Institut für Visualisierung und Interaktive Systeme
Abteilung Interaktive Systeme

Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Diplomarbeit Nr. 3020

Modellgestütztes End-User-Development für Multi-Touch-Benutzungsschnittstellen

Marc Dennis Paulenz

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Thomas Ertl
Betreuer:	Dr.-Ing. Thomas Schlegel
Begonnen am:	01. April 2010
Beendet am:	01. Oktober 2010
CR-Klassifikation:	D.2.2 [Design Tools and Techniques] - User Interfaces D.2.6 [Programming Environments] - Graphical environments, Interactive environments H.5.2 [User Interfaces] - Graphical user interfaces, Input devices and strategies, User-centered design

Inhaltsverzeichnis

Inhaltsverzeichnis **3**

Abbildungsverzeichnis **7**

Tabellenverzeichnis **11**

1. Einleitung **13**

1.1. Aktuelle Themen der Softwareentwicklung und Ziel der Arbeit 13

1.2. Aufbau der Arbeit 13

2. Grundlagen, Vorarbeiten und Stand der Technik **15**

2.1. End-User-Development 15

2.2. Modellierung von Benutzungsschnittstellen 16

2.2.1. Benutzerzentrierte Gestaltung (User-centered Design) *16*

2.2.2. Aufgaben- und Ablauf-Modellierung *17*

2.2.3. Anwendungs-Modellierung *20*

2.2.4. Benutzer-Modellierung *20*

2.2.5. Dialog-Modellierung *21*

2.2.6. Präsentations-Modellierung *21*

2.2.7. Zusammenspiel der Modelle *21*

2.3. Model-based User Interface Development Environments 22

2.3.1. UIDE *23*

2.3.2. MASTERMIND *23*

2.3.3. JANUS *23*

2.3.4. Teallach *24*

2.3.5. Weitere MB-UIDEs *24*

2.4. Multi-Touch-Benutzungsschnittstellen 25

2.4.1. Multi-Touch-Historie *25*

2.4.2. Aktuelle Multi-Touch-Technologien *26*

2.4.3. Interaktionsmethoden in Multi-Touch-Umgebungen *30*

2.4.4. Microsoft Surface Plattform *32*

2.4.5. Besonderheiten der Entwicklung von Multi-Touch- und Surface-Anwendungen *34*

3. Problemstellung und Lösungsansatz 38
3.1. Problemstellung 38
3.2. Lösungsansatz 39

4. Taxonomie für Benutzerinteraktionen 41
4.1. Grundlegende Benutzerinteraktionen 41
4.2. Werttypen 42
4.3. Eingabe 44
4.4. Ausgabe 45
4.5. Bearbeitung 45
4.6. Auswahl 45
4.7. Ausführung 46
4.8. Gesamtdarstellung der Benutzerinteraktionen 47

5. Abbildung der Benutzerinteraktionselemente auf Steuerelemente der Microsoft Surface-Umgebung 49
5.1. Eingabe 49
5.2. Ausgabe 53
5.3. Bearbeitung 54
5.4. Auswahl 55
5.4.1. Einzelauswahl 55
5.4.2. Mehrfachauswahl 57
5.4.3. Bereichsauswahl 57
5.5. Ausführung 58

6. Modell 61
6.1. Anwendungsmodell 63
6.2. Präsentationsmodell 66
6.3. Ablaufmodell 72

7. Architektur 76
7.1. Übersicht der Komponenten 76
7.2. Das ApplicationModel 78
7.3. Das PresentationModel 82
7.4. Das TaskModel 86

7.5. Die Ausführungskomponente ... 88

8. Realisierung ... 90

8.1. Grundlegende Informationen der Realisierung ... 90

8.2. Das Hauptmenü ... 90

8.3. Der Typeneditor ... 91

8.4. Der Oberflächeneditor ... 94

8.5. Der Ablaufeditor ... 98

8.6. Die Ausführung ... 99

8.6.1. Genereller Ablauf ... 100

8.6.2. Die Typerstellung ... 101

8.6.3. Die Benutzeroberflächenerstellung ... 101

8.6.4. Die Steuerung des Programmablaufs ... 103

9. Evaluation der Ergebnisse ... 105

9.1. Funktionsfähigkeit des Modells und der Implementierung ... 105

9.2. Endbenutzertauglichkeit ... 110

9.3. Multi-Touch-Unterstützung ... 111

10. Zusammenfassung und Ausblick ... 113

10.1. Zusammenfassung ... 113

10.2. Die Zukunft von *EDT4MT* ... 113

10.3. Die Zukunft des Modells ... 115

10.4. Die Zukunft von MB-UIDEs ... 115

Literaturverzeichnis ... 117

Erklärung ... 121

Abbildungsverzeichnis

Abbildung 2.1 Grundsätzlicher Aufbau von Aufgaben in TaskMODL aus [Trætteberg, 2002]. 18

Abbildung 2.2 Use-Case-Diagramm mit Visual Studio 2010 18

Abbildung 2.3 Ablauf der modell-basierten Erzeugung von Benutzungsschnittstellen aus [Schlungbaum, 1996]. 22

Abbildung 2.4 Popularität des Begriffs "multi-touch" von Anfang 2004 bis Mitte 2009 auf Basis von Google Trends aus [Acevedo Zambrano, 2009]. 26

Abbildung 2.5 Schematische Darstellung einer resistiven Touch-Oberfläche aus [Schöning, et al., 2008] 27

Abbildung 2.6 Schematische Darstellung einer oberflächenkapazitiven Touch-Oberfläche aus [Schöning, et al., 2008] 28

Abbildung 2.7 Schematischer Aufbau einer Rear DI Touch-Oberfläche aus [NUI Group, 2008] 29

Abbildung 2.8 Aufbau des Microsoft Surface Tisches aus [Surface HQ, 2009] 33

Abbildung 2.9 Softwarearchitektur der Surface-Plattform aus [Microsoft Corporation] 34

Abbildung 2.10 Verschieben eines Fotos aus [Microsoft Corporation, 2009] 35

Abbildung 2.11 Skalieren eines Fotos aus [Microsoft Corporation, 2009] 36

Abbildung 3.1 Grafische Darstellung der Bestandteile des Lösungsansatzes 40

Abbildung 4.1 Grundlegende Benutzerinteraktionen 41

Abbildung 4.2 Übersicht der verfügbaren Werttypen 43

Abbildung 4.3 Unterteilung der Benutzerinteraktion Auswahl 46

Abbildung 4.4 Übersicht der Benutzerinteraktionen 48

Abbildung 5.1 SurfaceCheckBox im Zustand "wahr" (oben) und "falsch" (unten) 49

Abbildung 5.2 SurfaceTextBox (oben) mit virtueller Bildschirmtastatur (unten) 50

Abbildung 5.3 SurfaceSlider mit Hilfsmarkierungen 51

Abbildung 5.4 SurfaceInkCanvas mit absolutem visuellen Wert 52

Abbildung 5.5 Übersicht der Eingabeelemente 53

Abbildung 5.6 Übersicht der Ausgabeelemente 54

Abbildung 5.7 SuraceListBox mit ausgewähltem Wert "Blau" 55

Abbildung 5.8 SurfaceListBox mit Scrollbalken 56

Abbildung 5.9 SurfaceRadioButtons mit ausgewähltem Wert "Blau" 56

Abbildung 5.10 Übersicht der Auswahlelemente 58
Abbildung 5.11 SurfaceButton 58
Abbildung 5.12 SurfaceMenu 59
Abbildung 5.13 ElementMenu 59
Abbildung 5.14 Übersicht der Ausführungselemente 60
Abbildung 6.1 Übersicht des Gesamtmodells 61
Abbildung 6.2 Basisdatentypen des Anwendungsmodells 64
Abbildung 6.3 Schematische Darstellung eines benutzerdefinierten Datentyps 65
Abbildung 6.4 Schematische Darstellung der Datentypen *Person* und *Geschäftsperson* .. 66
Abbildung 6.5 Übersicht der verschiedenen Layoutelemente 68
Abbildung 6.6 Beispiel eines Benutzersteuerelements zur Eingabe einer Person 72
Abbildung 6.7 Beispiel eines Ablaufmodells 74
Abbildung 7.1 Übersicht der Komponenten 76
Abbildung 7.2 Übersicht der Modell-Komponenten 77
Abbildung 7.3 Beispiel eines zusammengesetzten, benutzerdefinierten Datentyps 79
Abbildung 7.4 Beispiel von Vererbung bei benutzerdefinierten Datentypen 80
Abbildung 7.5 UML-Diagramm des Anwendungsmodells 80
Abbildung 7.6 Steuerelement zur Eingabeinteraktion für eine Person 83
Abbildung 7.7 UML-Diagramm des Präsentationsmodells 85
Abbildung 7.8 UML-Diagramm des Ablaufmodells 87
Abbildung 7.9 Grafische Darstellung eines Ablaufmodells 88
Abbildung 7.10 UML-Diagramm der Ausführungskomponente 89
Abbildung 8.1 Das Hauptmenü 91
Abbildung 8.2 Die Datentypenübersicht 92
Abbildung 8.3 Der Datentypeditor 93
Abbildung 8.4 Der Attributeditor 94
Abbildung 8.5 Der Oberflächeneditor 95
Abbildung 8.6 Bearbeitungsbereich eines Layoutelements 96
Abbildung 8.7 Elementbearbeitung eines Layoutelements 97
Abbildung 8.8 Der Ablaufeditor 98
Abbildung 8.9 UML-Sequenzdiagramm des generellen Ablaufs der Ausführung 100
Abbildung 8.10 Beispiel eines realen Interaktionselements 102

Abbildung 9.1 Erstellung des Datentyps *Address*.....106
Abbildung 9.2 Erstellung des Benutzersteuerelements *PersonInput*.....107
Abbildung 9.3 Anpassung des Benutzersteuerelements *PersonInput*.....108
Abbildung 9.4 Anpassung des Benutzersteuerelements *PersonSelection*.....108
Abbildung 9.5 Ablaufmodell der Gewinnspiel-Anwendung.....110

Tabellenverzeichnis

Tabelle 2.1 Gestenbasierte Interaktionen (Abbildungen aus [Wobbrock, et al., 2009]) .. 32

1. Einleitung

1.1. Aktuelle Themen der Softwareentwicklung und Ziel der Arbeit

Ein fundamentales Konzept [Ludewig, et al., 2007] und daher stets aktuelles Thema innerhalb der Softwareentwicklung ist die Modellierung. Die beinahe jedem Softwareentwickler bekannten Modelle reichen dabei von Datenbankmodellen wie ER-Diagrammen, über die verschiedensten Modelle der *Unified Modelling Language* bis hin zu Ablauf- und Datenflussmodellen aktueller Workflow-Systeme.

Ein weniger bekanntes Gebiet innerhalb der Modellierung besteht daraus, komplette Anwendungen inklusive Benutzern, Aufgaben, Daten, Dialogen und Benutzungsschnittstellen zu modellieren und diese anschließend auf Basis des Modells weitestgehend automatisch erzeugen zu lassen.

Neben der Modellierung ist auch immer wieder davon zu hören, wie unglücklich Endbenutzer mit den von ihnen verwendeten Anwendungen sind [Platt, 2006] und dass sie daher immer mehr dazu übergehen die Anwendungen an ihre Bedürfnisse anzupassen. Schätzungen gehen davon aus, dass die Anzahl dieser Benutzer im Jahr 2005 allein innerhalb der Vereinigten Staaten bei etwa 55 Millionen gelegen hat [Boehm, et al., 2000], was ein eindeutiges Anzeichen für die Aktualität dieses Themas ist. Vertieft man den Gedanken Anwendungen an die eigenen Bedürfnisse anzupassen, so kommt man schnell auf die Idee, dass es eine gute Alternative wäre, wenn man eine eigene Anwendung entwickeln könnte, die optimal an die eigenen Anforderungen angepasst ist, und dazu im Idealfall keinerlei Programmierkenntnisse benötigt.

Das letzte aktuelle Thema, das im Rahmen dieser Einleitung behandelt wird, ist die Multi-Touch-Technologie. Spätestens seit der Einführung von Apples iPhone im Jahr 2007 ist diese Technologie in aller Munde und erfreut sich, dank breiter Unterstützung, unter anderem durch Mobiltelefone oder verschiedene Microsoft-Plattformen, wie Surface oder Windows 7, immer größerer Beliebtheit.

Das Ziel dieser Diplomarbeit ist es diese drei genannten Themengebiete Software-Modellierung, End-User-Development und Multi-Touch miteinander zu verbinden und es dem Endbenutzer zu ermöglichen modellgestützte Anwendungen für Multi-Touch-Plattformen zu erstellen, ohne dass er dabei Erfahrungen in der Entwicklung von Anwendungen haben muss.

1.2. Aufbau der Arbeit

Um das genannte Ziel zu erreichen, werden in Kapitel 2 zunächst die benötigten Grundlagen gelegt und ein Einblick in wichtige Vorarbeiten und den aktuellen Stand

der Technik gegeben. Im Anschluss werden in Kapitel 3 die Probleme dieser aktuellen Technologien aufgezeigt und ein Ansatz zu deren Lösung beschrieben.

Basierend auf diesem Ansatz werden zunächst in Kapitel 4 typische Benutzerinteraktionen innerhalb von Softwaresystemen analysiert und in Form einer Taxonomie organisiert. Eine Abbildung der identifizierten Interaktionen auf Steuerelemente innerhalb der Microsoft Surface Plattform wird in Kapitel 5 betrachtet.

Um die aufgezeigten Schwächen aktueller Modelle zur Beschreibung von Anwendungen zu beheben, wird in Kapitel 6 ein neues Modell erstellt, das zum einen auf der Taxonomie der Benutzerinteraktionen basiert und zum anderen einen besonderen Wert auf die Endbenutzertauglichkeit legt.

Eine Software-Architektur, die als Grundlage für eine prototypische Implementierung des vorgestellten Modells verwendet werden kann und zusätzlich eine Möglichkeit bietet modellierte Anwendungen auszuführen, ist das Hauptthema in Kapitel 7. Die Realisierung dieser Architektur, sowie die Beschreibung der verschiedenen Editoren zur Bearbeitung des Modells werden im anschließenden Kapitel 8 behandelt.

Den Abschluss der Arbeit stellen eine Evaluation des Modells und der erstellten Anwendung (Kapitel 9), sowie eine Zusammenfassung der Arbeit und ein Ausblick auf Erweiterungs- und Verbesserungsmöglichkeiten (Kapitel 10) dar.

2. Grundlagen, Vorarbeiten und Stand der Technik

In diesem Kapitel wird ein Überblick über die benötigten Grundlagen dieser Arbeit gegeben. Dazu zählen die Klärung des Begriffs End-User-Development, eine Einführung in die verschiedenen Modelle zur Beschreibung von Benutzungsschnittstellen, eine Übersicht über verschiedene Werkzeuge zur modellgestützten Entwicklung von Benutzungsschnittstellen, sowie ein anschließender Überblick über den aktuellen Stand der Multi-Touch-Technik. In diesem Zusammenhang werden auch Besonderheiten dieser Technik, mit besonderem Fokus auf die Eigenheiten, Möglichkeiten und Einschränkungen der Microsoft Surface Plattform erläutert.

Weiterhin wird auf wichtige Vorarbeiten eingegangen und insbesondere deren Ergebnisse aufgezeigt und bewertet.

2.1. End-User-Development

Wie der Titel dieser Arbeit bereits zeigt, ist ein wichtiger Punkt das End-User-Development. Daher muss zunächst erklärt werden, was End-User (Endbenutzer) sind. Im Rahmen dieser Arbeit wird der End-User wie folgt definiert:

„Die End-User (Endbenutzer) einer Anwendung bezeichnet die Menge von Personen, welche die Anwendung benutzen und mit ihr umzugehen wissen."

Da dies alle Benutzer der Anwendung umfasst, lassen sich nur wenige Rückschlüsse auf Vorkenntnisse und Eigenschaften der Benutzer ziehen. So kann man lediglich davon ausgehen, dass die Benutzer die Funktionen der Anwendung kennen und diese zu benutzen wissen. Insbesondere ist beim End-User nicht davon auszugehen, dass dieser erste Erfahrungen in der Entwicklung von Anwendungen hat oder professioneller Softwareentwickler ist.

Ein Hauptunterschied zwischen professionellen Softwareentwicklern und Endbenutzer besteht in ihrem Wissen. Während Softwareentwickler über ein fundiertes Wissen im Bereich der Informatik und Softwareentwicklung verfügen, verfügen die meisten Endbenutzer nicht darüber, sondern haben, im Vergleich zu den Softwareentwicklern, einen entscheidenden Wissensvorteil in ihrer Arbeitsdomäne. Daraus lässt sich schließen, dass die Anforderungen an eine Anwendung bei den Endbenutzern wesentlich bekannter sind als bei den Softwareentwicklern, die diese Anwendung erstellen sollen und sich erst in die Arbeitsdomäne einarbeiten müssen. Eine mögliche Folge davon ist, dass sich die Endbenutzer die Anwendung so anpassen, sofern dies möglich und nötig ist, dass sie ihren Anforderungen besser entspricht. Beispiele dafür können spezielle Makrobefehle oder die Anordnungen von Schaltflächen in einer Büroanwendung sein, die dem Benutzer die Arbeit erleichtern, oder auch speziell angepasste Vorlagen für Tabellenkalkulationsprogramme. Diese Art

der Anpassung von Anwendungen durch den Endbenutzer wird End-User-Development genannt.

Eine Definition des Begriffs End-User-Development ist in [Costabile, 2002] zu finden, wo es heißt:

„*End User Development is a set of activities or techniques that allow people, who are non-professional developers, at some point to create or modify a software artifact.*"

Die Definition zeigt, dass nicht nur die genannte Anpassung von Anwendungen durch Endbenutzer beinhaltet, sondern ebenfalls die Erstellung neuer Anwendungen.

Eine Untersuchung von Boehm aus dem Jahr 2000 geht davon aus, dass im Jahr 2005 einer Zahl von 2,75 Millionen professionellen Anwendern mit Entwicklungserfahrung geschätzte 55 Millionen Endbenutzer gegenüberstehen, welche aktiv End-User-Development betreiben [Boehm, et al., 2000]. Da diese Zahl in den folgenden Jahren vermutlich weiter steigen wird, wird auch die Unterstützung von End-User-Development von immer größerer Bedeutung sein.

2.2. Modellierung von Benutzungsschnittstellen

Neben dem End-User-Development ist ein weiterer wichtiger Punkt dieser Arbeit die modellgestützte Entwicklung von Anwendungen mit Multi-Touch-Benutzungsschnittstellen. Daher erklärt dieser Abschnitt zunächst was im Rahmen dieser Arbeit unter einer Benutzungsschnittstelle zu verstehen ist, und stellt anschließend verschiedene Modelle vor, durch welche Benutzungsschnittstellen beschrieben werden können, und wie diese zueinander in Relation stehen.

2.2.1. Benutzerzentrierte Gestaltung (User-centered Design)

Zunächst muss dazu geklärt werden, was genau unter einer Benutzungsschnittstelle zu verstehen ist, die modelliert werden soll. Wie in [Trætteberg, 2002] beschrieben, kann eine mögliche Definition des Begriffs Benutzungsschnittstelle „alle direkt erfahrbaren Aspekte eines Gegenstandes oder Geräts" umfassen. Dies erschwert es allerdings bestimmte Gesichtspunkte auszublenden, da dazu beispielsweise auch Eigenschaften wie das Gewicht des Computer-Systems oder die Oberflächenbeschaffenheit der Eingabegeräte gehören. Da im Rahmen dieser Arbeit einige Eigenschaften durch die Verwendung der Microsoft Surface Plattform vorgegeben sind und nicht verändert werden können, ist es nötig diese allgemeine Definition der Benutzungsschnittstelle einzugrenzen. Ein zentraler Punkt dieser Arbeit ist die Endbenutzer-Tauglichkeit der entstehenden Software und des zugrundeliegenden Modells, weshalb es sich anbietet auch bei der Benutzungsschnittstelle eine benutzerzentrierte Sichtweise einzunehmen.

Als Grundlage dafür wird die *DIN EN ISO 13407 Benutzer-orientierte Gestaltung interaktiver Systeme* [Deutsches Institut für Normung, 2000] herangezogen, welche in

einem ihrer Kernpunkte die Aufgaben und Ziele der Benutzer, sowie deren Arbeitsabläufe und Arbeitsumgebung in das Zentrum der Betrachtung stellt. Dementsprechend werden die Themen Aufgaben- und Ablaufmodellierung auch die Kernpunkte der Modellierung von Benutzungsschnittstellen einnehmen. Bei einigen Methoden werden des Weiteren auch Benutzer-, Anwendungs-, Dialog- und Präsentationsmodelle erstellt. Die verschiedenen Modelle werden in den folgenden Abschnitten vorgestellt und die Zusammenhänge zwischen ihnen beschrieben. Ein Großteil der Erläuterungen dieses Kapitels basieren auf [Schlungbaum, 1996].

2.2.2. Aufgaben- und Ablauf-Modellierung

Ein entscheidender Aspekt der benutzerzentrierten Gestaltung ist die Erfassung der Ziele und Aufgaben der Benutzer. Dabei gibt es verschiedene Methoden, mit denen die Modellierung der Aufgaben und deren Ablauf (*task model*) realisiert werden kann. Einige dieser Modelle werden im Folgenden vorgestellt.

Modellierung mit GOMS

Das GOMS-Modell (*Goals, Operators, Methods, Selection Rules*) bietet eine auf der allgemeinen Theorie des Problemlösens basierende Methode zur Modellierung von Aufgaben. Dabei können verschiedene Aktivitäten durch die Anwendung von Operatoren modelliert werden, die durch Kontrollstrukturen zu höherwertigen Operatoren, sogenannten Methoden, kombiniert werden können [Schlegel, 2002].

Concur Task Trees (CTTs)

Die von Paternò [Paternò, 2000] entwickelte Sprache *Concur Task Trees* eignet sich zur Modellierung von Aufgaben. Basierend auf den zwei verschiedenen Aufgabentypen Interaction Tasks und Application Tasks, können mittels Vererbung, Hierarchien und temporären Operatoren Aufgabenabläufe modelliert werden. Ein grafischer Editor zur Erstellung und Validierung von CTTs ist in Form des CTTE[1] vorhanden.

TaskMODL

In seiner Arbeit [Trætteberg, 2002] stellt Trætteberg die TaskMODL (*Task Modelling Language*) vor, welche die vier Grundbestandteile von Workflows, namentlich Handlungsstruktur, Akteure, Werkzeuge und Informationen, direkt auf die Modellierung von Aufgaben überträgt [Meixner, et al., 2008]. Durch hierarchische Aufteilung in Super- und Subtasks, definierte Vor- und Nachbedingungen, sowie Operatoren für parallele, optionale oder selektive Ausführung der Aufgaben, können komplexe Aufgaben in einfachere Teilaufgaben zerlegt werden. Abbildung 2.1 zeigt den grundsätzlichen Aufbau von Aufgaben in der TaskMODL.

[1] CTTE steht unter http://giove.cnuce.cnr.it/ctte.html als Download zur Verfügung.

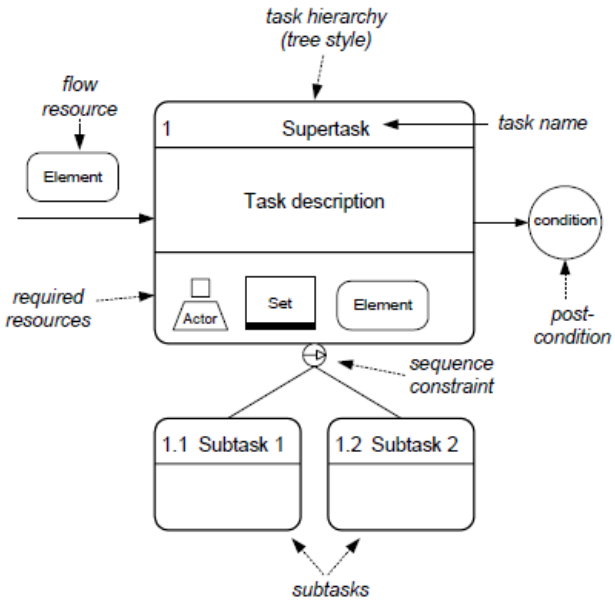


Abbildung 2.1 Grundsätzlicher Aufbau von Aufgaben in TaskMODL aus [Trætteberg, 2002].

Modellierung mit Use-Case-Diagrammen

Auch mit Hilfe der Use-Case-Diagramme der Unified Modelling Language (*UML*) ist es möglich Aufgaben zu modellieren. Mit diesen Diagrammen ist es möglich hierarchische Aufgabenstrukturen zu definieren, d.h. große Aufgaben in kleinere Teilaufgaben zu unterteilen, und mit Hilfe von Vererbungsbeziehungen Aufgaben zu erstellen, die wiederum auf anderen Aufgaben aufbauen und diese erweitern.

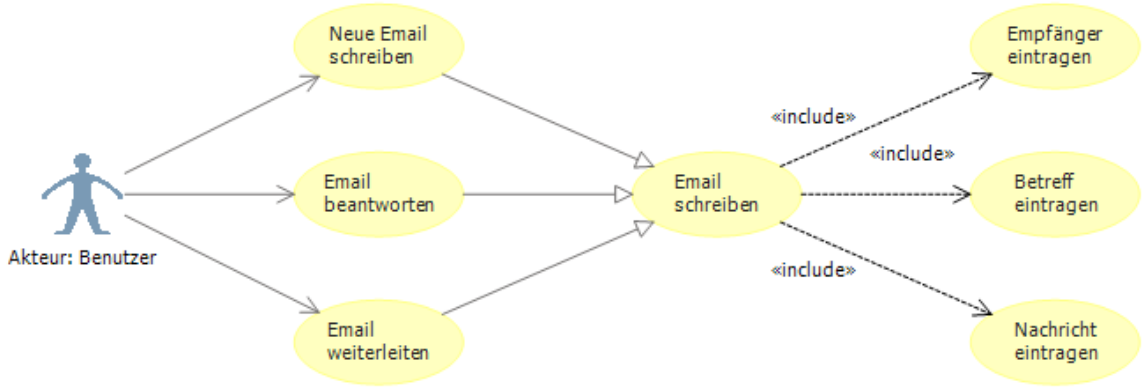


Abbildung 2.2 Use-Case-Diagramm mit Visual Studio 2010

Dabei gibt es verschiedene Relationen zwischen den Akteuren und den einzelnen Use-Cases. Diese werden im Folgenden erläutert und mit Hilfe von Abbildung 2.2 verdeutlicht:

- Assoziationen zwischen Akteuren und Use-Cases:
  Der Akteur „verwendet“ die entsprechenden Use-Cases. In der Abbildung entspricht dies den Verbindungen zwischen dem „Akteur: Benutzer“ und den drei Use-Cases „Neue Email schreiben“, „Email beantworten“ und „Email weiterleiten“.

- Generalisierung von Use-Cases:
  Mit Hilfe von Generalisierungsrelationen können verschiedene Spezial-Use-Cases zu einem generalisierten Use-Case zusammengefasst werden. In der Abbildung entspricht dies den Relationen zwischen den Spezial-Use-Cases „Neue Email schreiben“, „Email beantworten“ und „Email weiterleiten“ und der Generalisierung „Email schreiben“.
- Include-Beziehungen:
  Mittels Include-Beziehungen ist es möglich, dass Use-Cases andere Use-Cases verwenden. Dies bedeutet, dass im Verlauf des einen Use-Cases alle anderen Use-Cases ausgeführt werden, die mittels einer Include-Beziehung mit diesem in Relation stehen. In der Beispielabbildung verwendet der Use-Case „Email schreiben“ die Use-Cases „Empfänger eintragen“, „Betreff eintragen“ und „Nachricht eintragen“.
- Extend-Beziehung:
  Die Extend-Beziehung dient der Erweiterung von Use-Cases. Das bedeutet, dass ein Use-Case, der einen anderen erweitert, alle Abläufe des erweiterten Use-Cases beinhaltet und weitere Abläufe zu diesen vorhandenen hinzufügt.

Prinzipiell eignen sich die Use-Case-Diagramme der UML sehr gut dazu die grobe Struktur von Aufgaben zu modellieren. Allerdings ist es ohne Erweiterung nicht möglich die Abfolge der einzelnen Aufgaben zu modellieren, die ein Benutzer einhalten muss, um sein gewünschtes Ziel zu erreichen.

TOMBOLA

Die Modellierungssprache TOMBOLA (*Task Oriented Model Builder Object Lab*) besitzt ein „*umfangreiches Vokabular zur Spezifizierung des Aufgaben- und Datenmodells sowie eine flexible Möglichkeit zur Visualisierung des Zustands der Aufgabenausführung*“ [Uhr, 2001].

Ein wichtiges Ziel bei der Entwicklung von TOMBOLA war die Möglichkeit, dass die Benutzer, deren Aufgaben modelliert werden sollen, die erstellten Modelle überprüfen und nachvollziehen können. Daher wurde der TOMBOLA-Simulator entwickelt, der die vom Entwickler erstellten Aufgabenmodelle Schritt für Schritt ausführen kann, sodass der Benutzer in der Lage ist die Korrektheit zu evaluieren.

Auf TOMBOLA aufbauend wurde von Habbe der „*grafische Aufgabenmodell-Editor*“ (*GAME*) [Habbe, 2005] entwickelt, der eine grafische Oberfläche zur Erstellung von TOMBOLA-Modellen enthält. Dabei bietet GAME verschiedene Sichten auf das Modell:

- Graph-Sicht: Diese Sicht bietet sowohl eine freie als auch eine strukturierte Sicht auf die Objekte und deren Beziehungen untereinander.
- Hierarchie-Sicht: Visualisiert die Struktur des Modells in einer Baumstruktur
- Quelltext-Sicht: Diese zeigt den erzeugten TOMBOLA-Quelltext an und bietet die Möglichkeit diesen direkt zu bearbeiten.

Des Weiteren verfügt der Editor über eine intuitive Bedienung, eine Integration des TOMBOLA-Simulators und die Möglichkeit das Look-And-Feel der Anwendung und des Graphen zu konfigurieren.

Eigenständige Methoden zur Ablaufmodellierung

Da nicht alle vorgestellten Methoden eine Möglichkeit bieten die Abläufe der Aufgaben zu modellieren und auch einige weitere Methoden diese Funktion vermissen lassen, ist es notwendig auch die Abfolge der einzelnen Aufgaben modellieren zu können. Dazu eignen sich die folgenden Ansätze, die ebenfalls in [Schlegel, 2002] beschrieben werden. Dort können auch Verweise auf weiterführende Literatur gefunden werden.

- Data-Flow-Diagrams
- RFA
- Petrinetze
- State-Charts
- Vorgangskettendiagramme
- KSA
- ZÜD
- Event Schemata

2.2.3. Anwendungs-Modellierung

Die Modelle der Anwendungs-Modellierung (*application* oder *domain model*) ähneln den Modellen aus der klassischen objektorientierten Modellierung und enthalten allgemeine Informationen über die Anwendung, sowie einige spezifische Informationen für die Benutzungsschnittstelle.

Prinzipiell sollte das Anwendungsmodell erläutern, welche Entitäten innerhalb des Systems verwendet werden, welche Attribute und Methoden diese haben und welche Beziehungen zwischen den Entitäten bestehen. Die Anwendungsmodelle sind grob vergleichbar mit den Klassendiagrammen der UML, welche die verschiedenen Klassen, ihre Attribute und Methoden, sowie die Assoziationen zwischen den einzelnen Klassen darstellen. Erweitert wird dies häufig um spezifische Informationen der Benutzungsschnittstelle.

2.2.4. Benutzer-Modellierung

Mit Hilfe der Benutzer-Modellierung (*user model*) können die Charakteristiken verschiedener Arten von Benutzern oder Benutzergruppen repräsentiert werden. Dadurch wird es möglich verschiedene Benutzungsoberflächen für verschiedene Benutzer zur Verfügung zu stellen. Dies kann beispielsweise eine Benutzungsoberfläche für Administratoren und eine andere für normale Benutzer sein, die zum Großteil auf den gleichen Modellen basieren, aber den Benutzern jeweils andere Funktionen zur Verfügung stellen.

Dabei können die Merkmale der Benutzer in anwendungsunabhängige Eigenschaften, wie Präferenzen, Fähigkeiten und psychomotorische Qualifikation der Benutzer, sowie anwendungsabhängige Eigenschaften, wie Ziele oder Kenntnisse des Systems, eingeteilt werden.

2.2.5. Dialog-Modellierung

Bei der Dialog-Modellierung (*dialog* oder *dialogue model*) wird die Kommunikation zwischen dem Benutzer und dem System modelliert. Dabei wird einerseits beschrieben, wann der Benutzer Aktionen ausführen oder Eingaben tätigen kann, und andererseits, wann es dem System möglich ist den Benutzer nach Eingaben zu fragen und ihm Ergebnisse anzuzeigen. Generell beschreibt das Dialog-Modell also die Abfolge und Verschachtelung von Ein- und Ausgaben.

Ein Beispiel für eine Modellierungssprache für Dialoge ist die „*Dialogue Modelling Language*“ (*DiaMODL*):

Dialogue Modelling Language

Die *DiaMODL* basiert auf der „*Pisa interactor abstraction*“ und befasst sich hauptsächlich mit dem Datenfluss, der Aktivierung und der Abfolge der einzelnen Dialoge. Zudem wird sie teilweise auch für die konkrete Umsetzung der Dialoge, also die Zusammenhänge von Ein- und Ausgaben, beispielsweise über Fenster oder die Maus, eingesetzt. Eine ausführliche Beschreibung der DiaMODL kann unter [Trætteberg, 2002] eingesehen werden.

2.2.6. Präsentations-Modellierung

Das Präsentationsmodell (*presentation model*) beinhaltet alle Elemente, die dem Benutzer auf dem Bildschirm angezeigt werden, deren Anordnung und die visuellen Abhängigkeiten zwischen den Elementen. Dabei kann man zwischen statischen und dynamischen Elementen unterscheiden. Zu den statischen zählen beispielsweise Standardsteuerelemente wie Schaltflächen, Menüleisten oder Listenfelder, die sich während der Ausführung der Anwendung, mit Ausnahme des visuellen Zustands, wie Aktiviert/Deaktiviert oder Sichtbar/Unsichtbar, nicht verändern. Dem gegenüber stehen die dynamischen Elemente, die zur Darstellung von Ausgaben des Systems oder benutzerdefinierter Daten verwendet werden.

2.2.7. Zusammenspiel der Modelle

Der Ablauf bei der Generierung der Benutzeroberfläche und das Zusammenspiel der Modelle wird in [Schlungbaum, 1996] beschrieben und ist in Abbildung 2.3 sehen.

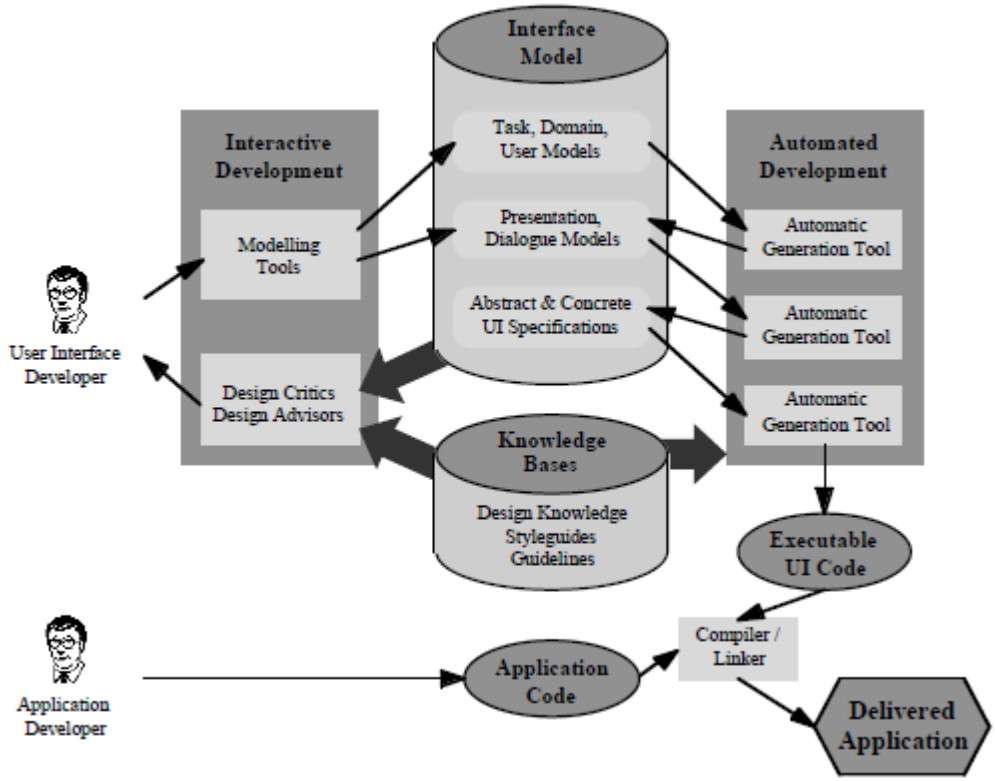


Abbildung 2.3 Ablauf der modell-basierten Erzeugung von Benutzungsschnittstellen aus [Schlungbaum, 1996].

Dabei erstellt der Benutzungsoberflächenentwickler zunächst die Modelle für die Benutzer und deren Aufgaben, sowie das Anwendungsmodell. Anschließend können in einigen Fällen auf Basis dieser Modelle das Präsentations- und das Dialog-Modell mit Hilfe von Generierungswerkzeugen automatisch erzeugt werden. Sofern möglich, kann der Entwickler die generierten Modelle anschließend anpassen. Für den Fall, dass kein Werkzeug für die automatische Generierung zur Verfügung steht, muss der Entwickler diese Modelle ebenfalls selber erstellen.

