resourcenverbrauch ist daher zwingend erforderlich. Das Verifikationssystem soll so konzipiert sein, dass Optimierungen realisierbar sind.

Die aufgeführten Anforderungen sollen bei der Realisierung des Verifikationsystems vollständig umgesetzt werden. Ziel ist ein System, das die gesamte Designverifikation basierend auf einer flexiblen Wissensbasis für verschiedenartige GUIs in angemessener Zeit durchführen kann.
Dieser Artikel befasst sich vorrangig mit dem Verifikationssystem als Ganzes und im Besonderen mit dem auf einem Bayes'schen Netzwerk basierenden Verifikationsverfahren.

4 Entwicklung des Verifikationssystems und Ergebnisse

4.1 Systemarchitektur - Komponenten und Interaktion

Das Verifikationssystem besitzt, wie in Abbildung 3 veranschaulicht, als Eingabemodule eine maschinenlesbare Form des *Soll*-Designs und die Informationen über den grafischen Inhalt des zu verifizierenden Screenshots (*Ist*-Design). Die Screenshotinhalte werden mittels Bildverarbeitungsmethoden nach Text- und Symbolelementen klassifiziert. Zu den klassifizierten Elementen können lokale Merkmale wie *Position* und *Farbe* ermittelt werden. Bei Vorhandensein eines Symbolerkenners kann zusätzlich für jedes Symbolelement das konkrete Symbol festgestellt werden. Alle Merkmale besitzen eine Wahrscheinlichkeitsangabe in Prozent, die sich aus den unsicheren Bildinformationen ergibt.
Im Systemkern werden die Daten aus den beiden Wissensquellen intern in Form eines Bayes'schen Netzwerks repräsentiert und ausgewertet (siehe 4.2 für Details). Das bedeutet im Wesentlichen die automatische Generierung des Bayes'schen Netzwerks. Einmal erzeugte Bayes'sche Netzwerke sollten gespeichert werden um ein immer wieder erneutes Erstellen durch das einfache Laden bereits existierender Netzwerke zu vermeiden und damit kürzere Ausführungszeiten zu gewährleisten. Das Ergebnis ist die Korrektheit jedes gefundenen Elements und die erkannten Elementzustände, beides jeweils mit Wahr-

scheinlichkeitsangabe. Die Fehlerhinweise können textuell oder auch visuell in Form von markierten Bereichen im Eingabe-Screenshot ausgegeben werden.

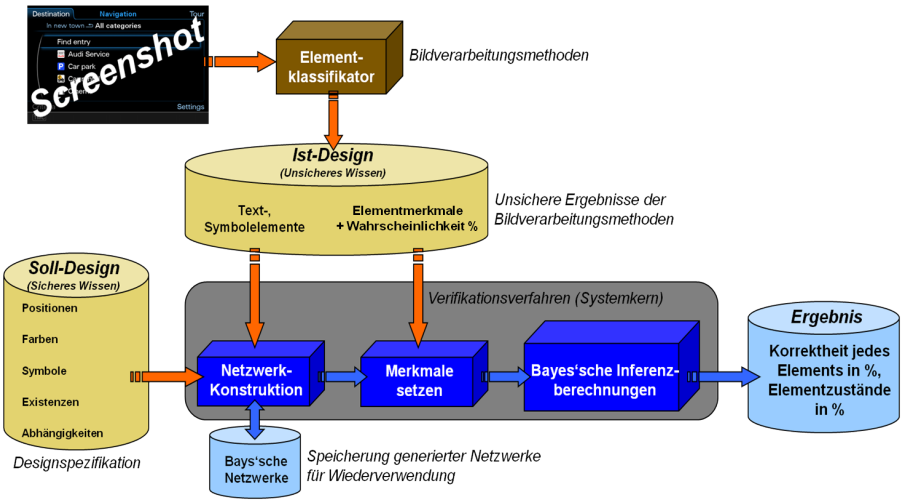

Abbildung 3: Komponenten des Verifikationssystems.

