ulm university universität ulm

**Fakultät für Ingenieurwissenschaften, Informatik und Psychologie**
Institut für Medieninformatik

# Abstrakte Interaktionsmodelle für die Integration in bestehende Benutzerschnittstellen

**Dissertation**

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

**Vorgelegt von:**
Dipl.-Inf. Mark Poguntke
aus Überlingen

**Gutachter:**
Prof. Dr.-Ing. Michael Weber
Prof. Dr. Dr.-Ing. Wolfgang Minker

**2016**

**Amtierende Dekanin:**
Prof. Dr. Tina Seufert

**Gutachter:**
Prof. Dr.-Ing. Michael Weber
Prof. Dr. Dr.-Ing. Wolfgang Minker

Tag der Promotion:
22. Februar 2016

# Abstract

The interaction with computer applications and data in everyday life has increased greatly due to the increasing availability of small, low-cost, and easy-to-use computers. Users have diverse individual preferences and different everyday routines that take place in different environments. While you can turn away from your tasks for a period of time when in the living room or office to interact with a mobile device, that interaction poses a safety risk within an automotive environment. The input and output devices that are potentially more suitable within the in-car environment are often not used due to missing integration possibilities. This reinforces the separation between the closed environment of the computer that is brought into the car and the actual interaction environment that is offered to the user from the vehicle. This is not only inconvenient and distracting, but also safety-critical.

For the work presented here, this is the starting point for further development of the vision of a customized use of computer applications in any context. Existing approaches use modeling to abstract from specific interactions, so that solutions can be derived for multiple environments. The resulting user interfaces, however, are often not satisfactory and only partially automatically generated. The presented approach combines the relevant model types domain, dialog and presentation model, into an abstract interaction model and introduces rules for the description of user interface elements of the system that is used for integration of the model. This provides the foundation for guaranteed, fully automated transformations created in a target system. The integration of exemplary applications into the automotive interaction space has been successfully demonstrated in a prototype implementation. The results were evaluated according to the requirements, and the possibilities and limitations of the approach were analyzed. The contributions of this work support closing the gap between application and interaction context, which is created when integrating unknown applications into existing user interfaces.

# Kurzfassung

Die Interaktion mit Computeranwendungen und Daten im Alltag hat durch die zunehmende Verfügbarkeit kleiner, kostengünstiger und leicht zu bedienender Computer stark zugenommen. Entsprechend vielfältig sind die Präferenzen eines jeden Nutzers. Auch sieht der Alltag jedes Nutzers anders aus und findet in unterschiedlichen Umgebungen statt. Während man sich im Wohnzimmer oder Büro für eine Zeit von seinen Aufgaben abwenden kann, um mit einem mobilen Gerät zu interagieren, stellt dies im Interaktionsraum Automobil eine Gefahr für die Sicherheit dar. Die für Interaktionsaufgaben potenziell besser geeigneten Elemente in der Zielumgebung Automobil werden aufgrund fehlender Integrationsmöglichkeiten häufig nicht genutzt. Dies verstärkt die Trennung zwischen der abgeschlossenen Umgebung des mit ins Auto gebrachten Computers und der eigentlichen Bedienumgebung, die dem Nutzer vom Fahrzeug angeboten wird. Dies ist nicht nur unbequem und ablenkend, sondern auch sicherheitskritisch.

Für die vorgestellte Arbeit liegt hier der Ausgangspunkt für die Weiterentwicklung der Vision einer angepassten Nutzung von Computeranwendungen in jedem Kontext. Bestehende Ansätze verwenden Modellierung, um von konkreten Interaktionen zu abstrahieren, sodass ausgehend davon Lösungen für mehrere Umgebungen abgeleitet werden können. Die resultierenden Benutzerschnittstellen sind allerdings häufig nicht zufriedenstellend und werden nur teilautomatisch erzeugt. Der vorgestellte Ansatz kombiniert die für das Problem relevanten Modellarten Domänen-, Dialog- und Präsentationsmodell in einem abstrakten Interaktionsmodell, und führt Regeln für die Beschreibung der Elemente eines Systems, in das integriert werden soll, ein. Dadurch wird die Basis für garantierte, vollautomatische Abbildungen in ein Zielsystem geschaffen. Exemplarische Anwendungsfälle im Interaktionsraum Automobil wurden in einer prototypischen Implementierung erfolgreich demonstriert. Die Ergebnisse wurden anhand der Anforderungen evaluiert und die Möglichkeiten und Grenzen des vorgestellten Ansatzes analysiert. Die Beiträge der Arbeit unterstützen das Schließen der Lücke zwischen Anwendung und Interaktionskontext, die bei der Integration unbekannter Anwendungen in existierende Benutzerschnittstellen besteht.

# Inhaltsverzeichnis


**1. Einleitung** **1**

1.1. Die überholte Vision des Ubiquitous Computing . . . . . 1
1.2. Problemstellung . . . . . 3
1.3. Ziele der Arbeit . . . . . 4
1.4. Einordnung der Arbeit . . . . . 5
1.5. Beiträge . . . . . 6
1.6. Kapitelüberblick . . . . . 7

**2. Grundlagen** **9**

2.1. Mensch-Computer Interaktion . . . . . 10
2.1.1. Historie und Begriffsklärung . . . . . 10
2.1.2. Teilgebiete . . . . . 11
2.2. Benutzerschnittstellen . . . . . 13
2.2.1. Grafisch-haptische Benutzerschnittstellen . . . . . 13
2.2.2. Haptische Ein- und Ausgabe . . . . . 15
2.2.3. Auditive Ausgabe und Sprachbedienung . . . . . 16
2.2.4. Weitere Formen von Benutzerschnittstellen . . . . . 19
2.2.5. Natürliche Interaktion über multimodale Benutzerschnittstellen . . 20
2.3. Anwendungsbeispiel: Interaktion im Automobil . . . . . 22
2.3.1. Historie von Benutzerschnittstellen im Automobil . . . . . 23
2.3.2. Interaktionsaufgaben im Fahrzeug . . . . . 24
2.3.3. Bedien- und Anzeigekonzepte für Infotainmentanwendungen . . . 24
2.4. Benutzerschnittstellenentwicklung . . . . . 32
2.4.1. Usability Engineering . . . . . 32
2.4.2. Die vier Säulen erfolgreicher Benutzerschnittstellenentwicklung . . 34
2.4.3. Methoden und Werkzeuge . . . . . 35

2.4.4. Benutzerschnittstellenentwicklung im Automobilbereich . . . . . 38
2.5. Fazit . . . . . 43

**3. Modellierung von Benutzerschnittstellen 45**
3.1. Überblick . . . . . 45
3.1.1. Modellbasierte Entwicklung: Kritik und Potenzial . . . . . 46
3.1.2. Modellgetriebene Entwicklung: Kritik und Potenzial . . . . . 47
3.2. Modellarten bei der Benutzerschnittstellentwicklung . . . . . 47
3.2.1. Aufgabenmodell . . . . . 48
3.2.2. Domänenmodell . . . . . 49
3.2.3. Dialogmodell . . . . . 50
3.2.4. Präsentationsmodell . . . . . 51
3.2.5. Kontextmodell . . . . . 52
3.3. CAMELEON Reference Framework . . . . . 52
3.4. Ansätze für die Modellierung . . . . . 54
3.4.1. Aufgabenmodellierung . . . . . 55
3.4.2. Abstrakte Benutzerschnittstellenmodellierung . . . . . 55
3.4.3. Spezifische Beschreibungssprachen . . . . . 56
3.4.4. Umfassende, durchgängige Modellierungsmethoden . . . . . 57
3.5. Fazit . . . . . 58

**4. Ein Konzept für Interaktionsmodelle zur Integration in bestehende Benutzerschnittstellen 61**
4.1. Anwendungsbeispiel: Integration in Benutzerschnittstellen im Automobil . 62
4.1.1. Integration unter Kontrolle der Fahrzeughersteller . . . . . 62
4.1.2. Integration über Schnittstellen zur Bildschirmübertragung . . . . . 64
4.1.3. Ansätze aus anderen Anwendungsbereichen . . . . . 64
4.1.4. Gewichtung des Spannungsfelds . . . . . 66
4.1.5. Zusammenfassung und Schlussfolgerungen für die Konzeptentstehung . . . . . 66
4.2. Konzeptentstehung . . . . . 68
4.3. Spezifikationsphase vs. Konzeptphase . . . . . 70
4.4. Ziele . . . . . 73
4.4.1. Ziel 1: Flexible abstrakte Beschreibung von Benutzerschnittstellen 74
4.4.2. Ziel 2: Automatische Weiterverarbeitung und Integration der abstrakten Beschreibung . . . . . 75

4.4.3. Ziel 3: Erweiterbarer, flexibler Entwicklungsprozess . . . . . 76
4.4.4. Ziel 4: Sicherstellung besonderer Anforderungen im Zielkontext . . 76
4.5. Anforderungen an die Ziele . . . . . 77
4.5.1. Anforderungen für die Modellierung . . . . . 77
4.5.2. Anforderungen für die Integration . . . . . 78
4.5.3. Anforderungen an einen Entwicklungsprozess . . . . . 79
4.6. Genereller Ansatz . . . . . 79
4.6.1. Relevante Modellarten . . . . . 79
4.6.2. Abbildungen zwischen Modellen - *Mapping-Problem* . . . . . 81
4.6.3. Kombination von Domänen-, Dialog- und Präsentationsmodell . . 82
4.6.4. Zwischenbewertung zur Erfüllbarkeit der Anforderungen . . . . . 83
4.7. *aDDaPt* - abstraktes Domänen-, Dialog- und Präsentationsmodell . . . . 85
4.7.1. Domänenmodell . . . . . 85
4.7.2. Dialogmodell . . . . . 87
4.7.3. Präsentationsmodell . . . . . 88
4.7.4. Beispiel: ToDo-Liste . . . . . 90
4.7.5. Beispiel: JoinMyRide . . . . . 91
4.7.6. Wiederverwendung von Modellen . . . . . 93
4.8. Integration von *aDDaPt* in bestehende Benutzerschnittstellen . . . . . 95
4.8.1. Überblick . . . . . 95
4.8.2. *UI rules:* Layout und Darstellung . . . . . 96
4.8.3. *UI rules:* Hierarchische Regeln für Transformationen . . . . . 98
4.8.4. UI-Generierung im Zielsystem . . . . . 99
4.9. Bewertung anhand der Anforderungen . . . . . 100
4.9.1. Modellierung . . . . . 100
4.9.2. Integration . . . . . 101
4.9.3. Entwicklungsprozess . . . . . 102
4.10. Fazit . . . . . 102

**5. Eine Architektur zur Integration externer Anwendungen in bestehende Benutzerschnittstellen 105**
5.1. Konzeption der Architektur . . . . . 106
5.2. Client-Server-Architektur . . . . . 106
5.2.1. Clients . . . . . 108
5.2.2. Server . . . . . 108

5.3. Kommunikation im Server . . . . . 109
5.3.1. Nachrichtenformat . . . . . 110
5.3.2. Verbinden von Clients . . . . . 113
5.4. Fazit . . . . . 114

**6. Umsetzung und Bewertung am Beispiel von Benutzerschnittstellen im Automobil 117**
6.1. Architektur für die Integration externer Anwendungen . . . . . 117
6.1.1. Überblick . . . . . 117
6.1.2. Ergebnis und Bewertung . . . . . 120
6.2. UI-Prototypen im Fahrzeug . . . . . 120
6.2.1. Überblick . . . . . 120
6.2.2. Ergebnis und Bewertung . . . . . 122
6.3. Integration von UI-Modellen verschiedener mobiler Plattformen . . . . . 122
6.3.1. Überblick . . . . . 122
6.3.2. Ergebnis und Bewertung . . . . . 125
6.4. Generische Transformationen . . . . . 125
6.4.1. Überblick . . . . . 125
6.4.2. Ergebnis und Bewertung . . . . . 126
6.5. Fazit . . . . . 128

**7. Ergebnisse 129**
7.1. *aDDaPt* für die flexible abstrakte Beschreibung von Benutzerschnittstellen 129
7.2. Automatische Weiterverarbeitung und Integration von *aDDaPt*-Modellen im Zielkonzept . . . . . 131
7.3. Flexible unabhängige Entwicklungsprozesse . . . . . 132
7.4. Erfüllung domänenspezifischer Anforderungen im Zielkontext . . . . . 133
7.5. Fazit . . . . . 134

**8. Diskussion 137**
8.1. Abdeckung möglicher Interaktionen . . . . . 137
8.2. Grenzen der Generierung . . . . . 140
8.3. Qualität resultierender Benutzerschnittstellen . . . . . 141
8.4. Flexibilität . . . . . 142
8.5. Anwendbarkeit . . . . . 143
8.6. Multimodale Benutzerschnittstellen . . . . . 143

8.7. Fazit . . . . . 146

**9. Zusammenfassung und Ausblick** **149**
9.1. Beiträge der Arbeit . . . . . 149
9.2. Schlussfolgerungen und Ausblick . . . . . 151

**A. Kommunikations-Architektur: Nachrichtenaustausch beim Verbindungsaufbau** **153**
A.1. Verbindung externer Provider mit Presentation-Client . . . . . 153
A.2. Verbindung von Eingabegeräten (Input-Clients) . . . . . 156

**B. aDDaPt: Semantische Transitionen** **159**

**Abbildungsverzeichnis und Bildnachweis** **163**

**Tabellenverzeichnis** **169**

**Literaturverzeichnis** **171**

**Danksagung** **182**

# 1 Einleitung

## 1.1. Die überholte Vision des Ubiquitous Computing

Computer, die sich in die Umgebung der Nutzer einpassen anstatt diese zu zwingen, in die Welt der Computer einzutauchen, stellen eine Kernidee in der Vision des *Ubiquitous Computing* von Mark Weiser dar [Wei91]. Weiser beschließt seinen vielzitierten Artikel aus dem Jahr 1991 mit einem bildhaften Vergleich. So soll in seiner Vision die Nutzung von Computern ähnlich erfrischend sein wie ein Waldspaziergang [Wei91].

*Ubiquitous Computing* ist eine Vision von Computern, die dem Anwender genau die Unterstützung im Alltag bieten, die er benötigt, indem sie sich in die Umgebung des Anwenders integrieren und dabei auf unnötige Komplexität verzichten. Über 20 Jahre nach der Prägung dieses Begriffs sind Computer, in Form von mobilen Geräten, so präsent wie nie zuvor: Smartphones, Tablets, Audio- und Videoplayer sind ein Teil des täglichen Lebens. Vernetzte Armbanduhren und intelligente Brillen sind die jüngeren Neuerungen auf diesem Markt tragbarer Computer. Anwendungen und Daten sind bereits jetzt mit Hilfe solcher Geräte ubiquitär verfügbar. Bei Betrachtung der Komplexität lässt sich außerdem der Trend einfacher Bedienbarkeit mit reduzierten klaren Gestaltungsmustern

in den Benutzerschnittstellen erkennen. Nicht zuletzt ist diese Reduktion der Komplexität ein Erfolgstreiber für Geräte wie Apple's[1] iPod und iPhone.

Der Einzug unzähliger, leicht zu bedienender Computer in den Alltag war keineswegs unbemerkt, und sie haben sich auch nicht im Hintergrund in den Alltag der Menschen integriert. Die Anwender haben ihren Alltag erweitert, teilweise stehen die tragbaren Begleiter sogar im Mittelpunkt. Der Wald des Alltags, in dem, wie in Mark Weiser's Zitat nahegelegt, trotz ubiquitärer Computer ein erfrischender Spaziergang möglich sein soll, ist ein Wald von Smartphones geworden. Eine minutenlang schweigende Gruppe junger Menschen im Restaurant, jeder Einzelne mit dem Smartphone beschäftigt, immer informiert, immer vernetzt - das ist Teil dieses Alltags. Die ursprüngliche Vision der allgegenwärtigen Computer ohne unnötige Komplexität scheint nur auf den ersten Blick erfüllt. Vielmehr wurde sie überholt - von gesellschaftsfähiger Technik für die Massen. Die Menschen tauchen freiwillig in diese Welt ein, in der Computer kleiner, günstiger und leistungsfähiger geworden sind als noch vor 20 Jahren.

Durch die technischen Fortschritte und zunehmende Verfügbarkeit von Computern und Internet hat sich das Nutzungsverhalten der Menschen verändert. In der 2011 veröffentlichten Studie *i-Car: Die junge Generation und das vernetzte Auto* [Bra11] wird deutlich, dass jungen Menschen im Alter von 18 bis 25 Jahren Handy und Internet deutlich wichtiger sind als das Automobil. Daraus kann man schließen, dass *individuelle Mobilität*, im ursprünglichen Sinne an die Verfügbarkeit eines Fahrzeugs geknüpft, von einer *digitalen Verfügbarkeit* von Informationen und Kontakten abgelöst wurde. Auch die Ergebnisse dieser Studie verdeutlichen, dass der Teil der immer verfügbaren Computeranwendungen und Daten in der Vision des *Ubiquitous Computing* nicht nur bereits erfüllt ist, sondern sich auch im Alltag der Menschen durchgesetzt hat. Doch für eine angenehme, nahtlose Integration von Computertechnologie in den Alltag besteht mehr Potenzial, als den kleinen Bildschirm eines Smartphones in die Aufmerksamkeit rücken zu müssen. Der Alltag findet auch im Wohnzimmer, im Auto oder im Büro statt. Im Wohnzimmer ist der Fernseher inzwischen Multimedia-Zentrale geworden, im Auto ist schon lange weit mehr als nur Fahren möglich, und das moderne Büro ist mit Wandprojektoren oder interaktiven Touchscreens ausgestattet. Für diese Arbeit liegt hier der Ausgangspunkt für die logische Weiterentwicklung der ursprünglichen Vision des Ubiquitous Computing: Die Umgebung soll sich je nach Situation anpassen und die nötigen Informationen und Interaktionsmöglichkeiten bereitstellen, so dass sie die entsprechende Situation perfekt

[1] Apple Inc., www.apple.com, aufgerufen am 27.10.2015

ergänzen und die Aufgaben des Nutzers immer und zu jeder Zeit sinnvoll unterstützen. Dass dies noch nicht möglich ist, liegt an technischen Hürden, die man als Entwickler und Anwender nicht für jede Umgebung neu überbrücken möchte und wegen vieler fehlender Schnittstellen auch nicht kann. Also wird häufig auf das vertraute Smartphone mit den dortigen Ein- und Ausgabemöglichkeiten zurückgegriffen. Sowohl aus technischer Sicht als auch aus Benutzersicht werden Lösungen benötigt, die eine flexible und einfache Nutzung der Umgebung zur Integration von Computertechnologie ermöglichen. Auf dem Weg zu einer Welt mit der Anwendungsvielfalt von Smartphones – aber ohne an diese gebunden zu sein.

## 1.2. Problemstellung

Um der Vision einer einfachen Nutzung einer Computeranwendung in jedem Kontext näher zu kommen, werden geeignete Schnittstellen zwischen jeder beliebigen Anwendung und den verschiedenen Nutzungskontexten benötigt. Ebenso muss eine Möglichkeit, in einen anderen, zuvor nicht bekannten Kontext zu wechseln, bereitgestellt werden. Dies ist die Grundlage, eine Anwendung entsprechend der Situation, in der sich der Anwender befindet, anzupassen und dadurch eine nahtlose Integration in die Interaktionsumgebung zu erzielen. Abbildung 1.1 illustriert diese Problemstellung. Die Verbindungen, durch Doppelpfeile dargestellt, deuten dabei zu definierende flexible Schnittstellen an.

Die möglichen Interaktionskontexte sind weder der Anwendung noch dem Anwender in vollem Umfang vorab bekannt. Über die Schnittstellen muss daher eine flexible, nahtlose und sichere Integration einer Anwendung in ein bestehendes Bedien- und Anzeigekonzept möglich sein, ohne das Wissen über dieses Konzept zur Entwicklungszeit zu besitzen. Zusätzliche Herausforderungen ergeben sich aus den Einschränkungen in einigen Kontexten, in denen Anwendungen genutzt werden. So unterliegt beispielsweise eine Benutzerschnittstelle im Auto strengen Anforderungen an sichere Bedienbarkeit ohne Ablenkung vom Fahrgeschehen. Solche Anforderungen müssen mit der Lösung des Problems ebenfalls adressiert werden.

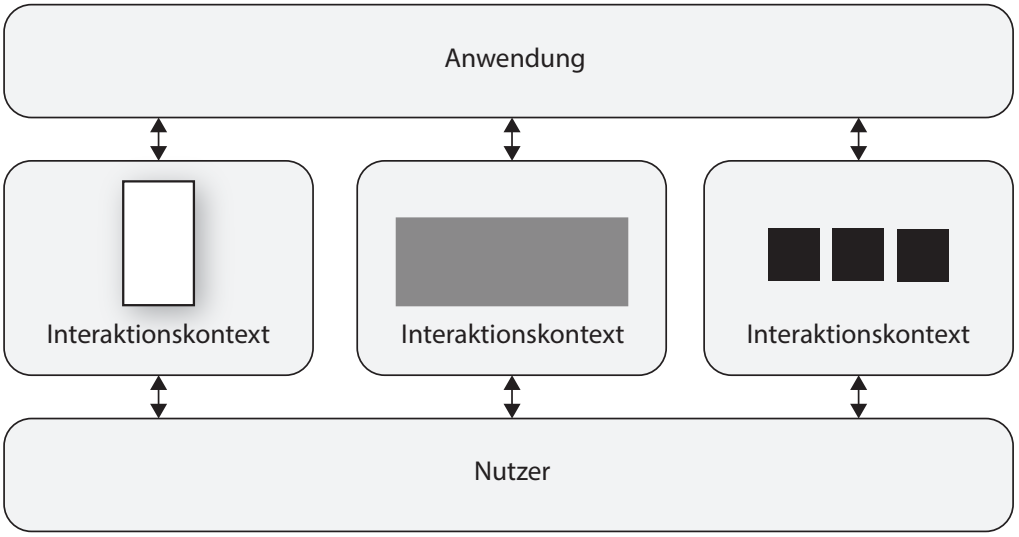


Abbildung 1.1.: Problemstellung: Flexible Schnittstellen zwischen Anwendung, verschiedenen Interaktionskontexten und Nutzer, um eine Computeranwendung perfekt und nahtlos in die Umgebung des Nutzers zu integrieren.

## 1.3. Ziele der Arbeit

Aus der Motivation, eine flexible Benutzerschnittstellenintegration über definierte Schnittstellen zu erzielen, die unabhängig von verwendeten Bedien- und Anzeigekonzepten oder eingesetzten Plattformen sind, ergeben sich die Ziele der Arbeit:

**Flexible abstrakte Beschreibung von Benutzerschnittstellen** Die Form der Beschreibung zu integrierender Anwendungen muss möglichst unabhängig von konkreten Bedien- und Anzeigekonzepten sein und ausreichend Flexibilität bieten, um eine dynamische Erzeugung von Benutzerschnittstellen aus dieser Beschreibung für unterschiedliche Zielsysteme zu erreichen.

**Automatische Weiterverarbeitung und Integration der abstrakten Beschreibung** Die abstrakte Beschreibung zu integrierender Anwendungen muss automatisch weiterverarbeitet werden können, sodass zur Laufzeit eine Benutzerschnittstelle für das

Zielsystem erzeugt werden kann. Dazu sind Regeln für die Weiterverarbeitung, insbesondere der Transformation in konkrete Benutzerschnittstellenelemente, erforderlich. Die abstrakten Elemente der Beschreibung müssen auf Ein- und Ausgabeelemente des Zielsystems transformiert werden können.

**Unabhängige, flexible Entwicklungsprozesse** Der Entwicklungsprozess einer Anwendung soll flexibel sein und möglichst unabhängig von einer späteren Integration in ein bestehendes System erfolgen können. Auf der Seite eines Zielsystems sollen Entwicklungs- und Integrationsprozess flexibel und erweiterbar sowie nicht an festgelegte Entwicklungswerkzeuge gebunden sein.

**Sicherstellung domänenspezifischer Anforderungen im Zielkontext** Das Vorgehen der Integration soll auch die Erfüllung besonderer Anforderungen der Anwendungsdomäne eines Zielkontextes sicherstellen. Beispiel für derartige Anforderungen finden sich im Kontext von Benutzerschnittstellen im Automobil, wo das Zielkonzept von rechtlicher Seite durch Regeln für minimale Fahrerablenkung, aber auch durch generelle Anforderungen an Komfort und Sicherheit beeinflusst wird.

## 1.4. Einordnung der Arbeit

Für eine abstrakte Beschreibung von Benutzerschnittstellen muss die Komplexität konkreter Ausprägungen reduziert werden. Die Modellbildung ist ein Kernkonzept der Informatik, um abstrakte Sichten auf konkrete Probleme zu erreichen [KB08]. Die relevanten Teile einer Benutzerschnittstelle können so zunächst auf einer abstrakteren Ebene beschrieben werden, um in späteren Schritten die konkreten Ausprägungen abzuleiten. Die vorliegende Arbeit lässt sich daher in den Bereich modellbasierter Entwicklung einordnen. Viele bestehende Ansätze für modellbasierte Benutzerschnittstellenentwicklung (u.a. [VLM+04][Mel10][Mei10][Sch07a]) haben dabei im Fokus, den Entwicklungsprozess besser zu unterstützen und gehen nicht davon aus, dass die Erzeugung geeigneter Benutzerschnittstellen auf der Zielplattform in einem Schritt automatisch erfolgt. Es hat sich gezeigt, dass vollautomatisch erzeugte Benutzerschnittstellen im Sinne modellgetriebener Entwicklung [LSHA07] häufig von unzureichender Qualität sind und das Ergebnis nicht vorhersehbar ist [MHP00]. Die vorliegende Arbeit trägt zur Schließung

dieser Lücke bei und untersucht die Möglichkeiten und Grenzen eines vollautomatischen Erzeugungsprozesses für die Integration unbekannter Anwendungen in eine bestehende Benutzerschnittstelle. Dazu müssen die abstrakt definierten Beschreibungen der Interaktionen und Benutzerschnittstellenelemente eine garantierte und konsistente Abbildung auf das Zielsystem ermöglichen. Es geht nicht um die schrittweise kontrollierbare Ableitung von Benutzerschnittstellen aus möglichst freien abstrakten Modellen. Die Ergebnisse der Arbeit tragen dazu bei, Teile von Benutzerschnittstellen in adaptive, multimodale Dialoge im Sinne des *Ubiquitous Computing* [Wei91] zu integrieren. Die Arbeit kann in diesem Sinne auch als Bindeglied zwischen verschiedenen Modalitäten für die Mensch-Computer Interaktion dienen, und eröffnet neue Sichten und Fragen im Bereich der Entwicklung von multimodalen Benutzerschnittstellen.

## 1.5. Beiträge

Die vorliegende Arbeit betrachtet die Erstellung von Benutzerschnittstellen für Anwendungen, die in ein bestehendes interaktives System integriert werden. Durch Modellierung streben bestehende Ansätze eine Erleichterung oder Automatisierung des Entwicklungsprozesses an, während in dieser Arbeit der Fokus auf der Integration bereits entwickelter Anwendungen in bereits entwickelte Systeme liegt. Dafür wird eine **abstrakte Modellierungsmethode** vorgestellt, die Domänen-, Dialog- und Präsentationsmodell vereint, um die Interaktionsmöglichkeiten mit einer Anwendung abstrakt beschreiben zu können. Regelbasierte Transformationen ermöglichen die Generierung von Benutzerschnittstellen aus diesen abstrakten Beschreibungen für ein Zielsystem. Auf Basis einer Client-Server-Architektur und dem Model-View-Controller (MVC) Pattern beschreibt die Arbeit außerdem eine **Architektur zur Integration** von derart modellierten Anwendungen in einen bestehenden, der Anwendung nicht bekannten, Kontext.

Das Modellierungskonzept sowie die Architektur werden prototypisch umgesetzt, evaluiert und diskutiert. Die Ergebnisse tragen zur Schließung der Lücke zwischen Anwendung und Interaktionsmöglichkeiten bei, die bei der Integration unbekannter Anwendungen in bestehende interaktive Systeme besteht.

## 1.6. Kapitelüberblick

Die Arbeit ist in insgesamt 9 Kapitel gegliedert. Das folgende Kapitel 2 führt in Grundlagen des Forschungs- und Anwendungsgebiets der Mensch-Computer Interaktion mit dem Fokus Benutzerschnittstellenentwicklung ein. Dabei werden Beispiele aus dem Automobilbereich diskutiert, die sich zur Veranschaulichung der Grundlagen eignen. In Kapitel 3 werden diese Grundlagen um Ansätze zur modellbasierten und modellgetriebenen Entwicklung von Benutzerschnittstellen erweitert. Aufbauend auf der Einordnung des Stands der Technik werden in Kapitel 4 die Ziele dieser Arbeit im Detail abgeleitet, welche dem Konzept für die Interaktionsmodellierung zur Integration von Anwendungen in bestehende Benutzerschnittstellen zu Grunde liegen. Eine Architektur zur technischen Integration der mit dem vorgestellten Konzept modellierten Anwendungen wird in Kapitel 5 erarbeitet. Ein besonders geeigneter Anwendungskontext zur Umsetzung und Bewertung des Konzepts findet sich am Beispiel von Benutzerschnittstellen im Automobil. In Kapitel 6 werden prototypische Umsetzungen innerhalb dieses Anwendungskontexts vorgestellt. Die Prototypen der verschiedenen Aspekte des Konzeptes werden beschrieben und bewertet. In Kapitel 7 werden die Ergebnisse der Arbeit zusammengefasst und Kapitel 8 beinhaltet eine umfangreiche Diskussion dieser Ergebnisse. Im abschließenden Kapitel 9 werden die Gesamtzusammenfassung der Beiträge sowie ein Ausblick auf zukünftige Entwicklungen in verwandten Bereichen gegeben.

# 2

# Grundlagen

In diesem Kapitel werden die wichtigsten Teilgebiete der Querschnittsdisziplin Mensch-Computer Interaktion sowie grundlegende Methoden der Benutzerschnittstellenentwicklung eingeführt. Diese Grundlagen werden im weiteren Verlauf der Arbeit benötigt, um das Verständnis für notwendige Teile von Benutzerschnittstellen zu schaffen sowie das Vorgehen zur Erstellung dieser Teile zu verstehen, bevor Methodik und Umsetzung des Konzepts weiter erarbeitet werden. Als Anwendungsbeispiel werden Benutzerschnittstellen und deren Entwicklung im Automobilbereich eingeführt. Aufgrund seiner Besonderheiten eignet sich dieser Bereich besonders gut als Anwendungsbeispiel für die vorliegende Arbeit.

## 2.1. Mensch-Computer Interaktion

Der folgende Abschnitt bietet einen Überblick über die wichtigsten Teilgebiete des Anwendungsgebiets der Mensch-Computer Interaktion, das sich mit der Schnittstelle zwischen Mensch und Computer befasst. Zunächst wird ein Überblick über die Historie sowie die Begriffe der Mensch-Computer Interaktion gegeben. Danach werden die für diese Arbeit wichtigsten Teilgebiete vorgestellt.

### 2.1.1. Historie und Begriffsklärung

Das Forschungs- und Anwendungsgebiet der *Mensch-Computer Interaktion* (HCI, engl. für *Human-Computer Interaction*) umfasst eine große Bandbreite an Themen und ist eine klassische Querschnittsdisziplin. Aktuelle Curricula an Universitäten und Hochschulen beinhalten dabei hauptsächlich Lehrveranstaltungen aus Informatik, Design, Ingenieurwissenschaften und der Psychologie. Technischen Fragestellungen zu Computerprogrammen stehen dabei ergonomische Fragestellungen zur Bedienung der Computerprogramme durch den Menschen gegenüber. Während die Geschichte der Computertechnologie nur bis etwa in die 1940er Jahre zurückreicht [Cer03], beschäftigten sich Menschen bereits vor mehreren Jahrhunderten mit ergonomischen Fragestellungen. Dazu gehört auch die Art und Weise der Bedienung von Maschinen bevor diese von Computertechnologie geprägt waren. Der Begriff der Maschine wird dabei folgendermaßen definiert:

> **Definition: Maschine**
> Eine Maschine ist eine mechanische, aus beweglichen Teilen bestehende Vorrichtung, die Kraft oder Energie überträgt und mit deren Hilfe bestimmte Arbeiten unter Einsparung menschlicher Arbeitskraft ausgeführt werden können [Bib].

Mit dem Einzug moderner Computer, die zunehmend für die Bedienung einer Maschine verwendet wurden, wurde in den 1980er Jahren der Begriff der Mensch-Computer Interaktion geprägt [DFAB04]. Als Folge kann der Begriff der *Mensch-Computer Interaktion* als Teil der *Mensch-Maschine Interaktion* (HMI, engl. für *Human-Machine Interaction*) gesehen werden. Werden alle nötigen Bedienschritte über einen Computer erledigt, ersetzt die Mensch-Computer Interaktion die Interaktion mit der Maschine. Der Mensch bedient dabei lediglich einen Computer, welcher über entsprechende Programme die

Steuerung einer Maschine übernimmt. Im Duden werden Computer und Programm folgendermaßen definiert:

> **Definition: Computer**
> Ein Computer ist eine programmgesteuerte, elektronische Rechenanlage [Bib].

> **Definition: Programm**
> Ein Programm ist eine Folge von Anweisungen für eine Anlage zur elektronischen Datenverarbeitung zur Lösung einer bestimmten Aufgabe[Bib].

Die Abgrenzung von HCI zur HMI ist nicht immer einfach. So lässt sich beispielsweise darüber streiten, ob das Drücken des Bremspedals im Auto eine Handlung des Fahrers im Sinne einer Mensch-Computer oder einer Mensch-Maschine Interaktion darstellt. Schließlich könnte es sich um ein Bremssystem handeln, das in erheblichem Maße computergesteuert ist. Die Bedienung des Bordcomputers, um etwa ein Navigationsziel einzugeben, kann dagegen eindeutig als Interaktion zwischen Fahrer und einem Computer eingeordnet werden.

In dieser Arbeit geht es um die Interaktion mit einem Computer oder einem System, das durch einen Computer gesteuert wird. Es wird daher der Begriff der Mensch-Computer Interaktion verwendet, wie von Dix et al. [DFAB04] folgendermaßen geprägt: *Mensch* ist ein einzelner Anwender oder eine Gruppe von Anwendern. *Computer* ist ein technisches rechnergestütztes System, mit dem der Mensch zur Erreichung eines bestimmten Ziels interagiert. Diese *Interaktion* wird für diese Arbeit daher folgendermaßen definiert:

> **Definition: Mensch-Computer Interaktion**
> Mensch-Computer Interaktion bezeichnet die direkte oder indirekte Kommunikation zwischen Mensch und Computer, um ein Ziel zu erreichen.

### 2.1.2. Teilgebiete

Eine fundierte Einordnung der Mensch-Computer Interaktion in Teilgebiete veröffentlichte die *Association for Computing Machinery* (ACM) bereits 1992. Ergebnis ist ein Curriculum mit Empfehlungen für die Lehre der Mensch-Computer Interaktion [ACM92], welches immer noch häufig als Grundlage für Einführungen in die vielen Teilgebiete

herangezogen wird. Abbildung 2.1 illustriert diese Teilgebiete der Mensch-Computer Interaktion: Der *Entwicklungsprozess (Development Process)* für HCI-Anwendungen wird von den beiden Bereichen *Mensch (Human)* und *Computer* mit den jeweiligen Anforderungen und Eigenschaften geprägt. So muss die Funktionsweise der Ein- und Ausgabe für den Menschen nachvollziehbar gestaltet werden. Man spricht dabei häufig von Anforderungen an eine intuitive Bedienung. Weiteren Einfluss auf die HCI haben das jeweilige *Einsatzgebiet* und der *Anwendungskontext (Use and Context)*. So ist es beispielsweise wichtig zu wissen, wie viele Menschen mit dem System direkt oder indirekt arbeiten, welche Anforderungen an die Privatsphäre bestehen oder welche Lichtverhältnisse und Lautstärkepegel erwartet werden. Die vorliegende Arbeit konzentriert sich auf Fragestellungen zu Entwicklungsprozessen und den resultierenden Benutzerschnittstellen (*Input and Output Devices*).

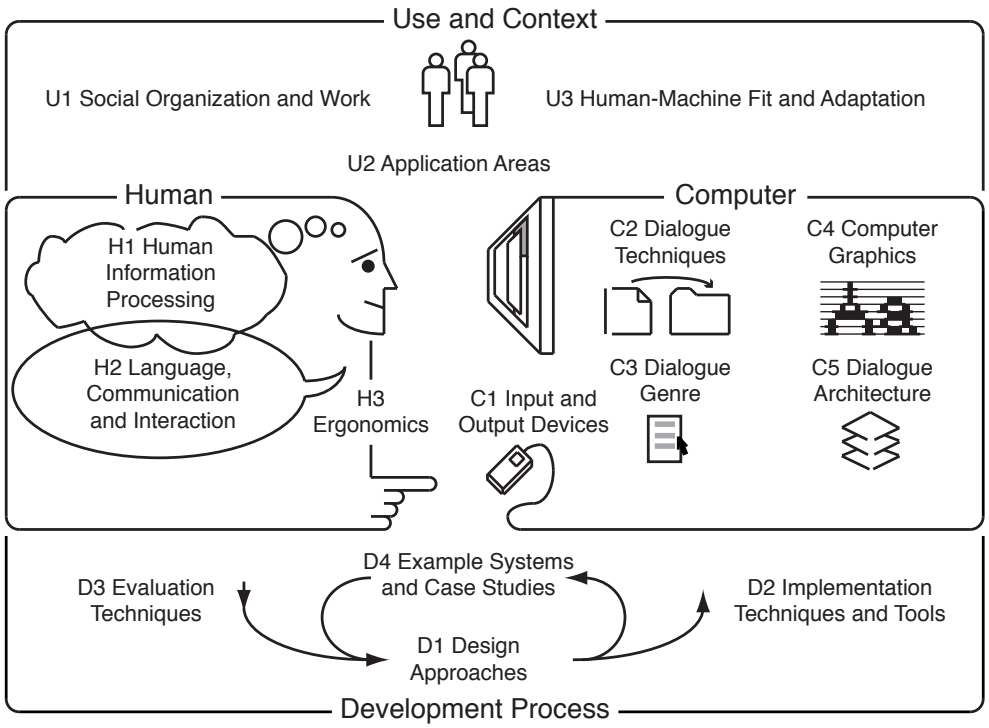


Abbildung 2.1.: Teilgebiete der Mensch-Computer Interaktion aus dem Curriculum der ACM [ACM92].

Eine Benutzerschnittstelle befindet sich zwischen Mensch und Computer und bezeichnet Ein- und Ausgabemöglichkeiten, mit denen der Benutzer Anweisungen an ein Computer-

system übermitteln und Rückmeldungen empfangen kann. Eine Begriffsdefinition sowie Einführungen in unterschiedliche Arten von Benutzerschnittstellen werden im folgenden Abschnitt gegeben.

## 2.2. Benutzerschnittstellen

Für die Mensch-Computer Interaktion werden Schnittstellen benötigt, die dem Menschen das Bedienen des Computers ermöglichen. Mit einer *Benutzerschnittstelle* (UI, engl. für *User Interface*) wird bestimmt, wie der Benutzer Anweisungen an ein Computersystem übermitteln und Rückmeldungen empfangen kann. Insgesamt gehören dazu, neben der bedienbaren Software, auch entsprechende Eingabegeräte, zum Beispiel Bedienknöpfe, Tastatur, Maus, Mikrophone oder Kameras, sowie Ausgabegeräte wie zum Beispiel Bildschirme oder Lautsprecher. In der europäischen Norm ISO 9241-110 wird der Begriff der Benutzerschnittstelle folgendermaßen definiert [Int90]:

> **Definition: Benutzerschnittstelle**
> Die Benutzerschnittstelle setzt sich zusammen aus allen Bestandteilen eines interaktiven Systems (Software oder Hardware), die Informationen und Steuerelemente zur Verfügung stellen, die für den Benutzer notwendig sind, um eine bestimmte Arbeitsaufgabe mit dem interaktiven System zu erledigen.

Für viele Umgebungen haben sich bestimmte Benutzerschnittstellen etabliert. Allen voran die grafisch repräsentierte Ausgabe über einen Bildschirm mit der Bedienung über Maus und Tastatur, wie in einem typischen Haushalt oder Büro zu finden. Darüber hinaus entstehen immer neue Formen von Benutzerschnittstellen als Kombination von Ein- und Ausgabegeräten, die im Forschungsbereich der HCI entwickelt und evaluiert werden. Ein Ziel solcher Entwicklungen ist, dass der Computer immer mehr in den Hintergrund rückt und die Interaktionen mit dem Computer natürlicher werden. Im Folgenden werden unterschiedliche Arten von Benutzerschnittstellen und deren Einsatzgebiete charakterisiert.

### 2.2.1. Grafisch-haptische Benutzerschnittstellen

Grafisch-haptische Benutzerschnittstellen sind derzeit die am häufigsten eingesetzten Benutzerschnittstellen im PC-Bereich [Mel10]. Die Kurzform *Grafische Benutzerschnitt-*

*stellen* (GUI, engl. für *Graphical User Interface*) hat sich im allgemeinen Sprachgebrauch durchgesetzt. Diese sind in den meisten Heim- und Büroumgebungen zu finden. Einer grafischen Ausgabe stehen dabei haptische Eingabegeräte, wie Maus oder Tastatur, gegenüber. Frühe GUIs waren textbasiert und wurden per Tastatur bedient. Abbildung 2.2 zeigt ein typisches rein textbasiertes Eingabefenster zur Dateiverwaltung, basierend auf Microsoft DOS (Abkürzung für *Disk Operating System*).

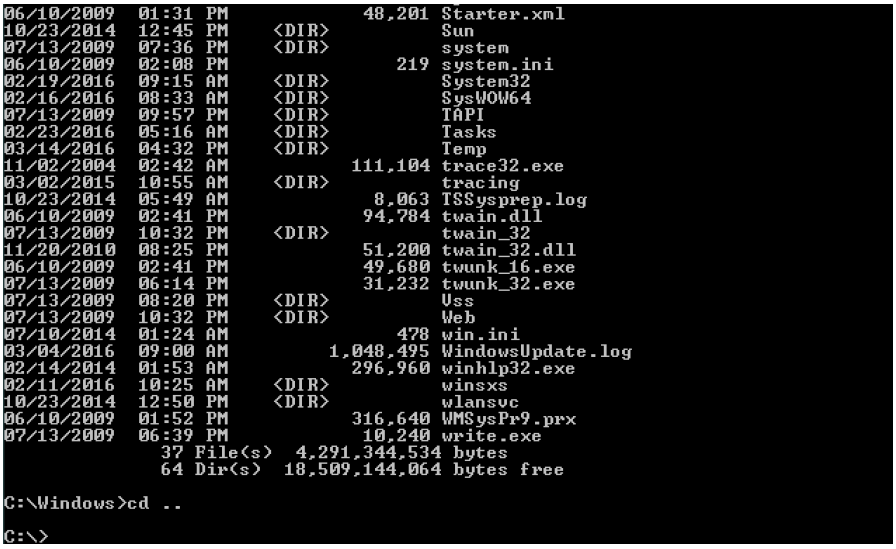


Abbildung 2.2.: Textbasiertes Eingabefenster zur Dateiverwaltung, basierend auf Microsoft DOS.

In den letzten Jahren haben sich neue Formen der Ein- und Ausgabe über grafische Benutzerschnittstellen etabliert. Dazu zählen etwa berührungssensitive Bildschirme, so genannte *Touchscreens*. Durch direktes Drücken mit dem Finger oder einem Stift auf dem Bildschirm wird mit dem Computer interagiert. Die grafische Ausgabe liegt direkt unter der berührbaren Fläche. Dies erlaubt die direkte Manipulation von grafisch repräsentierten Elementen. Auch besondere Fingergesten bieten neue Interaktionsmöglichkeiten, etwa für Vergrößern, Umblättern oder Weiterschalten. Durch die Verwendung mehrerer Finger, *Multitouch*, wird die Anzahl an Interaktionsmöglichkeiten weiter erhöht. Eine weitverbreitete Umsetzung dieser Art berührungssensitiver Bildschirme bietet die Firma Apple[1]. Abbildung 2.3 zeigt das *Apple iPad Air 2* mit seiner berührbaren grafischen Oberfläche.

[1] http://www.apple.com, aufgerufen am 27.10.2015

Abbildung 2.3.: Berührbare grafische Oberfläche eines *Apple iPad Air 2* (Bildnachweis: S. 168).

Mit dem Einzug von Soundkarten in Heimrechner und der Weiterentwicklung von Ein- und Ausgabegeräten haben sich einige Technologien verbreitet, die eine grafische Ausgabe an den Nutzer ergänzen oder ersetzen können. Dazu gehören die haptische Ausgabe, die Ausgabe von Tönen sowie Sprache, welche im Folgenden erläutert werden.

### 2.2.2. Haptische Ein- und Ausgabe

Zusätzlich zur haptischen Eingabe bieten einige Geräte auch haptische Rückmeldungen. Geräte mit haptischer Ausgabe stimulieren den Tastsinn und die sogenannte *Propriozeption*. Das ist die Wahrnehmung von Körperbewegungen sowie der Lage des eigenen Körpers im Raum [Iwa08]. Dies wird durch Oberflächenbeschaffenheit oder -temperatur sowie das Ausüben von Kräften auf einzelne Körperteile oder den ganzen Körper des Benutzers erreicht. Ein Beispiel für haptische Ausgabe ist das sogenannte *Force Feedback*, eine Rückmeldung von Kraft an den Benutzer. Abbildung 2.4 zeigt ein geöffnetes *Gamepad*, in dem Motoren für die Bereitstellung des Force Feedback verbaut sind. Dies kann etwa in einer virtuellen Spielumgebung eingesetzt werden, um beispielsweise den Eindruck von Unebenheiten auf dem Boden zu vermitteln.

Im Alltag des Menschen spielen Tastsinn und Propriozeption eine große Rolle und ein zunehmender Einsatz von haptischen Ein- und Ausgabegeräten bei der Mensch-Computer

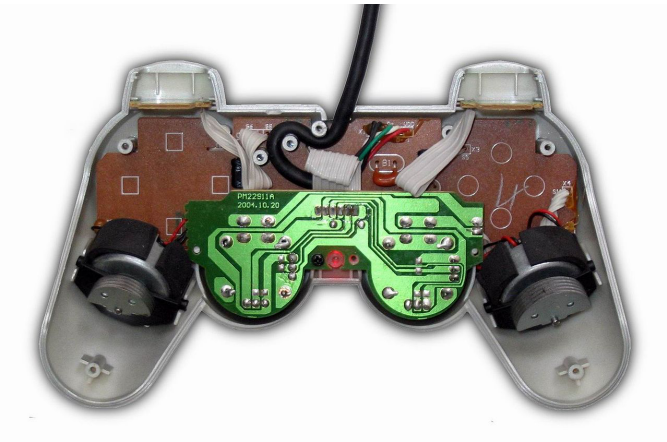

Abbildung 2.4.: Gamepad mit Motoren für *Force Feedback* (Bildnachweis: S. 168).

Interaktion kann erwartet werden [Iwa08]. Weit verbreitet ist bereits die Vibration eines Mobiltelefons als Hinweis auf eingehende Anrufe oder Nachrichten. Auch im Bereich der Computerspiele sind Controller mit haptischem Feedback üblich. Einen fundierten Überblick über Methoden und Technologien für haptische Schnittstellen und Beispiele aus Industrie und Forschung bietet zum Beispiel Iwata [Iwa08].

### 2.2.3. Auditive Ausgabe und Sprachbedienung

**Auditive Ausgabe** Die Verwendung auditiver Ausgaben hat sich in Heim- und Bürorechnern sowie mobilen Geräten durchgesetzt, um Warnungen, Hinweise oder Rückmeldungen über erfolgreich durchgeführte Interaktionen zu geben. So wird etwa eine eingehende E-Mail mit einem entsprechenden Ton verknüpft oder das Leeren des virtuellen Papierkorbs mit dem Geräusch begleitet, das beim Zerknüllen von Papier entsteht. Antwortzeiten auf auditive Signale sind häufig kürzer als auf visuelle Signale, allerdings werden auditive Signale auch eher als Störung empfunden [Kra94]. Daher werden auditive Ausgaben, vor allem deutliche Warntöne, eher für kritische Situationen verwendet, in denen die Aufmerksamkeit des Nutzers erforderlich ist. Nichtsprachliche auditive Ausgaben basieren hauptsächlich auf an die Realität angelehnten *auditiven Symbolen* [Gav94], etwa dem Geräusch für das Zerknüllen von Papier, oder abstrakteren Tonfolgen wie dem akustischen Signal für eine eingehende E-Mail. Diese werden auch *Earcons* genannt [BWE94]. Einige Ansätze verwenden außerdem die Techniken der Audifizierung (engl. *Audification*) oder Parameterabbildung (engl. *Parameter Mapping*)

[HH05]. Damit werden Datensätze hörbar gemacht oder bestimmte Parameter relevanter Daten auf Töne abgebildet. Veränderungen in Daten können so erkannt werden ohne die Notwendigkeit ständiger visueller Aufmerksamkeit. Ein Beispiel für eine solche Parameterabbildung ist das in vielen Fahrzeugen verfügbare akustische Warnsystem, das den verfügbaren Abstand des Fahrzeugs zu einem Hindernis hörbar macht. Die Frequenz und/oder Lautstärke der Warntöne verändert sich mit abnehmender Distanz.

Für die Repräsentation solcher veränderlicher Daten sowie Warnungen und Hinweise bieten sich nichtsprachliche auditive Ausgaben etwa als Ergänzung zu grafisch-haptischen Benutzerschnittstellen an. Bei der Kommunikation von absoluten Werten sind nichtsprachliche Ausgaben nur eingeschränkt geeignet. Hier kommen Möglichkeiten der Sprachausgabe in Betracht.

Um absolute Werte, etwa Zahlenwerte oder konkrete Informationen, zu kommunizieren, eignet sich ergänzend oder alternativ zu einer grafischen Ausgabe eine Sprachausgabe an den Nutzer [MB04]. Ein typisches Beispiel sind Fahrhinweise, die bei einem Navigationssystem im Fahrzeug per Sprache an den Fahrer übermittelt werden. Neben der reinen Sprachausgabe ermöglicht die Technologie entsprechend trainierter Spracherkenner die Eingabe von gesprochener Sprache. Damit kann dem Navigationssystem etwa das gewünschte Ziel per Sprache mitgeteilt werden. Der folgende Abschnitt geht auf sprachbedienbare Benutzerschnittstellen ein.

**Sprachbedienbare Benutzerschnittstellen** Bei der Ein- und Ausgabe über sprachbedienbare Benutzerschnittstellen wird ein Dialog zwischen einem Menschen und einem Computersystem über Sprache geführt. Systeme, die solche Schnittstellen anbieten, werden Sprachdialogsysteme (SDS, engl. für *Spoken Dialog System*) genannt [MBD05]. Ähnlich wie bei grafisch-haptischen Benutzerschnittstellen gab es auch im Bereich der sprachbedienbaren Benutzerschnittstellen eine Entwicklung von ersten eingeschränkten Systemen hin zu komplexeren Systemen mit mehr Freiheiten bei der Bedienung.

Bei Dienstleistungen über das Telefon, etwa einer Service-Hotline, kommen häufig Sprachdialogsysteme zum Einsatz. Einfache Systeme arbeiten mit Ja/Nein-Fragen oder beschränken die Antwortmöglichkeiten auf Auswahlzahlen. So könnte ein System den Dialog mit einem Anrufer etwa folgendermaßen starten:

> *“Guten Tag und herzlich willkommen! Wir helfen Ihnen gerne weiter: Haben Sie ein technisches Problem mit Ihrem Gerät, sagen Sie EINS. Haben Sie ein Problem oder Fragen zu Ihrem Vertrag, sagen Sie ZWEI.”*

Häufig ist es dabei auch möglich, die entsprechende Taste auf dem Telefon zu drücken. Dies war vor der Entwicklung und Verbreitung von Spracherkennern die einzige Möglichkeit eine Mensch-Computer Interaktion über ein Telefon zu realisieren. Heute gibt es Systeme, die einige Freiheiten beim Dialog über das Telefon zulassen. Ein Beispiel dafür ist die elektronische Fahrplanauskunft der Deutschen Bahn[2]. Das System kann ganze Sätze mit mehreren Informationen verarbeiten. Mit dieser Technologie sind Dialoge wie der folgende möglich:

> *“Herzlich Willkommen! Wie kann ich Ihnen helfen?”*
>
> “Ich möchte von Ulm nach Stuttgart fahren”
>
> *“An welchem Tag und zu welcher Zeit möchten Sie von Ulm nach Stuttgart fahren?”*
>
> ...

Hauptziel bei der Entwicklung neuer Sprachdialogsysteme ist es, so nah wie möglich an einen natürlichen Dialog, wie ihn Menschen untereinander führen würden, zu kommen. Die Forschung an solchen Systemen beinhaltet nicht nur die Vorabplanung möglicher Dialoge mit immer umfangreicheren Grammatiken, welche sprechbare Wörter und Sätze enthalten, sondern auch die Beeinflussung des Dialogs durch die emotionale Verfassung des Benutzers, die aktuelle Umgebung oder sonstige äußere Bedingungen wie die verfügbare Zeit für einen Dialog. Statistische Verfahren zur Spracherkennung und Entscheidung über die Dialogführung sind ebenfalls Gegenstand der weiteren Entwicklung an sprachbedienbaren Systemen. Für Details zu Aufbau und Funktionsweise von Sprachdialogsystemen sei auf weiterführende Literatur wie etwa von Jokinen und McTear verwiesen [JM10]. Minker et al. bieten außerdem einen umfassenden Überblick über Forschungsthemen im Bereich von Sprachdialogsystemen [MBD05].

[2] Elektronische Fahrplanauskunft der Bahn: Tel. 0800 1 50 70 90, www.bahn.de, aufgerufen am 02.03.2014

### 2.2.4. Weitere Formen von Benutzerschnittstellen

Im Folgenden werden weitere Formen von Benutzerschnittstellen, die Stand aktueller Forschung und Entwicklung sind, erläutert. Ein Fokus dabei ist, die reale Welt des Benutzers mit der virtuellen Welt des zu bedienenden Computersystems besser zu verbinden, um natürlichere Interaktion zu erreichen.