Auf dem Präsentations- und Dialog-Modell aufbauend, kann daraufhin die abstrakte und konkrete Spezifikation der Benutzungsschnittstelle generiert werden, die später zur ausführbaren Benutzungsschnittstelle transformiert werden kann.

2.3. Model-based User Interface Development Environments

Neben den einzelnen Modellen zur Beschreibung von Benutzungsschnittstellen gibt es bereits einige Werkzeuge, welche mehrere dieser Modelle zusammenfügen und optimaler Weise eine ganzheitliche Lösung zu deren Bearbeitung und Ausführung bieten. Diese Werkzeuge werden als „*Model-based User Interface Development Environments*“ (*MB-UIDE*) bezeichnet und folgen in der Regel einer der drei folgenden Möglichkeiten, um aus einem abstrakten Modell eine konkrete Benutzeroberfläche zu erzeugen:

- Direkte Interpretation des Modells
- Transformation des Modells in eine, von einem User Interface Management System (*UIMS*) interpretierbare, Form
- Direkte Erzeugung von Quellcode

Allerdings verwendet kaum eine MB-UIDE alle vorgestellten Modelle. Stattdessen wird häufig eine Teilmenge dieser Modelle verwendet, und die übrigen durch die MB-UIDE entweder nicht berücksichtigt, oder mit anderen Modellen zusammengeführt und diese als eine Art Mischform verwendet.

Im Folgenden wird eine Auswahl aktueller MB-UIDEs, deren Besonderheiten sowie die von ihnen verwendeten Modelle vorgestellt.

2.3.1. UIDE

Das „*User Interface Design Environment*“ (*UIDE*) ermöglicht es Entwicklern mit Hilfe des *Anwendungsmodells* (*application model*) und des *Benutzungsoberflächenmodells* (*interface model*) Anwendungen zu beschreiben und zu erstellen. Das Anwendungsmodell besteht aus den Aufgaben, die von den Endbenutzern des Systems ausgeführt werden, deren operativen Einschränkungen und den Datenobjekten, auf deren Basis die Aufgaben ausgeführt werden. Das Benutzungsoberflächenmodell dagegen enthält die einzelnen Komponenten der Benutzungsschnittstelle, die von der Anwendungslogik unabhängigen Aufgaben innerhalb der Benutzungsoberfläche und wiederum deren operative Einschränkungen. Weitere Inforationen zum UIDE können den Quellen [Schlegel, 2002] und [Schlungbaum, 1996] entnommen werden.

2.3.2. MASTERMIND

Beim MASTERMIND-Ansatz [Browne, et al., 1996] erstellt der Benutzungsschnittstellenentwickler neben dem Aufgaben- und Anwendungsmodell auch ein Präsentationsmodell. Das Anwendungsmodell wird auf Basis der *CORBA Interface Definition Language* (*IDL*) [Object Management Group, Inc., 2008] erstellt. Das Aufgabenmodell beschreibt in einer hierarchischen Struktur die Aufgaben des Endbenutzers, sowie die benötigten Angaben zur Steuerung der Benutzeroberfläche. Das Präsentationsmodell beschreibt den Aufbau der Benutzungsoberfläche und bietet die Möglichkeit die Anzeige bei Änderungen der zugrundeliegenden Daten automatisch zu aktualisieren.

2.3.3. JANUS

JANUS basiert auf einem erweiterten Anwendungsmodell, wie es auch zur Modellierung objektorientierter Anwendungen verwendet wird [Schlungbaum, et al., 1996]. Neben diesem werden keine weiteren Modelle verwendet. Zur Erstellung der Benutzungsschnittstellen verwendet JANUS mehrere umfangreiche Wissensdatenbanken, welche beispielsweise Styleguides und Erfahrungswerte enthalten. Zudem bietet JANUS die Möglichkeit den Quellcode der erzeugten Benutzungsschnittstelle für

mehrere Zielplattformen zu erstellen. Dies kann zum Beispiel C++-Code sein, oder auch Eingaben für UIMS.

2.3.4. Teallach

Teallach [Barclay, et al., 1999] verwendet als Ausgangssituation je ein Aufgaben-, Anwendungs- und Präsentationsmodell. Dabei wird beim Präsentationsmodell zwischen den folgenden zwei Objekttypen unterschieden:

- Abstrakte Interaktionsobjekte (abstract interaction objects = AIO)
- Konkrete Interaktionsobjekte (concrete interaction objects = CIO)

Die AIOs werden dabei auf die drei Interaktionstypen Inputter, Displayer und Editor zum Eingeben, Anzeigen und Bearbeiten von Daten abgebildet. Die CIOs bieten konkrete Realisierungen von Interaktionstypen, die später auch in dieser Form in der erzeugten Benutzungsschnittstelle verwendet werden.

Die generierten Benutzungsschnittstellen basieren auf der MVC-Architektur (model-view-controller [Krasner, et al., 1988]), wobei die verschiedenen AIOs je nach Interaktionstyp auf bestimmte Elemente der MVC-Architektur abgebildet werden.

2.3.5. Weitere MB-UIDEs

Neben den hier vorgestellten MB-UIDEs gibt es einige weitere, die im Folgenden kurz beschrieben werden:

- ADEPT [Markopoulos, et al., 1992]: Die Aufgaben, die der Benutzer mit dem System ausführen soll, werden beschrieben und iterativ erweitert und verfeinert. Das entsprechende Aufgabenmodell wird in der Spezifikationssprache TKS[2] definiert.
- DON [Kim, et al., 1990]: Basiert auf dem wissensbasierten UIDE und unterstützt den Entwurf formularbasierter Anwendungen mit Hilfe des „Organization Manager“ und „Presentation Manager“.
- FUSE [Lonczewski, et al., 1996]: Verwendet Aufgaben-, Anwendungs-, Benutzer und Dialog-Modelle. Die Dialog-Modelle werden auf Basis der anderen Modelle generiert und können anschließend vom Benutzungsschnittstellenentwickler bearbeitet werden. Als besonderes Merkmal von FUSE ist zu nennen, dass es eine automatische Generierung von Bedienungsanleitungen bietet.
- GENIUS [Janssen, et al., 1993]: Ermöglicht die Erzeugung von Prototypen für datenbankorientierte Anwendungen. Dabei verwendet es als Ausgangspunkt das vorhandene Datenmodell der Anwendung, das ein erweitertes Entity-Relationship-Modell ist. Das Produkt von GENIUS ist eine textuelle Beschrei-

[2] TKS: Task Knowledge Structure, siehe [Johnson, et al., 1988].

bung der Benutzungsschnittstelle, die als Eingabe für das UIMS ISA verwendet werden kann.

Für die folgenden MB-UIDEs können weitere Informationen unter [Schlungbaum, 1996] und [Schlegel, 2002] gefunden werden:

- HUMANOID
- ITS
- MECANO
- Mobi-D
- SCON
- TADEUS
- TRIDENT

2.4. Multi-Touch-Benutzungsschnittstellen

Neben den bereits genannten Themen End-User-Development, Modellierung der Benutzungsschnittstelle und deren Endbenutzer-orientierten Gestaltung, spielt die Multi-Touch-Technologie, insbesondere die Microsoft Surface Plattform, eine entscheidende Rolle in dieser Arbeit. Um diese verstehen, sowie die Vor- und Nachteile bewerten zu können, wird in diesem Abschnitt zunächst die Multi-Touch-Technologie im Allgemeinen und ihre Historie vorgestellt und anschließend die Besonderheiten und Eigenschaften der Microsoft Surface Plattform, sowie die Entwicklung von Anwendungen auf deren Basis näher betrachtet.

2.4.1. Multi-Touch-Historie

Um einen Einstiegspunkt in die Grundlagen der Multi-Touch-Benutzungsschnittstellen zu bekommen, ist es zunächst nötig den Begriff *Multi-Touch* zu definieren. Innerhalb dieser Arbeit gilt die folgende Definition:

Multi-Touch ist eine Menge von Interaktionstechniken, die es dem Benutzer ermöglicht graphische Benutzungsschnittstellen direkt und mit Hilfe mehrerer Finger gleichzeitig zu bedienen.

Der wesentliche Unterschied im Vergleich zu Single-Touch-Technologien besteht also darin, dass dem Benutzer nicht nur ein Berührungspunkt, sondern mehrere Berührungspunkte bis hin zu ganzen Berührungsflächen für Eingaben zur Verfügung stehen. Dies ermöglicht zudem die gleichzeitige Verwendung des Geräts durch mehrere Benutzer, was als Multi-User-Betrieb bezeichnet wird.

Obwohl die Entwicklung von Single-Touch-Geräten bereits in den 1970er Jahren erste Ergebnisse hervorbrachte und im Jahr 1982 im Rahmen einer Masterarbeit an der Universität von Toronto das erste Multi-Touch-Eingabegerät für die Mensch-Maschinen-Interaktion vorgestellt wurde [Mehta, 1982], führte diese Technologie, nicht zuletzt wegen der hohen Fertigungskosten, lange ein Schattendasein. Dieser Umstand wurde im Jahr 2005 durch Han und sein Paper „Low-Cost Multi-Touch

Sensing through Frustrated Total Internal Reflection“ [Han, 2005] gelöst. Im Februar 2006, als Han auf der TED-Konferenz[3] sein neuartiges Konzept öffentlich vorstellte [Han, 2006] und unter anderem als Videomitschnitt auf Youtube veröffentlichte, wurde Multi-Touch auch von der Öffentlichkeit wahrgenommen. Seit diesem Zeitpunkt gibt es eine rasante Entwicklung im Multi-Touch-Bereich, die nicht zuletzt dank der Unterstützung großer Unternehmen, wie beispielsweise Apple mit dem iPhone und Microsoft mit der Surface Plattform, so verlaufen konnte. Acevedo Zambrano zeigt in seiner Masterarbeit [Acevedo Zambrano, 2009] eine grafische Analyse des Suchbegriffs „multi-touch“ bei Google, die mit Hilfe von Google Trends erstellt wurde, und in Abbildung 2.4 zu sehen ist. Dabei sind die genannten Ereignisse und Produktpräsentationen hervorgehoben. Es ist deutlich zu erkennen, dass das öffentliche Interesse an Multi-Touch mit der Präsentation von Han zum ersten Mal geweckt und seitdem, durch weitere Vorstellungen in diesem Bereich, aufrechterhalten wurde.

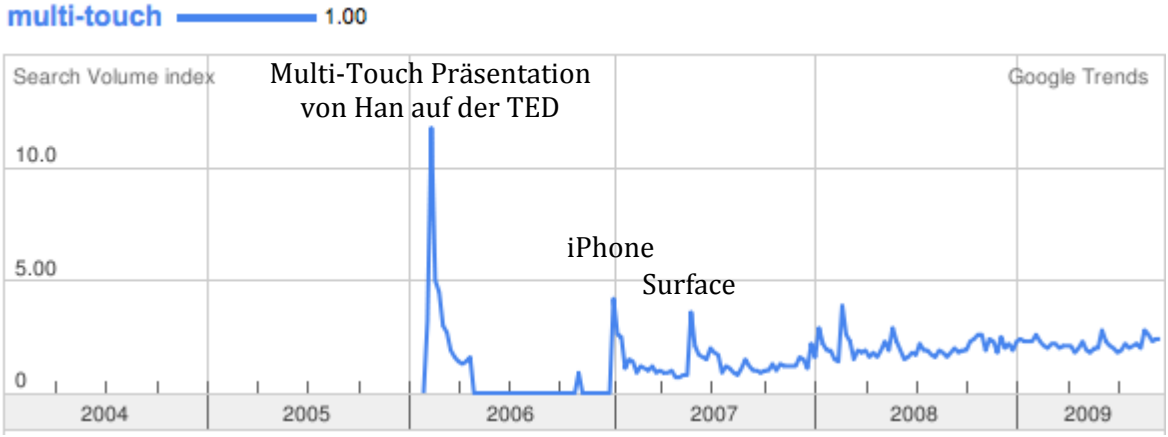


Abbildung 2.4 Popularität des Begriffs "multi-touch" von Anfang 2004 bis Mitte 2009 auf Basis von Google Trends aus [Acevedo Zambrano, 2009].

2.4.2. Aktuelle Multi-Touch-Technologien

In diesem Abschnitt werden zunächst die verschiedenen Technologien beschrieben, die aktuell in Touch- und Multi-Touch-Geräten verwendet werden. Prinzipiell können diese in die folgenden vier Kategorien eingeteilt werden, die in [Schöning, et al., 2008] und [Trümper, 2007] genannt werden:

- Resistive Touch-Oberflächen (Resistance based touch surfaces)
- Kapazitive Touch-Oberflächen (Capacitance based touch surfaces)
- Akustische Touch-Oberflächen (Acoustic based touch surfaces)
- Optische Oberflächen (Optical based touch surfaces)

[3] TED = Technology, Entertainment, Design. Seit 1984 jährlich stattfindende Konferenz in Kalifornien mit mittlerweile weiteren Ablegern weltweit.

Diese verschiedenen Technologien werden im Folgenden vorgestellt und ihre Vor- und Nachteile erläutert. Weitere Informationen zu den einzelnen Technologien können auch der Webseite unter [NUI Group, 2009] entnommen werden.

Resistive Touch-Oberflächen

Resistive Oberflächen bestehen aus zwei leitenden Schichten, die mit Substanzen wie Indiumzinnoxid bedeckt und durch eine isolierende Schicht aus Luft oder einem Silikon-Raster voneinander getrennt werden. Die Vorderseite des Geräts besteht dabei in der Regel aus einem flexiblen Material, die Rückseite aus einem Glassubstrat. Sobald der Benutzer einen Punkt berührt, werden die zwei leitenden Schichten verbunden und mit Hilfe der so entstehenden Spannung kann die Position des Berührpunktes ausgewertet werden. Eine schematische Darstellung einer resistiven Touch-Oberfläche ist in Abbildung 2.5 zu sehen [Schöning, et al., 2008]. Klare Vorteile der resistiven Touch-Oberflächen sind ihre verhältnismäßig geringen Herstellungskosten, ihr geringer Platzbedarf und ihre Haltbarkeit im alltäglichen Einsatz, weshalb sie hauptsächlich in mobilen Geräten wie PDAs, Spielekonsolen und Smartphones eingesetzt werden.

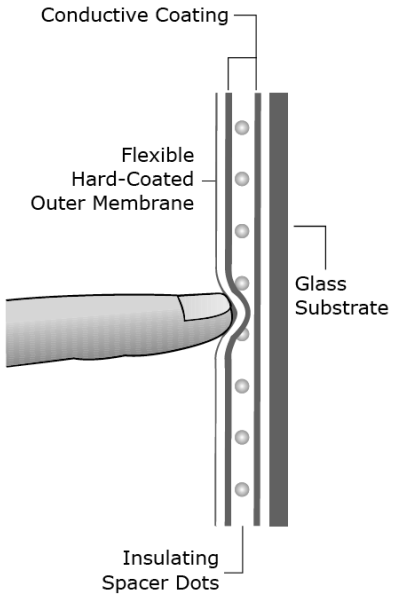


Abbildung 2.5 Schematische Darstellung einer resistiven Touch-Oberfläche aus [Schöning, et al., 2008]

Neben der verhältnismäßig schlechten Transparenz von 75%-85% der resistiven Touch-Oberflächen ist ihr größter Nachteil die Empfindlichkeit gegenüber spitzen und scharfkantigen Gegenständen.

Weitere Informationen über die Funktionsweise resistiver Touch-Oberflächen können dem Paper unter [Downs, 2005] entnommen werden.

Kapazitive Touch-Oberflächen

Kapazitive Touch-Oberflächen lassen sich prinzipiell in zwei Kategorien einteilen:

- Oberflächenkapazitanz (Surface Capacitance)
- Projektierte Kapazitanz (Projected Capacitance)

Obwohl beide Technologien ehemals für Single-Touch-Oberflächen entwickelt wurden, ist es möglich sie auch für Multi-Touch-Oberflächen zu verwenden. Dabei wird die maximale Anzahl gleichzeitiger Kontakte in der Regel durch den verwendeten Controller oder die Firmware beschränkt. Vorteile beider Technologien ist die hohe Transparenz der Anzeigegeräte, sowie die hohe Haltbarkeit und Zuverlässigkeit, weshalb sie häufig in rauen Umgebungen wie öffentlichen Anzeigegeräten oder Industrieanlagen, aber auch in Smartphones wie dem iPhone oder verschiedenen Tablet-PCs eingesetzt werden. Allerdings sind sie, besonders im Vergleich zu resistiven Touch-Oberflächen, sehr teuer in der Herstellung.

Geräte, die nach dem Prinzip der Oberflächenkapazitanz arbeiten, bestehen aus einer leitenden Schicht auf einer Glasplatte. Auf jeder Seite des Anzeigegeräts wird innerhalb dieser leitenden Schicht mit Hilfe von Elektroden ein uniformes elektrisches Feld erzeugt. Durch eine Berührung der Oberfläche mit einem Finger oder einem anderen leitenden Gegenstand, wird das elektrische Feld verändert und ein Mikroprozessor kann, aufgrund dieser Änderungen, die genaue Position der Berührung berechnen.

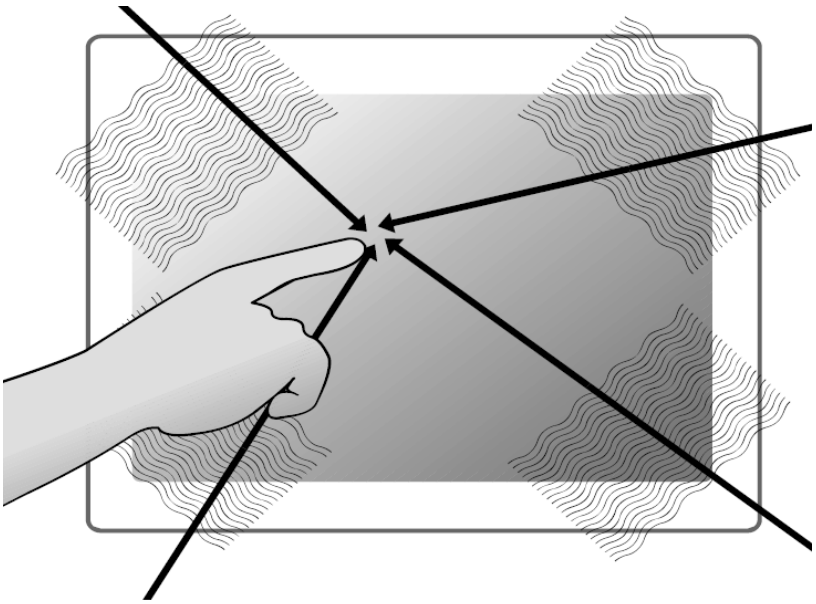


Abbildung 2.6 Schematische Darstellung einer oberflächenkapazitiven Touch-Oberfläche aus [Schöning, et al., 2008]

Im Gegensatz dazu verwenden Touch-Oberflächen, die nach dem Prinzip der projektierten Kapazitanz arbeiten, zwei voneinander isolierte, zueinander orthogonale Ebenen aus leitfähigen Mikrodrähten. Dabei liegt an einer der Ebenen eine Spannung an, die andere Ebene dient als Sensornetz. Sobald nun die Oberfläche berührt wird, ändert sich die elektrische Kapazität innerhalb der Sensorebene und die Berührungsposition kann berechnet werden. Im Gegensatz zu den oberflächen-

kapazitiven Oberflächen, lässt sich die Erkennung mehrerer Kontakte mit diesem Prinzip einfacher realisieren.

Akustische Touch-Oberflächen

Bei den akustischen Touch-Oberflächen wird die Position des Berührpunktes mit Hilfe von Mikrophonen an den Rändern der Projektionsfläche bestimmt. Aufgrund der Laufzeitunterschiede einer Berührung zu den einzelnen Mikrophonen kann ein Signalprozessor die genaue Position bestimmen.

Ein klarer Nachteil dieser Technik ist allerdings, dass die Position nur bestimmt werden kann, wenn sie von den Mikrophonen wahrgenommen werden kann, das heißt, dass einfache Berührungen nicht erkannt werden, sondern nur Klopf-Interaktionen, und daher auch Drag & Drop-Aktionen oder Gesten nicht möglich sind, da eine Verfolgung der Kontakte nicht unterstützt wird. Allerdings können verschiedene Klopfarten unterschieden werden, beispielsweise Klopfen mit der Faust, dem Fingerknochen oder metallischen Gegenständen.

Optische Touch-Oberflächen

Bei optischen Touch-Oberflächen wird mit Hilfe mehrerer Kameras ein Bild der Arbeitsfläche aufgenommen und anschließend von einem Bildverarbeitungssystem in Echtzeit ausgewertet, welches als Ergebnis zweidimensionale Koordinaten der Berührpunkte liefert. Häufig werden für die Aufnahmen Infrarotkameras verwendet und die Arbeitsfläche von Infrarot-Illuminatoren beleuchtet. Im Folgenden wird die Technik der diffusen Hintergrundbeleuchtung (Rear Diffuse Illumination = Rear DI) als Vertreter der optischen Touch-Oberflächen betrachtet.

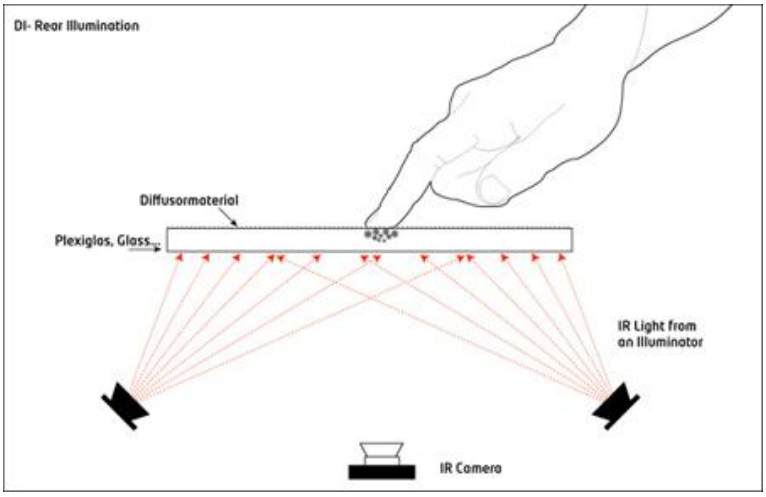


Abbildung 2.7 Schematischer Aufbau einer Rear DI Touch-Oberfläche aus [NUI Group, 2008]

Dabei werden ein Projektor, eine oder mehrere Infrarotkameras, sowie die Infrarotbeleuchtung hinter einer Projektionsoberfläche platziert. Der schematische Aufbau einer Rear DI Touch-Oberfläche ist in Abbildung 2.7 zu sehen. Dadurch wird der Bereich oberhalb der Oberfläche für die Infrarotkameras leicht erhellt und die Kameras

können somit alle Objekte anhand des reflektierten Infrarotlichts erkennen, die sich innerhalb dieses Bereichs befinden. Zudem ermöglicht dies auch die Erkennung von Objekten, die sich knapp oberhalb der Oberfläche befinden, wobei diese Objekte unschärfer werden, je weiter entfernt sie sich von der Oberfläche befinden. Neben Fingern werden allgemeine Objekte erkannt, die anhand ihrer Form oder bestimmter Bezugspunkte an ihrer Unterseite identifiziert werden können, wie es beispielsweise in [Costanza, et al., 2003] beschrieben ist.

Neben dieser vorgestellten Technik gibt es einige weitere auf optischer Basis, wie beispielsweise Front Diffuse Illumination (Front DI), Frustrated Total Internal Reflection (FTIR) oder Diffuse Surface Illumination (DSI), zu denen weitere Informationen den Arbeiten [Schöning, et al., 2008] und [Trümper, 2007] entnommen werden können.

2.4.3. Interaktionsmethoden in Multi-Touch-Umgebungen

Das Ziel dieses Abschnitts ist es, die grundlegenden Interaktionsmethoden in Multi-Touch-Umgebungen vorzustellen und dabei auf Punkten hinzuweisen, in denen sie sich im Vergleich zu traditionellen Endbenutzer-Computersystemen unterscheiden.

Neben einigen wenigen Spezialgebieten, die auf besondere Eingabegeräte wie Joysticks, Trackballs oder Grafiktabletts ausgerichtet sind, stellen Kombinationen aus einer Maus und einer Tastatur den Großteil der vom Endbenutzer verwendeten Geräte dar. Über die Tastatur kann der Benutzer schnell Daten eingeben oder mittels Hotkeys und Sondertasten spezielle Aktionen, wie zum Beispiel das Speichern eines Dokuments oder das Abspielen von Musik, auslösen. Die Maus dient hauptsächlich der Navigation zwischen einzelnen Anwendungen und Eingabebereichen, kann aber, sofern mehrere Tasten vorhanden sind, auch für Sonderaktionen verwendet werden, wie zum Beispiel zum Aufruf von Kontextmenüs, die bei einem Rechtsklick erscheinen.

Im Gegensatz dazu werden die meisten Multi-Touch-Oberflächen ausschließlich mit den Fingern und ohne Zuhilfenahme externer Geräte bedient. Da man daher weder über eine externe Tastatur zur Eingabe von Texten, noch über die erweiterten Eingabemöglichkeiten der Maus, wie beispielsweise den genannten Rechtsklick, verfügt, scheint dies die zur Verfügung stehenden Interaktionsmöglichkeiten zunächst einzuschränken. Prinzipiell sind mit den Fingern lediglich die folgenden drei Arten von Interaktionen möglich:

- Tipp- / Klopf-Interaktionen (Taps)
- Halte-Interaktionen (Holds)
- Bewegungs-Interaktionen (Movements)

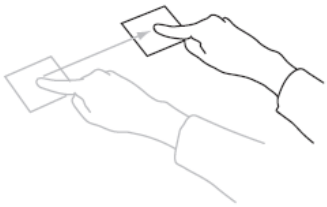
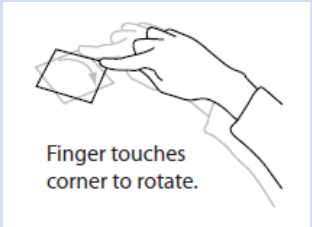
Obwohl dies, wie gesagt, zunächst nach einer Einschränkung klingt, eröffnet es gleichzeitig vollkommen neue Möglichkeiten intuitiv bedienbare, „natürliche“ Benutzeroberflächen (Natural User Interfaces = NUIs) zu Erstellen. Ziel dieser NUIs ist

es, die reale Welt und ihre physikalischen Gesetze soweit es geht auf die Benutzeroberfläche zu übertragen und dem Benutzer die Möglichkeit zu bieten die dargestellten Objekte direkt manipulieren zu können, wodurch eine möglichst natürliche Interaktion ermöglicht werden soll.

Um diese direkte Manipulation von Objekten zu ermöglichen, setzen die meisten Multi-Touch-Oberflächen, wie Apples iPhone oder Microsofts Surface auf eine Gestensteuerung. Dabei haben sich einige Gesten als Quasi-Standard herausgebildet, die sowohl intuitiv zu verstehen, als auch einfach auszuführen sind. Diese werden in Tabelle 2.1 aufgeführt und beschrieben.

Weitere Informationen zu Gesten, sowie Untersuchungen über deren Akzeptanz bei den Benutzern, sind unter [Wobbrock, et al., 2009] zu finden. Eine Übersicht der Gesten, die innerhalb der Surface-Plattform automatisch unterstützt werden, kann unter [Microsoft Corporation, 2009] eingesehen werden.

Neben der Interaktion mit Hilfe von Gesten, bieten einige Multi-Touch-Oberflächen, wie beispielsweise Rear DI (siehe Abschnitt 2.4.2), zusätzlich die Möglichkeit über reale Objekte, die auf die Oberfläche gelegt werden, mit dem System zu interagieren, womit die Verzahnung der realen mit der virtuellen Welt weiter erhöht wird.

Interaktion	Beschreibung	Schematische Darstellung
• **Auswählen**	Antippen des auszuwählenden Objekts	
• **Verschieben**	Zu verschiebendes Objekt berühren und den Finger an die gewünschte Position bewegen, während er weiterhin die Oberfläche berührt.	
• **Rotieren**	Zu rotierendes Objekt an einer Ecke berühren und an dieser in die gewünschte Richtung drehen.	

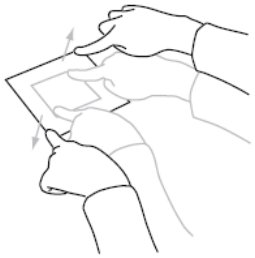
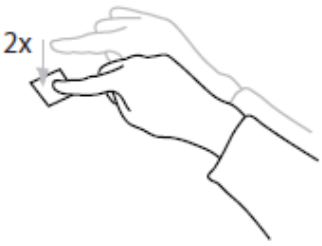
• **Skalieren**	Objekt an zwei Punkten berühren und die Finger zusammen (verkleinern) oder auseinander (vergrößern) bewegen.	
• **Öffnen**	Doppeltes antippen des zu öffnenden Objekts	

Tabelle 2.1 Gestenbasierte Interaktionen (Abbildungen aus [Wobbrock, et al., 2009])

2.4.4. Microsoft Surface Plattform

In diesem Abschnitt wird eine grundlegende Einführung in die Microsoft Surface Plattform gegeben. Dies beinhaltet neben einer kurzen Beschreibung der Entstehungsgeschichte hauptsächlich die verwendete Multi-Touch-Technik, sowie eine Übersicht der bereitgestellten Entwicklungsumgebung.

Die Konzeption eines „interaktiven Tisches“ wurde im Jahr 2001 von den Microsoft Mitarbeitern Steven Bathiche und Andy Wilson begonnen und zwei Jahre später Bill Gates, dem damaligen Vorstandvorsitzenden der Microsoft Corporation, vorgestellt, woraufhin das Entwicklungsteam vergrößert wurde. Nach über 80 Prototypen gelangte das Produkt *Microsoft Surface* 2008 zur Marktreife und wird seitdem weltweit vermarktet. Nachdem die für Entwicklung benötigten Werkzeuge zunächst nur an Käufer eines Surface-Tisches geliefert wurden, ist es mittlerweile möglich diese kostenfrei runterzuladen und frei zu verwenden.

Technische Basis

Die Microsoft Surface Plattform basiert auf der Rear DI Technik, die im Abschnitt 2.4.2 beschrieben wurde. Der grundlegende Aufbau des Microsoft Surface Tisches ist in Abbildung 2.8 zu sehen. Die vier entscheidenden Bestandteile des Surface sind in der Abbildung mit Ziffern hervorgehoben. Neben der Abdeckung aus Plexiglas (1) sorgt eine LED-Leuchte (2) für eine gleichmäßige Hintergrundbeleuchtung der Oberfläche, die mit 850nm Wellenlänge nahe dem Infrarotbereich des Lichts ist. Die Reflexionen werden von fünf Kameras (3) aufgenommen, die eine kombinierte Auflösung von 1280 mal 960 Pixeln erreichen. Die Projektion des Bildes übernimmt ein DLP-Projektor (4) mit einer Auflösung von 1024 mal 768 Pixeln bei einer sichtbaren Diagonale von 30 Zoll. Zudem befindet sich ein handelsübliches Computersystem im Tisch, auf welchem das Betriebssystem, die Surface-Umgebung an sich und die ein-

zelnen Surface-Applikationen ausgeführt werden. Die Kernkomponenten des Computersystems sind:

- Intel Core 2 Duo CPU mit 2,13 GHz
- ATI X1650 GPU mit 256 MB VRAM
- 2048 MB RAM
- 250 GB Festplattenkapazität

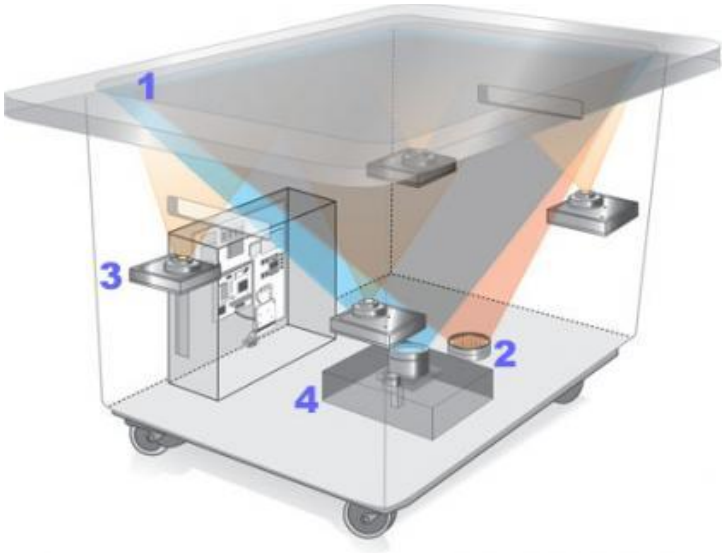


Abbildung 2.8 Aufbau des Microsoft Surface Tisches aus [Surface HQ, 2009]

Softwarebasis

Auf der im letzten Abschnitt beschriebenen Hardware des Surface-Tisches, setzen eine Reihe von Softwareschichten auf, die in Abbildung 2.9 zu sehen sind.

Das zugrundeliegende Betriebssystem der Surface-Plattform ist ein handelsübliches Windows Vista in der Business-Ausführung und bietet die grundlegenden Funktionalitäten für Administration, Sicherheit und die Verwaltung der Verzeichnisstruktur. Das „Vision System" verarbeitet die Rohdaten der Infrarotkameras und konvertiert sie in Daten, auf die über das Surface Software Development Kit (Surface SDK) zugegriffen werden kann. Das Surface SDK besteht aus den beiden Teilen „Presentation" und „Core". Die *Presentation*-Schicht dient dabei der Integration mit der Windows Presentation Foundation[4] [Marquardt, 2007] und beinhaltet einige Controls, die für die Verwendung auf der Surface-Plattform angepasst wurden. Die *Core*-Schicht hingegen kann mit beinahe jedem UI-Framework verwendet werden, das auf HWND basiert. Die Beispielanwendungen und Projektvorlagen des Surface-SDK, die auf der

[4] Die Windows Presentation Foundation (WPF) wurde mit der Version 3.0 des .NET Frameworks eingeführt und bietet neben einem vektorbasierten Rendering von Benutzeroberflächen auch eine effektive Trennung zwischen dem Aussehen der Anwendung und der Geschäftslogik.

Core-Schicht basieren, setzen dabei auf der Microsoft XNA[5] Entwicklungsplattform auf. Innerhalb einer Surface-Anwendung kann dabei nur eines der beiden Teile verwendet werden.

Die „Surface Shell“ ist für die Verwaltung der Surface-Anwendungen, der einzelnen Fenster, sowie deren Ausrichtung auf dem Tisch, und der Benutzersitzungen verantwortlich.

Die „Windows Integration“ dient der Verbindung des Windows-Betriebssystems und der Surface-Shell. Dies ist beispielsweise notwendig, wenn vom Benutzermodus in den administrativen Modus gewechselt werden soll, oder wenn innerhalb einer Surface-Anwendung eine Ausnahme ausgelöst wird.

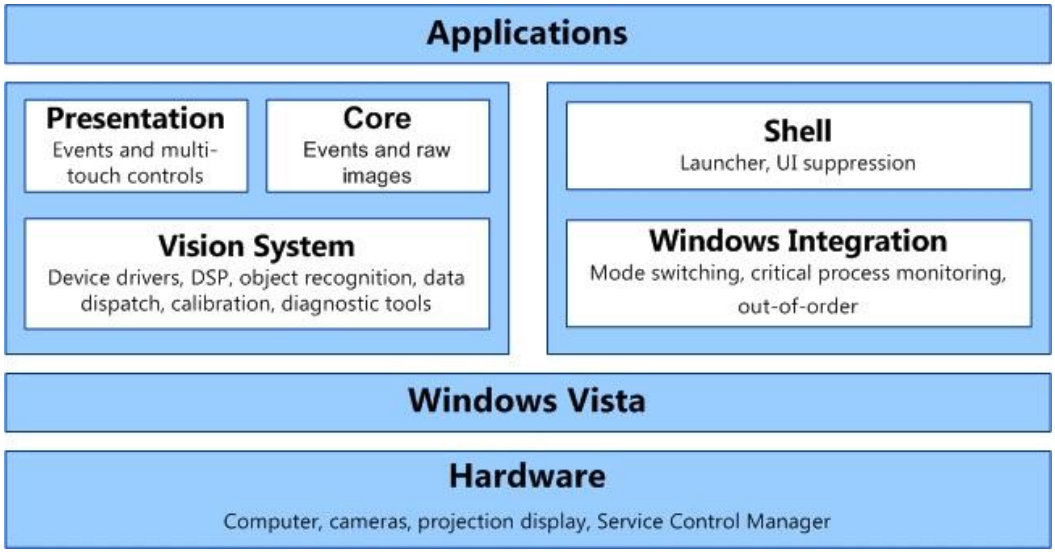


Abbildung 2.9 Softwarearchitektur der Surface-Plattform aus [Microsoft Corporation]

2.4.5. Besonderheiten der Entwicklung von Multi-Touch- und Surface-Anwendungen

Im Rahmen der „Microsoft Surface User Experience Guidelines“ [Microsoft Corporation, 2009] wurde von Microsoft eine Übersicht veröffentlicht, in welcher Besonderheiten beschrieben werden, die während der Entwicklung von Surface-Anwendungen beachtet werden sollten. Diese Empfehlungen werden in diesem Abschnitt erläutert.