4.2 Systemkern - Designverifikation mit einem Bayes'schen Netwerk

Die unsicheren Informationen über das *Ist*-Design können als Wahrscheinlichkeiten betrachtet werden und so stellt ein Bayes'sches Netzwerk einen geeigneten Formalismus für den Vergleich mit dem sicheren *Soll*-Design dar. Das Bayes'sche Netzwerk repräsentiert hierfür das Wissen aus der Designspezifikation. In Abbildung 4 ist als Beispiel ein Bayes'sches Netzwerk angegeben[3]. Repräsentiert wird ein Teil des Designs des *Title*-Bereichs, das in dem linken oberen Screenshot von Abbildung 1 zu sehen ist. Er beinhaltet ein Text- und ein Symbolelement, die im Rahmen dieses Beispiels betrachteten *Soll*-Merkmale dieser Elemente sind wie folgt:

- Textelement: Existenz obligatorisch, minimale y-Koordinate der umschließenden Boundingbox 35 Pixel, Farbe (in HSV[4]) 40, 97, 99 (orange)
- globales Merkmal: Distanz zwischen Textelement und Symbolelement 10 Pixel

Im Netzwerk gibt es drei Arten von Knoten bzw. Variablen. Die schwarzen Knoten repräsentieren Merkmalsvariablen, die als Zustände alle Merkmalswerte beeinhalten, die

[3] Das Bayes'sche Netzwerk wurde mit der OpenSource-Software „JavaBayes" [Java] visualisiert.

[4] Im HSV-Farbraum sind Farben als Tripel von Farbton (Hue), Sättigung (Saturation) und Hellwert (Value) definiert mit Hue={0...180°}, Value={0%...100%} und Saturation={0%...100%}.

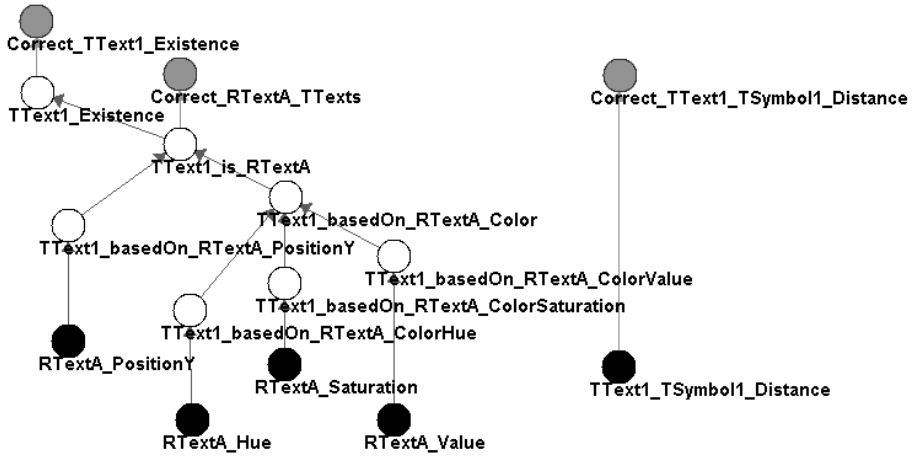

Abbildung 4: Beispiel für ein Bayes'sches Netzwerk zur Verifikation von Designspezifika des *Title*-Bereichs. Dargestellt ist lediglich die Zuweisung eines klassifizierten Textelements RTextA zum Zielelement TText1 und die Verifikation des Abstands zwischen den Zielelementen TText1 und TSymbol1.