#### Eingabe über Gesten

Die Eingabe über Hand- und Körpergesten ist eine Möglichkeit für natürlichere Bedienung. Der Mensch kann damit ohne zusätzliche Eingabegeräte virtuell repräsentierte Objekte verändern. In aktuellen Systemen werden für die Erkennung von Gesten Kameras und Bildverarbeitungssoftware verwendet. Bewegungen des Benutzers werden aufgezeichnet und ausgewertet. Die Firma Microsoft brachte 2010 ein System zur Gestensteuerung für die Spielekonsole *XBOX 360* unter dem Namen *Kinect* auf den Markt[3]. Abbildung 2.5 zeigt die Kamera-Sensorleiste des Kinect-Systems. Die Leiste wird so angebracht, dass sie sich im Blickfeld des Benutzers befindet, also zum Beispiel über dem Fernsehgerät, in dem die Anwendung läuft. Ein weiteres aktuelles System, das Gestensteuerung unterstützt, ist die Spielekonsole *Nintendo Wii*[4]. Hier muss allerdings das kabellose Bedienteil in der Hand gehalten werden, damit die Hand- und Armgesten erkannt werden. Die Funktionsweise basiert auf Bewegungssensoren, die im Bedienteil verbaut sind.

Abbildung 2.5.: Kamera-Sensorleiste *Kinect* zur Steuerung der Spielekonsole *Microsoft XBOX 360* über Gesten (Bildnachweis: S. 168).

[3] http://www.xbox.com/en-US/xbox-360/accessories/kinect, aufgerufen am 27.10.2015
[4] http://www.nintendo.com, aufgerufen am 27.10.2015

**Erweiterte und virtuelle Realität**

Im Bereich von erweiterter und virtueller Realität werden spezielle Benutzerschnittstellen entwickelt. Mit erweiterter Realität (engl. *Augmented Reality*) ist die Verknüpfung von Eigenschaften und Objekten der realen Welt mit virtuellen Informationen gemeint. Virtuelle Realität dagegen ist das vollständige Eintauchen des Benutzers in eine komplett virtuelle Umgebung. Begriffsdefinitionen und die Unterschiede zwischen erweiterter Realität und virtueller Realität sind nicht immer eindeutig. Beide Formen können prinzipiell gemeinsam behandelt werden, da es keine scharfe Trennung sondern einen fließenden Übergang gibt [MTUK94].

Beispiele erweiterter Realität werden mit sogenannten Datenbrillen, Datenhelmen oder Head-Up Displays [BGG86, AN03] erreicht. Dabei werden Informationen in das reale Sichtfeld des Benutzers projiziert. Typische Anwendungen sind etwa Navigationshilfen, die im Blickfeld des Fahrers angezeigt werden, oder digitale Informationen für Mitarbeiter in Fabriken, um etwa Daten eines Bauteils oder nächste Arbeitsschritte einzublenden. Eine weitere Möglichkeit, digitale Informationen mit realen Objekten zu verknüpfen, bieten sogenannte *Greifbare Benutzerschnittstellen* (TUI, engl. für *Tangible User Interfaces*) [IU97]. Das experimentelle Städteplanungssystem von Underkoffler und Ishii [UI99] arbeitet etwa mit Gebäudemodellen aus Draht, die auf einer interaktiven Oberfläche platziert werden können. Über die Veränderung der Tageszeit kann die Beleuchtung angepasst werden und so die Schattenentwicklung für die Städteplanung interaktiv nachvollzogen werden. Außerdem kann die Windrichtung und -stärke eingestellt werden, um den Einfluss der Gebäudeplatzierungen auf Luftströmungen zu erfahren. Dies wird auf der interaktiven Oberfläche über digital repräsentierte Windpfeile ermöglicht, die sich je nach Windrichtung, -stärke und Lage der Gebäude verändern.

### 2.2.5. Natürliche Interaktion über multimodale Benutzerschnittstellen

Aus den aktuellen Forschungsthemen kann ein Trend zu Benutzerschnittstellen, welche die reale Welt des Benutzers oder reale Objekte einbeziehen, abgeleitet werden. Außerdem kann erwartet werden, dass es verstärkt geeignete Kombinationen verschiedener Benutzerschnittstellenarten geben wird, um dem Benutzer die Auswahl der jeweils für ihn geeigneten Form der Interaktion zu überlassen. Dies ist er aus seiner realen Welt gewohnt.

Mark Weiser beschrieb bereits 1991 die Vision der allgegenwärtigen und gleichzeitig in den Hintergrund rückenden Computer als integrativer Bestandteil unseres Alltags [Wei91]:

> "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it." [Wei91, S. 94]

Weiser prägte für diese Vision den Begriff *Ubiquitous Computing*, die Allgegenwärtigkeit rechnergestützter Informationsverarbeitung. An vielen Universitäten und Hochschulen ist daraus ein Anwendungsschwerpunkt im Informatikstudiengang entstanden. Dabei werden mobile Rechnerplattformen, die Kommunikation zwischen ihnen, und Formen geeigneter Interaktion mit solchen vernetzten Systemen, untersucht.

In einer Umgebung allgegenwärtiger Computer, in der nicht - wie häufig bei Heim- oder Bürorechnern der Fall - die Computer selbst Gegenstand der menschlichen Aufmerksamkeit sein sollen, werden Interaktionsformen benötigt, die den Menschen je nach aktueller Situation optimal unterstützen. Um die Sinne des Menschen je nach Situation geeignet ansprechen zu können, wird häufig *Multimodalität* bei Benutzerschnittstellen eingesetzt. Dieser Begriff lässt sich auf die unterschiedlichen *Sinnesmodalitäten* des Menschen zurückführen. Als *Sinnesmodalität* werden die von einem bestimmten Sinnesorgan vermittelten Empfindungen bezeichnet [HW09]. Folglich gibt es beim Menschen fünf Sinnesmodalitäten, welche die beiden so genannten *Fernsinne* Sehen und Hören sowie die drei *Nahsinne* Riechen, Schmecken und Fühlen umfassen. Benutzerschnittstellen in der HCI sprechen diese Sinnesmodalitäten unterschiedlich an. So wird bei rein grafischer Ausgabe nur der Sehsinn angesprochen. Bei der kombinierten Ausgabe von Tönen oder Sprachsignalen zusammen mit einer grafischen Repräsentation werden die Sinne Sehen und Hören parallel angesprochen. Hier tritt also Multimodalität bezüglich der Ausgabe auf. Im Alltag kombinieren wir Reize über verschiedene Sinnesmodalitäten und leiten daraus Informationen oder Handlungen ab. Während einer Unterhaltung bewerten wir bei unserem Gegenüber etwa die Körperhaltung, Gesichtsausdruck, Inhalt und Prosodik des Gesprochenen und, wenn wir uns in seiner Nähe befinden, auch den Geruch. In unserem Handeln verwenden wir außerdem das Wissen über die Sinnesmodalitäten unseres Gegenübers und geben Informationen entsprechend unterschiedlich ab. In einer lauten Umgebung versuchen wir möglicherweise über Gesten mit Händen und Füßen ausreichend Informationen zu vermitteln. In einer ruhigen Umgebung, die etwa Diskretion beinhaltet, können wir entweder leise flüstern, geben Zusatzinformationen über

unseren Gesichtsausdruck oder Handgesten oder skizzieren Informationen auf einem Blatt Papier, das wir unserem Gegenüber unbemerkt zukommen lassen können. Für die Vision des *Ubiquitous Computing*, in der Computer sich in unseren Alltag möglichst unbemerkt integrieren, ist es daher wichtig, Wissen über unterschiedliche Sinnesmodalitäten in die Gestaltung der Benutzerschnittstellen einzubeziehen. Der Nutzer sollte die Möglichkeit haben, während der Interaktion seine bevorzugte Modalität zu wählen, und der Computer sollte die passenden Sinnesmodalitäten des Nutzers je nach Situation ansprechen können.

Ein Beispiel, in dem Multimodalität wichtig ist, sind Benutzerschnittstellen im Auto. Während des Fahrens sollte sich der Fahrer nicht auf eine grafische Ausgabe auf dem Bildschirm des Bordcomputers konzentrieren und außerdem die Hände nur eingeschränkt vom Lenkrad nehmen. Das Wechseln zwischen verschiedenen Modalitäten ist also erforderlich, wenn sich das System optimal in die Umgebung des Fahrers integrieren soll. Dies geschieht bereits heute durch Sprachdialogsysteme, Bedienelemente am Lenkrad und Head-Up Displays. Im folgenden Abschnitt wird die Mensch-Computer Interaktion im Automobil als besonderes Anwendungsbeispiel für diese Arbeit im Detail vorgestellt.

## 2.3. Anwendungsbeispiel: Interaktion im Automobil

*Infotainment*, als zusammengesetzter Begriff für Information und Unterhaltung (engl. Entertainment), hat im Automobilbereich einen wertbestimmenden Rang erreicht [MT07]. Entsprechende Anwendungen nehmen immer mehr Einzug in den Interaktionsraum Automobil. Automobilhersteller zeigen eine teilweise sehr tiefe Integration solcher Anwendungen in Hard- und Softwareplattformen mit fest verbauten Bedien- und Anzeigegeräten, die für den mehrjährigen Lebenszyklus eines Automobils festgelegt sind. Daraus folgt eine Einschränkung der Flexibilität bei der Entwicklung und Aktualisierung von Infotainmentanwendungen. Anforderungen an sichere Bedienbarkeit bei minimaler Ablenkung schaffen zusätzliche Herausforderungen. Daher stellt der Interaktionsraum Automobil einen besonders geeigneten Anwendungsfall für diese Arbeit dar.

Im Folgenden werden die historische Entwicklung von Benutzerschnittstellen im Interaktionsraum Automobil erarbeitet sowie aktuelle Ein- und Ausgabegeräte im Fahrzeug vorgestellt.

### 2.3.1. Historie von Benutzerschnittstellen im Automobil

Die ersten Möglichkeiten zur Interaktion zwischen Mensch und Fahrzeug beschränkten sich auf Kernaufgaben wie Gasgeben oder Lenken. Später kamen Anzeigeelemente für bestimmte Betriebszustände wie Tank- oder Ölstand hinzu. Als Information an den Fahrer oder Mitfahrer wurde in einigen Fahrzeugen eine Uhr angeboten. Erst 1932 brachte die Einführung des Autoradios die erste Interaktionsmöglichkeit ins Fahrzeug, die über den reinen Informationsbedarf hinausging [MT07]. Mit Erfindung der *8-Spur-Kassette* und *Compact Cassette* in den 1960er Jahren [Cam88] wurde der Grundstein für handliche Abspielgeräte eigener Musik gelegt. Autoradios mit integrierter Abspielfunktion für Tonbandkassetten verbreiteten sich entsprechend und erweiterten die Unterhaltungsmöglichkeiten. In den 1980er Jahren folgte die Entwicklung einer heute nicht mehr wegzudenkenden Form des Infotainments im Automobil: Erste Navigationssysteme erreichten den japanischen Markt in Fahrzeugen von Honda, Nissan und Toyota [Ake06]. Der Funktionsumfang solcher Geräte wurde in Folge immer weiter ausgebaut. So ist es heute selbstverständlich möglich, umfangreiche Einstellungen zur Route zu treffen oder etwa nach Sonderzielen, wie einem Restaurant oder einer Tankstelle, zu suchen.

Mit steigendem Funktionsumfang der Infotainmentsysteme im Fahrzeug wurden neue Formen von Benutzerschnittstellen benötigt. Einfache Schalter für einzelne Funktionen reichten nicht aus. BMW stellte 2001 das iDrive-Bedienkonzept vor [ZWS01], dessen Kernidee ein zentrales Bedienelement mit variabler Haptik war, der sogenannte iDrive-Controller. Damit konnten alle Infotainmentanwendungen, repräsentiert über einen zentralen Bildschirm, bedient werden. Heute finden sich vergleichbare Konzepte in vielen Mittel- und Oberklassefahrzeugen. Zusätzlich wird seit Ende der 1990er Jahre die Bedienung über Sprachdialogsysteme im Fahrzeug angeboten. 1996 führte Mercedes-Benz unter dem Namen Linguatronic eine Sprachbedienung zunächst nur für Telefonfunktionen ein [Hei01]. Die sprachliche Bedienung anderer Infotainmentanwendungen folgte. Heute bieten die meisten Hersteller höherklassiger Fahrzeuge ein Sprachdialogsystem für Kommunikationsanwendungen, Navigationssystem und Multimediaanwendungen an. Unterschiedliche grafische Ausgabemöglichkeiten von einfachen Anzeigesymbolen bis zu komplexen 3D-Grafiken oder Projektionen von Informationen in die Windschutzscheibe sind zu finden und zur haptischen Bedienung gibt es zahlreiche Elemente, wie Schalter und Schaltflächen, Dreh- und Drücksteller, Touchpads und Touchscreens. Berührungslose Gestensteuerung findet sich vermehrt in Konzeptfahrzeugen sowie in einigen Serienfahrzeugen.

### 2.3.2. Interaktionsaufgaben im Fahrzeug

Im Automobil werden drei Arten von interaktiven Aufgaben unterschieden [MT07][Bub03]: *Primäre*, *sekundäre* und *tertiäre* Aufgaben:

- *Primäre Aufgaben* sind Aufgaben, ohne die das eigentliche Fahren nicht möglich ist, wie die Betätigung von Gas- oder Bremspedal sowie Bedienung des Lenkrads.
- *Sekundäre Aufgaben* sind Aufgaben, die direkt mit der Fahraufgabe verknüpft sind, wie das Blinken beim Abbiegen oder die Betätigung des Scheibenwischers.
- *Tertiäre Aufgaben* dagegen werden nebenbei ausgeführt und dienen nicht direkt der Fahraufgabe. Sie betreffen Komfort, Kommunikation, Information und Unterhaltung.

Dadurch, dass die tertiären Aufgaben während des Fahrens parallel zu primären und sekundären Aufgaben erledigt werden, ergeben sich besondere Herausforderungen an deren Gestaltung für den Nutzer. Wesentlich sind dabei Anforderungen an Bediensicherheit und Bedienklarheit [MT07]. Der Fahrer oder Mitfahrer soll sich sicher sein können, die gewünschte Funktion ausgelöst zu haben und das System soll Klarheit darüber schaffen, dass es diesen Wunsch erfasst hat und die Funktion ausführen wird.

Fokus für diese Arbeit sind die tertiären Aufgaben, die zu einem großen Teil über die im Auto angebotenen Infotainmentanwendungen erledigt werden und das Automobil zu einem vielfältigen Informations- und Unterhaltungsraum werden lassen. Die meisten Hersteller stellen das Infotainmentsystem in einem gemeinsamen zentralen Bordcomputer dar, der sogenannten *Head Unit*, mit einem einheitlichen Bedien- und Anzeigekonzept dafür im Fahrzeug [MT07]. Im Folgenden werden typische Bedien- und Anzeigekonzepte im Automobilbereich genauer vorgestellt.

### 2.3.3. Bedien- und Anzeigekonzepte für Infotainmentanwendungen

Heutige Infotainmentanwendungen im Fahrzeug lassen sich in die Kategorien Entertainment, Information, Assistenz und Kommunikation einordnen [MT07]. Zum *Entertainment* gehören vor allem das Abspielen von Musik von verschiedenen Quellen. Neben Radioempfang und CD-Spieler werden zunehmend Schnittstellen zu externen Abspielgeräten und handlichen Massenspeichern, wie USB-Speichersticks oder SD-Karten, angeboten. Mit dem Einzug von Internetempfang im Fahrzeug wurde außerdem die Möglichkeit,

Internetradio zu hören, zunehmend beliebter. In Fahrzeugen des oberen Segments finden sich neben Audiofunktionen auch Videofunktionen, um Filme über TV-Empfang, Film-DVDs oder über das Internet im Fahrzeug anzuschauen. Aus Sicherheitsgründen sind Videofunktionen neben einigen weiteren Interaktionen nur im Stand möglich.

In die Kategorie *Information* fallen Informationen zum Fahrzeug wie Betriebsmittelstände oder die Geschwindigkeit, Informationen zur aktuellen Position und zum anvisierten Ziel, Informationen zum Fahrzeugumfeld und zum Verkehr. Für die Informationen zum Fahrzeug und Fahrzeugumfeld sind in aktuellen Fahrzeugen zahlreiche Sensoren wie Messfühler, Kameras und Radarsensoren verbaut. Über Navigationssoftware werden Informationen zur Routenführung bereitgestellt und aktuelle Verkehrsinformationen werden über das digitale Radiosignal, Satellit, oder aus dem Internet empfangen. Über Internetdienste werden außerdem immer mehr zusätzliche Informationsdienste, wie Wetterinformationen, Börsenkurse oder aktuelle Nachrichten im Fahrzeug angeboten.

Für die Kategorie *Assistenz* spielen Bedien- und Anzeigemöglichkeiten ebenfalls eine besondere Rolle, da viele Assistenzsysteme nur durch die geeignete Anzeige der relevanten Daten im Fahrzeug einen Wert erhalten. Dazu gehören etwa Systeme zur Unterstützung bei Spurhaltung, Abstandhaltung oder Parkvorgängen. Es gibt zahlreiche weitere Assistenzsysteme, die eine Integration in das Bedien- und Anzeigekonzept erfordern. Für einen Überblick sei auf weiterführende Literatur verwiesen [WHW09].

Zur Kategorie *Kommunikation* gehört vor allem die Integration von Telefonfunktionen. Mobiltelefone werden über Bluetooth oder andere integrierte Schnittstellen mit dem Fahrzeug-Infotainmentsystem verbunden und dadurch über die Telefonanwendung mit Freisprecheinrichtung bedienbar. Über die Anbindung an das Internet werden außerdem weitere Kommunikationsdienste wie das Empfangen und Versenden von E-Mails ermöglicht. Auch private Nachrichten oder Statusmeldungen über soziale Netzwerke wie Facebook[5] gehören zu Kommunikationsdiensten in aktuellen Fahrzeugen.

### Haptische Eingabe

Für die Bedienung von Infotainmentanwendungen haben sich neben Tastern und Schaltern sowie unterschiedlichen Formen von Hebeln, Wippschaltern und Drehstellern zentral angeordnete universelle Bedienelemente etabliert, die sich in den meisten Fahrzeugen im Griffbereich zwischen Fahrer- und Beifahrersitz befinden. Über sogenannte Dreh- und

[5] Facebook, http://www.facebook.com, aufgerufen am 27.10.2015

Drücksteller und teilweise variabel belegbare Taster kann über eine grafische Menüführung aus dem Angebot an Infotainmentanwendungen ausgewählt und die entsprechende Funktion aktiviert werden. Dies geschieht über die grafisch realisierte Fokussierung des Elements auf dem zentralen Bildschirm. Abbildung 2.6 illustriert drei aktuelle Formen von zentral angeordneten Eingabemöglichkeiten in Fahrzeugen der oberen Mittelklasse.

Abbildung 2.6.: Haptische Eingabegeräte im Griffbereich zwischen Fahrer und Beifahrer. *Links:* Mercedes-Benz E-Klasse, 2011; *Mitte:* BMW 650i, 2011; *Rechts:* Audi A6, 2014 (Bildnachweis: S. 168).

In einigen Fahrzeugen findet auch ein zentral angeordneter Touchscreen Verwendung, über den die Elemente auf dem Bildschirm direkt mit dem Finger ausgewählt werden können. Abbildung 2.7 zeigt den zentralen Touchscreen des Tesla Model S (2015).

Abbildung 2.7.: Zentraler 17-Zoll Touchscreen im Tesla Model S, 2015 (Bildnachweis: S. 168).

Inspiriert von der Eingabe über eine berührbare Fläche haben sich außerdem Touchpads in einigen Fahrzeugen durchgesetzt. In Abbildung 2.8 ist das Touchpad im Audi A8

dargestellt. Es kann je nach Kontext als Ziffernfeld oder für die freie Eingabe von Zeichen, etwa bei der Navigationszieleingabe, verwendet werden [Aud10].

Abbildung 2.8.: Zentral angeordnete Bedienelemente für Infotainmentanwendungen und Touchpad mit Tastenfeld und Zeichenerkennung im Audi A8, 2009 (Bildnachweis: S. 168).

Zusätzlich zu den zentral angeordneten Elementen bieten die meisten Fahrzeuge Bedienmöglichkeiten direkt auf dem Lenkrad an. Die Anzeige der darüber manipulierbaren Inhalte erfolgt üblicherweise über Displays im Anzeigebereich hinter dem Lenkrad oder über ein Head-Up-Display, das in die Windschutzscheibe projiziert wird. Abbildung 2.9 stellt das Multifunktionslenkrad in der Mercedes-Benz S-Klasse dar mit Tastenfeldern zur Bedienung von Infotainmentanwendungen.

Abbildung 2.9.: Multifunktionslenkrad mit Bedienelementen für Infotainmentanwendungen in der Mercedes-Benz S-Klasse, 2013 (Bildnachweis: S. 168).

Eine Analyse der bei der Interaktion im Fahrzeug verwendeten Knöpfe, Hebel und Anzeigen sowie deren Lage zur Ein- und Ausgabe im Fahrzeug wurde von Kern und Schmidt

durchgeführt, die einen detaillierten Überblick über verschiedene Interaktionselemente in Fahrzeugen bieten [KS09].

### Grafische Ausgabe

Für die grafische Ausgabe haben sich zentral angeordnete Bildschirme etabliert. Ein Teil der Funktionen ist außerdem im Bereich der Geschwindigkeitsanzeige, dem sogenannten Kombiinstrument, oder über ein Head-Up-Display als Projektion in der Windschutzscheibe verfügbar. Abbildung 2.10 stellt einen zentralen Bildschirm im Audi sowie ein Head-Up-Display über dem Kombiinstrument mit Geschwindigkeits- und Drehzahlanzeige im BMW dar.

Abbildung 2.10.: Anzeigetechnologie im Fahrzeug. *Links:* Zentraler Bildschirm, Audi A3, 2012; *Rechts:* Head-Up-Display und Kombiinstrument im BMW, 2012 (Bildnachweis: S. 168).

Für die Mitfahrer im Fond haben sich außerdem Systeme mit Bildschirmen in den Lehnen oder Kopfstützen von Fahrer- und Beifahrersitz etabliert. Das sogenannte *Rear-Seat-Entertainment* [MT07] dient der Unterhaltung der Passagiere und ist in aktuellen Fahrzeugen in der Regel nicht mit dem zentralen Bordcomputer verbunden, sondern bietet über ein eigenes System die Nutzung von Audio- oder Videofunktionen. Eine Möglichkeit für die Unterhaltung des Beifahrers bietet die *Split-Screen-Technologie*, wie sie etwa in der Mercedes-Benz S-Klasse (2010) zum Einsatz kommt. Die Technologie wird genutzt, um Fahrer und Beifahrer über die jeweiligen Blickwinkel unterschiedliche Anzeigen zu ermöglichen. Dadurch kann der Beifahrer etwa einen Film während der Fahrt über den zentralen Bildschirm schauen ohne den Fahrer abzulenken, der beispielsweise weiterhin die Navigationskarte auf dem selben Bildschirm sieht.

### Sprachdialogsysteme

Sprachdialogsysteme wurden zunächst nur für Telefonfunktionen entwickelt, 1996 bot Mercedes-Benz erstmals eine entsprechende Sprachbedienung unter dem Namen Linguatronic an [Hei01]. Der Vorteil besonders im Kontext des Fahrens, wo die visuelle Aufmerksamkeit auf Straße und Verkehrsgeschehen liegen muss, führte zur Weiterentwicklung von Sprachdialogsystemen im Fahrzeug. Heute sind die meisten Infotainmentsysteme zumindest zum Teil sprachbedienbar. In der Regel muss vor Beginn eines Sprachdialogs vom Fahrer eine Taste gedrückt werden, woraufhin der Spracherkenner für die Eingabe von Sprachbefehlen aktiviert wird. Für die Sprachdialoge sind nicht immer alle Befehle möglich und auch nicht beliebige Alternativen zu bestimmten Ausdrücken. Die im aktuellen Kontext sprechbaren Befehle werden daher häufig über den grafischen Bildschirm eingeblendet, um den Fahrer zu unterstützen.

Aktuelle Sprachdialogsysteme können für die Bedienung der Telefonanwendung, Audio- und Videoanwendungen sowie für die Bedienung der Navigationsanwendung genutzt werden. Bei der Zieleingabe in der Navigationsanwendung wird durch die zusammenhängende sprachliche Eingabe von Zielort, Straße und Hausnummer an einem Stück eine schnelle und komfortable Bedienmöglichkeit erreicht. Diese sogenannte *One-Shot-Suchanfrage* [McT10] ist Stand aktueller Technik und trägt zu natürlicheren Sprachdialogen bei, die Ziel bei der Entwicklung zukünftiger Systeme sind.

Abbildung 2.11.: Innenraum der Fahrzeugstudie F125*!* von Mercedes-Benz, 2012, mit 3D-Display, 3D-Touchpad, Sprachbedienung für den Fahrer sowie einem großen Bildschirm auf der Beifahrerseite mit Anwendungen, die per Handgesten gesteuert werden (Bildnachweis: S. 168).

### Trends und zukünftige Konzepte

Im Sinne eines ganzheitlichen Bedien- und Anzeigekonzeptes müssen die unterschiedlichen Ein- und Ausgabemöglichkeiten perfekt zu einem Gesamtsystem vereint werden. Automobilhersteller demonstrieren diese zukünftigen Konzepte in Forschungsfahrzeugen, wie beispielsweise im Fahrzeug F125*!* von Mercedes-Benz. Abbildung 2.11 zeigt den Innenraum des F125*!*. In diesem Forschungsfahrzeug wurden neue Technologien, wie ein 3D-Kombiinstrument, das ohne Brille funktioniert, und ein neuartiges 3D-Touchpad für die berührungslose Eingabe im Raum vorgestellt. Die Fahrzeugstudie beinhaltete außerdem eine natürliche Form der Sprachbedienung, die ohne eine Taste zur Aktivierung des Sprachdialogs auskommt, sowie ein Unterhaltungssystem für Mitfahrer mit großem Bildschirm auf der Beifahrerseite, das über Handgesten bedient werden kann. Ein weiterer aktueller Trend wurde im Concept S-Class Coupe auf der Consumer Electronics Show (CES) 2014[6] gezeigt: Eine kontextsensitive Benutzerschnittstelle, die seine Benutzer über die Zeit kennenlernt und sich entsprechend anpasst. Abbildung 2.12 zeigt eine mögliche Darstellung der Aktionen, die als Abkürzung zu wahrscheinlichen nächsten Aktionen des Nutzers dienen, bereitgestellt von einem lernenden System.

Abbildung 2.12.: Kontextsensitive Benutzerschnittstelle in einem lernenden System von Mercedes-Benz, Prototyp präsentiert auf der Consumer Electronics Show in Las Vegas, 2014 (Bildnachweis: S. 168).

Die Integration neuer Internetanwendungen und -dienste sowie neuer Smartphones und anderer tragbarer Geräte, wie interaktive Brillen oder intelligente Uhren, findet sich vermehrt in Prototypen und Produktankündigungen von Automobilherstellern. Neben Automobilausstellungen sind Computer- und Elektronikmessen, wie die CES, immer beliebtere Plattformen für Auftritte der Automobilbranche. Der Trend zunehmender Integration von Internetdiensten und externen Anwendungen hält an und wird in den Ankündigungen der Branche thematisiert. BMW hat 2008 die weltweit erste Internetverbindung mit Zugriff auf Internetseiten und -dienste in einem Serienfahrzeug vorgestellt [BMW08]. Viele Hersteller folgten danach und entwickeln ihre Lösungen weiter. Ford

[6] Consumer Electronics Show (CES), http://www.cesweb.org, aufgerufen am 27.10.2015

hat auf der CES 2014 Erweiterungen seiner SYNC AppLink-Lösung [7] für die Integration von Internetanwendungen und -diensten vorgestellt und Audi hat für 2015 die erste serienmäßige Integration der Hochgeschwindigkeitsverbindung 4G LTE angekündigt [Aud14], was dazu beitragen kann, den Umfang von Internetanwendungen im Fahrzeug weiter auszubauen.

Zahlreiche weitere Trends lassen sich aus dem schnelllebigen Bereich der Mobiltelefone und Computerspiele sowie aus Forschungs- und Entwicklungsprojekten ableiten. Die seit 2009 jährlich stattfindende Konferenz für Benutzerschnittstellen und interaktive Applikationen im Fahrzeug (AutoUI)[8] bietet aktuellen Forschungsprojekten speziell aus dem Automobilbereich eine Bühne. Dabei sind unter anderem die Themen Gestenerkennung, Emotions- und Fahrerzustandserkennung sowie erweiterte und virtuelle Realität vertreten, die Einfluss auf die Weiterentwicklung von Bedien- und Anzeigekonzepten im Fahrzeug haben werden.

Aus der Übersicht über aktuelle Arten von Benutzerschnittstellen wird deutlich, dass die Sinne des Menschen sowie sein natürliches Verhalten verstärkt bei der Entwicklung von interaktiven Anwendungen beachtet werden, um die Interaktionen natürlicher zu gestalten. Die akustische oder haptische Ausgabe von Informationen bietet umfangreiche Möglichkeiten, weitere Sinne des Menschen anzusprechen, um den Sehsinn zu entlasten. Auch ist etwa die direkte Eingabe über einen Touchscreen für viele Anwender intuitiver als die indirekte Eingabe über eine Maus oder ähnliche Geräte. Für die unterschiedlichen Umgebungen, in denen Mensch-Computer Interaktion stattfindet, wie etwa im Auto, müssen dabei zusätzlich die Rahmenbedingungen der Anwendung beachtet werden. So bietet sich im Auto eine umfassende Sprachbedienung während des Fahrens an, damit der Fahrer seine Augen weiter auf das Verkehrsgeschehen richten kann. Ein Touchscreen kann im Auto möglicherweise weniger geeignet sein aufgrund der Fahrbewegungen, die eine genaue Eingabe erschweren. Ebenso gilt im Fahrzeug: Ablenkende Inhalte auf den Bildschirmen im Fahrzeug müssen während der Fahrt unterbunden werden. Dies schränkt den Gestaltungsspielraum bei interaktiven Anwendungen weiter ein.

Der Umfang der Entwicklung interaktiver Anwendungen wird durch die Bestrebungen, mehr Sinne des Menschen flexibel anzusprechen und eine optimale Anpassung an die jeweilige Einsatzumgebung zu erreichen, größer. Außerdem werden die technischen

[7] Ford SYNC AppLink, http://www.ford.com/technology/sync/, aufgerufen am 27.10.2015

[8] International Conference on Automotive User Interfaces and Interactive Vehicular Applications, www.auto-ui.org, aufgerufen am 27.10.2015

Möglichkeiten umfangreicher. Für die Entwickler von Benutzerschnittstellen, die sich in der Regel aus einem heterogenen Team von Psychologen, Designern und Informatikern zusammensetzen, werden die Herausforderungen dadurch größer.

Der folgende Abschnitt gibt einen Überblick über die wichtigsten Aufgaben bei der Benutzerschnittstellenentwicklung und skizziert Methoden und Werkzeuge, die dabei herangezogen werden.

## 2.4. Benutzerschnittstellenentwicklung

Die steigende Vielfalt der technischen Möglichkeiten und Einflüsse bestehender und neuer Konzepte stellt eine Herausforderung für die Benutzerschnittstellenentwicklung dar, insbesondere in der Automobilbranche. In dieser Arbeit wird das Ziel verfolgt, flexible, abstrakte Benutzerschnittstellen zu beschreiben, die in bestehende, abgeschlossene Systeme, beispielsweise ein Infotainmentsystem im Auto, integriert werden können und den besonderen Anforderungen im Zielkontext gerecht werden.

Im folgenden Abschnitt wird zunächst *Usability Engineering* als etablierte, nutzerzentrierte Vorgehensweise zur Benutzerschnittstellenentwicklung vorgestellt. Dieses Wissen fließt in die Konzeptentwicklung in dieser Arbeit ein. Danach werden allgemeine Methoden und Werkzeuge zur Benutzerschnittstellenentwicklung eingeführt, gefolgt von einem Abschnitt, der genauer auf die Besonderheiten der Benutzerschnittstellenentwicklung für Systeme im Auto eingeht.

### 2.4.1. Usability Engineering

Um eine gebrauchstaugliche Benutzerschnittstelle zu erreichen, haben sich iterative Ansätze mit starkem Einbezug möglicher Benutzer des Zielsystems durchgesetzt. Einem ersten Entwurf von Teilaspekten folgt dabei häufig eine begleitende Evaluation mit möglichen Benutzern. Daraus ergeben sich Anpassungen und Implikationen für andere Teile des Gesamtsystems, die in den Prozess einfließen können. Ein etablierter ganzheitlicher Ansatz ist der *Usability Engineering Lifecycle* nach Mayhew [May99], der in Abbildung 2.13 illustriert wird. So folgt auf eine umfangreiche Anforderungsanalyse, in der Eigenschaften von Nutzern, Aufgaben und Umgebungen sowie generelle Gestal-

tungsrichtlinien erarbeitet werden, ein iterativer Prozess, in dem Dialogabläufe, Konzepte und konkrete Designaspekte entstehen.

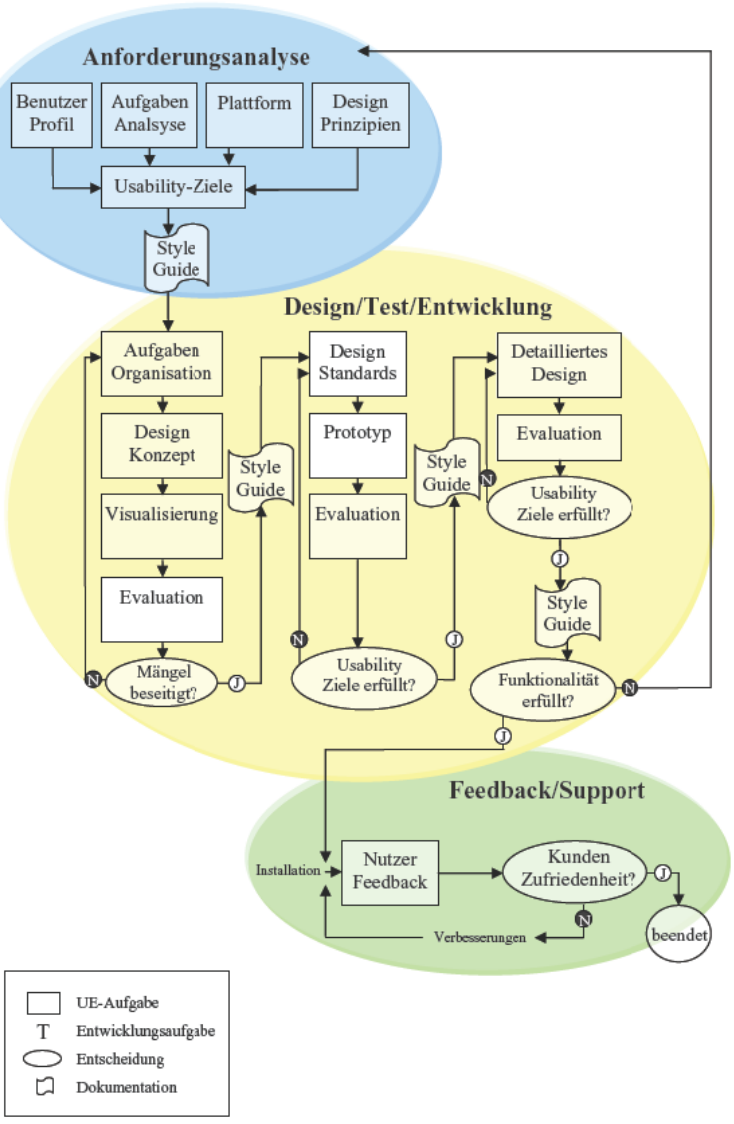


Abbildung 2.13.: Usability Engineering Lifecycle nach Mayhew [May99].

### 2.4.2. Die vier Säulen erfolgreicher Benutzerschnittstellenentwicklung

Shneiderman und Plaisant [SP10] beschreiben ein Modell erfolgreicher Benutzerschnittstellenentwicklung basierend auf folgenden vier Säulen, wie in Abbildung 2.14 dargestellt:

- Anforderungen an die Benutzerschnittstelle (*User Interface Requirements*)
- Richtliniendokumente und -prozesse (*Guidelines, Documents & Processes*)
- Softwarewerkzeuge für die Benutzerschnittstellenentwicklung (*User Interface Software Tools*)
- Expertenbeurteilung und Usability-Tests (*Expert Reviews & Usability Testing*)

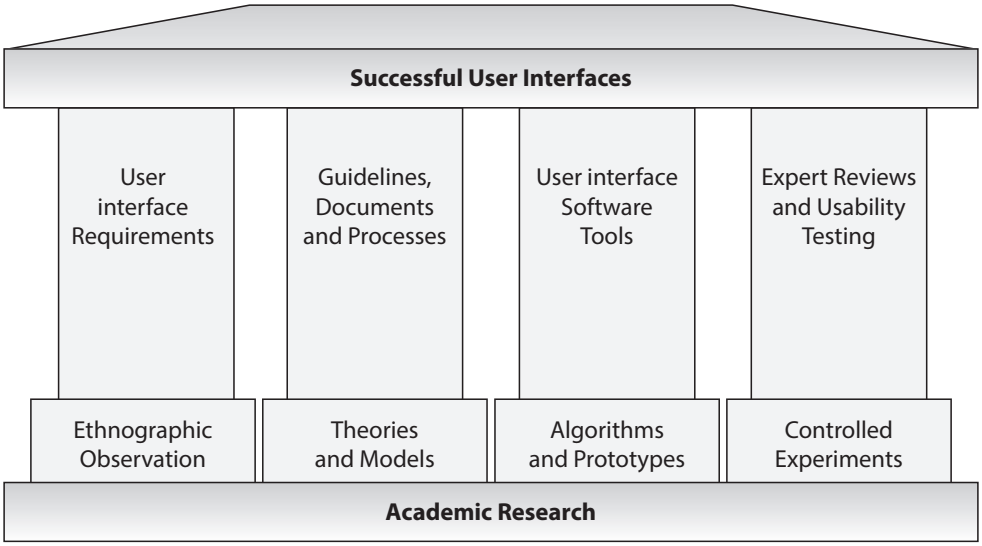


Abbildung 2.14.: Vier Säulen erfolgreicher Benutzerschnittstellenentwicklung (nach [SP10]).

Die Säule **User Interface Requirements** beinhaltet Methoden, um eine klare Definition der Benutzer oder Benutzergruppen zu erhalten sowie eine Zusammenstellung der Aufgaben, die sie über die Benutzerschnittstelle erfüllen müssen [SP10]. Hierzu gehören etwa ethnografische Analysen der Zielgruppe, bei Benutzerschnittstellen im Automobilbereich zum Beispiel der Fahrer. Außerdem muss der Grad der gewünschten Automatisierung entschieden werden, also welche Aufgaben vom System erledigt werden sollen und welche explizit der Benutzer erledigt.

Die Säule **Guidelines, Documents & Processes** hebt die Wichtigkeit hervor, Richtlinien für alle Beteiligten am Entwicklungsprozess bereitzustellen. Dazu gehören etwa ein einheitliches Glossar verwendeter Begriffe, Regeln für Gestaltung und Anordnung von Elementen der Benutzerschnittstelle sowie Informationen über die überbetriebliche Identität (CI, engl. für *Corporate Identity*), sofern vorhanden und gewünscht. Die Einhaltung solcher Richtlinien und Prozesse sollte für alle Beteiligten verbindlich gelten. Da auch hier iterative Anpassungen nötig sein können, ist es aber genauso wichtig, einen klaren Prozess für eventuell benötigte Ausnahmen und Aktualisierungen oder Erweiterungen bereitzuhalten.

Die Säule **User Interface Software Tools** besteht aus einem Satz von Werkzeugen, die bei der Entwicklung von Benutzerschnittstellen unterstützen. Dies können Softwarewerkzeuge für Spezifikationen oder die Erstellung von Prototypen sein. Wichtig ist es, früh eine simulierbare, realistische Version von Teilen der Benutzerschnittstelle zu haben, um falsche Vorstellungen zu vermeiden, die mit kostspieligen Änderungen am System zu einem späteren Zeitpunkt verbunden sein können [SP10].

Die letzte wichtige Säule für erfolgreiche Benutzerschnittstellenentwicklung ist der Einbezug von Experten und Endnutzern: **Expert Reviews & Usability Testing**. Dies geschieht etwa über Befragungen, Beobachtungen und Nutzerstudien, denen bestimmte Anwendungsszenarien zu Grunde liegen. Wichtig ist, dass solche Untersuchungen bereits von Anfang an stattfinden. So können erste Papier-Prototypen bereits mit einigen Experten, die nicht direkt an der Entwicklung beteiligt sind, besprochen werden und Nutzerstudien schon auf Basis früher Prototypen durchgeführt werden.

Im folgenden Abschnitt wird auf grundlegende Methoden und Werkzeuge bei der Benutzerschnittstellenentwicklung eingegangen. Im Vier-Säulen-Modell sind damit vorrangig die beiden mittleren Säulen *Richtliniendokumente und -prozesse* und *Softwarewerkzeuge für die Benutzerschnittstellenentwicklung* abgedeckt.

### 2.4.3. Methoden und Werkzeuge

#### Gestaltungs- und Entwicklungsrichtlinien

Richtlinien bezüglich Gestaltung und Anordnung der Benutzerschnittstellenelemente oder sonstiger Prozesse müssen bereits früh bei der Entwicklung festgelegt sein. Oftmals

existieren einige Richtlinien bereits zum Start des Prozesses, etwa wenn Gestaltungsrichtlinien teilweise aus früheren Projekten übernommen werden. Für die Richtliniendokumente sind Konsistenz und Vollständigkeit gefordert, insbesondere wenn es sich um eine arbeitsteilige Organisation handelt. Beispiel: Die Gestaltungsrichtlinien werden von Designern erstellt, Richtlinien oder Konzepte für Interaktionen von Psychologen oder Ergonomen.

Die folgenden vier Fragen dienen als Basis und müssen beantwortet werden können, um Richtlinien für die Benutzerschnittstellenentwicklung erfolgreich durchzusetzen:

- Wie werden Beteiligte über die Richtlinien unterrichtet?
- Wer kontrolliert die Durchsetzung?
- Wer entscheidet über Ausnahmen?
- Wie häufig können Richtlinien angepasst werden?

Shneiderman und Plaisant nennen diese Basis “Die vier E”, die für die englischen Begriffe *Education*, *Enforcement*, *Exemption* und *Enhancement* stehen [SP10].

#### Softwarewerkzeuge

Während der Benutzerschnittstellenentwicklung muss Klarheit über die Art der Beschreibung, Begriffe und Notationen herrschen. Unter Verwendung der Gestaltungs- und Entwicklungsrichtlinien müssen sowohl relevante Zustände der Benutzerschnittstelle, Ein- und Ausgabegeräte als auch mögliche Interaktionen spezifiziert werden. Die Verwendung von Softwarewerkzeugen hängt stark vom Entwicklungsstand und Ziel einer möglichen Implementierung ab. So kann für die frühen Phasen ein beispielhafter Interaktionsablauf oder die Zusammenstellung von grafischen Elementen auf prototypische Weise erstellt werden. Für illustrative Zwecke kann es hier ausreichen, mit Grafik- und Präsentationswerkzeugen (Microsoft PowerPoint, Adobe Photoshop oder Illustrator oder ähnlichen) zu arbeiten. Funktionierende Prototypen können etwa mit Autorenwerkzeugen (Adobe FlashBuilder, Microsoft Blend oder ähnliche) erstellt werden. Für Umsetzungen können natürlich auch Entwicklungsumgebungen wie Microsoft Visual Studio oder Eclipse[9] mit entsprechenden Einstellungen oder Erweiterungen verwendet werden[10]. Soll

[9] http://www.eclipse.org, aufgerufen am 27.10.2015
[10] Die genannten Firmen- und Produktnamen sind registrierte Markenzeichen

die entwickelte Benutzerschnittstelle außerdem für die tatsächliche Umsetzung in Form einer Spezifikation weitergegeben werden, können auch Softwarewerkzeuge für die Spezifikation relevant sein. Hier gibt es neben Werkzeugen aus der Softwaretechnik, sogenannten CASE-Tools (engl. für *Computer Aided Software Engineering*), inzwischen auch einige Softwarewerkzeuge, die sich auf Benutzerschnittstellenspezifikation spezialisiert haben. Bei solchen Werkzeugen wird Wert darauf gelegt, von einer konkreten Implementierung in einer Programmiersprache zu abstrahieren. Dadurch kann das resultierende Ergebnis, die Spezifikation, in Form eines *Modells* in weiteren Schritten für die konkrete Umsetzung verwendet werden. Ein Überblick über Methoden und Werkzeuge für eine derartige Modellierung wird in Kapitel 3 gegeben.

#### Formale Methoden

Ein systematisches Vorgehen mit Entwurf, Implementierung, Evaluierung, Verbesserung und Wartung wird bei der Entwicklung von Benutzerschnittstellen aus zwei unterschiedlichen Blickwinkeln betrachtet: Psychologie und Informatik. Dies führt dazu, dass eine der beiden Richtungen schwerpunktmäßig eingeschlagen wird, die aber von der jeweils anderen Disziplin wenig anerkannt wird [HT09]. Die Verbesserung von Entwicklungsmethoden wird dadurch häufig behindert. Die Verbesserungsvorschläge sind zu spezifisch um generelle Anwendung finden zu können, zu abstrakt für eine Anpassung auf konkrete Probleme oder sie kämpfen mit dem Dilemma, dass sie sich auf Verhaltensaspekte beziehen ohne auf die Gestaltung einzugehen obwohl die Gestaltung zuerst stattfindet und den effektiven Nutzen eines Systems stark beeinflussen kann [HT09].

Der Einsatz formaler Methoden, das heißt präzise Notationen und mathematische Modelle, kann dabei helfen, Beiträge aus den beteiligten Bereichen bei der Benutzerschnittstellenentwicklung klar und für alle nachvollziehbar zu beschreiben, wiederverwendbar zu gestalten und gegeneinander abzugrenzen. Die Stärken formaler Methoden liegen vor allem in ihrer Genauigkeit und Eindeutigkeit und der Anwendbarkeit formaler Überprüfungen oder Manipulationen [SG09]. So können etwa automatische Verifikationen von Benutzerschnittstellen durchgeführt werden. Nachteil im Bereich der Mensch-Computer Interaktion ist zum Einen, dass nur sehr wenige formale Methoden darauf eingehen, Dinge aus der Sicht des Nutzers zu beschreiben. Zum Anderen wird kritisiert, dass sie sich nicht vollständig eignen, konzeptuelle Gedanken angemessen zu beschreiben [SG09].

Für die Benutzerschnittstellenentwicklung muss abgewägt werden, wann Formalisierung stattfinden kann. Für stark abgegrenzte Teilaufgaben eignet sie sich, für andere Aufgaben muss möglicherweise auf konzeptuelle Modellierung mit mehr Freiheiten zurückgegriffen werden [SG09]. Sobald eine automatische Weiterverarbeitung der Modelle erforderlich ist oder die Modelle in einen klar vorgegebenen Prozess einfließen, ist ein entsprechender Formalisierungsgrad während der Entwicklung von Bedeutung. Diese Aspekte sind bei der Entwicklung von Benutzerschnittstellen im Automobilbereich besonders gegeben. Der folgende Abschnitt geht daher auf diesen Anwendungsbereich genauer ein.

### 2.4.4. Benutzerschnittstellenentwicklung im Automobilbereich

Im Automobilbereich spielen klare Prozesse und Arbeitsschritte sowie die Möglichkeit der Überprüfung einzelner Teile vor der Auslieferung eines Fahrzeuges eine bedeutende Rolle. Die lebenswichtige Anforderung an die Sicherheit im Straßenverkehr stellt die Entwickler der Infotainmentsysteme im Auto vor besondere Herausforderungen. Dieser Abschnitt geht auf die Entwicklung von Benutzerschnittstellen im Automobilbereich und die damit verbundenen Herausforderungen genauer ein und gibt einen Überblick über Methoden und Werkzeuge, die dort eingesetzt werden.

#### Anforderungen an Benutzerschnittstellen im Automobil

Trotz der Komplexität und des breiten Angebots bleibt die Interaktion mit Infotainmentanwendungen im Fahrzeug immer eine Nebenaufgabe. Dem Wunsch, mehr Infotainment im Fahrzeug zu haben, stehen die wichtigen Anforderungen an die besondere Interaktionsumgebung Automobil entgegen. Die kritische primäre Fahraufgabe und die wichtigen sekundären Aufgaben, die während des Fahrens erledigt werden müssen, haben stets höhere Priorität. Die Bedienung tertiärer Aufgaben während des Fahrens unterliegt gewissen Anforderungen, die generelle Auswirkungen auf die Gestaltung einer Benutzerschnittstelle im Automobil haben, wie im *European Statement of Principles on HMI for in-vehicle information and communication systems* (ESoP)[Eur99] oder dem Äquivalent für Nordamerika, dem *Statement of Principles, Criteria and Verification Procedures on Driver Interactions with Advanced InVehicle Information and Communication Systems* der *Alliance of Automobile Manufacturers* (AAM Guidelines) [All06] aufgeführt. Dabei handelt es sich um Richtlinien, denen sich viele Automobilhersteller verpflichtet haben. So sollen zum Beispiel Interaktionen unterbrechbar sein, keine unkontrollierbaren Töne

oder bewegte Bilder enthalten oder zeitkritische Antworten erfordern. Ebenso sollen allgemeine Standards zu Lesbarkeit, Symbolen, Abkürzungen und etablierte Gestaltungsmerkmale eingehalten werden. Der ISO-Standard 9241-110 stellt ergonomische Standards für die Dialoggestaltung in interaktiven Systemen zusammen, die in diesem Zusammenhang ebenfalls beachtet werden sollten. Dazu gehören geeignete Hilfen und Rückmeldungen, Konsistenz und Anpassung an den Arbeitskontext, in diesem Fall an unterschiedliche Fahrsituationen, die Minimierung unnötiger Interaktionen sowie eine minimale Erlernzeit, ebenso wie eine besonders im Fahrzeug anzustrebende intelligente Dialoggestaltung zur Fehlervermeidung seitens des Benutzers.

Die besondere Stellung des Interaktionsraums Automobil wird deutlich, wenn man einen Blick auf etablierte Einordnungen von Systemen und deren Anforderungen an die Gebrauchstauglichkeit wirft. So listen Shneiderman und Plaisant [SP10] folgende Systeme auf, die jeweils unterschiedliche Rahmenbedingungen für Ausmaß und die Ausrichtung der Gebrauchstauglichkeit mitbringen:

- Für das menschliche Leben sicherheitskritische Systeme
- Industrielle und kommerzielle Systeme
- Heim- und Unterhaltungssysteme
- Explorative, kreative und kollaborative Systeme
- Soziotechnische Systeme

Aufgrund des besonderen Interaktionsraums im Automobil, insbesondere während der Fahrt, spielen sowohl die Kategorie der sicherheitskritischen Systeme eine Rolle sowie wegen der zunehmenden Unterhaltungsanwendungen auch die Kategorie der Unterhaltungssysteme. Dies sind zwei Kategorien, die sich bei den meisten Systemen gegenseitig ausschließen. Das schafft die besondere Situation im Automobilbereich, die sich auch in den Entwicklungsprozessen niederschlägt. Auf der einen Seite stehen klassische mehrjährige Entwicklungszyklen des sicherheitskritischen Automobils inklusive der entsprechenden Software in Steuergeräten oder für Assistenzsysteme. Auf der anderen Seite steht der Bedarf nach leichtgewichtigen, modernen Infotainment-Systemen. Die damit in Verbindung stehenden Besonderheiten bei den Entwicklungsprozessen im Automobilbereich werden im Folgenden aufgegriffen.

### Besonderheiten bei Entwicklungsprozessen im Automobilbereich

In der Automobilindustrie ist es üblich, dass über mehrere Vorserien entwickelt wird, aus denen unterschiedliche Entwicklungsstände hervorgehen. Dabei sind unterschiedliche Organisationsbereiche und auch Zulieferer eingebunden. Je mehr Teilnehmer am Prozess beteiligt sind, desto schwieriger ist die Koordination von Teilergebnissen. Für die Benutzerschnittstellenentwicklung sind insbesondere die jeweiligen Dokumente oder Formate, um Ergebnisse zu kommunizieren, für die Teilnehmer unterschiedlich. Wie in den vorigen Abschnitten bereits erwähnt, denkt und handelt ein Designer oder ein Psychologe nicht mit dem gleichen Fokus wie ein Programmierer. Dennoch sind die Sicht- und Denkweisen aller Teilnehmer für eine qualitativ hochwertige Benutzerschnittstelle ausschlaggebend.

In iterativen Prozessen, die im Bereich der Benutzerschnittstellenentwicklung üblich sind, sind alle Teilnehmer an den jeweiligen Iterationen beteiligt und wichtige Entscheidungen werden zusammen getroffen. Im Automobilbereich ist dies nicht immer einfach möglich. Dies liegt zum Einen an Organisationsstrukturen, die häufige Absprachen aller Beteiligten kaum vorsehen. Zum Anderen wird ein ganzheitlicher Prozess durch das Fehlen einheitlicher Richtliniendokumente und Softwarewerkzeuge erschwert, die wichtige Säulen erfolgreicher Benutzerschnittstellenentwicklung darstellen, wie in Abschnitt 2.4.2 erwähnt.

Die Entwicklungszeit eines Automobils ist im Vergleich zu den Entwicklungszeiten im Bereich von Heimanwendungs-Software um ein Vielfaches länger. Die Entwicklung der Benutzerschnittstellen im Fahrzeug dauert entsprechend lange, da sie von der darunterliegenden Architektur, Ein- und Ausgabegeräten und Funktionsumfang des Fahrzeugs abhängig ist und diese Systeme, Geräte und Bauteile entsprechend im Gesamtprozess verankert sein müssen. Das führt häufig dazu, dass während der Entwicklung eher in unabhängigen Teilsystemen anstatt in einem gesamten, integrierten Bedien- und Anzeigekonzept gedacht wird. Die separate Entwicklung von Bordcomputer, Unterhaltungssystem und Sprachdialogsystem führt ohne klare Absprachen zu Inkonsistenzen beim Bedien- und Anzeigekonzept.