Nahtlosigkeit

Anwendungen, sie auf der Surface-Plattform ausgeführt werden, sollen für den Benutzer ein nahtloses Erlebnis darstellen, in das er sowohl geistig als auch emotional eintauchen kann. Dazu ist es nötig die natürliche, skeptische Wahrnehmung des Be-

[5] XNA ist eine plattformübergreifende Programmierschnittstelle zur Spiele- und 3D-Programmierung auf Basis von C#, Direct3D und DirectX.

nutzers zu beeinflussen und seine Bereitschaft auch unnatürliche oder sogar unmögliche Sachen als Wahrheit anzuerkennen zu unterstützen. Um dies zu erreichen, ahmen Surface-Anwendungen Objekte der realen Welt nach und erweitern sie um Möglichkeiten, die nur in der virtuellen Welt möglich sind.

Geselligkeit

Standardanwendungen erzeugen aufgrund ihrer Ein- und Ausgabemethoden soziale Barrieren. Die meisten Anwendungen und Systeme sind nur für einzelne Personen geeignet, da sie nur über eine einzelne Maus und Tastatur und in der Regel über keinen (Multi-)Touchscreen verfügen, wodurch eine gleichzeitige Verwendung durch mehrere Benutzer erschwert wird. Im Gegensatz dazu ermöglichen es Anwendungen auf Basis der Microsoft Surface Plattform mehreren Benutzern gleichzeitig mit dem System zu interagieren. Neben der gleichzeitigen Interaktion mit dem System an sich, sollen auch Interaktionen zwischen den Benutzern gefördert werden, die beispielsweise ihr Vorgehen aufeinander abstimmen müssen oder einander ihre Ergebnisse mitteilen.

Räumlichkeit

Die meisten traditionellen GUI-Anwendungen sind flach, eben und zweidimensional. Diese können durch Techniken wie Schattenwurf und einander überdeckende Elemente in eine zweieinhalbdimensionale Welt gehoben werden. Die Interaktionsmodelle der Microsoft Surface Plattform erweitern diese Darstellung und versuchen den dargestellten Objekten eine räumliche Tiefe zu verleihen und ein natürliches, dreidimensionales Verhalten zu ermöglichen. Dies wird beispielsweise dadurch erreicht, dass fokussierte Elemente, mit denen der Benutzer gerade interagiert, scheinbar ein wenig von der Fläche hochgehoben werden, um dem Benutzer einen dreidimensionalen Eindruck und Realismus zu vermitteln.

Übernatürliche Interaktionen

Neben der möglichst realistischen Abbildung der physikalischen Gesetze, wie beispielsweise das Verschieben eines Fotos auf der Tischoberfläche, das in Abbildung 2.10 schematisch dargestellt ist, erweitert die Microsoft Surface Plattform die Interaktionsmöglichkeiten mit Objekten auch in eine übernatürliche Richtung, die in der realen Welt so nicht möglich sind. Ein Beispiel dafür ist das einfache Skalieren von Fotos mit Hilfe einer Geste, welches in Abbildung 2.11 zu sehen ist.

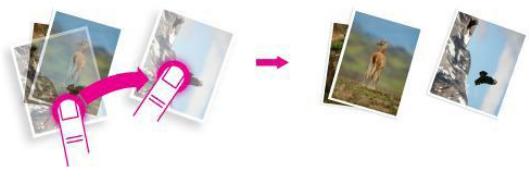


Abbildung 2.10 Verschieben eines Fotos aus [Microsoft Corporation, 2009]

Abbildung 2.11 Skalieren eines Fotos aus [Microsoft Corporation, 2009]

Um dieses übernatürliche Verhalten zu erreichen wird empfohlen zunächst die realen Interaktionsmöglichkeiten der Objekte zu realisieren und diese um weitere, intuitive Möglichkeiten zu erweitern.

Kontextuelle Umgebungen

Surface-Anwendungen erwidern die Interaktionen der Benutzer und Objekte, indem sie basierend auf dem Zustand des internen und externen Umfelds reagiert. Die Wechsel der Zustände sind abhängig vom aktuellen Zustand des Systems, sowie den Interaktionen und Verhalten der Benutzer. Dabei soll den Benutzern eine direkte Rückmeldung über die Auswirkungen ihrer Interaktionen gegeben werden, so dass sie die Auswirkungen erwarten und die Anwendung ihren Intentionen entsprechend steuern können.

Scaffolding

Scaffolding[6] bezeichnet eine Vorgehensweise, die aus dem Bereich der konstruktivistischen Lerntheorie stammt und dem Benutzer durch Anleitungen oder Denkanstößen beim Lernprozess und dem Verstehen neuer Zusammenhänge helfen soll. Dabei werden aufbauend auf vorhandenen Wissensgrundlagen weiterführende Zusammenhänge erläutert, wodurch es dem Lernenden leichter fällt diese zu verstehen.

Übertragen auf die Microsoft Surface Plattform bedeutet dies, dass dem Benutzer nur die Möglichkeiten angezeigt werden sollten, die innerhalb des aktuellen Kontexts sinnvoll und für ihn hilfreich sind. Zudem sollten alle größeren Herausforderungen in kleinere Abschnitte aufgeteilt werden, sodass es dem Benutzer leichter fällt diese zu verstehen und nachzuvollziehen.

Ausführungsästhetik

Um dem Benutzer ein nahtloses Erlebnis bieten zu können, ist es unerlässlich, dass dieses Erlebnis nicht durch eine schlechte Ausführungsgeschwindigkeit oder eine verzögerte Reaktionszeit der Anwendung getrübt wird. Außerdem sollten möglichst viele Sinne des Benutzers angesprochen werden, also neben der rein visuellen Wahrnehmung der Benutzeroberfläche beispielsweise auch akustische Effekte in die

[6] Scaffolding vom Englischen „scaffolding“, das so viel bedeutet wie Gerüst oder Grundlage.

Anwendung integriert werden. Dadurch wird es möglich, dass der Benutzer in die Anwendung eintauchen kann, seine Vorstellungskraft gefördert wird und er insgesamt eine positive Erfahrung bei der Verwendung der Anwendung macht.

3. Problemstellung und Lösungsansatz

Dieses Kapitel bietet einen Überblick über die Probleme aktueller, modellgestützter Entwicklungsumgebungen im Bezug zu den Rahmenbedingungen dieser Arbeit und stellt anschließend einen Ansatz vor, wie diese Probleme gelöst werden können.

3.1. Problemstellung

Die meisten aktuellen Entwicklungsumgebungen zur Erstellung modellgestützter Anwendungen haben einige gravierenden Nachteile, die in den folgenden Abschnitten beschrieben werden.

Fehlende Endbenutzer-Unterstützung

Die meisten in dieser Arbeit vorgestellten MB-UIDE-Plattformen verfügen über ein sehr komplexes Modell und, falls überhaupt vorhanden, einen schwer zu bedienenden Editor für diese. Dies ist unter anderem eine Folge davon, dass die Zielgruppe der meisten Plattformen Personen sind, die mit der Entwicklung von Anwendungen vertraut sind, also in der Regel professionelle Softwareentwickler.

Für die meisten erfahrenen Entwickler ist es nicht schwer in Modellen zu denken, da sie in ihrer täglichen Arbeit mit diesen umgehen und sie verstehen müssen. Dazu zählen beispielsweise ER-Diagramme zur Modellierung von Datenbanken oder UML-Diagramme für die Modellierung von Anwendungen.

Neben dem Umgang mit Modellen ist es für Softwareentwickler ebenfalls möglich Methoden zu implementieren, damit diese innerhalb des Modells verwendet werden können. Ein Beispiel dafür ist das Anwendungsmodell von Teallach [Barclay, et al., 1999].

Im Gegensatz dazu ist der Umgang mit Modellen und die Entwicklung von Software für viele Endbenutzer eine Herausforderung, da sie es nicht gewohnt sind und daher viel Einarbeitungszeit benötigen, die sie in vielen Fällen vermutlich nicht aufbringen möchten. Im Rahmen einer Anwendung für Endbenutzer ist dies dementsprechend kein akzeptabler Zustand.

Fehlende Multi-Touch-Unterstützung

Wie in den Grundlagen dieser Arbeit bereits erwähnt, sind Multi-Touch-Umgebungen eine recht neue Erscheinung innerhalb von Endbenutzer-Anwendungen und die meisten MB-UIDE-Plattformen verfügen nicht über eine Unterstützung dieser Technologie.

Dies ist vor bei beinahe allen Plattformen der Fall, insbesondere bei solchen, die eine ausführbare Anwendung erstellen indem sie direkt aus dem zugrundeliegenden Modell Quellcode erzeugen. In diesem Fall müsste die gesamte Erzeugung des Quellcodes so angepasst werden, dass die Multi-Touch-Technologie unterstützt wird.

Bei der direkten Interpretation des Modells müsste die Interpretations-Komponente angepasst werden, welche die Modellelemente entsprechend in die Benutzungsschnittstelle übersetzt.

Neben diesen technischen Aspekten, spielt auch die Modellierung der Software eine entscheidende Rolle. Durch die Verwendung von Multi-Touch-Umgebungen wird die gleichzeitige Verwendung einer Anwendung von mehreren Benutzern ermöglicht, wodurch beispielsweise die parallele Ausführung von mehreren Aufgaben unterstützt werden kann. Viele der vorgestellten Modelle bieten nur eine sequentielle oder selektive Ausführung von Aufgaben an.

Das Ziel dieser Arbeit ist es, die beschriebenen Nachteile aktueller MB-UIDE-Plattformen in Bezug auf Endbenutzer- und Multi-Touch-Unterstützung durch einen neuen Lösungsansatz zu beheben. Dazu wird ein endbenutzertaugliches Modell zur Beschreibung von Multi-Touch-Anwendungen erstellt, welches in Form einer prototypischen Implementierung umgesetzt wird, die zusätzlich eine Komponente zur Ausführung dieses Modells enthält.

3.2. Lösungsansatz

Um die im vorherigen Abschnitt angesprochenen Ziele zu erreichen, ist es unumgänglich den wird im Rahmen dieser Arbeit zunächst eine benutzerzentrierte Sicht eingenommen, welche die Interaktionen zwischen den Benutzern und dem zu erstellenden System beschreibt. Um eine weitere Verfeinerung der Benutzerinteraktion zu erreichen, werden diese in Kombination mit Werttypen behandelt und zu einer Taxonomie zusammengefasst.

Diese erstellte Taxonomie bildet die Grundlage für ein Modell, auf dessen Basis Endbenutzer Multi-Touch-Anwendungen beschreiben können. Dazu verwendet es die folgenden drei aufeinander aufbauenden Teilmodelle:

- Anwendungsmodell
  Modelliert die Daten, die innerhalb der modellierten Anwendung verwendet werden können.
- Präsentationsmodell
  Modelliert die verschiedenen Interaktionen auf den beschriebenen Daten.
- Ablaufmodell
  Modelliert die Abfolge der beschriebenen Interaktionen.

Dabei bauen diese drei Modelle auf den folgenden Prinzipien auf:

- Möglichst geringe Komplexität
  Damit die vorgestellten Modelle auch von Endbenutzern ohne viel Einarbeitungszeit verwendet werden können, wird ihre Komplexität so gering wie möglich gehalten werden, wodurch zusätzlich die Fehlerwahrscheinlichkeit verringert wird.

- Möglichst hohe Einflussnahme
  Damit die erstellten Anwendungen den Anforderungen der Benutzer möglichst gut entsprechen, werden diese möglichst viel Einfluss auf die erstellten Benutzungsschnittstellen haben. Dazu gehört beispielsweise, dass sie die Anordnung der Elemente festlegen können.
- Multi-Touch-Unterstützung
  Wo möglich soll eine Unterstützung für Multi-Touch gegeben sein. Ein Beispiel dafür ist die Modellierung mehrerer, parallel ablaufender Aufgaben im Rahmen der Ablaufmodellierung.

Neben der Modellierung besteht der vorgestellte Lösungsansatz weiterhin aus einer Komponente, welche auf Basis des vorgestellten Modells eine ausführbare Anwendung erzeugt.

Eine grafische Darstellung des Lösungsansatzes, sowie eine Übersicht der Kapitel, in denen die beschriebenen Lösungsbestandteile ausführlich behandelt werden, sind in Abbildung 3.1 zu sehen.

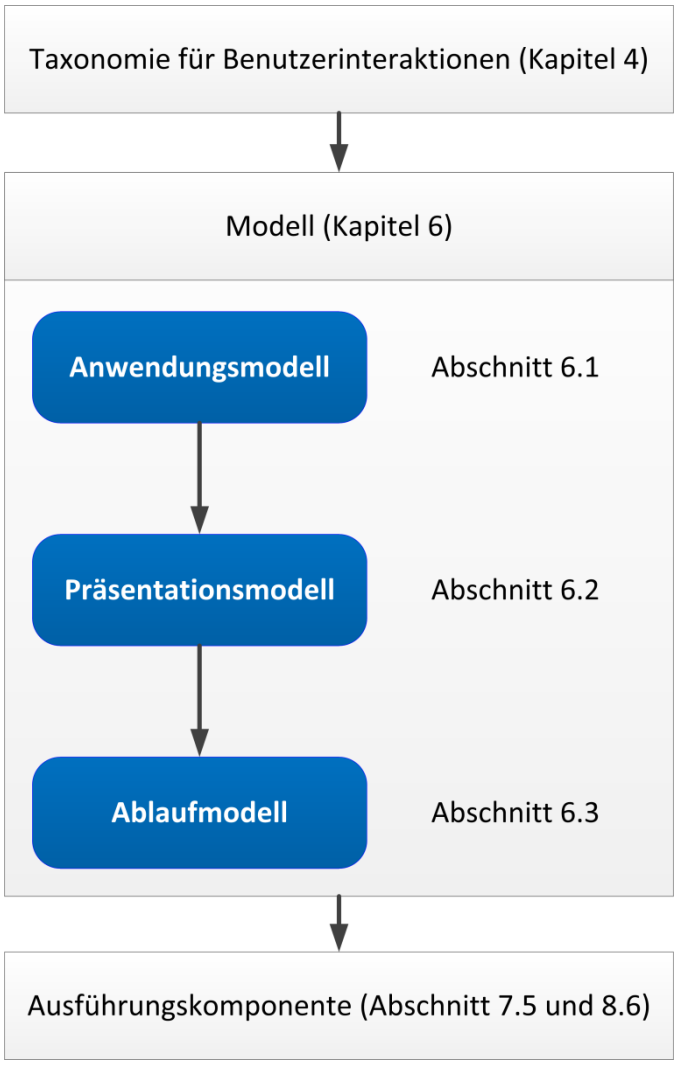


Abbildung 3.1 Grafische Darstellung der Bestandteile des Lösungsansatzes

4. Taxonomie für Benutzerinteraktionen

Wie im vorherigen Kapitel erläutert, basiert das, im Rahmen dieser Arbeit zu erstellende, Modell zur Beschreibung von Benutzungsschnittstellen auf einer Taxonomie von Benutzerinteraktionen und -problemstellungen. In diesem Kapitel wird zunächst diese Taxonomie erstellt, deren Realisierung auf Basis der Microsoft Surface Plattform im darauffolgenden Kapitel beschrieben wird.

4.1. Grundlegende Benutzerinteraktionen

Basierend auf [Barclay, et al., 1999] und [Meixner, et al., 2008] können die grundlegenden Benutzerinteraktionen in fünf elementaren Kategorien eingeteilt werden, die im Folgenden vorgestellt werden und deren Bedeutung im Rahmen dieser Arbeit erläutert wird:

- Eingabe:
  Bei der Eingabe übermittelt der Benutzer einzelne, neue und absolute Werte an das System, die er explizit eingibt.
- Ausgabe:
  Die Ausgabe dient dazu den Benutzer über die aktuellen Werte innerhalb des Systems zu informieren.
- Bearbeitung:
  Mit Hilfe der Bearbeitung wird es dem Benutzer ermöglicht Werte, die sich bereits innerhalb des Systems befinden, zu editieren.
- Auswahl:
  Die Auswahl ermöglicht es dem Benutzer einen einzelnen oder mehrere Werte aus einer vorgegebenen Menge von Auswahlmöglichkeiten auszuwählen.
- Ausführung:
  Die Ausführung bildet das Auslösen genau einer Funktion dar.

Eine graphische Darstellung der grundlegenden Benutzerinteraktionen ist in Abbildung 4.1 zu sehen.

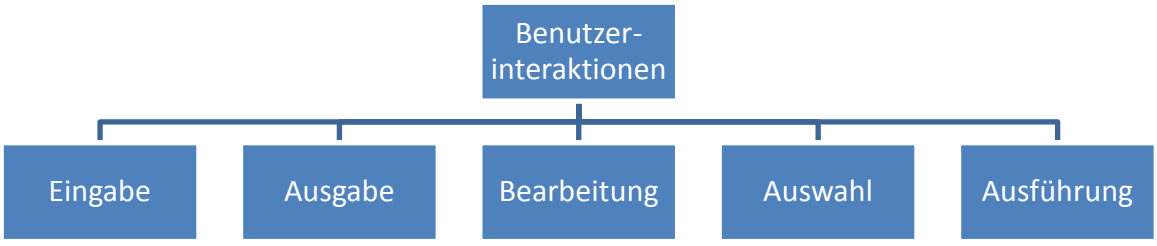


Abbildung 4.1 Grundlegende Benutzerinteraktionen

Ausgehend von diesen grundlegenden Interaktionsmöglichkeiten können verschiede Verfeinerungen für die einzelnen Interaktionen definiert werden, die in den folgenden Abschnitten dieses Kapitels beschrieben werden.

4.2. Werttypen

Um die grundlegenden Benutzerinteraktionen weiter zu unterteilen, gibt es prinzipiell zwei verschiedene Möglichkeiten. Entweder man verfeinert die Art der Interaktion, also beispielsweise *wie* der Benutzer Werte eingeben kann, oder man unterscheidet die verschiedenen Arten von Werten, die vom Benutzer verarbeitet werden können. In dieser Arbeit wird die zweite Variante gewählt und daher werden die grundlegenden Werttypen aufgezählt und beschrieben:

- Boolesche Werte
  Boolesche, auch Wahrheitswerte genannt, sind die einfachste Art von Werten, da sie lediglich die zwei Zustände „wahr“ und „falsch“ annehmen können. In seltenen Fällen können sie auch einen dritten Zustand („undefiniert“) einnehmen.
- Textuelle Werte
  Textuelle Werte umfassen neben einzelnen Buchstaben, Ziffern und Worten auch mehrzeilige Texte jeglicher Art. Prinzipiell kann man dabei zwischen zwei verschiedenen textuellen Werten unterscheiden:
  - Beliebige Textwerte
  - Formatierte Textwerte

  Während beliebige Textwerte alle möglichen textuellen Werte umfassen, folgen formatierte Textwerte einem vorgegebenen Format. Damit kann beispielsweise sichergestellt werden, dass nur valide Emailadressen, Postleitzahlen oder Uhrzeit- und Datumwerte im System vorhanden sind.
- Numerische Werte
  Die numerischen Werte umfassen alle Werte, die Zahlen repräsentieren. Dies sind im Speziellen die folgenden Werte:
  - Natürliche Zahlen
  - Ganze Zahlen
  - Gleitkommazahlen
- Visuelle Werte
  Die visuellen Werte lassen sich in drei verschiedene Unterkategorien einteilen:
  - Absolute visuelle Werte
  - Relative visuelle Werte
  - Geometrische visuelle Werte

  Die absoluten visuellen Werte bestehen dabei aus einer Menge von absoluten (zweidimensionalen) Koordinaten, wie zum Beispiel (0,7) oder (4,2), sowie einer Farbangabe für jede Koordinate. Dadurch ist es beispielsweise möglich pixelbasierte Bilddateien zu repräsentieren.
  Relative visuelle Werte entsprechen einer Menge relativer Koordinaten, die auf den Bereich ([0, 1], [0, 1]) beschränkt sind. Dadurch ist es möglich diese abhängig vom verfügbaren Platz darzustellen oder die Werte zu skalieren.

Die dritte Unterkategorie stellen die geometrischen visuellen Werte dar. Diese umfassen keine expliziten Angaben zu Koordinaten, sondern bestehen aus geometrischen Figuren, wie Rechtecken, Kreisen und Linien. Diese verfügen zusätzlich über Eigenschaften wie Füllfarbe sowie Umrandungsfarbe und –dicke.

Eine grafische Übersicht der beschriebenen Werttypen ist in Abbildung 4.2 zu sehen.

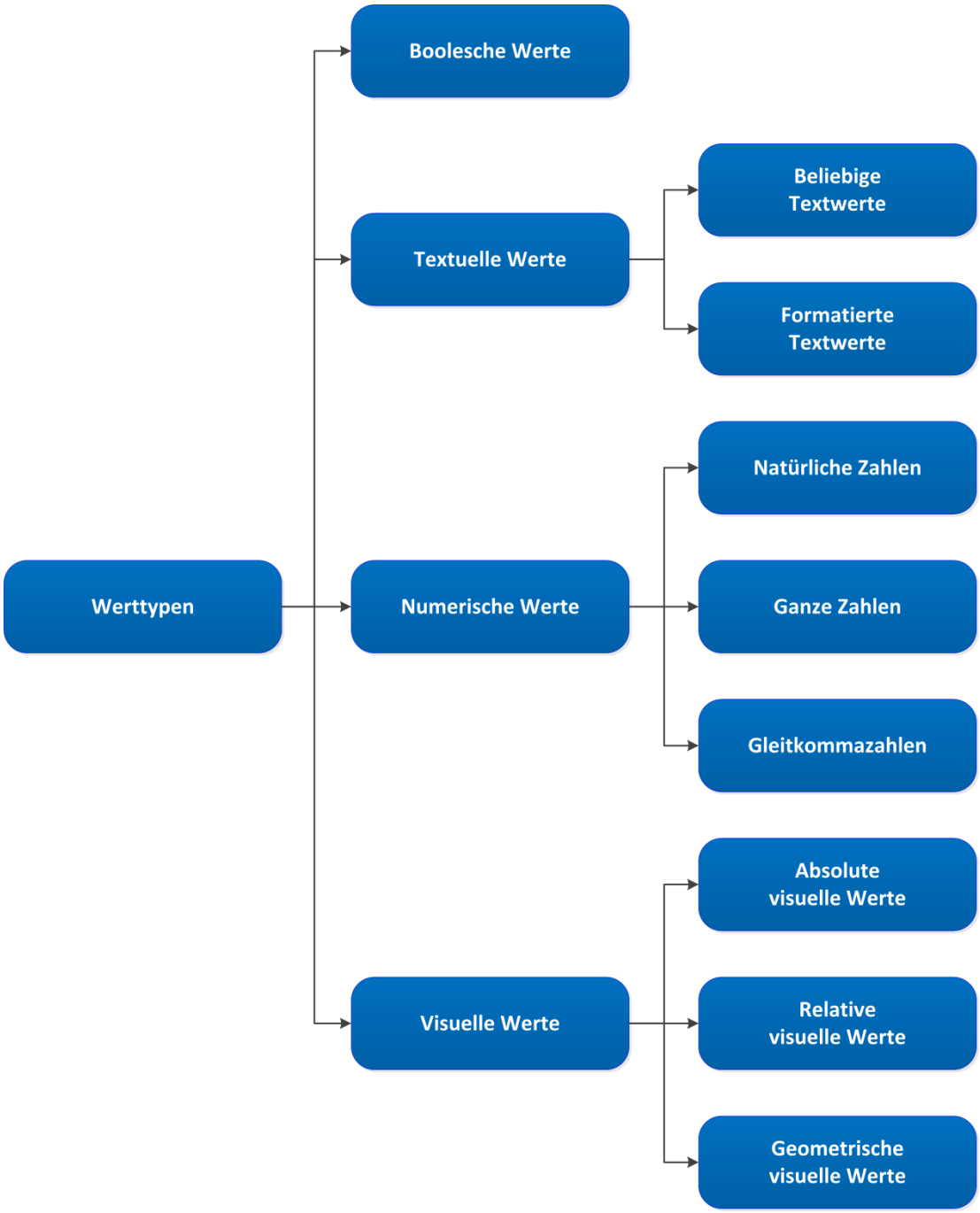


Abbildung 4.2 Übersicht der verfügbaren Werttypen

Nachdem nun die grundlegenden Werttypen definiert sind, kann in den folgenden Abschnitten auf die einzelnen Benutzerinteraktionen eingegangen werden.

4.3. Eingabe

Da dieser Abschnitt die Eingabe von Werten umfasst, muss zunächst geklärt werden, wie eine Eingabe in einer Multi-Touch-Umgebung, die ohne Tastatur und Maus auskommt, realisiert werden kann. Eine Möglichkeit zur Eingabe von Werten, besteht darin, eigene Benutzersteuerelemente zur Verfügung zu stellen, welche für die Eingabe der Werte optimiert sind. Ein Beispiel dafür sind Checkboxen für boolesche Werte[7]. Eine einfache und in der Microsoft Surface Plattform direkt integrierte Variante zur Eingabe nicht vordefinierter Werte, ist die Anzeige einer Bildschirmtastatur, über die Werte eingegeben werden können. Allerdings kann man sich auch weitere Eingabemethoden vorstellen, beispielsweise mit Hilfe von Tag-Objekten, die spezielle Buchstaben, Ziffern oder Sonderzeichen repräsentieren.

Prinzipiell werden durch die Eingabe alle beschriebenen Werttypen unterstützt. Daher werden in diesem Abschnitt nur die Werttypen aufgeführt, bei denen es Besonderheiten in Bezug auf die Eingabe gibt, die einer genaueren Betrachtung bedürfen.

Eingabe textueller Werte

Wie bereits erwähnt, lassen sich textuelle Werte in die Kategorien *beliebige Textwerte* und *formatierte Textwerte* einteilen, die beide bei der Eingabe unterstützt werden. Allerdings kann die Eingabe formatierter Textwerte auf verschiedene Arten realisiert werden. Eine Möglichkeit besteht darin, dass während der Eingabe geprüft wird, ob der eingegebene Wert valide ist. Ist dies nicht der Fall, wird die letzte Eingabe verworfen. Diese Technik wird beispielsweise bei der Verwendung von Masken zur Eingabe eingesetzt.

Eine weitere Möglichkeit besteht darin den eingegebenen Text, zum Beispiel mit Hilfe regulärer Ausdrücke, zu überprüfen, sobald das Eingabeelement den Fokus verliert. Ist der eingegebene Text dann nicht valide, wird der Fokus automatisch zurück auf das Eingabeelement gesetzt.

Als letzte Möglichkeit besteht die Eingabe formatierter Textwerte zu realisieren besteht darin, dass nur eine valide Eingabe möglich ist. So können die Elemente, die zu einer Eingabe eines nicht validen Wertes führen würden, beispielsweise deaktiviert werden. Dadurch wird es dem Benutzer gar nicht erst ermöglicht ungültige Eingaben zu tätigen.

Eingabe numerischer Werte

Die Eingabe numerischer Werte kann als Spezialfall der Eingabe textueller Werte angesehen werden, da man durch die Angabe von Formaten sicherstellen kann, dass ausschließlich numerische Werte eingegeben werden können. Allerdings kann man

[7] Eine klare Trennung zwischen den verschiedenen Benutzerinteraktionen ist nicht immer möglich. So kann die Eingabe boolescher Werte durch eine Combobox auch als Auswahl angesehen werden.

dies auch auf anderen Wegen erreichen, beispielsweise indem nur Eingabeelemente für Zahlen zur Verfügung stellt, wie es bei einem Nummernblock einer normalen Tastatur der Fall ist.

4.4. Ausgabe

Das Gegenstück der Eingabe bildet die Ausgabe. Ziel dieser ist es, den Benutzer über den Zustand des Systems zu informieren und ihm die vorhandenen Daten anzuzeigen. Prinzipiell können dabei die gleichen Werttypen ausgegeben werden, die vom Benutzer auch eingegeben werden können.

4.5. Bearbeitung

Die Bearbeitung von Werten kann als Kombination der Benutzerinteraktionen Eingabe und Ausgabe angesehen werden. Zunächst werden die bereits im System vorhandenen Werte angezeigt und können anschließend durch die Eingabe neuer Werte editiert werden. Der hauptsächliche Unterschied zwischen der Eingabe und der Bearbeitung besteht darin, dass durch die Eingabe neue Werte in das System gebracht werden können, mit Hilfe der Bearbeitung aber bereits vorhandene Werte editiert werden.

4.6. Auswahl

Die Auswahl ermöglicht dem Benutzer Werte aus einer vorgegebenen Menge von Auswahlmöglichkeiten auszuwählen. Dabei kann man prinzipiell zwischen den folgenden drei verschiedenen Arten der Auswahl unterscheiden:

- Einzelauswahl
  Die Einzelauswahl ermöglicht es dem Benutzer aus einer Menge von Werten einen einzelnen Wert auszuwählen.
- Mehrfachauswahl
  Im Gegensatz zur Einzelauswahl ermöglicht die Mehrfachauswahl die gleichzeitige Auswahl von mehreren Werten.
- Bereichsauswahl
  Die Bereichsauswahl ermöglicht es dem Benutzer einen bestimmten Bereich und damit alle darin enthaltenen Werte auszuwählen. Insbesondere bei numerischen Werten ist diese Art der Auswahl sinnvoll.

Die grundlegende Methode zur Definition der zur Auswahl stehenden Elemente ist deren explizite Angabe. Dadurch ist es möglich, dass alle möglichen Werttypen dem Benutzer zur Auswahl angeboten werden können. Im Allgemeinen ist es allerdings sinnvoll, dass innerhalb einer Menge von Auswahlmöglichkeiten nur ein Werttyp verwendet wird und nicht jedes Element einen anderen besitzt.

Beispiele für die explizite Angabe der Auswahlmöglichkeiten ist eine Menge von Personennamen („Max Mustermann“, „Dennis Paulenz“, „Thomas Schlegel“) oder Postleitzahlen („70469“, „71229“, „71636“).

Neben der expliziten Angabe der zur Auswahl besteht bei numerischen Werten zusätzlich die Möglichkeit die zur Auswahl stehenden Werte implizit mit Hilfe eines Wertebereichs und einer optionalen Schrittgröße zwischen den Werten anzugeben. Dies geschieht durch die Angabe der Intervallgrenzen und gegebenenfalls der Schrittgröße. Ein Beispiel für diese Methode der Angabe auswählbarer Werte ist das Intervall [0,1] mit der Schrittweite 0,2, woraus die verfügbaren Werte „0“, „0,2“, „0,4“, „0,6“, „0,8“ und „1,0“ resultieren.

Diese Art der Definition der verfügbaren Auswahlmöglichkeiten eignet sich vor allem dann, wenn es viele Elemente sind und die Abstände zwischen den einzelnen Elementen immer gleich sind.

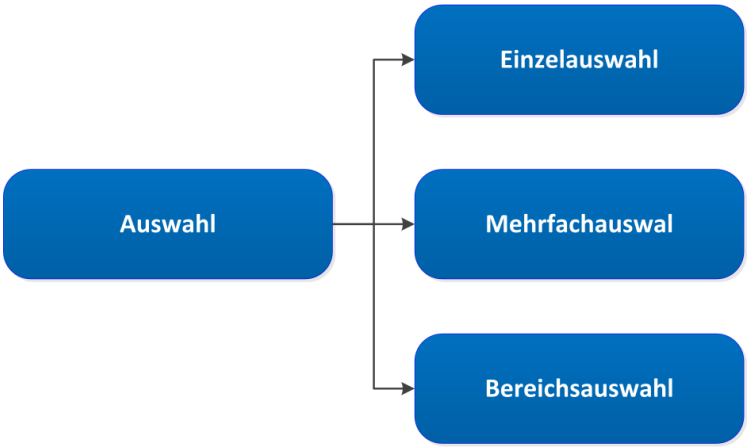


Abbildung 4.3 Unterteilung der Benutzerinteraktion Auswahl

4.7. Ausführung

Durch die Ausführung kann der Benutzer gezielt Funktionen der Anwendung auslösen. Dies können beispielsweise der Wechsel zu einer neuen Ansicht, das Speichern des aktuellen Zustands, das Zurücksetzen eines Werts oder das Beenden der Anwendung sein.

Im Gegensatz zu den anderen Benutzerinteraktionen, ist es bei den Elementen der Ausführung nicht möglich diese den verwendeten Werttypen zuzuordnen, da diese unabhängig davon sind und sich lediglich dadurch unterscheiden, wodurch die Funktion ausgelöst wird. Da dies aber in den Aufgabenbereich der letztendlichen Implementierung innerhalb einer spezifischen Plattform fällt, wird dieser Aspekt erst später in Abschnitt 5.5 behandelt.

Im Rahmen dieser Arbeit stehen die folgenden Ausführungsinteraktionen zur Verfügung:

- Interaktion abschließen

Schließt die aktuelle Interaktion ab und markiert sie als beendet.

- Programm beenden
  Beendet die Anwendung, in welcher die aktuelle Interaktion ausgeführt wird.
- Zufälligen Wert auswählen
  Ermöglicht bei einer Auswahlinteraktion die zufällige Auswahl eines Wertes.

Als weitere Ausführungsinteraktionen, die im Rahmen dieser Arbeit nicht realisiert werden, aber in einer späteren Ausbaustufe denkbar und sinnvoll wären, sind die folgenden zu nennen:

- Speichern
  Sowohl zum Speichern des Zustands der aktuellen Interaktion, als auch des Gesamtsystems.
- Laden
  Entspricht dem Gegenstück zum Speichern und lädt entweder den Zustand einer Interaktion oder des Gesamtsystems.
- Interaktion zurücksetzen
  Setzt die aktuelle Interaktion in den Ursprungszustand zurück.
- Rückgängig
  Macht die zuletzt getätigte Interaktion des Benutzers rückgängig.
- Wiederherstellen
  Stellt eine rückgängig gemachte Interaktion wieder her.
- Abbrechen
  Bricht die aktuelle Interaktion ab.

4.8. Gesamtdarstellung der Benutzerinteraktionen

Dieser Abschnitt bildet den Abschluss der dem Modell zugrundeliegenden Taxonomie von Benutzerinteraktionselementen und zeigt die vorgestellten Elemente in einer Gesamtansicht.

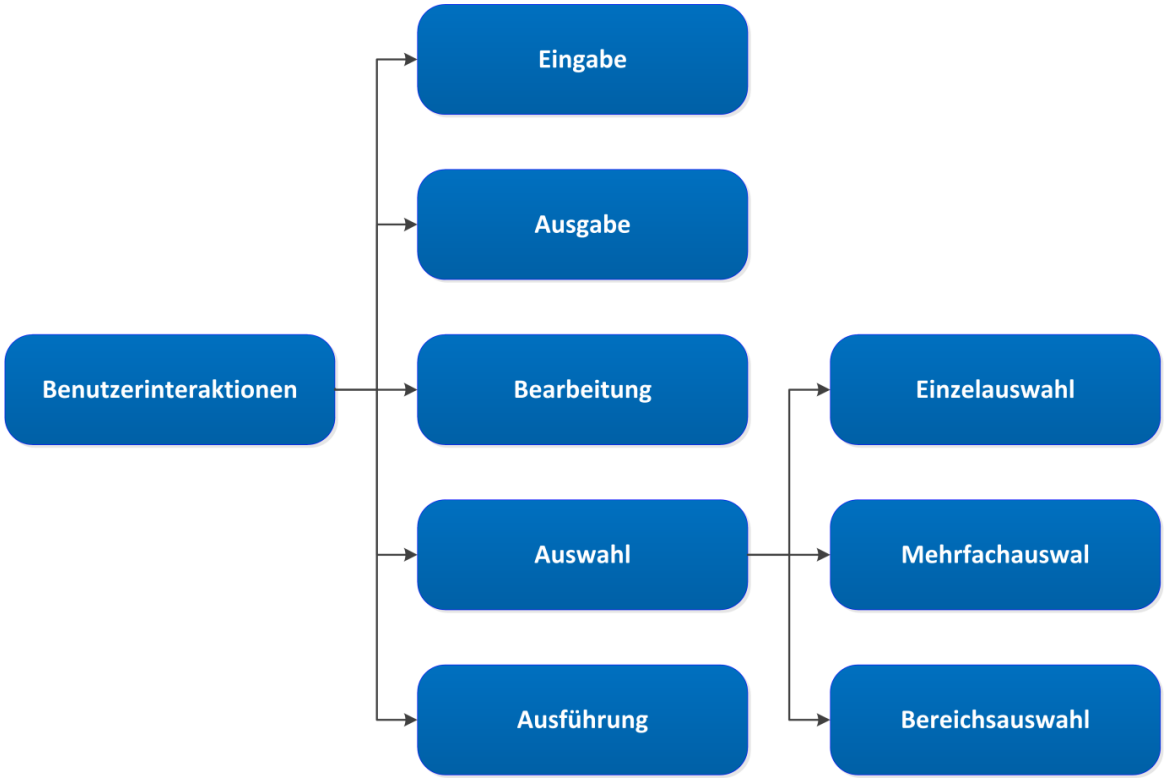


Abbildung 4.4 Übersicht der Benutzerinteraktionen

5. Abbildung der Benutzerinteraktionselemente auf Steuerelemente der Microsoft Surface-Umgebung

Nachdem im vorherigen Kapitel erläutert wurde, welche Interaktionsmöglichkeiten dem Benutzer bei der Verwendung von Benutzungsschnittstellen zur Verfügung stehen, ist das Ziel dieses Kapitels die Abbildung dieser abstrakten Elemente und Werttypen auf reale Benutzersteuerelemente, die entweder standardmäßig durch die Microsoft Surface Plattform zur Verfügung gestellt werden, oder innerhalb anderer Plattformen bekannt sind und somit auf die Microsoft Surface Plattform portiert werden können.