gesetzt werden können. Die grauen Knoten sind beobachtbare Variablen, sie geben anhand der Wahrscheinlichkeitsverteilung über die Zustände „wahr", „falsch" und „unbekannt" Auskunft über die Korrektheit des jeweiligen Merkmals. Merkmalsvariablen und beobachtbare Variablen sind für eine bessere Strukturierung und um aus Gründen der Rechenzeit möglichst kleine Wahrscheinlichkeitstabellen zu erhalten nicht direkt, sondern über die in weiß dargestellten Zwischenvariablen miteinander verknüpft. Das Präfix „T" vor dem Elementtyp in den Variablenbezeichnungen steht für „Target" und bezeichnet ein Zielelement aus der Designspezifikation. Das Präfix „R" steht für „Recognized" und bezeichnet ein klassifiziertes Element. Um das *Ist*-Design mit der Designspezifikation zu

Text1_basedOn_RTextA_PositionY \ RTextA_PositionY	33	34	35	36	37	anders	U
W	0.7	0.95	1	0.95	0.7	0	0
F	0.3	0.05	0	0.05	0.3	1	0
U	0	0	0	0	0	0	1

Übergangswahrscheinlichkeiten zur Beschreibung der konditionalen Abhängigkeit der Zuweisung eines erkannten Textelements RTextA zur Zieleinheit TText1 von dem Merkmal y-Position.

TText1_basedOn_RTextA_PositionY	W			F			U		
TText1_is_RTextA \ TText1_basedOn_RTextA_Color	W	F	U	W	F	U	W	F	U
W	1	0	0.5	0	0	0	0.5	0	0
F	0	1	0	1	1	1	0	1	0
U	0	0	0.5	0	0	0	0.5	0	1

Übergangswahrscheinlichkeiten für die Zuweisung zur Zieleinheit, die konditional abhängig von den Teilzuweisungen bzgl. den Merkmalen y-Position und Farbe ist.

Tabelle 1: Übergangswahrscheinlichkeiten für Variablen des Bayes'schen Netzwerks zur Verifikation des *Title*-Bereichs (Zustände: W=wahr, F=falsch, U=unbekannt).

vergleichen, muss als erstes eine Zuweisung zwischen den klassifizierten und den spezifizierten Elementen vorgenommen werden. Das geschieht anhand der lokalen Elementmerkmale, deren Werte als Zustände in den Merkmalsvariablen gesetzt werden können. Im

Beispiel wird über die Variable $Correct_RTextA_TTexts$ geprüft, ob das klassifizierte Textelement A dem Zieltext 1 zugewiesen werden kann. Die Übergangswahrscheinlichkeiten für zwei an dieser Zuweisung beteiligten Variablen sind in Tabelle 1 aufgeführt. Der Zustand „unbekannt“ ist für jede Variable definiert und erforderlich, da aufgrund von fehlenden Bildverarbeitungsmethoden nicht zwangsläufig alle Merkmale gegeben sind. Das soll aber nicht zu falschen Verifikationsergebnissen führen, sondern lediglich zu einem Hinweis, dass dieser Teil aufgrund eines unbekannten Merkmals nicht verifiziert werden kann.
Anschließend wird die Existenz der gefundenen Zielelemente verifiziert. Für eindeutig zugewiesene Zielelemente können globale Merkmale verifiziert werden, wie bspw. mit den Variablen $TText1_TSymbol1_Distance$ und $Correct_TText1_TSymbol1_Distance$ der *Abstand* zwischen Zieltext 1 und Zielsymbol 1. Die Zustände der Merkmalsvariablen und die Übergangswahrscheinlichkeiten für die Zuweisung bzw. Verifikation sind geeignet zu definieren. Unsicherheiten können dann als Wahrscheinlichkeiten exakt in den jeweiligen Merkmalsvariablen gesetzt und in den Inferenzberechnungen verarbeitet werden. Im Beispiel ist die *y-Position* des Textelements mit 35 Pixeln fest definiert. Die Übergangswahrscheinlichkeiten zu der Variablen $TText1_basedOn_RTextA_PositionY$ in Abbildung 1 zeigen wie das Bayes'sche Netzwerk in der Lage ist Schwankungen der Merkmalswerte, die durch unsichere Bildinformationen verusacht sein können, geeignet zu behandeln. Wenn die *y-Position* 35 Pixel beträgt, entspricht das zu 100% dem Zielelement. Bei einer Abweichung von einem Pixel sind es noch 95% und bei zwei Pixeln 70%, größere Abweichungen sind unter dem Zustand „anders“ zusammengefasst und führen zu 0% Übereinstimmung. D.h. durch die Definition der Übergangswahrscheinlichkeiten kann festgelegt werden, wie stark Abweichungen vom *Soll* gewichtet werden. Auf die gleiche Weise werden die übrigen Merkmale für die Zuweisung und die Verifikation ausgewertet.
Wenn für ein Bayes'sches Netzwerk bestehend aus den Variablen $A_1, ..., A_n$ die Übergangswahrscheinlichkeiten $P(A_i|Pa_{A_i}), i = [1, ..., n]$ gegeben sind, können für jede Variable A_i die Zustandswahrscheinlichkeiten durch Marginalisierung bzw. Variableneliminierung mit Formel 1 berechnet werden, Pa_{A_i} bezeichnet die Elternknoten von A_i (siehe auch [JN07] Kapitel 2.3.2 bis 2.3.4).