### Methoden und Werkzeuge

Die Erstellung von Spezifikationen zur Übergabe an Zulieferer oder interne Auftragnehmer für die Serienentwicklung spielt durch die oben beschriebenen Randbedingungen im Automobilbereich eine große Rolle bei der Benutzerschnittstellenentwicklung. Designer und Psychologen arbeiten häufig den Spezifikateuren zu. Methoden und Werkzeuge, die im Automobilbereich verbreitet sind, fokussieren daher in erster Linie die Unterstützung der Spezifikation. Schnelle Iterationen sind während der Entwicklung im Prozess kaum vorgesehen. Dadurch entsteht der Wunsch, dass alle Prozessbeteiligten mit den gleichen Werkzeugen oder optimal zueinander passenden Werkzeugen arbeiten, um Abstimmungen der Teilergebnisse reibungslos zu ermöglichen. Einige Softwareunternehmen bieten Werkzeuge an, die genau dies ermöglichen sollen, wie etwa Elektrobit[11] mit EB GUIDE Studio.

Ergebnisse von Designern, wie Grafiken, Bilder oder 3D-Modelle, können in EB GUIDE Studio importiert und zu unterschiedlichen Benutzerschnittstellenelementen kombiniert werden. Diese Elemente bieten die Basis für die Erstellung von sogenannten *View States*, das sind Ansichten einer konkreten Komposition von Benutzerschnittstellenelementen. Über Zustandsautomaten wird dann spezifiziert, wie die *View States* und andere relevante Zustände zueinander in Beziehung stehen und welche Ereignisse zu welcher Zeit zu einem Übergang von einem Zustand zu einem anderen führen. EB GUIDE Studio bietet auch die Möglichkeit, die so spezifizierten Benutzerschnittstellen zu simulieren. Es können die Ein- und Ausgabegeräte, also etwa das zentrale Bedienelement und mehrere Displays, spezifiziert werden und die benötigten Ereignisse, wie ein Tastendruck oder das Drehen des Bedienelements, simuliert werden. Dadurch erhält der Entwickler einen realen Eindruck der spezifizierten Benutzerschnittstellen während der ganzen Spezifikationszeit. Abbildung 2.15 zeigt ein einfaches Modell mit grafischer Repräsentation in EB GUIDE Studio.

Durch die besonderen Entwicklungsprozesse im Automobilbereich, in denen sich nicht nur mit internen, anderen Verantwortlichen der Fahrzeugentwicklung, wie Bauteil- oder Baureihenverantwortlichen, sondern insbesondere auch Zulieferfirmen abgestimmt und die Entwicklung der Benutzerschnittstelle auf diese Prozesse zugeschnitten werden muss, wurde die Notwendigkeit von domänenspezifischen Werkzeugen wie EB Guide Studio erst erzeugt. Andere Beispiele von Werkzeugen, die in diesem Zusammenhang

[11] Elektrobit Automotive, http://www.elektrobit.com, aufgerufen am 27.10.2015

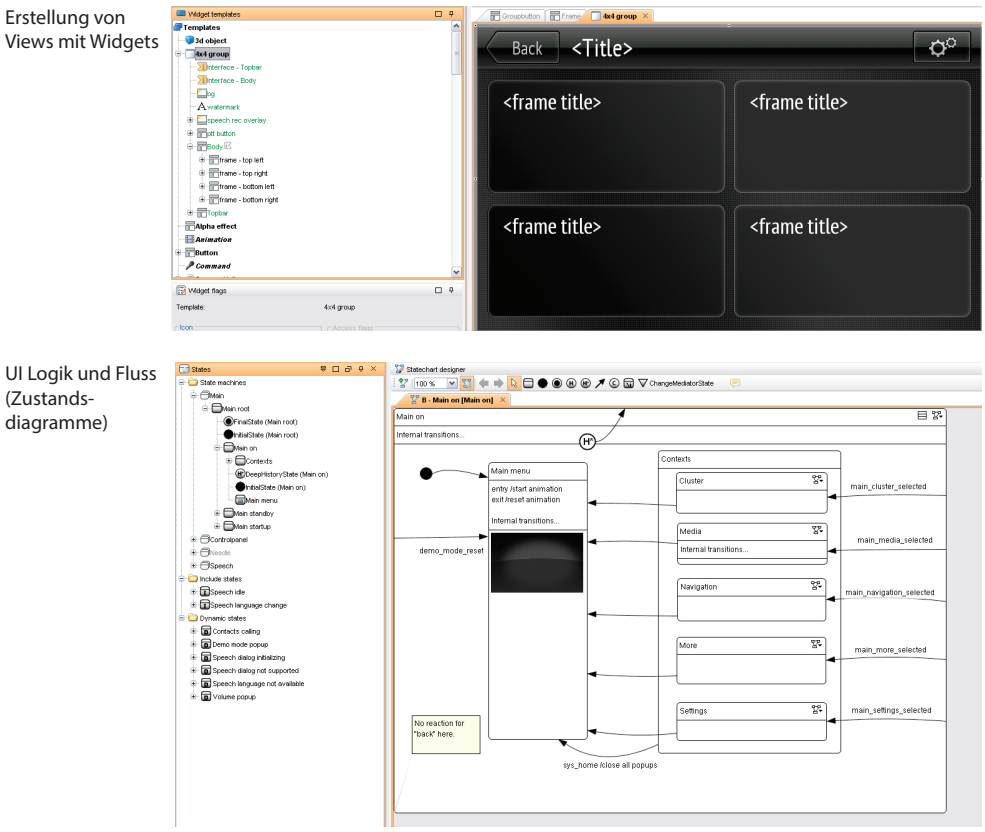


Abbildung 2.15.: Auszug aus der Software EB GUIDE Studio, die begleitend zum Förderprojekt *AutomotiveHMI* (`www.automotive-hmi.org`) verwendet wurde. Oben: Die Erstellung von *Views* durch *Widgets*. Unten: Ablaufsteuerung über Zustandsdiagramm.

eingesetzt werden, sind Vaps XT der Firma Pregasis[12], das für die Entwicklung sicherheitskritischer Mensch-Maschine Schnittstellen optimiert wurde, oder auch verschiedene aus der Informatik bekannte Modellierungswerkzeuge für die Modellierung mit standardisierten Modellierungssprachen, wie der UML.

In diesem Zusammenhang haben sich von 2009 bis 2013 die Partner des Förderprojekts *Automotive HMI*[13] zusammengefunden, um gemeinsam für die Automobilindustrie eine systematische Betrachtung und Erarbeitung eines modellbasierten Ansatzes durchzuführen. Die Ergebnisse wurden öffentlich zugänglich gemacht, insbesondere die Vorbereitung der Standardisierung, wie etwa von Hess et al. [HGM+12]. Abbildung 2.16

[12] Pregasis, http://www.presagis.com/, aufgerufen am 27.10.2015
[13] Förderprojekt: Automotive HMI, http://www.automotive-hmi.org, aufgerufen am 27.10.2015

illustriert die betrachteten Elemente in einer Ansicht *HMI View*. Ein gemeinsames Format *Infotainment Specification Format* (ISF) wird ebenfalls über die Webseite mit einer freizügigen, unentgeltlichen Lizenz angeboten, um angewandt und weiter entwickelt zu werden. Die vorliegende Arbeit entstand in enger Kooperation mit dem Förderprojekt. Allerdings war die Voraussetzung für die Entwicklung des Konzept dieser Arbeit, dass kein besonderer einheitlicher Prozess, wie es das Ziel des Förderprojekts war, Voraussetzung für die spätere Integration der Modelle sein soll. Der Integrationsprozess kann aber durch einen gut abgestimmten Entwicklungsprozess begünstigt werden.

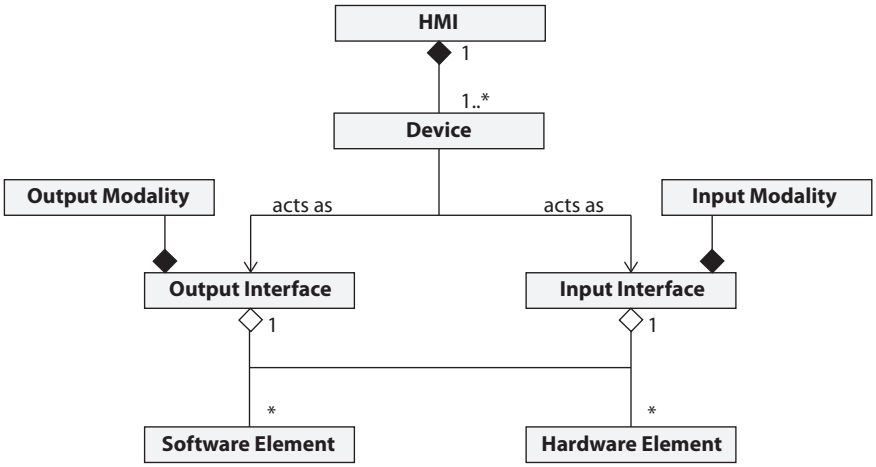


Abbildung 2.16.: Illustration zur *HMI View* aus der systematischen Betrachtung und Erarbeitung eines modell-basierten Ansatzes für die HMI-Entwicklung im Automobilbereich [HGM+12].

Die Arbeiten im Förderprojekt gehen nur teilweise auf Ansätze zur Einbindung neuer, komplett unbekannter Benutzerschnittstelleninhalte, die von externen Geräten oder aus dem Web kommen, ein. Hier liegt ein Fokus für die Entwicklung des Konzeptes dieser Arbeit.

## 2.5. Fazit

In diesem Kapitel wurden die Historie, wichtige Begriffe sowie grundlegende Zusammenhänge der Mensch-Computer Interaktion eingeführt. Verschiedene Arten von Benutzerschnittstellen und typische Einsatzszenarien wurden erläutert. Daraus wurde das Ziel für

zukünftige Systeme abgeleitet, Benutzerschnittstellen perfekt an die Sinne des Nutzers sowie an die Bedingungen der Umgebung, in der das System eingesetzt wird, anzupassen. Um dieses Ziel zu erreichen, muss bei der Entwicklung von Benutzerschnittstellen mehr Komplexität bewältigt werden. Verschiedene Modalitäten, unterschiedliche Nutzereigenschaften und -vorlieben sowie Umgebungsbedingungen müssen einbezogen werden. Für die Benutzerschnittstellenentwicklung wurde dafür ein Überblick über die grundlegenden Methoden und Werkzeuge gegeben, um diese wichtigen Faktoren von Anfang bis Ende der Entwicklung einzubeziehen. Der Einsatz formaler Methoden als Beitrag für konsistente, eindeutige Teilergebnisse wurde dabei diskutiert. Der Formalisierungsgrad bei der Beschreibung von Benutzerschnittstellen hängt von der Art des Entwicklungsprozesses und der beteiligten Rollen ab. Auf der einen Seite eignen sich formale Methoden weniger bei konzeptuelleren Arbeiten. Auf der anderen Seite ermöglichen formalere Methoden die automatische Weiterverarbeitung von Teilergebnissen, etwa um Verifikationen durchzuführen oder Teile von Benutzerschnittstellen auf der Basis klarer Regeln zu generieren.

Die Bedeutung von Infotainmentanwendungen und entsprechenden Benutzerschnittstellen im Automobilbereich hat immer weiter zugenommen. Dieser Bereich wurde daher als Anwendungsbeispiel für diese Arbeit eingeführt. Es wurde ein Rückblick auf die Historie des Interaktionsraums Automobil sowie typische Bedien- und Anzeigekonzepte im Fahrzeug gegeben. Die Entwicklungsprozesse von Benutzerschnittstellen im Automobil werden von den besonderen Anforderungen, Sicherheitsrelevanz und Lebensdauer eines Automobils, sowie den bestehenden Prozessen in der Automobilindustrie beeinflusst. Modellierung von Benutzerschnittstellen hat sich für die oft formalen Anforderungen in diesem Bereich durchgesetzt, wie der Einsatz kommerzieller Werkzeuge für die modell-basierte Entwicklung sowie auch das Förderprojekt *Automotive HMI* mit einer Modellierungssprache als Ergebnis zeigen. Der Ansatz der Interaktionsmodellierung, der im Verlauf dieser Arbeit weiter erarbeitet wird, eignet sich daher besonders gut für die Anwendung im Bereich von Benutzerschnittstellen im Automobil.

# 3

# Modellierung von Benutzerschnittstellen

Im vorhergehenden Kapitel wurde die Eignung von Modellierung zur Benutzerschnittstellenentwicklung insbesondere für Benutzerschnittstellen im Automobilbereich hervorgehoben. Modellierung hat sich dort vor allem bewährt, um formalen Anforderungen gerecht zu werden. In diesem Kapitel wird das Thema der Benutzerschnittstellenmodellierung daher weiter vertieft, um die relevanten Modellarten hierbei herauszustellen. Dieses Wissen wird für die weitere Erarbeitung des Konzepts zur Interaktionsmodellierung in dieser Arbeit benötigt. Zunächst wird im Folgenden ein Überblick über modellbasierte und modellgetriebene Entwicklung gegeben, danach werden die verschiedenen Modellarten zur Benutzerschnittstellenentwicklung vorgestellt und Ansätze für die Modellierung analysiert und bewertet.

## 3.1. Überblick

Die Bildung von Modellen ist ein Kernkonzept in der Informatik, um Komplexität zu bewältigen, Probleme auf mehrere einfachere Teilprobleme zu reduzieren und unterschiedlichen Rollen unterschiedliche Sichten in einem Entwicklungsprozess zu ermöglichen

[MHP00, KB08]. Die Modellierung von Systemen ist im Bereich der Softwareentwicklung weit verbreitet. Komplexe Sachverhalte eines Softwaresystems werden abstrakt und für den Menschen verständlich in Form von Modellen dargestellt. Aufbau, Funktionsweise und Zusammenhänge zwischen Teilen des zu entwickelnden Systems werden somit vor der eigentlichen Implementierung spezifiziert. In der Softwareentwicklung hat sich vor allem die Unified Modeling Language (UML) [OMG09] durchgesetzt, die in der Praxis in vielen Projekten erfolgreich eingesetzt wird.

Mit steigender Komplexität werden auch für die Entwicklung der Benutzerschnittstellen vermehrt Ansätze der Modellierung angewandt, die in der akademischen Welt große Beachtung finden (Cameleon Projekt [CAM04], Dissertationen [Mel10][Mei10][Sta08][Tra02], Bücher [MSB11][Pat00]). In der Industrie haben sich Modellierungsansätze für die Entwicklung von Benutzerschnittstellen nur wenig durchgesetzt [Sch07a]. Ein Grund dafür liegt in der fehlenden Unterstützung durch geeignete Tools [DFHS05]. Auch gibt es sehr viele unterschiedliche Modellierungssprachen für Benutzerinteraktion, die sich noch nicht ausreichend im praktischen Einsatz bewähren konnten.

Werden Modelle bei der Entwicklung eingesetzt, wird dafür häufig der Begriff der *modellbasierten Entwicklung* verwendet. Durch die Bildung von Modellen können Teile der Benutzerschnittstelle zunächst auf einer höheren Abstraktionsebene beschrieben werden. In späteren Schritten können Details ausgearbeitet und hinzugefügt werden. Die Modelle dienen etwa der Dokumentation und Kommunikation zwischen Entwicklern, die die Benutzerschnittstelle darauf basierend umsetzen. Werden die Modelle nicht nur als Basis für die Kommunikation und für die Entwicklung verwendet, sondern zur automatischen Erzeugung von Quellcode eingesetzt, wird der Begriff der *modellgetriebenen Entwicklung* verwendet [LSHA07].

### 3.1.1. Modellbasierte Entwicklung: Kritik und Potenzial

Modellbasierte Ansätze für Benutzerschnittstellenentwicklung wurden in mehreren Arbeiten analysiert. Stanciulescu [Sta08] hat die wichtigsten Probleme zusammengefasst. So steht einer hohen Schwelle zum Erlernen der neuen Technik eine stark begrenzte Mächtigkeit bezüglich der damit beschreibbaren Benutzerschnittstellen entgegen. Bei der Ableitung von Benutzerschnittstellen aus Modellen besteht außerdem das Problem der schwierigen Vorhersagbarkeit, wie genau zwischen Modell und finaler Benutzerschnittstelle abgebildet wird. Einige Modellierungsarten sind außerdem stark an das

darunterliegende System gebunden und können nicht in andere Systeme exportiert werden.

Auf der anderen Seite bietet ein modellbasierter Ansatz Vorteile. So sind vor allem die Wiederverwendbarkeit von Modellen sowie Konsistenz durch formale Vorgehensweise zu nennen. Außerdem ermöglicht ein modellbasierter Ansatz einen nutzerzentrierten Entwicklungszyklus, der jedem Beteiligten eine angemessene Abstraktion bietet und durch das Wegfallen von Details in frühen Phasen exploratives, arbeitsteiliges Vorgehen fördert [RJB99]. Die Sicht der Designer kann sich zum Beispiel auf das Nötigste beschränken, um Designentscheidungen zu erarbeiten, und Softwareentwickler können auf der Ebene von Komponenten und Schnittstellen denken.

### 3.1.2. Modellgetriebene Entwicklung: Kritik und Potenzial

Bei der modellgetriebenen Entwicklung, also wenn Modelle auch zur automatischen Erzeugung von Quellcode eingesetzt werden, wird vor allem kritisiert, dass die vollautomatisch erzeugten Benutzerschnittstellen von unzureichender Qualität sind und das Ergebnis der Generierung nicht vorhersehbar ist [MHP00]. Auch ist die Menge an verschiedenen erzeugbaren Benutzerschnittstellenelementen durch den Prozess begrenzt. Eine händische Programmierung von Benutzerschnittstellen bietet mehr Freiheiten und daher den Vorteil der Flexibilität gegenüber rein modellgetriebenen Ansätzen.

Das Potenzial eines konsequenten modellgetriebenen Prozesses liegt in der Trennung von Implementierung der Funktionalität und Erstellung der Interaktionselemente. Die Funktionalität muss bei einer Portierung auf eine andere Plattform nicht reimplementiert werden, was zu einer Aufwandsreduktion führt [MHP00]. Die Vorteile basieren auf der Annahme eines verlässlichen, automatischen Generierungsprozesses. Dieser muss für den jeweiligen Einsatzzweck hinreichend demonstriert werden.

## 3.2. Modellarten bei der Benutzerschnittstellentwicklung

Im Rahmen der Entwicklung von Benutzerschnittstellen sind einige grundlegende Arten von Modellen zu unterscheiden. Dabei sind vor allem folgende Modellarten, die sich gegenseitig ergänzen, von Bedeutung [Sch07a, PE99, Pue97]:

- *Aufgabenmodell*: Beschreibt die Ziele des Nutzers und Wege, diese zu erreichen.

- *Domänenmodell*: Enthält durch die Domäne bereitgestellte Objekte und Funktionen.
- *Dialogmodell*: Beschreibt das Verhalten des Nutzers und des Systems im Dialog.
- *Präsentationsmodell*: Beschreibt Typen und Anordnung von Interaktionsobjekten, z.B. Widgets in einer grafischen Oberfläche.
- *Kontextmodell*: Gruppierung der Modelle zu Nutzer, Plattform und Umgebung:
  - *Nutzermodell*: Beschreibt Nutzer oder Gruppen von Nutzern und ihre Eigenschaften
  - *Plattformmodell*: Beschreibt verfügbare Interaktionsgeräte und deren Eigenschaften
  - *Umgebungsmodell*: Beschreibt die Umgebung, in der die Interaktion stattfindet, z.B. Ort und Umweltverhältnisse.

Im Folgenden werden die Modellarten genauer erläutert und einige Beispiele gegeben.

### 3.2.1. Aufgabenmodell

Das Aufgabenmodell kann als Kern eines modellbasierten Entwicklungsprozesses angesehen werden [Mei10]. In einem Aufgabenmodell werden die Aufgaben, die ein Nutzer mit einem System bewältigen können muss, analysiert und explizit dargestellt. Dies beinhaltet einzelne Teilaufgaben und deren hierarchische Struktur sowie Informationen über die Möglichkeiten der Abfolge von Aufgaben [CMN83, PMM97, Wel01]. Eine aufgabenzentrierte Entwicklung von Benutzerschnittstellen ist unerlässlich, da ein interaktives System nur dann nützlich ist, wenn der Benutzer es geeignet für seine Aufgaben verwenden kann. Es gibt einige Notationsformen für die explizite Modellierung von Aufgaben. Eine bekannte und weit verbreitete Sprache zur hierarchischen Modellierung von Aufgaben ist *ConcurTaskTrees (CTT)* [PMM97]. Damit können die vier Aufgabenarten *Interaktionsaufgabe*, *Nutzeraufgabe*, *Anwendungsaufgabe* und *Abstrakte Aufgabe* hierarchisch angeordnet und deren temporale Abfolge modelliert werden. Dafür stehen die acht Temporaloperatoren *Choice, OrderIndependence, Interleaving, Synchronisation, Disabling, SuspendResume, SequentialEnabling und SequentialEnablingInfo* (geordnet nach Ausführungsprioritäten von hoch zu niedrig) bereit.

Für ein einfaches Aufgabenmodell einer ToDo-Listen-Anwendung, im Folgenden *ToDo-Liste* genannt, können etwa folgende Aufgaben und Operatoren verwendet werden: Die Anwendungsaufgabe *Present entry list* beschreibt die Ausgabe der aktuellen Einträge auf der Liste durch das System. Sie wird gefolgt von der Möglichkeit einen Eintrag hinzuzufügen, repräsentiert durch die abstrakte Aufgabe *Add Entry*, sowie einen Eintrag zu löschen, repräsentiert durch die abstrakte Aufgabe *Delete Entry*. Zwischen diesen beiden abstrakten Aufgaben steht der temporale Operator *Choice* ([]), da sie unabhängig voneinander ausgewählt werden können. Die abstrakten Aufgaben werden durch *Interaktionsaufgaben* für das Hinzufügen eines Eintrages sowie Auswählen und Löschen eines Eintrages gefolgt von den jeweiligen *Anwendungsaufgaben*, um die Aktion durchzuführen, konkretisiert. Dabei handelt es sich jeweils um aufeinanderfolgend zu erledigende Aufgaben, was durch den Temporaloperator *Enabling* (>>) ausgedrückt wird. Das Modell in CTT ist in Abbildung 3.1 dargestellt.

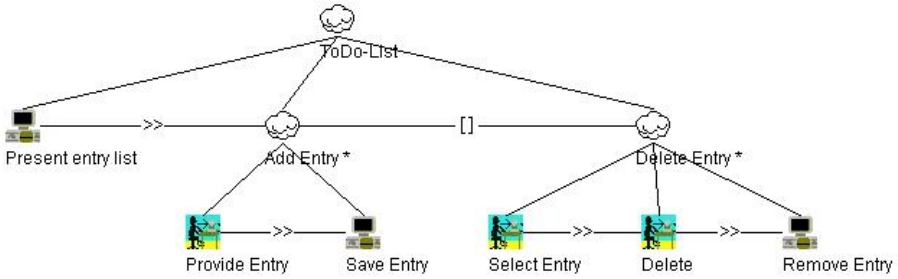


Abbildung 3.1.: Aufgabenmodell in CTT für das Beispiel *ToDo-Liste*.

### 3.2.2. Domänenmodell

Das Domänenmodell ist für das Verständnis der Applikation wichtig und kann als Erweiterung eines einfachen Datenmodells gesehen werden [Pue97]. Domänenmodelle definieren Objekte der Domäne, deren Eigenschaften und die Beziehungen zwischen ihnen. Verschiedene Notationsformen erlauben die Erstellung von Domänenmodellen. Ein Beispiel sind sogenannte *Entity-Relationship (ER) Diagramme* [Che76], die als Modelle für Daten und deren Beziehungen zueinander im Bereich von Datenbanksystemen etabliert sind. Abbildung 3.2 illustriert ein ER-Diagramm für das oben eingeführte Beispiel der ToDo-Liste. Eine *List* besteht dabei aus n *Entries*. Und ein *Entry* hat eine *ID* und einen *Text*.

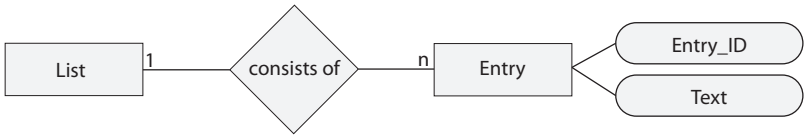


Abbildung 3.2.: Entity-Relationship Diagramm für das Beispiel *ToDo-Liste*.

### 3.2.3. Dialogmodell

Das Dialogmodell beschreibt den Dialog zwischen zwei Teilnehmern, typischerweise Mensch und Computer. Für diese Arbeit wird ein Dialog als die gesamte Interaktion zwischen zwei Teilnehmern verstanden, der aus einer logisch zusammenhängenden Menge einzelner Interaktionen besteht [Mel10]. Eine einzelne Interaktion ist dabei das aufeinander bezogene Handeln zweier oder mehrerer Teilnehmer. Das Dialogmodell muss also mögliche Handlungen des Nutzers sowie die Handlungen des Computers beschreiben und deren wechselseitige Beziehung. Das Modell kann sehr abstrakt sein und beispielsweise eine andere Sicht auf das Aufgabenmodell darstellen [Mei10]. Es kann aber auch sehr konkret sein, und Wissen über mögliche Nutzer- und Systemereignisse oder verwendete Technologien einbeziehen. Eine typische Form der Notation für Dialogmodelle sind Zustandsdiagramme, wie etwa in der UML beschrieben [OMG09]. Aber auch einige andere Dialogmodelle wurden in der Vergangenheit erfolgreich eingesetzt, wie zum Beispiel Dialognetze, Attributgrammatiken und Dialog-Templates [Pue97]. Das Dialogmodell wird im Rahmen der Modellierung von Benutzerschnittstellen häufig als Bindeglied zwischen Aufgaben- und Präsentationsmodell gesehen [Mei10].

Abbildung 3.3 stellt ein mögliches Dialogmodell für das Beispiel der ToDo-Liste als Zustandsdiagramm dar. Die Anwendung startet mit der Liste als Interaktionszustand. Hier kann, wenn gewünscht, ein Eintrag ausgewählt werden. Durch *Add* oder *Delete* wird in die Zustände *New* oder *Delete* gewechselt. *Confirm* oder *Cancel* führt zurück, wobei die jeweiligen Systemaktionen *AddEntry* oder *DeleteEntry* ausgelöst werden.

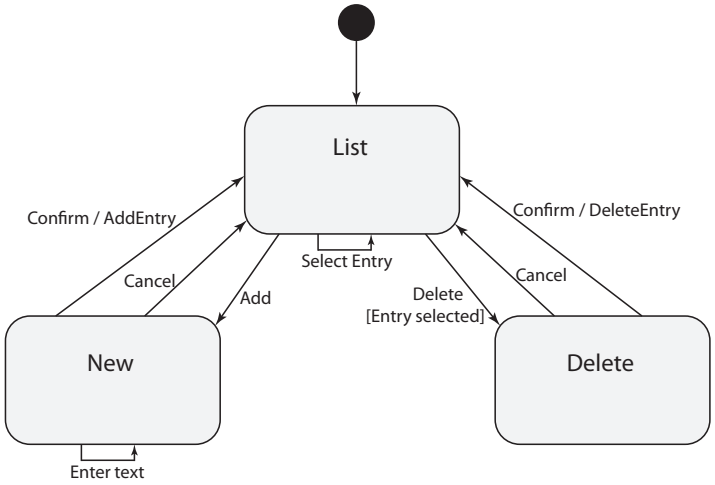


Abbildung 3.3.: Dialogmodell als Zustandsdiagramm für das Beispiel *ToDo-Liste*.

### 3.2.4. Präsentationsmodell

Das Präsentationsmodell definiert die Darstellung der Benutzerschnittstelle [Mei10]. Der Begriff der Benutzerschnittstelle wurde bereits in Abschnitt 2.2 definiert als die Zusammensetzung aller Bestandteile eines interaktiven Systems, die Informationen und Steuerelemente zur Verfügung stellen, welche für den Benutzer notwendig sind, eine bestimmte Arbeitsaufgabe mit dem interaktiven System zu erledigen. Um die Verbindung zum Dialogmodell deutlich zu machen, wird von De Melo die Benutzerschnittstelle als „Summe aller Interaktionselemente, die in einem Dialog verwendet werden können" beschrieben [Mel10, S. 8]. Ein Interaktionselement ist dabei ein Element, durch das Interaktion erfolgt.

Im Präsentationsmodell müssen die relevanten Informationen über die Interaktionselemente abgebildet werden können. Der Zusammenhang zwischen Präsentationsmodell und Dialogmodell wird von Puerta so beschrieben, dass das Präsentationsmodell die Art und Weise bestimmt, wie die Interaktionselemente in den jeweiligen Dialogzuständen erscheinen [Pue97]. Das Dialogmodell beschreibt, wann diese Zustände jeweils auftreten.

### 3.2.5. Kontextmodell

Der Kontext eines interaktiven Systems bezeichnet die jeweilige Situation, die eine Benutzerschnittstelle beeinflussen kann [HIR02]. Dafür gibt es typischerweise verschiedene Modelle, die zusammen den Kontext bilden. Schäfer [Sch07b] nennt hier *Nutzermodell*, *Plattformmodell* und *Umgebungsmodell*.

Um eine Benutzerschnittstelle zu entwickeln, die an die zu erwartenden Nutzer angepasst ist, werden in Form von Nutzermodellen typische Nutzer mit ihren Eigenschaften klassifiziert. Dies wird etwa über die Definition von Stereotypen erreicht. Das sind feste, teilweise klischeehafte Bilder von typischen Nutzern [Ric79]. Die Definition eines Stereotypen kann etwa in Form einer Tabelle mit Eigenschaften und deren Werten erfolgen.

Zusätzlich zu den Nutzern gibt ein Plattformmodell Aufschluss über die technischen Möglichkeiten der Ausführungsplattform. Die *Foundation for Intelligent Physical Agents (FIPA)* stellt dafür etwa eine Vorlage für Geräteprofile bereit[1]. Zu den Eigenschaften der Geräte gehören die Bildschirmauflösung, die Audio-Ausstattung und sonstige Hard- und Softwareeigenschaften, die für die Ausführung einer interaktiven Anwendung relevant sein können.

Ein Umgebungsmodell definiert den Kontext, der nicht direkt im Nutzermodell oder im Plattformmodell abgedeckt ist [Sch07b]. Dazu gehören etwa Ort und Position von Teilen des interaktiven Systems, Lautstärke in der Einsatzumgebung, Temperatur oder Helligkeit. Diese äußeren Faktoren beeinflussen die Gebrauchstauglichkeit eines interaktiven Systems teilweise maßgeblich. Je nach Art der Anwendung sind andere Faktoren von Bedeutung. Für eine Anwendung der virtuellen Realität (siehe Abschnitt 2.2.4) muss die Umgebung etwa in Form von 3D-Modellen vorliegen [Sch07b].

## 3.3. CAMELEON Reference Framework

Die im vorigen Abschnitt vorgestellten Modellarten haben sich in der Vergangenheit durchgesetzt, um die Entwicklung von Benutzerschnittstellen aus den verschiedenen nötigen Blickwinkeln zu unterstützen. Über konkrete Notationsformen herrscht wenig Einigkeit. Es gibt zahlreiche Modellierungssprachen für verschiedene Einsatzzwecke. Auch

[1] Foundation for Intelligent Physical Agents, FIPA, http://www.fipa.org, aufgerufen am 27.10.2015

wird nicht in jedem Entwicklerteam oder für jedes Projekt die gesamte Bandbreite an Modellen gleichermaßen verwendet. Die unterschiedlichen Anforderungen der jeweiligen Einsatzumgebung führen zu zugeschnittenen Entwicklungsprozessen, die sich möglicherweise nicht auf andere Einsatzgebiete übertragen lassen. Um dies zu vermeiden und eine solche Übertragung auf mehrere Einsatzgebiete von vornherein zu ermöglichen, ist ein einheitliches Vorgehen erforderlich. Die Entwicklung von Benutzerschnittstellen, die auf mehreren Plattformen unter Beibehaltung der Gebrauchstauglichkeit laufen, wird unter der Anforderung der *Plastizität* zusammengefasst [CCT+03]. Mit dieser Anforderung wurde im europäischen Verbundprojekt CAMELEON [CAM04] eine Meta-Architektur entwickelt, die sich im Bereich der Modellierung von Benutzerschnittstellen als *CAMELEON Reference Framework* (CRF) etabliert hat [CCT+03]. Darin werden verschiedene Abstraktionsebenen für Benutzerschnittstellenmodelle definiert: *Aufgaben und Konzepte* (T&C, engl.: Task and Concepts), *abstrakte Benutzerschnittstelle* (AUI, engl.: Abstract User Interface), *konkrete Benutzerschnittstelle* (CUI, engl.: Concrete User Interface) und *finale Benutzerschnittstelle* (FUI, engl.: Final User Interface).

Auf der Ebene der *Aufgaben und Konzepte* werden die ersten sogenannten *initialen Modelle* erstellt. Initiale Modelle werden direkt von einem Entwickler spezifiziert und sind Ausgangspunkt für mögliche Transformationen in weitere Modelle. Relevante Modelle auf dieser Ebene sind Aufgabenmodell, Domänenmodell sowie Kontextmodelle (Nutzer, Plattform, Umgebung).

Die Ebene der *abstrakten Benutzerschnittstelle* enthält eine modalitäten- und plattformunabhängige Beschreibung der Benutzerschnittstelle [CCT+03], welche unter Anwendung von Transformationsregeln aus den Modellen der Ebene *Aufgaben und Konzepte* erzeugt werden kann.

Über weitere Transformationen können Modelle für die *konkrete Benutzerschnittstelle* abgeleitet werden. Diese enthält Informationen zur Interaktionsmodalität und nutzt verfügbare Interaktionselemente, ist aber noch plattformunabhängig [CCT+03].

Die *finale Benutzerschnittstelle* wird aus der konkreten Benutzerschnittstelle für die Zielplattform generiert. Sie ist in ausführbarem oder interpretierbarem Quellcode ausgedrückt.

Abbildung 3.4 illustriert diese Referenz-Architektur mit den verschiedenen Modellen und Ebenen. *Config 1* und *Config 2* entsprechen dabei zwei unterschiedlichen Kon-

figurationen, beispielsweise zwei unterschiedlichen Plattformen in unterschiedlichen Umgebungen.

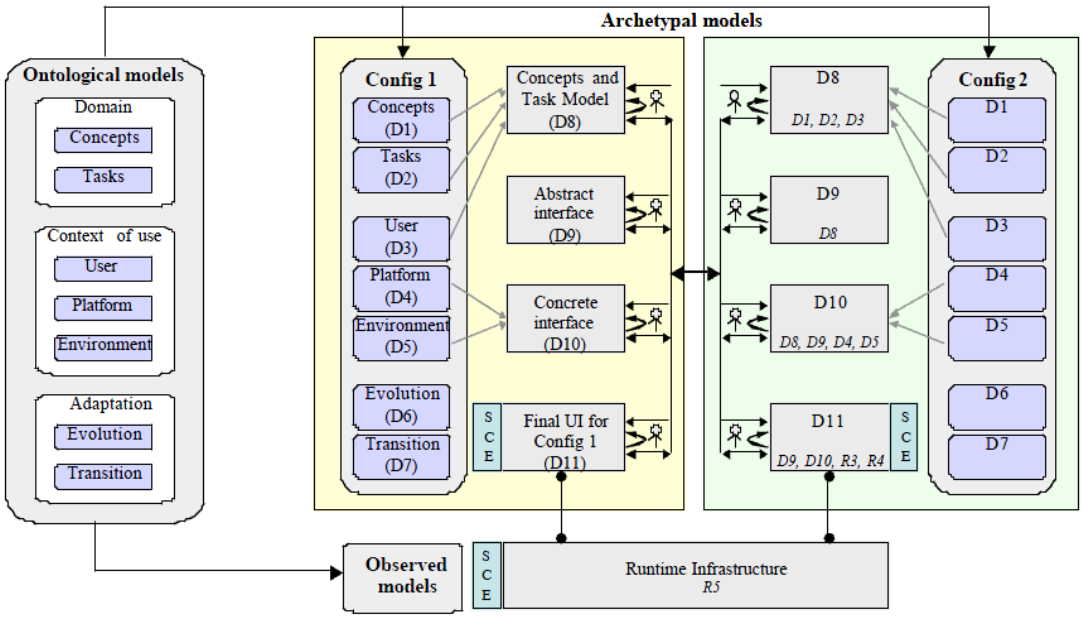


Abbildung 3.4.: Das Cameleon Reference Framework [CCT+03]

Das CRF ist so konzipiert, dass der Einstieg in die Modellierung auf jeder Ebene möglich ist [CCT+03]. Im folgenden Abschnitt werden Ansätze für die Modellierung auf den unterschiedlichen Ebenen vorgestellt.

## 3.4. Ansätze für die Modellierung

Das CAMELEON Reference Framework (CRF), wie im vorhergehenden Abschnitt vorgestellt, dient als grundlegende Einteilung verschiedener Ebenen, in denen die Modellierung von Benutzerschnittstellen erfolgt. Im Folgenden werden bestehende Ansätze aus der Aufgabenmodellierung, das ist die oberste Ebene T&C im CRF, der abstrakten Benutzerschnittstellenmodellierung, dies betrifft die Ebenen AUI und CUI im CRF, sowie spezifischer Beschreibungssprachen, dies betrifft die Ebenen CUI und FUI, zusammengefasst. Anschließend werden umfassendere Methoden, die mehrere oder alle Ebenen des CRF durchgängig abdecken sollen, beschrieben und bewertet.

### 3.4.1. Aufgabenmodellierung

Die Entscheidung, welche Aufgaben ein Nutzer mit einem System erledigen können muss, steht an erster Stelle bei der Entwicklung von Benutzerschnittstellen. Modellierungsmethoden und Notationen unterstützen die Analyse und explizite Darstellung dieser Aufgaben. Eine bekannte Methode und Notationsform sind die hierarchischen Aufgabenbäume *ConcurTaskTrees* (CTT) [PMM97], wie bereits in Abschnitt 3.2.1 aufgeführt. Eine vergleichbare Notation ist die *User Action Notation* (UAN) [HSH90], die zur formalen Kommunikation zwischen UI-Designern und UI-Programmierern entwickelt wurde. Die Aufgaben werden in Form von Tabellen notiert. Auf Basis von UAN wurde für die *Extended User Action Notation* (XUAN) die Modellierungsmächtigkeit so erweitert, dass auch temporale Operatoren unterstützt werden [GEM94]. Damit unterscheidet sich XUAN von CTT nicht in der Modellierungsmächtigkeit [Luy04], sondern vor Allem in der Art der Repräsentation der Aufgaben, die über Tabellen und nicht über grafische Syntax erfolgt. Es gibt zahlreiche weitere entwickelte Notationen und Werkzeuge zur Unterstützung der Aufgabenmodellierung, eine Vorstellung und Bewertung bietet beispielsweise Meixner [Mei10]. XUAN und CTT werden dabei als die umfangreichsten Ansätze hinsichtlich der Modellierungsmächtigkeit aufgeführt. Letztlich gilt CTT aber als bekannteste Form dank der Unterstützung durch eine grafische Repräsentation der Aufgabenhierarchie.

### 3.4.2. Abstrakte Benutzerschnittstellenmodellierung

Die Modellierung von Aufgaben beinhaltet noch keine umfangreichen Informationen über die Auslegung der Benutzerschnittstelle. Aufgabenmodellierung, wie mit CTT, ist allerdings im CRF der erste Schritt, um eine abstrakte Benutzerschnittstelle, AUI, abzuleiten. Um ein AUI zu beschreiben eignen sich Beschreibungssprachen, die typischerweise unabhängig von der Implementierung und Zielplattform sind, so genannte *User Interface Description Languages* (UIDL) [SV03]. Die Sprache Teresa XML [BCPS04] beinhaltet beispielsweise eine Repräsentation des Aufgabenmodells in CTT und enthält zusätzlich eine abstrakte Beschreibung des Verhaltens der Benutzerschnittstelle. Die *User Interface Markup Language* (UIML) [APB+99] enthält keine direkte Repräsentation des Aufgabenmodells, sie besteht aus einer abstrakten Benutzerschnittstellenbeschreibung, Abbildungen von Benutzerschnittstellenelementen auf Applikationslogik sowie Vorlagen für die Wiederverwendung bereits definierter Elemente. Als Grundlage für das Format,

in dem UIDLs abgelegt werden, ist die *eXtensible Markup Language* (XML), die für applikationsspezifische, hierarchische Strukturierung von Daten entwickelt wurde, weit verbreitet. Auch Teresa XML und UIML werden in einem XML-Format gespeichert. Weitere Hauptvertreter von UIDLs zur abstrakten Beschreibung von Benutzerschnittstellen werden von Souchon und Vanderdonckt [SV03] aufgeführt und bewertet: *Abstract User Interface Markup Language* (AUIML), *eXtensible Interface Markup Language* (XIML), *Seescoa (Software Engineering for Embedded Systems using a Component Oriented Approach) XML* sowie *Web Services Experience Language* (WSXL) sowie viele andere weniger bekannte oder eingeschränktere Ansätze. Letztlich ist die Anzahl an Beiträgen in diesem Feld sehr groß und es kann generell nicht zu einer einzigen Sprache geraten werden, da Einsatzfeld und Ziele variieren [SV03].

### 3.4.3. Spezifische Beschreibungssprachen

In dieser Arbeit wird zwischen Sprachen und Notationen für abstrakte Modellierung, wie im vorigen Abschnitt beschrieben, und Beschreibungssprachen für bestimmte Benutzerschnittstellen unterschieden, die im Folgenden als *spezifische Beschreibungssprachen* bezeichnet werden. Im Gegensatz zu abstrakten Modellierungssprachen sind spezifische Beschreibungssprachen bereits für eine bestimmte Zielumgebung zugeschnitten. Das Ergebnis kann dennoch als Benutzerschnittstellenmodell bezeichnet werden, da für die letztliche Umsetzung, also die finale Benutzerschnittstelle, häufig noch ein weiterer Schritt benötigt wird. So kann etwa eine Benutzerschnittstelle in der *Hypertext Markup Language* (HTML) beschrieben sein. Bevor sie angezeigt werden kann, muss sie allerdings noch ein Browser interpretieren. Dabei kann es auch sein, dass unterschiedliche Browser leicht unterschiedliche Interpretationen vornehmen.

Die meisten Beschreibungssprachen nutzen den Standard der XML. XML ist flexibel einsetzbar, da die zu verwendenden Elemente und deren Regeln frei über die sogenannte *Document Type Definition* (DTD) oder *XML Schema Definition* (XSL) definiert werden können. Damit lässt sich jede hierarchische Datenstruktur, so auch die Struktur von Benutzerschnittstellenelementen, in XML abbilden. Microsoft hat die XML verwendet und die *eXtensible Application Markup Language* (XAML) definiert. Damit lassen sich Benutzerschnittstellen zur Darstellung im Windows Presentation Framework (WPF) oder zur Darstellung in Silverlight[2], Microsofts freies Browser-Plugin für interaktive, animierte

[2] http://www.microsoft.com/silverlight, aufgerufen am 27.10.2015

Inhalte, erstellen. Silverlight ist Microsofts Gegenstück zum Flash-Player von Adobe[3]. Benutzerschnittstellen von Anwendungen für den Flash-Player lassen sich entsprechend mit Adobes Beschreibungssprache MXML (Eigenname, keine Abkürzung) definieren, die ebenfalls auf ein XML-Format setzt. Beispiel für eine Beschreibungssprache, die nicht auf XML basiert, ist die *Qt Modeling Language* (QML), mit der Benutzerschnittstellen für Qt-Anwendungen erstellt werden können[4]. Auch QML setzt auf eine hierarchische Strukturierung von Benutzerschnittstellenelementen und nutzt dafür geschweifte Klammern zur Schachtelung.

### 3.4.4. Umfassende, durchgängige Modellierungsmethoden

Neben der Verwendung einzelner Ansätze für einzelne Ebenen im CRF, wie zum Beispiel die Verwendung von CTT zur Aufgabenmodellierung oder QML für die finale Beschreibung der Benutzerschnittstelle, verfolgen einige Ansätze das Ziel der Durchgängigkeit über alle oder einen großen Teil der Ebenen. UsiXML [VLM+04] ist ein Beispiel eines solchen umfassenden Ansatzes. Ausgehend von einem abstrakten Modell wird durch die Definition von Kontexten sowie Transformationen und deren Anwendung ein konkretes Modell für die Zielplattform erstellt, das schließlich in der finalen Benutzerschnittstelle resultiert.

Aus der Domäne der Softwaretechnik ist die in diesem Kapitel bereits erwähnte Modellierungssprache UML bekannt, die seit 2001 standardisiert ist und sich in der Praxis der Softwaretechnik bewährt hat. De Melo [Mel10] liefert eine detaillierte Analyse und Bewertung hinsichtlich der Eignung von UML zur Modellierung von Benutzerschnittstellen im Vergleich zu anderen Ansätzen und stellt UML als Sprache mit den meisten Vorzügen bei der Erweiterbarkeit von Modalitäten, Angemessenheit, Verständlichkeit, Ausdrucksstärke, Universalität, Werkzeugunterstützung und Verbreitung heraus. Die Aufgabenmodellierung erfolgt dabei mit Aktivitätsdiagrammen, welche für die Benutzerschnittstellenbeschreibung in den Zustandsdiagrammen verwendet werden.

Mit der Aufgabenmodellierung im Kern wurde außerdem der Ansatz der Benutzungsmodellierung entwickelt, der das Aufgabenmodell um Domänenmodell, Kontextmodell, Plattform- und Benutzermodell ergänzt. Dafür wurde die Sprache *useML* entwickelt

[3] Adobe Flash-Player: http://www.adobe.com/products/flashplayer.html, aufgerufen am 27.10.2015

[4] Qt Project Website, http://www.qt.io, aufgerufen am 27.10.2015; Comment on the name QML, http://blog.qt.digia.com/blog/2009/08/21/qt-declarative-api-changes/#comment-4727, aufgerufen am 27.10.2015

[Mei10, MSB11]. Damit ist es möglich, alle relevanten Aspekte abstrakter Benutzerschnittstellen durch eine Sprache abzubilden, worauf im Erstellungsprozess der finalen Benutzerschnittstellen aufgebaut werden kann.

Die erwähnten Ansätze sind nur ein Teil der zahlreichen Beschreibungssprachen und Modellierungsmethoden auf den unterschiedlichen Ebenen des CRF. Wie einige Analysen der Methoden, z.B. in [SV03, Mel10], zeigen, überschneiden sich die Einsatzfelder der einzelnen Lösungen zwar, jede Methode hat allerdings seine Besonderheit oder teilweise auch Alleinstellungsmerkmale. Eine klare Empfehlung für eine universell einsetzbare Methode gibt es nicht, jedoch bietet die UML die meisten Vorzüge und eine sehr flexible Ausgangslage [Mel10].

## 3.5. Fazit

Die Modellierung von Benutzerschnittstellen als Methode zur Unterstützung des Entwicklungsprozesses, die modellbasierte Entwicklung, sowie als Methode zur automatischen Erzeugung von Quellcode, die modellgetriebene Entwicklung, wurde in diesem Kapitel vorgestellt. Das Potenzial der Modellierung von Benutzerschnittstellen reicht von der Wiederverwendbarkeit der Modelle im Entwicklungsprozess bis hin zur komplett automatisch generierten finalen Benutzerschnittstelle, wenn die Modelle als Basis automatisch verarbeitet und in das finale Ergebnis überführt werden können.

Die wichtigsten Modellarten für die Benutzerschnittstellenentwicklung sind Aufgabenmodell, Domänenmodell, Dialogmodell, Präsentationsmodell und Kontextmodell. Das Cameleon Reference Framework (CRF) wurde als Meta-Architektur für die Einordnung modellbasierter Ansätze auf verschiedenen Abstraktionsstufen eingeführt. Die Modellierung von Aufgaben, die ein Nutzer mit einer Interaktionsanwendung erfüllen möchte, hat sich durchgesetzt, wobei auch unterschiedliche Notationen und Abstraktionsstufen entstanden sind. *Concur Task Trees* (CTT) wurde als bekannteste Notationsform vorgestellt.

Die automatische Weiterverarbeitung von Aufgabenmodellen, um daraus etwa Präsentations- und Dialogmodelle abzuleiten, wurde in einigen Ansätzen untersucht, wie etwa bei useML oder UsiXML. Für eine Erzeugung von weiterverwendbaren Präsentations- und Dialogmodellen, die in spezifischen Kontexten verwendet werden sollen, fehlen allerdings, wenn das Aufgabenmodell Grundlage ist, entscheidende Informationen. Auf-

gabenmodelle verbergen Komplexität, die in weiteren Schritten hinzukommen muss, um das Modell der Benutzerschnittstelle zu konkretisieren. Dazu zählen Informationen über die technische Umgebung, vorhandene Interaktionsmöglichkeiten und Geräte. Oft müssen Teile eines Kontexts bereits im eigentlich unabhängigen Aufgabenmodell berücksichtigt werden, damit es weiterverarbeitet werden kann. Die Anforderungen an Aufgabenmodelle, um Startpunkt für einen universellen modellgetriebenen Prozess zur Erzeugung passender Benutzerschnittstellen zu sein, sind so umfangreich und spezifisch, dass an vielen Stellen die ursprüngliche Intention, das mentale Modell zu erledigender Aufgaben mit einem System formal zu notieren, in Frage gestellt werden muss. Ansätze, die das mentale Modell der Aufgaben nicht explizit enthalten, haben daher einige Vorteile bezüglich der automatischen Weiterverarbeitung. Nichtsdestotrotz ist die Notwendigkeit, eine Interaktion mit der Überlegung, welche Aufgaben eigentlich erledigt werden sollen, zu beginnen, unbestritten. Einige der Ansätze, wie etwa die useML, vereinen die Modellierung von Aufgaben daher mit weiteren Modellen, sodass die Aufgaben in einem gemeinsamen abstrakten Interaktionsmodell aufgehen.

Für das Ziel dieser Arbeit, eine automatische Weiterverarbeitung der Modelle bis zur Transformation in eine finale Benutzerschnittstelle zu erreichen, ist der generelle Ansatz, die notwendige Aufgabenmodellierung mit weiteren Modellen zu einem abstrakten Ausgangsmodell zu vereinen, besonders interessant und wird im Folgenden weiter entwickelt. Im Sinne optimaler Flexibilität spricht außerdem die Verwendung einer möglichst allgemeingültigen, vielfältig anwendbaren Notation, wie etwa der UML. Die direkte Verwendung einer der in diesem Kapitel aufgeführten Beschreibungssprachen kann an dieser Stelle nicht entschieden werden, da das Ziel dieser Arbeit über die konkreten Einsatzzwecke der genannten Sprachen hinausgeht und mehr Flexibilität fordert.

Im folgenden Kapitel wird das Konzept für Interaktionsmodelle zur Integration in bestehende Benutzerschnittstellen vorgestellt. Es beginnt mit dem Anwendungsbeispiel der Integration von Anwendungen in bestehende Benutzerschnittstellen im Automobil. Bereits in Kapitel 2 wurde dieser Anwendungsbereich wegen des besonderen Kontexts als interessant für die vorliegende Arbeit herausgestellt. Durch die Analyse, wie bestehende Lösungen die Integration unbekannter Anwendungen ins Automobil erreichen, wird der benötigte Ansatz dieser Arbeit weiter eingegrenzt. Darauf aufbauend wird das Konzept für abstrakte Interaktionsmodelle zur flexiblen Integration in bestehende Benutzerschnittstellen erarbeitet.

# 4

# Ein Konzept für Interaktionsmodelle zur Integration in bestehende Benutzerschnittstellen

In den Kapiteln 2 und 3 wurden für diese Arbeit relevante Grundlagen der Benutzerschnittstellenentwicklung eingeführt. Dabei wurde insbesondere die Modellierung von Interaktion auf unterschiedlichen Ebenen, wie sie in zahlreichen Ansätzen verwendet wird, betrachtet und bewertet. In dieser Arbeit wird ein Konzept für die Benutzerschnittstellenmodellierung erarbeitet, das die benötigte minimale und maximale Abstraktionsebene enthält, um Anwendungen vollautomatisch in verschiedene bestehende Benutzerschnittstellen integrieren zu können.

Zunächst wird die Integration unbekannter Anwendungen in den Interaktionsraum Automobil diskutiert und bestehende Lösungen und Ansätze dafür analysiert. Der Interaktionsraum Automobil wurde bereits in Kapitel 2 wegen seiner besonderen Eignung als Anwendungsbeispiel für diese Arbeit eingeführt. Die Bewertung der bestehenden Ansätze fließt zusammen mit den Erkenntnissen aus den Grundlagenkapiteln 2 und 3

in die Erstellung und Ausarbeitung des Konzeptes ein, das im weiteren Verlauf dieses Kapitels vorgestellt wird.

## 4.1. Anwendungsbeispiel: Integration in Benutzerschnittstellen im Automobil

Die Integration von Anwendungen ins Fahrzeug wurde bisher hauptsächlich aus Sicht der Anbieter von externen Anwendungen sowie aus Sicht der Fahrzeughersteller beleuchtet. Welche Sichtweise die besten Ergebnisse für den Anwender bringt, ist dabei umstritten: Auf der einen Seite möchte der Anwender seine individuellen Anwendungen sehr flexibel im Fahrzeug nutzen, also zum Beispiel sein persönliches Mobiltelefon integrieren können mit Zugriff auf für ihn relevante Anwendungen. Auf der anderen Seite ist eine sichere und bequeme Bedienung im Fahrzeug wünschenswert, was eine Kontrollierbarkeit der Interaktionsmöglichkeiten seitens des Fahrzeugherstellers erfordert. Man kann von einem Spannungsfeld zwischen den drei Parteien Fahrzeughersteller, Anbieter von externen Anwendungen sowie dem Anwender sprechen. Dieses Spannungsfeld ist in Abbildung 4.1 illustriert. Im Folgenden werden bestehende Ansätze zur Integration externer Anwendungen diskutiert und das Spannungsfeld daraufhin bewertet.