Da es prinzipiell jeder Person mit Entwicklungserfahrung möglich ist weitere Benutzersteuerelemente für die Microsoft Surface Plattform zu erstellen, kann die hier vorgestellte Abbildung der abstrakten Interaktionen auf reale Steuerelemente nicht vollständig sein und erhebt diesen Anspruch auch nicht. Sie dient vielmehr als eine beispielhafte Umsetzung, die weitestgehend auf bereits vorhandene Steuerelemente zurückgreift und einige weitere, neu erstellte Benutzersteuerelemente verwendet.

Damit die Verbindung zwischen der Beschreibung der Benutzerinteraktionselemente im vorherigen Kapitel und der entsprechenden Beschreibung der Realisierung in diesem Kapitel einfacher wird, ist dieses Kapitel strukturell genauso aufgebaut wie das vorherige.

5.1. Eingabe

Für die Eingabe von Werten, gibt es innerhalb des Surface SDKs verschiedene Benutzersteuerelemente, die Folgenden vorgestellt werden.

Eingabe boolescher Werte

Zur Eingabe boolescher Werte steht die SurfaceCheckBox zur Verfügung. Diese bietet die Möglichkeit zwischen den Werten „wahr“ (true) und „falsch“ (false) zu wechseln und hat im nichtinitialisierten Zustand, also bevor einer der genannten Werte zum ersten Mal gesetzt wurde, den Wert „undefiniert“ (null). Zudem wird neben der Schaltfläche zum Setzen des Wertes eine Beschriftung angezeigt, welche dem Benutzer signalisiert welchen Wert er gerade verändert.

Abbildung 5.1 zeigt eine beispielhafte Darstellung der SurfaceCheckBox im Zustand „wahr“ und „falsch“.

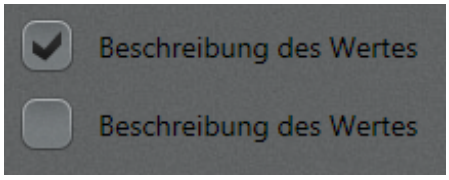


Abbildung 5.1 SurfaceCheckBox im Zustand "wahr" (oben) und "falsch" (unten)

Eingabe textueller Werte

Um dem Benutzer die Eingabe von textuellen Werten zu ermöglichen, steht die SurfaceTextBox zur Verfügung. Sobald diese den Eingabefokus erhält, also sobald der Benutzer sie mit dem Finger berührt, wird dabei eine virtuelle Bildschirmtastatur angezeigt, mit deren Hilfe der Benutzer den gewünschten Text eingeben kann.

Prinzipiell kann auf diese Art jeder beliebige Wert eingegeben werden, sofern die einzelnen Symbole auf der virtuellen Tastatur vorhanden sind. Falls sie dies nicht sind, muss eine eigene Bildschirmtastatur erstellt werden, welche die entsprechenden Symbole anbietet.

In Abbildung 5.2 sind eine SurfaceTextBox in den Standardeinstellungen sowie die virtuelle Bildschirmtastatur zu sehen.

Abbildung 5.2 SurfaceTextBox (oben) mit virtueller Bildschirmtastatur (unten)

Außer der abgebildeten einzeiligen SurfaceTextBox gibt es weitere Einstellungsmöglichkeiten, die es erlauben auch mehrzeilige Texte einzugeben und den Zeilenumbruch in Bezug auf die Wortgrenzen zu beeinflussen.

Um mit dieser SurfaceTextBox die Eingabe auf formatierte Werte einzuschränken, wird innerhalb des Surface SDKs keine fertige Lösung geliefert. Um dies zu erreichen, besteht die Möglichkeit auf das „PreviewTextInput-“ beziehungsweise das „TextChanged-Event“ zu reagieren und den Wert zwischen dem Betätigen der Taste und dem Eintrag in die SurfaceTextBox auf die entsprechenden Bedingung zu überprüfen und ihn nur in dem Fall einzutragen, falls die Bedingung erfüllt ist. Im Rahmen dieser Arbeit wird dafür eine SurfaceMaskedTextBox erstellt, welche diese Aufgaben erfüllt und als Kriterium für das Eintragen des Wertes auf einen wählbaren

regulären Ausdruck zurückgreift. Um dem Benutzer die Arbeit zu erleichtern, werden reguläre Ausdrücke für die folgenden, häufig verwendeten textuellen Werte vordefiniert, die vom Benutzer einfach ausgewählt werden können:

- Datum
- Emailadresse
- IP-Adresse

Neben der SurfaceTextBox ist ebenfalls noch die SurfacePasswordBox verfügbar, welche zur Eingabe von Werten gedacht ist, deren Wert dabei nicht direkt angezeigt werden soll, wie es beispielsweise bei geheimen Passwörtern der Fall ist. Da die SurfacePasswordBox in ihrer Darstellung kaum von der SurfaceTextBox abweicht und lediglich der eingegebene Text unkenntlich gemacht wird, wird an dieser Stelle auf eine Abbildung verzichtet.

Eingabe numerischer Werte

Wie bereits im vorherigen Kapitel erwähnt, gibt es verschiedene Arten die Eingabe numerischer Werte zu realisieren. Eine Möglichkeit wäre es auf die, im Rahmen dieser Arbeit erstellte SurfaceMaskedTextBox zurückzugreifen, bei der die Eingabe mit Hilfe von regulären Ausdrücken sichergestellt wird. Dabei kann auf die beschriebene SurfaceMaskedTextBox zurückgegriffen werden, wobei reguläre Ausdrücke für die verschiedenen numerischen Werttypen vorgegeben werden könnten:

- Natürliche Zahlen
- Ganze Zahlen
- Gleitkommazahlen

Des Weiteren ist es möglich weitere Elemente und reguläre Ausdrücke zu erstellen, welche beispielsweise für eine Eingabe von rationalen oder periodischen Zahlen geeignet sind und diese in das entstehende System zu integrieren.

Im Rahmen dieser Arbeit wird ein anderer Weg eingeschlagen, indem Steuerlemente verwendet werden, die explizit für die Eingabe numerischer Werte erstellt wurden. Für die Eingabe von Gleitkommazahlen ist dies der SurfaceSlider. Dieser bietet durch die Angabe eines Minimal- und Maximalwertes und der stufenlosen Auswahl innerhalb dieses Bereichs eine optimale Möglichkeit zur Eingabe von Gleitkommazahlen. Dabei können zur besseren Orientierung des Benutzers Markierungen ober- und/oder unterhalb der Achse angezeigt werden. Abbildung 5.3 zeigt einen SurfaceSlider mit Hilfsmarkierungen unterhalb der Achse.

Abbildung 5.3 SurfaceSlider mit Hilfsmarkierungen

Ein Steuerelement zur Eingabe ganzer und natürlicher Zahlen ist nicht standardmäßig im Surface SDK enthalten. Da es aber in anderen Benutzeroberflächen-Frameworks, zum Beispiel bei der WPF ab Version 4.0, Steuerelemente für genau diesen Zweck gibt, beispielsweise das NumericUpDown-Steuerelement, ist eine Portierung solch eines Elements auf die Microsoft Surface Plattform möglich. Im Rahmen dieser Arbeit wird solch eine Portierung durchgeführt und das SurfaceNumericUpDown-Element erstellt.

Eingabe visueller Werte

Die Eingabe visueller Werte wird im Rahmen dieser Arbeit nicht vollständig unterstützt. Von den drei, in Abschnitt 4.2 beschriebenen Unterkategorien der visuellen Werte werden ausschließlich die absoluten visuellen Werte unterstützt.

Die Eingabe dieser Werte wird durch das SurfaceInkCanvas realisiert, welches dem Benutzer die Möglichkeit bietet durch einfache Berührungen pixelbasierte Bilder zu erstellen. Dabei kann er zwischen verschiedenen Farben wählen und durch die Druckstärke seiner Berührung die Breite der gezeichneten Figuren bestimmen. Ein Beispiel für die Eingabe eines absoluten visuellen Wertes durch das SurfaceInkCanvas ist in Abbildung 5.4 zu sehen.

Abbildung 5.4 SurfaceInkCanvas mit absolutem visuellen Wert

Im Gegensatz zu absoluten visuellen Werten, kann sich die Eingabe relativer und geometrischer visueller Werte, die in dieser Implementierung nicht unterstützt wird, an anderen Standardanwendungen orientieren. Relative visuelle Werte, die eine große Ähnlichkeit mit vektorbasierten Grafiken aufweisen, können sich beispielsweise an Adobe Photoshop oder Microsoft Expression Design anlehnen, welche für die Erstellung vektorbasierter Grafiken geeignet sind. Die geometrischen visuellen Werte könnten durch einen Editor unterstützt werden, der dem Benutzer verschiedene geometrische Figuren anbietet, die dieser anschließend auf der Ober-

fläche platzieren und bearbeiten kann. Dies unterstützen die meisten gängigen Bildeditoren wie Microsoft Paint, Adobe Photoshop oder Inkscape.

Übersicht der Eingabeelemente

Um die Abbildung der Eingabeinteraktionen in Verbindung mit den verschiedenen Werttypen einfacher zu erkennen, wird diese in Abbildung 5.5 dargestellt.

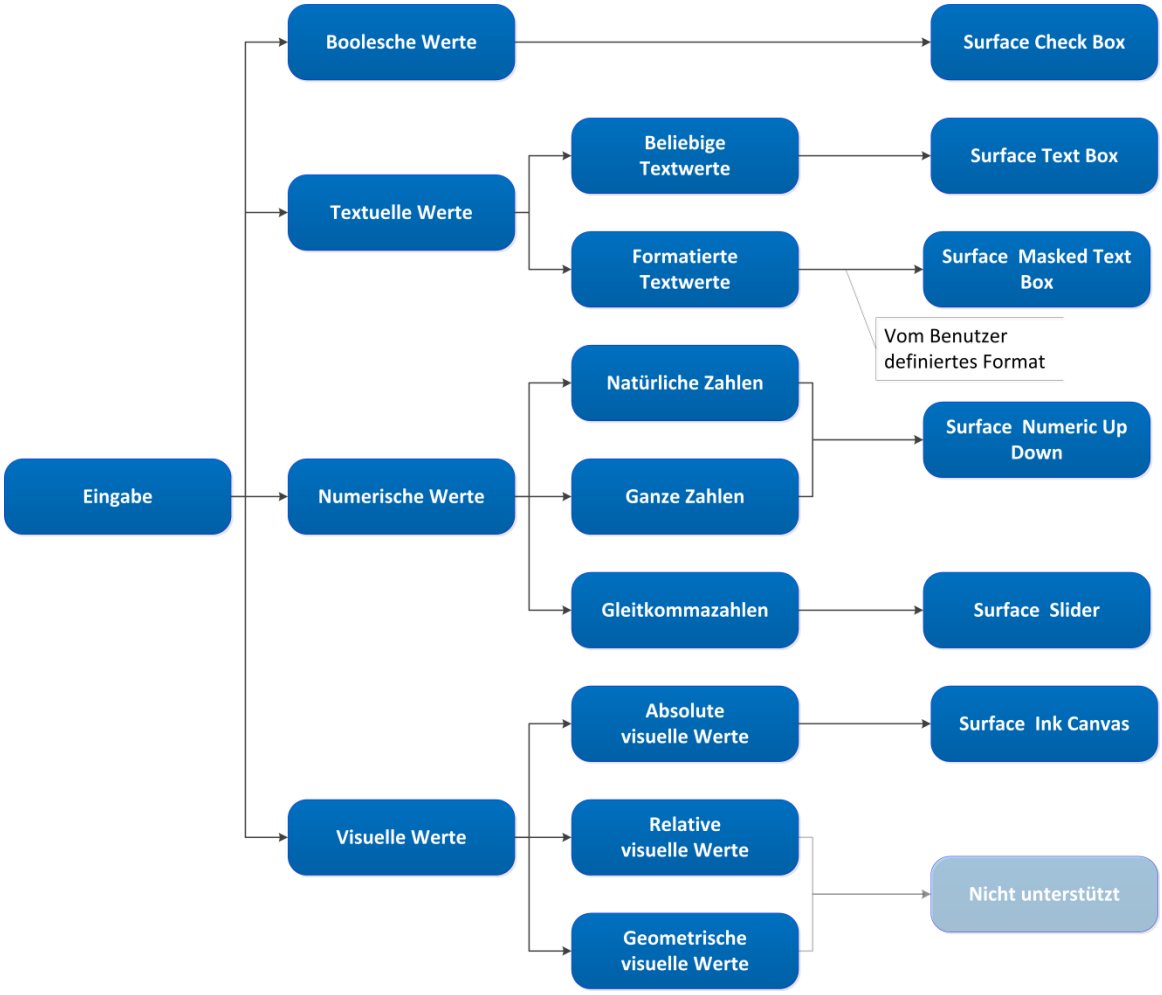


Abbildung 5.5 Übersicht der Eingabeelemente

5.2. Ausgabe

Da bei der Eingabe von Werten die eingegebenen Werte auch angezeigt werden, stehen für die Ausgabe von Werten, die gleichen Steuerelemente zur Verfügung, die auch für die Eingabe verwendet werden können. Diese können so angepasst werden, dass sie lediglich die Werte anzeigen, aber keine Eingaben akzeptieren. Dies kann zum einen dadurch erreicht werden, dass die Steuerelemente deaktiviert werden, zum andern dadurch, dass ein Fokussieren des Elements durch den Benutzer unterbunden wird.

Zudem stehen für alle Werte, die in einen Text umgewandelt werden können, die Steuerelemente Label und TextBlock zur Verfügung, die standardmäßig in der WPF

vorhanden sind. Diese können ebenfalls dazu verwendet werden die Eingabeelemente zu beschriften, damit dem Benutzer verdeutlicht werden kann, welcher Wert mit dem entsprechenden Eingabeelement gesetzt wird. Die Unterschiede zwischen einem Label und einem TextBlock liegen dabei im Umfang der Einstellungsmöglichkeiten. So bietet ein TextBlock wesentlich mehr Möglichkeiten die Formatierung des angezeigten Textes zu beeinflussen, beispielsweise durch Unterstreichungen und kursive Abschnitte innerhalb des Textes.

Übersicht der Ausgabeelemente

Somit ergibt sich für die Abbildung der Interaktionselemente auf die konkreten Benutzersteuerelemente das in Abbildung 5.6 zu sehende Schaubild.

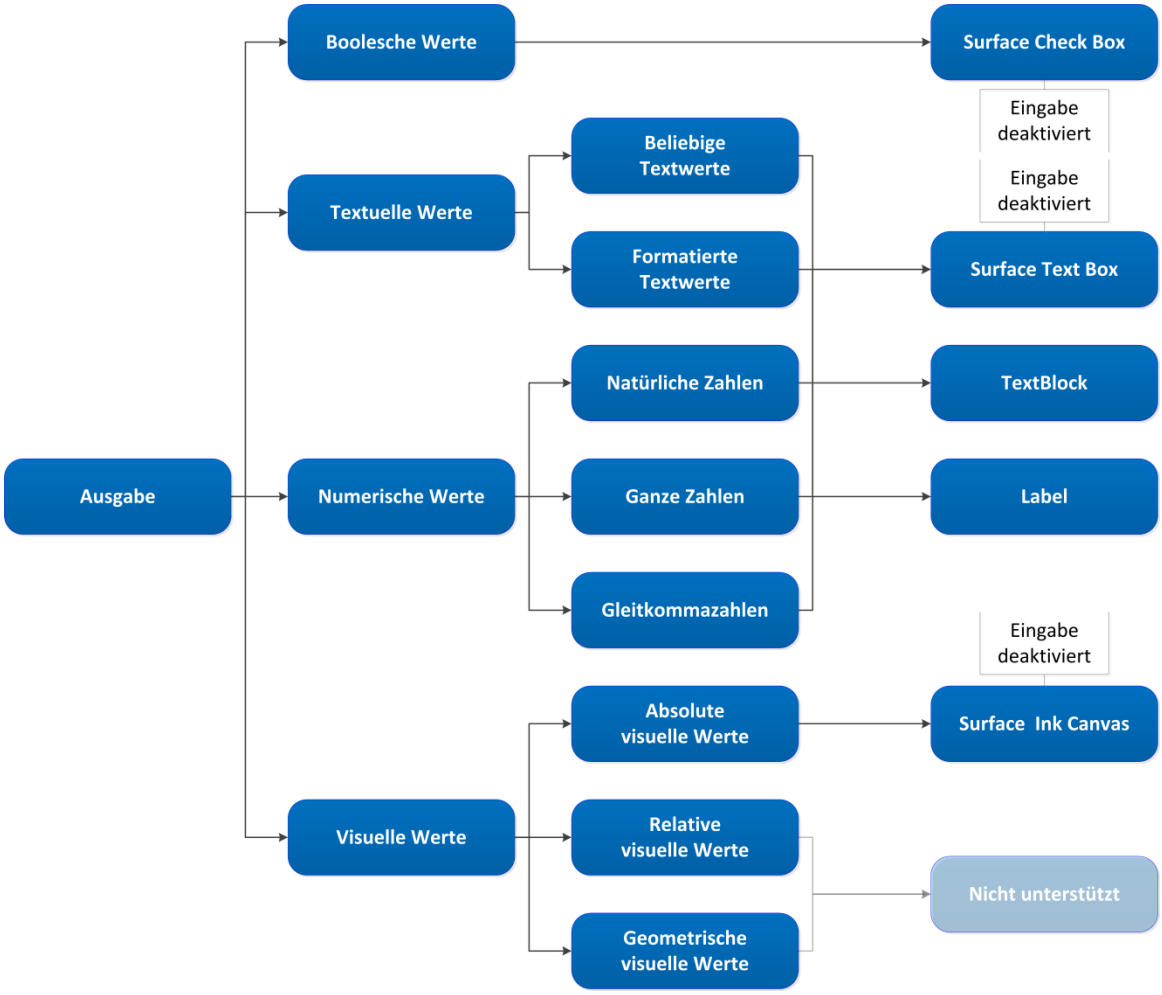


Abbildung 5.6 Übersicht der Ausgabeelemente

5.3. Bearbeitung

Da es sich bei der Bearbeitung um eine Kombination aus der Ein- und Ausgabe handelt, können dafür die gleichen Steuerelemente verwendet werden, die für beide dieser Benutzerinteraktionstypen genutzt werden können. Explizit sind dies:

- SurfaceCheckBox

- SurfaceTextBox
- SurfaceMaskedTextBox
- SurfaceInkCanvas

Die Übersicht der Bearbeitungselemente entspricht also der in Abbildung 5.5 dargstellten Übersicht der Eingabeelemente.

5.4. Auswahl

Für die Auswahl von Werten stellt das Surface SDK eine große Anzahl an Steuerelementen zur Verfügung. Einige davon eignen sich primär zur Auswahl einzelner Werte, wohingegen andere auch die gleichzeitige Auswahl mehrerer Werte ermöglichen. Welche sich wofür eignen wird in den folgenden Abschnitten beschrieben.

5.4.1. Einzelauswahl

Im Bereich der Einzelauswahl gibt es neben den Steuerelementen zur Auswahl beliebiger Werte auch solche, die sich besonders für die Eingabe von booleschen und numerischen Werten eignen. Diese werden in diesem Abschnitt vorgestellt.

Beliebige Werte

Zur Auswahl einzelner Werte aus einer Menge beliebiger Werte, eignet sich zum einen die SurfaceListBox. Diese erhält als Eingabeparameter eine Liste von beliebigen Objekten, die sie anzeigt und dem Benutzer zur Auswahl anbietet. In Abbildung 5.7 ist eine SurfaceListBox zu sehen, welche verschiedene Farben zur Auswahl anbietet, wovon die Farbe „Blau“ aktuell ausgewählt ist.

Abbildung 5.7 SuraceListBox mit ausgewähltem Wert "Blau"

Falls die Anzahl der Elemente so hoch ist, dass die Höhe der SurfaceListBox überschritten wird, wird automatisch ein Scrollbalken eingeblendet mit dessen Hilfe der Benutzer die übrigen Elemente in den dargestellten Bereich bringen kann. Eine beispielhafte Darstellung dieses Verhaltens ist in Abbildung 5.8 illustriert.

Abbildung 5.8 SurfaceListBox mit Scrollbalken

Neben der SurfaceListBox besteht noch die Möglichkeit zur Verwendung von SurfaceRadioButtons. Diese Variante eignet sich vor allem bei einer festen Anzahl von Auswahlmöglichkeiten, die zum einen im Voraus bekannt und zum anderen nicht allzu umfangreich sind. Dabei werden mehrere SurfaceRadioButtons in einer Gruppe zusammengefasst, sodass immer nur ein Button selektiert sein kann. In Abbildung 5.9 ist die bereits bekannte Farbauswahl, basierend auf SurfaceRadioButtons zu sehen.

Abbildung 5.9 SurfaceRadioButtons mit ausgewähltem Wert "Blau"

Prinzipiell lassen sich die vorgestellten Steuerelemente auch zur Auswahl von booleschen und numerischen Werten verwenden, allerdings ist es bei diesen Werttypen auch möglich spezielle Steuerelemente zu verwenden, die auf deren Eigenheiten spezialisiert sind. Diese werden in den folgenden zwei Abschnitten beschrieben.

Boolesche Werte

Innerhalb des Surface SDKs gibt es kein spezielles Steuerelement ausschließlich zur Auswahl boolescher Werte. Wie im vorherigen Kapitel begründet, ist es aber sinnvoll dem Benutzer ein solches anzubieten, damit ihm weitere Arbeit erspart werden und er seine Ziele somit schneller erreichen kann. Daher wird im Rahmen dieser Arbeit ein neues Steuerelement auf Basis von SurfaceRadioButtons erstellt, das SurfaceBooleanSelection, welches dem Benutzer die Funktionalität zur Auswahl eines booleschen Wertes anbietet.

Numerische Werte

Prinzipiell lassen sich numerisch Werte auf dieselbe Weise auswählen, wie sie sich auch eingeben lassen. Eine eindeutige Trennung zwischen diesen beiden Interaktionen ist in diesem Fall nicht möglich. Im Rahmen der Implementierung dieser Arbeit wird aus technischen Gründen für die Auswahl von numerischen Werten wiederum die SurfaceListBox verwendet.

5.4.2. Mehrfachauswahl

Zur Realisierung einer Mehrfachauswahl gibt es grundsätzlich zwei verschiedene Alternativen. Die erste, allerdings nicht optimale Möglichkeit besteht darin mehrere Elemente einer Einzelauswahl anzuzeigen und die jeweils selektierten Werte zusammenzufassen. Beispielsweise können zur Auswahl zweier numerischer Werte zwei SurfaceSlider angezeigt werden, an denen der Benutzer verschiedene Werte setzen kann. Das Ergebnis besteht dann aus der Vereinigung der beiden Werte. Der Vorteil dieser Variante besteht darin, dass dem Benutzer eine exakte Anzahl von ausgewählten Werten vorgegeben werden kann und er nicht eine beliebige Anzahl auswählen kann. Allerdings muss dazu die Anzahl auch schon zur Entwicklungszeit feststehen und kann nicht dynamisch zur Laufzeit gesetzt werden. Zudem kann es vorkommen, dass der Bildschirm bei dieser Variante schnell überfüllt ist und der Benutzer von der enormen Anzahl an Steuerelementen die Übersicht verlieren kann.

Die zweite Möglichkeit ist die Verwendung von Steuerelementen, welche eine explizite Auswahl mehrerer Werte ermöglichen. Von den genannten Steuerelementen bietet diese Funktionalität lediglich die SurfaceListBox. Bei dieser kann über den SelectionMode zwischen der Einzel- und Mehrfachauswahl gewechselt werden. Eine Alternative zur SurfaceListBox stellt die Verwendung von SurfaceCheckBoxes dar. Dabei wird für jeden zur Auswahl stehenden Wert eine SurfaceCheckBox angezeigt und die ausgewählten Elemente entsprechen den Elementen, deren SurfaceCheckBoxes auf den Wert „wahr“ gesetzt sind.

5.4.3. Bereichsauswahl

Die Bereichsauswahl wird durch kein Steuerelement des Surface SDKs standardmäßig unterstützt. Im Rahmen dieser Arbeit wird zwar kein eigenes Steuerelement erstellt, das diese Funktionalität anbietet, aber im Folgenden wird eine mögliche Realisierung solch eines Elements beschrieben: Dieses kann aus zwei Einzelauswahl-Elementen zusammengesetzt werden, bei denen das erste den Anfangswert und das zweite den Endwert des Bereichs definiert. Dazu wird zweimal die gleiche Auswahlliste angezeigt und dem Benutzer die Auswahl der Werte ermöglicht. Intern werden anschließend alle dazwischenliegenden Werte, zusammen mit dem Anfangs- und Endwert selektiert und als Selektionsergebnis zurückgegeben. Je nach Datentyp der auswählbaren Elemente, könnte das dafür geeignete Steuerelement (SurfaceListBox oder SurfaceSlider) automatisch gewählt.

Übersicht der Auswahlelemente

Für die verwendeten Steuerelemente zur Abbildung der Auswahlelemente ergibt sich das unter Abbildung 5.10 zu sehende Schaubild.

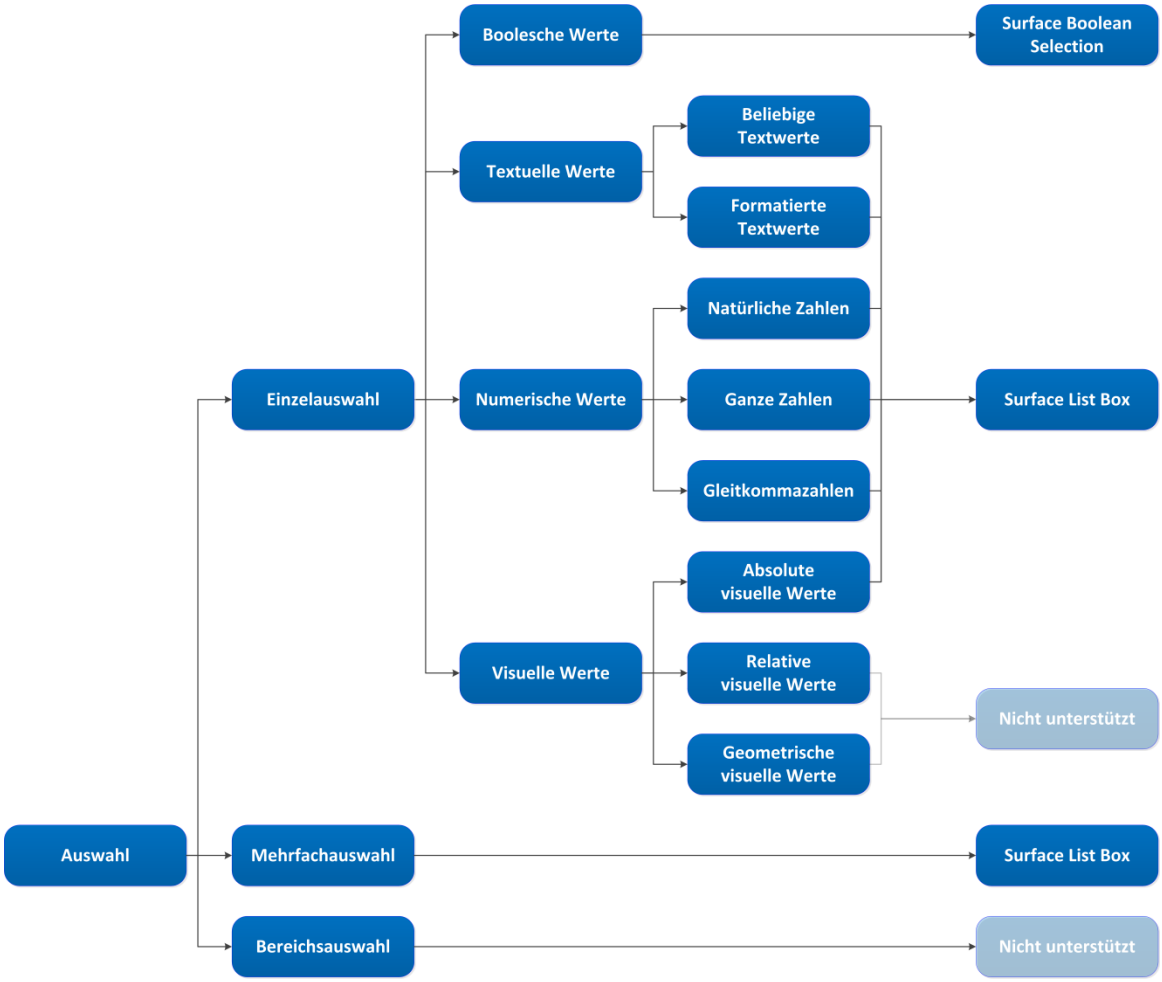


Abbildung 5.10 Übersicht der Auswahlelemente

5.5. Ausführung

Zur Ausführung von Aktionen, stehen innerhalb des Surface SDKs einige Steuerelemente zur Verfügung, die in diesem Abschnitt vorgestellt werden. Die Steuerelemente können in zwei Gruppen eingeteilt werden. Zum einen sind dies solche, die auf Schaltflächen basieren, also bei einer Berührung ein Kommando oder ein Event ausgelöst wird. Zum anderen sind dies Tags, die durch das Hinzufügen auf die Oberfläche des Surface-Tisches eine Aktion auslösen können.

Ausführung durch Schaltflächen

Für die Ausführung mit Hilfe von Schaltflächen kann natürlich der SurfaceButton verwendet werden. Diesem kann als Parameter ein Kommando übergeben werden, welches die entsprechende Aktion auslöst, sobald der Benutzer den SurfaceButton betätigt. Ein SurfaceButton ist in Abbildung 5.11 zu sehen.

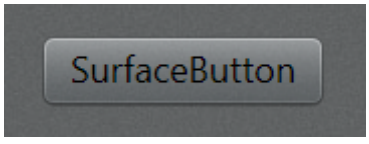


Abbildung 5.11 SurfaceButton

Falls mehrere Schaltflächen hierarchisch strukturiert werden sollen, bietet das Surface SDK neben dem SurfaceMenu, das den Standard-Windows-Menüleisten nachempfunden ist und in Abbildung 5.12 zu sehen ist, auch das ElementMenu an. Dieses bietet eine baumähnliche Struktur, welche bei Berührungen der Knoten durch den Benutzer immer weiter aufklappt. Ein ElementMenu mit den gleichen Menüeinträgen wie das gezeigte SurfaceMenu ist in Abbildung 5.13 zu sehen.

Abbildung 5.12 SurfaceMenu

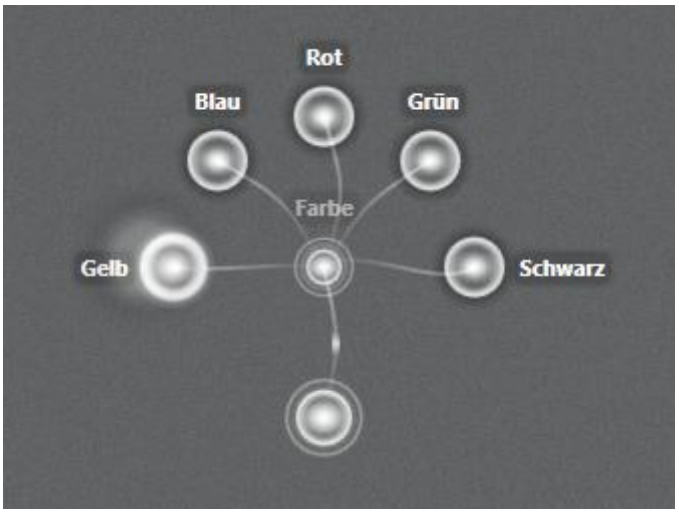


Abbildung 5.13 ElementMenu

Sowohl das SurfaceMenu als auch das ElementMenu erlauben die Zuweisung von Kommandos an die einzelnen Elemente, sodass sie als eine Art Sammlung von SurfaceButtons angesehen werden können.

Ausführung durch Tags

Als Alternative zu Schaltflächen können dank der optischen Erkennung der Surface Plattform auch Tags zum Ausführen von Funktionen verwendet werden. Da die Verwendung von Tags nicht grundsätzlich dazu ausgelegt ist, müsste diese Funktionalität durch ein Steuerelement realisiert werden, welches ähnlich wie die Schaltflä-

chen, ein Kommando auslöst, sobald ein konfigurierbares Tag auf der Oberfläche des Tisches erkannt wird.

Übersicht der Ausführungselemente

Da die Ausführungselemente unabhängig von Werttypen sind, ergibt sich für ihre Abbildung auf Steuerelemente der Microsoft Surface Plattform die in Abbildung 5.14 zu sehende Darstellung.

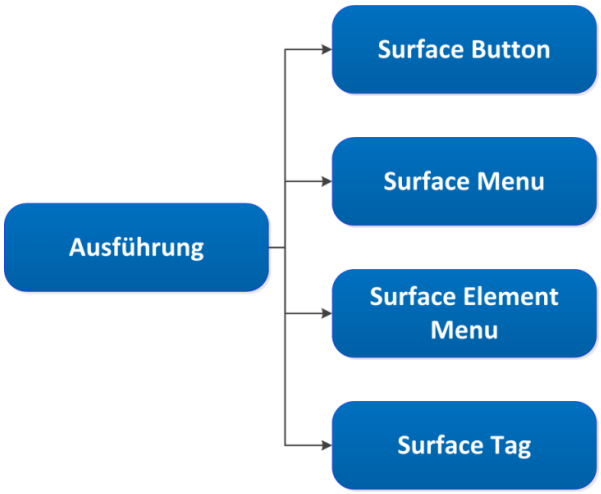


Abbildung 5.14 Übersicht der Ausführungselemente

6. Modell

Da die modellgestützte Entwicklung von Anwendungen einen entscheidenden Teil dieser Arbeit darstellt, ist es notwendig zunächst ein Modell zu definieren, auf dem die zu erstellenden Anwendungen basieren.

Ähnlich wie bei einigen in Abschnitt 2.3 beschriebenen MB-UIDEs, beispielsweise MASTERMIND [Browne, et al., 1996] und Teallach [Barclay, et al., 1999], besteht auch das in dieser Arbeit vorgestellte Modell aus den drei Teilmodellen Anwendungs-, Präsentations- und Ablaufmodell. Die Aufgaben dieser drei Teilmodelle sind:

- Anwendungsmodell
  Im Anwendungsmodell werden alle Datentypen definiert, die innerhalb der modellierten Anwendung verwendet werden können. Dazu orientiert es sich an den Konzepten der objektorientierten Programmierung zur Definition von Datentypen. Das Anwendungsmodell wird in Abschnitt 6.1 ausführlich behandelt.
- Präsentationsmodell
  Das Präsentationsmodell baut auf dem Anwendungsmodell auf und ermöglicht es, die in Kapitel 4 beschriebenen Benutzerinteraktionen in Form von Benutzersteuerelementen auch für die benutzerdefinierten Datentypen zur Verfügung zu stellen. Abschnitt 6.2 dient der genauen Beschreibung des Präsentationsmodells.
- Ablaufmodell
  Durch das Anwendungsmodell wird die Abfolge der im Präsentationsmodell definierten Benutzersteuerelemente beschrieben. Zusätzlich können Datenquellen in den Ablauf integriert werden, welche die Verwendung externer Daten ermöglichen. Das Ablaufmodell ist Gegenstand des Abschnitts 6.3.

Eine grafische Darstellung der Abhängigkeiten dieser drei Teilmodelle ist in Abbildung 6.1 zu sehen.

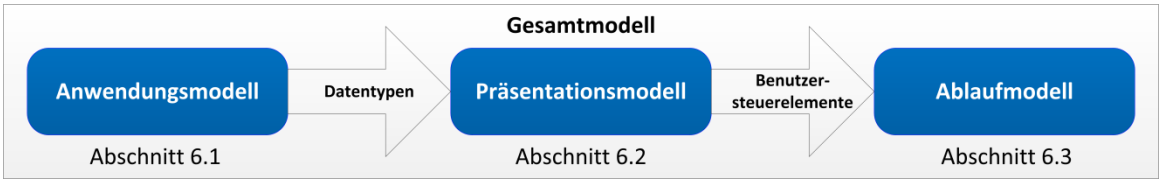


Abbildung 6.1 Übersicht des Gesamtmodells

Die Einteilung in diese drei Teilmodelle hat mehrere Gründe, die im Folgenden erläutert werden:

- Ein Ziel dieser Arbeit ist unter anderem Endbenutzern die Erstellung eigener, dialogbasierter Anwendungen für Multi-Touch-Benutzungsschnittstellen zu ermöglichen. Die Dialoge erfassen und verarbeiten dabei ganz bestimmte Daten, auf die sie zugeschnitten sind. Daher ist es notwendig diese Daten zu modellieren, wozu im vorgestellten Modell das Anwendungsmodell dient.

- Im Gegensatz zu JANUS und GENIUS, die auf Basis des Anwendungsmodells die Benutzungsschnittstelle und den Ablauf der Anwendung mit Hilfe von softwaretechnischen Erkenntnissen automatisch generieren [Schlungbaum, et al., 1996], verfolgt das in dieser Arbeit vorgestellte Modell einen anderen Ansatz. Der Benutzer des Modells soll die Erstellung der Benutzungsschnittstellen beeinflussen können, indem er beispielsweise Informationen über die Anordnung oder Strukturierung der Elemente hinterlegt. Daher war die Verwendung eines Präsentationsmodells eine logische Folgerung aus der genannten Anforderung.
- Nachdem das Anwendungsmodell der Modellierung der im System enthaltenen Daten dient und das Präsentationsmodell die Benutzungsschnittstelle beschreibt, muss schließlich noch der Ablauf der einzelnen Dialoge definiert werden. Dazu dient das Ablaufmodell.