$$P(A_1, ..., A_n) = \prod_{i=1}^{n} P(A_i|Pa_{A_i}) => P(A_i) = \sum_{A_j, j \neq i} \prod_{i=1}^{n} P(A_i|Pa_{A_i}) \tag{1}$$

Bspw. ist die Zuweisung zwischen erkanntem Textelement und Zielelement konditional von den Merkmalen *y-Position* und *Farbe* abhängig. Gegeben die Zustandswahrscheinlichkeiten der Elternknoten $A = TText1_basedOn_RTextA_PositionY$ und $B = TText1_basedOn_RTextA_Color$, sowie die Übergangswahrscheinlichkeiten (siehe rechts in Tabelle 1) ergibt sich unter Anwendung der Marginalisierung die Formel 2 zur Berechnung der Zustandswahrscheinlichkeiten der Variablen $C = TText1_is_RTextA$.

$$\begin{aligned} P(A, B, C) &= P(C|A, B)P(A)P(B) \\ => P(C) &= \sum_A \sum_B P(C|A, B)P(A)P(B) \end{aligned} \tag{2}$$

$P(A)$ und $P(B)$ bezeichnen die, basierend auf den Merkmalen gesetzten, Zustandswahrscheinlichkeiten der jeweiligen Variablen und $P(C|A,B)$ die definierten Übergangswahrscheinlichkeiten; summiert wird über alle Zustandskombinationen.
Zur Verdeutlichung des Umgangs mit Unsicherheiten ist in Tabelle 2 aufgelistet, wie sich die Sicherheiten für die Zuweisung zwischen einem klassifizierten Element und einem Zielelement ergeben. In der Zeile „Merkmale“ ist angegeben, inwieweit der jeweilige Merkmalswert bzgl. der verwendeten Übergangswahrscheinlichkeiten dem *Soll* entspricht. Die Klassifikatorsicherheit besagt, wie sicher das gefundene Element als Text- bzw. Symbolelement klassifizieren werden konnte und bestimmt die Übergangswahrscheinlichkeiten für Variable $TText1_is_RTextA$. D.h. bspw. bei einer Klassifikatorsicherheit von 95%, dass die Zuweisung selbst auch nur noch maximal 95% sicher sein kann. Anhand der Wahrscheinlichkeitstabellen lässt sich also der Einfluss der *Soll*-Abweichung auf das Verifikationsergebnis steuern. Je nach Genauigkeit der jeweiligen Bildverarbeitungsmethode kann der Toleranzbereich gewählt werden in dem eine *Soll*-Abweichung noch zu einem relativ geringem Maße als Fehler gewertet wird. Für die Verifikation der MMI-Screenshots kann eine Sicherheit von mindestens 80% gefordert werden, da hier die Bildinformationen in der Regel nicht so große Unsicherheiten aufweisen und eine Abweichung vom Soll daher eher aufgrund eines echten Fehlers auftritt. Natürlich ist es möglich, dass weniger