### 4.1.1. Integration unter Kontrolle der Fahrzeughersteller

Bevor es Internetverbindungen im Fahrzeug gab, stand zur Entwicklungszeit fest, welche Anwendungen und Daten in das Infotainmentsystem integriert werden. Die Kontrolle lag voll beim Fahrzeughersteller. Diese Sichtweise steht auch heute noch vielfach im Vordergrund. Die Hersteller bestimmen mit eigenen Lösungen Art und Umfang der Anwendungen, die integriert werden können und öffnen durch eigene Schnittstellen eine Möglichkeit, die Anwendungen später zu erweitern. BMW betreibt beispielsweise ein eigenes System zur Integration kontrollierter Anwendungen über das Apple iPhone. Neue Anwendungen können so für die Integration im Fahrzeug abgestimmt und für BMW-Kunden freigegeben werden [BMW13]. Mercedes-Benz hat mit DriveStyle ein ähnliches System der Smartphone-Integration eigener Anwendungen, das unabhängig vom Fahrzeug erweitert werden kann [Dai13a]. Audi, unter dem Namen Audi connect [Aud14], und Ford mit SYNC AppLink [For13] bieten umfangreiche Integrationen von Internetdiensten und externen Anwendungen ins Fahrzeug, jeweils unter der eigenen

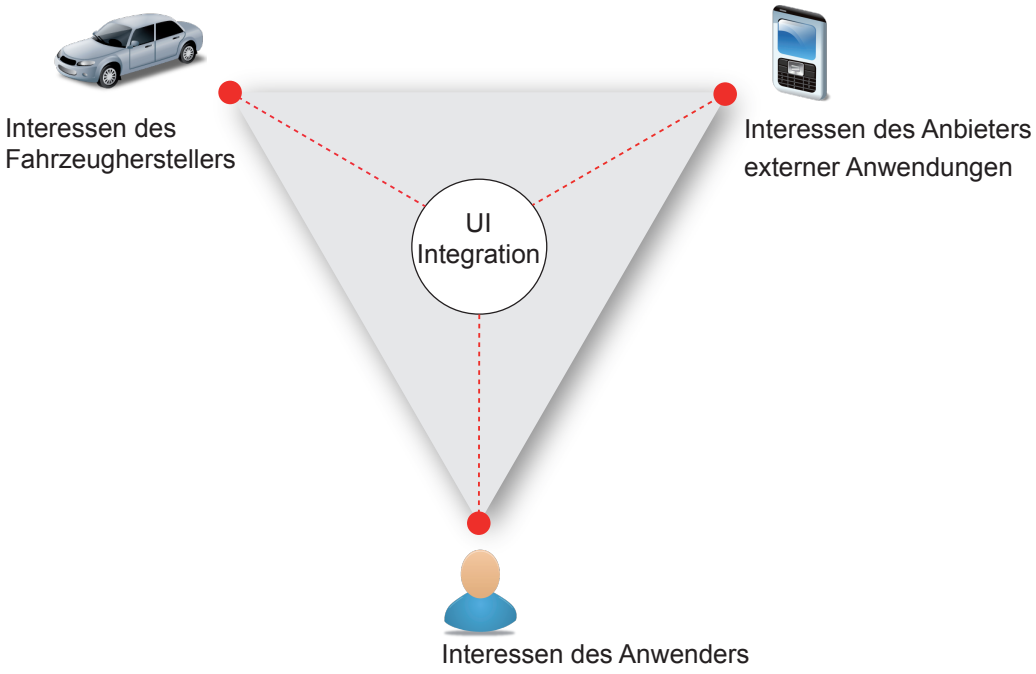


Abbildung 4.1.: Spannungsfeld der Integration externer Applikationen ins Fahrzeug mit den drei Parteien Fahrzeughersteller, Anbieter von externen Anwendungen und Anwender

Kontrolle und durch Aufbau von Partnerschaften zu Anbietern. Ein eigenes Backend mit der Möglichkeit, eigene entwickelte Apps im Fahrzeug zugänglich zu machen, bietet beispielsweise auch Mercedes-Benz mit COMAND Online [Dai13b].

Die sicherheitsrelevanten Anforderungen im Hinblick auf das besondere Umfeld der Bedienung im Auto während der Fahrt können durch eine kontrollierte Integration neuer Dienste und Anwendungen optimal durch die Fahrzeughersteller sichergestellt werden. Außerdem bietet diese Kontrolle auch die perfekte Integration im Sinne des eigenen Bedien- und Anzeigekonzepts und unterstützt markenstrategische Entscheidungen im wichtig gewordenen Feld des Infotainments im Auto. Die langen Entwicklungs- und Lebenszeiten eines Automobils erschweren es, mit den im Bereich der Konsumentenelektronik bekannten kurzen Innovations- und Lebenszyklen mitzuhalten. Daher haben sich auch einige andere Ansätze entwickelt, auf die im Folgenden weiter eingegangen wird.

### 4.1.2. Integration über Schnittstellen zur Bildschirmübertragung

Über standardisierte Schnittstellen wird bereits seit einiger Zeit zum Beispiel das Mobiltelefon des Fahrers erfolgreich in die Fahrzeugumgebung integriert. Eine Integration kompletter Anwendungen ist eines der nächsten Ziele, und wurde zum Beispiel im Standardisierungsgremium der Arbeitsgruppe *Consumer Electronics for Automotive* (CE4A) unter dem Titel *Terminal Mode* [CE408] vorangetrieben. Die Arbeit ging im *Car Connectivity Consortium* (CCC) unter dem Namen *MirrorLink* weiter, Version 1.0 wurde 2013 veröffentlicht [Car13]. Damit wird es Anbietern von Apps ermöglicht, diese Anwendungen über Bildschirmübertragung sicher in Fahrzeugen, die MirrorLink unterstützen, anzeigen zu können und über die Eingabemöglichkeiten im Fahrzeug bedienbar zu machen. Auch einzelne Hersteller mobiler Geräte versuchen mit eigenen Lösungen zur Bildschirmübertragung den Weg ins Automobil zu schaffen, wie etwa Apple, das mit iOS7 die technische Grundlage für „Apple Experience im Auto“[App13] legte.

Eine standardisierte Schnittstelle bietet eine gute Möglichkeit, viele neue Dienste und Apps ins Fahrzeug zu bringen, ohne dass für jeden Fahrzeughersteller individuell entwickelt werden muss. Für den Automobilhersteller bedeutet dies die Abgabe eines Teils der Kontrolle über das Erscheinungsbild der Benutzerschnittstelle. Eine Differenzierung zu anderen Fahrzeugen oder neue, innovative Design- und Bedienkonzepte eines Fahrzeugherstellers können damit nur eingeschränkt gewährleistet werden.

### 4.1.3. Ansätze aus anderen Anwendungsbereichen

#### Universal Remote Console (URC)

Die Idee, externe Anwendungen fernzusteuern, wird in einigen Ansätzen untersucht, die auch auf den Automobilbereich angewandt werden könnten. So bietet beispielsweise das *Universal Remote Console* (URC) Framework von Zimmermann und Vanderheiden [ZV10] die Vorlage eines Standards [ISO08] für eine universelle Fernbedienung, welche beliebige unterstützte Geräte steuern kann. Das URC Framework unterscheidet zwischen zwei Klassen von Geräten, den universellen Fernsteuerungen (URC) und den Empfängern, also zu steuernden Geräten, den *Targets*. Das URC stellt das UI des Targets nach den individuellen Vorlieben und Einstellungen des Benutzers dar. Diese werden über XML-Dokumente von den Targets bereitgestellt. Der Anwendungsfall, der

im Zusammenhang mit URC häufig genannt wird, ist die Steuerung von Hauselektronik über eigene, individuelle Geräte, die URCs. In einem solchen Szenario wird ein *Universal Control Hub* (UCH) eingerichtet, der die verschiedenen URCs mit den zu steuernden Geräten, den Targets, verbindet [ZV07]. Die Architektur sieht vor, dass der Zugang zu Backend-Servern möglich ist. Das UCH lädt UI-Beschreibungen von diesen Servern herunter und stellt sie den URCs bereit. Für einen Einsatz in einer Umgebung mit eingeschränktem Internetzugriff müsste diese Architektur entsprechend angepasst werden. Im Automobilbereich wäre aber eine entsprechend angepasste Lösung denkbar. Der zentrale Bordcomputer könnte als URC fungieren und eine Verbindung zu Targets, wie etwa einem Smartphone im Fahrzeug, könnte statt durch einen UCH über eine angepasste Umsetzung hergestellt werden, um die UI-Beschreibung zu übertragen. Für einen flexiblen Einsatz könnte der zentrale Bordcomputer im Auto außerdem als Target fungieren, um die Steuerung durch das Smartphone oder andere Geräte zu ermöglichen. Dadurch wäre auch die Erweiterung um neue Eingabemethoden im Fahrzeug denkbar. Eine stärkere Regulierung der Funktionalität oder eine entsprechend angepasste Lösung ist allerdings auch dafür nötig, um unkontrollierten Zugriff auf die Funktionalität des Bordcomputers zu vermeiden. Das URC-Framework ist daher nicht direkt für die Verwendung im Fahrzeug einsetzbar.

#### Ansätze mit Web-Technologien

Aufgrund der weiten Verbreitung von HTML und der umfangreichen Version HTML5 [Wor14] werden häufig auch Ansätze mit Webtechnologien untersucht, in denen neue Applikationen beispielsweise als Webseite oder Webdienst eingebunden werden. Sonnenberg [Son10] diskutiert die Einbindung dynamischer Webdienste über HTML5 ins Fahrzeug. Es wird dabei davon ausgegangen, dass für die Darstellung der externen Anwendung auf dem mobilen Gerät und die Darstellung im Fahrzeug dasselbe HTML-Dokument verwendet werden kann. Im Fahrzeug wird das Dokument mithilfe von Stylesheets auf die Darstellung im Fahrzeug angepasst. Hüger [HÏ1] empfiehlt sogar den Einsatz finaler UI-Beschreibungen auf Basis von HTML. Im Kern sind HTML-basierte Anwendungen im Fahrzeug bereits im Einsatz, wie etwa bei den Mercedes-Benz Apps in COMAND Online [Dai13b]. Über einen Browser in der Fahrzeugsoftware sind damit flexible Erweiterungen um neue Anwendungen zur Laufzeit möglich. Allerdings sind derartige Ansätze auf Webtechnologien beschränkt. Für andere Anwendungen muss erst eine HTML-Version erstellt werden. Aufgrund der großen Verbreitung von HTML sowie

der Standardisierung des Formats ist dessen Verwendung für plattformübergreifende Benutzerschnittstellenbeschreibungen ein pragmatischer Weg, der viele Anwendungen abdecken kann.

### 4.1.4. Gewichtung des Spannungsfelds

Die drei Interessenten im Spannungsfeld der Integration externer Anwendungen sind Fahrzeughersteller, Anbieter von externen Anwendungen sowie der Anwender. Bei einer Lösung unter vollständiger Kontrolle des Fahrzeugherstellers wird die Entscheidung darüber, welche Anwendungen zur Integration verfügbar sind, dem Anwender vorweggenommen. Er kann dann nicht frei entscheiden, welche Anwendungen er integrieren möchte. Bei einer Lösung unter Kontrolle der Anbieter externer Anwendungen kann der Anwender zwar durch Kauf oder Installation entscheiden, allerdings ist eine perfekte Integration in das Fahrzeug seiner Wahl nicht gewährleistet. Für eine perfekte Abstimmung der Anwendung auf das Bedien- und Anzeigesystem im Fahrzeug ist eine Kontrollierbarkeit seitens des Fahrzeugherstellers erforderlich. Da die Kontrollierbarkeit für eine Integration im sicherheitskritischen Anwendungsfeld Automobil erforderlich ist, ist dieser Punkt ausschlaggebend. Aus technischer Sicht muss daher dem Fahrzeughersteller diese Kontrollierbarkeit ermöglicht werden. Der Anwender sollte die Möglichkeit haben, aktiv den Nutzungskontext zu entscheiden. Befindet sich der Anwender im Fahrzeug, gilt allerdings der Nutzungskontext Fahrzeug und wird durch die Verantwortung des Fahrzeugherstellers für Sicherheit und optimale Bedienbarkeit während der Fahrt eingeschränkt. Um das Spannungsfeld der verschiedenen Interessen zu lösen, sollte also mit folgender Interessensreihenfolge gewichtet werden: Fahrzeughersteller, Anwender, Anbieter externer Anwendungen. Der Anwender folgt dabei den Einschränkungen im Nutzungskontext Fahrzeug, soll aber ansonsten frei entscheiden können, welche Anwendungen er in diesem Kontext bedienen möchte. Der Anbieter externer Anwendungen folgt ganz im Sinne eines nutzer-orienterten Ansatzes diesen Interessen des Anwenders. Das relativ gewichtete Spannungsfeld ist in Abbildung 4.2 illustriert.

### 4.1.5. Zusammenfassung und Schlussfolgerungen für die Konzeptentstehung

Die Öffnung des Interaktionsraums Automobil für externe Unterhaltungs- oder Informationsanwendungen ist durch kritische Anforderungen eingeschränkt. Die Erfüllung dieser

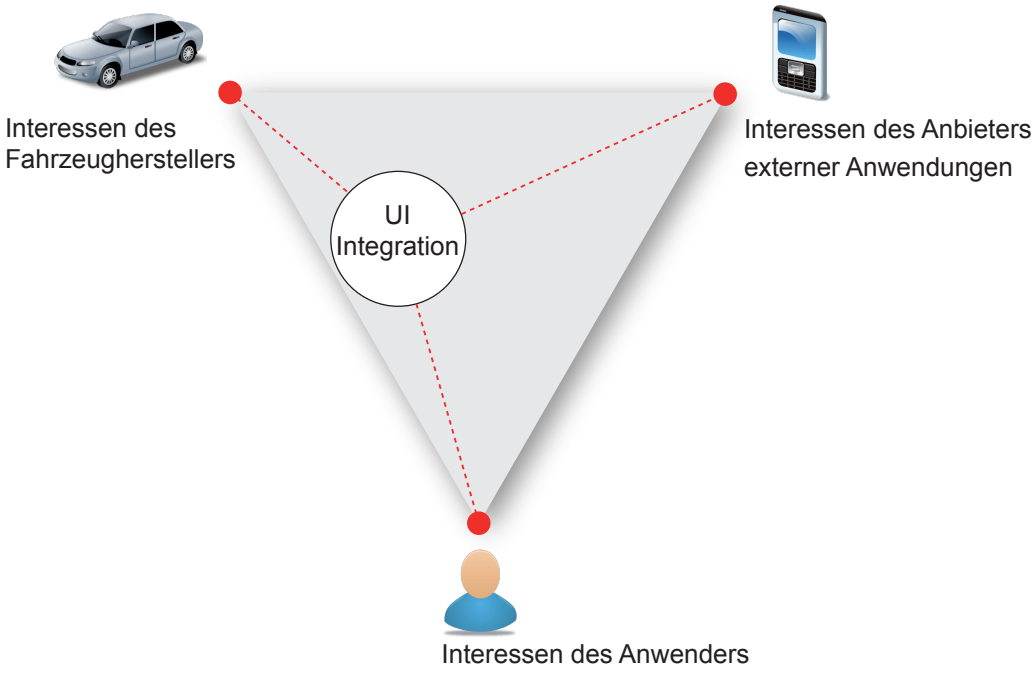


Abbildung 4.2.: Gewichtetes Spannungsfeld der Integration externer Applikationen ins Fahrzeug.

Anforderungen seitens der Anbieter externer Anwendungen ist nicht immer möglich, da Abstimmungen auf die fahrzeugspezifischen Bedien- und Anzeigesysteme in jedem Fahrzeug unterschiedlich sein können. Liegt die Verantwortung der Integration allerdings komplett beim Fahrzeughersteller, entsteht für diesen zusätzlicher Entwicklungs- und Pflegeaufwand je Fahrzeugmodell. Um den Interessen aller Anwender im Fahrzeug gerecht zu werden, muss eine Vielzahl an Anwendungen in unterschiedlichen Versionen für die verschiedenen Fahrzeuge bereitgehalten werden. Daher wurden einige Ansätze entwickelt, die Beschreibungen von Benutzerschnittstellen verwenden, welche durch Anwendung von eigenen Stil- und Layoutdokumenten oder durch Transformationen eine Benutzerschnittstelle im Zielsystem entsprechend des dortigen Konzepts erzeugen können. Neben Ansätzen zur Vereinheitlichung und Standardisierung solcher Beschreibungsformate haben sich vor allem Ansätze unter Verwendung von Webtechnologien durchgesetzt. Durch die große Verbreitung von Applikationen auf Basis von Webtechnologien, die im Browser ablaufen, können dadurch viele Applikationen abgedeckt werden. Für die Konzeptentwicklung in dieser Arbeit sind diese Ansätze ein Ausgangspunkt.

Allerdings soll das Konzept noch deutlich flexibler ansetzen und unabhängig von der zu Grunde liegenden Plattform oder Technologie Anwendungen in das Zielsystem integrieren können. Im Bereich der Benutzerschnittstellen im Automobil hat sich aufgrund der besonderen Anforderungen und der langen Lebenszeit eines Infotainmentsystems im Fahrzeug ein hohes Maß an Gründlichkeit in Entwicklung und Test, häufig unter Verwendung modell-basierter Prozesse und Werkzeuge, etabliert. Für die Ausarbeitung eines Konzepts für die Integration sind diese Rahmenbedingungen vorteilhaft, insbesondere im Hinblick auf die Verwendung von abstrakten Modellen, die in die bestehenden Benutzerschnittstellen integriert werden sollen. Daher ist die Integration unbekannter Anwendungen in Benutzerschnittstellen im Automobil ein besonders interessanter Anwendungsfall, der bei der Umsetzung und Evaluierung des Konzepts im weiteren Verlauf der Arbeit weiter verwendet wird. Im folgenden Abschnitt wird zunächst die Entstehung und Ausarbeitung des Konzepts für die Interaktionsmodellierung im Detail beschrieben.

## 4.2. Konzeptentstehung

Die Form der Integration neuer Anwendungen in ein bestehendes Bedien- und Anzeigekonzept ist abhängig von Interessen der beteiligten Anbieter und Nutzer sowie dem Nutzungskontext. Für diese Arbeit wurde eine Einteilung in Software-Lösungen und Hardware-Lösungen erarbeitet. Eine weitere Einteilung unterscheidet zwischen Lösungen für die Integration zur Entwicklungszeit und Lösungen für die Integration zur Laufzeit. Anhand dieses Rasters, das in Tabelle 4.1 dargestellt ist, können individuelle Lösungen eingeordnet werden. Die Einordnung muss dabei nicht in genau einem Feld des Rasters erfolgen.

Externe Anwendungen, die in die eigene Benutzerschnittstelle integriert werden sollen, können in vielen Fällen von Beginn der Entwicklungszeit an berücksichtigt werden (Tabelle 4.1, Feld oben links). Zur Laufzeit werden dann lediglich die Daten der externen Quelle geladen. Eine Medienspieler-Anwendung ist ein Beispiel für eine solche Lösung, die vorab integriert und mit dem System mitgeliefert wird. Sind Anwendungen noch nicht zur Entwicklungszeit bekannt, können Schnittstellen definiert werden, um diese zu einem späteren Zeitpunkt, also nach der Entwicklung, in die eigene Benutzerschnittstelle zu integrieren (Tabelle 4.1, Feld unten links). Wird eine Anwendung inklusive Benutzerschnittstelle gemäß dieser Schnittstellendefinition entwickelt, kann sie in das bestehende System zur Laufzeit integriert werden, etwa durch Installation oder eine

| | Software-Lösung | Hardware-Lösung |
|---|---|---|
| Entwicklungszeit | Individuelle Entscheidung über externe Anwendungen und Integration in die Benutzerschnittstellenentwicklung von Beginn an | Festlegen unterstützter Hardware und Entwicklung der Benutzerschnittstellen für diese von Beginn an |
| Laufzeit | Einbindung der Benutzerschnittstellen neuer Anwendungen über vorab definierte Software-Schnittstellen | Einbindung von Hardware inkl. Benutzerschnittstellen neuer Anwendungen über vorab definierte Hardware-Schnittstellen |

Tabelle 4.1.: Lösungen zur Integration externer Anwendungen in bestehende Benutzerschnittstellen - Einteilung in Software- und Hardware-Lösungen für die Entwicklungszeit und für die Laufzeit

Software-Aktualisierung. Steht zur Entwicklungszeit eine bestimmte zu integrierende Hardware fest, das kann zum Beispiel ein bestimmtes Smartphone sein, kann deren Hardware-Spezifikation vorab verwendet werden, um eine feste Integration des Gerätes zu erreichen (Tabelle 4.1, Feld oben rechts). Dabei kann zum Beispiel das Gerät seinen Anwendungsinhalt über die Ein- und Ausgabeelemente des bestehendes Systems verfügbar machen. Steht eine bestimmte zu integrierende Hardware noch nicht zur Entwicklungszeit fest, können generische Schnittstellen definiert werden, um entsprechende Hardware zur Laufzeit integrieren zu können (Tabelle 4.1, Feld unten rechts).

Für diese Arbeit ist die Unterstützung einer bestimmten Hardware weniger relevant, da es um die Darstellung der Benutzerschnittstellen externer Anwendungen geht, und wie diese in bestehende Systeme integriert werden können, ohne sie zur Entwicklungszeit definiert haben zu müssen. Daher liegt der Fokus auf den Software-Lösungen, die sowohl für Anwendungen, die von externer Hardware integriert werden sollen, als auch für Anwendungen von anderen externen Quellen, wie externen Servern oder dem Internet, relevant sind (Tabelle 4.1, linke Spalte).

Bei Software-Lösungen, die eine Integration externer Anwendungen von Beginn der Entwicklungszeit an vorsehen, steht dem Hauptvorteil, der vollen Kontrolle über Darstellung und Bedienung, der Nachteil der fehlenden Flexibilität bezüglich der Erweiterbarkeit zur Laufzeit gegenüber. Für die zu integrierende Anwendung muss außerdem für jedes Bedien- und Anzeigekonzept, das unterstützt werden soll, Aufwand für Implementierung und Test aufgebracht werden. Lösungen für die Integration zur Laufzeit erlauben

| | Software-Lösung |
|---|---|
| Entwicklungszeit | (+) Volle Kontrolle über Benutzerschnittstelle<br>(-) Geringe Erweiterbarkeit zur Laufzeit<br>(-) Aufwand für Implementierung und Test je Zielkonzept |
| Laufzeit | (+) Erweiterbar mit neuen Anwendungen zur Laufzeit<br>(+) Kontrollierbar über Schnittstellendefinition<br>(-) Aufwand für Implementierung und Test je Zielkonzept |

Tabelle 4.2.: Wichtige Vor- und Nachteile von Software-Lösungen für die Entwicklungszeit und für die Laufzeit

eine flexible Erweiterung eines bestehenden Systems um neue Anwendungen. Der Zusatzaufwand bleibt aber bestehen, da Benutzerschnittstellen für externe Anwendungen individuell und für jedes Zielsystem gemäß der entsprechenden Schnittstellendefinition erstellt werden müssen. Tabelle 4.2 fasst diese Vor- und Nachteile zusammen.

Für diese Arbeit wird auf modellgetriebene Entwicklung gesetzt, da damit der Zusatzaufwand für Implementierung und Test reduziert werden kann [GPR06]. Da Flexibiliät im Sinne späterer Erweiterbarkeit besonders für Systeme, die eine lange Lebenszeit haben sollen, wichtig ist, muss die Lösung diese Erweiterbarkeit zur Laufzeit beinhalten.

Diese Rahmenbedingungen sind der Ausgangspunkt für die weitere Ausarbeitung des Konzepts. Für diese Arbeit ist es wichtig, die benötigten minimalen und maximalen Abstraktionsstufen zu erarbeiten. Unabhängig von konkreten Sprachen muss daraus dann ein Konzept für die abstrakte Interaktionsmodellierung entstehen mit dem Ziel einer automatisierbaren Integration dieser Modelle in existierende Benutzerschnittstellen. Für die automatisierbare Verarbeitung ist eine wichtige Voraussetzung, dass die Modelle formal sind, also für Computer nachvollziehbare Interpretationen zulassen. Der folgende Abschnitt geht auf diesen Aspekt genauer ein. Danach wird das Konzept, beginnend mit der Verfeinerung der Ziele der Arbeit und den damit verbundenen Anforderungen, weiter vertieft.

## 4.3. Spezifikationsphase vs. Konzeptphase

Wie in Kapitel 2 und 3 bereits erwähnt, ist eine wichtige Voraussetzung für die Modellierung im Sinne automatisierbarer Weiterverarbeitung, dass die Modelle formal sind.

Für die Entwicklung eines Bedienkonzeptes ist Formalität allerdings weniger relevant, da sich die eigentlichen Benutzerschnittstellenelemente sowie Logik und Fluss erst in dieser Phase ergeben. In einem kreativen Prozess entstehen dabei zunächst Funktions- und Eigenschaftenumfang der Systeme, die logischen und visuellen Elemente der Benutzerschnittstelle sowie die Eigenschaften verwendeter Ein- und Ausgabegeräte. Die Aufgabe formaler Beschreibungen von (Teil-)Ergebnissen dieser Phase kann den Kreativprozess stören, daher ist es wichtig, mit der formalen Beschreibung nicht zu früh zu beginnen. Der Übergang von der kreativen Phase der Konzeptentwicklung in die formale Entwicklungsphase ist eine Herausforderung. Zeitpunkt und die Form der Übergabe der Konzeptergebnisse werden nicht immer für alle Beteiligten zufriedenstellend gewählt. So kann es sein, dass Benutzerschnittstellengestalter noch kreativ sein wollen, wichtige Entscheidungen für die finale Umsetzung aber schon von Softwareingenieuren festgelegt werden müssen.

In vielen modernen Organisationsstrukturen haben sich deshalb hochdynamische Prozesse entwickelt, unter Einbezug von sogenannten *User Experience (UX) Teams* [SP10], die sich ausschließlich dem Bedienerlebnis der Endkunden widmen sollen, und häufig querschnittlich besetzt sind, vom Designer bis zum Softwareentwickler. Sie sollen dafür sorgen, dass dieses Bedienerlebnis, die User Experience, von Anfang an betrachtet wird und eindeutig bei der Übergabe an die Entwicklerteams definiert ist. Die Entwicklung der User Experience entsteht unter Einbezug technischer Machbarkeitsanalysen und einem Gespür für Ästhetik und ist bis zum finalen Ergebnis kreativ und flexibel [SP10]. In dieser Arbeit wird auf diesen kreativen Teil der Benutzerschnittstellenentwicklung nur am Rande eingegangen. Vielmehr wird davon ausgegangen, dass der Kreativprozess in einer formalen oder zumindest semi-formalen Beschreibung endet, die von technisch orientierten Entwicklungsteams oder Auftragnehmern verwendet werden kann, um Logik und Fluss und zu verwendende Ein- und Ausgabegeräte für die Benutzerschnittstelle im Sinne des UX Teams umzusetzen.

Der Aspekt der Formalisierung, insbesondere deren Zeitpunkt im Gesamtentwicklungsprozess, wird in Abbildung 4.3 veranschaulicht. In der kreativen und weniger formalen Konzeptphase entsteht die User Experience unter Einbezug von bereits bekannten Faktoren der Zielplattform und -umgebung. Die Spezifikation der Zielplattform und -umgebung kann sich in der Konzeptphase im Zusammenspiel mit dem UX Team in einem gewissen Rahmen anpassen. Sobald sie feststeht, fließt sie in die technische und formale Phase der Umsetzung (*Implementation*) ein. Ebenso fließen die Ergebnisse der kreativen UX

Design Phase in die Umsetzung ein. Sie müssen dafür das nötige Maß technischer Korrektheit, Vollständigkeit und Formalität aufweisen. Form und Übergabezeitpunkt einer solchen UX Spezifikation muss zwischen UX Team und Umsetzungsteam abgestimmt sein und kann etwa ein Dokument, Programmcode sowie Grafiken enthalten. Die Übergabe kann auch in einem iterativen Prozess aus mehreren Teilen bestehen, kann vom UX Team, einem Spezifikationsteam, vom Umsetzungsteam oder in einem gemeinsamen Ansatz durchgeführt werden. Abbildung 4.3 verdeutlicht, dass es letztlich eine Formalisierung für die Umsetzung geben muss, denn die Grundlage maschinenlesbaren Codes ist formal.

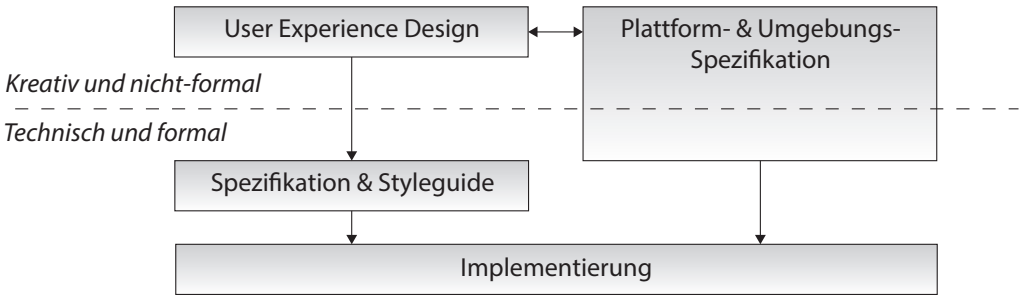


Abbildung 4.3.: Formalisierung (gestrichelte Linie) bei der Entwicklung einer Benutzerschnittstelle im Vergleich zum Gesamtentwicklungsprozess (vereinfacht, ohne Iterationen).

In dieser Arbeit wird auf die besonderen Umstände der Übergabe von Konzeptergebnissen in die Umsetzung an einigen Stellen eingegangen. Es wird allerdings nicht versucht, ein einheitliches Format für die Form von UX Spezifikation und Styleguide zu definieren. Dies ist Gegenstand weiterer Arbeiten, die beispielsweise im Rahmen des Förderprojekts *Automotive HMI*[1] entstanden sind. Für das eigentliche Konzept setzt diese Arbeit ab dem Zeitpunkt der erfolgten Formalisierung an und verdeutlicht anhand von Beispielen deren mögliche Form, sofern dies für die Absicherung des Ansatzes in dieser Arbeit benötigt wird.

[1] Förderprojekt: Automotive HMI, http://www.automotive-hmi.org, aufgerufen am 27.10.2015

## 4.4. Ziele

Wie im vorigen Abschnitt erläutert, stellt die Spezifikation einer Benutzerschnittstelle zusammen mit entsprechenden Gestaltungsrichtlinien und -elementen das benötigte Wissen bereit, um die Benutzerschnittstelle umzusetzen und an entsprechende Applikationen zu binden. Der Grad der Abstraktion und damit auch der Grad der Formalisierung kann für unterschiedliche Projekte, Teams oder Organisationen verschieden sein. Die beiden folgenden Beispiele illustrieren Unterschiede im Informationsgehalt von Benutzerschnittstellenspezifikationen.

**Beispiel für den Informationsgehalt einer abstrakten Spezifikation** *Der Startbildschirm enthält mittig ausgerichtet maximal 4 Auswahlflächen mit dem Logo der jeweiligen Applikation darauf abgebildet. Sind mehr als 4 Applikationen vom Nutzer installiert, wird eine Möglichkeit zum Weiterschalten der Applikationen-Liste eingeblendet.*

**Beispiel für den Informationsgehalt einer detaillierteren Spezifikation** *Der Startbildschirm folgt dem Raster* GridHome*. Im Bereich* GridElementHomeCenter *wird eine vertikale Liste* ButtonlistVertical *angezeigt mit dem Wert 4 für* MaxDisplayedElements*. Die Listenansicht wird an den Datensatz* ListApplications *geknüpft, der Logos der installierten Applikationen enthält. Im rechten Bereich* GridElementHomeRight *wird ein Button* TextButton *angelegt mit Textaufschrift „Weiter" und folgendem Verhalten: Der Button ist unsichtbar und deaktiviert, wenn die Anzahl an Elementen in* ListApplications *kleiner oder gleich 4 ist oder das Ende der Liste erreicht ist und keine weiteren Elemente mehr folgen; der Button ist sichtbar und aktiviert, wenn die Anzahl größer als 4 ist und das Ende der Liste nicht erreicht ist.* [...]

Das erste Beispiel mit einer abstrakteren Beschreibung, die viele Details dem Umsetzungsteam überlässt, ist textuell leicht zu erfassen, allerdings stellt sich jeder das Ergebnis etwas anders vor. Dennoch enthält auch diese abstrakte Beschreibung einen gewissen Grad an Formalität, etwa dass maximal 4 Applikationen auf dem Startbildschirm sichtbar sein können. Das zweite Beispiel ist in der rein textuellen Form schwieriger zu lesen und erfassen. Häufig wird die Lesbarkeit einer solche Detailspezifikation daher über grafische Repräsentationen, z. B. Zustandsdiagramme, verbessert. Diese detaillierte Spezifikation lässt dem Umsetzungsteam kaum Freiheiten, da alle Elemente,

auf die verwiesen wird, ebenfalls bereits mit ihrem Detailverhalten und Aussehen definiert sind. Obwohl das erste Beispiel sehr vage erscheint, ist es denkbar, dass eine mögliche Umsetzung genau der Beschreibung aus dem zweiten Beispiel entsprechen wird. Dies ist etwa dann der Fall, wenn das Umsetzungsteam auf definierte Elemente der Zielplattform zurückgreift, die zuvor im Sinne der detaillierten Spezifikation aus dem zweiten Beispiel definiert wurden.

Die Ziele der Arbeit adressieren die Notwendigkeit der verschiedenen Abstraktionsebenen und Übertragung abstrakter Beschreibungen auf mögliche Zielplattformen. Ist ein bereits erarbeitetes Benutzerschnittstellenkonzept im Einsatz, soll es möglich sein, abstrakt definierte Benutzerschnittstellen externer Anwendungen im Sinne der erarbeiteten User Experience zu integrieren. Das soll für verschiedene Benutzerschnittstellenkonzepte möglich sein, ohne dass die abstrakte Beschreibung auf das Zielsystem verweisen muss. Um dies zu erreichen, müssen vier Teilziele in dieser Arbeit erreicht werden, die im Folgenden beschrieben werden.

### 4.4.1. Ziel 1: Flexible abstrakte Beschreibung von Benutzerschnittstellen

In einer flexiblen abstrakten Beschreibung einer Benutzerschnittstelle soll es möglich sein, dass aus jedem abstrakten Interaktionselement mindestens ein spezifisches Benutzerschnittstellenelement abgeleitet werden kann, wie in Abbildung 4.4 illustriert. Die Beschreibung muss möglichst unabhängig von konkreten Bedien- und Anzeigekonzepten sein und ausreichend Flexibilität bieten, um Benutzerschnittstellen für unterschiedliche Zielsysteme dynamisch ableiten zu können.

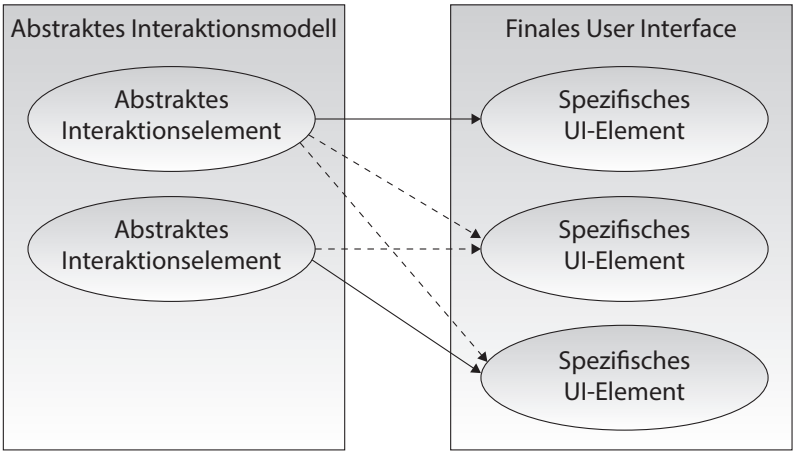


Abbildung 4.4.: *Ziel 1* ist ein Modellierungskonzept für die flexible abstrakte Beschreibung von Benutzerschnittstellen

### 4.4.2. Ziel 2: Automatische Weiterverarbeitung und Integration der abstrakten Beschreibung

Die abstrakte Beschreibung muss automatisiert weiterverarbeitet werden können, sodass durch Transformation der Beschreibung eine finale Benutzerschnittstelle zur Laufzeit erzeugt werden kann, die zum Zielkonzept passt (Abbildung 4.5). Nur wenn dieses Ziel erreicht wird, ist die notwendige Unabhängigkeit zwischen abstraktem Interaktionsmodell und finaler Benutzerschnittstelle gegeben.

Abbildung 4.5.: *Ziel 2* ist ein Konzept für einen Transformationsprozess für die automatische Weiterverarbeitung und Integration der abstrakten Beschreibung

### 4.4.3. Ziel 3: Erweiterbarer, flexibler Entwicklungsprozess

Ein erweiterbarer, flexibler Entwicklungsprozess für Benutzerschnittstellen muss erreicht werden, um die spätere Integration von neuen Anwendungen, die eine abstrakte Beschreibung der Benutzerschnittstelle mitbringen, in diese bestehenden Benutzerschnittstellen zu ermöglichen. Ohne die Bereitstellung von Abbildungen zwischen abstrakten Elementen auf die Elemente des Zielsystems ist keine Integration unbekannter Anwendungen denkbar. Diese Abbildungen müssen ein Teilergebnis des Entwicklungsprozesses des Zielsystems sein.

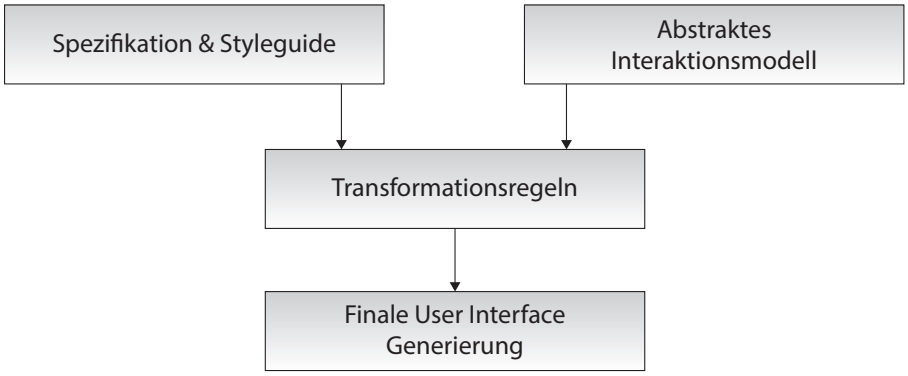


Abbildung 4.6.: *Ziel 3* ist ein Konzept für einen erweiterbaren, flexiblen Entwicklungsprozess von Benutzerschnittstellen

### 4.4.4. Ziel 4: Sicherstellung besonderer Anforderungen im Zielkontext

In dieser Arbeit wird davon ausgegangen, dass der Zielkontext, in den eine Anwendung integriert werden soll, von kritischen Rahmenbedingungen beeinflusst werden kann. In einem automatisierten Prozess, in dem Benutzerschnittstellen für einen Zielkontext generiert werden sollen, muss dieser Aspekt daher besonders beachtet werden. Im Automobil beispielsweise wird das Zielkonzept von rechtlicher Seite durch Regeln für minimale Fahrerablenkung, aber auch durch generelle Anforderungen an Komfort und Sicherheit im Fahrzeug beeinflusst. Das Ziel der Sicherstellung besonderer Anforderungen im Zielkontext muss also durchgängig betrachtet und erfüllt werden.

## 4.5. Anforderungen an die Ziele

Die Anforderungen lassen sich in die eigentliche *Modellierung*, den Prozess der *Integration* in bestehende Benutzerschnittstellen, und den *Entwicklungsprozess* einordnen. Für die Definition der Anforderungen wurden Ergebnisse und Aspekte aus verschiedenen Arbeiten [Mei10][Sch07b][Sch07a][Mel10] zusammen mit eigenen Erfahrungen in der praktischen Arbeit mit Benutzerschnittstellenentwicklern, Werkzeugen und Prozessen analysiert.

### 4.5.1. Anforderungen für die Modellierung

#### Unabhängigkeit

Die Art der Modellierung soll unabhängig von Betriebssystem, Ein- und Ausgabegeräten, Werkzeugen oder eingesetzten Programmiersprachen sein, da das abstrakte Modell Ausgangslage für die automatische Verarbeitung und Übertragung in eine vorher noch nicht bekannte Umgebung ist.

#### Anpassbarkeit

Um für einfachere Anwendungen eine leichtgewichtige Modellierung zu erlauben, aber gleichzeitig geeignet für komplexere Anwendungen zu sein, soll das Konzept flexibel anpassbar sein. Dies umfasst vor allem die flexible Definition der Anwendungsdomäne, die Möglichkeit sich auf diese zu beschränken sowie das Modell dennoch einfach für andere Bereiche anzupassen oder zu erweitern.

#### Gebrauchstauglichkeit

Die Gebrauchstauglichkeit (engl. *Usability*, nach der englischen Bezeichnung) umfasst nach Norm EN ISO 9241-11 das Ausmaß, mit dem das Produkt oder System, in diesem Falle die Modellierungsform, in einem bestimmten Anwendungskontext, nämlich der Benutzerschnittstellenentwicklung, geeignet ist, um festgelegte Ziele damit effektiv, effizient und zufriedenstellend zu erreichen. Da die abstrakte Modellierung durch Anwender

durchgeführt wird, ist die Gebrauchstauglichkeit dafür wichtig und muss betrachtet werden. Die dazugehörende Entwicklung optimaler Werkzeuge zur Erstellung der Modelle baut auf dieser Grundlage des Konzepts auf.

### 4.5.2. Anforderungen für die Integration

#### Vollständigkeit

Die Integration soll den vollen Umfang der abstrakten Interaktionsmöglichkeiten abdecken können. Das Konzept soll also ermöglichen, die zu Grunde liegende Interaktion mit einer Anwendung über die vorgestellte Form der Modellierung vollständig abzubilden. Außerdem müssen die Informationen über die Zielplattform die Definition der zu Grunde liegenden abstrakten Form der Interaktion beinhalten, um die Integration zu erlauben.

#### Eindeutigkeit

Die modellierte Interaktion muss sich eindeutig auf das Zielsystem abbilden können. Sofern es Mehrdeutigkeiten gibt, muss der Integrationsprozess eine Entscheidung treffen, die zu einer eindeutigen, reproduzierbaren Abbildung kommt.

#### Kontextsensitivität

Der Integrationsprozess soll die flexible Verwendung von Kontextinformationen der jeweiligen Zielplattformen ermöglichen können.

#### Automatisierbarkeit

Der Integrationsprozess soll automatisiert ablaufen. Lediglich zum Zwecke der Veränderung von Gestaltungselementen soll manueller Eingriff stattfinden können.

### 4.5.3. Anforderungen an einen Entwicklungsprozess

Die folgenden Anforderungen beziehen sich auf den Entwicklungsprozess, mit dem eine Benutzerschnittstelle erstellt wird, die später eine Integration aus abstrakten Modellen erlauben kann.

#### Formalität

Diese Anforderung bezieht sich auf die Benutzerschnittstelle im Zielsystem. Die Spezifikation dieser muss zugänglich und zum notwendigen Grade formalisiert sein, sodass die Automatisierbarkeit der Integration ermöglicht werden kann.

#### Testbarkeit

Der automatische Integrationsprozess muss vorab ausreichend testbar sein, um den reibungslosen Ablauf einer Integration abzusichern.

## 4.6. Genereller Ansatz

Im Folgenden wird der generelle Ansatz beschrieben, mit dem die vorgestellten Anforderungen zur Erreichung der Ziele adressiert werden. Nach einer Zwischenanalyse zur Erfüllung der Anforderungen wird der Ansatz im darauffolgenden Abschnitt im Detail vorgestellt und analysiert.

### 4.6.1. Relevante Modellarten

In Kapitel 3 wurden die wichtigsten Modellarten zur Benutzerschnittstellenentwicklung eingeführt. Einige der Modellarten sind an die Einsatzumgebung sowie zu erwartende Nutzer der zu bedienenden Anwendung gebunden: Benutzermodell, Kontextmodell und Plattformmodell. Für eine abstrakte Modellierung einer Anwendung, die in eine noch unbekannte Zielumgebung integriert werden soll, sind diese nicht relevant und können auch nicht sinnvoll abstrahiert werden. Vielmehr sind die Informationen oder

| *Relevante Modelle einer Anwendung in einem Ausgangssystem* | *Relevante Modelle für die zu integrierende Anwendung in ein Zielsystem* |
|---|---|
| **Aufgabenmodell** | (Aufgabenmodell) |
| (Benutzermodell) | **Benutzermodell** |
| (Kontextmodell) | **Kontextmodell** |
| **Domänenmodell** | (Domänenmodell) |
| (Plattformmodell) | **Plattformmodell** |
| **Dialogmodell** | **Dialogmodell** |
| **Präsentationsmodell** | **Präsentationsmodell** |

Tabelle 4.3.: Relevante Modellarten (in Fettschrift) für die Integration einer Anwendung in eine bestehende Benutzerschnittstelle (Zielsystem)

Modelle der Zielumgebung zu Benutzer, Kontext und Platfform relevant. Damit bleiben Aufgabenmodell, Domänenmodell, Dialogmodell und Präsentationsmodell, die in die abstrakte Interaktionsmodellierung der zu integrierenden Anwendung einfließen müssen. Das Domänenmodell enthält die Applikationslogik und das Aufgabenmodell enthält den Kern der Interaktionsmöglichkeiten. Das Präsentationsmodell ist für die Darstellung der Benutzerschnittstelle relevant, und das Dialogmodell dient als Bindeglied zwischen Aufgabenmodell und Präsentationsmodell und stellt den dynamischen Teil der Benutzerschnittstelle dar. Während Aufgabenmodell und Domänenmodell direkt mit der Anwendung verknüpft sind und in eine Zielumgebung übernommen werden, sind Dialogmodell und Präsentationsmodell häufig nicht so leicht zu integrieren, da auch die Zielumgebung bereits einen typischen Dialogablauf, Ein- und Ausgabemöglichkeiten sowie Präsentationsobjekte bereitstellt. Für eine saubere Integration müssen dazu geeignete Abbildungen gefunden werden. Tabelle 4.3 führt die genannten relevanten Modellarten jeweils im Ausgangssystem und Zielsystem auf und stellt sie gegenüber. In der Tabelle wird das Problem, zwischen den Dialogmodellen sowie Präsentationsmodellen geeignete Abbildungen finden zu müssen, ersichtlich. Das Finden von geeigneten Abbildungen zwischen Modellen wird in der Literatur *Mapping Problem* genannt. Der folgende Abschnitt geht darauf ein.

### 4.6.2. Abbildungen zwischen Modellen - *Mapping-Problem*

Alle Informationen, die in den genannten Modellen stecken, müssen miteinander verknüpft werden, um die Erstellung der finalen Benutzerschnittstelle für das Zielsystem zu erreichen. Diese Transformationen zwischen Modellen, um zu einem Zielmodell zu gelangen, stellen ein bedeutendes Problem modellgetriebener Ansätze dar [GLR+02]. Puerta hat im Zusammenhang der Modellierung für die Benutzerschnittstellenentwicklung den Begriff „Mapping-Problem" geprägt [PE99]. Um das Mapping-Problem zu lösen, stehen in vielen Ansätzen Transformationen aus Aufgabenmodellen im Fokus [CLC04, LV04, TVTB12]. Meixner [Mei10] fasst fünf Lösungsansätze zusammen:

- Modellableitung über fest vorgeschriebene Transformationsregeln
- Partielle Modellableitung, bei der ein existierendes Modell durch Modellableitung um neue Informationen ergänzt wird
- Modellverknüpfung, bei der Elemente unterschiedlicher Modelle miteinander verknüpft werden
- Modellmanipulation durch Entwickler
- Modellaktualisierung, bei der die Daten eines Modells bei Änderungen an einem anderen Modell automatisch aktualisiert werden

Vollautomatische Transformationen haben sich als fehleranfällig erwiesen und bisher kaum durchgesetzt [MHP00]. Für eine Benutzerschnittstellenintegration zur Laufzeit ist es allerdings erforderlich, dass die Transformationen automatisch ablaufen können.

Um die Problematik des Findens von Transformationen zu vereinfachen, wurden Ansätze zur Vereinigung von Modellarten entwickelt. De Melo verwendet die UML und erstellt das Domänenmodell als UML-Klassendiagramm, das Aufgabenmodell als UML-Aktivitätsdiagramm und das Dialogmodell als UML-Zustandsdiagramm [Mel10]. Dies erlaubt ein einheitliches Vorgehen bei der Entwicklung des dynamischen Verhaltens. Die Einbindung oder Erzeugung von Präsentationsmodellen steht bei diesem Ansatz allerdings nicht im Fokus und geschieht nicht automatisch.

Ein weiterer Ansatz ist die Einführung einer Komposition mehrerer Modelle in einem so genannten *Benutzungsmodell*. Kern ist dabei das Aufgabenmodell. Dieses wird um Domänenmodell, Kontextmodell, Plattform- und Benutzermodell ergänzt. Als Notation wird die XML-basierte Auszeichnungssprache (ML, engl. *Markup language*) *useML*

verwendet [MSB11, Mei10]. Mit diesem Ansatz werden komplexe Transformationen in abstrakte Benutzerschnittstellen ermöglicht, die Erzeugung von Präsentations- und Dialogmodellen findet semiautomatisch statt. Die resultierenden Modelle bleiben abstrakt. Für eine kontrollierbare, vollständige Benutzerschnittstellenintegration zur Laufzeit bedarf es daher Erweiterungen.

### 4.6.3. Kombination von Domänen-, Dialog- und Präsentationsmodell

Bisherige Ansätze beziehen stark das Aufgabenmodell und deren Repräsentation, die Notation des Modells, ein. Aber gerade das Aufgabenmodell und deren hierarchische Strukturierung ist stark von der Person, die es entwickelt, abhängig. Jeder Modellierer hat sein eigenes mentales Aufgabenmodell in Form hierarchischer Unterteilung der auszuführenden Aufgabe [Seb88]. Folglich ist es schwierig, alleine aus dem Aufgabenmodell eine geeignete Repräsentation der späteren Dialoge und Präsentation der Benutzerschnittstelle einheitlich abzuleiten.

Für diese Arbeit werden die Aufgabenmodelle von der abstrakten Repräsentation getrennt. Dies hat den Vorteil, dass die äußerst wichtige, initiale Arbeit der Erstellung von Aufgabenmodellen in der für den jeweiligen Entwickler bevorzugten Form geschehen kann, beispielsweise als mentales, informelles Modell oder in einer beliebigen bereits vorhandenen Notationsform. Es fließt nicht in die automatische Weiterverarbeitung ein. Da das Dialogmodell als dynamische Komponente der Benutzerschnittstelle das Bindeglied zwischen Aufgabenmodell und Präsentationsmodell darstellt, muss hierauf ein besonderer Fokus gelegt werden. Es wird davon ausgegangen, dass das Aufgabenmodell in Dialogmodell, wie bereits erläutert, und Präsentationsmodell, in Form der Interaktionsobjekte als Repräsentation der Aufgaben, aufgeht. Durch die direkte Erstellung abstrakter Dialog- und Präsentationsmodelle kann also auf Zwischentransformationen, die häufig von Aufgabenmodellen ausgehen, verzichtet werden, und eine direkte Integration ins Zielsystem erfolgen. Dies kann durch die Modellabbildung des kombinierten, abstrakten Modells in die finale Benutzerschnittstelle erreicht werden. Eine weitere Voraussetzung, damit diese Modellabbildung funktionieren kann, ist, dass die Elemente der Applikationslogik in Form eines Domänenmodells vorliegen. Diese können dann an die Elemente des Präsentationsmodells geknüpft werden. Die Kombination von Domänen-, Dialog- und Präsentationsmodell stellt daher die Kernidee des Konzepts dar. Deren Form wird im weiteren Verlauf des Kapitels erarbeitet.

Ähnlich wie im CAMELEON Reference Framework, das in Kapitel 3 vorgestellt wurde, kann zwischen abstrakter Ebene, konkreten Zielkonzepten und finaler Benutzerschnittstelle unterschieden werden. Die Ebene der *Tasks and Concepts* entfällt allerdings mit dem oben beschriebenen Ansatz des kombinierten Modells und geht in einer gemeinsamen abstrakten Ebene auf. Dieser grundlegende Ansatz wurde bereits vorgestellt [MHW+09] und von de Melo mit dem Fokus der abstrakten Interaktionsmodellierung mit UML weiter erarbeitet [Mel10]. Wie im vorigen Abschnitt erwähnt, beinhaltet dieser Ansatz keine automatisierbare Einbindung von Präsentationsmodellen, um direkt in eine finale Benutzerschnittstelle zu integrieren. Dies ist Voraussetzung für die Lösung der Problemstellung in dieser Arbeit und wird daher im weiteren Verlauf der Arbeit adressiert. Die Einordnung des Ansatzes eines kombinierten abstrakten Modells und deren Integration in die finale Benutzerschnittstelle anhand des CAMELEON Reference Framework wird in Abbildung 4.7 dargestellt. Auf das kombinierte abstrakte Interaktionsmodell werden Transformationsregeln angewandt, um die Benutzerschnittstelle direkt in finale Benutzerschnittstellen zu integrieren. Die Transformationsregeln umfassen direkte Modellabbildungsvorschriften und können auch Bedienmuster (engl. *Patterns*) für die Interaktion, Gestaltungs- und Designregeln, wie etwa von Pederiva et al. als Beautifications beschrieben [PVn+08], enthalten. Dadurch wird eine kontrollierbare Integration in das jeweilige Zielsystem angestrebt.

### 4.6.4. Zwischenbewertung zur Erfüllbarkeit der Anforderungen

Nach der Erarbeitung des generellen Ansatzes erfolgt in diesem Abschnitt eine kurze Zwischenbewertung zur Erfüllung der Anforderungen. Bevor der Ansatz weiter detailliert wird, wird dadurch die Richtung des generellen Ansatzes im Hinblick auf die Erfüllbarkeit der Anforderungen gefestigt.

**Modellierung: Unabhängigkeit, Anpassbarkeit, Gebrauchstauglichkeit** Der generelle Ansatz sieht vor, dass Plattform-, Benutzer- und Kontextmodelle nicht in die Modellierung des abstrakten Modells einfließen und das Domänenmodell flexibel und separat definiert werden kann. Diese Vorgehensweise ist im Einklang mit den Anforderungen für die Modellierung an Unabhängigkeit von Plattformen, Interaktionsgeräten oder eingesetzten Programmiersprachen sowie an die Anpassbarkeit durch flexible Definition der Anwendungsdomäne. Die Gebrauchstauglichkeit kann an dieser Stelle noch nicht

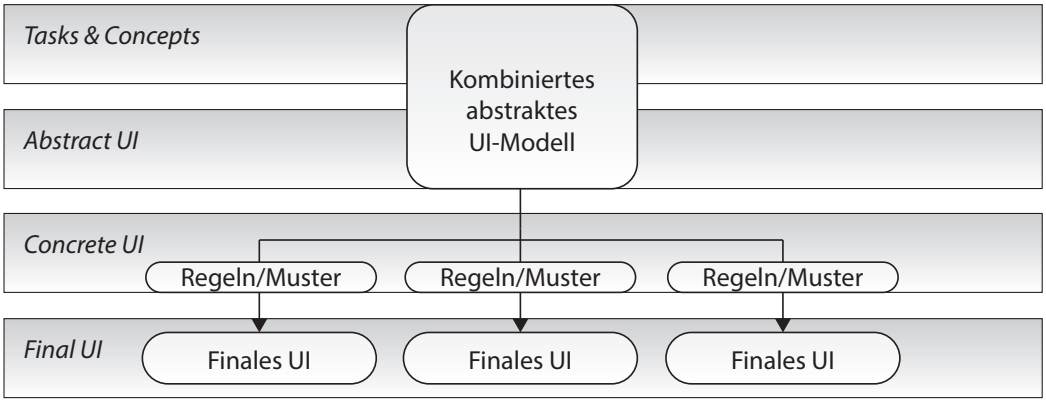


Abbildung 4.7.: Einordnung des kombinierten abstrakten Interaktionsmodells in die Abstraktionsebenen des CAMELEON Reference Framework. Die Ebenen der Aufgaben und Konzepte und der abstrakten Benutzerschnittstelle werden zusammengelegt. Die Transformationsregeln enthalten konkrete Informationen über das Zielkonzept, in das die modellierte Benutzerschnittstelle integriert werden soll.

genauer betrachtet werden und muss anhand bewährter Modellierungsformen und den Erfahrungen damit im praktischen Einsatz bei der weiteren Konzeptentwicklung betrachtet und bewertet werden.