Neben diesen enthaltenen Modellen, gibt es einige weitere, die in Abschnitt 2.2 beschrieben wurden, im Rahmen des erstellten Modells aber keine Rolle spielen. Auf das Benutzermodell wurde aus mehreren Gründen verzichtet. Zum einen wurde es aufgrund der Einfachheit nicht verwendet, da ein Endbenutzer mit der Beschreibung von verschiedenen Benutzerrollen nicht vertraut ist und die Verknüpfung der einzelnen Rollen mit den dafür vorgesehenen Elementen der Benutzungsschnittstelle eine sehr komplexe Aufgabe darstellt. Zum anderen ist eine Unterscheidung der Benutzer der erstellten Anwendungen nicht zwingend erforderlich oder gar möglich. Da es sich um Multi-User-Anwendungen handelt, müssten alle Benutzer, die gleichzeitig mit der Anwendung arbeiten auch die gleiche Rolle repräsentieren. Dies folgt unter anderem daraus, dass eine Unterscheidung der einzelnen Benutzer anhand der Kontakte nicht sinnvoll möglich ist[8].

Das beschriebene Modell zur Dialogmodellierung wird nicht gänzlich weggelassen, sondern wird implizit im Präsentationsmodell, welches die Interaktionen zwischen dem Benutzer und dem System auf Elementebene definiert, und im Ablaufmodell, welches die Interaktionsabfolge beschreibt, umgesetzt.

Da, wie bereits erwähnt, auch die Modelle von MASTERMIND und Teallach auf diesen Modellen basieren, stellt sich die Frage, warum im Rahmen dieser Arbeit ein neues Modell definiert wird und nicht die bereits vorhandenen verwendet werden. Zum einen sind die Modelle von MASTERMIND und Teallach sehr komplex und bieten teilweise mehr Funktionalität als sie benötigt wird, wie beispielsweise die Definition von Methoden auf Datentypebene, was im Rahmen eines Werkzeugs für Endbenutzer nicht angebracht ist, da dieser diese Methoden implementieren müsste. Andererseits werden einige Anforderungen nicht oder nicht komplett unterstützt,

[8] Prinzipiell wäre eine Unterscheidung der Benutzer beispielsweise anhand der Fingerabdrücke möglich, allerdings sind die Auflösungen aktueller Kameras dazu meistens nicht in der Lage und der Verwaltungsaufwand der Fingerabdrücke und der Benutzerzuordnung immens.

wie die parallele Ausführung von Aufgaben oder die Möglichkeit Multi-Touch-Anwendungen zu erstellen.

Ein weiterer Punkt, der gegen die Verwendung eines der bereits verfügbaren Modelle spricht, ist die in Kapitel 4 beschriebene Taxonomie von Benutzerinteraktionen, welche die Basis des Modells darstellt.

Eine Beschreibung der in dieser Arbeit verwendeten Modelle und dem daraus resultierenden Gesamtmodell wird in den folgenden Abschnitten gegeben.

6.1. Anwendungsmodell

Das Anwendungsmodell dient der Modellierung aller Datentypen, die innerhalb der modellierten Anwendung verwendet werden können. Als Grundlage dieses Modells dient die objektorientierte Programmierung, aus der einige der Konzepte bekannt sind, die in dem beschriebenen Anwendungsmodell verwendet werden. Dies beinhaltet unter anderem die Möglichkeit Datentypen mittels Komposition aus anderen Datentypen zusammenzusetzen, oder durch Vererbung eine hierarchische Strukturierung und damit verbundene Beziehungen zwischen Datentypen zu definieren.

Ebenso wie in vielen Programmiersprachen, existieren auch in diesem Anwendungsmodell Basisdatentypen. Diese entsprechen den in Abschnitt 4.2 vorgestellten Werttypen und sind in Abbildung 6.2 dargestellt. Diese kann der Benutzer dazu nutzen, um neue, benutzerdefinierte Datentypen zu erstellen, indem er sie zu neuen Datentypen kombiniert. Dabei besteht jeder benutzerdefinierte Datentyp aus folgenden Elementen:

- Name
  Stellt eine eindeutige Bezeichnung eines Datentyps dar. Mit Hilfe des Namens ist es sowohl dem Benutzer, als auch dem System selber möglich, den Datentyp eindeutig zu identifizieren.
- Beschreibung
  Bietet dem Benutzer die Möglichkeit eine Beschreibung des Datentyps zu hinterlegen, in der beispielsweise Informationen über die gedachte Verwendung oder Besonderheiten des Datentyps angegeben sind.
- Attribute
  Attribute stellen die einzelnen Datenfelder eines Datentyps dar. Dazu muss jedem Attribut ein eindeutiger Name, sowie ein Datentyp zugewiesen werden. Optional kann der Benutzer eine Beschreibung des Attributs angeben.
  Zur Auswahl des Datentyps eines einzelnen Attributs stehen dabei sowohl Basisdatentypen als auch benutzerdefinierte Datentypen zur Verfügung. Damit wird es dem Benutzer ermöglicht Datentypen wiederzuverwenden und zusammengesetzte Datentypen zu erstellen.

Falls der Basisdatentyp *Formatierter Textwert* als Datentyp des Attributs verwendet wird, so kann der Benutzer zudem das entsprechende Format angeben.

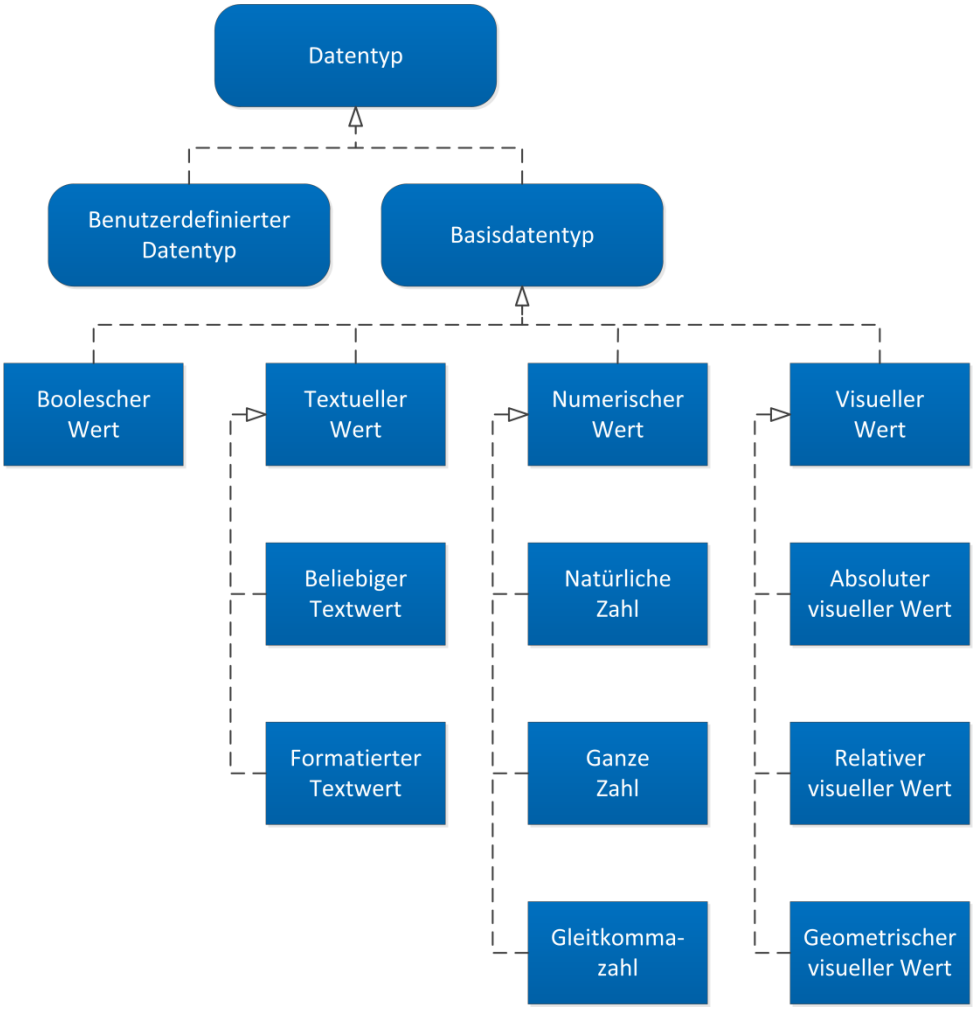


Abbildung 6.2 Basisdatentypen des Anwendungsmodells

- Supertyp
  Um die bereits erwähnte Vererbungshierarchie zu ermöglichen, verfügt jeder benutzerdefinierte Datentyp über eine Eigenschaft, die seinen Supertyp definiert. Der Supertyp muss dabei ein benutzerdefinierter Datentyp sein. Eine Mehrfachvererbung mit mehreren Supertypen ist dabei nicht vorgesehen.
  Ein Datentyp verfügt automatisch über alle Attribute, über die auch sein Supertyp verfügt, und kann diese nicht anpassen. D.h. dass sowohl die Namen, als auch die Datentypen der einzelnen Attribute durch den Supertyp vorgegeben sind. Zusätzliche Attribute können natürlich weiterhin definiert werden.
  Ist kein Supertyp gesetzt, so verfügt der Datentyp dementsprechend über keinen Supertyp und somit auch über keine vererbten Attribute.

Eine schematische Darstellung des Aufbaus eines benutzerdefinierten Datentyps ist in Abbildung 6.3 zu sehen.

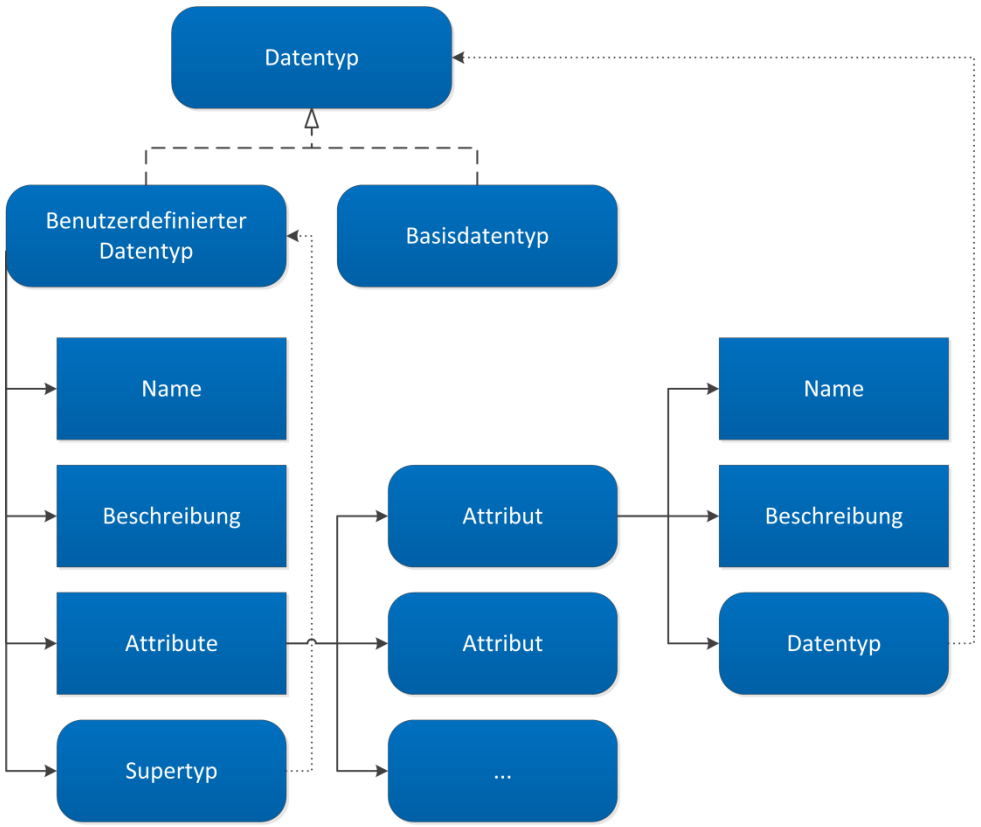


Abbildung 6.3 Schematische Darstellung eines benutzerdefinierten Datentyps

Im Gegensatz zu Teallach, welches dem Benutzer ermöglicht Methoden auf Datentypebene zu definieren [Barclay, et al., 1999], wird im Rahmen des hier vorgestellten Anwendungsmodells darauf verzichtet. Neben der erhöhten Komplexität des Modells wäre es unvermeidbar gewesen, dass die einzelnen Methoden implementiert werden müssten, was im Rahmen einer Anwendung für Endbenutzer nicht vertretbar ist.

Zum Ende dieses Abschnitts soll nun noch ein Beispiel folgen, welches sowohl die Definition eines benutzerdefinierten Datentyps als auch eines davon abgeleiteten Datentyps veranschaulicht. Das Ziel dieses Beispiels ist es einen Datentyp zu erstellen, welcher eine Person definiert. Diese Person verfügt über einen Vor- und einen Nachnamen, sowie eine Adresse, die in einem weiteren benutzerdefinierten Datentyp gekapselt ist. Die Adresse besteht dabei aus einem Straßennamen und dem Namen der Stadt.

Anschließend wird, basierend auf dem Datentyp *Person* ein weiterer, davon abgeleiteter Datentyp erstellt, der eine Geschäftsperson repräsentiert und daher zusätzlich über den Namen des Arbeitgebers verfügt.

Eine schematische Darstellung der Datentypen *Person* und der davon abgeleiteten *Geschäftsperson* ist in Abbildung 6.4 zu sehen. Dabei wurde auf die Darstellung des Datentyps *Adresse* verzichtet, da dieser entsprechen den dargestellten Datentypen definiert wird.

Name:	Person	
Beschreibung:	Repräsentiert eine Person. Enthält Informationen über den Vor- und Nachnamen, sowie die Adresse.	
Attribute:	Vorname:	Beliebiger Textwert
	Nachname:	Beliebiger Textwert
	Adresse:	Adresse
Supertyp:	keiner	

Name:	Geschäftsperson	
Beschreibung:	Repräsentiert eine Geschäftsperson. Diese basiert auf einer Person und erweitert diese um den Namen des Arbeitgebers.	
Attribute:	Arbeitgeber:	Beliebiger Textwert
Supertyp:	Person	

Abbildung 6.4 Schematische Darstellung der Datentypen *Person* und *Geschäftsperson*

Nachdem die Beschreibung des Anwendungsmodells damit vollständig ist, wird im folgenden Abschnitt das auf dem Anwendungsmodell basierende Präsentationsmodell beschrieben.

6.2. Präsentationsmodell

Das Präsentationsmodell beschreibt alle Elemente, die innerhalb der Benutzungsschnittstelle verwendet werden können. Als Grundlagen für das Präsentationsmodell dienen sowohl das im vorherigen Abschnitt vorgestellte Anwendungsmodell als auch die in Kapitel 4 beschriebenen Interaktionen.

Ziel dieses Modell ist es, für die modellierten Datentypen die beschriebenen Interaktionen Ein- und Ausgabe, Bearbeitung und Auswahl zur Verfügung stellen zu können, sowie Ausführungselemente zu definieren. Um dies zu erreichen, können die

vorhandenen Interaktionen für die Basisdatentypen zu neuen Benutzersteuerelementen kombiniert werden. Dies ähnelt sehr der Definition von benutzerdefinierten Datentypen im Anwendungsmodell, wodurch das Wissen des Benutzers, das er sich bei der Anwendungsmodellierung angeeignet hat, auch in diesem Bereich hilfreich ist.

Damit der Endbenutzer aktiv Einfluss auf die Erstellung der Benutzungsschnittstelle nehmen kann, wie es bereits in der Einleitung dieses Kapitels als Ziel dieses Modells genannt wurde, können innerhalb der Benutzersteuerelemente nicht nur die Interaktionen für die Basistypen kombiniert werden, sondern auch Elemente für die Anordnung und Strukturierung der Interaktionen, sowie bereits erstellte Benutzersteuerelemente. Die einzelnen verfügbaren Elemente werden im Folgenden beschrieben:

Interaktionselemente

Die Interaktionselemente entsprechen den in Kapitel 4 beschriebenen Benutzerinteraktionen und stehen für alle in Abschnitt 6.1 beschriebenen Basisdatentypen zur Verfügung.

Als Besonderheit dabei sind die Ausführungs-Interaktionen zu betrachten, da diese von der Implementierung des Modells abhängig sind. Je nachdem welche Optionen zur Ausführung innerhalb der Implementierung verfügbar sind, beispielsweise das Beenden einer Interaktion oder das Speichern des aktuellen Zustands, ist die Verwendung der Ausführungs-Interaktion unterschiedlich. Eine exemplarische Implementierung der Ausführungs-Interaktionen wird in Abschnitt 7.3 näher beschrieben.

Layoutelemente

Die Layoutelemente dienen der Anordnung und Strukturierung von Elementen. Dabei gibt es die folgenden Layoutelemente, die mit ihren Aufgaben und Besonderheiten aufgeführt sind:

- Stapelelemente
  Dienen der einfachen Anordnung von Elementen, die entweder horizontal oder vertikal zueinander positioniert werden können.
  Stapelelemente sind in vielen UI-Frameworks wie SWT unter dem Namen StackLayout oder in der WPF als StackPanel bekannt.
- Gitterelemente
  Innerhalb von Gitterelementen können Zeilen und Spalten definiert und die dargestellten Elemente auf die einzelnen Zellen verteilt werden. Auch Gitterelemente sind in anderen Frameworks bekannt, beispielsweise als GridLayout in SWT oder Grid in WPF.
- Andockelemente
  Durch Andockelemente können die enthaltenen Elemente an die vier Kanten sowie die Mitte des Layoutelements „angedockt“ werden. In der WPF ist ein solches Element unter dem Namen DockPanel bekannt.

- Gruppierungselemente
  Im Gegensatz zu den anderen Layoutelementen, welche die Anordnung der Elemente beeinflussen, dienen Gruppierungselemente der Strukturierung von Elementen, indem sie diese zu logischen Gruppen zusammenfassen und mit einem Titel versehen können. Gruppierungselemente können mit allen anderen Layoutelementen kombiniert werden, um die Anordnung der Elemente innerhalb des Gruppierungselements festzulegen. In der WPF wird ein solches Layoutelement als GroupBox bezeichnet.

Eine beispielhafte Darstellung der verschiedenen Layoutelemente ist in Abbildung 6.5 zu sehen.

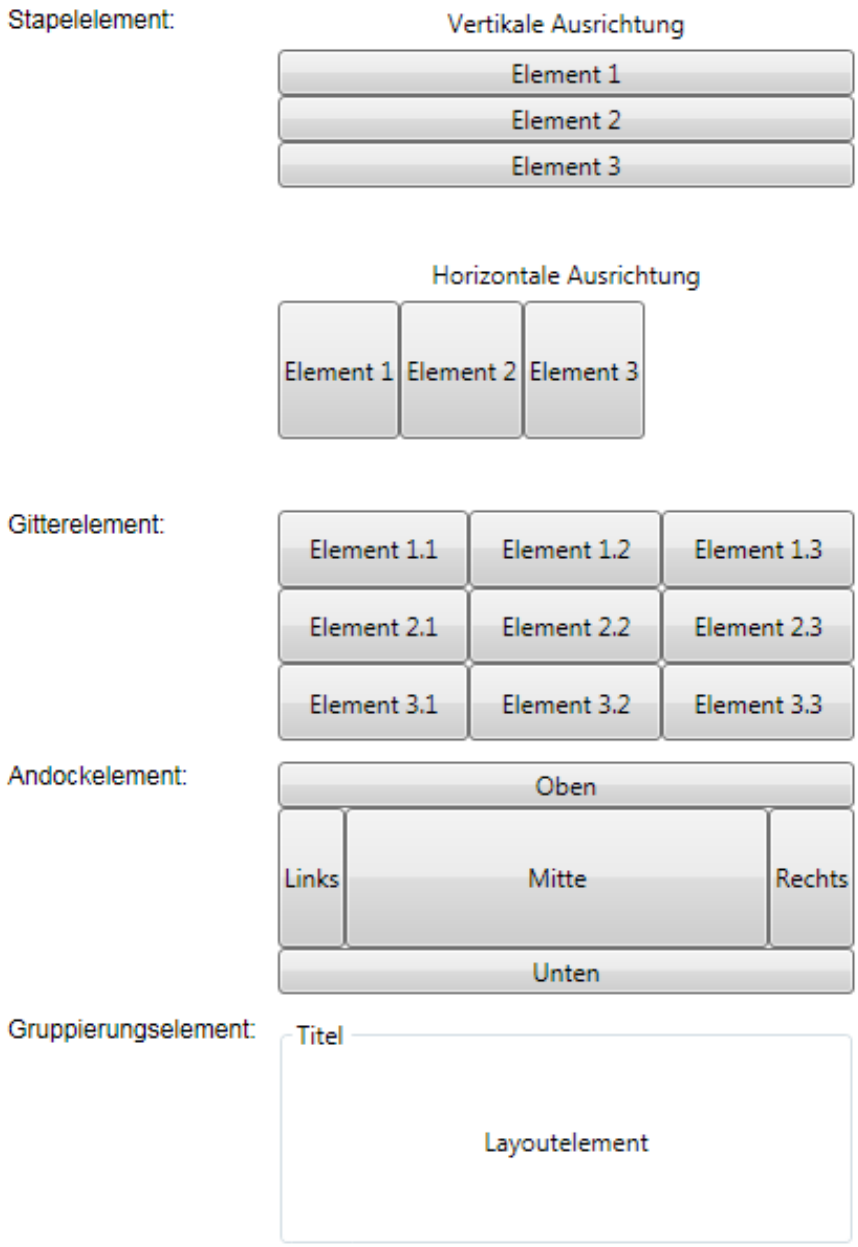


Abbildung 6.5 Übersicht der verschiedenen Layoutelemente

Innerhalb der Layoutelemente können beliebig viele andere Elemente enthalten sein. Dies können sowohl Interaktions- und Layoutelemente, als auch Benutzersteuerelemente sein.

Benutzersteuerelemente

Nachdem die vorgestellten Interaktions- und Layoutelemente feste Bestandteile des Präsentationsmodells sind, stellen die Benutzersteuerelemente jene Elemente dar, die vom Benutzer definiert werden. Diese Benutzersteuerelemente bestehen aus den folgenden Teilen:

- Name
  Der Name bietet dem Benutzer die Möglichkeit die einzelnen Benutzersteuerelemente zu benennen, wobei der Name diese nicht zwangsläufig eindeutig identifiziert.
- Durchgeführte Benutzerinteraktion
  Die durchgeführte Benutzerinteraktion legt fest, welcher der verschiedenen Interaktionstypen durch das Benutzersteuerelement realisiert wird. Dabei stehen die in Kapitel 4 beschriebenen Benutzerinteraktionen zur Verfügung.
- Verknüpfter benutzerdefinierter Datentyp
  Durch den verknüpften, benutzerdefinierten Datentyp wird definiert, für welchen Datentyp dieses Benutzersteuerelement geeignet ist.
- Inhalt
  Der Inhalt beschreibt die Elemente, die bei der Darstellung des Benutzersteuerelements verwendet werden. Dabei kann als Inhalt nur ein einzelnes Element gesetzt werden. Falls mehrere Elemente dargestellt werden sollen, so müssen diese mit Hilfe von Layoutelementen organisiert werden.

Im Zusammenhang mit dem verknüpften benutzerdefinierten Datentyp bleibt noch zu klären, wie dies unter Berücksichtigung der in Abschnitt 6.1 beschriebenen Vererbungshierarchie der Datentypen zu verstehen ist. Prinzipiell gibt es drei verschiedene Möglichkeiten, wie ein Benutzersteuerelement mit dem assoziierten benutzerdefinierten Datentyp und den damit verbundenen Supertypen in Verbindung gebracht werden kann. Diese werden im Folgenden erläutert:

- Die einfachste Möglichkeit besteht darin, dass ein Benutzersteuerelement ausschließlich für den verknüpften benutzerdefinierten Datentyp verwendet werden kann, was die Vererbungshierarchie der Datentypen allerdings ad absurdum führen würde.
- Die zweite Alternative wäre sich an der objektorientierten Programmierung anzulehnen, bei der ein Supertyp anstelle des Subtyps verwendet werden kann, wobei allerdings lediglich die Methoden und Eigenschaften des Supertyps verwendet werden können. Dies würde bedeuten, dass ein Benutzersteuerelement, das für einen gewissen Datentyp verwendet werden kann, auch für alle davon abgeleiteten Datentypen verwendet werden kann. Die

Folge dabei wäre allerdings, dass die zusätzlichen Attribute des abgeleiteten Datentyps nicht vom Benutzersteuerelement genutzt werden könnten, da diese nicht bekannt sind. Die Attribute würden bei einer Eingabe-Interaktion beispielsweise leer bleiben oder einen Standardwert einnehmen.

- Die dritte und letzte Alternative ähnelt einer Umkehrung der zweiten Möglichkeit. Im Gegensatz zur objektorientierten Programmierung könnte hier ein Subtyp anstelle des Supertyps verwendet werden. Dies würde im Fall der Benutzersteuerelemente bedeuten, dass ein Benutzersteuerelement, welches für einen bestimmten Datentyp verwendet werden kann, auch für alle Supertypen des Datentyps zur Verfügung steht. In diesem Fall wäre die Folge, dass die Attribute, die lediglich im Subtyp enthalten sind, zwar im Benutzersteuerelement verarbeitet, aber nicht an den verknüpften Datentyp weitergeleitet werden, sondern beispielsweise einfach verworfen werden.

Im Rahmen dieser Arbeit werden die letzten beiden Varianten kombiniert. Dadurch ist es möglich ein Benutzersteuerelement zur Interaktion mit einem bestimmten Datentyp auch für all dessen Sub- und Supertypen zu verwenden. Eine Beschreibung dieser Realisierung ist in Abschnitt 8.6.3 zu finden.

Ein weiterer wichtiger Punkt bei der Erstellung von Benutzersteuerelementen ist die Verbindung zwischen den dargestellten Elementen und dem verknüpften, benutzerdefinierten Datentyp. Die Art der Verbindung ist dabei abhängig von der Art des Elements. Falls es sich um ein Interaktionselement handelt, so wird die Verbindung direkt mit einem Attribut des verknüpften Datentyps hergestellt, wobei der Datentyp des Attributs ein Basisdatentyp sein muss.

Bei einem Layoutelement gibt es zwei Möglichkeiten eine Verbindung herzustellen. Zum einen kann wiederum eine Verbindung zu einem Attribut des verknüpften Datentyps definiert werden, wobei der Typ des Attributs in diesem Fall ein benutzerdefinierter Datentyp sein muss. Dadurch stehen für die Verbindungen innerhalb des Layoutelements nicht mehr die Attribute des verknüpften Datentyps des Benutzersteuerelements zur Verfügung, sondern die des Datentyps des verknüpften Attributs. Als zweite Möglichkeit kann auch keine explizite Verbindung zu einem Attribut angegeben werden, wodurch innerhalb des Layoutelements die gleichen Attribute zu Verknüpfung zur Verfügung stehen, wie außerhalb des Elements.

Als letzte mögliche Verbindung besteht die zwischen dem verknüpften Datentyp und einem anderen Benutzersteuerelement. Dieses kann nur mit einem Attribut des verknüpften Datentyps verbunden werden, mit dem es Datentyp-mäßig übereinstimmt. D.h. der Datentyp des Attributs muss identisch mit dem verknüpften Datentyp des Benutzersteuerelements sein.

Diese Art der Verbindung von Elementen mit den Daten ähnelt sehr der Datenbindung innerhalb der WPF [Marquardt, 2007], bei der die dargestellten Daten an Ei-

genschaften einer zugrundeliegenden Instanz eines Objekts, dem Datenkontext (*DataContext*), gebunden werden können.

Um die in diesem Kapitel beschriebenen Zusammenhänge besser verstehen zu können, folgt nun ein Beispiel, welches ein Benutzersteuerelement darstellt, das zur Eingabe des aus dem vorherigen Abschnitt bekannten, benutzerdefinierten Datentyps *Person* geeignet ist. Eine grafische Darstellung ist in Abbildung 6.6 zu sehen.

Zur besseren Identifikation durch den Benutzer wird dem Benutzersteuerelement zunächst ein bezeichnender Name gegeben, in diesem Fall *PersonEingabeElement*, sowie der Interaktionstyp des Elements definiert, was in diesem Beispiel einer Eingabe entspricht. Da eine Person eingegeben werden soll, wird das Benutzersteuerelement mit dem benutzerdefinierten Datentyp *Person* verknüpft. Der Übersicht halber wurde auf die Angabe der Datentypen der einzelnen Attribute des *Datentyps* Person verzichtet. Diese können Abbildung 6.4 entnommen werden.

Anschließend wird der Inhalt des Benutzersteuerelements definiert. Da mehrere andere Elemente angezeigt werden sollen, ist es zunächst notwendig ein Layoutelement zu definieren, welches die Elemente organisiert. Im gezeigten Beispiel wird dazu ein Stapelelement mit horizontaler Ausrichtung verwendet. Darin befinden sich zwei Eingabeinteraktionen, die direkt mit den Attributen *Vor*- beziehungsweise *Nachname* verbunden sind. Zudem wird dem Stapelelement ein weiteres Stapelelement hinzugefügt, diesmal mit vertikaler Ausrichtung, und mit dem Attribut *Adresse* verbunden. Innerhalb dieses inneren Stapelelements stehen zur Verbindung nun die Attribute des benutzerdefinierten Datentyps *Adresse* zur Verfügung, welche auch durch die enthaltenen Eingabeinteraktionen verwendet werden.

Alternativ wäre es auch möglich gewesen ein bereits definiertes Benutzersteuerelement anstelle des inneren Stapelelements zu verwenden, welches für die Eingabe einer Adresse geeignet ist.

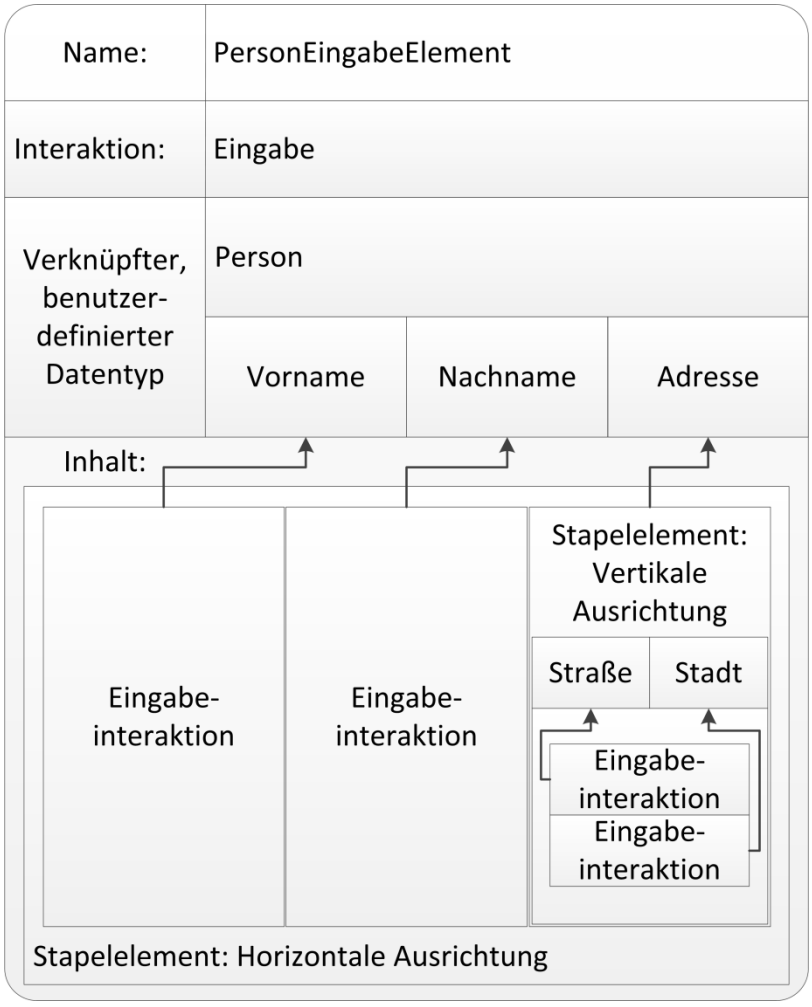


Abbildung 6.6 Beispiel eines Benutzersteuerelements zur Eingabe einer Person

Nachdem in diesem Abschnitt das Präsentationsmodell beschrieben wurde, folgt im nächsten Abschnitt die Beschreibung des Ablaufmodells, welches die Abfolge der einzelnen Benutzersteuerelemente und den Datenfluss modelliert.

6.3. Ablaufmodell

Die Aufgabe des Ablaufmodells ist es, sowohl die Abfolge der einzelnen Benutzersteuerelemente zu beschreiben, als auch den damit verbundenen Datenfluss. Dieser wird zum einen implizit durch die Folge der Benutzersteuerelemente definiert, zum anderen werden weitere Elemente beschrieben, die explizit in den Datenfluss eingreifen.

Als Vorlage für das in dieser Arbeit vorgestellte Ablaufmodell dienten dabei bekannte Standards wie die *Business Process Modeling Notation* (BMPN) [White, 2004] zur Definition von Workflows oder den Aktivitätsdiagrammen der UML [Erler, et al., 2005]. Um die Verwendung des Modells allerdings möglichst einfach zu halten, wurden nicht alle in den genannten Standards enthaltenen Elemente umgesetzt. So besteht beispielsweise nicht die Möglichkeit Ereignisse oder Gateways zu definieren oder Bedingungen für Verzweigungen zu definieren.

Zur Modellierung des Ablaufs und des damit verbundenen Datenflusses stehen innerhalb des Ablaufmodells verschiedene Elemente zur Verfügung, die im Folgenden beschrieben werden.

Aufgaben

Aufgaben repräsentieren einzelne Benutzersteuerelemente, die im Präsentationsmodell definiert wurden. Durch diese werden die Interaktionen des Benutzers mit dem System und umgekehrt modelliert. Eine Aufgab entspricht einer Aktivität in der BPMN.

Datenquellen

Durch Datenquellen können Daten aus externen Quellen, wie Datenbanken, Dateien oder dem Internet in die Anwendung gebracht werden und stehen anschließend dort zur Verfügung. Ohne Datenquellen wäre es ausschließlich durch Eingabeinteraktionen möglich neue Daten in die Anwendung zu bringen, was bei großen Datenmengen eine ziemliche Belastung des Benutzers und damit nicht akzeptabel wäre. Außerdem ist es mit Hilfe von Datenquellen ebenfalls möglich die vorhandenen Daten zu persistieren.

Verknüpfungen

Durch Verknüpfungen wird die Abfolge der einzelnen Aufgaben beschrieben. Dabei ist der Ausgangspunkt einer Verknüpfung entweder eine Aufgabe oder eine Datenquelle. Das Ziel einer Verknüpfung kann immer eine Aufgabe sein. Falls der Ausgangspunkt eine Aufgabe ist, kann das Ziel zudem eine Datenquelle sein.

Die Verwendung von Datenquellen hat mehrere Gründe. Neben der genannten Vereinfachung externe Daten innerhalb der Anwendung verwenden zu können, ist ein wichtiger Punkt der Einsatzbereich des Modells. Wie bereits in der Einleitung dieses Kapitels erwähnt, ist dies die dialogbasierte Bearbeitung von Daten. In den meisten Anwendungen werden Daten in Datenbanken oder als Dateien auf der Festplatte gespeichert. Um diese auch innerhalb der Anwendung bearbeiten und speichern zu können, wurden die Datenquellen eingeführt. Da diese Datenquellen allerdings Programmlogik erfordern, beispielsweise für den Zugriff auf die Datenbank und die Übersetzung der Daten in Datentypen, die innerhalb der modellierten Anwendung verwendet werden können, müssen diese Datenquellen im Rahmen der Realisierung des Modells definiert und implementiert werden, weshalb eine genauere Beschreibung an dieser Stelle ausbleibt. Eine beispielhafte Implementierung einer Datenquelle wird in Abschnitt 7.4 vorgestellt.

Nachdem die einzelnen Elemente des Ablaufmodells beschrieben sind, muss noch deren Semantik beschrieben werden. Um dies besser verstehen zu können, ist in Abbildung 6.7 eine Grafik zu sehen, welche ein beispielhaftes Ablaufmodell visualisiert.

In dieser Grafik sind alle Bestandteile eines Ablaufmodells zu sehen. Aufgaben werden durch blaue Rechtecke dargestellt, Datenquellen durch weiße Kreise und Verknüpfungen durch Pfeile zwischen den einzelnen Elementen.

Anhand dieser Grafik wird zunächst die Abfolge der einzelnen Elemente erläutert. Prinzipiell wird eine Aufgabe dann begonnen, wenn alle vorherigen Aufgaben vollständig ausgeführt wurden. Im dargestellten Beispiel wird Aufgabe 2 begonnen, sobald Aufgabe 1.1 beendet wurde. Aufgabe 3 hingegen beginnt erst, sobald Aufgabe 2 und Aufgabe 1.2 vollendet wurde. Zudem ist es möglich, dass nach Beendigung einer Aufgabe mehrere andere Aufgaben beginnen können. Im gezeigten Beispiel beginnen Aufgabe 4.1 und Aufgabe 4.2 sobald Aufgabe 3 beendet ist. Dadurch ist eine parallele Ausführung von Aufgaben möglich. Durch die Verwendung einer Multi-Touch-Technologie können parallele Aufgaben tatsächlich parallel und durch mehrere Benutzer gleichzeitig bearbeitet werden.