Merkmale	Farbe 100%, y-Position 100%			Farbe 100%, y-Position 90%			Farbe 100%, y-Position 70%			Farbe 70%, y-Position 70%		
Klassifikator-sicherheit	100%	95%	80%	100%	95%	80%	100%	95%	80%	100%	95%	80%
Zuweisung	100%	95%	80%	90%	90%	72%	70%	67%	65%	49%	47%	39%

Tabelle 2: Auswirkung der unsicheren Bildinformationen auf die Zuweisungssicherheit zwischen einem klassifizierten Element und einem Zielelement.

oder mehr Elemente klassifiziert werden, als in der Spezifikation vorgesehen sind. Bspw. könnten zwei Symbolelemente neben dem Textelement stehen oder das Textelement fehlen. Da das nicht der Spezifikation entspricht, wäre das ein zu identifizierender Fehler. D.h. das Bayes'sche Netzwerk besteht genau genommen aus einer Grundstruktur, dass aus der Designspezifikation erstellt wird und aus einer dynamischen Erweiterung, die sich aus der Designspezifikation und den klassifizierten Elementen ergibt. Je nach Anzahl der klassifizierten Elemente werden die Teilnetze mit den Elternknoten $TText\alpha_is_RText\Theta$ und $TSymbol\beta_is_RSymbol\Lambda$ ($\alpha, \beta \in N$ und $\Theta, \Lambda \in \{A, B, ...\}$) der Grundstruktur hinzugefügt oder weggelassen, die Variablen zur Verifikation der Existenz bilden jeweils die Verbindungsstellen.
Wie in den Screenshot-Beispielen in Abbildung 1 ersichtlich, sind einige Elementmerkmale kontextspezifisch, z.B. die Farbe mancher Textelemente. Neben der Existenz muss zudem auch geprüft werden, dass jedes klassifizierte Element genau einem Zielelement zugewiesen werden konnte. Beides kann ebenfalls im Bayes'schen Netzwerk als konditionale Wahrscheinlichkeit repräsentiert werden.
Anhand eines fehlerhaften Screenshots wird im Folgenden die Verifikationsleistung des entwickelten Verfahrens untersucht. Es wurde ein Bayes'sches Netzwerk erstellt, das das

Design des in Abbildung 5 links dargestellten Wizardmenüs beschreibt, folgende nicht direkt im Screenshot erkennbare Designspezifika sind ebenfalls enthalten:

- Textelement, Textbox und selektiertes Icon treten gemeinsam auf
- Textelement befindet sich zentriert in Textbox
- Benachbarte Icons haben definierte Abstände zueinander
- Maximal ein selektiertes Icon (visuell dargestellt durch orange Farbe, siehe Icon „Höhen“ in Wizardmenü links in Abbildung 5)
- Textbox befindet sich in einem definierten Abstand über selektiertem Icon

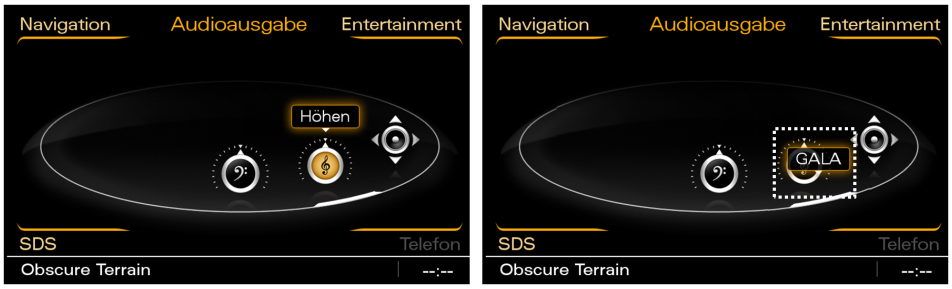

Abbildung 5: Links ein korrektes und rechts ein fehlerhaftes Wizardmenü, der Fehlerbereich ist mit einem gestrichelten Rahmen markiert.