**Integration: Vollständigkeit, Eindeutigkeit, Kontextsensitivität, Automatisierbarkeit** Der generelle Ansatz schränkt die Vollständigkeit der Abdeckung abstrakter Interaktionselemente nicht ein. Die Eindeutigkeit der Abbildung eines abstrakten Modells auf das Zielsystem lässt sich über klar definierte 1:1-Basisabbildungen regeln. Beide Aspekte müssen bei der weiteren Konzeptentwicklung genauer betrachtet werden. Kontextsensitivität sowie Automatisierbarkeit der Integration kann über die Verwendung einer strukturierten Beschreibungssprache erreicht werden. Kontextinformationen können damit über bestimmte Parameter definiert werden. Automatisierbarkeit der Transformation ist auf der Basis einer strukturierten, maschinenlesbaren Beschreibungssprache möglich.

**Entwicklungsprozess: Formalität, Testbarkeit** Durch eine klar definierte Struktur der Beschreibungssprache sowie der Transformationsregeln kann Formalität und die Testbarkeit des automatischen Integrationsprozesses bei der weiteren Entwicklung des Konzepts sichergestellt werden.

**Fazit** Insbesondere die Anforderungen an Unabhängigkeit und Anpassbarkeit der Modellierung werden mit dem generellen Ansatz klar adressiert. Die Erfüllung der weiteren Anforderungen für die Modellierung, Integration sowie den Entwicklungsprozess kann auf Basis des generellen Ansatzes erreicht werden. Im Folgenden wird der Ansatz daher als Startpunkt für die detaillierte Ausarbeitung, Anwendung und Bewertung des Konzepts verwendet.

## 4.7. *aDDaPt* - abstraktes Domänen-, Dialog- und Präsentationsmodell

Nach dem Überblick über den generellen Ansatz und deren Entstehung folgt in diesem Abschnitt die detaillierte Vorstellung des abstrakten Interaktionsmodells, das aus der Kombination eines Domänen-, Dialog- und Präsentationsmodells besteht. Es wird im Folgenden mit ***aDDaPt*** als Abkürzung für ***a****bstract* ***D****omain,* ***D****ialog* ***a****nd* ***P****resen****t****ation model* bezeichnet.

### 4.7.1. Domänenmodell

Die Domäne wird in *aDDaPt* durch die Definition von Klassen, in denen Grunddatentypen sowie weitere eigene Typen und Datenstrukturen enthalten sind, beschrieben. Dadurch wird die Beschreibung einer komplexen Domäne aus Datensicht möglich. Im Gegensatz zur bestehenden Sicht auf die Domänenmodellierung, die ein für die Plattform spezifisches Funktionsmodell beinhaltet [Reu03], wird auf plattformspezifische Informationen verzichtet. Das Domänenmodell soll unabhängig für alle Plattformen gelten.

Für diese Arbeit ist das Domänenmodell der erste wichtige Schritt für die spätere Ableitung geeigneter Benutzerschnittstellenelemente. Das Wissen über die dort verwendeten Datentypen muss auf der Zielplattform verarbeitbar sein, also im Regelsatz zur Benutzerschnittstellengenerierung integriert sein. Ein Domänenmodell besteht aus den

verschiedenen Teilen der Domäne mit ihren jeweiligen Attributen und deren Typen. Attribute beschreiben die Domäne aus Datensicht. Sie haben jeweils einen Typ, der entweder ein Basisdatentyp ist, ein eigens definierter komplexerer Typ oder eine Sammlung von Werten eines bestimmten Typs (*Collection*). Basisdatentypen sind fest definiert. Es können aber weitere benötigte Typen dem Basissatz hinzugefügt werden. Für diese Arbeit wurden **Boolean** (Wahrheitswert), **String** (Zeichenkette), **Number** (Zahlenwert) und **Image** (Repräsentation eines Bildes) als Basisdatentypen festgelegt. Dieser Satz hat sich als ausreichend für diese Arbeit herausgestellt, da beliebige komplexere Datentypen damit definiert werden können.

Domänen können in Form von Klassen dargestellt werden. Eine mögliche grafische Darstellung und Notationsform ist daher die Klassendiagramm-Notation der UML. Da bei *aDDaPt* davon ausgegangen wird, dass die eigentliche Anwendungslogik auf dem bereits zugeordneten System oder Gerät bleibt, muss das Domänenmodell nur die für die Erstellung der Benutzerschnittstelle relevanten Elemente beinhalten. Die Definition der gültigen Datenbereiche wird beispielsweise von der Applikation bereitgestellt und muss nicht im *aDDaPt*-Domänenmodell neu definiert werden. Auch beinhaltet das Domänenmodell nur die Methoden, die für die abstrakte Beschreibung der Benutzerschnittstelle relevant sind. Abbildung 4.8 illustriert Aufbau und Darstellung eines *aDDaPt*-Domänenmodells anhand von Beispielattributen und -typen, sowie, falls benötigt, Methoden.

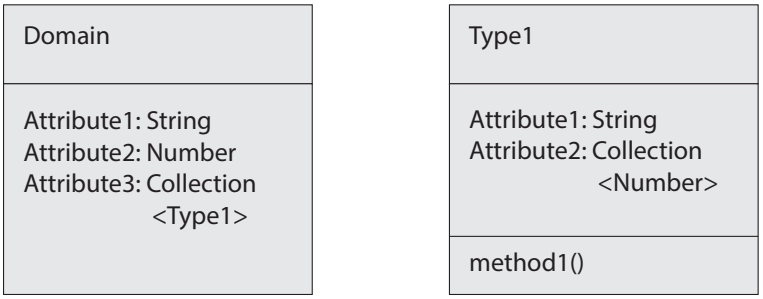


Abbildung 4.8.: *aDDaPt Domain model*: Darstellung mit Attributen inklusive einer Sammlung von Werten (*Collection*) eines eigenen Datentyps (*Type1*), der mit mehreren Attributen und einer Methode (*method1*) definiert ist.

### 4.7.2. Dialogmodell

Das Dialogmodell beschreibt das dynamische Verhalten, also den Ablauf der Interaktion zwischen Nutzer und Computer. Im Gegensatz zu einigen Ansätzen, in denen das Dialogmodell in abstraktere sowie konkretere Teile aufgespalten wird [Mei10], wird im *aDDaPt*-Ansatz nur genau ein Dialogmodell vorgesehen, da die konkrete Ausprägung der Benutzerschnittstelle in der Zielumgebung zum Modellierungszeitpunkt nicht bekannt ist.

Im Dialogmodell geben Zustände zusammen mit Nutzer- und Systemereignissen, die zu Zustandsübergängen führen, den Rahmen für die verschiedenen Interaktionsmöglichkeiten mit der Anwendung vor.

Nutzerereignisse sind Ereignisse, die vom Nutzer ausgehen. **ACT <Name> [Boolean]** bezeichnet eine allgemein mögliche Nutzeraktion, die abhängig von einem Wahrheitswert aktiv oder inaktiv ist. **SELECT(<Number>:<identifier>) <Collection>** bezeichnet die Auswahl einer bestimmten Anzahl (<Number>) von Elementen aus mehreren Elementen (<Collection>) als konkrete Nutzeraktion. Der Bezeichner <identifier> wird vergeben, um die Auswahl im Modell referenzieren zu können.

Der Ansatz erlaubt auch die Verwendung abstrakter Semantik, wie etwa mit der Transition **MODIFY <Type>**. Diese bezeichnet eine Nutzeraktion, die ein Datenobjekt (vom Typ <Type>) verändert. Bei der Verwendung von Transitionen, die abstrakte Semantik tragen, muss sichergestellt sein, dass diese abstrakte Semantik definiert ist. Ein erarbeiteter Satz solcher semantischer Transitionen findet sich im Anhang B.

Systemereignisse werden mit **SYS <Name> [Boolean]** gekennzeichnet. Sie sind an einen Wahrheitswert gebunden. Ist der Wert zu einem entsprechenden Systemereignis, das zu einer Transition gehört, die aus dem aktiven Zustand heraus führt, wahr, wird der entsprechende Zustandsübergang ausgeführt.

Die Bedingungen, Boolean-Werte, an Transitionen, werden wie in Programmiersprachen ausgedrückt. Zusätzlich können spezielle Alias-Bezeichner für entsprechende Boolean-Ausdrücke zum Zwecke einer lesbareren Modellierung definiert und verwendet werden. Ein Beispiel dafür, das in dieser Arbeit verwendet wurde, ist die Bedingung *[notEmpty(<Attribut>)]*, welche wahr ist, wenn <Attribut> gesetzt wurde.

An einer Transition kann zusätzlich eine Aktion angegeben werden, die beim Zustandsübergang ausgeführt wird. Aktionen können nur Methoden aus dem Domänenmodell

sein, falls welche definiert sind. Wie oben erwähnt wird mit *aDDaPt* nicht die komplette Anwendung neu modelliert, daher werden nur Aktionen angegeben, die für die Benutzerschnittstelle relevant sind.

Abbildung 4.9 illustriert den generellen Aufbau des *aDDaPt*-Dialogmodells als Zustandsdiagramm.

Abbildung 4.9.: *aDDaPt Dialog model*: In einem Zustandsdiagramm wird das dynamische Verhalten durch Ereignisse beschrieben, die an optionale Bedingungen geknüpft sind, und Interaktionszustandsübergänge sowie Aktionen auslösen können.

### 4.7.3. Präsentationsmodell

Das Präsentationsmodell ergänzt das Dialogmodell um Präsentationsobjekte. Es wird zwischen Objekten der Präsentation ohne Interaktion sowie Objekten, die Interaktion beinhalten, unterschieden. Die Objekte verwenden die Attribute, die im Domänenmodell definiert wurden.

Objekte der Präsentation ohne Interaktion werden mit **PRESENT <Type>** bezeichnet. Das dem Nutzer präsentierte Objekt vom Typ <Type> ist dabei ein Attribut, das im Domänenmodell definiert wurde. Entsprechend des Zielkonzepts soll dieses Objekt also dem Nutzer in der Benutzerschnittstelle präsentiert werden, sofern der das Objekt beinhaltende Zustand aktiv ist.

Objekte, die Interaktion durch den Nutzer erwarten, werden mit **PROVIDE <Type>** bezeichnet. Der Nutzer soll dabei das Objekt vom Typ <Type> bereitstellen (engl. *to provide*). Dies geschieht durch entsprechende Eingabemöglichkeiten, welche vom Zielkonzept sowie dem erwarteten Typ abhängen. Als besonderer Fall ist hier auch die

Bestimmung einer Anzahl (<Number>) von Elementen für eine Datensammlung (<Collection>) möglich: **PROVIDE (<Number>:<identifier>) <Collection>**. Der Bezeichner <identifier> wird vergeben, um die neuen Elemente im Modell referenzieren zu können. Dies ist insbesondere wichtig, wenn es mehrere bereitzustellenden Auswahlen der gleichen Datensammlung in einem Zustand gibt.

Abbildung 4.10 illustriert die Verwendung von Präsentationsobjekten in den Zuständen.

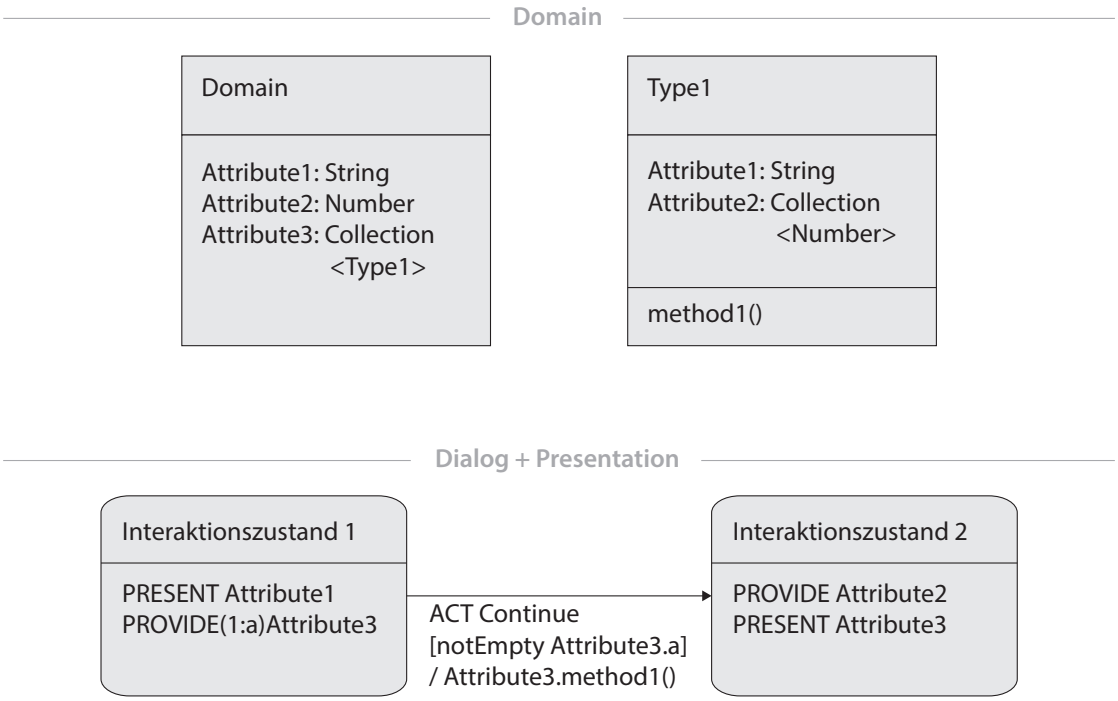


Abbildung 4.10.: *aDDaPt model*: Domänenmodell und Dialogmodell mit Präsentationsmodell. Beispiele zur Darstellung von Präsentationsobjekten sowie Beispiel für das Nutzerereignis ACT.

Die Elemente von *aDDaPt* werden im Folgenden in konkreten Beispielen verwendet, um das Konzept weiter zu veranschaulichen. Die vorgestellten Beispiele, Teile davon oder weitere ähnliche Beispiele werden außerdem im weiteren Verlauf der Arbeit wieder aufgegriffen.

### 4.7.4. Beispiel: ToDo-Liste

Die Beispielanwendung der *ToDo-Liste*, die bereits in Kapitel 3 verwendet wurde, wird in diesem Abschnitt mit *aDDaPt* beschrieben. In der Anwendung lassen sich Einträge zu einer Aufgabenliste hinzufügen und löschen.

Die Domäne für die ToDo-Liste verwendet einen Typ *Entry*, der das jeweilige ToDo beinhaltet (Attribut *description*). Die Einträge bilden dann eine Liste *entryList* vom Typ *Collection<Entry>*. Zusätzlich dazu kann die Domäne verschiedene weitere Attribute vorsehen. So gehört zur Anwendung beispielsweise der Inhalt von Beschriftungen oder Abfragen, in diesem Beispiel die Attribute *addLabel* und *confirmLabel* und außerdem ein Name für die Liste, um mehrere Listen voneinander zu unterscheiden. Das *aDDaPt Domain model* für die Anwendung *ToDo*-Liste ist in Abbildung 4.11 dargestellt.

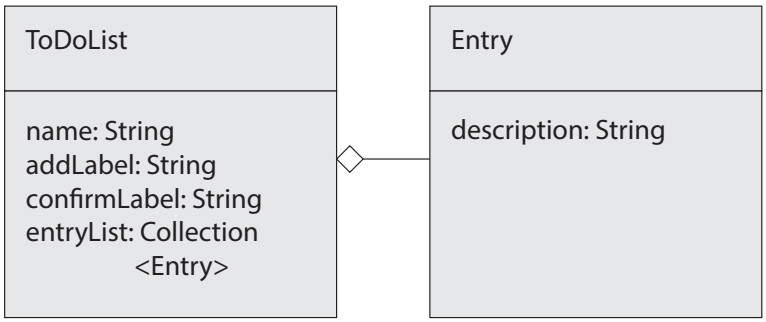


Abbildung 4.11.: *aDDaPt Domain model* für die Beispielanwendung ToDo-Liste

Das abstrakte Dialog- und Präsentationsmodell wird, wie vorgestellt, in einem Zustandsdiagramm modelliert, welches Attribute des Domänenmodells verwendet. Abbildung 4.12 enthält Dialog- und Präsentationsmodell der Anwendung. Startzustand ist *ToDo List*, die Darstellung der Listeneinträge. Von hier können die Zustände *Add Entry* und *Delete Entry* erreicht werden, um einen Eintrag hinzuzufügen oder einen ausgewählten Eintrag zu löschen. In diesen Zuständen werden die entsprechenden Präsentationsobjekte definiert, die den Interaktionszustand zusammen mit den Transitionen und den an diese geknüpften Bedingungen vollständig definieren.

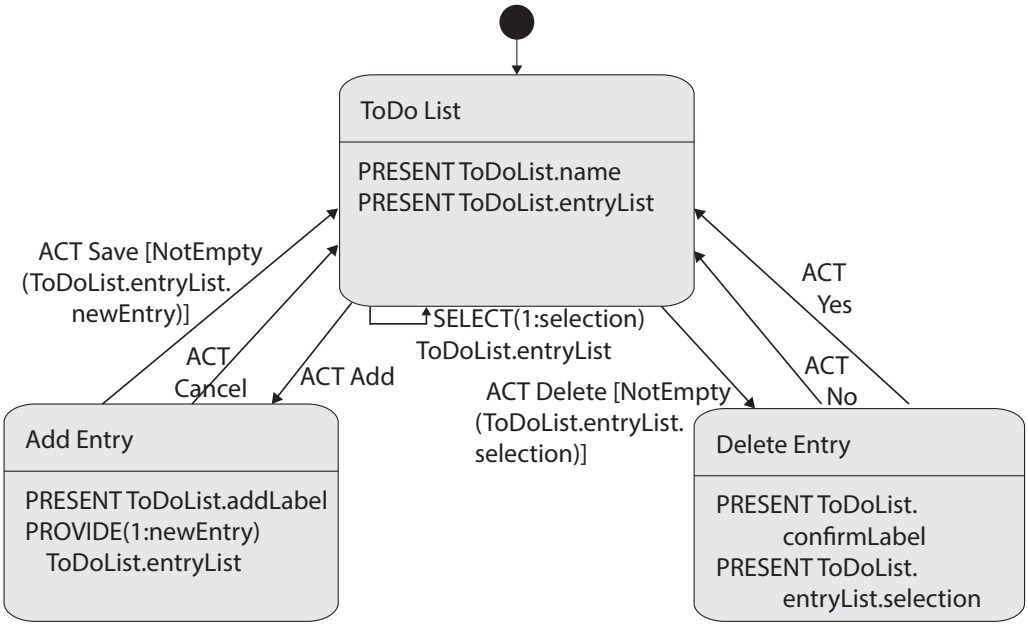


Abbildung 4.12.: *aDDaPt* Dialog- und Präsentationsmodell für die Beispielanwendung ToDo-Liste

### 4.7.5. Beispiel: JoinMyRide

Eine Mitfahrgelegenheit-Anwendung, *JoinMyRide*, wird als weiteres Beispiel für die abstrakte Modellierung sowie weitere Aspekte des *aDDaPt*-Konzepts aufgegriffen. *JoinMyRide* wurde als Beispiel-Anwendung zur Evaluierung von *aDDaPt* und weiterer darauf aufbauender Konzepte entworfen und umgesetzt [Dei11]. Die Funktionalität aus Nutzersicht lässt sich folgendermaßen beschreiben: Der Nutzer, der eine Mitfahrgelegenheit anbieten möchte, kann eine Fahrt anlegen mit Start, Zwischenstopps und Ziel sowie Datum und Zeit der Fahrt. Potenzielle Mitfahrer sehen diese Fahrt und können ihr Interesse dafür über die Anwendung kundtun. Der Anbieter der Fahrt kann die Liste potenzieller Mitfahrer sehen, eine Fahrt bearbeiten, also Details der Fahrt ändern, oder die Fahrt löschen. Abbildung 4.13 beinhaltet zunächst das Domänenmodell. Die Domäne JoinMyRide beinhaltet darin eine Liste der angelegten Fahrten sowie verschiedene Beschriftungen für das Hinzufügen und Bearbeiten einer Fahrt. Fahrt und Mitfahrer sind als Typen definiert sowie die Attribute für Datum und Zeit. Als Datum sowie als Zeit wird abstrakt ein String, date-string sowie time-string, definiert. Zu einem Datum gehören

dabei logisch ein Monat aus einer Liste von Monaten sowie ein Tag, zu einer Zeit gehören Stunden und Minuten. An diesem Beispiel sieht man, dass das *aDDaPt*-Domänenmodell lediglich die abstrakte Datensicht, die für die Erstellung einer Benutzerschnittstelle benötigt wird, nämlich das Auswählen von Datum und Zeit, enthält. Es gibt keine Angabe über gültige Werte oder sonstige Eigenschaften, da diese Logik von der Anwendung kommt.

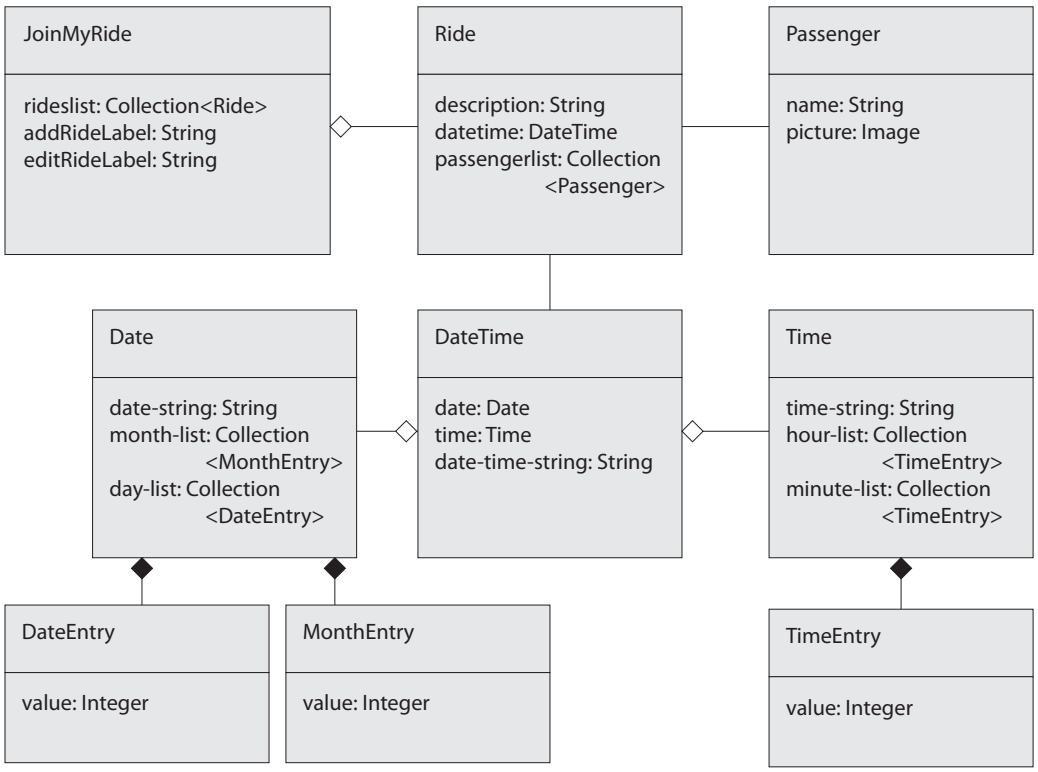


Abbildung 4.13.: *aDDaPt Domain model* für die Beispielanwendung JoinMyRide

Abbildung 4.14 enthält die *aDDaPt* Dialog- und Präsentationsmodellierung. So startet die Anwendung in der Übersicht der angelegten Fahrten (*Rides Overview*). Von hier kann dann eine Fahrt hinzugefügt (*Add Ride*) oder die Details einer Fahrt angesehen werden (*Ride Details*). Ist die Detailansicht aktiv, ist vorgesehen, dass der Nutzer die Details der Fahrt ändern kann. Außerdem kann er aus der Detailansicht in die Ansicht der Passagierliste (*Passengers Overview*) wechseln, die zur jeweiligen Fahrt gehört.

Es ist in diesem Beispiel nur die Ansicht der Passagierliste vorgesehen, nicht aber, die Passagierliste zu manipulieren, zum Beispiel einen potenziellen Passagier zu bestätigen oder zu löschen. Die Erweiterung darum im *aDDaPt*-Modell wäre jedoch einfach möglich.

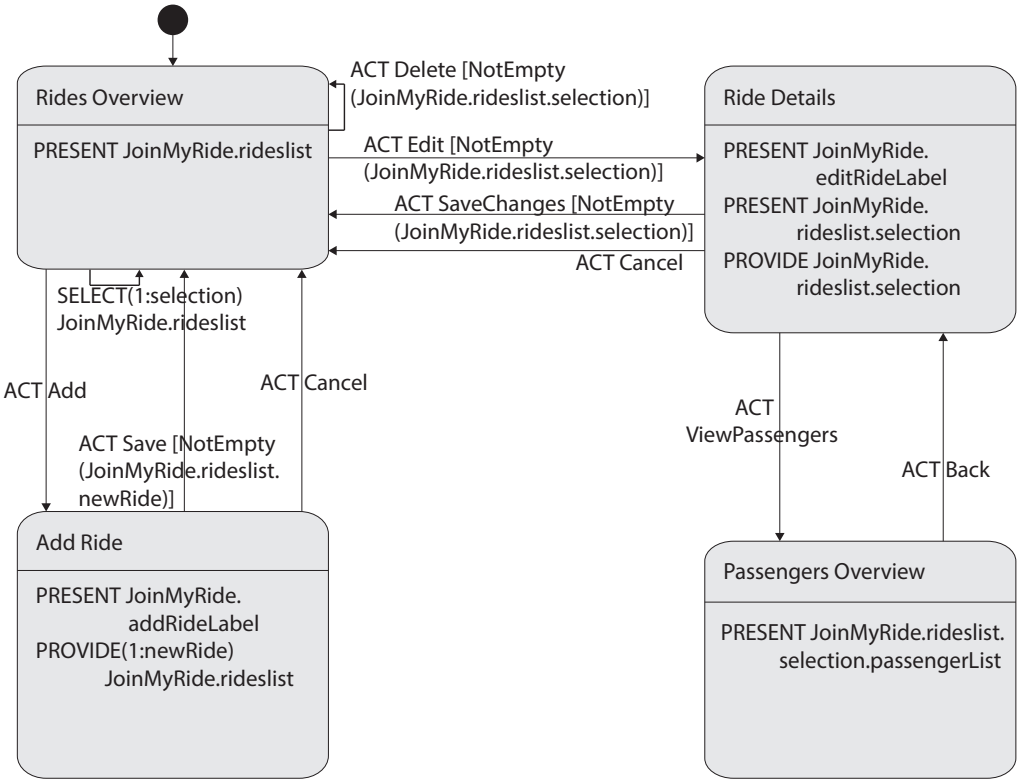


Abbildung 4.14.: *aDDaPt* Dialog- und Präsentationsmodell für die Beispielanwendung *JoinMyRide*

### 4.7.6. Wiederverwendung von Modellen

Neben der Modellierung konkreter Anwendungen ist die abstrakte Modellierung besonders auch für allgemeine Interaktionselemente, die von mehreren Anwendungen verwendet werden, interessant. So ist beispielsweise die Kernidee einer ToDo-Liste bezüglich der Datenstruktur und zumindest Teilen der Interaktion die gleiche wie die einer Mitfahrerliste. Ähnliche Listen kommen in einer Vielzahl weiterer Anwendungen vor. Daher liegt es nahe, oft verwandte Konzepte der Interaktion als wiederverwendba-

re Komponenten vorzubereiten. Abbildung 4.15 illustriert das Modell einer derartigen generischen Liste, im gezeigten Fall mit der Möglichkeit für den Nutzer zwischen zwei Optionen je Eintrag auszuwählen.

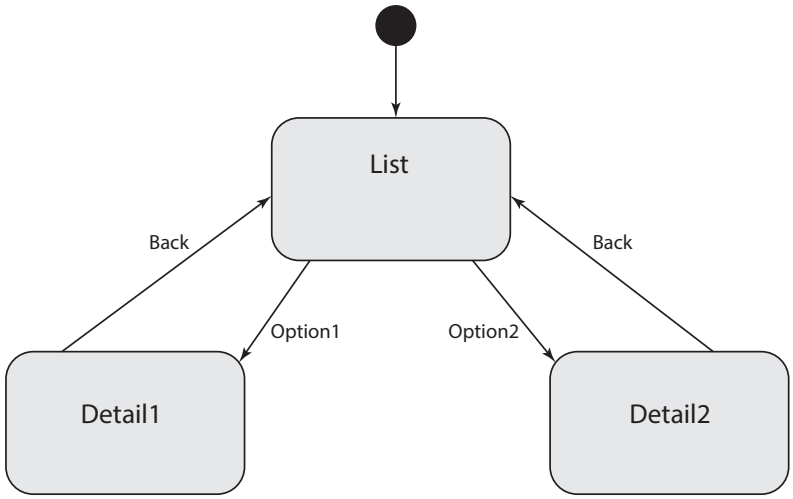


Abbildung 4.15.: Vereinfachte Darstellung eines generisches Listenmodells für eine Liste, bei der zwei Optionen je Eintrag auswählbar sind.

Ein weiteres Beispiel ist eine Komponente zur Auswahl eines Datums und/oder Zeit, die als abstraktes Modellierungselement in zahlreichen Modellen wiederverwendet werden könnte.

Wiederverwendbare Modelle der Interaktion sind insbesondere für Zielsysteme interessant, die strikten Rahmenbedingungen unterliegen. Dazu gehört das Infotainmentsystem eines Automobils. Wegen der langen Lebensdauer eines Automobils sowie Anforderungen zur Sicherheit ist der Umfang der Interaktion mit dem Infotainmentsystem klar definiert. Die Einbindung weiterer Anwendungen nach der ersten Inbetriebnahme muss dieser Definition folgen.

*aDDaPt* ermöglicht die Flexibilität in der Modellierung und die Wiederverwendbarkeit bestehender Modellierungsmuster. Da sich viele Anwendungsfälle die abstrakte Inter-

aktionslogik teilen, ist es insbesondere für die Abbildung auf Zielkonzepte hilfreich, zu wissen, wie das generische Modell zu einer Interaktion aussieht. So können Transformationsregeln bestimmte Muster in der Modellierung erkennen, um etwa aus den für die Zielplattform verfügbaren Widgets das passende für die Integration der Benutzerschnittstelle auszuwählen.

Im folgenden Abschnitt wird der Prozess der Integration von *aDDaPt* in bestehende Benutzerschnittstellen erläutert, der auf diesem Prinzip der Erkennung von Mustern, um Regeln anzuwenden, aufbaut.

## 4.8. Integration von *aDDaPt* in bestehende Benutzerschnittstellen

### 4.8.1. Überblick

Für die Integration einer Applikation auf Basis eines *aDDaPt*-Modells fließen das bestehende Bedienkonzept und die Benutzerschnittstellenelemente des Zielsystems zusammen mit den *aDDaPt*-Elementen in die Erzeugung der finalen Benutzerschnittstelle ein. Abbildung 4.16 illustriert den Prozess.

Die Integration von *aDDaPt*-Modellen beginnt mit der Bereinigung und Vorbereitung des Modells, sodass im weiteren Prozess nur benötigte Elemente verwendet werden, also beispielsweise keine Informationen, die lediglich für die Ansicht des Modells als Diagramm relevant sind. Dieses bereinigte *aDDaPt*-Modell geht in die Erzeugung der Benutzerschnittstelle, die UI Generierung, ein. Das Zielsystem, in das integriert werden soll, stellt dazu seine Fähigkeiten in Form von Transformationsregeln (*UI rules*) bereit. Für das *aDDaPt*-Modell wird nun überprüft, ob die erforderlichen Fähigkeiten des Zielsystems vorhanden sind. Unstimmigkeiten werden durch eine Regelhierarchie, die auf unterster Ebene auf garantierte Basisabbildungen zurückgreift, aufgelöst. Die auf diese Weise eindeutig erzeugte Benutzerschnittstelle für das Zielsystem wird dann in diesem dargestellt. Wie die Kommunikation der Systemkomponenten funktioniert, damit die Benutzerschnittstelle bedienbar und aktuell gehalten wird, wird in Kapitel 5 erläutert. Im Folgenden wird genauer auf den Aufbau der Regeln eingegangen.

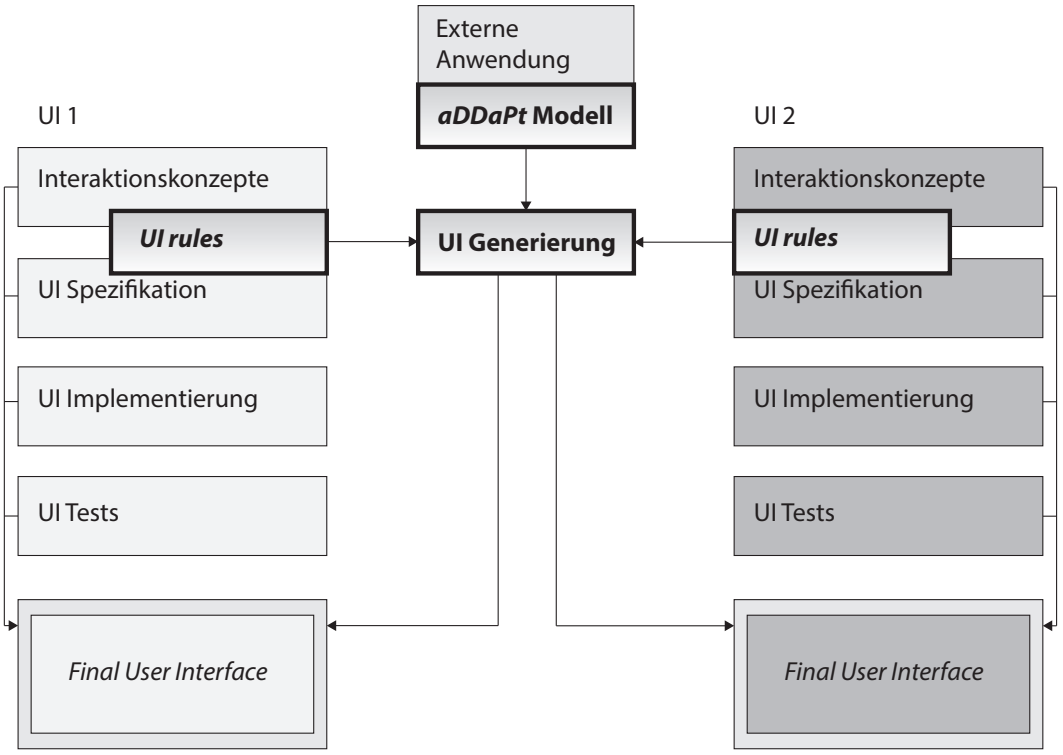


Abbildung 4.16.: Weiterverarbeitung von *aDDaPt* im Überblick. UI 1 und UI 2 sind dabei zwei verschiedene Benutzerschnittstellen von Zielsystemen.

### 4.8.2. *UI rules:* Layout und Darstellung

Wichtiger Teil der *UI rules* sind Informationen über mögliche Anordnungen von Benutzerschnittstellenelementen, Regeln für das Layout. Das Layout wird vom Zielsystem festgelegt und es gibt mindestens ein Basislayout, das für zu integrierende Benutzerschnittstellen ausschlaggebend ist. Abbildung 4.17 illustriert einen möglichen Aufbau eines solchen Basislayouts. Die Elemente PRESENT, PROVIDE sowie SELECT sind in diesem Beispiel zeilenweise angeordnet im Hauptfeld der Applikation. Einstiegspunkt in die jeweilige Applikation bietet die Titelzeile des Layouts, ein *Provider* steht dabei für eine bereitgestellte Applikation. Getrennt vom Hauptbereich in diesem Beispiel sind ACT-Transitionen, die sich als Einträge in einer unten angeordneten Menüzeile wiederfinden.

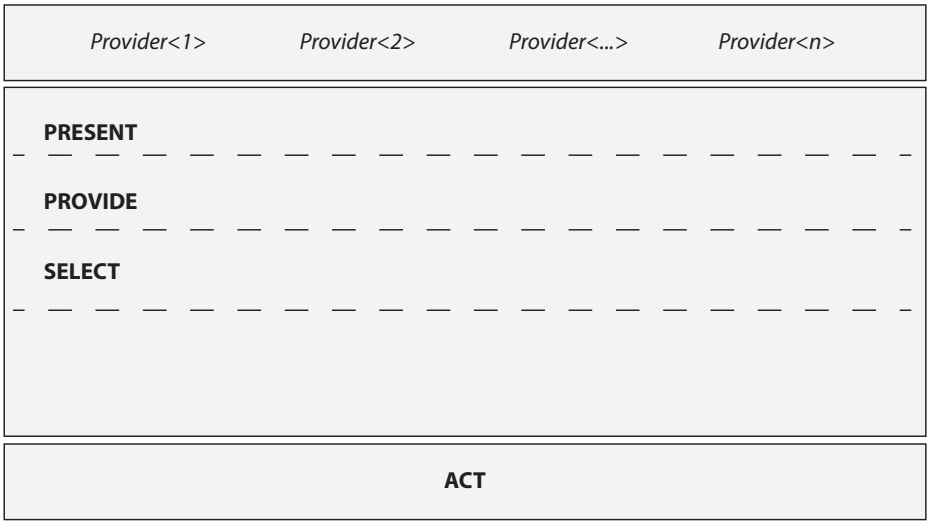


Abbildung 4.17.: *UI rules:* Beispiel eines Basislayouts: Verbundene Provider, das sind die Applikationen, sind in einer Zeile oben, ACT-Transitionen sind in einer Zeile unten, und PRESENT, PROVIDE und SELECT Elemente sind, logisch in Zeilen getrennt, im mittleren Hauptbereich dargestellt.

Neben einem Basislayout, welches immer dann Anwendung findet, wenn keine andere Layout-Regel greift, sind weitere Layouts denkbar, die vom Zielsystem bereitgestellt werden. Abbildung 4.18 illustriert ein Alternativlayout mit zwei logischen Spalten im mittleren Hauptbereich. In der linken Spalte stehen nur PRESENT-Elemente. Ein solches Layout könnte etwa für die Präsentation von Bildern mit kurzen Texten zu den Bildern vorgesehen werden. Dazu würde eine Regel bereitgestellt, die dieses Layout bei Zuständen der Anwendung vorsieht, in denen genau ein *Image*-Element sowie ein *String*-Element präsentiert werden sollen. Das Bild könnte beispielsweise links, die Beschreibung rechts, gemäß des Bedien- und Anzeigekonzepts, angeordnet werden.

Die Layout-Regeln können beliebig komplex sein und liegen in der Verantwortung der Zielumgebung. Voraussetzung ist lediglich, dass mindestens ein Basislayout vorgesehen ist, das keine Einschränkung der Anwendbarkeit durch eine Regel beinhaltet.

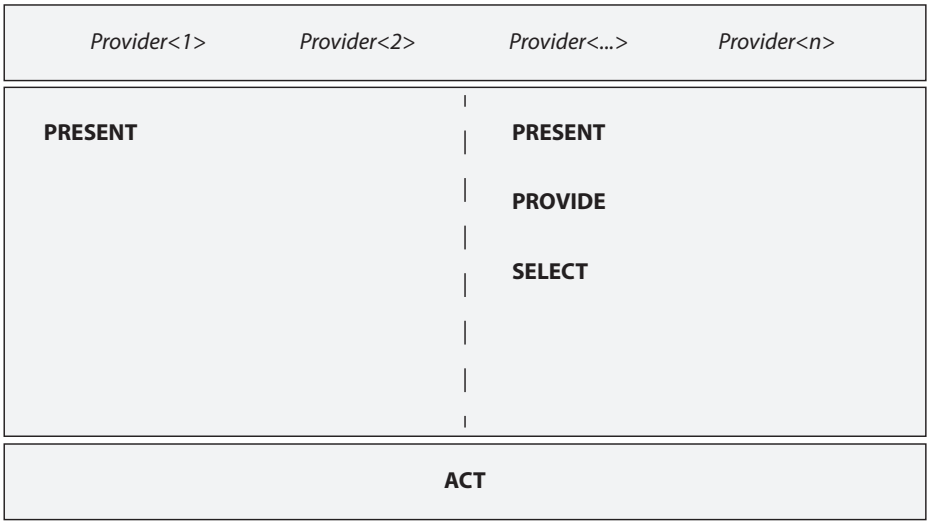


Abbildung 4.18.: *UI rules:* Beispiel eines alternativen Layouts: Der mittlere Hauptbereich ist in zwei logische Spalten eingeteilt, links können nur PRESENT-Inhalte präsentiert werden, rechts PRESENT, PROVIDE und SELECT Elemente.

### 4.8.3. *UI rules:* Hierarchische Regeln für Transformationen

Weiterer Bestandteil der *UI rules* ist die Hierarchie der Abbildungsregeln für *aDDaPt*-Elemente auf Elemente im Zielsystem. *UI rules* können aus Basisabbildungen, einfachen Regeln und kombinierten Regeln bestehen. Sind die Voraussetzungen für die Anwendung von kombinierten Regeln erfüllt, werden diese zuerst angewandt, danach einfache Regeln und erst, wenn keine sonstige Regel greift, wird auf die Basisabbildung, also eine direkte Abbildung des *aDDaPt*-Datentyps auf ein Element im Zielsystem, zurückgegriffen. Abbildung 4.19 illustriert das Konzept der hierarchischen Regeln.

*Beispiel der Anwendung der Regelhierarchie:* Eine Basisabbildung für eine ACT-Transition von Zustand A auf Zustand B sieht als Element des Zielsystems einen Button vor. Eine einfache Regel könnte besagen, dass wenn mehr als 4 ACT-Transitionen aus einem Zustand A hinausgehen, im Zielsystem ein Auswahlmenü anstelle von 4 Buttons verwendet wird. Diese Regel, wenn die Voraussetzung erfüllt ist, hat dann Vorrang vor der Basisabbildung.

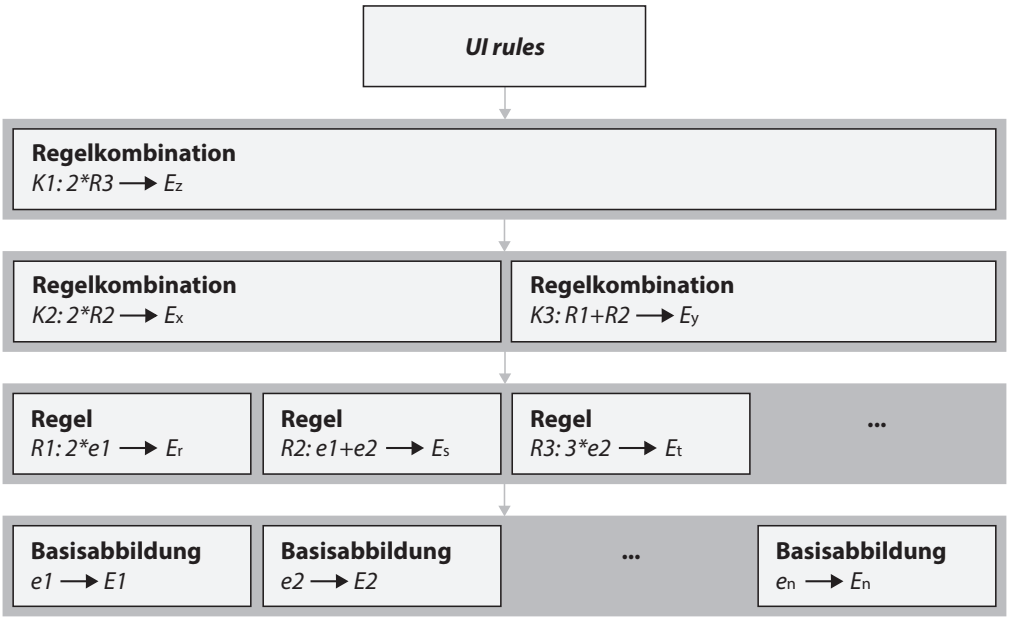


Abbildung 4.19.: *UI rules:* Konzept der Regelhierarchie. Kleinster gemeinsamer Nenner sind die Basisabbildungen, die 1:1 Elemente aus dem *aDDaPt*-Modell (e..) auf Elemente des Zielsystems (E..) abbilden. Darauf aufbauend sind Regeln (R..) und Kombinationen der Regeln (K..) möglich.

Da die *aDDaPt*-Modelle in einer XML-Struktur gespeichert und weitergegeben werden können, bietet sich für die Umsetzung der Regeln die Implementierung einer XSL-Transformation an. Aufgrund der Anforderung an Flexibilität sind die Umsetzungsmöglichkeiten aber nicht darauf beschränkt. Programmiertechnisch können die Regeln in jeder beliebigen Form und Programmiersprache als Satz logischer Funktionen umgesetzt werden.

### 4.8.4. UI-Generierung im Zielsystem

Die Generierung einer Benutzerschnittstelle im Zielsystem auf Basis eines *aDDaPt*-Modells unterliegt Vorbedingungen, die erfüllt werden müssen. Insbesondere müssen die minimalen Schnittstellen der Widgets festgelegt werden, um die Basisabbildung aus einem *aDDaPt*-Modell auf diese Widgets sicherstellen zu können. Die Festlegung der Layout-Regeln und Abbildungshierarchie baut darauf auf. Auf Basis der minimalen

Widgetschnittstellen können weitere Transformationsregeln definiert werden, die in die Regelhierarchie einfließen.

Die Entwicklung eines Benutzerschnittstellenkonzepts für ein Zielsystem, das später die Integration von *aDDaPt*-Modellen erlaubt, kann unabhängig erfolgen und muss keinem festgelegten Entwicklungsprozess folgen. Die Anforderung an Flexibilität ist damit erfüllt, da ein System auch zu einem späteren Zeitpunkt erweitert werden könnte. Dies bedeutet allerdings Mehraufwand, da für das bereits entwickelte Konzept formale Schnittstellen und Regeln erstellt werden müssen. Optimal ist daher, wenn das Zielsystem von Anfang an einem klaren Entwicklungsprozess folgt, der zum Ergebnis die definierten Widgetschnittstellen sowie die Regelhierarchie hat.

## 4.9. Bewertung anhand der Anforderungen

Das *aDDaPt*-Konzept wird in diesem Abschnitt anhand der Anforderungen bewertet. In Kapitel 6 werden im Rahmen der Umsetzungen einige Aspekte zur Erfüllung der Anforderungen nochmals verstärkt aufgegriffen. Die Bewertung an dieser Stelle ist daher auf das Notwendigste beschränkt, um das Konzept im weiteren Verlauf der Arbeit anwenden und tiefergehend evaluieren zu können.

### 4.9.1. Modellierung

**Unabhängigkeit** Die *aDDaPt*-Modellierung verwendet keine Elemente, die an ein bestimmtes Betriebssystem, Hardware oder eingesetzte Programmiersprachen gebunden sind. Ein *aDDaPt*-Modell wird erst in der Zielumgebung durch die Generierung in Programm-Code überführt, unter der Kontrolle des Zielsystems. Die Unabhängigkeit von Modalitäten liegt allerdings in der Verantwortung des Modellierers, der das Modell beispielsweise auf eine Modalität fokussieren kann, wenn dies gewünscht ist.

**Anpassbarkeit** Neben der Freiheit, das Modell auf bestimmte Modalitäten zu fokussieren, kann der Modellierer durch die Freiheit bei der Domänenmodellierung auch später noch Aspekte des Modells anpassen oder in eine andere Richtung erweitern. Dies wird durch den Abstraktionsgrad bei *aDDaPt* ermöglicht. Die Anpassbarkeit bei neuem oder erweitertem Domänenwissen ist damit gewährleistet.

**Gebrauchstauglichkeit** Bei der Entwicklung des *aDDaPt*-Konzepts wurde darauf geachtet, bewährte Methoden der Informatik zu verwenden, um die Gebrauchstauglichkeit beim Einsatz von *aDDaPt* zu gewährleisten. Klassen- und Zustandsdiagramme sind sehr verbreitete Bestandteile in der Entwicklung von Computersystemen und haben vermehrt Einzug in die Entwicklung von Benutzerschnittstellen erhalten, wie in Kapitel 3 bereits erläutert. Standardwerkzeuge aus dem kommerziellen und nicht-kommerziellen Bereich sichern das dort etablierte Mindestmaß an Gebrauchstauglichkeit in der Anwendung des Modellierungskonzepts.

### 4.9.2. Integration

**Vollständigkeit** Das Konzept der Regelhierarchie, das vorsieht, dass bei der Integration eines *aDDaPt*-Modells mindestens ein Basislayout sowie Basisabbildungen für jedes abstrakte Element auf ein Element des Zielsystems bestehen, stellt die Vollständigkeit der Integration sicher. Ein Modell einer Anwendung, welches Elemente beinhaltet, für die nicht mindestens eine Basisabbildung gilt, wird vom Integrationsprozess komplett abgelehnt.

**Eindeutigkeit** Die Eindeutigkeit der Abbildung wird durch den klar festgelegten Transformationsprozess sichergestellt. Die Regelhierarchie wird von kombinierten, komplexeren Regeln bis zu den Basisabbildungen durchlaufen, solange keine zutreffende Regel gefunden wird. Beim ersten Finden einer gültigen Regel wird diese angewandt. Dieser Prozess läuft immer gleich ab und das Ergebnis ist reproduzierbar.

**Kontextsensitivität** Die flexible Verwendung von Kontextinformationen der Zielplattform bei der Integration wird über die *aDDaPt*-Modellierung generell ermöglicht. Dazu können mehrere Domänenmodelle oder Dialogmodelle erstellt werden, die Kontextinformationen enthalten. Diese können dann verwendet werden, wenn die Zielplattform die zusätzlichen Informationen verarbeiten kann. Da immer ein allgemeines *aDDaPt*-Modell vorhanden sein muss, wird sichergestellt, dass die Integration auch in unbekannten Kontexten funktioniert.

**Automatisierbarkeit** *aDDaPt*-Modelle können in XML oder einer vergleichbaren Datenstruktur gehalten werden. Automatische Transformationen einer solchen Datenstruk-

tur in eine Struktur, die vom Zielsystem zur Anzeige der Benutzerschnittstelle verwendet wird, ist mit Standardmethoden, wie beispielsweise XSLT, möglich. Fehler im Modell oder beim Finden von Regeln führen nicht zu einem fehlerhaften Ergebnis sondern zur Ablehnung der Integration der Anwendung. Dies dient gleichzeitig der Absicherung gegen sicherheitskritische Fehler. Läuft die Transformation jedoch fehlerfrei durch, ist keine manuelle Bearbeitung mehr nötig.

### 4.9.3. Entwicklungsprozess

**Formalität** Formalität der Benutzerschnittstelle im Zielsystem, also der Darstellungsregeln und Definition der verwendeten Elemente, geht als Voraussetzung in den Entwicklungsprozess ein. Dies ermöglicht die Erstellung eindeutiger Abbildungsregeln für die Transformation aus einem *aDDaPt*-Modell, was die Formalität der Transformation gewährleistet.

**Testbarkeit** Sowohl das Zielsystem, in das integriert wird, als auch die Anwendung, die integriert werden soll, ist zunächst unabhängig testbar, da sich die Entwicklungen nicht gegenseitig beeinflussen. Die Testbarkeit der eigentlichen Integration wird dadurch sichergestellt, dass die Transformationsregeln des Zielsystems mit beliebigen Testanwendungen vollständig überprüft werden können. Ein fehlerhaftes Modell wird abgelehnt. Ob eine zu integrierende Anwendung an sich fehlerhaft ist, kann nicht vom Zielsystem überprüft werden, allerdings kann die Gefahr eines Fehlverhaltens, das andere Teile des Zielsystems beeinflusst, durch methodisches Testen des Integrationsprozesses vorab minimiert werden.

## 4.10. Fazit

Ein Konzept für abstrakte Interaktionsmodelle und deren Integration in bestehende Benutzerschnittstellen wurde auf Basis eines modellgetriebenen Ansatzes erarbeitet. Dadurch wird der Aufwand für Implementierung und Test einer Anwendung je Zielkonzept reduziert bei gleichzeitiger Erhaltung der Kontrolle über die Darstellung der Benutzerschnittstelle. Da Aufgabenmodelle nicht genügend eindeutige Informationen für einen vollautomatischen Integrationsprozess bereitstellen, wurde eine Kombination von

Domänen-, Dialog- und Präsentationsmodell als abstraktes Interaktionsmodell (***aDDaPt*** für **a**bstract **D**omain, **D**ialog **a**nd **P**resen**t**ation model) erarbeitet. Ein Domänenmodell beschreibt die Grunddatentypen und komplexen Datenstrukturen der Anwendung. Diese werden vom zentralen Modell der Interaktionsbeschreibung, dem Dialogmodell, verwendet. In den Definitionen der Dialogzustände werden abstrakte Präsentationsmodelle integriert. Zusammen entsteht das *aDDaPt*-Modell. Grundlage für die Form der Notation ist die Verwendung von *Harel's State Charts* [Har87], da diese relevante Interaktionszustände abbilden können. Die UML nutzt *State Charts*, weshalb sie als Sprache für die abstrakten Interaktionsmodelle in Frage kommt. Insbesondere die Möglichkeit der automatischen Verarbeitung ist bei Verwendung von UML sichergestellt. Für eine praktische Umsetzung in UML spricht außerdem die Werkzeugunterstützung. Dennoch ist die Form der Notation nicht an eine konkrete, standardisierte Modellierungssprache gebunden.