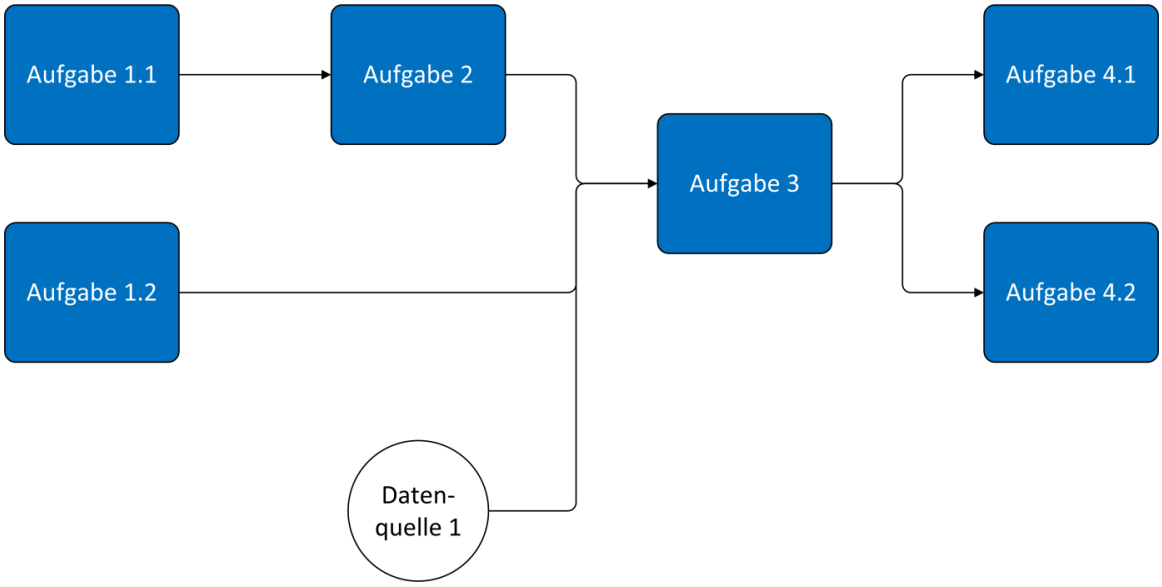


Abbildung 6.7 Beispiel eines Ablaufmodells

Neben der Abfolge der Elemente, ist der zweite wichtige Aspekt des Ablaufmodells der Datenfluss. Dieser wird zum einen implizit durch die Verknüpfungen geregelt, zum anderen durch die Datenquellen.

Abhängig vom Interaktionstyp einer Aufgabe ist auch der Datenfluss unterschiedlich. Dabei gelten die folgenden Regeln:

- Eine Eingabe-Interaktion erzeugt eine Instanz des verknüpften, benutzerdefinierten Datentyps. Diesen gibt sie an alle Aufgaben weiter, die mit der Eingabe-Interaktion verknüpft sind. Würde es sich bei der Aufgabe 1.1 im vorherigen Beispiel um eine Eingabe-Interaktion für den Datentyp *Person* handeln, so würde die eingegebene Person an Aufgabe 2 weitergegeben werden.
  Alle Dateninstanzen, die an eine Eingabe-Interaktion übergeben werden, werden von dieser nicht berücksichtigt und daher verworfen.

- Eine Ausgabe-Interaktion erwartet eine Instanz des verknüpften, benutzerdefinierten Datentyps als Eingangsparameter zu bekommen. Im Gegensatz zur Eingabe-Interaktion wird bei einer Ausgabe-Interaktion keine Datentypinstanz weitergeleitet.
- Eine Bearbeitungs-Interaktion kombiniert in gewisser Weise die verschiedenen Verhaltensweisen der Ein- und Ausgabe-Interaktionen. Sie erwartet eine Instanz des Datentyps als Eingangsparameter und gibt diesen nach Abschluss der Bearbeitung an alle folgenden Aufgaben weiter.
- Die letzte relevante Interaktion, die Auswahl, hat ein ähnliches Verhalten. Allerdings erwartet diese als Eingabe eine Menge von Instanzen des verknüpften, benutzerdefinierten Datentyps. Nach Beendigung der Auswahl, wird abhängig davon, ob es sich um eine Einzel-, Mehrfach- oder Bereichsauswahl gehandelt hat, entweder ein oder mehrere ausgewählte Elemente an die folgenden Aufgaben weitergeleitet.

Eine weitere Frage, die geklärt werden muss, ist das Verhalten bei mehreren Eingangsparametern. Dies ist in der gezeigten Abbildung beispielsweise bei Aufgabe 3 der Fall, die sowohl Eingangsparameter von Aufgabe 1.2 und Aufgabe 2, als auch von der Datenquelle 1 bekommt. Dabei ist zu unterscheiden, ob die Zielaufgabe mehrere Datensätze verarbeiten kann, beispielsweise um einen daraus auszuwählen, oder ob sie einen einzelnen Datensatz erwartet, zum Beispiel um diesen zu bearbeiten. Im ersten Fall können alle Datensätze zu einer Liste kombiniert und diese an die folgende Aufgabe weitergeleitet werden. Dabei ist allerdings zu beachten, dass die einzelnen Ausgangsparameter vom gleichen Datentyp sind und mit dem erwarteten Datentyp des Eingangsparameters der folgenden Aufgabe übereinstimmen.

Im zweiten Fall muss einer der Eingangsparameter ausgewählt werden, der an die folgende Aufgabe übergeben wird. Wie dies geschieht, bleibt den einzelnen Implementierungen des Modells überlassen, da alle Möglichkeiten vor und Nachteile bieten, die von der Verwendung der Implementierung abhängen.

7. Architektur

Dieses Kapitel beschreibt die grundlegende Architektur einer exemplarischen Implementierung des im vorherigen Kapitel vorgestellten Modells zur Beschreibung von Multi-Touch-Anwendungen durch Endbenutzer. Dabei werden nicht alle Aspekte des Modells vollständig implementiert, wobei auf fehlende Bestandteile im Rahmen der Beschreibungen der einzelnen Komponenten hingewiesen wird.

In den folgenden Abschnitten wird neben einer Übersicht, welche die einzelnen Komponenten des Gesamtsystems vorstellt und die Verbindungen zwischen diesen Komponenten beschreibt, auch separat auf die Komponenten eingegangen und deren interne Architektur vorgestellt.

7.1. Übersicht der Komponenten

Die vorgestellte Architektur besteht im Kern aus zwei Komponenten, die in Abbildung 7.1 zu sehen sind.

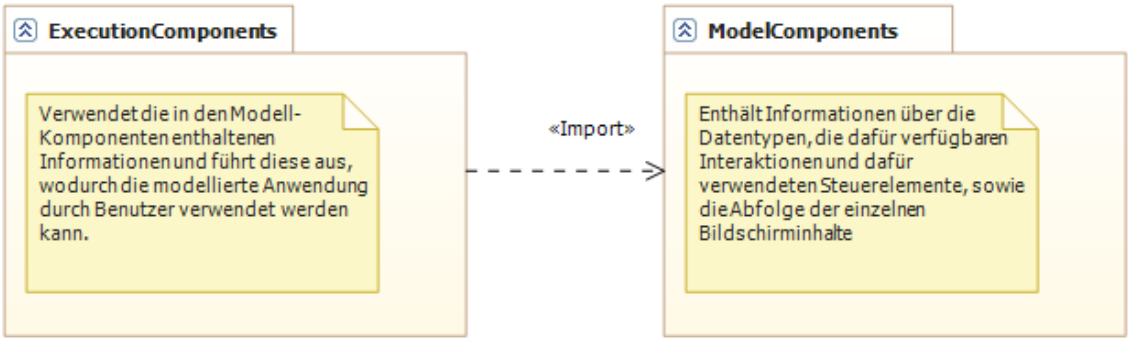


Abbildung 7.1 Übersicht der Komponenten

Dabei repräsentieren die Modell-Komponenten (*ModelComponents*) eine beispielhafte Implementierung des zugrundeliegenden Modells, das in Kapitel 6 vorgestellt wurde. Sie erhalten also die Informationen über Datentypen, Benutzersteuerelemente und Abläufe. Die Ausführungskomponente (*ExecutionComponents*) hingegen ist für das Ausführen des vollständigen Modells zuständig. Diese generiert aus dem Präsentationsmodell die Benutzungsoberfläche, erzeugt die Objekte auf welche die einzelnen Steuerelemente zugreifen und organisiert den Ablauf der einzelnen Bildschirminhalte, sowie die Behandlung der Ausführungsinteraktionen. Eine ausführliche Beschreibung der Ausführungskomponente ist in Abschnitt 7.5 zu finden.

Im Folgenden wird der genaue Aufbau der Modell-Komponenten beschrieben. Diese bestehen aus den drei, im Modell vorgegebenen Teilen Anwendungsmodell, Präsentationsmodell und Ablaufmodell. Die Abhängigkeiten zwischen diesen drei Modellbestandteilen sind in Abbildung 7.2 zu sehen.

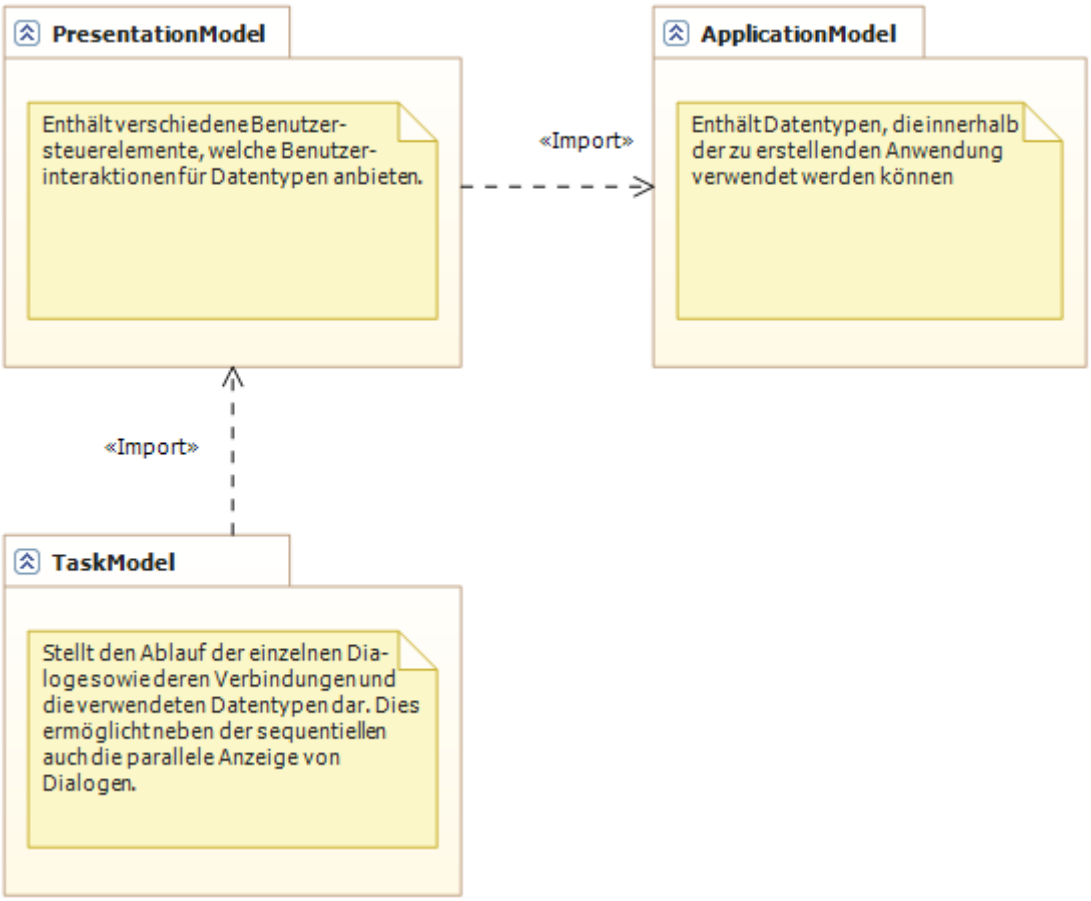


Abbildung 7.2 Übersicht der Modell-Komponenten

Grundlegend implementiert jede dieser drei Komponenten eines der in den Abschnitten 6.1 bis 6.3 beschriebenen Modelle. Dementsprechend orientieren sich auch ihre Aufgaben und Zuständigkeiten an den entsprechenden Vorgaben und werden im Folgenden noch einmal in Erinnerung gerufen:

ApplicationModel

Das *ApplicationModel* implementiert eine beispielhafte Umsetzung des in Abschnitt 6.1 beschriebenen Anwendungsmodells. Es enthält alle Datentypen, die innerhalb der zu erzeugenden Anwendung verwendet werden können. Dies sind neben einigen Basisdatentypen auch alle Datentypen, die der Endbenutzer selber erzeugt hat. Dabei kann er sowohl neue Datentypen aus bereits erstellten oder standardmäßig vorhandenen Basisdatentypen zusammenstellen, oder bereits erstellte Datentypen mittels Vererbung erweitern. Das *ApplicationModel* wird in Abschnitt 7.2 ausführlich betrachtet.

PresentationModel

Das *PresentationModel*, welches das in Abschnitt 6.2 aufgeführte Präsentationsmodell implementiert, bietet dem Benutzer die Möglichkeit Steuerelemente zu erstellen, welche für jeweils eine der in Kapitel 4 genannten Interaktionen in Kombination mit dem entsprechenden Datentyp geeignet sind. Dabei können die in Kapitel 5 er-

läuterten Interaktionen für die Basisdatentypen zu neuen Benutzersteuerelementen für benutzerdefinierte Datentypen kombiniert werden, die anschließend wieder innerhalb anderer Benutzersteuerelemente verwendet werden können. Eine genaue Beschreibung des *PresentationModel* ist in Abschnitt 7.3 zu finden.

TaskModel

Das in Abschnitt 6.3 beschriebene Ablaufmodell, wird durch das TaskModel implementiert. Es basiert auf dem *PresentationModel* und dient der Modellierung des Programmablaufs. Dabei wird eine Abfolge von Benutzersteuerelementen definiert, die entweder sequentiell oder parallel ausgeführt werden können. Dabei werden zum einen die Datentypen des Anwendungsmodells, zum anderen die Steuerelemente des Präsentationsmodells verwendet.

Die parallele Ausführung ist eine beispielhafte Implementierung zur Unterstützung einer Multi-Touch- und Multi-User-Umgebung, die durch die Microsoft Surface Plattform bereitgestellt wird. Ausführlichere Informationen zum *TaskModel* können Kapitel 7.4 entnommen werden.

Der Aufbau der einzelnen Komponenten des Systems wird in den folgenden Abschnitten beschrieben.

7.2. Das ApplicationModel

Die Aufgabe des *ApplicationModel* besteht darin, die innerhalb des gesamten Modells verwendeten Datentypen zu definieren. Dabei stehen dem Benutzer einige Basisdatentypen zur Verfügung, die *ValueTypes*, welche den in Abschnitt 6.1 behandelten Basisdatentypen des Modells entsprechen. Diese können zu neuen, benutzerdefinierten Datentypen zusammengefasst werden, ähnlich wie es in objektorientierten Programmiersprachen der Fall ist. Innerhalb der benutzerdefinierten Datentypen, können neben den Basisdatentypen auch weitere, vorher vom Benutzer definierte Datentypen verwendet werden. Ein Beispiel eines benutzerdefinierten Datentyps ist in Abbildung 7.3 zu sehen. Dieses Beispiel ist dem in Abschnitt 6.1 behandelten Beispiels eines benutzerdefinierten Datentyps nachempfunden und verwendet eine vereinfachte Darstellung.

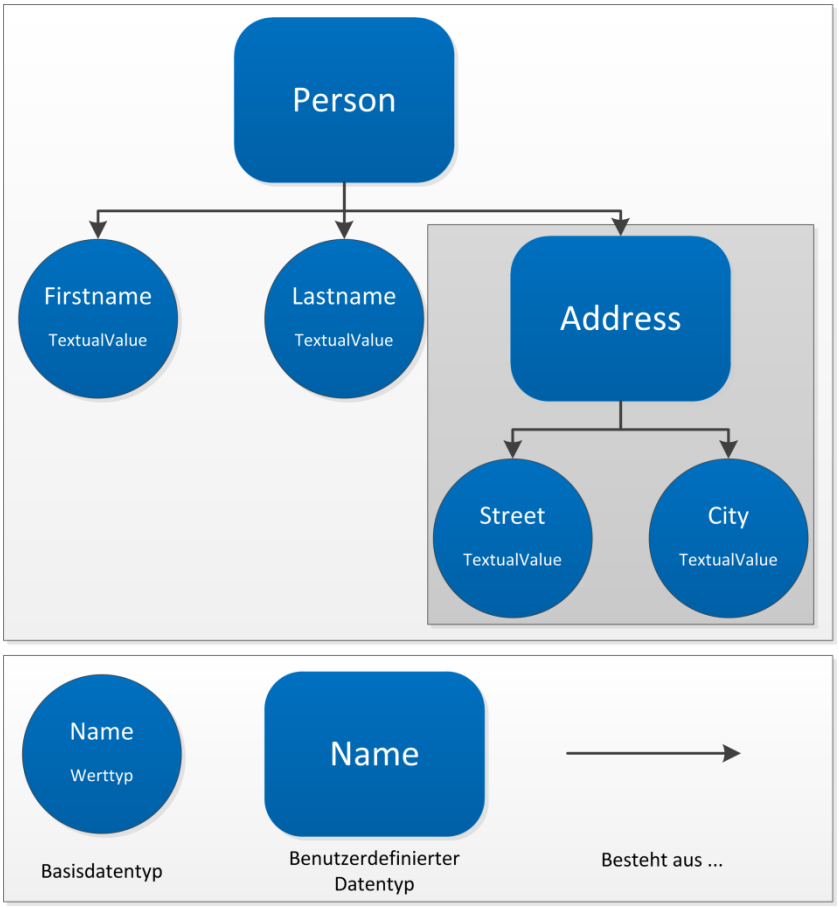


Abbildung 7.3 Beispiel eines zusammengesetzten, benutzerdefinierten Datentyps

Dieses beschreibt die Zusammensetzung des Datentyps *Person*, welcher aus den beiden Attributen *Firstname* und *Lastname* des Basisdatentyps *TextualValue*, sowie dem weiteren, benutzerdefinierten Datentyp *Address* besteht, welcher wiederrum die zwei Attribute *Street* und *City* enthält, die ebenfalls vom Basisdatentyp *TextualValue* sind.

Neben dieser Zusammenstellung von Datentypen aus Basis- und benutzerdefinierten Datentypen, besteht im Rahmen des Modells die Möglichkeit eine Vererbungshierarchie zu definieren. Dadurch können Spezialisierungen von bereits bestehenden, benutzerdefinierten Datentypen erstellt werden. Ein Beispiel dafür wäre ein, von dem bereits bekannten Datentyp *Person* abgeleiteter Datentyp *BusinessPerson*, der zusätzlich zu den Attributen, die durch den Datentyp *Person* definiert werden (*Firstname*, *Lastname* und *Address*), ein weiteres Attribut für den Arbeitgeber einer Geschäftsperson enthält. Dies ist in Abbildung 7.4 dargestellt.

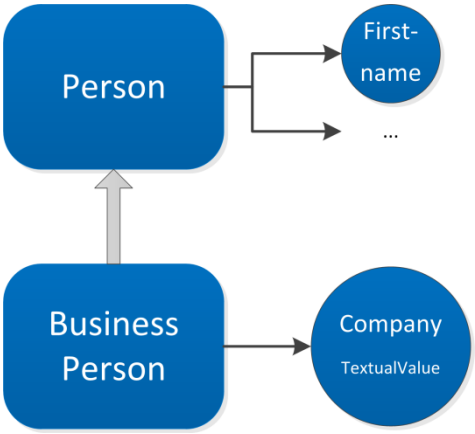


Abbildung 7.4 Beispiel von Vererbung bei benutzerdefinierten Datentypen

Um diese beschriebenen Möglichkeiten zu realisieren, besteht das Anwendungsmodell aus sechs Teilen. Diese wurden als Interfaces definiert, um Verwendung des Anwendungsmodells von dessen realer Implementierung zu entkoppeln. Eine Übersicht des Anwendungsmodells ist in Abbildung 7.5 als UML-Diagramm zu sehen, die darin enthaltenen Elemente werden im Folgenden beschrieben.

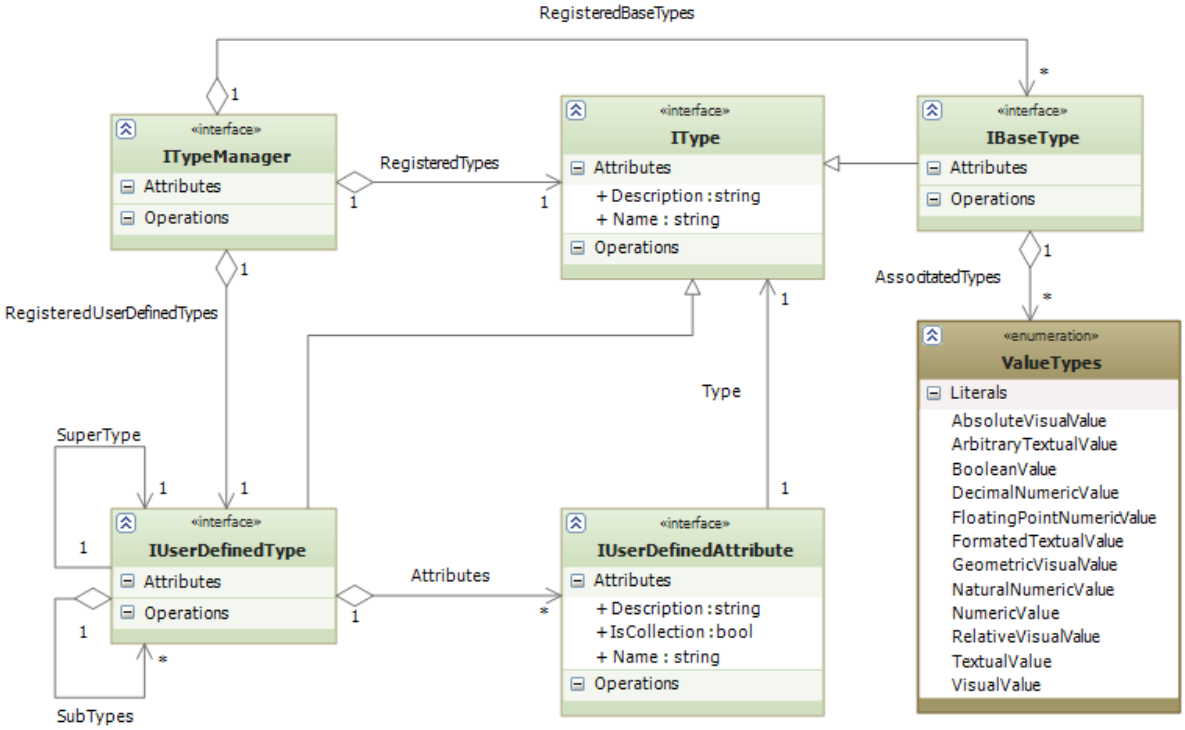


Abbildung 7.5 UML-Diagramm des Anwendungsmodells

ValueTypes

Die Enumeration *ValueTypes* entspricht den in Abschnitt 6.14.2 beschriebenen Basisdatentypen, die innerhalb des Modells zur Verfügung stehen. Diese sind der Einfachheit halber als Enumeration und nicht als Vererbungshierarchie beschrieben, auch wenn dies in einer späteren Ausbaustufe mit mehr Werttypen eventuell sinnvoll wäre.

IType

Dieses Interface stellt sowohl die Grundlage für Basisdatentypen, als auch für benutzerdefinierte Datentypen dar. Es enthält neben dem Namen des Datentyps auch eine Beschreibung, wofür dieser verwendet werden kann.

IBaseType

Repräsentiert die Basisdatentypen, die innerhalb des Modells zur Verfügung stehen. Dazu enthält es eine Aufzählung aller *ValueTypes*, die mit dem entsprechenden Basisdatentyp verknüpft sind. Dies ist notwendig, da die Basisdatentypen des Modells hierarchisch strukturiert sind und daher zu einem konkreten Basisdatentyp mehrere Basisdatentypen gehören können. Dies ist beispielsweise bei den allgemeinen textuellen Werten der Fall, die sowohl beliebige als auch formatierte Textwerte wiederspiegeln.

IUserDefinedType

Durch das Interface IUserDefinedType werden alle Datentypen beschrieben, die vom Benutzer erstellt wurden. Für die Realisierung der bereits genannten Vererbungshierarchie enthält jeder benutzerdefinierte Datentyp sowohl einen Verweis auf den Supertyp, den er erweitert, als auch eine Liste aller Subtypen, die von ihm abgeleitet sind.

Die Datentypen, aus denen der benutzerdefinierte Datentyp zusammengesetzt ist, werden in einer Liste von benutzerdefinierten Attributen verwaltet.

IUserDefinedAttribute

Die genannten, vom Benutzer definierten Attribute, die innerhalb der benutzerdefinierten Datentypen verwendet werden können, werden durch dieses Interface beschrieben. Neben einer Beschreibung und einem Namen enthält das Attribut auch die Informationen, von welchem Datentyp es ist, sowie eine Angabe, ob es sich bei dem Attribut um eine Aufzählung des gewählten Datentyps handelt oder es einen einzelnen Wert dieses Typs darstellt.

Falls als Datentyp der Wert *FormattedTextualValue* gewählt wird, so kann der Benutzer zusätzlich das zugrundeliegende Format auswählen. Er kann dabei aus den drei vordefinierten Formatvorlagen für ein Datum, eine IP-Adresse oder eine Email-Adresse wählen, oder das Format selber definieren.

ITypeManager

Der TypeManager verwaltet alle Datentypen, die innerhalb des Modells zur Verfügung stehen. Dies sind neben den Basisdatentypen hauptsächlich die benutzerdefinierten Datentypen. Dazu stellt er neben den Aufzählungen, welche die Datentypen enthalten, auch Methoden zum Hinzufügen und Entfernen von neuen Datentypen zur Verfügung.

Zur Bearbeitung des Anwendungsmodells durch den Benutzer wird der in Abschnitt 8.3 beschriebene Typeneditor verwendet.

Damit ist die Beschreibung des Anwendungsmodells abgeschlossen, welches die Grundlage für das Präsentations- und Ablaufmodell sowie die Ausführungskomponente bildet.

7.3. Das PresentationModel

Der umfangreichste Bestandteil der Modell-Komponenten ist das *PresentationModel*. Es implementiert das in Abschnitt 6.2 behandelte Präsentationsmodell und dient der Modellierung von Benutzersteuerelementen, welche Interaktionen für verschiedene Datentypen bereitstellen.

Dazu können die vorhandenen Interaktionen für die Basisdatentypen zu neuen Benutzersteuerelementen kombiniert werden und damit die Interaktionen auch für benutzerdefinierte Datentypen zur Verfügung stellen.

Beim Erstellen der Steuerelemente wird zunächst der zugrundeliegende benutzerdefinierte Datentyp, sowie die Interaktion definiert, die durch dieses Steuerelement realisiert werden soll. Anschließend können Layoutcontainer, Interaktionen für die Basisdatentypen und bereits erstellte Steuerelemente zusammengesetzt werden.

Um dies anschaulicher zu gestalten, greifen wir auf die im vorherigen Abschnitt beschriebenen Datentypen *Person* und *Address* zurück. Möchte der Benutzer ein neues Steuerelement zur Eingabe einer Person erstellen, so wählt er als Datentyp des zu erstellenden Steuerelements den Typ *Person* aus und setzt den Interaktionstyp auf den Wert *Eingabe* (*Input*). Anschließend wählt er einen der verfügbaren Layoutcontainer aus und kombiniert darin Eingabeinteraktionen für die Attribute *First*- und *Lastname* und für den enthaltenen Datentyp *Address*. Eine grafische Darstellung des Weges vom Datentyp zum Benutzersteuerelement für die Eingabe einer Person ist in Abbildung 7.6 zu sehen.

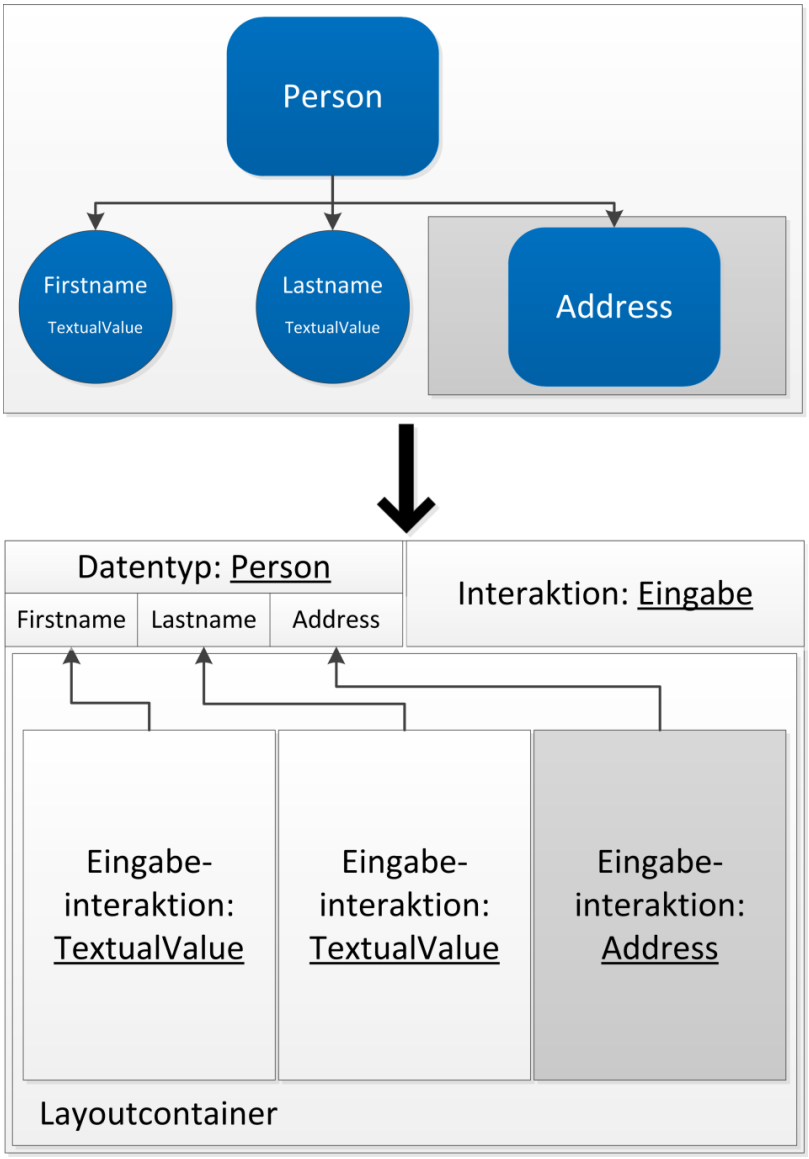


Abbildung 7.6 Steuerelement zur Eingabeinteraktion für eine Person

Da die Datentypen sowie die Interaktionstypen die Grundlage für alle Steuerelemente darstellen, ist es prinzipiell möglich diese automatisch generieren zu lassen. Da alle benutzerdefinierten Datentypen auf eine Sammlung von Basisdatentypen zurückzuführen sind und für diese die entsprechenden Steuerelemente innerhalb des Systems bekannt sind, könnten diese vollautomatisch zu den Benutzersteuerelementen für die zusammengesetzten Datentypen erzeugt werden. Innerhalb dieser Arbeit wird diese Funktion allerdings nicht umgesetzt, auf ihr aufbauend wäre eine Realisierung allerdings mit mittlerem Aufwand durchführbar.

Bedingt durch die Vererbungshierarchie der benutzerdefinierten Datentypen, kann diese auch auf die Benutzersteuerelemente übertragen werden. Die Möglichkeiten, wie dies realisiert werden kann, sind in Abschnitt 6.2 unter dem Punkt Benutzersteuerelemente aufgeführt. Im Rahmen dieser Arbeit werden dabei die zwei zuletzt aufgeführten Möglichkeiten, mit den entsprechenden Vor- und Nachteilen unterstützt. Konkret heißt dies, dass Benutzersteuerelemente, die für einen bestimmten,

benutzerdefinierten Datentyp geeignet sind, sowohl für alle Subtypen als auch für alle Supertypen dieses Datentyps verwendet werden können. Dadurch ist es beispielsweise möglich, dass ein Benutzersteuerelement, das zur Eingabe eines Objekts vom Datentyp *Person* auch für solche vom Typ *BusinessPerson* verwendet werden kann. Andersrum können Benutzersteuerelemente zur Eingabe von *BusinessPerson*-Objekte auch für *Person*-Objekte verwendet werden. Eine ausführliche Beschreibung, wie dies erreicht werden kann, wird in Abschnitt 8.6.3 aufgezeigt.

Das UML-Klassendiagramm in Abbildung 7.7 zeigt die Architektur des Präsentationsmodells. Dieses besteht hauptsächlich aus den Elementen *IBaseInterfaceObjec*,t *IContainerObject*, *IInteractionObject*, *IUserControlObject* und dem *IInterfaceManager*, welcher der Verwaltung der Benutzersteuerelemente dient. Die Aufgaben dieser Elemente werden im Folgenden vorgestellt.

IBaseInterfaceObject

Dieses Interface dient als Basiselement für alle Elemente, die innerhalb der modellierten Benutzungsschnittstelle dargestellt werden können. Dies umfasst alle folgenden Elemente mit Ausnahme des *IInterfaceManager*.

Der *DataContext* dieses Elements beschreibt dabei den Datentyp, mit dem das entsprechende Element verknüpft ist und auf den sich die Verknüpfungen innerhalb der Benutzersteuerelemente beziehen.

IContainerObject

Diese Elemente entsprechen den, aus Abschnitt 6.2 bekannten, Layoutelementen und dienen dazu mehrere andere Elemente zu gruppieren. Von den beschriebenen Layoutelementen werden im Rahmen dieser Implementierung lediglich die Stapel- und Gruppierungselemente umgesetzt. Diese werden durch die Elemente *GroupContainer* und *StackContainer* der Enumeration *ContainerType* definiert. Die Eigenschaft *Header* enthält den Titel des Gruppierungselements, falls dies als Container gewählt ist.

IInteractionObject

Die von *IInteractionObject* abgeleiteten Typen entsprechen den in Abschnitt 6.2 beschriebenen Interaktionselementen, die auf den Basisdatentypen des Modells aufbauen. Das bei allen Interaktionselementen, mit Ausnahme des *ICommandInteractionObject*, vorhandene Attribut *DisplayedAttribute* beinhaltet die Information darüber, welches Attribut des *DataContext* von diesem Element verarbeitet werden soll. Da die Ausführungs-Interaktionen nicht von einem bestimmten Datentyp abhängen, verfügen diese nicht über diese Verbindung zum *DataContext*.

Die *ICommandInteractionObject*-Elemente verfügen über einen *CommandType*, der angibt um was für eine Ausführungsinteraktion es sich bei diesem Element handelt. Im Rahmen dieser Arbeit werden drei verschiedene Möglichkeiten zur Verfügung gestellt, die entweder die aktuelle Aufgabe, siehe Abschnitte 6.3 und 7.4, die gesamte

Ausführung beenden, oder im Rahmen eines Auswahlinteraktion die Möglichkeit zur zufälligen Auswahl eines Elements ermöglichen.

IUserControlObject

Die in Abschnitt 6.2 beschriebenen Benutzersteuerelemente werden durch das Interface *IUserControlObject* repräsentiert und bilden den Kern des *PresentationModel*. Durch diese kann der Benutzer Interaktionen für selbstdefinierte Datentypen erstellen, die im späteren Verlauf der Modellierung zur Verfügung stehen.

Eine Besonderheit stellen die Benutzersteuerelemente dar, die zur Auswahl eines Elements dienen. Diese *ISelectionInteractionObject*-Elemente, können im Rahmen der vorgestellten Implementierung ausschließlich direkt als Inhalt eines Benutzersteuerelements und nicht innerhalb von anderen *IContainer*-Objekten verwendet werden. Zudem steht in der aktuellen Implementierung ausschließlich der Auswahlmodus *SingleSelection* zur Verfügung.