Der in Abbildung 5 rechts dargestellte Wizard desselben Menüs weist hingegen Fehler auf. Das identifizierte *Ist*-Design besteht aus einem Textelement und fünf Symbolelementen (Wizardring, Cursor, Textbox, zwei nicht selektierte Icons). Das mittlere Icon konnte aufgrund der Verdeckung durch das Textelement vom Klassifikator nicht identifiziert werden und existiert demnach nicht für die Verifikation. In Tabelle 3 sind die klassifizierten Elemente, ihre Klassifikatorsicherheit und ihre Merkmale mit der prozentualen *Soll*-Abweichung aufgelistet. Durch Inferenzberechnungen ergeben sich die in Tabelle 4 aufgelisteten Zustandsverteilungen für die beobachtbaren Variablen, die die Zuweisung zwischen einem klassifizierten Element und einem Zielelement prüfen. An dieser Stelle kann bereits durch das Fehlschlagen der Zuweisung eines klassifizierten Elements zu Zielelement Icon 2 der Fehler abgeleitet werden, dass Icon 2 fehlt. Durch weitere Inferenzberechnungen kann festgestellt werden, dass Textelement und Textbox ohne ein selektiertes Icon existieren. Die gefundenen Designfehler entsprechen damit genau den tatsächlichen Fehlern. Des Weiteren kann, da Icon 2 fehlt, der Abstand zwischen Icon 1 und 2, sowie 2 und 3 nicht verifiziert werden. Für diese globalen Merkmale hat daher der Zustand „unbekannt“ 100%-ige Wahrscheinlichkeit und durch Inferenz ergibt sich das auch für die dazugehörigen Verifikationsvariablen. Dasselbe gilt für den Abstand zwischen Textbox und selektiertem Icon.

ISBN 978-3-88579-286-4

Klassifiziertes Element	Klassifikator-sicherheit	Merkmalsentsprechung zum Soll
Text A	99%	Farbe: Hue 99%, Saturation 99%, Value 100% => HSV-Farbwerte insgesamt 98%
Symbol A	Wizardring 99%	Mittelposition (x,y) 100%
Symbol B	Cursor 96%	-
Symbol C	Textbox 97%	-
Symbol D	nicht-selektiertes Icon 99%	Mittelposition (x,y) 100%
Symbol E	nicht-selektiertes Icon 99%	Mittelposition (x,y) 100%

Tabelle 3: Wahrscheinlichkeiten für klassifizierte Elemente im Wizardmenü und deren Merkmale bzgl. der Entsprechung zum *Soll*.

Zuweisung	Zustandswahrscheinlichkeiten
Text A -> Zieltextelement	98% wahr, 2% falsch
Symbol A -> Zielsymbol Wizardring	99% wahr, 1% falsch
Symbol B -> Zielsymbol Cursor	96% wahr, 4% falsch
Symbol C -> Zielsymbol Textbox	97% wahr, 3% falsch
Symbol D -> Zielsymbol Icon 1, nicht-selektiert	99% wahr, 1% falsch
? -> Zielsymbol Icon 2	100 % falsch
Symbol E -> Zielsymbol Icon 3, nicht-selektiert	99% wahr, 1% falsch

Tabelle 4: Wahrscheinlichkeiten der Variablenzustände für die Zuweisung zwischen klassifiziertem Element und Zielelement.