Die Integration von *aDDaPt*-Modellen in ein bestehendes Zielkonzept erfolgt durch einen automatischen Transformationsprozess. Auf Grundlage eines garantierten Basislayouts und Basisabbildungen, welche vom Zielsystem definiert werden, wird eine Benutzerschnittstelle aus dem *aDDaPt*-Modell generiert. Dadurch, dass mindestens eine Basisabbildung pro *aDDaPt*-Element vorhanden ist, wird die vollständige Abbildbarkeit sichergestellt. Eine Regelhierarchie, bestehend aus weiteren Regeln aufbauend auf Basislayout und Basisabbildungen, sorgt für besser auf das Zielsystem zugeschnittene Abbildungen.

Das *aDDaPt*-Konzept basiert auf einem ganzheitlichen Ansatz, der es ermöglicht, flexible Benutzerschnittstellen zu entwickeln, die neue Anwendungen vollautomatisch integrieren können, unabhängig von Ein- und Ausgabegeräten, verwandter Software oder Hardware. Im folgenden Kapitel wird eine Architektur vorgestellt, die eine solche flexible Integration ermöglicht.

# 5

# Eine Architektur zur Integration externer Anwendungen in bestehende Benutzerschnittstellen

Soll eine Anwendung, die, wie im vorigen Kapitel beschrieben, als abstraktes Interaktionsmodell vorliegt, in ein Zielsystem integriert werden, muss die Architektur des Zielsystems entsprechend dafür ausgelegt sein. Um die Integration von mit *aDDaPt* modellierten Anwendungen in einer realen Umgebung zu demonstrieren, wurde eine Architektur entwickelt und prototypisch umgesetzt. Die konzipierte Architektur dient als Referenzarchitektur für mehrere Umsetzungen, die aufeinander aufbauen. Als Basis für die Integration verschiedener Applikationen und Benutzerschnittstellen, die verschiedene Aspekte dieser Arbeit beleuchten, ist die Architektur ein wichtiger Beitrag für die Konzeptabsicherung.

In diesem Kapitel wird die Architektur beschrieben bevor in Kapitel 6 die Umsetzungen als wichtiger Teil der Evaluierung des Gesamtkonzepts vorgestellt werden.

## 5.1. Konzeption der Architektur

Zu integrierende Anwendungen formen zusammen mit den im Zielsystem bereits bestehenden Anwendungen ein gemeinsames System. Es handelt sich dabei um ein Netzwerk von individuellen Anwendungen, da bei *aDDaPt* davon ausgegangen wird, dass eine zu integrierende Anwendung autark existiert und zunächst auch nichts über das Zielsystem weiß. Peer-to-Peer und Client-Server Architekturen sind die beiden dominierenden Paradigmen moderner Netzwerarchitekturen [KR08, Tan03]. In einer Peer-to-Peer Architektur sind die Teilnehmer, die Clients, gleichberechtigt und kommunizieren direkt miteinander. Es gibt keine festgelegte, ständig verfügbare Infrastruktur. Dagegen wird bei einer Client-Server Architektur davon ausgegangen, dass der Server immer im Netzwerk verfügbar ist und damit eine feste Infrastruktur darstellt. Für den Anwendungsfall, dass eine externe Anwendung in ein bestehendes System integriert werden soll, ist die Client-Server Architektur daher besser geeignet.

Die Client-Server Architektur soll es ermöglichen, dass sich externe Anwendungen als Clients am System anmelden und mit dem Zielsystem kommunizieren können, um alle relevanten Informationen austauschen zu können. Im Sinne einer sauberen Trennung zwischen Anwendungsdaten und -logik sowie Präsentation, wie das Model-View-Controller (MVC) Pattern nahelegt [GJVH94, BHS07], wird in der konzipierten Architektur zwischen *View*-Clients zur Präsentation und *Model*-Clients zur Haltung der Anwendungsdaten und -logik unterschieden. Die Rolle des *Controllers* übernimmt der zentrale Server, indem er zwischen *View*-Clients und *Model*-Clients vermittelt.

Im Folgenden werden zunächst die Clients und Server in der Gesamtarchitektur im Detail vorgestellt. Zur Kommunikation zwischen Server und Clients wurde ein Nachrichtenformat entwickelt, das im darauf folgenden Abschnitt erläutert wird.

## 5.2. Client-Server-Architektur

Die Architektur beinhaltet einen zentralen *Server*, der Nachrichten der Clients annimmt, verarbeitet oder weiterleitet. Als Clients werden, wie oben angedeutet, View-Clients sowie Model-Clients unterschieden. View-Clients beinhalten alle Informationen, die zur Darstellung der Daten benötigt werden. Sie werden in der Architektur *Presentation-Clients* genannt. Model-Clients stellen jeweils die Anwendungslogik und -daten bereit

und werden in der Architektur *Provider-Clients* genannt. Zusätzlich enthält die Architektur *Input-Clients*. Dadurch wird es möglich, Eingabemodalitäten für ein Zielsystem flexibel zu halten. Insbesondere können dadurch verschiedene Eingabegeräte und deren Abhängigkeit zur Anwendung flexibel einbezogen werden, ohne Presentation- oder Provider-Clients zu manipulieren. Für einfache Eingabemodalitäten ist es ausreichend, dass der Server eine Nutzeraktion, die von einem Eingabegerät ausgeht, empfängt und Presentation- und Provider-Clients entsprechend informieren kann. Eingabegeräte, die beispielsweise haptisches oder visuelles Feedback anbieten, können weitere Informationen über Zustand und Änderungen an der Anwendung mitgeteilt bekommen. Die flexible Architektur ermöglicht dies, genauso wie die Erweiterung um weitere, bisher nicht vorgesehene Clients. Die Kommunikation innerhalb des Servers läuft über *Server-Addins*. Für die Clients existiert jeweils ein Server-Addin, welches beim Start des Servers gestartet wird. Es stellt damit die Schnittstelle zur Verbindung zwischen Clients und Server bereit. Weitere Server-Addins für weitere Clients können entsprechend entwickelt und ergänzt werden. Abbildung 5.1 illustriert die Client-Server-Architektur. Der folgende Abschnitt beschreibt zunächst die Clients im Detail.

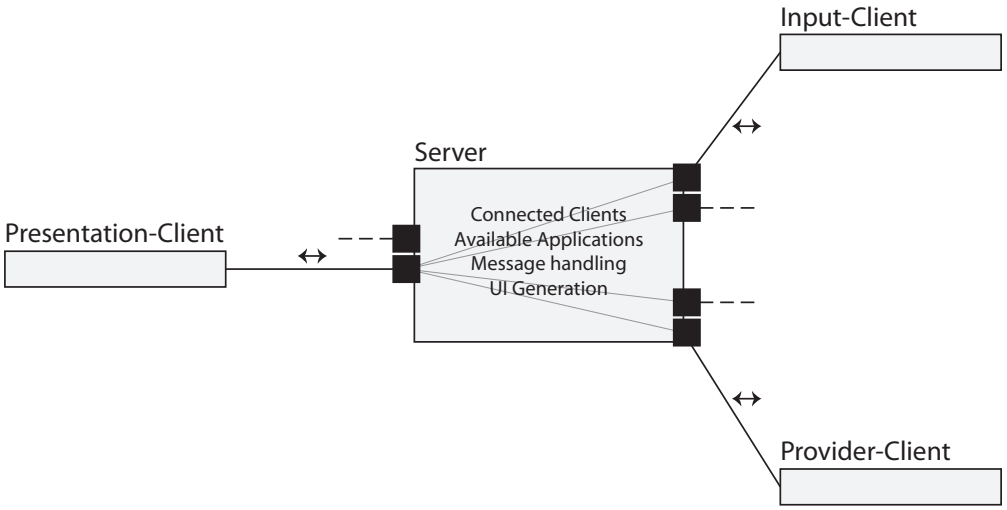


Abbildung 5.1.: Übersicht der Client-Server-Architektur mit dem Server und seinen Kernaufgaben, möglichen Clients und Kommunikationswegen. Server-Addins sind in Form der schwarzen Quadrate illustriert.

### 5.2.1. Clients

**Provider-Client** Ein Provider-Client ist eine externe Anwendung, die sich in ein bestehendes System integrieren soll. Ein Provider-Client muss ein *aDDaPt*-Modell beinhalten und dieses zum Server übertragen.

**Presentation-Client** Ein Presentation-Client stellt die Möglichkeiten, mit einer Anwendung zu interagieren, dar. Er ist nicht auf eine Ausgabemodalität beschränkt und es kann auch mehrere Presentation-Clients geben. Ein Presentation-Client ist nur für die Darstellung zuständig, die Anwendungsdaten und -logik bleiben Teil des Provider-Clients. Vom Nutzer vorgenommene Änderungen über den Presentation-Client müssen also an den Provider-Client geschickt werden, welcher die Daten dann verarbeitet oder zum Beispiel einen Zustandswechsel der Anwendung durchführt.

**Input-Client** Ein Input-Client regelt die Eingaben des Nutzers und leitet diese an den Server weiter. Von dort werden sie an den Presentation-Client geschickt, welcher sie verarbeitet und bei Bedarf reagieren kann. Für einige Eingabegeräte würde eine einseitige Kommunikation vom Input-Client zum Server ausreichen. Intelligentere Eingabegeräte, die vom Zustand der Anwendung abhängen, weil sie zum Beispiel Rückmeldungen an den Nutzer geben können, erfordern dagegen sowohl die Kommunikation zum als auch vom Server.

### 5.2.2. Server

Die Kernaufgabe des Servers ist die Sicherstellung der Kommunikation zwischen den Clients. Diese Kommunikation findet innerhalb des Servers zwischen den Server-Addins statt. Ein Server-Addin übermittelt Nachrichten, die vom verbundenen Client kommen, an den Server zur weiteren Bearbeitung. Der Server löst dann je nach Art der empfangenen Nachricht ein Event aus, auf das die Server-Addins reagieren können. Jedes Server-Addin muss sich also für die ihn interessierenden Events registrieren. Abbildung 5.2 illustriert die Komponenten innerhalb des Servers, die im Folgenden genauer beschrieben werden.

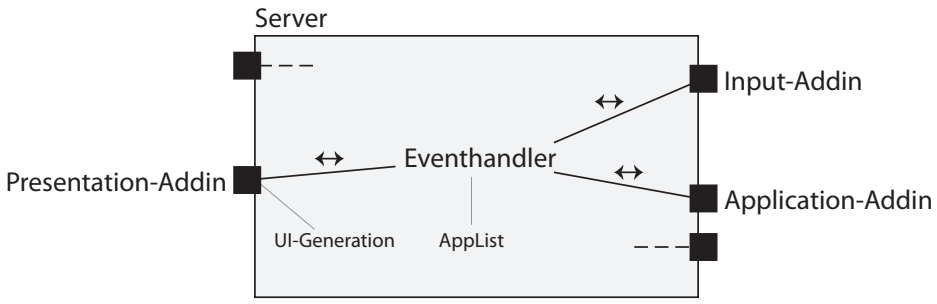


Abbildung 5.2.: Übersicht Server: Der Eventhandler im Server regelt Empfang und Verbreitung von Nachrichten. UI-Generation ist eine vom Server bereitgestellte Methode, die vom Presentation-Addin aufgerufen werden kann. AppList dient der Zwischenspeicherung von Informationen zu bekannten und verbundenen Applikationen.

**Server-Addins** Die Server-Addins stellen die Schnittstelle zwischen Client und Server dar. Ein Server-Addin ist dafür verantwortlich, Nachrichten des Clients an den Server weiterzuleiten und auf relevante Events im Server zu hören und zu reagieren. Die Kommunikation zwischen Server und Server-Addins läuft über ein definiertes Nachrichtenformat, welches im kommenden Abschnitt detaillierter vorgestellt wird. Die Kommunikation zwischen Client und Addin ist dagegen nicht festgeschrieben und hängt von vorhandenen Technologien, wie der Verfügbarkeit von Funknetz oder Bluetooth ab.

**UI-Generierung** Der Server ist in der Architektur für die UI-Generierung auf Basis des vom Provider-Client empfangenen *aDDaPt*-Modells verantwortlich. Das UI-Generierungsmodul ist flexibel gestaltbar mit der Voraussetzung einer einheitlichen Wissensbasis über die *aDDaPt*-Elemente zwischen Presentation-Client und Provider-Client.

## 5.3. Kommunikation im Server

Wie im vorigen Abschnitt erwähnt, wird innerhalb des Servers, also zwischen Server und Server-Addins, über ein definiertes Nachrichtenformat kommuniziert. Für diese Arbeit wurde dazu das Format der *TKMessage* definiert, wobei *TK* für *T*ransformation und *K*ommunikation steht. *aDDaPt*-Modelle müssen für die Integration in das Zielsystem

transformiert werden, und die relevanten Informationen dazu müssen zwischen den involvierten Clients kommuniziert werden. Die Server-Addins leiten TKMessage-Nachrichten vom Client an den Server weiter oder erstellen zu verschickende TKMessage-Nachrichten auf Basis einer empfangenen Nachricht des Clients, wenn dieser nicht im TKMessage-Format sendet. Das Server-Addin registriert sich außerdem beim Server für bestimmte Events, die von der Art des Clients abhängen. Vom Server empfangene Events leitet ein Server-Addin dann entsprechend an den verbundenen Client weiter.

### 5.3.1. Nachrichtenformat

Für diese Arbeit wurde ein Nachrichtenformat definiert, das der Kommunikation im Server dient. Format ist XML und jede Nachricht besteht mindestens aus dem Wurzelelement *TKMessage* sowie einem Attribut *Eventcode*, welches die Art der Nachricht festlegt. Weitere Elemente sind *Data*, *Transition* oder *Userinput* Tags. Von diesen drei Tags kann eine TKMessage nur jeweils eines enthalten, abhängig vom Eventcode.

**TKMessage** Ein Minimalbeispiel einer *TKMessage*-Nachricht mit Eventcode ist das folgende, das bei der Verbindung eines Input-Clients verschickt wird:

```
<TKMessage Eventcode="INPUT_CONNECTED" />
```

**Attribut: Eventcode** Für die Verbindung oder Trennung von Provider, Presentation oder Input-Clients sind folgende Eventcodes vorgesehen:

```
PROVIDER_CONNECTED / PROVIDER_DISCONNECTED
PRESENTATION_CONNECTED / PRESENTATION_DISCONNECTED
INPUT_CONNECTED / INPUT_DISCONNECTED
```

Der Eventcode `TRANSITION` deutet auf einen Zustandsübergang hin und enthält je nach Sender die Information über einen erfolgten Zustandsübergang oder die Anforderung eines solchen.

Der Eventcode `DATA_CHANGE` kennzeichnet eine Informationsnachricht über Datenbelegung, beispielsweise über Datenänderung, und wird zwischen Presentation- und Provider-Client ausgetauscht. `MODEL` als Eventcode kennzeichnet das *aDDaPt*-Modell

einer Anwendung. Dieses sowie die initiale Datenbelegung können bei Bedarf mit den Eventcodes `REQUEST_MODEL` und `REQUEST_DATA` angefordert werden. Benutzereingaben von einem Input-Client werden mit dem Eventcode `USER_INPUT` gekennzeichnet. Ein Input-Client kann außerdem Informationen über den Zustand der Anwendung anfordern und hat damit die Möglichkeit, Rückmeldung über diesen Zustand geben zu können. Für diese Anforderung wird der Eventcode `REQUEST_INPUTCONFIGURATION` verwendet, worauf ein Presentation-Client mit `CONFIGURE_INPUT` antwortet.

**Tag: Transition** Eine Nachricht mit dem Eventcode `TRANSITION` ist eine Aufforderung des Presentation-Client an den Provider-Client, den Applikationszustand zu wechseln. Über die Gültigkeit entscheidet der Provider-Client, da dort die Applikationslogik liegt. Wird die Transition ausgeführt, schickt der Provider-Client eine `TRANSITION`-Nachricht zurück, was einer Bestätigung der Anfrage gilt. Eine `TRANSITION`-Nachricht beinhaltet einen `Transition`-Tag und ist wie folgt aufgebaut:

```
<TKMessage Eventcode="TRANSITION">
   <Transition FromState="<FromState>"
      ToState="<ToState>"
      Identifier="<Identifier>" />
</TKMessage>
```

Jede Transition hat einen Ausgangszustand *FromState* und einen Endzustand *ToState*. Mit dem *Identifier* wird die Transition bezeichnet. Dieser Identifier kann entweder aus dem *aDDaPt*-Modell abgeleitet werden, der Name der Transition sein oder generisch erzeugt werden. *FromState* und *ToState* zusammen müssen eindeutig sein.

**Tag: Data** Daten werden im Tag `Data` verschickt, welcher in einer Nachricht mit dem Eventcode `DATA_CHANGE` enthalten sein kann. Voraussetzung für den Datenversand ist, dass Sender und Empfänger über die gleichen Datentypen verfügen, also die gleiche Sprache bezüglich der Daten sprechen. Mit dem `Data`-Tag können beliebige Daten verschickt werden. Durch das *aDDaPt*-Domänenmodell werden die Datentypen, die für die Anwendung relevant sind und ausgetauscht werden müssen, definiert.

Innerhalb des Data-Tags werden `TKData` für die einzelnen Daten mit den Attributen `Type`, `ID` und `Data` verwendet, wie im folgenden Beispiel illustriert:

```
<TKMessage Eventcode="DATA_CHANGE">
   <Data>
      <TKData Type="String" ID="myString">
         <Data>Text</Data>
      </TKData>
      <TKData Type="Integer" ID="myInteger">
         <Data>4711</Data>
      </TKData>
      <TKData Type="Collection" ID="myIntegerList">
         <Data>
         <!-- XML serialized List data ... -->
         </Data>
      </TKData>
   </Data>
</TKMessage>
```

Für komplexe Datentypen, wie *Collection*, kann der Inhalt des Data-Tags als ein XML serialisiert werden, wobei Steuerzeichen mit den jeweiligen HTML-Entities ersetzt werden. Dadurch können alle Daten als String versendet werden. Dies erleichtert die Kommunikation, da der XML-Serializer im Server nicht alle möglichen Datentypen kennen muss, was auch das Erweitern um Datentypen erleichtert.

**Tag: Userinput** Die Kommunikation von einem Eingabegerät, einem Input-Client, zum Presentation-Client läuft über ein `Userinput`-Tag, welches in Nachrichten mit dem Eventcode `USER_INPUT` enthalten ist. Mit dem Attribut *Type* wird die Art der Eingabe, also zum Beispiel der Bezeichner eines Eingabegerätes, beschrieben. Weitere Attribute sind *Action* sowie *Value*. Für eine Tastatureingabe sieht eine TKMessage mit dem Eventcode `USER_INPUT` beispielsweise so aus:

```
<TKMessage Eventcode="USER_INPUT">
   <Userinput Type="KEYBOARD" Action="PRESS" Value="A" />
</TKMessage>
```

### 5.3.2. Verbinden von Clients

Über das oben beschriebene Nachrichtenformat können sich Clients zum Server verbinden. Dazu schickt ein Client dem Server eine TKMessage mit dem Eventcode `PROVIDER_CONNECTED`, wenn es sich um einen Provider-Client handelt, oder entsprechend `INPUT_CONNECTED` oder `PRESENTATION_CONNECTED` für Input- und Presentation-Clients.

Der Server entscheidet je nach Typ eines sich anmeldenden Clients über den weiteren Nachrichtenaustausch. Von einem Provider-Client muss per `REQUEST_MODEL` das *aDDaPt*-Modell angefordert werden, und ein Presentation-Client muss über alle vorhandenen Provider-Clients informiert werden. Dazu kann der Server eine TKMessage mit Eventcode `PROVIDER_CONNECTED` an einen verbundenen Presentation-Client schicken, welcher daraufhin das *aDDaPt*-Modell für die Präsentation der entsprechend generierten Benutzerschnittstelle anfordert. Zusätzlich zum *aDDaPt*-Modell fordert ein Presentation-Client mit `REQUEST_DATA` die initiale Datenbelegung an, was für die Darstellung der Anwendung erforderlich ist. Der Provider-Client beantwortet diese Anfrage mit `TRANSITION` und `DATA_CHANGE` Nachrichten, die vom Server an den Presentation-Client weitergereicht werden. Daraus bezieht der Presentation-Client dann den Zustand der Anwendung und Datenbelegungen.

Verbindet sich ein Input-Client und fordert dieser mit `REQUEST_INPUTCONFIGURATION` weitere für ihn relevante Informationen an, muss jeder Presentation-Client darauf entsprechend mit `CONFIGURE_INPUT` antworten. Diese Information muss ebenfalls bei der Neuanmeldung eines Presentation-Clients ausgetauscht werden. Dieser Austausch wird mit der Nachricht `REQUEST_INPUTCONFIGURATION` entsprechend von jedem Input-Client an jeden neu verbundenen Presentation-Client eingeleitet. Der Server dient dabei als Weiterleiter der Nachrichten.

Abbildung 5.3 beinhaltet eine Veranschaulichung des Nachrichtenaustauschs bei der Verbindung eines dem Server nicht bekannten Provider-Clients. Interaktionssequenzdiagramme der weiteren Situationen beim Verbindungsaufbau zwischen Clients und Server befinden sich in Anhang A.

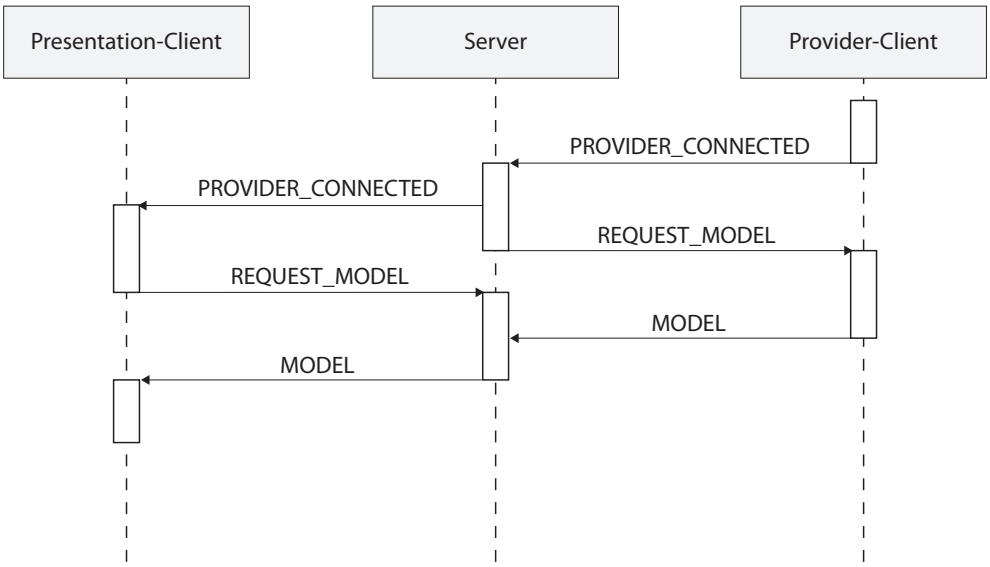


Abbildung 5.3.: Nachrichtenaustausch zwischen Clients und Server: Verbindung eines dem Server nicht bekannten Provider-Clients.

## 5.4. Fazit

Grundlage für eine saubere Integration von *aDDaPt*-Modellen ist eine Architektur, die zwischen Daten, Darstellung und Bedienlogik, trennt. Für diese Arbeit wurde daher eine Architektur auf dieser Grundlage, im Sinne des Model-View-Controller Patterns, entwickelt. Da sich externe Anwendungen in eine bestehende Infrastruktur integrieren sollen, wurde eine Client-Server-Architektur gewählt, die es ermöglicht, dass sich Anwendungs-Clients mit dem jeweils für die Integration benötigten *aDDaPt*-Modell an einem Server anmelden. Der Server dient als zentraler Controller und verwaltet die Kommunikation zwischen den Anwendungs-Clients, Presentation-Clients zur Darstellung der Anwendungsdaten sowie Input-Clients zur Bedienung durch den Nutzer.

Das Nachrichtenformat *TKMessage*, mit dem Server und Clients miteinander kommunizieren, wurde auf Basis von XML umgesetzt. Es ist eine Definition von Nachrichtenteilen auf der Basis von Standards und ist problemlos erweiterbar und austauschbar. Ebenso ist die technische Schnittstelle, über die sich Clients und Server verbinden, ganz im Sinne maximaler Flexibilität nicht festgelegt.

Die Client-Server-Architektur gibt keine Reihenfolge vor, mit der sich Clients zum Server verbinden, und es sind keine Änderungen oder Anpassungen an bestehenden Clients

oder Teilen im Server nötig, wenn neue, bisher unbekannte Clients hinzukommen. Dies schafft die Grundlage für eine flexible Konfiguration bestehender Arten von Clients sowie die Erweiterbarkeit um neue Clients.

Die entwickelte Referenzarchitektur ist nicht auf ein konkretes Einsatzszenario beschränkt und flexibel einsetzbar. Sie stellt damit eine wichtige Grundlage für die Umsetzung und Evaluierung der Integration von *aDDaPt*-Modellen in bestehende Benutzerschnittstellen dar, und wurde für die Umsetzungen der Beispiele verwendet, die im folgenden Kapitel beschrieben werden.

# 6

# Umsetzung und Bewertung am Beispiel von Benutzerschnittstellen im Automobil

Das Konzept abstrakter Interaktionsmodellierung sowie die Architektur zur Integration derartig modellierter Anwendungen in bestehende Benutzerschnittstellen werden in diesem Kapitel am Beispiel des Interaktionsraums Automobil demonstriert und evaluiert. Für die Evaluierung von *aDDaPt* wurden mehrere prototypische Umsetzungen zu verschiedenen charakteristischen Elementen von *aDDaPt* entwickelt. Im Folgenden werden diese Umsetzungen beschrieben und bewertet.

## 6.1. Architektur für die Integration externer Anwendungen

### 6.1.1. Überblick

Basierend auf dem in Kapitel 5 erarbeiteten Konzept wurde eine Client-Server-Architektur umgesetzt, die als Grundlage für die Demonstration unterschiedlicher Benutzerschnittstellen und deren Integration in bestehende Benutzerschnittstellen genutzt werden

konnte. Für die Umsetzung der Architektur waren die Anforderungen an Unabhängigkeit, Erweiterbarkeit und Austauschbarkeit der technischen Schnittstelle ausschlaggebend.

#### Server

Der Server dient als zentrale Steuerungskomponente. Die Umsetzung erfolgte auf einem Windows-Rechner in C#. Die Server-Addins wurden als dynamische Programmbibliotheken, DLL-Dateien (DLL, engl. für *Dynamic Link Libraries*), hinterlegt. Das erarbeitete Nachrichtenformat *TKMessage* wurde in XML verwendet. Die *aDDaPt*-Modelle wurden in XML an den Server geschickt, per XML-Transformation verarbeitet und zunächst an einen beispielhaften Presentation-Client geschickt, der als *Microsoft Silverlight* Anwendung umgesetzt wurde.

#### Clients

Zum Testen der umgesetzten Architektur wurde zunächst die in der Arbeit bereits vorgestellte Anwendung der ToDo-Liste umgesetzt, die als Provider-Client auf einem externen iOS-Gerät, also einem iPad oder iPhone, läuft. Außerdem wurde als Input-Client eine weitere iOS-Anwendung umgesetzt, die lediglich den Touchscreen des Gerätes benutzt und daher als Eingabegerätsimulation zu verstehen ist. Zusätzlich wurde als weitere Eingabemöglichkeit ein Dreh- und Drücksteller, wie in vielen Fahrzeugen vorhanden, angebunden, um die Flexibilität der Architektur zu demonstrieren. Wie im vorigen Abschnitt erwähnt, diente eine Silverlight-Anwendung als Presentation-Client.

Abbildung 6.1 illustriert die Anbindung der Geräte und umgesetzten Anwendungen in der Architektur. Die Kommunikation zwischen Clients und Server-Addins wurde über Socket-Verbindungen beispielhaft realisiert.

#### ClientOverSocket-Addins

Der Nachrichtenaustausch zwischen Clients und Server wurde auf Basis einer abstrakten Klasse `ClientOverSockets` demonstriert, welche die Funktionalität der Kommunikation zwischen Clients und Server-Addins als auch zwischen Server-Addins und Server über eine Socketverbindung gewährleistet.

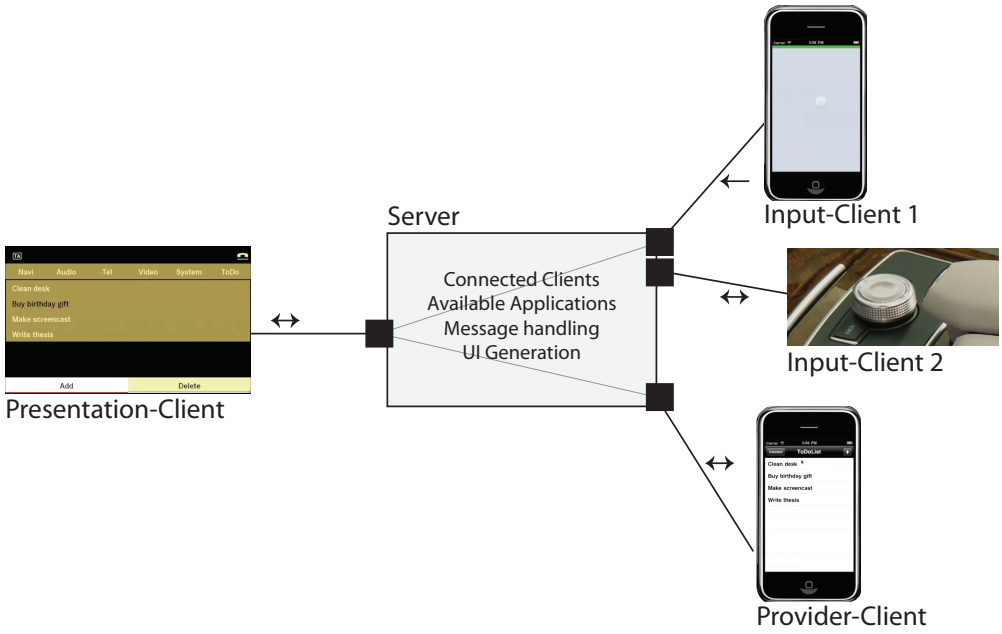


Abbildung 6.1.: Demonstration der Client-Server-Architektur mit Beispielumsetzung: Silverlight-Anwendung als *Presentation*, Touchapplikation oder zentrales Bedienelement im Automobil als *Input*-Clients, iOS-Applikation als *Provider*.

**Addin für Input-Clients** Ein Addin für Input-Clients dient der Kommunikation zwischen einem Eingabegerät und dem Server. Für einfache Eingabegeräte kann die bestehende Funktionalität der ClientOverSocket-Klasse, Bereitstellen der Socketverbindung und Senden und Empfangen von Nachrichten, übernommen werden. Um einen ungewollten Verbindungsabbruch korrekt zu kommunizieren, wird die Methode `ClientDisconnected` überschrieben, um eine INPUT_DISCONNECTED Nachricht zu verschicken.

**Addin für Provider-Clients** Ein Addin für Provider-Clients dient der Kommunikation zwischen einer externen, einzubindenden Anwendung und dem Server. Das Addin registriert sich für die Nachrichten, die von der externen Anwendung benötigt werden, wie in Kapitel 5 definiert. Die Nachrichten aus Server-Events werden an alle verbundenen Clients weitergeleitet. Auch hier wird die `ClientDisconnected`-Methode überschrieben, um PROVIDER_DISCONNECTED kommunizieren zu können.

**Addin für Presentation-Clients** Für die Presentation-Clients, also die Benutzerschnittstellen im Zielsystem, existiert ebenfalls ein Server-Addin, das sich für die benötigten Nachrichten des Servers registriert. Nachrichten mit dem Eventcode MODEL werden speziell behandelt, da diese das abstrakte Interaktionsmodell enthalten. Dieses muss der UI-Generierungskompenente des Servers weitergegeben werden. Für einen auf Microsoft Silverlight basierenden UI-Prototyp wird zum Beispiel das *aDDaPt*-Modell, das in einem XML-Format verschickt wird, vom Server mit einem XSL-Transformator in XAML für Silverlight transformiert. Ähnlich wie bei den anderen Server-Addins wird hier die Nachricht PRESENTATION_DISCONNECTED durch Überschreiben der `ClientDisconnected`-Methode ergänzt, um einen Verbindungsabbruch kommunizieren zu können.

### 6.1.2. Ergebnis und Bewertung

Mit der Umsetzung der Architektur wurde eine technische Realisierung der Integration einer externen Anwendung in bestehende Benutzerschnittstellen demonstriert. Das verwendete Nachrichtenformat *TKMessage* ist dabei flexibel erweiter- oder austauschbar, und die technische Schnittstelle der Kommunikation bedient sich vorhandener Standards, einer Socket-Verbindung über Funknetz, sie kann flexibel durch eine andere Umsetzung ersetzt werden.

Die Umsetzung weist die Anwendbarkeit des erarbeiteten Architekturkonzepts nach und dient außerdem als Grundlage für die Evaluierung der Integration verschiedener Anwendungen, mit *aDDaPt* beschrieben, in bestehende Benutzerschnittstellen.

## 6.2. UI-Prototypen im Fahrzeug

### 6.2.1. Überblick

Im Rahmen der Arbeit sind verschiedene UI-Prototypen entstanden, die als Presentation-Clients eingesetzt wurden, um die Anwendbarkeit des *aDDaPt*-Konzepts anhand von Beispielanwendungen überprüfen zu können. Die verwendeten Interaktionselemente sowie Darstellungsregeln wurden systematisch angelegt und Austauschformate entwickelt, die als Zielformat aus einem Generierungsprozess geeignet waren und von

den UI-Prototypen verarbeitbar waren. Um die Unabhängigkeit der verwendeten UI-Technologie zu unterstreichen wurden mit Adobe Flex/Flash und Microsoft Silverlight zwei verschiedene Technologien verwendet. Zusätzlich wurden die *aDDaPt*-Modelle erfolgreich in verschiedene Interaktionskonzepte, wie ein Touchscreen-Konzept oder ein Anzeigekonzept mit separatem Eingabegerät, integriert. Abbildung 6.2 zeigt Screenshots einiger UI-Prototypen, die begleitend zur Erarbeitung des *aDDaPt*-Konzepts entstanden sind.

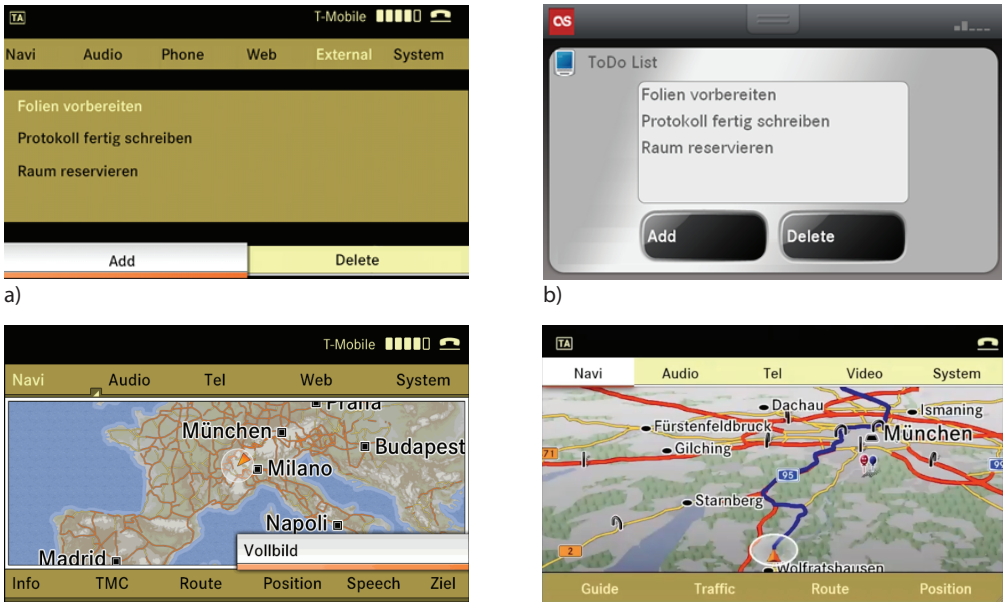


Abbildung 6.2.: Beispiele von UI-Prototypen, die begleitend zur Arbeit entstanden sind: a)+b) Adobe Flash Umsetzungen von UIs auf Basis des gleichen Anwendungsmodells; c) UI-Umsetzung in Adobe Flash; d) UI-Umsetzung in Microsoft Silverlight.

Die eigentliche Erstellung von Benutzerschnittstellen oder die systematische technische Entwicklung der Interaktionselemente ist kein Bestandteil des Konzepts abstrakter Interaktionsmodelle in dieser Arbeit. Voraussetzung für die erfolgreiche Anwendbarkeit von *aDDaPt* ist lediglich, dass Zielsysteme, insbesondere entsprechende Presentation-Clients, ein Austauschformat akzeptieren, um erweiterbar zu sein. Dieses Austauschformat soll weder vereinheitlicht werden müssen, noch soll eine zu integrierende Anwendung dieses Format des Zielsystems kennen müssen. Daher war die Entwicklung verschiedener UI-Prototypen mit verschiedenen Schwerpunkten und in verschiedenen Technologien ein wichtiger Beitrag zur Absicherung des *aDDaPt*-Konzeptes.

### 6.2.2. Ergebnis und Bewertung

Die Umsetzung verschiedener Benutzerschnittstellen mit Möglichkeit der Generierung neuer UI-Teile auf Basis von *aDDaPt* demonstriert die Unabhängigkeit der *aDDaPt*-Modellierung. Eine Generierung kann in beliebigen, entsprechend vorbereiteten, Zielsystemen erfolgen, unabhängig von der verwendeten UI-Technologie.

In den folgenden Abschnitten werden die Beiträge beschrieben, die mit Hilfe der entstandenen UI-Prototypen erfolgreich umgesetzt werden konnten und zur weiteren Evaluierung des *aDDaPt*-Konzeptes dienten.

## 6.3. Integration von UI-Modellen verschiedener mobiler Plattformen

### 6.3.1. Überblick

Als Quelle für externe Anwendungen kommen zahlreiche verschiedene mobile Geräte in Frage, die jeweils auch unterschiedliche Interaktionskonzepte verwenden. Weit verbreitete Smartphone-Betriebssysteme sind Android, Windows Phone, sowie iOS. Jedes Betriebssystem bringt dabei eigene Interaktionskonzepte ein, die auch die für die Plattform entwickelten Apps beeinflussen. Windows verwendet die Metapher eines veränderlichen Blickfensters auf einen Ausschnitt einer größeren Szene, iOS baut auf vertikale Auswahllisten und horizontalen Transitionen zwischen diesen, ebenso wie Android, das aber zusätzlich mehr Hardware-Buttons für die Navigation verwendet. Wie sich diese Unterschiede im Umgang mit *aDDaPt*-Modellen verhalten, wurde daher im Rahmen der Arbeit analysiert und anhand eines Demonstrationsszenarios illustriert und umgesetzt. Basis dafür war die bereits in Kapitel 4 vorgestellte Beispielanwendung *JoinMyRide*, eine Anwendung zur Planung von Mitfahrgelegenheiten. Als Presentation-Client wurde ein UI-Prototyp, umgesetzt auf Basis von Microsoft Silverlight, wie im vorigen Abschnitt erwähnt, verwendet und angepasst. Die Integration basiert außerdem auf der in Abschnitt 6.1 beschriebenen Umsetzung der Architektur.

### *aDDaPt*-Modell für *JoinMyRide*

Das *aDDaPt*-Modell für die Beispielanwendung *JoinMyRide*, das in Kapitel 4.7 vorgestellt wurde, wurde in UML modelliert und gilt für alle Implementierungen auf den drei Plattformen Windows, iOS und Android. Damit konnte demonstriert werden, dass *aDDaPt* plattformunabhängig verwendet werden kann.

### Einbindung plattformspezifischer Interaktionen in *aDDaPt*

Um die Flexibilität von *aDDaPt* zu zeigen, wurde zusätzlich die Möglichkeit untersucht, plattformspezifische Interaktionen der mobilen Geräte so weit wie möglich zu modellieren, um dem Nutzer bekanntes Verhalten bestimmter Interaktionselemente bei der Integration in ein Zielsystem zu erhalten. Dazu wurde ein Mehr-Ebenen-Abstraktionskonzept auf Basis von *aDDaPt* entwickelt und umgesetzt. Die im Zielsystem generierte Benutzerschnittstelle stellt ein modelliertes Interaktionskonzept nur dar, wenn das Zielsystem dies semantisch kann. Ansonsten erzeugt der Generator auf Basis der nächsthöheren Abstraktionsstufe ein Benutzerschnittstellenelement aus seinem vorhandenen Repertoire, das zur abstrakten Interaktion passt. Abbildung 6.3 illustriert dieses Konzept der Abstraktion auf mehreren Ebenen.

**Beispiel: Zurück-Funktion** Auf dem Windows Phone führt der Zurück-Button einen Schritt zurück in der erfolgten Navigation. In Android führt der Zurück-Button in den meisten Fällen zum nächsten Schritt, der vom Nutzer zu erledigen ist, also semantisch gesehen vorwärts, zum Beispiel um eingegebene Änderungen zu speichern und weiter mit dem nächsten Schritt zu machen. In der Umsetzung wurden also die beiden unterschiedlichen Zurück-Buttons in der Modellierung unterschieden. In iOS gibt es ein *Undo*-Verhalten, das beim Schütteln des mobilen Gerätes ausgelöst wird und wie ein Zurück funktioniert, allerdings nur für die letzte Aktion.

**Beispiel: Kontextmenüs** Unterschiedliche mobile Plattformen verwenden unterschiedliche Interaktionskonzepte für Kontextmenüs. In Windows gibt es beispielsweise das auf einen Eintrag bezogene Kontextmenü, das über ein längeres Drücken auf den Eintrag geöffnet wird. In iOS gibt es das gestenbasierte Kontextmenü, das über ein Wischen

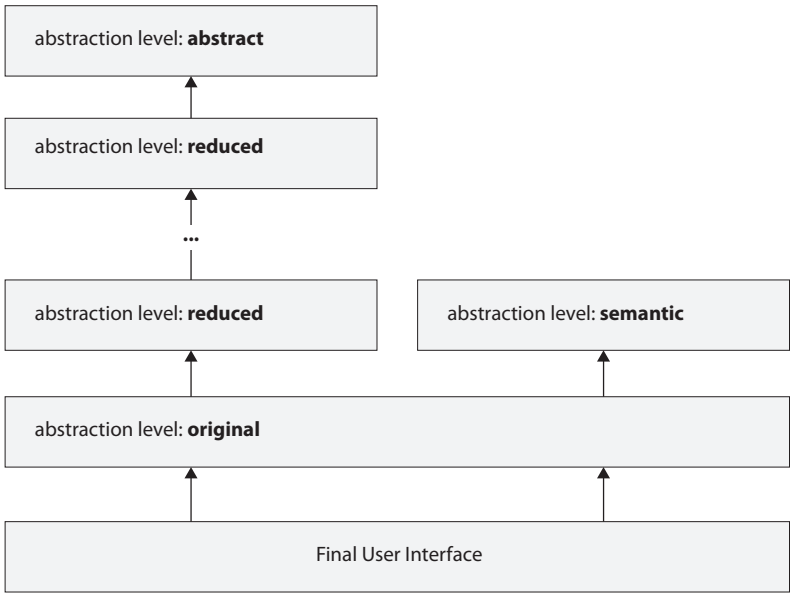


Abbildung 6.3.: Konzept der Mehr-Ebenen-Hierarchie mit unterschiedlichen Abstraktionsstufen. Die Generierung erfolgt auf der Ebene, die am nächsten zum *Final User Interface* ist, sofern eine anwendbare Regel gefunden wird.

nach links auf dem Eintrag geöffnet wird. Semantisch gesehen sind beide Interaktionsformen gleich. Die Form der Integration hängt also von den Fähigkeiten des Zielsystems ab, ein eintragsbezogenes Kontextmenü anzuzeigen.

**Beispiel: Unterschiedliche Listen** iOS verwendet *Cover Flow*, um Elemente, die aus einem Bild und einem Text bestehen, anzuzeigen. Semantisch ist dies eine Liste mit Einträgen aus Bild und Textbeschriftung, die zum Beispiel im Windows Phone als vertikale Liste dargestellt wird. Je nach Fähigkeiten der Zielplattform kann sich die Darstellung entsprechend anpassen.

Für die detaillierte Analyse und Ausarbeitung der *aDDaPt*-Erweiterung um mehr semantische Ebenen sei auf die im Rahmen dieser Arbeit entstandene Ausarbeitung von Manuel Deil verwiesen [Dei11].

### 6.3.2. Ergebnis und Bewertung

Die Modellierung und Einbindung von Anwendungen, die auf verschiedenen mobilen Plattformen entwickelt wurden, in ein Zielsystem auf Basis der *aDDaPt*-Modellierung demonstriert, dass die Anwendbarkeit des *aDDaPt*-Konzeptes auch unabhängig von der verwendeten Technologie der Provider-Clients ist.

Die Analyse der spezifischen Interaktionsmöglichkeiten verschiedener mobiler Plattformen und die darauf aufbauende prototypisch umgesetzte Erweiterung des Konzeptes um eine Mehr-Ebenen-Hierarchie demonstriert außerdem die Flexibilität von *aDDaPt* sowie die Ermöglichung von Kontextsensitivität.

## 6.4. Generische Transformationen

### 6.4.1. Überblick

Als weitere Beispielanwendung wurde ein Branchenverzeichnis, vergleichbar mit den *Gelben Seiten*, für die Demonstration des *aDDaPt*-Ansatzes umgesetzt und in verschiedene Zielsysteme integriert. Als systematische Erweiterungsmöglichkeit wurde dabei zusätzlich der Aspekt generischer Transformationen untersucht, also die Erzeugung von zwei Benutzerschnittstellen für verschiedene Zielkonzepte mit nur einem Generator [M̈2]. Zielkonzept 1 war die Umgebung mit einem externen Dreh- und Drücksteller zur Eingabe, Zielkonzept 2 war eine Touchscreenapplikation mit direkter Manipulation der Elemente. *aDDaPt* wurde um semantische Transitionen erweitert, die es einem generischen Transformationsprozess ermöglichen, Benutzerschnittstellen für mehrere Zielkonzepte zu generieren.

#### Generierung für unterschiedliche Zielsysteme

In der Umsetzung wurde ein Zielsystem mit abgesetztem Dreh- und Drücksteller zur Eingabe sowie ein Zielsystem mit Touchscreen verwendet. Diese beiden Systeme unterscheiden sich bezüglich der Interaktion vor allem in folgenden Punkten:

**Fokus und Aktivierung von Elementen** In einem abgesetzten Konzept muss ein Menüeintrag oder Button explizit fokussiert werden, bevor er ausgewählt werden kann.

Mit einem Touchscreen kann ein Button direkt manipuliert werden ohne vorher aktiv ausgewählt zu werden.

**Gültige Aktionen zu einem Zeitpunkt** In einem abgesetzten Konzept kann nur mit dem aktuell fokussierten Element interagiert werden. Mit einem Touchscreen sind alle sichtbaren Elemente theoretisch verfügbar zur direkten Interaktion.

**Variabilität der Eingabe** In einem abgesetzten Konzept erfolgen einzelne Eingabesignale diskret über das Eingabegerät. Mit einem Touchscreen können mehr Faktoren des Eingabesignals wie Dauer des Drucks, Position und Stärke einbezogen werden.

**Verbindung von Eingaben** In einem abgesetzten Konzept sind Eingaben klar getrennt und nacheinandergeschaltet. Mit einem Touchscreen können durch eine Interaktion mehrere Eingaben quasi zeitgleich erfolgen, beispielsweise um ein Element in eine andere Liste zu ziehen.

Das *aDDaPt*-Modell für die Gelbe-Seiten-Anwendung dient in beiden Zielsystemen als Input für den Generierungsprozess. Neben der damit erfolgten Demonstration der Anwendbarkeit von *aDDaPt* in einer weiteren Umsetzung wurde zusätzlich untersucht, wie ein Generierungsprozess vereinheitlicht werden kann. Dafür wurden zusätzliche semantische Informationen auf dem Level der abstrakten Modellierung eingeführt, wie beispielsweise `MODIFY`, das die Veränderung eines Datenobjekts beschreibt. Durch Hinzunahme von Wissensbasen über Interaktionselemente der Zielsysteme wird die schrittweise Transformation des abstrakten Modells zum entsprechenden finalen UI unter Verwendung eines generischen Transformators erreicht. Die erweiterten Transitionen und dazugehörige Modellierungsregeln wurden als Ergänzungsmöglichkeit definiert und finden sich in Anhang B, der auch das entsprechend angepasste abstrakte Modell der Beispielanwendung *Gelbe Seiten* enthält. Screenshots der generierten Benutzerschnittstellen für die beiden Zielsysteme sind in Abbildung 6.4 dargestellt.

### 6.4.2. Ergebnis und Bewertung

Mit Hilfe der *aDDaPt*-Modellierung des Gelbe-Seiten-Beispiels wurde anhand einer weiteren Anwendung, die von einem weiteren Entwickler erstellt wurde, eine erfolgreiche prototypische Integration in zwei unterschiedliche Zielsysteme gezeigt. Damit wurde nicht nur die Anwendbarkeit von *aDDaPt* erneut bestätigt. Auch die Gebrauchstauglichkeit

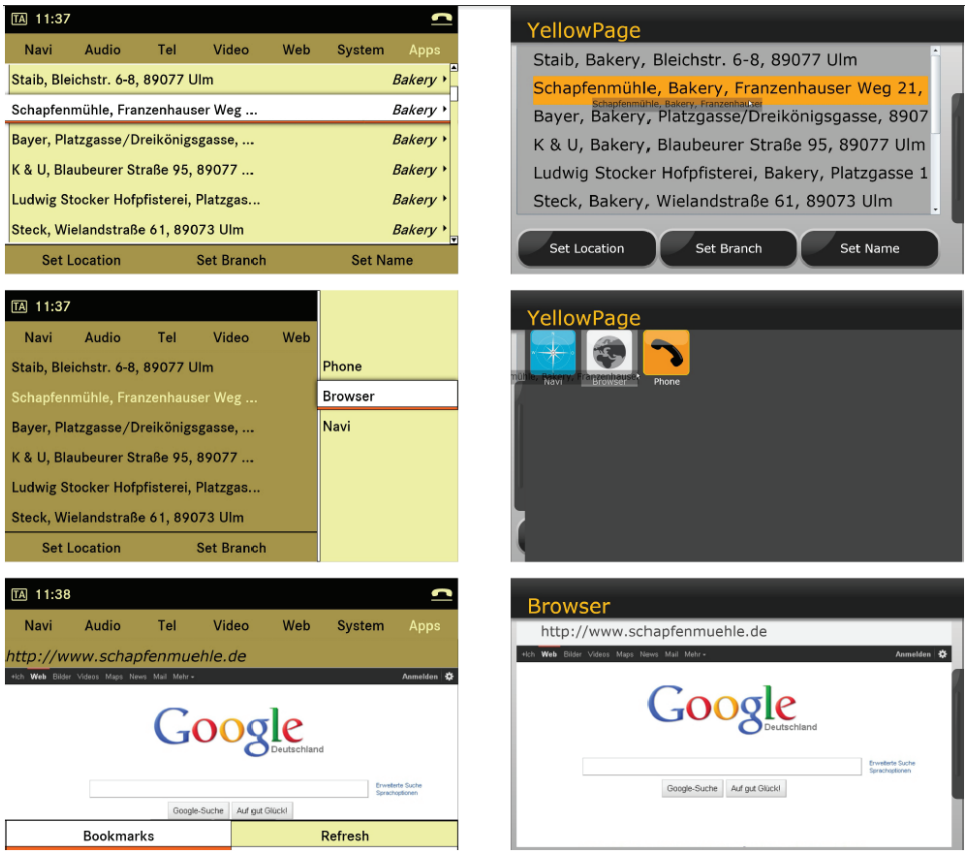


Abbildung 6.4.: Screenshots der generierten Benutzerschnittstellen für die Beispielumsetzung der *Gelbe Seiten*-Anwendung (aus [MÏ2]). Links: Ein Kontextmenü erlaubt den Aufruf verfügbarer Zusatzdienste, wie einen Browser. Rechts: Im Touchscreen-Konzept kann ein ausgewählter Datensatz per *Drag and Drop* mit einem entsprechenden Zusatzdienst direkt geöffnet werden.

für Entwickler spiegelt sich darin wider, da sich ein weiterer Entwickler schnell und erfolgreich in *aDDaPt* und den Integrationsprozess einarbeiten konnte.

Mit der Erweiterung um semantische Transitionen wird außerdem das Potenzial der Optimierung der Transformationen in verschiedene Zielsysteme demonstriert, womit Flexibilität und Erweiterbarkeit von *aDDaPt* unterstrichen werden.

## 6.5. Fazit

*aDDaPt* als Konzept für die abstrakte Interaktionsmodellierung, um Anwendungen in bestehende Benutzerschnittstellen zu integrieren, wie in Kapitel 4 vorgestellt, wurde in diesem Kapitel anhand mehrerer prototypischer Umsetzungen am Beispiel des Interaktionsraums Automobil evaluiert. Die in Kapitel 5 vorgestellte Referenzarchitektur wurde umgesetzt und diente als Basis zur Entwicklung der verschiedenen Beispielumsetzungen.

Die Beispielumsetzungen demonstrieren die Anwendbarkeit von *aDDaPt* mit verschiedenen UI-Technologien für alle Arten von Clients innerhalb der konzipierten Architektur. Die Erweiterung um verschiedene Aspekte in der Modellierungsmethode als auch in der Umsetzung der Transformationen verdeutlicht außerdem die Erfüllung der Anforderungen, insbesondere Unabhängigkeit und Anpassbarkeit der Modellierung sowie Eindeutigkeit, Kontextsensitivität und Automatisierbarkeit der Integration.

In den *aDDaPt*-Ansatz sowie das Demonstrator-System haben sich außerdem immer wieder verschiedene Entwickler, die an den Analysen und Erweiterungen weiterer Aspekte beteiligt waren, in kurzen Zeiträumen von wenigen Monaten komplett eingearbeitet, was die Gebrauchstauglichkeit des Konzepts unterstreicht. Letztlich wurde durch die begleitenden Umsetzungen der komplette Prozess der Entstehung, Anpassung und Evaluierung des Konzeptes von *aDDaPt* maßgeblich unterstützt. Die Ergebnisse der Arbeit werden im folgenden Kapitel zusammengefasst.