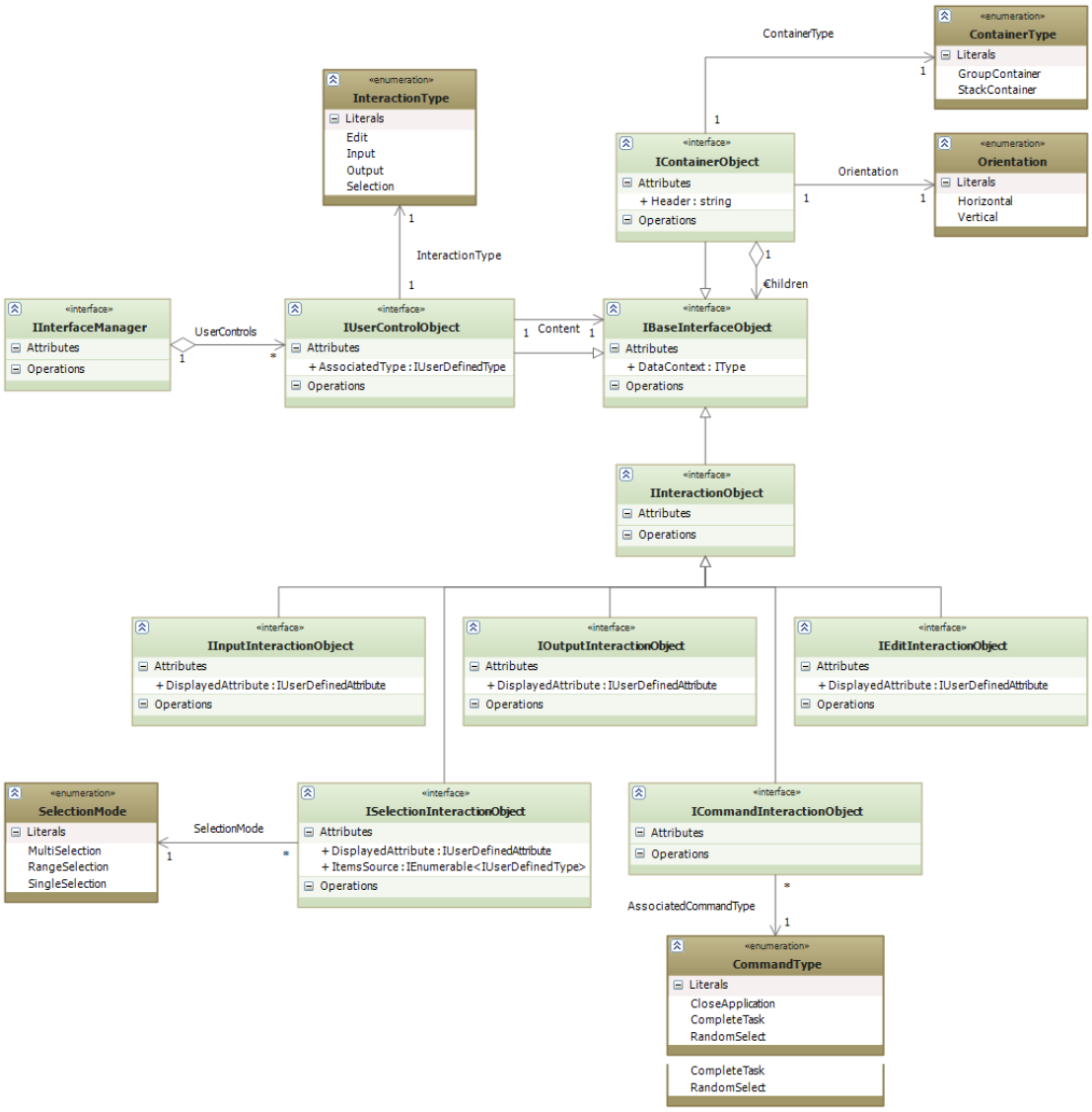


Abbildung 7.7 UML-Diagramm des Präsentationsmodells

IInterfaceManager

Ähnlich dem ApplicationModel gibt es auch beim PresentationModel eine Verwaltungsinstanz, den *IInterfaceManager*, welcher alle erstellten Benutzersteuerelemente verwaltet.

Dem Benutzer wird die Bearbeitung des Präsentationsmodells durch den in Abschnitt 8.4 beschriebenen Oberflächeneditor ermöglicht.

Die modellierten Benutzungssteuerelemente bilden schließlich die Basis für den letzten Abschnitt der Modellierung, in dem der Ablauf der einzelnen Elemente definiert wird.

7.4. Das TaskModel

Nachdem in den vorherigen beiden Abschnitten die Realisierungen des Anwendungs- und Präsentationsmodells vorgestellt wurden, folgt nun, als letzter Bestandteil der Modell-Komponenten, die Realisierung des aus Abschnitt 6.2 bekannten Ablaufmodells. Dieses wird durch das TaskModel implementiert und dient dazu den Ablauf der einzelnen Interaktionen auf Basis der Benutzersteuerelemente zu modellieren. Dazu werden die gewünschten Benutzersteuerelemente ausgewählt und ihre Abfolge festgelegt.

Wie die Vorgabe aus dem Ablaufmodell, unterstützt auch die Implementierung die parallele Ausführung von Interaktionen. Dadurch wird sowohl die Multi-Touch- als auch Multi-User-Fähigkeit der Microsoft Surface Plattform genutzt und ermöglicht das gleichzeitige Arbeiten mehrerer Benutzer mit der erstellten Software.

Das UML-Diagramm des Ausführungsmodells ist in Abbildung 7.8 zu sehen. Es besteht im Wesentlichen aus den verschiedenen Aufgaben (*ITask*) sowie deren Verwaltung (*ITaskManager*). Zudem existiert, zur Implementierung einer Datenquelle, das *IDataSource*-Interface. Die Aufgaben dieser Elemente sind:

ITask

Die ITask-Objekte repräsentieren die einzelnen Aufgaben innerhalb des Ablaufmodells. Dazu enthalten sie einen Verweis auf das Benutzersteuerelement, das im Rahmen der Aufgabe angezeigt werden soll. Zur Modellierung der Abfolge der einzelnen Aufgaben, enthält jede Aufgabe Informationen über die nach ihr folgenden Aufgaben, sowie die Aufgaben die abgeschlossen sein müssen, damit die Aufgabe begonnen werden kann.

ITaskManager

Ebenso wie die Verwaltungselemente im Anwendungs- und Präsentationsmodell, dient auch dieses der Verwaltung der vorhandenen Aufgaben.

IDataSource

Dieses Interface repräsentiert die beschriebenen Datenquellen. Dazu enthält es Referenzen auf alle *ITask*-Objekte, bei denen es als Eingangsparameter verwendet wird. Im Rahmen dieser Implementierung wird eine beispielhafte Datenquelle bereitgestellt, welche eine Liste von Personen erzeugt.

Ein Beispiel für die Modellierung eines einfachen Ablaufs ist in Abbildung 7.9 dargestellt. Es zeigt zunächst den Start der Anwendung, gefolgt von der parallelen Eingabe von zwei Instanzen vom Datentyp *Person*. Sobald die zwei Eingabeinteraktionen beendet sind, wird eine der erzeugten Personen ausgewählt und anschließend angezeigt. Zuletzt folgt das Ende der Anwendung.

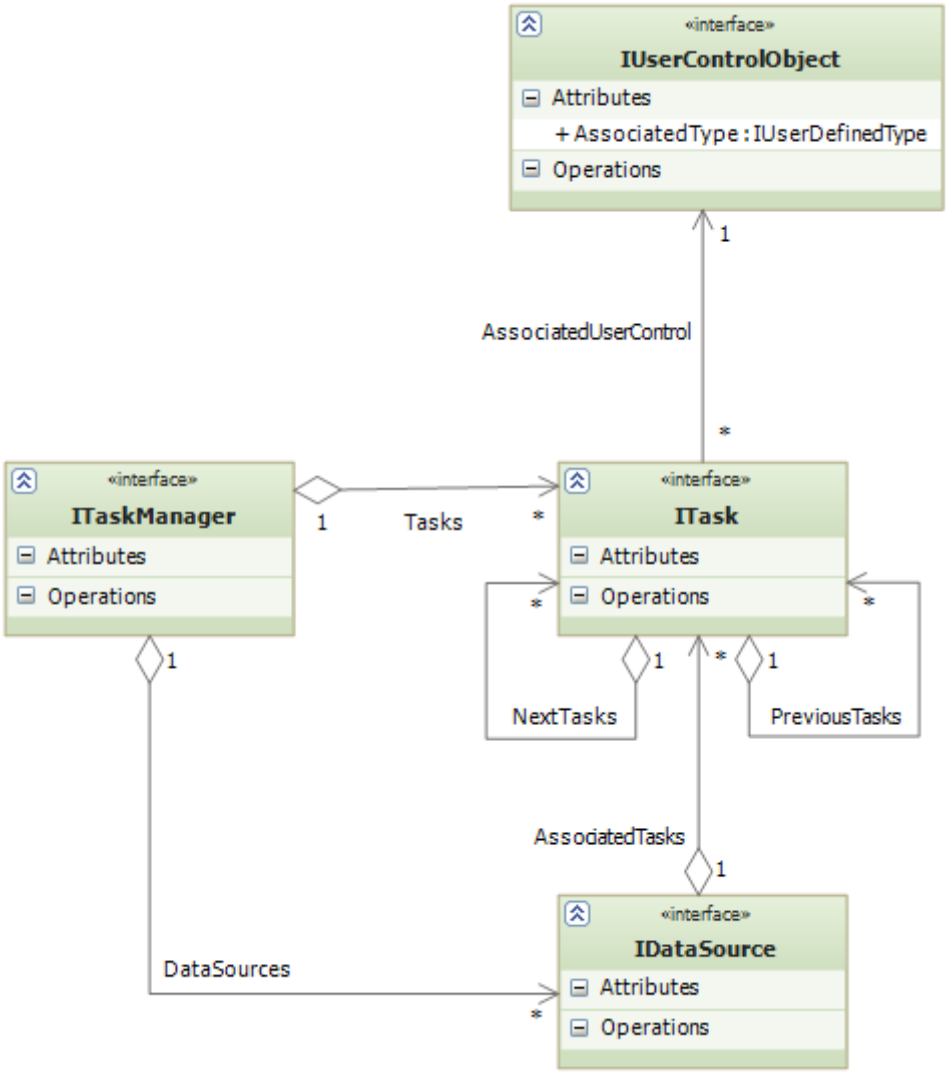


Abbildung 7.8 UML-Diagramm des Ablaufmodells

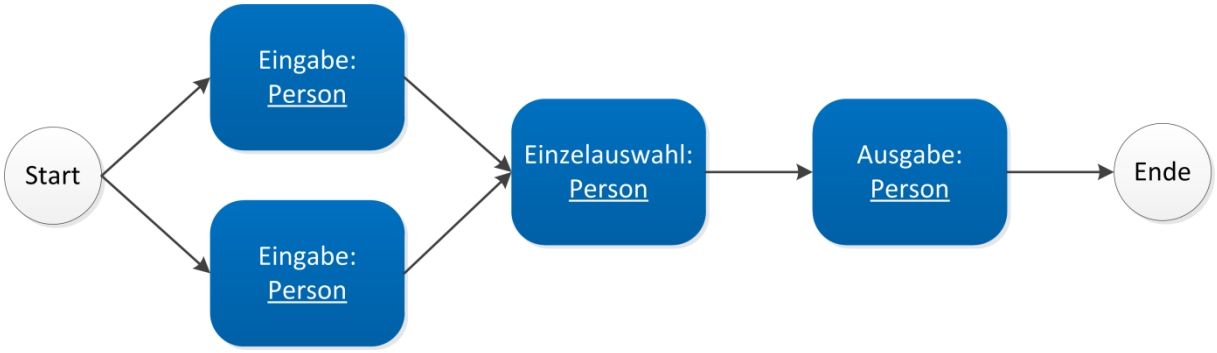


Abbildung 7.9 Grafische Darstellung eines Ablaufmodells

Zur Bearbeitung des Ablaufmodells kann der Benutzer den in Abschnitt 8.5 beschriebenen Ablaufeditor verwenden.

7.5. Die Ausführungskomponente

Die Aufgabe der Ausführungskomponente ist es, auf Basis des erstellten Modells eine ausführbare Anwendung zu erstellen. Dabei wird das Modell interpretiert, die Daten dynamisch erzeugt, die abstrakten Interaktionen durch konkrete Realisierungen ersetzt und anschließend der Ablauf der Anwendung gesteuert. Es ist dabei nicht vorgesehen, dass aus dem Modell eine eigenständig lauffähige Anwendung erstellt wird, vielmehr wird die Ausführungskomponente zur Ausführung des Modells zwingend benötigt.

Intern besteht die Ausführungskomponente aus den in Abbildung 7.10 gezeigten Elementen. Neben den bereits bekannten Verwaltungskomponenten für die Modell-Komponenten sind dies:

ITypeGenerator

Der *ITypeGenerator* dient dazu, aus den, im *ITypeManager* beschriebenen, benutzerdefinierten Datentypen reale .NET-Datentypen zu erzeugen. Dazu wird mit Hilfe des *Code Document Object Models* (*CodeDOM*)[9] dynamisch Programmcode erzeugt. Zudem übernimmt der Typengenerator auch die Instanziierung der Datentypen und deren Verwaltung.

[9] CodeDOM ist seit der Version 2.0 ein Bestandteil des .NET Frameworks. Es ermöglicht die dynamische Erzeugung, Ausgabe und Kompilierung von Quellcode in verschiedenen Sprachen des Frameworks.

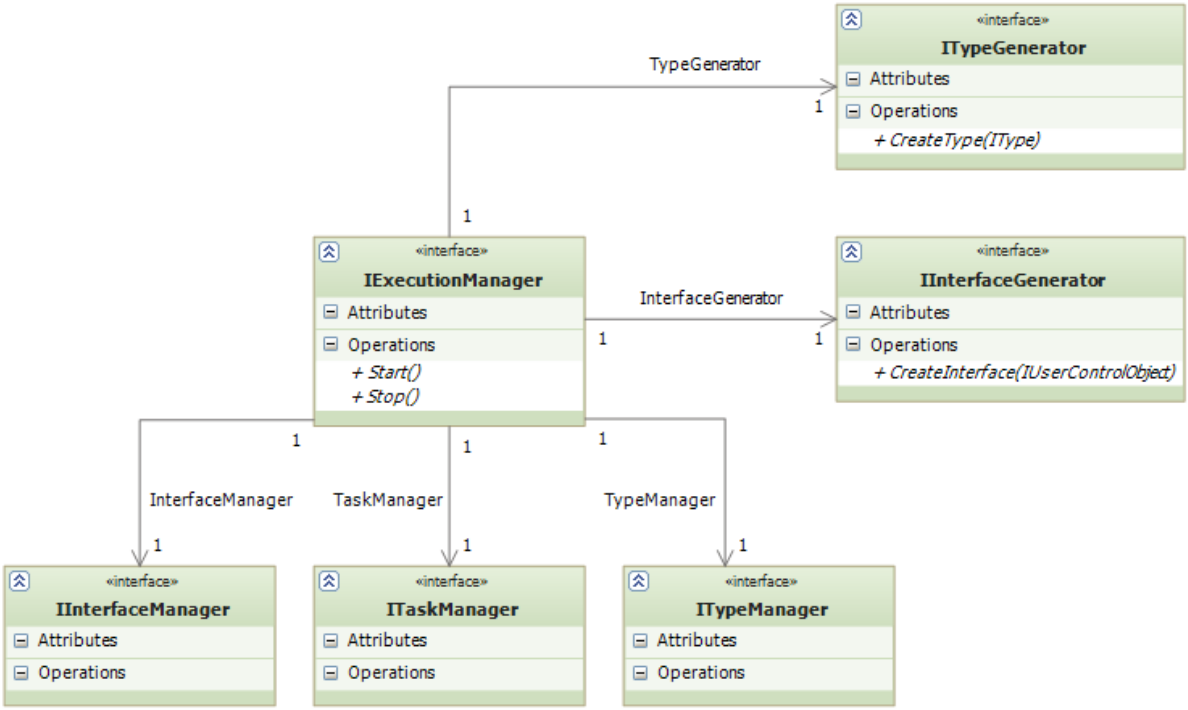


Abbildung 7.10 UML-Diagramm der Ausführungskomponente

IInterfaceGenerator

Der *IInterfaceGenerator* erzeugt aus den einzelnen Benutzersteuerelementen reale Interfaceobjekte, die dem Benutzer auf der Oberfläche angezeigt werden können. Dazu werden nicht nur die einzelnen Steuerelemente erstellt, sondern auch die Verbindung zwischen den Elementen und den zugrundeliegenden Datenobjekten hergestellt. Dazu verwendet der *IInterfaceGenerator* das aus der WPF bekannte DataBinding, welches eine einfache Synchronisation zwischen den angezeigten Werten und den verwendeten Daten ermöglicht.

IExecutionManager

Die Aufgabe des *IExecutionManagers* besteht darin, die Erzeugung von Datentypen und Interfaceobjekten, sowie den Ablauf der Elemente zu steuern. Dazu verarbeitet er die Informationen des Anwendungsmodells mit Hilfe des *ITypeGenerators*, liest die Daten des Ablaufmodells aus und übersetzt diese in die entsprechenden Benutzersteuerelemente aus dem Präsentationsmodell, welche anschließend mit Hilfe des *IInterfaceGenerators* in die Oberflächen übersetzt werden.

Zudem steuert der *IExecutionManager* den Ablauf der Anwendung, indem er auf die Ausführungselemente der Interfaceobjekte reagiert und diese verarbeitet.

Mit der Ausführungskomponente ist die Beschreibung der Architektur abgeschlossen. Im folgenden Kapitel werden die Realisierung dieser Architektur, die Bearbeitung der Modelle mit Hilfe von Editoren sowie die Ausführung der Anwendung genauer betrachtet.

8. Realisierung

Die im Rahmen dieser Arbeit erstellte Anwendung namens „*End-user Development Tool for Multi-Touch*“ (EDT4MT) basiert auf der im vorherigen Kapitel vorgestellten Architektur und stellt eine exemplarische Implementierung des in Kapitel 6 beschriebenen Modells dar.

Ziel dieses Kapitels ist es, die verschiedenen Editoren sowie die Komponente zur Ausführung der erzeugten Modelle zu beschreiben, wie sie durch den Benutzer wahrgenommen und verwendet werden. Während im vorherigen Kapitel die Architektur des zugrundeliegenden Modells und der Ausführungskomponente beschrieben wurde, beschreibt dieses Kapitel die Benutzeroberfläche zur Bearbeitung der Modelle und zum Starten der Ausführung.

Die Editoren umfassen den Typeditor zur Bearbeitung des *ApplicationModel*, den Oberflächeneditor für das *PresentationModel* und den Ablaufeditor für das *TaskModel*. Die Funktionen der einzelnen Editoren werden in den entsprechenden Abschnitten beschrieben.

Bei der Ausführungskomponente wird zusätzlich die interne Funktionsweise näher betrachtet und die Abläufe innerhalb der Ausführung erläutert.

8.1. Grundlegende Informationen der Realisierung

Bevor in den folgenden Abschnitten auf die einzelnen Editoren und die Ausführungskomponente eingegangen wird, wird in diesem Abschnitt zunächst die Basis der Implementierung beschrieben.

Die Hard- und Softwaregrundlage der Implementierung bildet die Microsoft Surface Plattform, wie sie zum Zeitpunkt der Erstellung dieser Arbeit verfügbar war und in Abschnitt 2.4.4 ausführlich beschrieben ist. Die aktuellste und im Rahmen dieser Arbeit verwendete Version des Microsoft Surface SDKs ist Version 1.0 SP1. Dabei basiert die erstellte Anwendung auf der im SDK enthaltenen Presentation-Komponente, welche für Anwendungen auf Grundlage der WPF verwendet wird. Als Entwicklungsumgebung wurde Microsoft Visual Studio 2010 Ultimate verwendet

8.2. Das Hauptmenü

Das Hauptmenü der Anwendung bietet zunächst Schaltflächen, mit denen die Editoren für die verschiedenen Modell-Komponenten gestartet werden können. Eine vierte Schaltfläche ermöglicht die Ausführung der modellierten Anwendung. Die Benutzeroberfläche des Hauptmenüs ist in Abbildung 8.1 dargestellt.

Abbildung 8.1 Das Hauptmenü

8.3. Der Typeneditor

Der Typeneditor ermöglicht dem Benutzer das Erstellen, Bearbeiten und Löschen von Datentypen. Um diese Funktionalität bereitzustellen, besteht der Typeneditor aus den drei Komponenten Datentypenübersicht, Datentypeditor und Attributeditor, die im Folgenden vorgestellt werden:

Datentypenübersicht

Die Datentypenübersicht, die in Abbildung 8.2 zu sehen ist, zeigt neben den Basisdatentypen in der oberen Hälfte auch alle Datentypen an, die der Benutzer selbst erstellt hat. Die Übersicht der Basisdatentypen dient dabei lediglich der Anzeige und stellt keine Möglichkeiten zu deren Bearbeitung zur Verfügung. Neben den Namen der Basisdatentypen wird ebenfalls eine kurze Beschreibung angezeigt.

Innerhalb der Übersicht über die benutzerdefinierten Daten, gibt es mehrere Spalten. In der linken Hälfte werden die aktuellen Datentypen angezeigt, in der rechten die Subtypen des ausgewählten Datentyps der linken Spalte. Sobald der Benutzer einen Subtyp in der rechten Spalte auswählt, werden die Einträge der rechten Spalte automatisch in die linke verschoben und rechts die Subtypen des ausgewählten Datentyps angezeigt. Über verschiedene Schaltflächen kann der Benutzer weitere Datentypen hinzufügen und entfernen, sowie die Bearbeitung eines Datentyps anstoßen. Die Navigation von den Benutzerdatentypen auf der linken Seite zu ihrem Supertyp erfolgt über die Schaltfläche unterhalb der Liste.

Die Aufteilung der Datentypen auf mehrere Spalten hat mehrere Vorteile. Zum einen bietet sie eine bessere Übersicht der Datentypen im Vergleich zu einer Lösung mit einer einzigen Liste, in denen alle Datentypen aufgeführt sind. Zum anderen können die Datentypen dadurch hierarchisch strukturiert werden, was die Zusammenhänge

zwischen den einzelnen Datentypen, also die Assoziationen als Sub- und Supertypen, verdeutlicht, womit sie dem Benutzer einfacher zugänglich sind. Beide Vorteile könnten beispielsweise auch durch die Verwendung einer *TreeView*-Komponente erreicht werden, die allerdings kein Bestandteil des Surface SDKs ist. Daher wurde die Realisierung mit Hilfe der zweispaltigen Navigation durchgeführt.

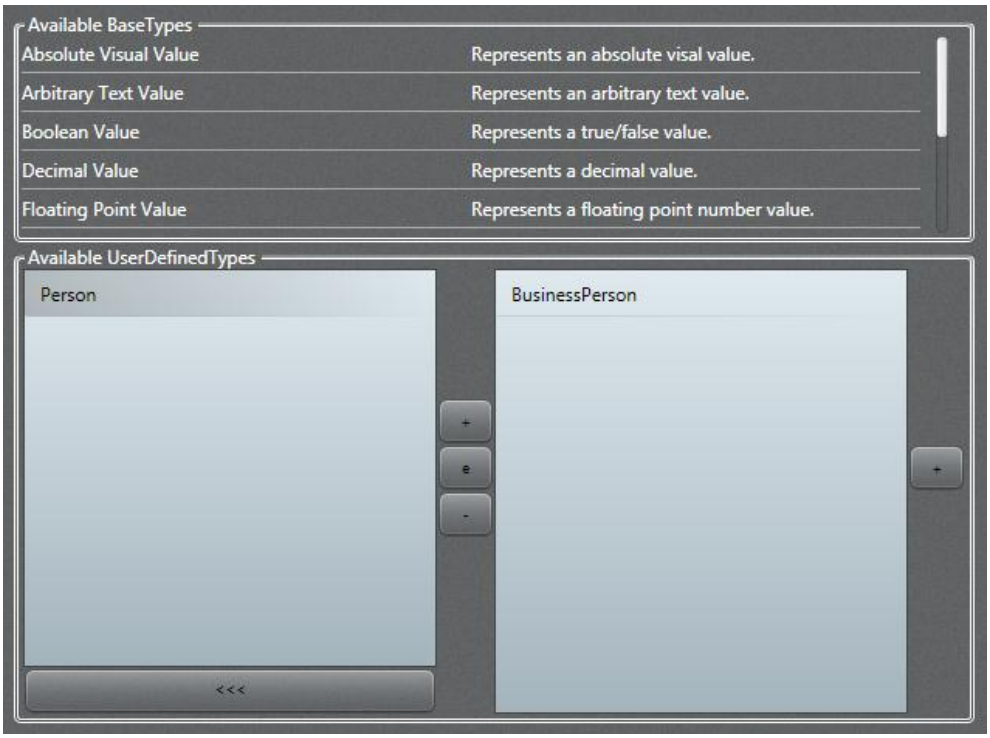


Abbildung 8.2 Die Datentypenübersicht

Datentypeditor

Der Datentypeditor wird angezeigt, sobald der Benutzer einen in der Datentypübersicht ausgewählten Datentyp bearbeiten möchte. Er bietet Möglichkeiten zum Ändern des Namens und der Beschreibung des Datentyps, sowie eine Übersicht der im Datentyp enthaltenen Attribute. Diesen können weitere Attribute hinzugefügt sowie bereits vorhandene bearbeitet und entfernt werden. Der Datentypeditor ist in Abbildung 8.3 dargestellt.

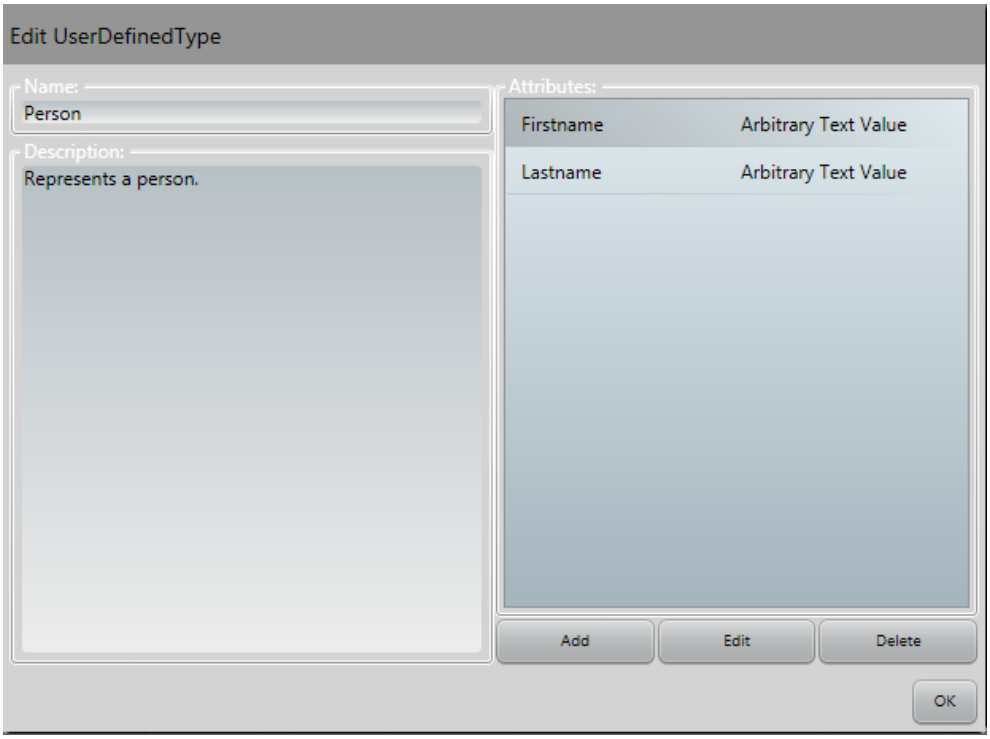


Abbildung 8.3 Der Datentypeditor

Attributeditor

Ähnlich dem Datentypeditor, der in Abbildung 8.4 dargestellt ist, dient der Attributeditor der Bearbeitung von Attributen.

Der Benutzer kann dem Attribut wiederum einen Namen und eine Beschreibung geben und festlegen, ob es sich bei dem Attribut um eine Liste von Objekten des gewählten Datentyps handelt, oder um ein einzelnes Objekt. In der Liste auf der rechten Seite kann er anschließend den Datentyp des Attributs festlegen. Dabei stehen sowohl alle Basis- als auch alle benutzerdefinierten Datentypen zur Auswahl.

Falls als Datentyp der Wert *Formatted Text Value* ausgewählt ist, so steht zusätzlich eine Möglichkeit zur Auswahl der Formatierung bereit. Dabei kann der Benutzer entweder einen der drei vordefinierten Werte für ein Datum, eine Email- oder eine IP-Adresse auswählen, oder selber eine Formatierung angeben. Diese basiert auf den im .NET Framework implementierten regulären Ausdrücken und der entsprechenden Syntax.

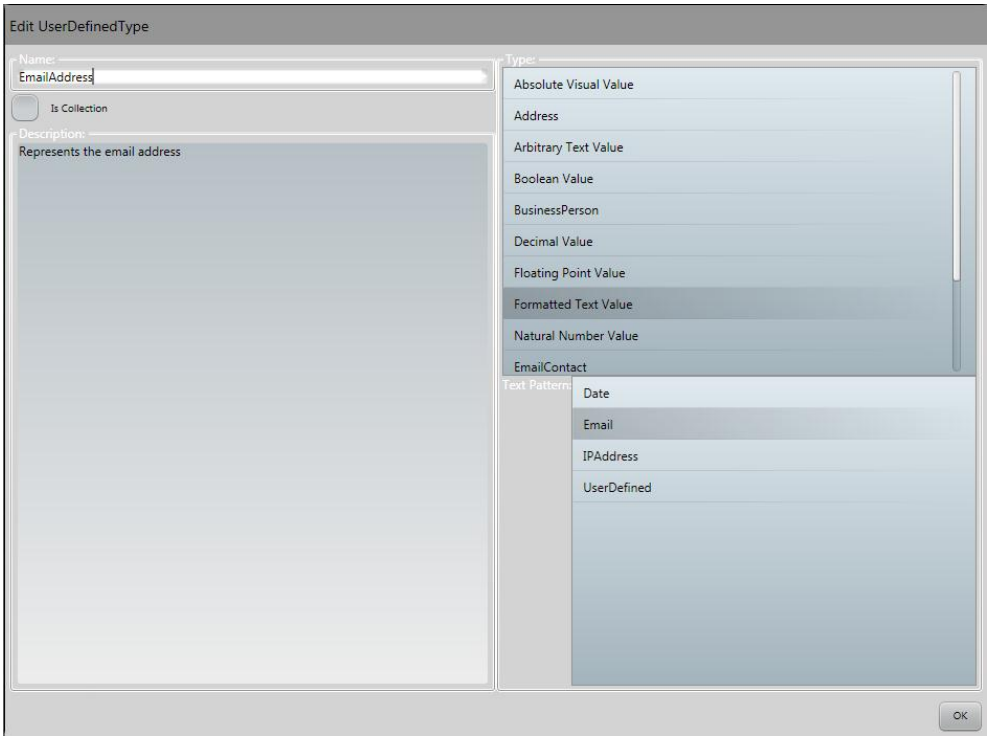


Abbildung 8.4 Der Attributeditor

Damit ist die Beschreibung des Typeneditors abschlossen. Der nächste Editor, der im folgenden Abschnitt beschrieben wird, ist der Oberflächeneditor.

8.4. Der Oberflächeneditor

Der Oberflächeneditor ermöglicht dem Benutzer die Bearbeitung des in Abschnitt 7.3 vorgestellten *PresentationModel*. In dem Editor ist es möglich neue Benutzersteuerelemente zu erstellen, vorhandene zu bearbeiten sowie diese zu löschen. Eine Darstellung des Oberflächeneditors ist in Abbildung 8.5 zu sehen. In dieser Abbildung sind die folgenden Bereiche des Editors markiert:

Übersicht

In der Übersicht auf der linken Seite werden alle verfügbaren Benutzersteuerelemente angezeigt. Über zwei Schaltflächen kann der Benutzer neue Benutzersteuerelemente erstellen, beziehungsweise das ausgewählte Element entfernen.

Sobald der Benutzer ein Benutzersteuerelement aus der Liste der verfügbaren Elemente auswählt, wird es auf der rechten Seite angezeigt und kann dort bearbeitet werden.

Daten des Benutzersteuerelements

In diesem Bereich werden die Daten des bearbeiteten Benutzersteuerelements angezeigt. Dies sind der Name des Elements, der vom Benutzer geändert werden kann, sowie der gewählte Interaktions- und der verknüpfte, benutzerdefinierte Datentyp.

Diese beiden Einstellungen können nicht geändert werden, da dies Auswirkungen hätte, die nicht automatisch korrigiert werden könnten.

Insbesondere beziehen sich diese Auswirkungen auf die Verbindungen zwischen den Attributen des Datentyps und den Interaktionselementen.

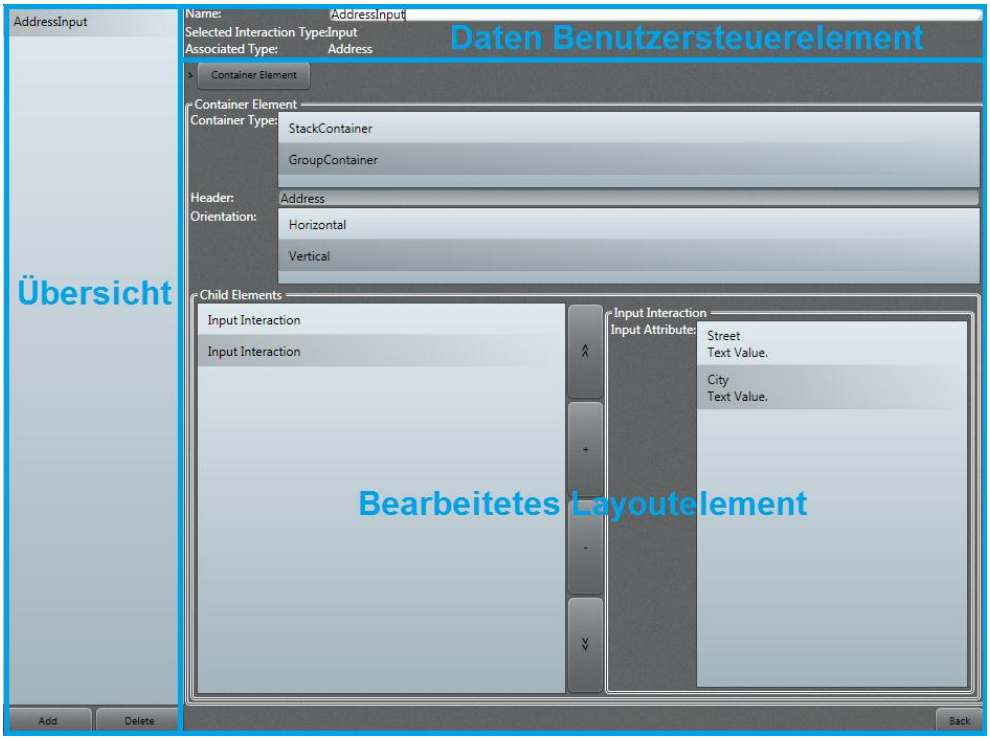


Abbildung 8.5 Der Oberflächeneditor

Bearbeitetes Layoutelement

Der umfangreichste Bereich des Oberflächeneditors dient zur Bearbeitung der Layout- beziehungsweise Containerelemente. Da die einzelnen Interaktions- und Benutzersteuerelemente immer innerhalb eines Layoutelements angeordnet werden müssen, wird auch stets diese Ansicht angezeigt.

Innerhalb der Bearbeitung des ausgewählten Layoutelements gibt es mehrere Bereiche. Diese sind in Abbildung 8.6 gekennzeichnet und dienen den folgenden Aufgaben.

- Navigationsbereich
  Mit Hilfe des Navigationsbereichs kann zwischen den einzelnen Layoutelementen navigiert werden. Dieser Bereich ist notwendig, da immer nur ein Layoutelement gleichzeitig bearbeitet werden kann. Falls der Benutzer ein Layoutelement innerhalb des aktuellen Layoutelements bearbeiten möchte, so wählt er dieses zur Bearbeitung aus. Innerhalb des Navigationsbereichs wird dieses neben dem vorherigen Layoutelement angezeigt. Durch eine Betätigung eines der Elemente innerhalb des Navigationsbereichs kann der Benutzer zu diesem Element wechseln und es bearbeiten. Alle dahinterliegenden Elemente werden aus dem Navigationsbereich entfernt.

Eine ähnliche Art der Navigation wird beispielsweise bei Webseiten verwendet, bei denen der Weg von der Hauptseite zur aktuell angezeigten Seite dargestellt wird.