5 Schlussfolgerung und Ausblick

Wie die Ergebnisse im vorangegangen Kapitel zeigen, kann mittels eines Bayes'schen Netzwerks GUI-Design verifiziert werden. Neben lokalen Merkmalen sind auch Abhängigkeiten zwischen den grafischen Elementen verifizierbar. Die in diesem Artikel aufgeführten Beispiele beziehen sich jeweils nur auf einen bestimmten Inhalt eines Screenbereichs (vgl. 2.3) und das Bayes'sche Netzwerk repräsentiert nur diesen. Deshalb können offensichtlich nur Abhängigkeiten zwischen Elementen verifiziert werden, die im selben Screenbereich liegen. Diese Einschränkung ist lediglich für eine einfache Demonstration gewählt worden, das Verfahren ist ebenso in der Lage Abhängigkeiten zwischen verschiedenen Screenbereichen zu verifizieren. Trotz des komplexen Leistungsumfangs stellt ein Bayes'sches Netzwerk mit der Möglichkeit der grafischen Darstellung eine Lösung dar, dessen Struktur und Ergebnisse einen hohen Grad an Plausibiltät aufweisen und ermöglicht eine vollständige Abdeckung der geforderten Designtests. Es stellt zudem eine geeignete Verarbeitung und Kombination unsicherer Bildinformationen bereit, indem für jedes Merkmal individuell die mögliche *Soll*-Abweichung eingestellt werden kann. Im Rahmen des Artikels wurde die Leistungsfähigkeit des entwickelten Verifikationsverfahrens am Beispiel der GUI ei-

ISBN 978-3-88579-286-4

nes aktuellen MMI dargelegt. Das Verfahren soll ebenso für die Verifikation zukünftiger MMI eingesetzt werden können, als auch universell für GUI aus anderen Bereichen. Eine dahingehende Analyse des Potenzial ist vorgesehen, bei der auch der Trend zu dreidimensionalen GUI-Inhalten ein wichtiges Kriterium sein wird.
Da eine maschinenlesbare Spezifikation des GUI-Designs und der Klassifikator als Eingabemodule, sowie die automatische Generierung des Bayes'schen Netzwerks auf Basis der Eingabedaten (vgl. 4.1) erst in der Entstehung sind und aktuell nicht vorliegen, kann das Verifikationsverfahren bisher nur für ausgewählte Beispiele getestet werden, für die die Eingabedaten manuell bereitgestellt wurden. Erst wenn die Eingabemodule vorhanden sind, kann das Verfahren für ein gesamtes MMI durchgeführt werden und umfangreiche qualitative und quantitative Ergebnisse liefern. Die Entwicklung einer maschinenlesbaren Designspezifikation stellt eine echte Herausforderung dar, weil hierfür Änderungen an dem seit Jahren existierenden und bewährten Designentwurfsprozess vorgenommen werden müssen. Aktuell ist das Design informell in mehreren verschiedenartigen Dateiformaten beschrieben. Manche Designinformationen sind zudem nur implizit oder unvollständig angegeben.
Im Folgenden werden zwei weiterführende Überlegungen formuliert um das entwickelte Verfahren zu erweitern und einen umfangreichen Einsatz in der MMI-Testautomatisierung zu gewährleisten.

5.1 Wissensgewinn durch Lernen

In Kapitel 4.1 wurde darauf hingewiesen, dass das Bayes'sche Netzwerk basierend auf den Eingabedaten und einer vorgegebenen Grundstruktur automatisch generiert werden soll. Auf diese Weise können allerdings nur die Informationen ins Bayes'sche Netzwerk gelangen, die in der formalen Spezifikation enthalten sind.
Für Bayes'sche Netzwerke existieren verschiedene Methoden um datenbasiert Struktur und Wahrscheinlichkeiten automatisch zu lernen bzw. ein existierendes Netzwerk durch Lernen zu erweitern. In [KN04], Teil II werden einige Algorithmen erläutert. Bei der Verifikation des GUI-Designs entsprechen die Daten den Inhalten von Screenshots[5]. Unter Verwendung der Software „BN PowerConstructor" [CBL97] wurden bereits Experimente durchgeführt. Weil einige Designinformationen wie bspw. Abhängigkeiten zwischen Elementen und Elementzuständen bislang nicht formal spezifiziert sind und es schwer ist, den aktuellen Prozess des Designentwurfs dahingehend zu verändern, kann Lernen eine gute Alternative sein um auch diese Designspezifika verifizieren zu können. Mit einem Grundnetzwerk, das die gegebene formale Spezifikation repäsentiert und Daten, die korrekte Abhängigkeiten zwischen Elementen beschreiben, konnten bereits gute Ergebnisse erzielt werden.