# 7

# Ergebnisse

Mit *aDDaPt* wurde eine Modellierungsmethode zur abstrakten Beschreibung von Benutzerschnittstellen entwickelt. *aDDaPt*-Modelle können automatisiert weiterverarbeitet werden und zur Laufzeit in ein Zielsystem integriert werden. Die Entwicklungsprozesse für das Zielsystem sowie für die zu integrierende Anwendung sind dabei unabhängig voneinander, wodurch die Flexibilität bei der Erstellung von Anwendungen, die in verschiedene Zielsysteme integriert werden sollen, erhalten wird. Dadurch können außerdem domänenspezifische Anforderungen im Zielkontext erfüllt werden, was in dieser Arbeit am Beispiel von Benutzerschnittstellen im Automobil demonstriert wurde. Die Ergebnisse werden im Folgenden zusammengefasst.

## 7.1. *aDDaPt* für die flexible abstrakte Beschreibung von Benutzerschnittstellen

In dieser Arbeit wurde der Ansatz der abstrakten Domänen-, Dialog- und Präsentationsmodellierung – *aDDaPt* – entwickelt. Das Aufgabenmodell, was in vielen Arbeiten als

Kern der Interaktionsmodellierung angesehen wird, fließt dabei nicht als eigenständiges Modell ein. Aufgaben des Systems werden im Domänenmodell abgebildet. Aufgaben des Nutzers werden im Dialogmodell und Präsentationsmodell abgebildet. Eine Verknüpfung von System- und Nutzeraufgaben findet außerdem im Präsentationsmodell statt, indem die Manipulation oder Befüllung von Attributen der Domäne in bestimmten Präsentationszuständen modelliert wird. Durch die enge Integration des Aufgabenmodells in Domänen-, Dialog- und Präsentationsmodell wird das Mapping-Problem, was bei der Abbildung abstrakter Aufgaben in konkrete Interaktionsmodelle entsteht, vermieden.

Ein weiterer Grund für diese Art der Integration des Aufgabenmodells ergibt sich aus der Zielsetzung in dieser Arbeit, dass die Interaktionsmodellierung separat von der Anwendungsentwicklung stattfinden können soll. Soll ein *aDDaPt*-Modell für eine bereits existierende Anwendung erstellt werden, so sind die Aufgaben bereits in der Anwendung enthalten. Der *aDDaPt*-Modellierer verwendet die Aufgaben, aber es ist nicht erforderlich, dass sie aus einem Modell in weitere Modelle transformiert werden. Dies ermöglicht zusätzlich, dass beispielsweise nur ein Teil der Anwendung für das *aDDaPt*-Modell verwendet wird, wenn Teile der Anwendung im bestehenden Kontext nicht zur Abstraktion und Verwendung in anderen Umgebungen geeignet sind. Der Zusammenhang zwischen den Modellarten bei der *aDDaPt*-Modellierung ist in Abbildung 7.1 illustriert.

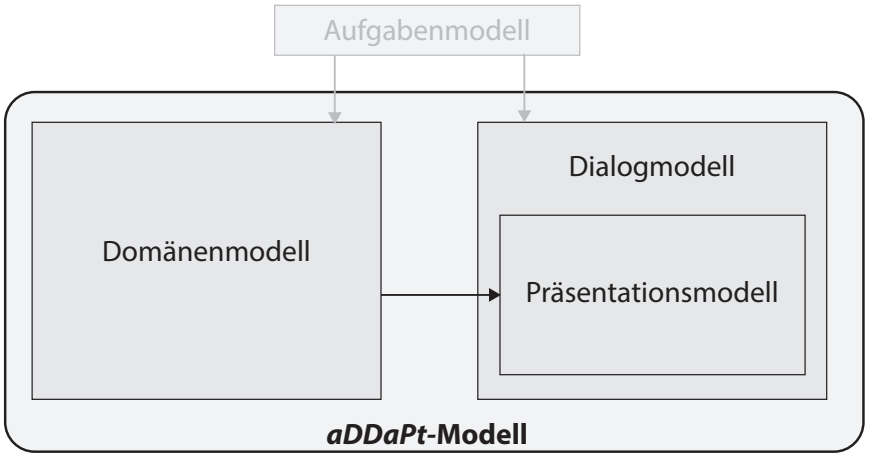


Abbildung 7.1.: *aDDaPt* kombiniert Elemente verschiedener Modellarten in ein abstraktes Interaktionsmodell, bestehend aus Domänen-, Dialog- und Präsentationsmodell.

Für den Nachweis der Anwendbarkeit von *aDDaPt* wurde in dieser Arbeit ein Demonstratorsystem mit mehreren Umsetzungen von Beispielanwendungen entwickelt. *aDDaPt* wurde erfolgreich verwendet, um Modelle für die Beispielanwendungen zu erstellen und mithilfe von Transformationsregeln in bestehende Systeme zu integrieren. Dabei wurde auch sichergestellt, dass *aDDaPt* nicht an eine zu Grunde liegende Architektur oder Technologie gebunden ist, um das resultierende Austauschformat verwenden zu können, wie etwa bei durch Webtechnologien getriebenen Ansätzen. Mit einer einfachen grafischen Repräsentation des *aDDaPt*-Formats in Form von Klassen- und Zustandsdiagrammen ist außerdem sichergestellt, dass sich der Ersteller der Modelle nicht auf die Ebene einer konkreten Technologie begeben muss. Die mit *aDDaPt* modellierten Interaktionsanwendungen wurden mit Hilfe der entwickelten Referenzarchitektur voll funktional in einen bestehenden interaktiven Prototyp einer Infotainmentumgebung im Fahrzeug integriert.

## 7.2. Automatische Weiterverarbeitung und Integration von *aDDaPt*-Modellen im Zielkonzept

Um ein *aDDaPt*-Modell automatisch weiterzuverarbeiten und daraus eine Benutzerschnittstelle für das Zielsystem zu erzeugen, wird eine Regelhierarchie erstellt, die in den Transformationsprozess einfließt. Mindestanforderung an die Regeln ist, dass im Zielsystem ein Basislayout vorliegt und für jedes Präsentationsobjekt sowie jede Transitionsart im abstrakten Modell jeweils ein Element im Zielsystem vorhanden ist, auf das abgebildet werden kann. Darauf aufbauend wurde in der Arbeit das Konzept der Regelhierarchie erarbeitet und unter Verwendung der Referenzarchitektur für die modellierten Beispielanwendungen erfolgreich umgesetzt. Der Transformationsprozess findet im Server statt, der auch die Kommunikation zwischen den Clients im System übernimmt.

Die Anwendbarkeit des Grundkonzepts der Transformationen wurde durch die Umsetzung erfolgreich nachgewiesen. Die Festlegung von Basislayout und Basisabbildungen ist die einzige unbedingte Anforderung an die Erstellung der Regelhierarchie. Dadurch ist die Integration im Zielsystem nicht nur kontrollierbar und vorhersehbar, sondern insbesondere auch flexibel erweiterbar. Diese Erweiterbarkeit wurde aufbauend auf dem Grundkonzept der Regelhierarchie untersucht. Um eine bessere Abbildung konkreter

Elemente einer Ausgangsplattform in das Zielsystem zu erreichen, wurden semantische Abstraktionsstufen in der Regelhierarchie hinzugefügt. Auf der abstraktesten Stufe wurden außerdem Erweiterungen des Modellierungssatzes um semantische Transitionen untersucht und umgesetzt. Dadurch lässt sich die Integration in unterschiedliche Zielsysteme mit nur einem generischen Transformationsprozess erreichen. Diese Erweiterungsmöglichkeiten unterstreichen deutlich die Flexibilität in der Anwendung von *aDDaPt*.

## 7.3. Flexible unabhängige Entwicklungsprozesse

Die Auswirkungen auf mögliche Entwicklungsprozesse wurden bei der Erarbeitung des *aDDaPt*-Konzeptes untersucht. Auf der Seite des Anbieters einer externen Anwendung ist es wichtig, dass die Entwicklung dieser möglichst wenig beeinflusst wird. Dies wird dadurch erreicht, dass die Entwicklung losgelöst von der Bereitstellung des abstrakten Interaktionsmodells ist. Entwickler externer Anwendungen müssen sich von Seiten der Benutzerschnittstelle nicht an einen Modellierungsprozess halten. Das *aDDaPt*-Modell kann separat erstellt werden und muss auch nicht durch den ursprünglichen Entwickler der Anwendung geschehen. Eine technische Schnittstelle für den Datenaustausch und Aufruf von Anwendungsfunktionen muss allerdings vorhanden sein. Dafür wurde ein Nachrichtenaustauschformat und eine Client-Server-Architektur entwickelt, die prototypisch umgesetzt und als Demonstrationsplattform von mehreren Beispielumsetzungen verwendet wurde. Das Nachrichtenformat ist flexibel einsetzbar und erweiterbar und die Architektur ist nicht an eine bestimmte technische Umsetzung der Verbindung von Clients geknüpft.

Auf der Seite des Systems, in welchem die Integration stattfinden soll, sind Vorbereitungsarbeiten für den Integrationsprozess nötig. Die Kontrolle über eine optimale Integration liegt beim Zielsystem, daher muss dieses entsprechend vorbereitet werden. Der Transformationsprozess ist fest in das Zielsystem integriert und kann nicht separat ausgelagert werden. Dies bedeutet, der Transformationsregelsatz muss während der Entwicklung des Systems erstellt werden, ebenso wie die technische Integration in das Benutzerschnittstellen-Framework. Die Benutzerschnittstellen-Technologie wird durch *aDDaPt* nicht vorgeschrieben oder beeinflusst, allerdings müssen die Transformationsregeln auf die verwendete Technologie angepasst sein. In den prototypischen Umsetzungen im Rahmen dieser Arbeit wurde die erfolgreiche Integration von Anwendun-

gen, die mit *aDDaPt* modelliert wurden, für mehrere Benutzerschnittstellen-Technologien gezeigt.

## 7.4. Erfüllung domänenspezifischer Anforderungen im Zielkontext

Die Anwendungsbeispiele, die im Rahmen dieser Arbeit umgesetzt wurden, wurden für den Automobilbereich zugeschnitten, um die Erfüllung domänenspezifischer Anforderungen in diesem Zielkontext zu untersuchen.

Benutzerschnittstellen im Automobil sind durch rechtliche Anforderungen an minimale Fahrerablenkung eingeschränkt. Entsprechend war es wichtig, sicherzustellen, dass neue zu integrierende Anwendungen dem Bedien- und Anzeigekonzept im Zielsystem folgen. Durch die klare Kontrolle über die Integration durch das Zielsystem, insbesondere was die Umsetzung der Integrationsregeln und der Kommunikationsarchitektur betrifft, konnte das für die prototypische Umsetzung der Anwendungsbeispiele erfolgreich demonstriert werden.

Eine weitere Rahmenbedingung im Automobilbereich ist die lange Lebenszeit des Automobils im Vergleich zu viel kürzeren Lebenszeiten von Consumer Electronics wie etwa Mobiltelefonen, Tablets oder persönlichen Computern. Durch die klare Trennung der technologischen Umsetzung und des abstrakten Interaktionsmodells sowie die Möglichkeit, mehrere Abstraktionsstufen zu bedienen, ist eine Abwärtskompatibilität gegeben, sodass die Integration von neuen Anwendungen auch über die lange Lebenszeit eines Automobils möglich ist.

Auch die etablierten Entwicklungsprozesse im Automobilbereich wurden in die Konzeptentwicklung und Umsetzung dieser Arbeit einbezogen. Viele Automobilhersteller entwickeln Infotainmentsysteme mit einem klassischen Zulieferermodell. Teilweise verhalten sich Module dabei wie eine Black Box und lediglich die Schnittstellen zwischen Modulen sind definiert. Dieses Modell kann mit *aDDaPt* leicht bedient werden, da die Entwicklungsprozesse einer zu integrierenden Anwendung sowie des Zielsystems im Fahrzeug unabhängig voneinander stattfinden können. Schnittstellen zur Integration und Kommunikation der relevanten Module wurden definiert und die Architektur prototypisch umgesetzt.

All diese Anforderungen, die man aktuell im Automobilbereich findet, wurden bei der Entwicklung von *aDDaPt* adressiert. In den letzten Jahren zeichnet sich mehr und mehr ab, dass auch Infotainmentsysteme im Automobilbereich offener und flexibler werden. Es ist daher denkbar, dass die Sicherstellung der Anforderungen für die Anwendbarkeit von *aDDaPt* in zukünftigen Systemen eher einfacher wird.

## 7.5. Fazit

In dieser Arbeit wurde eine Modellierungsmethode für die flexible abstrakte Beschreibung von Benutzerschnittstellen entwickelt, die es ermöglicht, Anwendungen automatisiert in verschiedenen Interaktionskontexten, nahtlos und abgestimmt auf die Zielumgebung, zu integrieren. Abbildung 7.2 illustriert dieses Ergebnis. Das *aDDaPt*-Modell besteht aus der Kombination eines abstrakten Domänen-, Dialog- und Präsentationsmodells.

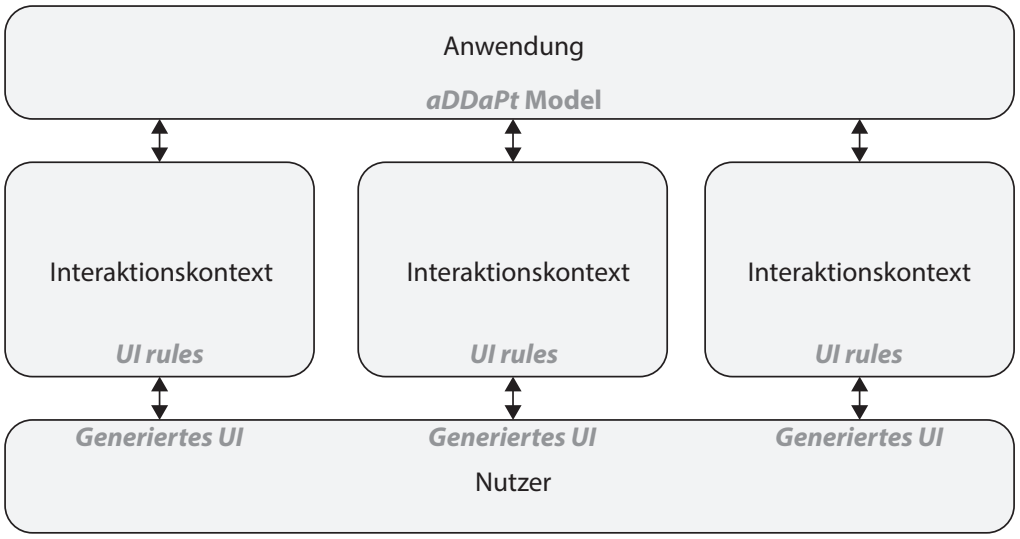


Abbildung 7.2.: Ergebnis: Flexible, abstrakte Modellierungsmethode, mit der die Integration einer Computeranwendung in verschiedenen Interaktionskontexten nahtlos und abgestimmt auf die Zielumgebung ermöglicht wird.

Da die Entwicklungsprozesse der Anwendung sowie des Zielsystems unabhängig voneinander erfolgen und die Kontrolle der Integration im Zielsystem liegt, eignet sich der

Ansatz inbesondere für Zielsysteme, deren Benutzerschnittstellen kritischen Rahmenbedingungen unterliegen, wie Infotainmentsysteme im Fahrzeug.

Die Ergebnisse tragen zur Schließung der Lücke zwischen Anwendung und Interaktionsmöglichkeiten bei, die bei der Integration unbekannter Anwendungen in bestehende interaktive Systeme besteht, und bedienen damit die Vision einer einfachen Nutzung einer Computeranwendung in jedem Kontext, welche in der Problemstellung in Abschnitt 1.2 motiviert wurde. Bevor eine Gesamtzusammenfassung mit Ableitung möglicher zukünftiger Schritte auf Basis der erreichten Ergebnisse die Arbeit abschließt, werden die vorstehend zusammengefassten Ergebnisse im folgenden Kapitel ausführlich diskutiert.

# 8

# Diskussion

In diesem Kapitel werden die Ergebnisse der Arbeit diskutiert, um wichtige Entscheidungen bei der Konzeptentstehung im Zusammenhang mit der Abwägung von Vor- und Nachteilen herauszustellen, sowie die Gebiete zu identifizieren, in denen zukünftige Arbeiten das Themenfeld der abstrakten Interaktionsmodellierung und Benutzerschnittstellengenerierung weiter voranbringen könnten.

## 8.1. Abdeckung möglicher Interaktionen

Ein Kernaspekt zur Diskussion ist das Maß der Abbildungsmöglichkeiten aller Interaktionen aus dem *aDDaPt*-Modell in die finale Benutzerschnittstelle und umgekehrt. Eine Anforderung an *aDDaPt* ist die garantierte Abbildbarkeit, da der Prozess vollautomatisch ablaufen können muss, ohne dass sich externe Anwendung und Zielsystem kennen. Durch die 1:1 Basisabbildungen im Regelsatz wird sichergestellt, dass es immer eine Abbildung gibt. Allerdings wird damit nicht automatisch sichergestellt, dass derart erzeugte Abbildungen optimal für die Bedienung der Anwendung sind, insbesondere wenn

es sich um ein komplexes Interaktionskonzept handelt, was auf der Zielplattform in der Form nicht vorhanden ist. Bei der Verwendung von *aDDaPt* kann mit diesem Problem jedoch flexibel umgegangen werden, wenn sich der Modellierer bewusst darüber ist. Das folgende Beispiel verdeutlicht dies.

**Beispiel** Es soll eine Navigationsanwendung mit Karte von einem externen Gerät in die Benutzerschnittstelle im Automobil integriert werden. Der Nutzer kann damit Sonderziele (POI, engl. für *Point of Interest*) suchen. Verfügt das System im Fahrzeug bereits über eine Kartennavigation, könnte diese als Grundlage der Integration verwendet werden und Daten und Funktionen der externen Anwendung übernehmen. Stellt das Fahrzeug-System allerdings keine eigene Kartennavigation bereit, ist die Integration einer Karte als Benutzerschnittstellenelement eine Herausforderung, die zur Nichtbedienbarkeit der Anwendung führen könnte. Eine prinzipiell mögliche Herangehensweise wäre die Erweiterung der Basistypen des *aDDaPt*-Domänenmodells um einen neuen Typ, zum Beispiel *MapLocation*. Damit würde die Integration der Anwendung in ein System, was diesen Typ nicht bereitstellt, abgelehnt. Im Sinne einer abstrakten Modellierung kann der Modellierer die Anwendung aber auch ohne Erweiterung der Basistypen beschreiben. Aus Nutzersicht gibt es beispielsweise folgende Interaktionen und Darstellungen: Eingabe einer Stadt, eines Suchbegriffs, sowie Auswahl aus Ergebnissen, die einen Namen oder Titel haben, eine Entfernung und Geokoordinaten. Diese Interaktionen und zugehörigen Daten lassen sich abstrakt ohne spezielle Datentypen beschreiben. Abbildung 8.1 illustriert ein solches *aDDaPt*-Modell für die Sonderziel-Suche, das nur die bekannten Basistypen von *aDDaPt* verwendet sowie daraus definierte eigene Typen. Abbildung 8.2 illustriert, wie aus diesem Modell, abhängig von den Fähigkeiten des Zielsystems, eine Benutzerschnittstelle mit oder ohne Verwendung einer Navigationskarte erzeugt werden kann.

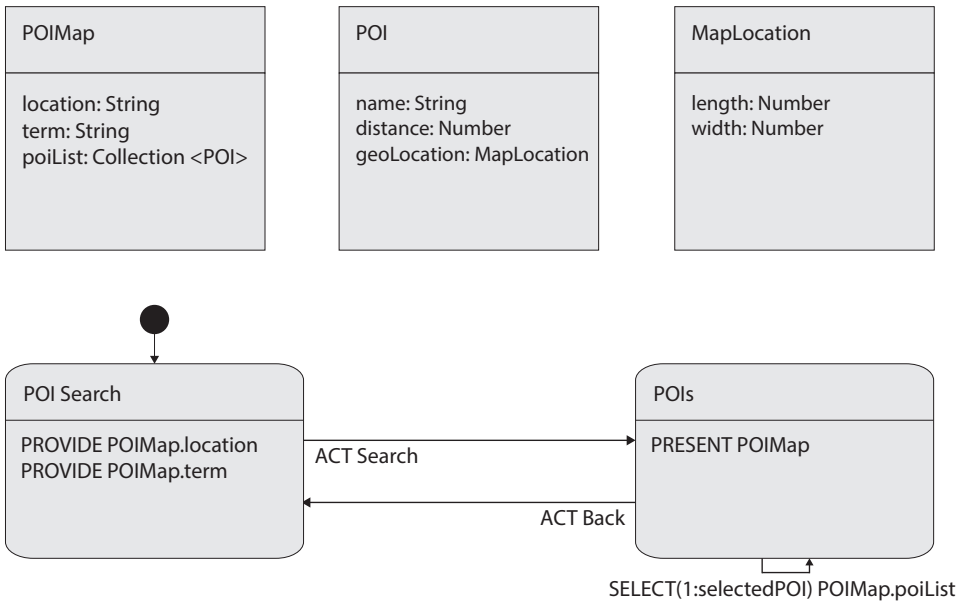


Abbildung 8.1.: Beispiel eines *aDDaPt*-Modells für eine Anwendung zur Sonderzielsuche (*POI Search*).

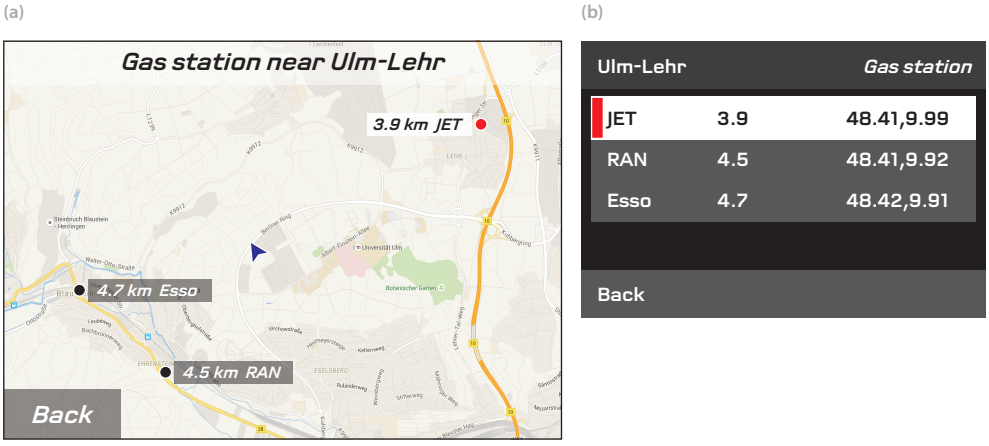


Abbildung 8.2.: Mögliche Darstellung der generierten Benutzerschnittstelle für die Sonderzielsuche in (a) ein System mit Navigationsmodul und (b) ein System ohne Navigationsmodul.

Das Beispiel zeigt, dass der Modellierer einen großen Einfluss auf die Integration und Generierung im Zielsystem hat. Im Rahmen dieser Arbeit wurde auch untersucht, wie dieser Einfluss durch verschiedene Abstraktionsstufen ausgenutzt werden kann, um optimale Abbildungen anhand der jeweiligen Fähigkeiten des Ausgangs- und Zielsystems zu finden [Dei11]. Im obigen Beispiel wäre es diesbezüglich sinnvoll, semantisches Wissen über einen Typ *MapLocation* zu haben, um eindeutig die Abbildung der Daten auf eine Navigationskarte mit Geo-Koordinaten erzielen zu können. Zukünftige Arbeiten müssen diesen Aspekt von *aDDaPt* weiter untersuchen.

## 8.2. Grenzen der Generierung

Ein automatischer Generierungsprozess ist nur so gut wie die Regeln zur Generierung. Und ein altes System bleibt auch bei der Integration neuer Anwendungen alt, denn das bestehende Benutzerschnittstellenkonzept bleibt. Ebenso sind keine Überraschungen bei der Generierung zu erwarten, solange auf Basis eines klaren Regelwerks generiert wird. *aDDaPt* baut auf dieser Grundlage auf mit dem Fokus auf flexibler Erweiterbarkeit eines abgeschlossenen, bestehenden Systems ohne in dieses System eingreifen zu müssen. Die Grenzen der Generierung sind dann erreicht, wenn Kreativität, in Form von neuen Benutzerschnittstellenelementen oder aus bisherigen Regeln fallenden Anordnungen von Elementen, generiert werden soll. Das *aDDaPt*-Konzept wurde allerdings auch nicht mit dem Ziel einer derartigen künstlichen Intelligenz entwickelt, da sich in bisherigen Arbeiten gezeigt hat, dass die Ergebnisse freier, vollautomatischer Benutzerschnittstellengenerierung nicht zufriedenstellend sind, wie unter anderem von Meixner aufgeführt [Mei10].

Neben den Grenzen bei der Generierung von *Kreativität* stößt *aDDaPt* auch bei der Abwärtskompatibilität von modernen Benutzerschnittstellen- und Darstellungstechnologien an gewisse Grenzen. Typische Beispiele moderner Technologien sind die Verwendung dreidimensionaler Inhalte (3D) sowie Animationen. Bei reiner bildlicher Darstellung eines 3D-Elements wird *aDDaPt* prinzipiell nicht beeinflusst, da es logisch gesehen genauso wie ein zweidimensionales Bild angezeigt wird. Schwieriger wird es, wenn an die Interaktion mit dem 3D-Element im Ausgangssystem bestimmte Ausgaben oder Methoden der Anwendung geknüpft sind. Eine Benutzerschnittstellengenerierung, ohne alle 3D-Möglichkeiten der Zielplattform vorab zu kennen, kann zu Effekten führen, die die Intention bei der ursprünglichen Erstellung der 3D-Interaktion nicht wiederspiegeln.

Wenn das Zielsystem beispielweise ein 3D-Modell nur statisch anzeigen kann, so ist die Möglichkeit zur Angabe des Rotationswinkels für den Nutzer wenig brauchbar. Um dies zu vermeiden, muss durch die Modellierung sichergestellt werden, dass das abstrakte Interaktionsmodell auch ohne 3D-Elemente angemessen repräsentiert werden kann. Beim Thema animierter Inhalte ist die Macht des Modellierers, diese abstrakt abzubilden, etwas geringer. Zwar ist es theoretisch denkbar, alle Animationsparameter abstrakt zu beschreiben sodass eine Zielplattform, die ein frei konfigurierbares Animations-Framework bereitstellt, diese darstellen kann, allerdings gibt es hierfür zielgerichtetere Methoden und Werkzeuge. Das *aDDaPt*-Konzept beschränkt sich für diese Arbeit auf die bereits in der Zielplattform vorhandenen Animationen, welche für neu zu integrierende Anwendungen übernommen werden. Animationen, wie Transitionen zwischen Bildschirmanzeigen, werden daher ohne weitere Definition in *aDDaPt* so übernommen, wie sie von der Zielplattform vorgesehen werden.

Trotz der Grenzen kann der Modellierer sehr präzise entscheiden, wie die Benutzerschnittstelle abstrakt repräsentiert wird, und dadurch den Generierungsprozess beeinflussen. Das Beispiel der Sonderzielsuche im vorigen Abschnitt verdeutlicht dies. Diese Macht des Modellierers wird durch die Kombination von Semantik und Syntax in einem Modell möglich. Dadurch, dass der Modellierer sowohl die Domäne im Datenmodell beschreibt, als auch diese definierten Datenelemente bei der Interaktionsmodellierung in Dialog- und Präsentationsmodell zusammenführt, wird es möglich, den Integrationsprozess komplett zu automatisieren und kontrollierbar zu halten. *aDDaPt* fügt sich in die Lücke zwischen abstrakter, relativ freier Aufgabenmodellierung und finaler Benutzerschnittstellenbeschreibung ein. Dies ermöglicht die Automatisierung bei größtmöglicher Freiheit des Modellierers.

## 8.3. Qualität resultierender Benutzerschnittstellen

Die in den vorigen Abschnitten aufgeführten Einschränkungen bei der Abdeckung von Interaktionen sowie die Grenzen der Generierungsmöglichkeiten mit dem *aDDaPt*-Ansatz eröffnen die Diskussion um die Qualität der resultierenden Benutzerschnittstellen. Es ist unbestritten, dass eine manuelle Entwicklung einer Benutzerschnittstelle, zugeschnitten auf den Benutzungskontext, eine höhere Qualität der resultierenden Benutzerschnittstelle erzielt. Eine manuelle Entwicklung ist allerdings nicht immer möglich, insbesondere dann wenn der Nutzungskontext nicht bekannt ist oder sich ändert. Der Zusatzaufwand,

der bei der Neuentwicklung einer Benutzerschnittstelle für eine Zielplattform entsteht, kann außerdem nicht immer aufgebracht werden. Zuletzt fehlen auch häufig die technischen Möglichkeiten, das im Betrieb befindliche System mit allen Freiheiten zu erweitern. Dies ist insbesondere bei sicherheitskritischen Systemen, die vor der Inbetriebnahme durch aufwändige Testprozesse abgesichert werden müssen, nur mit viel Zusatzaufwand möglich.

Die Diskussion um die Qualität der resultierenden Benutzerschnittstellen findet auf unterschiedlichen Ebenen statt. Für die Ebene einer freien, kreativen Benutzerschnittstellenentwicklung von Anfang bis Ende ist *aDDaPt* nicht geeignet. Einige Ansätze in der Vergangenheit haben diesen Versuch weitestgehend erfolglos gewagt, wie in Kapitel 3 und in Kapitel 6 aufgeführt. Qualitätseinbußen wegen fehlender Freiheiten bei *aDDaPt* steht jedoch die Möglichkeit gegenüber, ein bestehendes System, das bereits entwickelte Benutzerschnittstellenkonzepte verwendet, vollautomatisch zu erweitern. *aDDaPt* bietet einen schlanken, technisch abgesicherten deterministischen Ansatz dafür.

## 8.4. Flexibilität

Im Zusammenhang der im vorigen Abschnitt erwähnten Qualität resultierender Benutzerschnittstellen wird häufig auf die fehlende Flexibilität hingewiesen. So scheint es einfacher und flexibler, ein bestehendes System so bereitzustellen, dass Updates zu späteren Zeitpunkten einfach eingespielt werden können, die dann zielgerichtet auch die Benutzerschnittstelle erweitern. Dies würde außerdem mögliche Probleme bei der Qualität eliminieren, da fehlende oder neue Benutzerschnittstellenteile nachgeliefert werden können. Von Seiten des Systembetreibers wäre dies umsetzbar, allerdings würde es bedeuten, dass jede externe Anwendung individuell integriert werden müsste, was zusätzlichen Entwicklungs- und Testaufwand, wie oben bereits erwähnt, mitbringt. Hierfür sind häufig weder Entwicklungs- noch Wartungsresourcen eingeplant. Zusätzlich würde dieser Aufwand bei jedem System mit jeder neuen Anwendung entstehen.

Im *aDDaPt*-Ansatz ist der einmalige Aufwand der Bereitstellung des Integrationsmoduls pro System höher, dafür entsteht kein zusätzlicher Aufwand bei der Integration beliebig vieler neuer Anwendungen. Der Aufwand für die Vorbereitung zur Integration auf der Seite einer externen Anwendung wird ebenfalls reduziert, da die Entwickler nicht für jedes Zielsystem ein individuelles Modell erstellen müssen, sondern nur ein *aDDaPt*-Modell pro Anwendung. Letztlich ermöglicht *aDDaPt* eine flexiblere Erweiterbarkeit

bestehender Systeme bei etwas weniger Flexibilität bezüglich der Entwicklung oder Anpassung an neue oder veränderte grundlegende Benutzerschnittstellenkonzepte in einem Zielsystem. Besteht mehr Anpassungsbedarf, muss dies über zusätzliche Software- oder Firmware-Updates mit einem Update des Generierungsmoduls gelöst werden.

## 8.5. Anwendbarkeit

Leichte Erlern- und Anwendbarkeit ist gerade bei abstrakten, formalen Modellierungsmethoden kritische Voraussetzung, um für den Einsatz in einem Projekt ausgewählt zu werden. Da *aDDaPt* bezüglich der Notation für die Anwender auf das bekannte und anerkannte Konzept von Zustandsdiagrammen aufbaut, ist eine Basis für geeignete Anwendbarkeit vorhanden. Die Anwendbarkeit von UML für die Modellierung von Interaktionsanwendungen wurde außerdem bereits in einigen Arbeiten untersucht und positiv bewertet [DP11] [Mel10]. Insbesondere die Anwendbarkeit der UML zur Spezifikation von Benutzerschnittstellen wurde mit Nutzern in einer verwandten Arbeit abgesichert [DP10].

Im Rahmen der Umsetzung der Demonstrationsbeispiele für diese Arbeit wurden außerdem immer wieder neue Entwickler mit *aDDaPt* vertraut gemacht und in kurzer Zeit eingearbeitet. Für Teile der Arbeit haben neue Entwickler in nur wenigen Monaten abgeschlossene Beiträge für die technische Konzeptabsicherung beigesteuert, inklusive Einarbeitung in *aDDaPt*. Die Anwendbarkeit von *aDDaPt* durch Entwickler im Rahmen dieser Arbeit kann daher bestätigt werden. Zukünftige Untersuchungen müssen die Anwendbarkeit der Modellierungskonzepte sowie mögliche Werkzeuge oder Methoden weiter im Detail bestätigen.

## 8.6. Multimodale Benutzerschnittstellen

Moderne Benutzerschnittstellen sind häufig multimodal sowohl bezüglich Eingabe als auch bezüglich Ausgabe. Sehsinn, Hörsinn und Tastsinn werden über unterschiedliche Ausgabegeräte angesteuert. Ebenso werden unterschiedliche Eingabegeräte verwendet um beispielsweise sowohl textuelle Eingaben als auch Spracheingaben verarbeiten

zu können. *aDDaPt* wurde für verschiedene Beispiele grafisch-haptischer Benutzerschnittstellen prototypisch umgesetzt und evaluiert. Es wurde kein Fokus auf weitere Modalitäten oder die Kombination von Modalitäten gesetzt. Die folgenden Beispiele illustrieren, dass *aDDaPt* grundsätzlich nicht darauf beschränkt ist, nur grafisch-haptische Benutzerschnittstellen zu beschreiben. Es sind allerdings weitere Arbeiten notwendig, um mehr Aspekte der Erzeugung multimodaler Benutzerschnittstellen zu untersuchen.

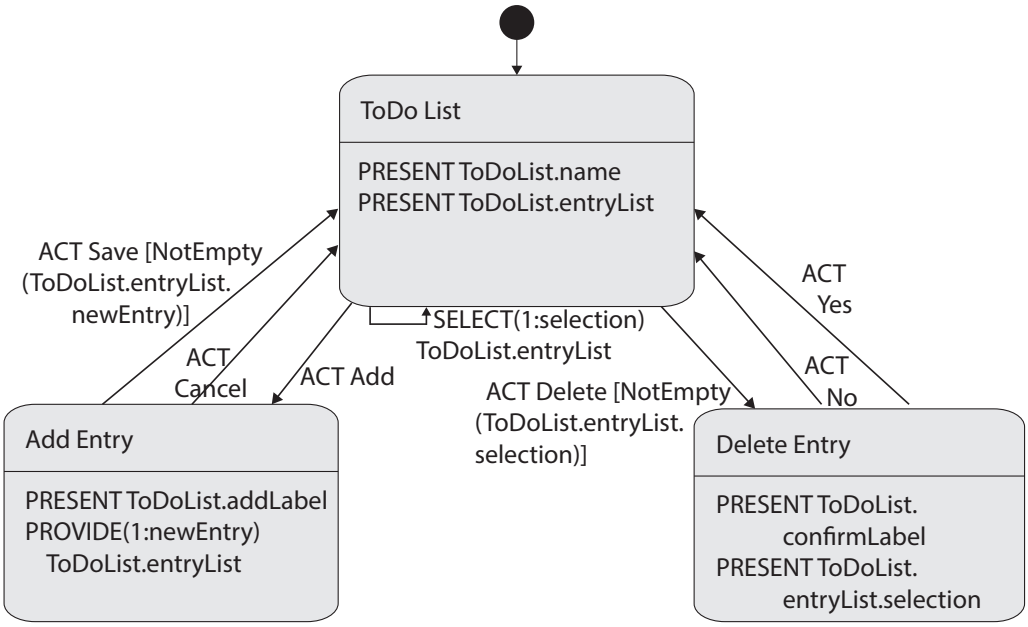


Abbildung 8.3.: *aDDaPt* Dialog- und Präsentationsmodell für die Beispielanwendung ToDo-Liste

**Beispiel: Sprachdialog für die ToDo-Liste** Abbildung 8.3 zeigt erneut das *aDDaPt*-Dialog- und Präsentationsmodell der bereits vorgestellten Anwendung der *ToDo*-Liste. Um einen Sprachdialog daraus abbilden zu können, sind folgende beispielhafte Aspekte für das Generierungsmodul denkbar: PRESENT-Objekte des Typs String werden im Basisregelsatz bei Betreten des Zustands vorgelesen. Handelt es sich um eine Liste von Objekten, die eine gewisse Größe überschreitet, wird zuerst nachgefragt, ob die Liste vorgelesen werden soll. PROVIDE-Objekte werden Schritt für Schritt gefüllt per einfachem Dialog. Gehen ACT-Transitionen aus dem Zustand heraus, fragt das System außerdem standardmäßig, was der Nutzer als Nächstes tun möchte, sofern alle PROVIDE-Objekte befüllt wurden.

Ein Beispiel für einen generierten Dialog aus dem Modell der ToDo-Liste:

- *System: Willkommen! Marias Einkaufsliste. Eintragsliste enthält 13 Einträge. Möchten Sie alle Einträge vorgelesen bekommen oder was möchten Sie als Nächstes tun?*
- Nutzer: Eintrag hinzufügen
- *System: Neuer Eintrag. Bitte sprechen Sie 1 Eintrag für Eintragsliste*
- Nutzer: Milch
- *System: Milch. Was möchten Sie als Nächstes tun?*
- Nutzer: Speichern
- *System: Speichern. Marias Einkaufsliste. Eintragsliste enthält 14 Einträge. Möchten Sie alle Einträge vorgelesen bekommen oder was möchten Sie als Nächstes tun?*

Es ist ersichtlich, dass dies kein besonders schöner Dialog ist. Allerdings ist er als Basisabbildung denkbar und funktioniert. Eine Optimierung der Ableitung von Sprachdialogen muss in weiteren Arbeiten untersucht werden und geht über den Umfang dieser Arbeit hinaus.

**Beispiel: Kombinierte Modalitäten für die ToDo-Liste** Die Regeln für die Generierung des Sprachdialogs können mit grafischen Darstellungen kombiniert werden. So kann zum Beispiel eine Liste standardmäßig nur grafisch angezeigt werden, und der Sprachdialog darauf verweisen. Die Sprachausgabe könnte folgendermaßen generiert werden:

*System: Willkommen! Marias Einkaufsliste. Was möchten Sie als Nächstes tun?*.

Auch ist es denkbar, einen Sprachdialog aus einem beliebigen Zustand heraus zu starten, wenn der Nutzer diesen aktiviert. Die ACT-Transitionen müssten dann beispielsweise als mögliche Nutzeraktionen in das Sprachdialogsystem aufgenommen werden.

Der Sprachdialog, gestartet aus dem Zustand *Delete Entry* bei selektiertem Eintrag *Milch*, würde beispielsweise so aussehen:

- *System: Eintrag löschen? Milch*
- Nutzer: Ja

- *System: Gelöscht: Milch*

Die Bestätigung des gelöschten Eintrags kann nur sprachlich ausgegeben werden, wenn die Anwendung diese Information angemessen übergeben kann, da sie nicht Teil des abstrakten Modells ist. Um diesen und weitere modalitätsspezifische Aspekte weiter zu untersuchen, und die Machbarkeit anhand einer entsprechenden Architektur des Sprachdialogsystems zu demonstrieren, sind ebenfalls weitere Arbeiten notwendig, die über den Umfang dieser Arbeit hinausgehen.

Generell wird jedoch durch diese einfachen Beispiele deutlich, dass *aDDaPt* nicht an eine Modalität gebunden ist. Eingabe- und Ausgabegeräte werden in der vorgestellten Architektur, die Grundlage für die Umsetzungen ist, separat vom *aDDaPt*-Modell gehalten und sind flexibel austauschbar. Weitergehende Arbeiten müssen die Machbarkeit der Verwendung von *aDDaPt* mit Ein- und Ausgabegeräten für weitere Modalitäten sowie deren Kombination genauer untersuchen.

## 8.7. Fazit

Das Konzept der abstrakten Domänen-, Dialog- und Präsentationsmodellierung, *aDDaPt*, bewegt sich zwischen abstrakter, relativ freier Aufgabenmodellierung und finaler Benutzerschnittstellenbeschreibung. Dieser zielgerichtete Kompromiss ermöglicht die garantierte Abbildbarkeit korrekt modellierter Benutzerschnittstellen. Die Freiheit des Modellierers spielt dabei eine bedeutende Rolle, da er mit dem Modell Semantik und Syntax vereint und damit die Generierung unmittelbar beeinflusst. Benutzbarkeit und Qualität der resultierenden Benutzerschnittstellen hängen also zum großen Teil von der Modellierung ab. Zukünftige Arbeiten müssen den weiteren Einbezug semantischer Aspekte in die Generierung untersuchen, um die Regelsätze weiter zu optimieren und das Risiko schlechter Modellierung einzuschränken.

Für die modellbasierte Unterstützung der Benutzerschnittstellenentwicklung ist *aDDaPt* nicht vorgesehen und könnte den Kreativprozess negativ beeinflussen. Die Modellierung und Integration auf Basis von *aDDaPt* zielt vielmehr auf die nachgelagerte Erweiterung bestehender Systeme mit bereits definierten Benutzerschnittstellenelementen ab. Letztlich ermöglicht *aDDaPt* eine flexible Erweiterbarkeit bestehender Systeme bei etwas weniger Flexibilität bezüglich der Entwicklung oder Anpassung an neue oder veränderte grundlegende Benutzerschnittstellenkonzepte in einem Zielsystem.

Die Anwendbarkeit von *aDDaPt* durch Entwickler konnte im Rahmen der Erstellung der Beispielmodelle sowie der prototypischen Integration in Beispielbenutzerschnittstellen bestätigt werden. Zukünftige Untersuchungen müssen die Anwendbarkeit der Modellierungskonzepte sowie mögliche Werkzeuge oder Methoden weiter im Detail bestätigen.

Während *aDDaPt* im Kern modalitätsunabhängig ist, wurden die entwickelten Modelle prototypisch nur in grafisch-haptische Benutzerschnittstellen integriert. Die grundlegenden Überlegungen zur Generierung in Sprachdialogsysteme oder kombinierte multimodale Systeme müssen weitergeführt und umfangreich untersucht werden.

# 9

# Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde das Ziel verfolgt, Benutzerschnittstellen für Anwendungen in zuvor unbekannte Kontexte zu integrieren. Dafür wurde ein Konzept entwickelt, umgesetzt und bewertet, welches eine flexible Benutzerschnittstellenintegration über definierte Schnittstellen ermöglicht, die unabhängig von verwendeten Bedien- und Anzeigekonzepten oder eingesetzten Plattformen sind. In diesem abschließenden Kapitel werden die Beiträge der Arbeit zusammengefasst und ein Ausblick auf zukünftige Entwicklungen und Forschungsfragen gegeben.

## 9.1. Beiträge der Arbeit

Die Problemstellung und Ziele der Arbeit wurden in Kapitel 1 anhand der Vision einer einfachen Nutzung einer Computeranwendung in jedem Kontext motiviert. Wichtige Begriffe und Wissen zur Mensch-Computer Interaktion (HCI, engl. *Human-Computer Interaction*) und Methoden zur Entwicklung von Benutzerschnittstellen wurden in Kapitel 2 erarbeitet, um den Forschungskontext der Arbeit zu definieren und die notwendigen Grundlagen

für die Entwicklung von Lösungsansätzen zu festigen. Als besonders geeignetes Anwendungsbeispiel, wegen der fest integrierten Interaktionsmöglichkeiten sowie des besonderen Interaktionskontexts, wurden Benutzerschnittstellen im Automobil vorgestellt und Anforderungen an diese herausgestellt. Modellierung von Benutzerschnittstellen hat sich für die oft formalen Anforderungen in diesem Bereich durchgesetzt. Ansätze zur modellbasierten und modellgetriebenen Entwicklung von Benutzerschnittstellen wurden daher in Kapitel 3 diskutiert und gegenübergestellt. Die wichtigsten Modellarten wurden vorgestellt und der grundlegende Ansatz, die notwendige Aufgabenmodellierung mit weiteren Modellen zu einem abstrakten Ausgangsmodell zu vereinen, wurde anhand bestehender Literatur und ersten Anforderungsanalysen erarbeitet. Um optimale Flexibilität zu erreichen, wurde die Verwendung eines möglichst allgemeingültigen, vielfältig anwendbaren Ansatzes, etwa auf der Basis der UML, herausgearbeitet.

Das Konzept für Interaktionsmodelle zur Integration in bestehende Benutzerschnittstellen wurde in Kapitel 4 weiter erarbeitet. Um Aufwand für Implementierung und Test je Zielkonzept zu reduzieren, aber gleichzeitig die Kontrolle über die Darstellung der Benutzerschnittstelle zu behalten, wurde ein modellgetriebener Ansatz als bestmögliche Lösung bestimmt. Da die Verwendung von klassischen Aufgabenmodellen nicht genügend eindeutige Informationen für einen vollautomatischen Integrationsprozess der Modelle in bestehende Benutzerschnittstellen bereitstellt, wurde eine Kombination von Domänen-, Dialog- und Präsentationsmodell als abstraktes Interaktionsmodell (***aDDaPt***, als Abkürzung für ***a****bstract* ***D****omain,* ***D****ialog* ***a****nd* ***P****resen****t****ation model*) erarbeitet. Die Notationsform des *aDDaPt*-Modells wurde auf Grundlage von Klassen- und Zustandsdiagrammen im Sinne der UML definiert. Der Prozess der Weiterverarbeitung von *aDDaPt*-Modellen wurde so erarbeitet, dass durch die Verwendung eines garantierten Basislayouts sowie Basisabbildungsregeln eine vollständige Abbildbarkeit sichergestellt werden kann. Eine Regelhierarchie auf Grundlage der Basisregeln sorgt außerdem für Flexibilität in der Erzeugung der Benutzerschnittstellen. Mit dem *aDDaPt*-Ansatz werden flexible Benutzerschnittstellen ermöglicht, die neue Anwendungen vollautomatisch integrieren können, unabhängig von Ein- und Ausgabegeräten, verwandter Software oder Hardware.

Eine Referenzarchitektur zur flexiblen Einbindung von *aDDaPt*-Modellen in bestehende Benutzerschnittstellen wurde auf Basis des Model-View-Controller (MVC) Patterns prototypisch umgesetzt und in Kapitel 5 vorgestellt. Auf Basis dieser Architektur wurden verschiedene Demonstrationsanwendungen und Zielsysteme entwickelt, die in Kapitel

6 am Beispiel des Nutzungskontexts Automobil bewertet wurden. Wie oben erwähnt, eignet sich der Automobilbereich besonders gut für die Anwendung von *aDDaPt*, da sich dort wegen der besonderen Anforderungen an sichere und ablenkungsfreie Bedienbarkeit im Fahrzeug ein methodischer, oft modellbasierter Entwicklungsprozess etabliert hat, um den hohen Anforderungen für Entwicklung und Test gerecht werden zu können. Flexibilität und Abwärtskompatibilität von *aDDaPt* passen ebenfalls besonders gut zum Automobilbereich wegen der langen Lebenserwartung der Systeme.

Die Beispielumsetzungen demonstrieren die Anwendbarkeit von *aDDaPt* mit einer Bandbreite an verwendeten Technologien für alle Arten von Clients innerhalb der konzipierten Architektur. Die Erfüllung der Anforderungen an ein Konzept zur Integration von Benutzerschnittstellen in ein bestehendes Zielsystem konnte erfolgreich nachgewiesen werden. Durch die Flexibilität in der Modellierung und Integration können Anwendungen sowohl auf einen minimierten Satz an Interaktionen reduziert werden, um auch in Systemen mit weniger Interaktionselementen integrierbar zu sein, als auch zielgerichtet auf vorhandene komplexe Interaktionselemente einer noch unbekannten Zielumgebung abgebildet werden.

## 9.2. Schlussfolgerungen und Ausblick

Mobile Geräte und Anwendungen verwenden eine Vielzahl proprietärer Formate und Interaktionselemente. Dies verhindert eine nahtlose Integration der Anwendungen in unterschiedlichen Kontexten und Zielsystemen. Das in dieser Arbeit entstandene Konzept für abstrakte Interaktionsmodelle und deren Integration in bestehende Benutzerschnittstellen trägt zur Lösung dieses Problems bei. Der Ansatz beschränkt sich dabei nicht auf eine zu Grunde liegende Architektur oder Technologie, wie etwa bei durch Webtechnologien getriebenen Ansätzen, sondern erhält die Flexibilität sowohl bezüglich der Technologie im Zielsystem als auch bezüglich der verwendeten Technologie einer Anwendung, die integriert werden soll.

Ausgehend von den Ergebnissen dieser Arbeit wurden Themenfelder identifiziert, die in zukünftigen Arbeiten weiter untersucht werden können. Während es nicht leicht vorstellbar ist, dass komplett neue Benutzerschnittstellenelemente oder grafische Designs Ergebnis einer vollautomatischen Generierung sein werden, so kann durch den Ausbau der semantischen Abstraktion für die Benutzerschnittstellenbeschreibung dieses Thema

weiter vorangetrieben werden, sodass der Freiheitsgrad im Aufbau der Generierungsregeln vergrößert werden kann. Im vorgestellten Ansatz sind die Generierungsregeln stark an die Modalität und konkreten Elemente in der Zielplattform gebunden. Eine Lockerung dieser Bindung kann zu mehr Freiheiten führen, sofern die Semantik klar beschrieben ist. Dies würde außerdem die Generierung intelligenterer multimodaler Benutzerschnittstellen noch besser ermöglichen.

Für eine breite Evaluierung der Anwendbarkeit der vorgestellten Ansätze müssten die Modellierungskonzepte in Form von Entwicklungswerkzeugen oder -prozessen bereitgestellt werden, was nicht im unmittelbaren Fokus der Arbeit stand. Hier können zukünftige Arbeiten ansetzen, um die Prozesse zu optimieren und detailliert zu evaluieren.

Die erarbeiteten Konzepte wurden im Rahmen der Umsetzung in dieser Arbeit evaluiert. Sie fließen außerdem weiter in Benutzerschnittstellenprojekte für den Automobilbereich ein, und damit das Wissen, das während der Entstehung dieser Arbeit aufgebaut und stets angewandt wurde. Die Ergebnisse bringen die Weiterentwicklung von übergreifenden UI-Beschreibungssprachen, Standards, grundlegenden Architekturen und Entwicklungsprozessen voran. Die vorliegende Arbeit ist daher ein wichtiger Beitrag auf dem Weg zur Vision einer angepassten Nutzung von Computeranwendungen in jedem Kontext, wie sie ursprünglich von Mark Weiser unter dem Namen *Ubiquitous Computing* [Wei91] aufgestellt wurde.

# A

# Kommunikations-Architektur: Nachrichtenaustausch beim Verbindungsaufbau

Das Nachrichtenformat *TKMessage*, das beim Datenaustausch zwischen verbundenen und bestehenden Clients verwendet wird, wurde in Kapitel 5 vorgestellt, in dem auch ein Interaktionssequenzdiagramm für den Nachrichtenaustausch im Falle der Verbindung eines dem Server noch nicht bekannten Provider-Clients Illustriert wird. Im Folgenden werden die weiteren Situationen beim Verbindungsaufbau zwischen den Clients und Server illustriert.

## A.1. Verbindung externer Provider mit Presentation-Client

Die folgenden Interaktionssequenzdiagramme veranschaulichen den Nachrichtenaustausch bei der Verbindung externer Anwendungen (Provider-Clients) sowie der Verbindung von Presentation-Clients.

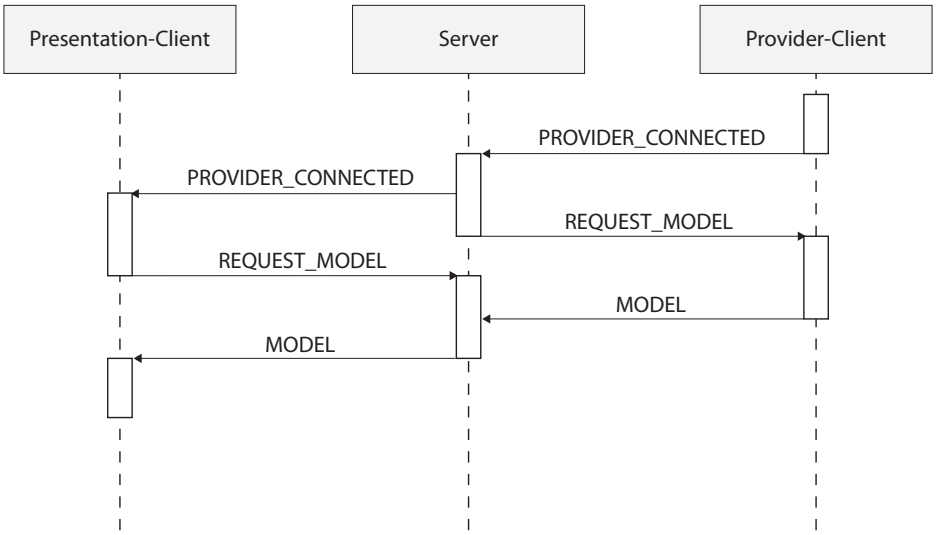


Abbildung A.1.: Nachrichtenaustausch zwischen Clients und Server bei der Verbindung eines dem Server nicht bekannten Provider-Clients.