[5] Für das Lernen dürfen die Screenshots logischerweise keine Fehler aufweisen, andernfalls repräsentiert das gelernte Bayes'sche Netzwerk falsches Design.

ISBN 978-3-88579-286-4

5.2 Verifikationsverfahren bei störungsbehafteten Kamerabildern

Bisher wird das entwickelte Verifikationsverfahren auf störungsfreien Framegrabber-Bildern angewendet. Eine interessante Fragestellung ist, ob das Verfahren ebenfalls verwertbare Ergebnisse liefert, wenn die Merkmale der grafischen Elemente größere Abweichungen gegenüber dem *Soll* aufweisen. In diesem Fall müssen die Übergangswahrscheinlichkeiten toleranter gegenüber Abweichungen eingestellt werden. Für das Bayes'sche Netzwerk stellt das kein Problem dar, jedoch wird die Grenze zwischen Unsicherheit und echtem Fehler undeutlicher. Neben dem MMI gibt es im Audi auf dem Armaturenbrett ein weiteres zu testendes Display. Hier bietet die Hardware keine Framegrabber-Schnittstelle an und Screenshots können nur mittels einer Kamera erzeugt werden, was oftmals störungsbehaftet ist. Die Möglichkeiten des Verifikationsverfahrens sollen daher auch für diesen Anwendungsfall untersucht werden.

Literatur

[BK08] Eckard Bringmann, Andreas Kraemer *Model-Based Testing of Automotive Systems.* International Conference on Software Testing Verification and Validation, 485–493, 2008.

[BS84] Bruce G. Buchanan, Edward H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, 1984.

[CBL97] Jie Cheng, David Bell, Weiru Liu. *Learning Bayesian Networks from Data : An Efficient Approach Based on Information Theory.* (October):1–41, 1997.

[CYM10] Tsung-Hsiang Chang, Tom Yeh, Robert C. Miller. *GUI testing using computer vision.* Proceedings of the 28th international conference on Human factors in computing systems CHI 10, (Figure 1):1535–1544, 2010.

[DHH10] Linshu Duan, Alexander Hoefer, Heinrich Hussmann. *Model-Based Testing of Automotive HMIs based on a Test-Oriented HMI Specification Model*. IEEE, 2010.

[Fle07] Thomas Fleischmann. *Model Based HMI Specification in an Automotive Context.* Human Interface and the Management of Information, 4557:31–39, 2007.

[HMA10] Yingping Huang et al. *Development of an automated testing system for vehicle infotainment system.* The International Journal of Advanced Manufacturing Technology, Seiten 1–14, 2010.

[Hol08] Jain L. Holmes, Hrsg. *Innovations in Bayesian Networks - Theory and Applications. Series: Studies in Computational Intelligence*, Jgg. 156. Springer, 2008.

[Java] JavaBayes. `http://www.cs.cmu.edu/~javabayes`. (Stand 25.03.2011)

[JN07] Finn V. Jensen, Thomas D. Nielsen. *Bayesian networks and decision graphs*. Springer, 2007.

[KN04] Kevin B. Korb, Ann E. Nicholson. *Bayesian Artificial Intelligence*, Jgg. 7. Springer, 2004.

[usiXML] UsiXML. `http://www.usixml.org`. (Stand 30.03.2011)