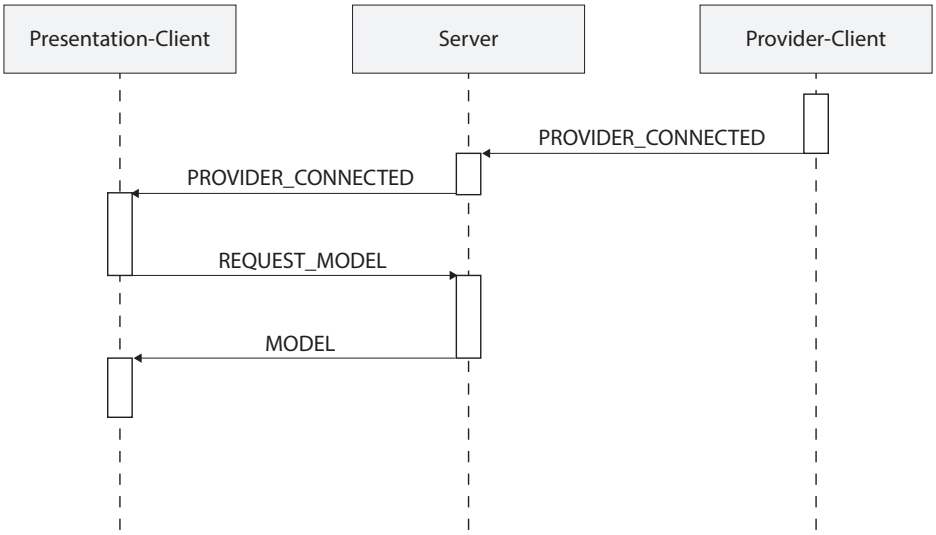


Abbildung A.2.: Nachrichtenaustausch bei der Verbindung eines dem Server bereits bekannten Provider-Clients. Der Presentation-Client fordert das gespeicherte Model beim Server an, wo es bereits vorhanden ist.

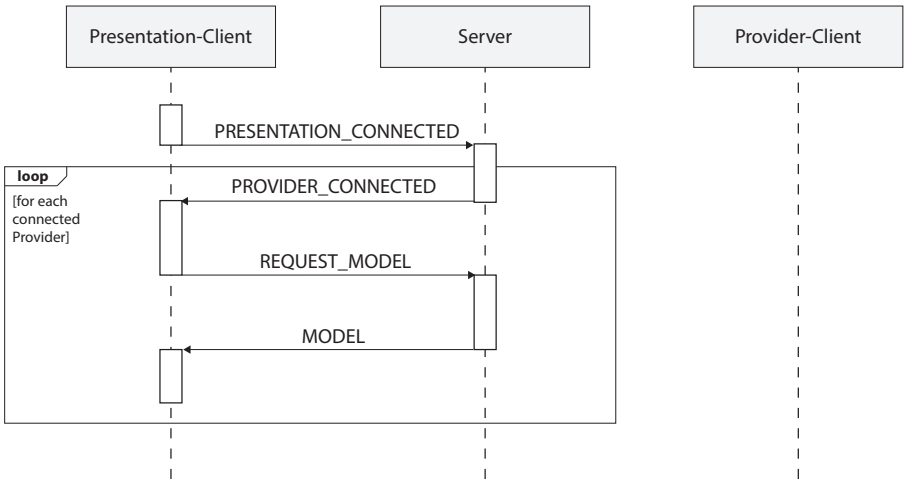


Abbildung A.3.: Anmeldung eines neuen Presentation-Clients. Jeder bekannte verbundene Provider-Client sowie das entsprechende Model wird dem Presentation-Client bekannt gemacht.

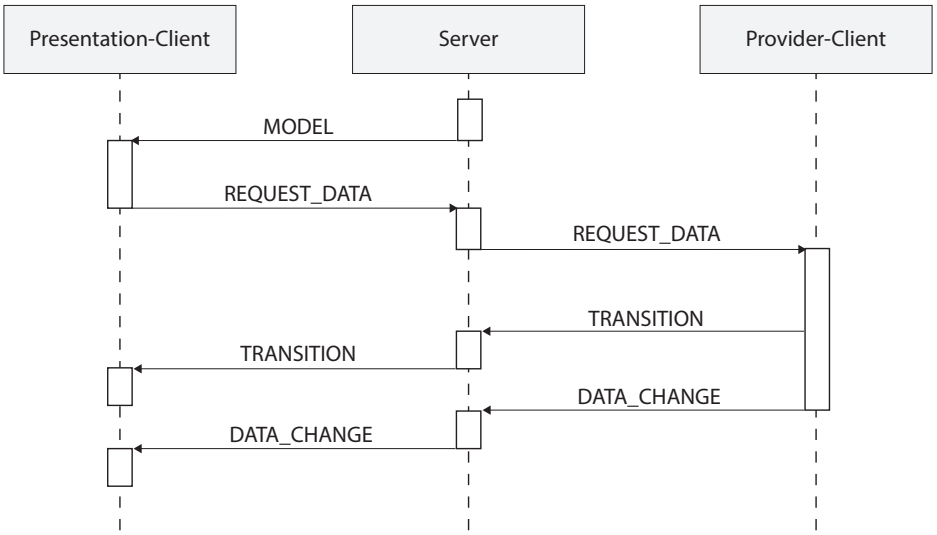


Abbildung A.4.: Initiale Datenbelegung und Zustand. Nach der Anmeldung eines Provider-Clients fordert der Presentation-Client die Datenbelegung an. Der Provider-Client sendet zusätzlich zur Datenbelegung eine initiale Transition um den zu startenden Zustand zu kommunizieren.

## A.2. Verbindung von Eingabegeräten (Input-Clients)

Die folgenden beiden Interaktionssequenzdiagramme veranschaulichen den Nachrichtenaustausch zwischen Presentation-Client und Input-Client.

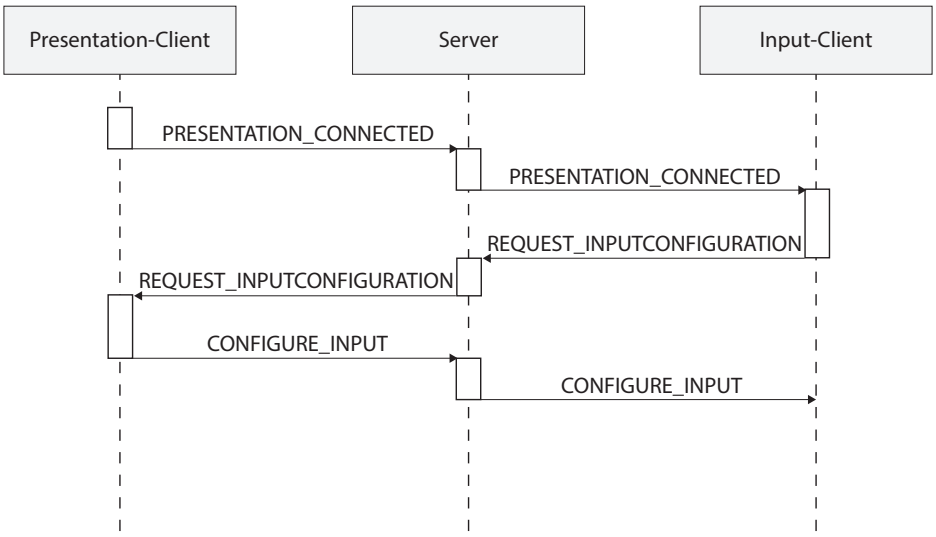


Abbildung A.5.: Anmeldung eines Presentation-Clients: Der Input-Client reagiert und fragt die relevanten Daten zur korrekten Konfiguration an, die für Rückmeldung gebende Teile eines Eingabegerätes relevant sind.

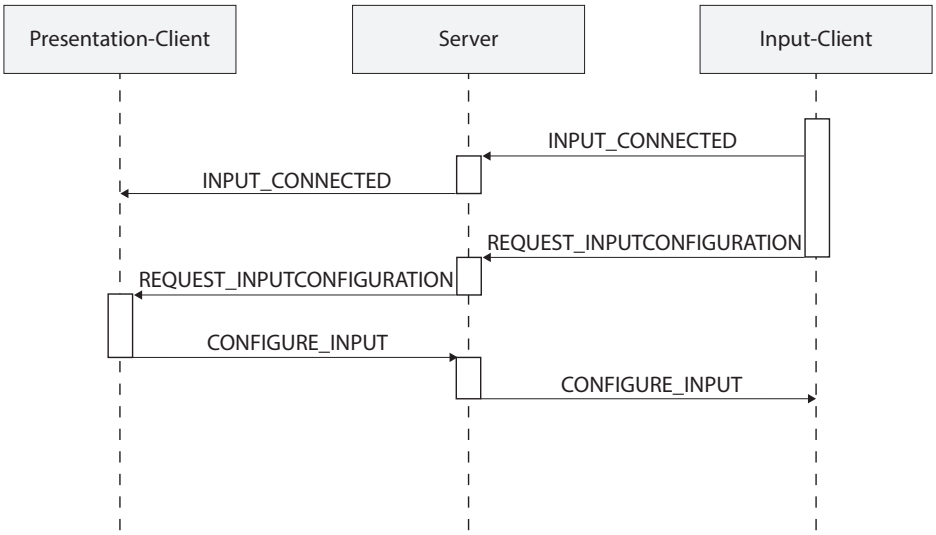


Abbildung A.6.: Anmeldung eines Input-Clients mit Anfrage zur Konfiguration Rückmeldung gebender Teile des Eingabegerätes.

# B

# aDDaPt: Semantische Transitionen

Eine Erweiterung des Kernkonzepts von *aDDaPt* sind semantisch annotierte Transitionen, die im Rahmen der Arbeit untersucht wurden. Ein möglicher Satz semantisch annotierter Transitionen wurde für die Umsetzung der Gelbe-Seiten-Anwendung, die im Abschnitt 6.4. vorgestellt wurde, verwendet, und besteht aus den folgenden Transitionen:

`NAVIGATE` wird für eine Transition in einen neuen Zustand verwendet. *Besonderheiten:* Darf nicht an ein Datenobjekt gebunden sein und kann durch Bedingungen eingeschränkt sein. Keine andere Transition darf zum selben Zielzustand führen.

`INVOKE` wird verwendet wenn durch die Transition eine Systemapplikation aufgerufen wird. *Besonderheiten:* Geht die Transition aus einem Unterzustand heraus, wird ein gebundenes Datenobjekt als Parameter mitgeliefert.

`ACTIVATE` beschreibt eine Transition, die den Unterzustand eines Datenobjekts, an das diese gebunden ist, betritt. *Besonderheiten:* Eine SELECT-Transition muss davor ausgeführt worden sein.

`CANCEL` wird verwendet, wenn eine Transition zum Ausgangszustand zurückführt. *Besonderheiten:* Dabei werden keine Änderungen an Daten übernommen.

`SELECT` fokussiert einen Eintrag und ist daher an den Container des Eintrags gebunden. *Besonderheiten*: Ermöglicht die ACTIVATE-Transition für den entsprechenden Eintrag.

`MODIFY` ermöglicht die Veränderung eines Datenobjekts, an das die Transition gebunden ist. *Besonderheiten*: Betritt einen neuen Zustand, in den keine anderen Transitionen eingehen dürfen. Dieser Zustand muss mindestens ein PROVIDE-Objekt beinhalten. Aus dem Zustand heraus können nur ADD, UPDATE, DELETE oder CANCEL führen.

`ADD` fügt ein Datenobjekt, das über PROVIDE bereitgestellt wurde, zur entsprechenden Collection hinzu. *Besonderheiten:* Erfordert eine vorhergehende Transition MODIFY, die das Datenobjekt bereitstellt.

`DELETE` entfernt das gebundene Datenobjekt. *Besonderheiten:* Erfordert eine vorhergehende Transition MODIFY, die das Datenobjekt bereitstellt.

`UPDATE` ersetzt das gebundene Datenobjekt. *Besonderheiten:* Erfordert eine vorhergehende Transition MODIFY, die das Datenobjekt bereitstellt.

Für weitere Details sei auf die im Rahmen dieser Arbeit entstandene Ausarbeitung von Timo Müller [Mï2] verwiesen.

Die folgenden Abbildungen beinhalten Domänen- und Dialogmodell der Gelbe-Seiten-Anwendung aus Abschnitt 6.4.

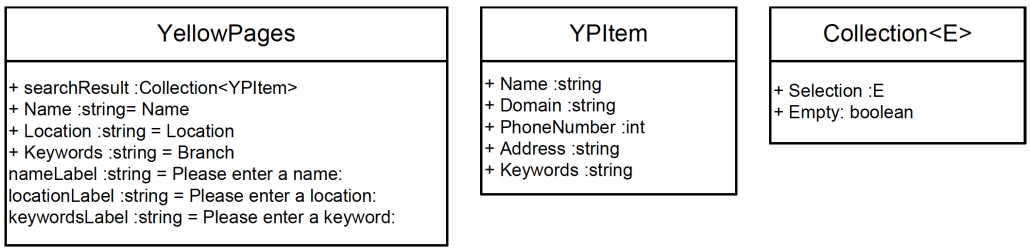


Abbildung B.1.: *aDDaPt*-Domänenmodell für die Gelbe-Seiten-Anwendung aus Abschnitt 6.4.

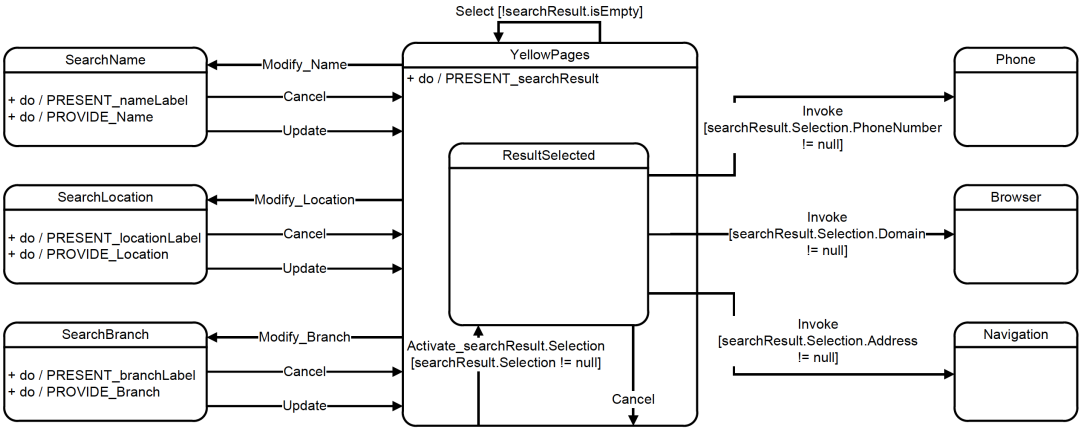


Abbildung B.2.: *aDDaPt*-Dialogmodell für die Gelbe-Seiten-Anwendung aus Abschnitt 6.4.

# Abbildungsverzeichnis

1.1. Problemstellung: Flexible Schnittstellen zwischen Anwendung, verschiedenen Interaktionskontexten und Nutzer, um eine Computeranwendung perfekt und nahtlos in die Umgebung des Nutzers zu integrieren. . . . . 4

2.1. Teilgebiete der Mensch-Computer Interaktion aus dem Curriculum der ACM [ACM92]. . . . . 12
2.2. Textbasiertes Eingabefenster zur Dateiverwaltung, basierend auf Microsoft DOS. . . . . 14
2.3. Berührbare grafische Oberfläche eines *Apple iPad Air 2* (Bildnachweis: S. 168). . . . . 15
2.4. Gamepad mit Motoren für *Force Feedback* (Bildnachweis: S. 168). . . . . 16
2.5. Kamera-Sensorleiste *Kinect* zur Steuerung der Spielekonsole *Microsoft XBOX 360* über Gesten (Bildnachweis: S. 168). . . . . 19
2.6. Haptische Eingabegeräte im Griffbereich zwischen Fahrer und Beifahrer. *Links:* Mercedes-Benz E-Klasse, 2011; *Mitte:* BMW 650i, 2011; *Rechts:* Audi A6, 2014 (Bildnachweis: S. 168). . . . . 26
2.7. Zentraler 17-Zoll Touchscreen im Tesla Model S, 2015 (Bildnachweis: S. 168). . . . . 26
2.8. Zentral angeordnete Bedienelemente für Infotainmentanwendungen und Touchpad mit Tastenfeld und Zeichenerkennung im Audi A8, 2009 (Bildnachweis: S. 168). . . . . 27
2.9. Multifunktionslenkrad mit Bedienelementen für Infotainmentanwendungen in der Mercedes-Benz S-Klasse, 2013 (Bildnachweis: S. 168). . . . . 27
2.10. Anzeigetechnologie im Fahrzeug. *Links:* Zentraler Bildschirm, Audi A3, 2012; *Rechts:* Head-Up-Display und Kombiinstrument im BMW, 2012 (Bildnachweis: S. 168). . . . . 28

2.11. Innenraum der Fahrzeugstudie F125*!* von Mercedes-Benz, 2012, mit 3D-Display, 3D-Touchpad, Sprachbedienung für den Fahrer sowie einem großen Bildschirm auf der Beifahrerseite mit Anwendungen, die per Handgesten gesteuert werden (Bildnachweis: S. 168). . . . . 29
2.12. Kontextsensitive Benutzerschnittstelle in einem lernenden System von Mercedes-Benz, Prototyp präsentiert auf der Consumer Electronics Show in Las Vegas, 2014 (Bildnachweis: S. 168). . . . . 30
2.13. Usability Engineering Lifecycle nach Mayhew [May99]. . . . . 33
2.14. Vier Säulen erfolgreicher Benutzerschnittstellenentwicklung (nach [SP10]). 34
2.15. Auszug aus der Software EB GUIDE Studio, die begleitend zum Förderprojekt *AutomotiveHMI* (`www.automotive-hmi.org`) verwendet wurde. Oben: Die Erstellung von *Views* durch *Widgets*. Unten: Ablaufsteuerung über Zustandsdiagramm. . . . . 42
2.16. Illustration zur *HMI View* aus der systematischen Betrachtung und Erarbeitung eines modell-basierten Ansatzes für die HMI-Entwicklung im Automobilbereich [HGM+12]. . . . . 43

3.1. Aufgabenmodell in CTT für das Beispiel *ToDo-Liste*. . . . . 49
3.2. Entity-Relationship Diagramm für das Beispiel *ToDo-Liste*. . . . . 50
3.3. Dialogmodell als Zustandsdiagramm für das Beispiel *ToDo-Liste*. . . . . 51
3.4. Das Cameleon Reference Framework [CCT+03] . . . . . 54

4.1. Spannungsfeld der Integration externer Applikationen ins Fahrzeug mit den drei Parteien Fahrzeughersteller, Anbieter von externen Anwendungen und Anwender . . . . . 63
4.2. Gewichtetes Spannungsfeld der Integration externer Applikationen ins Fahrzeug. . . . . 67
4.3. Formalisierung (gestrichelte Linie) bei der Entwicklung einer Benutzerschnittstelle im Vergleich zum Gesamtentwicklungsprozess (vereinfacht, ohne Iterationen). . . . . 72
4.4. *Ziel 1* ist ein Modellierungskonzept für die flexible abstrakte Beschreibung von Benutzerschnittstellen . . . . . 75
4.5. *Ziel 2* ist ein Konzept für einen Transformationsprozess für die automatische Weiterverarbeitung und Integration der abstrakten Beschreibung . . 75
4.6. *Ziel 3* ist ein Konzept für einen erweiterbaren, flexiblen Entwicklungsprozess von Benutzerschnittstellen . . . . . 76

4.7. Einordnung des kombinierten abstrakten Interaktionsmodells in die Abstraktionsebenen des CAMELEON Reference Framework. Die Ebenen der Aufgaben und Konzepte und der abstrakten Benutzerschnittstelle werden zusammengelegt. Die Transformationsregeln enthalten konkrete Informationen über das Zielkonzept, in das die modellierte Benutzerschnittstelle integriert werden soll. . . . . 84
4.8. *aDDaPt Domain model*: Darstellung mit Attributen inklusive einer Sammlung von Werten (*Collection*) eines eigenen Datentyps (*Type1*), der mit mehreren Attributen und einer Methode (*method1*) definiert ist. . . . . 86
4.9. *aDDaPt Dialog model*: In einem Zustandsdiagramm wird das dynamische Verhalten durch Ereignisse beschrieben, die an optionale Bedingungen geknüpft sind, und Interaktionszustandsübergänge sowie Aktionen auslösen können. . . . . 88
4.10. *aDDaPt model*: Domänenmodell und Dialogmodell mit Präsentationsmodell. Beispiele zur Darstellung von Präsentationsobjekten sowie Beispiel für das Nutzerereignis ACT. . . . . 89
4.11. *aDDaPt Domain model* für die Beispielanwendung ToDo-Liste . . . . . 90
4.12. *aDDaPt* Dialog- und Präsentationsmodell für die Beispielanwendung ToDo-Liste . . . . . 91
4.13. *aDDaPt Domain model* für die Beispielanwendung JoinMyRide . . . . . 92
4.14. *aDDaPt* Dialog- und Präsentationsmodell für die Beispielanwendung *JoinMyRide* . . . . . 93
4.15. Vereinfachte Darstellung eines generisches Listenmodells für eine Liste, bei der zwei Optionen je Eintrag auswählbar sind. . . . . 94
4.16. Weiterverarbeitung von *aDDaPt* im Überblick. UI 1 und UI 2 sind dabei zwei verschiedene Benutzerschnittstellen von Zielsystemen. . . . . 96
4.17. *UI rules:* Beispiel eines Basislayouts: Verbundene Provider, das sind die Applikationen, sind in einer Zeile oben, ACT-Transitionen sind in einer Zeile unten, und PRESENT, PROVIDE und SELECT Elemente sind, logisch in Zeilen getrennt, im mittleren Hauptbereich dargestellt. . . . . 97
4.18. *UI rules:* Beispiel eines alternativen Layouts: Der mittlere Hauptbereich ist in zwei logische Spalten eingeteilt, links können nur PRESENT-Inhalte präsentiert werden, rechts PRESENT, PROVIDE und SELECT Elemente. 98

4.19. *UI rules:* Konzept der Regelhierarchie. Kleinster gemeinsamer Nenner sind die Basisabbildungen, die 1:1 Elemente aus dem *aDDaPt*-Modell (e..) auf Elemente des Zielsystems (E..) abbilden. Darauf aufbauend sind Regeln (R..) und Kombinationen der Regeln (K..) möglich. . . . . 99

5.1. Übersicht der Client-Server-Architektur mit dem Server und seinen Kernaufgaben, möglichen Clients und Kommunikationswegen. Server-Addins sind in Form der schwarzen Quadrate illustriert. . . . . 107
5.2. Übersicht Server: Der Eventhandler im Server regelt Empfang und Verbreitung von Nachrichten. UI-Generation ist eine vom Server bereitgestellte Methode, die vom Presentation-Addin aufgerufen werden kann. AppList dient der Zwischenspeicherung von Informationen zu bekannten und verbundenen Applikationen. . . . . 109
5.3. Nachrichtenaustausch zwischen Clients und Server: Verbindung eines dem Server nicht bekannten Provider-Clients. . . . . 114

6.1. Demonstration der Client-Server-Architektur mit Beispielumsetzung: Silverlight-Anwendung als *Presentation*, Touchapplikation oder zentrales Bedienelement im Automobil als *Input*-Clients, iOS-Applikation als *Provider*. 119
6.2. Beispiele von UI-Prototypen, die begleitend zur Arbeit entstanden sind: a)+b) Adobe Flash Umsetzungen von UIs auf Basis des gleichen Anwendungsmodells; c) UI-Umsetzung in Adobe Flash; d) UI-Umsetzung in Microsoft Silverlight. . . . . 121
6.3. Konzept der Mehr-Ebenen-Hierarchie mit unterschiedlichen Abstraktionsstufen. Die Generierung erfolgt auf der Ebene, die am nächsten zum *Final User Interface* ist, sofern eine anwendbare Regel gefunden wird. . . . . 124
6.4. Screenshots der generierten Benutzerschnittstellen für die Beispielumsetzung der *Gelbe Seiten*-Anwendung (aus [M̈12]). Links: Ein Kontextmenü erlaubt den Aufruf verfügbarer Zusatzdienste, wie einen Browser. Rechts: Im Touchscreen-Konzept kann ein ausgewählter Datensatz per *Drag and Drop* mit einem entsprechenden Zusatzdienst direkt geöffnet werden. . . 127

7.1. *aDDaPt* kombiniert Elemente verschiedener Modellarten in ein abstraktes Interaktionsmodell, bestehend aus Domänen-, Dialog- und Präsentationsmodell. . . . . 130

7.2. Ergebnis: Flexible, abstrakte Modellierungsmethode, mit der die Integration einer Computeranwendung in verschiedenen Interaktionskontexten nahtlos und abgestimmt auf die Zielumgebung ermöglicht wird. . . . . 134

8.1. Beispiel eines *aDDaPt*-Modells für eine Anwendung zur Sonderzielsuche (*POI Search*). . . . . 139
8.2. Mögliche Darstellung der generierten Benutzerschnittstelle für die Sonderzielsuche in (a) ein System mit Navigationsmodul und (b) ein System ohne Navigationsmodul. . . . . 139
8.3. *aDDaPt* Dialog- und Präsentationsmodell für die Beispielanwendung ToDo-Liste . . . . . 144

A.1. Nachrichtenaustausch zwischen Clients und Server bei der Verbindung eines dem Server nicht bekannten Provider-Clients. . . . . 154
A.2. Nachrichtenaustausch bei der Verbindung eines dem Server bereits bekannten Provider-Clients. Der Presentation-Client fordert das gespeicherte Model beim Server an, wo es bereits vorhanden ist. . . . . 154
A.3. Anmeldung eines neuen Presentation-Clients. Jeder bekannte verbundene Provider-Client sowie das entsprechende Model wird dem Presentation-Client bekannt gemacht. . . . . 155
A.4. Initiale Datenbelegung und Zustand. Nach der Anmeldung eines Provider-Clients fordert der Presentation-Client die Datenbelegung an. Der Provider-Client sendet zusätzlich zur Datenbelegung eine initiale Transition um den zu startenden Zustand zu kommunizieren. . . . . 155
A.5. Anmeldung eines Presentation-Clients: Der Input-Client reagiert und fragt die relevanten Daten zur korrekten Konfiguration an, die für Rückmeldung gebende Teile eines Eingabegerätes relevant sind. . . . . 156
A.6. Anmeldung eines Input-Clients mit Anfrage zur Konfiguration Rückmeldung gebender Teile des Eingabegerätes. . . . . 157

B.1. *aDDaPt*-Domänenmodell für die Gelbe-Seiten-Anwendung aus Abschnitt 6.4. . . . . 160
B.2. *aDDaPt*-Dialogmodell für die Gelbe-Seiten-Anwendung aus Abschnitt 6.4. 161

**Bildnachweis**

Die folgende Liste enthält Angaben zu Abbildungen in dieser Arbeit, die nicht selbst erstellt wurden:

Abbildung 2.3.: Von Nutzer:Netspy (Eigenes Werk), CC BY 3.0, (Quelle: https://commons.wikimedia.org/w/index.php?curid=44055812, aufgerufen am 14.03.2016).

Abbildung 2.4.: Von Nutzer:Mikli (Eigenes Werk), CC BY-SA 2.5, (Quelle: https://commons.wikimedia.org/wiki/File:Speed_Link_FirstStrike_USB-Gamepad.jpg, aufgerufen am 30.03.2016).

Abbildung 2.5.: Von Nutzer:Evan-Amos (Eigenes Werk), gemeinfrei, (Quelle: https://upload.wikimedia.org/wikipedia/commons/6/67/Xbox-360-Kinect-Standalone.png, aufgerufen am 14.03.2016).

Abbildung 2.6. links: Von Daimler AG, Pressebild, nichtkommerzielle Nutzung für Bildungszwecke, http://media.daimler.com/, aufgerufen am 30.03.2016.

Abbildung 2.6. Mitte: Von BMW AG, Pressebild, nichtkommerzielle Nutzung für Bildungszwecke, https://www.press.bmwgroup.com, aufgerufen am 30.03.2016.

Abbildung 2.6. rechts: Von Audi AG, Pressebild, nichtkommerzielle Nutzung für Bildungszwecke, https://www.audi-mediacenter.com, aufgerufen am 30.03.2016.

Abbildung 2.7.: Von Edward Colson (Eigenes Werk), privat bereitgestellt.

Abbildung 2.8.: Von Audi AG, Pressebild, nichtkommerzielle Nutzung für Bildungszwecke, https://www.audi-mediacenter.com, aufgerufen am 30.03.2016.

Abbildung 2.9.: Von Daimler AG, Pressebild, nichtkommerzielle Nutzung für Bildungszwecke, http://media.daimler.com/, aufgerufen am 30.03.2016.

Abbildung 2.10. links: Von Audi AG, Pressebild, nichtkommerzielle Nutzung für Bildungszwecke, https://www.audi-mediacenter.com, aufgerufen am 30.03.2016.

Abbildung 2.10. rechts: Von BMW AG, Pressebild, nichtkommerzielle Nutzung für Bildungszwecke, https://www.press.bmwgroup.com, aufgerufen am 30.03.2016.

Abbildung 2.11.: Von Daimler AG, Pressebild, nichtkommerzielle Nutzung für Bildungszwecke, http://media.daimler.com/, aufgerufen am 30.03.2016.

Abbildung 2.12.: Von Daimler AG, Pressebild, nichtkommerzielle Nutzung für Bildungszwecke, http://ces.mercedes-pressevents.com/, aufgerufen am 30.03.2016.

# Tabellenverzeichnis

4.1. Lösungen zur Integration externer Anwendungen in bestehende Benutzerschnittstellen - Einteilung in Software- und Hardware-Lösungen für die Entwicklungszeit und für die Laufzeit . . . . . 69
4.2. Wichtige Vor- und Nachteile von Software-Lösungen für die Entwicklungszeit und für die Laufzeit . . . . . 70
4.3. Relevante Modellarten (in Fettschrift) für die Integration einer Anwendung in eine bestehende Benutzerschnittstelle (Zielsystem) . . . . . 80

# Literaturverzeichnis


[ACM92] ACM: *Curricula for Human-Computer Interaction*. http://old.sigchi.org/cdg/. Version: 1992. – Aufruf: 12.10.2011

[Ake06] AKERMAN, James R. (Hrsg.): *Cartographies of travel and navigation*. University of Chicago Press, 2006

[All06] ALLIANCE OF AUTOMOBILE MANUFACTURERS: *Statement of Principles, Criteria and Verification Procedures on Driver Interactions with Advanced In-Vehicle Information and Communication Systems*. Juni 2006. – http://www.autoalliance.org - Aufruf: 30.03.2016

[AN03] ABEL, Heinz-Bernhard ; NOLL, Heinrich: *Head-Up-Display*. http://www.freepatentsonline.com/EP1291700A1.html. Version: März 2003. – Aufruf: 30.03.2016

[APB+99] ABRAMS, Marc ; PHANOURIOU, Constantinos ; BATONGBACAL, Alan L. ; WILLIAMS, Stephen M. ; SHUSTER, Jonathan E.: UIML: An Appliance-Independent XML User Interface Language. In: *Eighth International World Wide Web Conference*. Toronto, Canada, Mai 1999

[App13] APPLE INC.: *Apple Unveils iOS 7*. Juni 2013. – Pressemitteilung

[Aud10] AUDI AG: *Das MMI touch*. http://www.audi.de/de/brand/de/neuwagen/a8/a8/ausstattung/multi_media_interface/mmi_touch.html. Version: 2010. – Aufruf: 22.09.2010

[Aud14] AUDI AG: *Audi and AT&T offer 4G LTE connectivity starting with 2015 Audi A3 sedans*. Januar 2014. – Pressemitteilung

[BCPS04] BERTI, Silvia ; CORREANI, Francesco ; PATERNÒ, Fabio ; SANTORO, Carmen: The TERESA XML Language for the Description of Interactive Systems

at Multiple Abstraction. In: *Leveles, Proceedings Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages*, 2004, S. 103–110

[BGG86] BOYER, Claude ; GAUTHIER, Francois ; GERBE, Jean-Pierre: *Head-up display*. `http://www.google.de/patents?id=qKA5AAAAEBAJ`. Version: Juli 1986. – Aufruf: 30.03.2016

[BHS07] BUSCHMANN, Frank ; HENNEY, Kevlin ; SCHMIDT, Douglas C.: *Pattern-oriented software architecture: on patterns and pattern languages*. Bd. 5. John Wiley & Sons, 2007

[Bib] BIBLIOGRAPHISCHES INSTITUT GMBH - DUDENVERLAG: *Duden online*. `http://www.duden.de/`. – Aufruf: 21.11.2011

[BMW08] BMW: *World Debut for Unrestricted Use of the Internet: BMW ConnectedDrive Brings the World Wide Web to the Car Display in a BMW*. Februar 2008. – Pressemitteilung

[BMW13] BMW: *BMW Announces Four New App Integrations for iPhone*. März 2013. – Pressemitteilung

[Bra11] BRATZEL, Stefan: *i-Car: Die junge Generation und das vernetzte Auto.* März 2011. – www.auto-institut.de - Aufruf: 30.03.2016

[Bub03] BUBB, Heiner: Fahrerassistenz - primär ein Beitrag zum Komfort oder für die Sicherheit? In: *Der Fahrer im 21. Jahrhundert.* VDI-Berichte 1768 (2003), S. 25–46

[BWE94] BREWSTER, Stephen A. ; WRIGHT, Peter C. ; EDWARDS, Alistair D. N.: A detailed investigation into the effectiveness of earcons. In: KRAMER, Gregory (Hrsg.): *Auditory Display: sonification, audification, and auditory interfaces*. Reading, MA, USA : Addison-Wesley, 1994

[Cam88] CAMRAS, Marvin: *Magnetic recording handbook*. Springer, 1988. – 718 S.

[CAM04] CAMELEON PROJECT: *CAMELEON Project Homepage*. 2004. – http://giove.isti.cnr.it/projects/cameleon.html - Aufruf: 02.03.2014

[Car13] CAR CONNECTIVITY CONSORTIUM: *MirrorLink™ 1.0 Device Specification Now Available for Download*. Februar 2013. – Pressemitteilung

[CCT+03] CALVARY, Gaëlle ; COUTAZ, Joëlle ; THEVENIN, David ; LIMBOURG, Quentin ; BOUILLON, Laurent ; VANDERDONCKT, Jean: A Unifying Reference Framework for Multi-Target User Interfaces. In: *Interacting with Computers* 15 (2003), S. 289–308

[CE408] CE4A EG TERMINAL MODE: *Functional Mobile Device Integration, Version 0.6*. 2008. – Arbeitspapier

[Cer03] CERUZZI, Paul E.: *A history of modern computing*. 2. MIT Press, 2003. – 445 S.

[Che76] CHEN, Peter Pin-Shan: The Entity-Relationship Model - Toward a Unified View of Data. In: *ACM Transactions on Database Systems* 1 (1976), S. 9–36

[CLC04] CLERCKX, Tim ; LUYTEN, Kris ; CONINX, Karin: The mapping problem back and forth: customizing dynamic models while preserving consistency. In: *Proceedings of the 3rd annual conference on Task models and diagrams*. New York, NY, USA : ACM, 2004 (TAMODIA '04). – ISBN 1–59593–000–0, 33–42

[CMN83] CARD, Stuart K. ; MORAN, Thomas P. ; NEWELL, Allen: *The Psychology of Human-Computer Interaction*. Routledge, 1983

[Dai13a] DAIMLER AG: *Innovation made in Silicon Valley: Mercedes-Benz presents new features of its Digital DriveStyle App at Google I/O*. Mai 2013. – Pressemitteilung

[Dai13b] DAIMLER AG: *Mercedes-Benz presents new COMAND Online features at CeBIT show*. März 2013. – Pressemitteilung

[Dei11] DEIL, Manuel: *Interfaces for Abstract Interaction Models on Mobile Platforms*, Universität Ulm, Diplomarbeit, 2011

[DFAB04] DIX, Alan ; FINLAY, Janet ; ABOWD, Gregory D. ; BEALE, Russell: *Human-Computer Interaction*. Pearson Education Limited, 2004

[DFHS05] *Kapitel* Engineering Human Computer Interaction and Interactive Systems. In: DITTMAR, Anke ; FORBRIG, Peter ; HEFTBERGER, Simone ; STARY, Chris: *Support for Task Modeling - A "Constructive" Exploration*. Springer, 2005, S. 59–76

[DP10] DAUSEND, Marcel ; POGUNTKE, Mark: Spezifikation multimodaler Interaktionsanwendungen mit UML. In: *Mensch & Computer 2010: Interaktive Kulturen*, 2010, S. 215 – 224

[DP11] DAUSEND, Marcel ; POGUNTKE, Mark: Ausführbare UML-Modelle multimodaler Interaktionsanwendungen. In: *i-com - Zeitschrift für interaktive und kooperative Medien* 3 (2011), S. 33 – 39

[Eur99] EUROPEAN COMMISSION: *European Statement of Principles on HMI for in-vehicle information and communication systems*. `ftp://ftp.cordis.europa.eu/pub/telematics/docs/tap_transport/hmien.pdf`. Version: 1999. – Aufruf: 18.01.2010

[For13] FORD MOTOR COMPANY: *Ford Announces SYNC AppLink Mobile Applications Catalog and AppLink 2.0 Functionality Coming to Europe*. September 2013. – Pressemitteilung

[Gav94] GAVER, William W.: Using and creating auditory icons. In: KRAMER, Gregory (Hrsg.): *Auditory Display: sonification, audification, and auditory interfaces*. Reading, MA, USA : Addison-Wesley, 1994

[GEM94] GRAY, Phil ; ENGLAND, David ; MCGOWAN, Steve: XUAN: Enhancing UAN to Capture Temporal Relationships among Actions. In: COCKTON, Gilbert (Hrsg.) ; DRAPER, Stephen (Hrsg.) ; WEIR, George R. S. (Hrsg.): *People and Computers IX*. Cambridge University Press, 1994, S. 301 – 312

[GJVH94] GAMMA, Erich ; JOHNSON, Ralph ; VLISSIDES, John ; HELM, Richard: *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional Computing Series, 1994

[GLR+02] GERBER, Anna ; LAWLEY, Michael ; RAYMOND, Kerry ; STEEL, Jim ; WOOD, Andrew: Transformation: The Missing Link of MDA. In: CORRADINI, Andrea (Hrsg.) ; EHRIG, Hartmut (Hrsg.) ; KREOWSKI, Hans-Jörg (Hrsg.) ; ROZENBERG, Grzegorz (Hrsg.): *Graph Transformation* Bd. 2505. Springer Berlin Heidelberg, 2002. – ISBN 978–3–540–44310–0, S. 90–105

[GPR06] GRUHN, V. ; PIEPER, D. ; ROTTGERS, C.: *MDA: Effektives Software-Engineering mit UML2 und Eclipse*. Springer, 2006. – ISBN 9783540287469

[Hï1] HÜGER, Fabian: User interface transfer for driver information systems: a survey and an improved approach. In: *Proceedings of the 3rd International*

*Conference on Automotive User Interfaces and Interactive Vehicular Applications*. New York, NY, USA : ACM, 2011 (AutomotiveUI '11). – ISBN 978–1–4503–1231–8, S. 113–120

[Har87] HAREL, David: Statecharts: A visual formalism for complex systems. In: *Science of computer programming* 8 (1987), Nr. 3, S. 231–274

[Hei01] HEISTERKAMP, Paul: Linguatronic - Product-Level Speech System for Mercedes-Benz Cars. In: *HLT '01 Proceedings of the first international conference on Human language technology research*, 2001

[HGM+12] HESS, Steffen ; GROSS, Anne ; MAIER, Andreas ; ORFGEN, Marius ; MEIXNER, Gerrit: Standardizing model-based in-vehicle infotainment development in the German automotive industry. In: *Proceedings of the 4th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*. New York, NY, USA : ACM, 2012 (AutomotiveUI '12). – ISBN 978–1–4503–1751–1, 59–66

[HH05] HERMANN, Thomas ; HUNT, Andy: Guest Editor's Introduction: An Introduction to Interactive Sonification. In: *IEEE Multimedia* 12 (2005), S. 20–24

[HIR02] HENRICKSEN, Karen ; INDULSKA, Jadwiga ; RAKOTONIRAINY, Andry: Modeling Context Information in Pervasive Computing Systems. In: *Lecture Notes in Computer Science* 2414 (2002), S. 167–180

[HSH90] HARTSON, H. R. ; SIOCHI, Antonio C. ; HIX, Deborah: The UAN: A User-oriented Representation for Direct Manipulation Interface Designs. In: *ACM Trans. Inf. Syst.* 8 (1990), Juli, Nr. 3, S. 181–203

[HT09] HARRISON, Michael ; THIMBLEBY, Harold: The Role of Formal Methods in Human-Computer Interaction. In: HARRISON, Michael (Hrsg.) ; THIMBLEBY, Harold (Hrsg.): *Formal Methods in Human-Computer Interaction*. Cambridge University Press, 2009

[HW09] HUPPELSBERG, Jens ; WALTER, Kerstin: *Kurzlehrbuch Physiologie*. 3. Georg Thieme Verlag KG, 2009

[Int90] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *DIN EN ISO 9241-110. Grundsätze der Dialoggestaltung*. 1990

[ISO08] ISO/IEC: *ISO/IEC 24752: Information technology - User interfaces - Universal remote console*. 2008

[IU97] ISHII, Hiroshi ; ULLMER, Brygg: Tangible bits: towards seamless interfaces between people, bits and atoms. In: *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM, 1997. – ISBN 0–89791–802–9, S. 234–241

[Iwa08] IWATA, Hiroo: Haptic Interface. In: JACKO, Julie A. (Hrsg.) ; SEARS, Andrew (Hrsg.): *The Human-Computer Interaction Handbook*. L. Erlbaum Associates Inc., 2008

[JM10] JOKINEN, Kristiina ; MCTEAR, Michael ; MCTEAR, Michael (Hrsg.): *Spoken Dialogue Systems*. Morgan & Claypool, 2010

[KB08] KASTENS, Uwe ; BÜNING, Hans K.: *Modellierung: Grundlagen und formale Methoden*. Hanser Verlag, 2008

[KR08] KUROSE, James F. ; ROSS, Keith W.: *Computer Networking: A Top-Down Approach*. 4. Pearson Education, 2008

[Kra94] KRAMER, Gregory: An introduction to auditory display. In: KRAMER, Gregory (Hrsg.): *Auditory Display: sonification, audification, and auditory interfaces*. Reading, MA, USA : Addison-Wesley, 1994

[KS09] KERN, Dagmar ; SCHMIDT, Albrecht: Design Space for Driver-based Automotive User Interfaces. In: *Proceedings of the First International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, 2009

[LSHA07] LINK, Stefan ; SCHUSTER, Thomas ; HOYER, Philip ; ABECK, Sebastian: Modellgetriebene Entwicklung grafischer Benutzerschnittstellen. In: *i-com* 6 (2007), S. 37–43

[Luy04] LUYTEN, Kris: *Dynamic User Interface Generation for Mobile and Embedded Systems with Model-Based User Interface Development*, Transnationale Universität Limburg, Diss., 2004

[LV04] LIMBOURG, Quentin ; VANDERDONCKT, Jean: Addressing the mapping problem in user interface design with UsiXML. In: *Proceedings of the 3rd annual conference on Task models and diagrams*. New York, NY, USA : ACM, 2004 (TAMODIA '04). – ISBN 1–59593–000–0, 155–163

[MÏ12] MÜLLER, Timo: *Towards Generic Transformations of Abstract UI Models for Integration to Automotive UIs*, Universität Ulm, Diplomarbeit, 2012

[May99] MAYHEW, Deborah J.: *The usability engineering lifecycle: a practitioner's handbook for user interface design*. Morgan Kaufmann, 1999. – 542 S.

[MB04] MCGOOKIN, David K. ; BREWSTER, Stephen A.: Understanding Concurrent Earcons: Applying Auditory Scene Analysis Principles to Concurrent Earcon Recognition. In: *ACM Transactions on Applied Perceptions* 1 (2004), S. 130–155

[MBD05] MINKER, Wolfgang (Hrsg.) ; BÜHLER, Dirk (Hrsg.) ; DYBKJAER, Laila (Hrsg.): *Spoken multimodal human-computer dialogue in mobile environments*. Springer, 2005

[McT10] MCTEAR, Michael: Trends, Challenges and Opportunities in Spoken Dialogue Research. In: MINKER, Wolfgang (Hrsg.) ; LEE, Gary G. (Hrsg.) ; NAKAMURA, Satoshi (Hrsg.) ; MARIANI, Joseph (Hrsg.): *Spoken Dialogue Systems Technology and Design*. Springer, 2010, S. 135–158

[Mei10] MEIXNER, Gerrit: *Entwicklung einer modellbasierten Architektur für multimodale Benutzungsschnittstellen*. 2010

[Mel10] MELO, Guido de: *Modellbasierte Entwicklung von Interaktionsanwendungen*. Verlag Dr. Hut, 2010. – 162 S.

[MHP00] MYERS, Brad ; HUDSON, Scott E. ; PAUSCH, Randy: Past, present, and future of user interface software tools. In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 7 (2000), S. 3–28

[MHW+09] MELO, Guido de ; HONOLD, Frank ; WEBER, Michael ; POGUNTKE, Mark ; ANDRÉ: Towards a flexible UI model for automotive human-machine interaction. In: *Proceedings of the 1st International Conference on Automotive User Interfaces and Interactive Vehicular Applications*. New York, NY, USA : ACM, 2009 (AutomotiveUI '09). – ISBN 978–1–60558–571–0, 47–50

[MSB11] MEIXNER, Gerrit ; SEISSLER, Marc ; BREINER, Kai: Model-Driven Useware Engineering. In: HUSSMANN, Heinrich (Hrsg.) ; MEIXNER, Gerrit (Hrsg.) ; ZUEHLKE, Detlef (Hrsg.): *Model-Driven Development of Advanced User Interfaces* Bd. 340. Springer Berlin Heidelberg, 2011. – ISBN 978–3–642–14561–2, S. 1–26

[MT07] MEROTH, Ansgar ; TOLG, Boris: *Infotainmentsysteme im Kraftfahrzeug*. Springer, 2007

[MTUK94] MILGRAM, Paul ; TAKEMURA, Haruo ; UTSUMI, Akira ; KISHINO, Fumio: Augmented Reality: A class of displays on the reality-virtuality continuum. In: *SPIE, Telemanipulator and Telepresence Technologies* 2351-34 (1994), S. 282–292

[OMG09] OMG: *UML 2.2 Superstructure Specification*. 2009

[Pat00] PATERNÒ, Fabio: *Model-based design and evaluation of interactive applications*. Springer, 2000. – 192 S.

[PE99] PUERTA, Angel ; EISENSTEIN, Jacob: Towards a general computational framework for model-based interface development systems. In: *Knowledge-Based Systems* 12 (1999), S. 433–442

[PMM97] PATERNÒ, Fabio ; MANCINI, Cristiano ; MENICONI, Silvia: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In: *Proceedings of the IFIP TC13 International Conference on HCI*, 1997

[Pue97] PUERTA, Angel: A Model-Based Interface Development Environment. In: *IEEE Software* 14 (1997), S. 40–47

[PVn+08] PEDERIVA, Inés ; VANDERDONCKT, Jean ; NA, Sergio E. ; PANACH, Ignacio ; PASTOR, Oscar: The Beautification Process in Model-Driven Engineering of User Interfaces. In: BARANAUSKAS, Cécilia (Hrsg.) ; PALANQUE, Philippe (Hrsg.) ; ABASCAL, Julio (Hrsg.) ; BARBOSA, Simone Diniz J. (Hrsg.): *Proc. INTERACT* Bd. 4662/2008, 2008 (LNCS), S. 411–425

[Reu03] REUTHER, Achim: *useML - systematische Entwicklung von Maschinenbediensystemen mit XML.* Technische Universtität Kaiserslautern, 2003

[Ric79] RICH, Elaine: User modeling via stereotypes. In: *Cognitive Science* 3 (1979), S. 329–354

[RJB99] RUMBAUGH, James ; JACOBSON, Ivar ; BOOCH, Grady: *The Unified modeling language reference manual*. Reading, Mass. : Addison-Wesley, 1999

[Sch07a] SCHÄFER, Robbie: *Model-Based Development of Multimodal and Multi-Device User Interfaces in Context-Aware Environments*. Shaker, 2007

[Sch07b] SCHÄFER, Robbie: A Survey on Transformation Tools for Model Based User Interface Development. In: *Lecture Notes in Computer Science* 4550 (2007), S. 1178–1187

[Seb88] SEBILLOTTE, Suzanne: Hierarchical planning as method for task analysis: the example of office task analysis. In: *Behaviour & Information Technology* 7 (1988), Nr. 3, S. 275–293

[SG09] SCHIELE, Franz ; GREEN, Thomas: HCI Formalisms and Cognitive Psychology: The Case of Task-Action Grammar. In: HARRISON, Michael (Hrsg.) ; THIMBLEBY, Harold (Hrsg.): *Formal Methods in Human-Computer Interaction*. Cambridge University Press, 2009

[Son10] SONNENBERG, Jan: Service and User Interface Transfer from Nomadic Devices to Car Infotainment Systems. In: *Proceedings of the Second International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI 2010)*, 2010

[SP10] SHNEIDERMAN, Ben ; PLAISANT, Catherine: *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 5. Addison-Wesley, 2010

[Sta08] STANCIULESCU, Adrian: *A Methodology for Developing Multimodal User Interfaces of Information Systems*. Presses univ. de Louvain, 2008. – 314 S.

[SV03] SOUCHON, Nathalie ; VANDERDONCKT, Jean: A Review of XML-compliant User Interface Description Languages. In: JORGE, Joaquim A. (Hrsg.) ; NUNES, Nuno J. (Hrsg.) ; CUNHA, J. F. (Hrsg.): *Interactive Systems. Design, Specification, and Verification* Bd. 2844. Springer Berlin Heidelberg, 2003, S. 377–391

[Tan03] TANENBAUM, Andrew S.: *Computer Networks*. 4. Prentice Hall PTR, 2003

[Tra02] TRAETTEBERG, Hallvard: *Model-based User Interface Design*, Norwegian University of Science and Technology, Diss., 2002

[TVTB12] TRAN, Vi ; VANDERDONCKT, Jean ; TESORIERO, Ricardo ; BEUVENS, François: Systematic generation of abstract user interfaces. In: *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*. New York, NY, USA : ACM, 2012 (EICS '12). – ISBN 978–1–4503–1168–7, 101–110

[UI99] UNDERKOFFLER, John ; ISHII, Hiroshi: URP: A Luminous-tangible workbench for urban planning and design. In: *CHI 1999: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1999

[VLM+04] VANDERDONCKT, Jean ; LIMBOURG, Quentin ; MICHOTTE, Benjamin ; BOUILLON, Laurent ; TREVISAN, Daniela ; FLORINS, Murielle: UsiXML: a User Interface Description Language for Specifying Multimodal User Interfaces. In: *Proceedings of W3C Workshop on Multimodal Interaction WMI*, 2004, S. 1–7

[Wei91] WEISER, Mark: The Computer for the 21st Century. In: *Scientific American* 265 (1991), S. 94–104

[Wel01] WELIE, Martijn van: *Task-based user interface design*, Vrije Universiteit, Diss., 2001

[WHW09] WINNER, Hermann (Hrsg.) ; HAKULI, Stephan (Hrsg.) ; WOLF, Gabriele (Hrsg.): *Handbuch Fahrerassistenzsysteme - Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Vieweg + Teubner, 2009. – 694 S.

[Wor14] WORLD WIDE WEB CONSORTIUM: *HTML 5.1. A vocabulary and associated APIs for HTML and XHTML. W3C Working Draft 04 February 2014*. Februar 2014. – http://www.w3.org/TR/html51/ - Aufruf: 02.03.2014

[ZV07] ZIMMERMANN, Gottfried ; VANDERHEIDEN, Gregg: The Universal Control Hub: An Open Platform for Remote User Interfaces in the Digital Home. In: JACKO, Julie A. (Hrsg.): *Human-Computer Interaction. Interaction Platforms and Techniques* Bd. 4551. Springer Berlin Heidelberg, 2007, S. 1040–1049

[ZV10] ZIMMERMANN, G. ; VANDERHEIDEN, G.: A dream... The universal remote console. In: *ISO Focus+* (2010), S. 11–13

[ZWS01] ZELLER, A. ; WAGNER, A. ; SPRENG, M.: iDrive - Zentrale Bedienung im neuen 7er von BMW /iDrive - centralised operation in the new 7series of BMW. In: *Elektronik im Kraftfahrzeug. Tagung der VDI-Gesellschaft Fahrzeug- und Verkehrstechnik* VDI-Berichte 1646 (2001), S. 997–1009

**Danksagung**

Prof. Dr.-Ing. Michael Weber, vielen Dank für die wissenschaftliche Betreuung der Arbeit und den stets guten Rat über die letzten Jahre, insbesondere in der Zeit, in der meine neue Arbeitsstelle sowie die Zeitverschiebung die Betreuung sicherlich nicht immer einfach machten! Prof. Dr. Dr.-Ing. Wolfgang Minker, vielen Dank für die Übernahme des Zweitgutachtens und die damit verbundenen sehr hilfreichen weiteren Einblicke während der Arbeit. Dr. André Berton, vielen Dank für die fachliche Betreuung während meiner Zeit bei Daimler und das Interesse am Thema über die Projekte bei Daimler hinaus. Ich freue mich insbesondere, dass wir den Kontakt im Berufsalltag über die Doktorandenzeit hinaus intensiv aufrechterhalten können. Meinen Mitstreitern und Kollegen Marcel, Guido, Felix, Frank, Sandro, Hansjörg, Daniel und Sven möchte ich außerdem für die zahlreichen hilfreichen Diskussionen, Feedback und Kritik sowie für eine tolle gemeinsame Doktorandenzeit danken!

Außerdem bin ich sehr dankbar darüber, dass ich begleitend zur Entwicklung dieser Arbeit einige hochmotivierte und begabte Studenten für das Themenfeld begeistern konnte. Insbesondere sind dabei folgende studentische Abschlussarbeiten entstanden, die die Entwicklung und Evaluierung dieser Arbeit unterstützt haben:

- Paul Steinhilber, Bachelorarbeit, Kommunikation zwischen externen Infotainmentanwendungen und Fahrzeug-HMI zur Interaktion über generierte Benutzerschnittstellen, 2011, Universität Ulm.
- Manuel Deil, Diplomarbeit, Interfaces for Abstract Interaction Models on Mobile Platforms, 2011, Universität Ulm.
- Timo Müller, Diplomarbeit, Towards Generic Transformations of Abstract UI Models for Integration to Automotive UIs, 2012, Universität Ulm.

Paul, Manuel und Timo, ganz besonderen Dank an Euch für Eure hervorragenden Beiträge und die tolle Zusammenarbeit!

Zu guter Letzt danke ich meiner Partnerin, Familie und Freunden. Danke, dass ihr für mich da seid und mich in der Zeit der Anfertigung dieser Arbeit genau richtig unterstützt habt!

Mark Poguntke, 18. November 2015