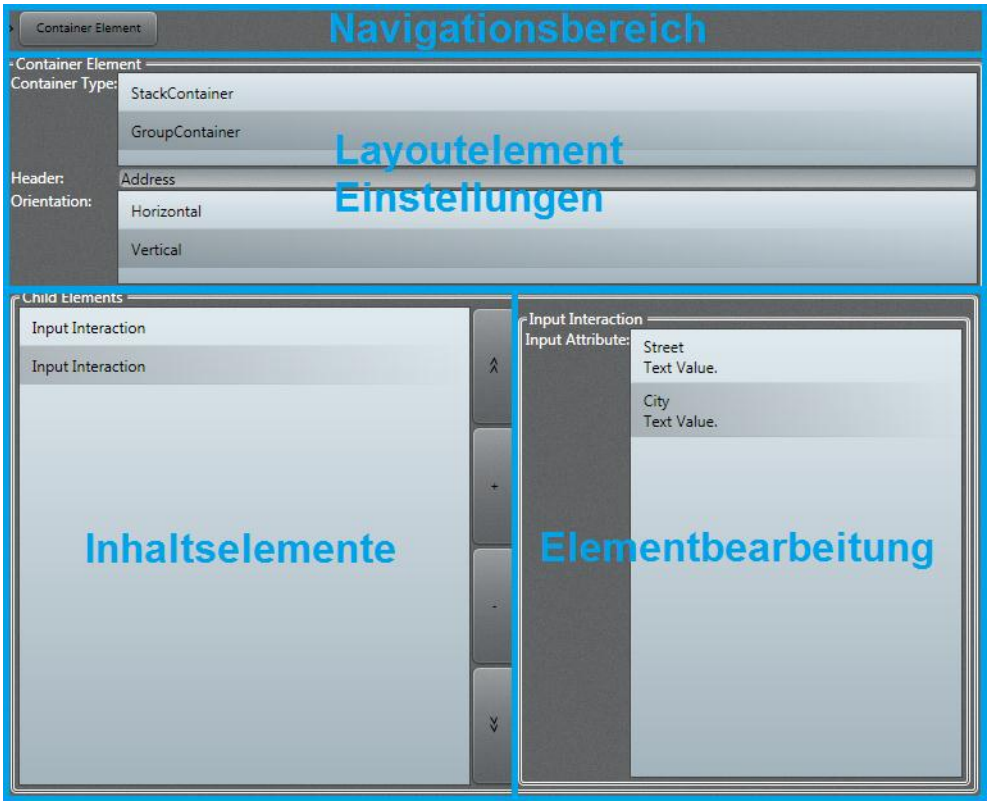


Abbildung 8.6 Bearbeitungsbereich eines Layoutelements

- Layoutelement Einstellungen
  Innerhalb dieses Bereichs können die allgemeinen Einstellungen des Layoutelements vorgenommen werden. Dies ist zum einen der Typ des Elements, also ob es sich um ein Stapel- oder Gruppierungselement handelt, sowie die Ausrichtung der enthaltenen Elemente. Falls der Benutzer ein Gruppierungselement ausgewählt hat, kann er zusätzlich den Titel des Elements angeben.
- Inhaltselemente
  In diesem Bereich werden die enthaltenen Kindelemente des Layoutelements angezeigt. Je nach gewählter Ausrichtung werden diese horizontal oder vertikal ausgerichtet. Über vier Schaltflächen kann der Benutzer neue Elemente hinzufügen, oder bereits vorhandene entfernen beziehungsweise die Reihenfolge der Elemente ändern.
  Das aktuell ausgewählte Element kann im Bereich der Elementbearbeitung angepasst werden.
- Elementbearbeitung
  Das aktuell ausgewählte Inhaltselement kann in diesem Bereich bearbeitet werden. Je nach Typ dieses Elements, stehen verschiedene Optionen zur Verfügung. Bei Eingabe-, Ausgabe- und Bearbeitungsinteraktionen kann der Benutzer auswählen, mit welchem Attribut des zugrundeliegenden Datentyps

das entsprechende Element verknüpft werden soll. Dabei werden dem Benutzer alle Attribute angezeigt, die als Datentyp einen der vorhandenen Basisdatentypen enthalten. In der dargestellten Abbildung wird eine Eingabeinteraktion mit dem Attribut *City* des Datentyps verknüpft.
Da Auswahlinteraktionen ausschließlich als direkter Inhalt von Benutzersteuerelementen vorkommen können, gibt es für sie keine spezielle Ansicht innerhalb der Elementbearbeitung.
Entsprechend stehen bei einem ausgewählten Benutzersteuerelement alle die Attribute zur Verfügung, deren Datentyp dem des Benutzersteuerelements entspricht.
Eine Besonderheit in diesem Bereich stellt die Bearbeitung eines Layoutelements dar. In diesem Fall kann der Benutzer zunächst auswählen, ob dieses Element über den gleichen Datenkontext verfügt, wie das umschließende Element, oder ob der Datenkontext auf ein Attribut gesetzt wird, dessen zugrundeliegender Datentyp ein benutzerdefinierter Datentyp ist. Eine beispielhafte Darstellung dieses Bereichs ist in Abbildung 8.7 zu sehen.
Zudem kann der Benutzer über eine Schaltfläche das gewählte Layoutelement bearbeiten. Dadurch wird es zum Navigationsbereich hinzugefügt und als aktuell bearbeitetes Layoutelement angezeigt.

Abbildung 8.7 Elementbearbeitung eines Layoutelements

Einen Unterschied dieser Anzeige gibt es bei der Bearbeitung von Benutzersteuerelementen, die eine Auswahl ermöglichen. Da die Auswahl mehrere Elemente benötigt, aus denen die Auswahl getroffen wird, wird bei solchen Steuerelementen ein anderer Inhalt angezeigt. Die zur Auswahl stehenden Elemente werden automatisch als Datenkontext gesetzt, sodass der Benutzer nicht das Attribut auswählen kann, welches diese Elemente enthält. Stattdessen kann er auswählen, welches Attribut des verknüpften Datentyps in der Auswahlliste angezeigt werden soll. Die Möglichkeit die Auswahlart einzustellen, ist auch in dieser Ansicht möglich.

Nachdem der Benutzer nun die Datentypen und die Steuerelemente, um mit diesen interagieren zu können, definiert hat, muss er als letzten Schritt der Modellierung noch die Abfolge der einzelnen Interaktionen definieren. Dazu steht ihm der im folgenden Abschnitt beschriebene Ablaufeditor zur Verfügung.

8.5. Der Ablaufeditor

Mit Hilfe des Ablaufeditors kann das *TaskModel* bearbeitet werden. Der Benutzer kann dazu Aufgaben und Datenquellen erstellen und diese anschließend verknüpfen. Eine beispielhafte Darstellung, die sich am Beispiel aus Abschnitt 7.4 orientiert, ist in Abbildung 8.8 zu sehen.

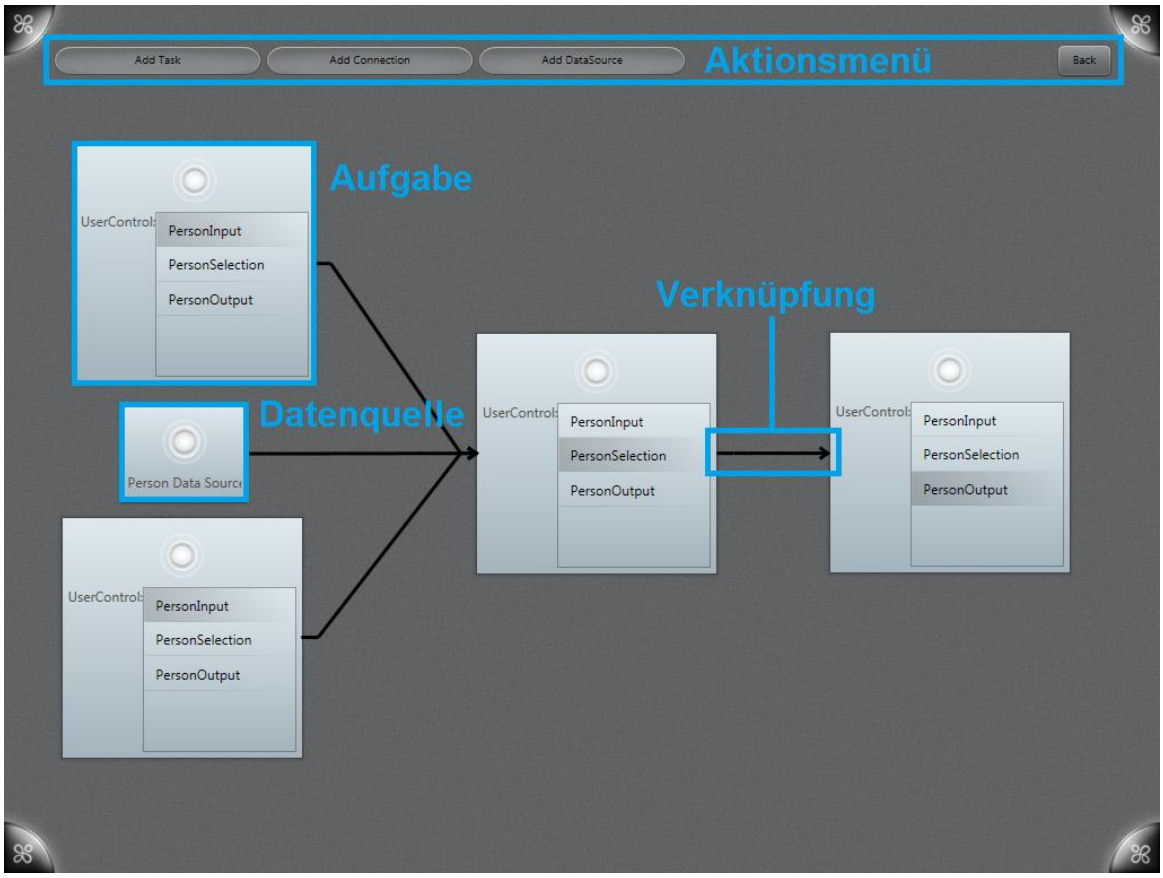


Abbildung 8.8 Der Ablaufeditor

Im Gegensatz zu dem genannten Beispiel, verfügt die Abbildung noch über eine Datenquelle, die auf der linken Seite zwischen den beiden Aufgaben zu sehen ist.

Die einzelnen Elemente des Ablaufeditors sind:

- Aktionsmenü
  Über das Aktionsmenü kann der Benutzer verschiedene Aktionen steuern. So kann er über die drei Schaltflächen auf der linken Seite den Modus zum Hinzufügen von Elementen starten. Um eine Aufgabe oder eine Datenquelle hinzuzufügen, wählt er die entsprechende Schaltfläche aus und berührt die Oberfläche dort, wo er das entsprechende Element hinzufügen möchte. Zum

Hinzufügen einer Verknüpfung wählt er die Schaltfläche aus, berührt mit dem Finger das Startelement der Verknüpfung und bewegt ihn zum Zielelement der Verknüpfung, wo er schließlich den Finger von der Oberfläche entfernt.

- Aufgaben
  Die Aufgaben entsprechen den einzelnen Schritten innerhalb des Ablaufs. Der Benutzer wählt dabei das Benutzersteuerelement aus, das bei der Ausführung der Aufgabe angezeigt werden soll.
- Datenquellen
  Datenquellen ermöglichen die Integration von externen Daten. Im Rahmen dieser Arbeit steht eine Datenquelle für Personen zur Verfügung, welche eine Liste von diesen zur Verfügung stellt.
- Verknüpfungen
  Mit Hilfe von Verknüpfungen zwischen Datenquellen und Aufgaben wird deren Ablauf beschrieben. Um die Richtung der Verknüpfung zu erkennen, sind diese als Pfeile zwischen den Elementen dargestellt.

Um eine Aufgabe, Datenquelle oder Verknüpfung zu entfernen, kann der Benutzer das bei jedem dieser Elemente vorhandene Element-Menü verwendet werden. Bei Aufgaben und Datenquellen ist dies immer sichtbar, bei Verknüpfungen nur wenn der Benutzer diese berührt.

Sowohl die Aufgaben, als auch die Datenquellen können durch den Benutzer verschoben und beliebig angeordnet werden. Um dies zu ermöglichen werden die im Microsoft Surface SDK enthaltenen Steuerelemente *ScatterView* als Container und verschiedene *ScatterViewItems* für die einzelnen Elemente verwendet. Eine automatische Ausrichtung der beschriebenen Elemente ist im Rahmen dieser Arbeit nicht vorgesehen, könnte aber mit einigen Anpassungen hinzugefügt werden.

Mit Hilfe der drei beschriebenen Editoren, kann der Benutzer alle Bestandteile des Modells bearbeiten. Um auf dessen Basis eine lauffähige Anwendung zu erstellen, kann der Benutzer die Ausführungskomponente verwenden, die im nächsten Abschnitt beschrieben wird.

8.6. Die Ausführung

In diesem Abschnitt wird die Realisierung der Ausführungskomponente, deren Architektur in Abschnitt 7.5 vorgestellt wurde, beschrieben. Im Gegensatz zu JANUS (Abschnitt 2.3.3) und Teallach (Abschnitt 2.3.4) wird dabei kein Quellcode für eine eigenständig lauffähige Anwendung erzeugt, sondern das vorhandene Modell interpretiert und daraus dynamisch sowohl die Datentypen und die Benutzeroberfläche, als auch Ablauf der einzelnen Elemente erzeugt.

Um die Ausführung besser verstehen zu können, wird zunächst der generelle Ablauf beschrieben, bevor die einzelnen Generatoren beschrieben werden.

8.6.1. Genereller Ablauf

Der generelle Ablauf der Ausführung ähnelt dem der Modellierung. Zunächst werden aus den benutzerdefinierten Datentypen mit Hilfe des *ITypeGenerator* zur Laufzeit .NET-Datentypen erstellt. Anschließend werden die einzelnen Steuerelemente durch den *IInterfaceGenerator* erstellt, die vom Benutzer im *PresentationModel* beschrieben wurden. Zuletzt folgt der umfangreichste Abschnitt der Ausführung, nämlich die Verknüpfung der Steuerelemente mit Instanzen der Datentypen und Datenquellen, sowie die Steuerung des Ablaufs und Datenflusses. Dazu müssen zunächst über den *ITypeGenerator* die Instanzen der Datentypen erzeugt werden und an die Benutzeroberflächenelemente gekoppelt werden, die durch den *IInterfaceGenerator* erstellt wurden. Für diese Aufgaben steht der *IExecutionManager* zur Verfügung.

Eine vereinfachte Darstellung dieses Ablaufs ist in Abbildung 8.9 dargestellt.

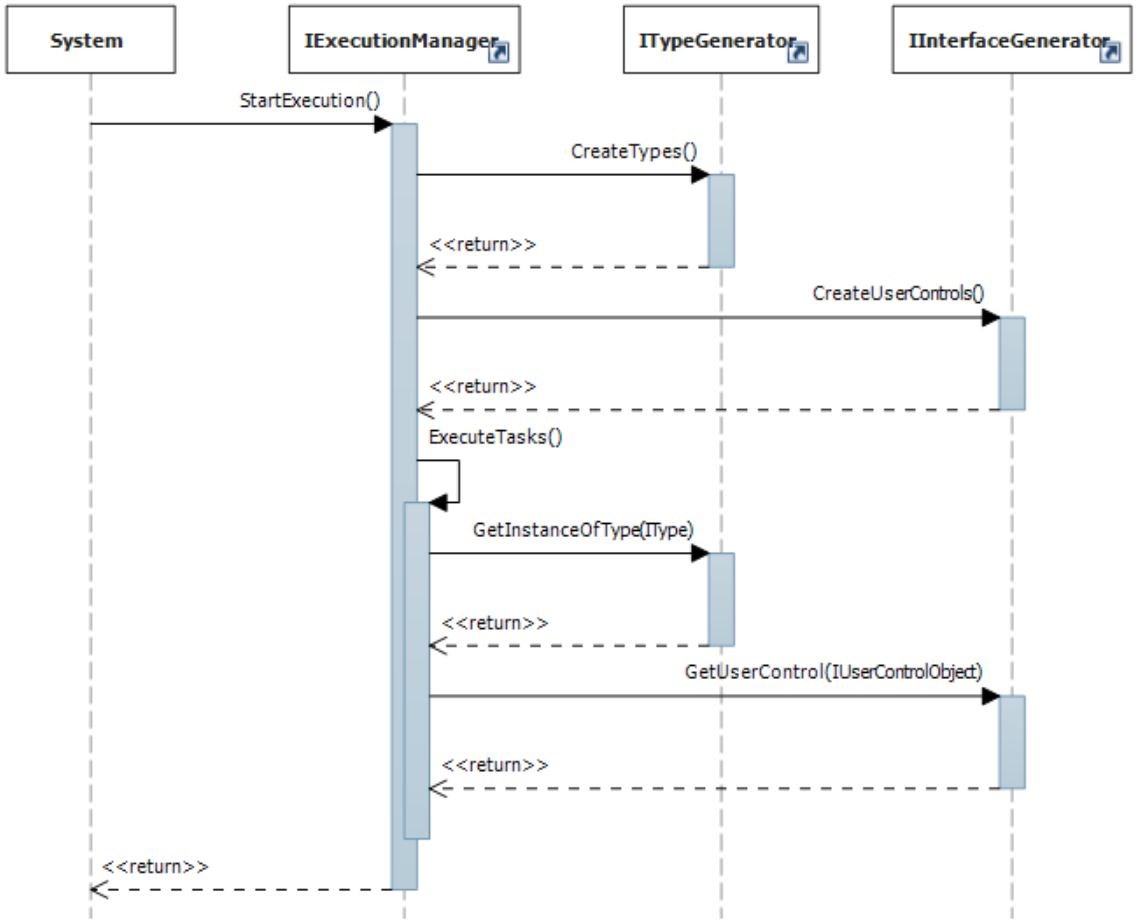


Abbildung 8.9 UML-Sequenzdiagramm des generellen Ablaufs der Ausführung

Die Funktionsweise der einzelnen Generatoren wird in den folgenden Abschnitten beschrieben.

8.6.2. Die Typerstellung

Der *ITypeGenerator*, der für die Erstellung der Datentypen zuständig ist, verwendet als Ausgangspunkt die im *ITypeManager* modellierten Datentypen, für die er mit Hilfe der CodeDOM-Komponenten des .NET Frameworks entsprechende .NET-Datentypen erzeugt. Dazu wird für jeden benutzerdefinierten Datentyp eine *CodeTypeDeclaration* erstellt, die letztendlich eine Klasse beschreibt. Um später über Änderungen der Daten benachrichtigt zu werden, wird zusätzlich das *INotifyPropertyChanged*-Interface implementiert. Innerhalb der erstellten Klasse werden die einzelnen Attribute des benutzerdefinierten Datentyps durch Properties realisiert, welche bei Änderungen das *PropertyChanged*-Event des genannten Interfaces auslösen.

Nachdem für alle benutzerdefinierten Datentypen eine entsprechende Deklaration erstellt wurde, werden diese alle mit Hilfe des C#-Compilers in eine *Assembly* übersetzt, die allerdings nur im Speicher vorhanden ist und nicht persistiert wird. Auf Basis dieser *Assembly* können anschließend Instanzen der Datentypen erstellt werden. Dabei wird nicht nur eine leere Instanz des Datentyps erstellt, sondern auch dessen Properties initialisiert. Im Fall von Basisdatentypen des .NET Frameworks werden diese mit den Standardwerten initialisiert, im Fall von anderen benutzerdefinierten Typen mit einer Instanz dieses Typs.

Einschränkungen

Durch die beschriebene Initialisierung der Datentypen ist es im Rahmen dieser Arbeit nicht möglich zyklische Abhängigkeiten zwischen einzelnen Datentypen zu realisieren, denn aufgrund der standardmäßigen Initialisierung der Instanzen der Datentypen würde dies zu einer Endlosschleife führen.

Eine weitere Einschränkung ergibt sich durch die Verwendung der CodeDOM-Komponenten und die Erzeugung der *Assemblies* im Speicher. Durch die mögliche Komposition von Datentypen ist es nicht ohne großen Aufwand möglich diese auf mehrere *Assemblies* zu verteilen, weshalb sie innerhalb der gleichen *Assembly* definiert sein müssen. Dadurch ist es nicht möglich dynamisch weitere Datentypen zu erstellen, vielmehr müssen bei einer Änderung eines Datentyps alle anderen Datentypen auch neu erstellt und kompiliert werden.

8.6.3. Die Benutzeroberflächenerstellung

Die Erstellung der Benutzeroberfläche besteht aus zwei Teilen. Zum einen umfasst dies die Erstellung der Steuerelemente entsprechend den Vorgaben des Modells, zum anderen wird die Verknüpfung dieser Steuerelemente mit den Daten ebenfalls im Rahmen der Benutzeroberflächenerstellung durchgeführt.

Erstellung der Steuerelemente

Die Umsetzung der im Modell enthaltenen Interaktionsobjekte in konkrete Steuerelemente der Microsoft Surface Plattform orientiert sich an der in Kapitel 5 be-

schriebenen Abbildung dieser Elemente. Zusätzlich wird neben jedem Interaktionselement angezeigt, welches Datum dadurch bearbeitet wird. Eine beispielhafte Darstellung eines solchen Elements ist in Abbildung 8.10 zu sehen.

Eine besondere Rolle innerhalb der Interaktionselemente spielen die Ausführungselemente. Diesen wurde kein Datentyp zugeordnet, sondern eine Aktion die sie ausführen sollen. Diese Aktionen werden in Kommandos umgewandelt, die innerhalb der Steuerung des Ablaufs behandelt werden.

Im Rahmen dieser Implementierung werden auch die Auswahlinteraktionselemente speziell behandelt. Diese verfügen standardmäßig über zwei Ausführungselemente, von denen eines eine zufällige Auswahl eines Elements ermöglicht und das andere die Auswahl beendet, wodurch bei der Ausführung mit der folgenden Aufgabe fortgefahren werden kann.

Abbildung 8.10 Beispiel eines realen Interaktionselements

Neben den Interaktionselementen, werden die *ILayoutContainer*-Elemente folgendermaßen abgebildet:

- Stapelelemente → StackPanel
- Gruppierungselemente → GroupBox

Die Benutzersteuerelemente werden auf die gleiche Art und Weise abgebildet, da sie im Prinzip wieder auf Interaktions- und Layoutelemente zurückzuführen sind.

Verknüpfung mit den Daten

Die zweite wichtige Aufgabe der Benutzeroberflächenerstellung ist die Verknüpfung der Steuerelemente mit den zugrundeliegenden Daten. Da jedem Element der Benutzeroberfläche im *PresentationModel* ein Attribut wurde, für dessen Verarbeitung das Element zuständig ist, kann diese Verknüpfung mit den Bordmitteln der WPF behandelt werden. Die WPF bietet das sogenannte *DataBinding* an, welches es erlaubt bestimmte Eigenschaften von Steuerelementen an Properties des Datenkontextes zu binden. Da diese Datenbindung über den Namen des Properties funktioniert, kann dazu der Name des verknüpften Attributs aus dem *PresentationModel* verwendet werden. Diese Verbindung funktioniert auch, sofern noch gar kein Datenkontext gesetzt wurde, weshalb sie bereits im Voraus definiert werden kann.

Als Besonderheit ist dabei zu erwähnen, dass diese Datenbindung allein über den Namen der Properties funktioniert und nicht über den Datentyp. Falls eine Datenbindung auf ein Property verweist, das im aktuellen Datenkontext nicht verfügbar ist, wird diese ignoriert. Dadurch ist es möglich, die in Abschnitt 7.3 beschriebenen Möglichkeiten bezüglich der Vererbung von Datentypen und Benutzersteuerelementen zu realisieren. Bei der Verwendung eines Benutzersteuerelements zur Anzeige eines *BusinessPerson*-Objekts mit einem *Person*-Objekt, wird beispielsweise das

Element zur Anzeige des Arbeitgebers angezeigt, da aber innerhalb des *Person*-Objekts kein entsprechendes Property vorhanden ist, bleibt es leer. Wird andersrum ein Benutzersteuerelement zur Anzeige eines *Person*-Objekts für ein *BusinessPerson*-Objekt verwendet, so wird der Arbeitgeber nicht angezeigt, da kein entsprechendes Element verfügbar ist.

Einschränkungen

Die wesentliche Einschränkung bei der Erstellung der Benutzeroberfläche besteht darin, dass keine zyklischen Abhängigkeiten zwischen den Benutzersteuerelementen vorhanden sein dürfen. Anderenfalls käme es, wie bei der Datentyperstellung auch, zu einer Endlosschleife.

8.6.4. Die Steuerung des Programmablaufs

Der letzte Bestandteil der Ausführung ist die Steuerung des Programmablaufs und des Datenflusses. Er basiert auf dem *ITaskManager* und beginnt mit den dort referenzierten *ITask*-Objekten, welche den Start der Anwendung definieren. Für ein besseres Verständnis verwenden wir hier den in Abbildung 8.8Abbildung 8.8 Der Ablaufeditor dargestellten Ablauf als Ausgangssituation.

Die Startaufgaben sind in diesem Fall die beiden auf der linken Seite dargestellten Aufgaben, die jeweils zur Eingabe einer Person dienen. Für jede dieser Aufgaben wird zunächst die Realisierung des verknüpften Benutzersteuerelements (*PersonInput*) vom *IInterfaceGenerator* angefordert und mit Hilfe eines *ScatterViewItems* auf der Benutzeroberfläche dargestellt. Anschließend wird der Datenkontext des Elements gesetzt. Da es sich bei beiden Aufgaben um Eingabeinteraktionselementen handelt, wird dazu eine Instanz des verknüpften Datentyps (*Person*) vom *ITypeGenerator* angefordert und als Datenkontext des Benutzeroberflächenelements gesetzt.

Sobald beide Eingabeinteraktionen beendet sind, beispielsweise weil die Schaltfläche zum Beenden der Aufgabe betätigt wurde, wird die nächste Aufgabe begonnen. Für diese wird zunächst ebenfalls das entsprechende Benutzeroberflächenelement bezogen und anschließend der Datenkontext gesetzt. Da es sich nicht um eine Eingabe-, sondern um eine Auswahlinteraktion handelt, werden dazu die Datenkontexte der vorherigen Aufgaben, sowie der Datenquelle kombiniert. Dazu wird ein separater Datenkontext erstellt, der in einer Liste alle vorherigen Datenkontexte vereint und zusätzlich ein Property bereitstellt, welches das während der Interaktion ausgewählte enthält. Dabei bleibt zu erwähnen, dass im Rahmen der vorgestellten Implementierung bei der Bereitstellung des Datenkontexts für ein Element keine Überprüfung auf die Kompatibilität der Datentypen durchgeführt wird, sondern der Benutzer selber dafür verantwortlich ist.

Falls für eine Aufgabe, welche nur ein einzelnes Datenobjekt als Datenkontext erwartet, mehrere Datenkontexte zur Verfügung stehen, so wird der erste in der Auflistung dieser Datenkontexte verwendet.

Sobald die Auswahlinteraktion beendet wurde, wird anschließend die letzte Aufgabe genau wie die bisherigen auch gestartet und die ausgewählte Person ausgegeben. Sobald diese beendet wurde, wird die gesamte Ausführung beendet.

9. Evaluation der Ergebnisse

Nachdem in den vorherigen Kapiteln ein Modell zur endbenutzertauglichen Modellierung von Multi-Touch-Anwendungen vorgestellt und im Rahmen einer prototypischen Implementierung umgesetzt wurde, bleibt noch zu zeigen, dass die erläuterten Ansätze die an sie gestellten Anforderungen auch erfüllen.

9.1. Funktionsfähigkeit des Modells und der Implementierung

Zunächst wird in diesem Abschnitt die Funktionsfähigkeit des Modells und dessen prototypischer Implementierung evaluiert. Um dabei einen Bezug zu realen Anwendungsgebieten zu erstellen, gehen wir von folgender Problemstellung aus, die mit Hilfe des vorgestellten Modells und der entsprechenden Implementierung gelöst werden soll:

- Ziel:
  Die Ausgangssituation ist ein Gewinnspiel, bei dem aus einer Menge von Personen ein Gewinner ausgewählt werden soll.
- Teilnehmerdaten:
  Die Teilnehmerdaten stehen zum einen in einer Datenquelle zur Verfügung, können zum anderen aber auch vor der zufälligen Auswahl des Gewinners durch den Benutzer zusätzlich eingegeben werden. Am Gewinnspiel können sowohl Personen, als auch Geschäftspersonen teilnehmen.
- Ermittlung des Gewinners:
  Der Gewinner wird zufällig aus allen Teilnehmern ausgewählt. Anschließend kann dieser bearbeitet werden, beispielsweise um die Adresse in ein bestimmtes Format zu ändern. Anschließend wird der Gewinner ausgegeben, damit ihm sein Gewinn übermittelt werden kann.

Dieses Beispiel wird nun zunächst in ein Modell umgesetzt, welches als Grundlage für die anschließende Ausführung dient. Dabei wird insbesondere die Vorgehensweise der Modellierung noch einmal verdeutlicht.

Schritt 1: Datentypmodellierung

Im Rahmen der Anforderungen wurde definiert, dass sowohl Personen, als auch Geschäftspersonen am Gewinnspiel teilnehmen können. Aus diesem Grund müssen zunächst diese beiden benutzerdefinierten Datentypen erstellt werden, weshalb der Typeneditor gestartet wird.

In diesem wird zunächst der Datentyp *Address* erstellt, welcher über die zwei Attribute *Street* und *City* vom Basisdatentyp *Text Value* verfügt. Eine Abbildung des Datentyps ist in Abbildung 9.1 zu sehen. Dabei ist das Eingabefeld für den Namen des Datentyps mit 1, die angelegten Attribute mit 2 gekennzeichnet.

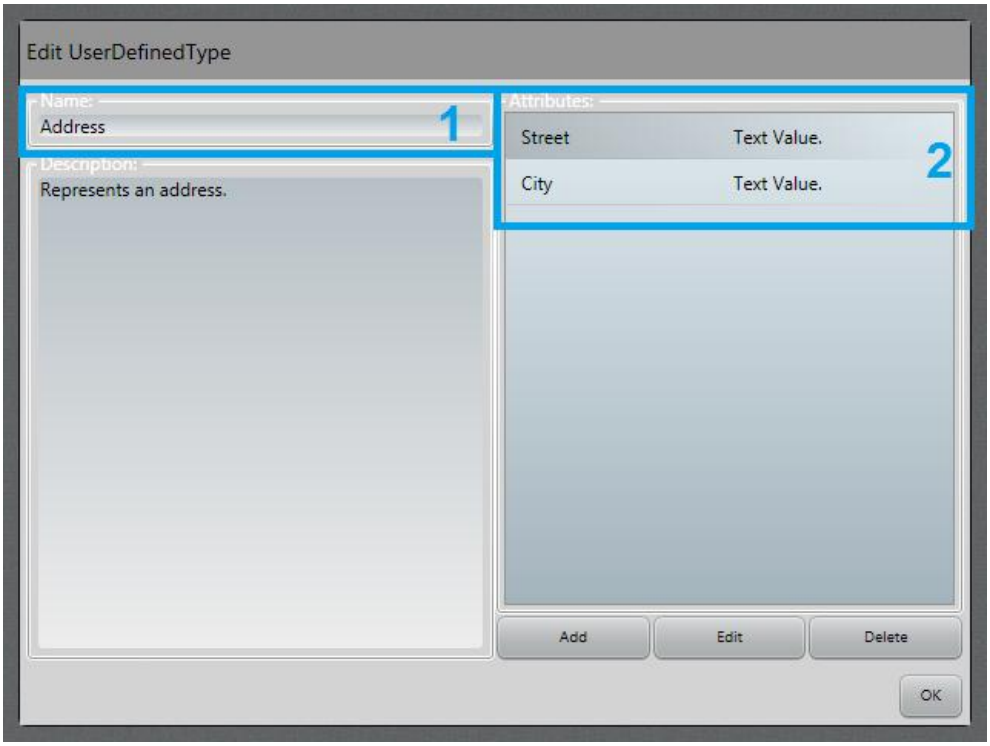


Abbildung 9.1 Erstellung des Datentyps *Address*

Anschließend wird der Datentyp Person erstellt, welcher neben den Attributen *Firstname* und *Lastname* vom Typ *Arbitraty Text Value* auch über die Adresse der Person verfügt, welche im Attribut *Address* mit eben diesem Namen enthalten ist.

Als letzter Schritt innerhalb der Datentypmodellierung, wird ein Subtyp des Datentyps *Person* erstellt, der *BusinessPerson* genannt wird und neben den bereits vorhandenen Attributen ein weiteres für den Arbeitgeber enthält. Dieses Attribut vom Typ *Text Value* wird mit dem Namen *Company* bezeichnet.

Eine grafische Darstellung der modellierten Datentypen ist in Abbildung 7.3 und Abbildung 7.4 zu sehen.

Schritt 2: Oberflächenmodellierung

Basierend auf den modellierten Datentypen werden nun die folgenden Benutzersteuerelemente erstellt:

- AddressInput
  Dieses Element dient der Eingabe einer Adresse und enthält dafür zwei Eingabeinteraktionselemente für die beiden verfügbaren Attribute des Datentyps *Address*.
- AddressEdit
  Mit diesem Element kann eine Adresse bearbeitet werden. Um dies zu ermöglichen werden zwei Bearbeitungsinteraktionselemente mit den verfügbaren Attributen verknüpft.
- AddressOutput

Durch dieses Element kann eine Adresse ausgegeben werden. Dementsprechend verfügt es über zwei Ausgabeinteraktionen, die mit den entsprechenden Attributen verknüpft werden.

- PersonInput
  Entspricht dem AddressInput-Element, allerdings für eine *Person* anstelle einer Adresse.
- BusinessPersonEdit, BusinessPersonOutput
  Verhalten sich wie die entsprechenden Elemente zum Bearbeiten und Ausgeben einer Adresse, wobei sie eine *BusinessPerson* ausgeben.
- PersonSelection
  Bietet die Möglichkeit ein einzelnes *Person*-Objekt aus einer Menge von *Person*-Objekten auszuwählen.

Im Folgenden wird die Erstellung der Elemente *PersonInput* und *PersonSelection* beschrieben. Die weiteren Elemente können äquivalent erstellt werden, wobei das Element *AddressInput* bereits vor dem *PersonInput*-Element erstellt werden muss.

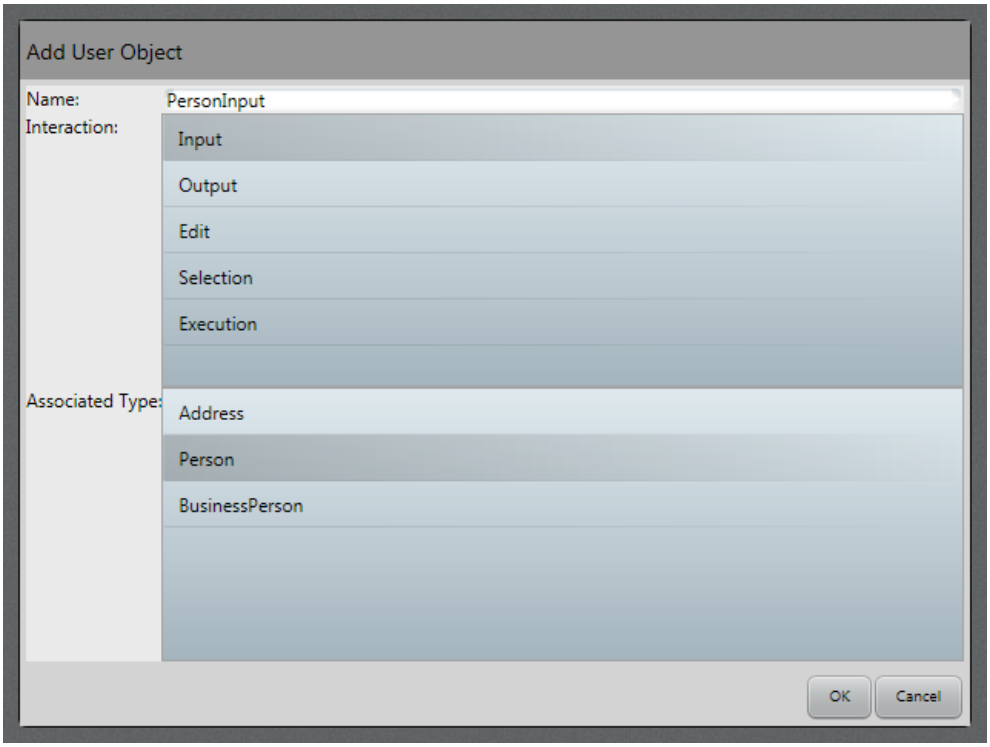


Abbildung 9.2 Erstellung des Benutzersteuerelements *PersonInput*

Zunächst wird, wie in Abbildung 9.2 zu sehen ist, ein neues Benutzersteuerelement erstellt, dem bei der Erstellung der Datentyp *Person* und die Interaktion *Input* zugeordnet werden. Für das automatisch enthaltene Layoutelement wird der Typ auf *StackContainer* und die Ausrichtung auf *Vertical* gesetzt. Anschließend werden zwei *Input Interaction* Elemente hinzugefügt, wobei das erste mit dem *Firstname*- und das zweite mit dem *Lastname*-Attribut verknüpft wird. Als drittes Element wird das Benutzersteuerelement *AddressInput* hinzugefügt, welches mit dem *Address*-Attribut verbunden wird. Zuletzt wird eine *Command Interaction* hinzugefügt und der *Com-*

mand Type auf den Wert *CompleteTask* gesetzt. Damit ist das *PersonInput*-Element vollendet. Abbildung 9.3 zeigt eine Abbildung der entsprechenden Oberfläche.

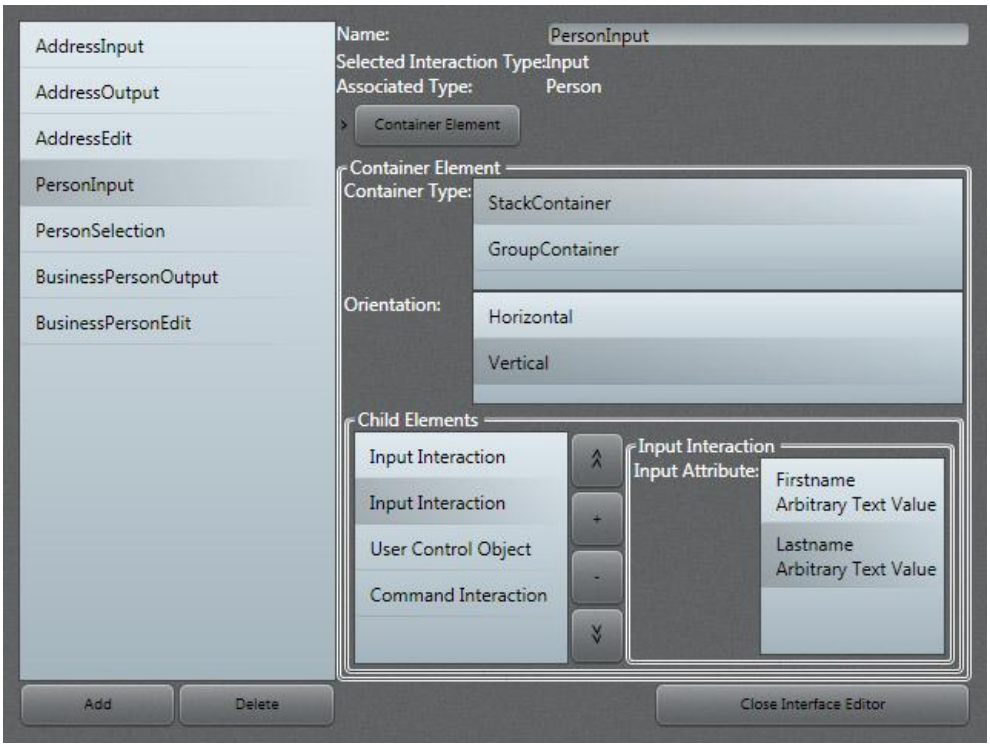


Abbildung 9.3 Anpassung des Benutzersteuerelements *PersonInput*

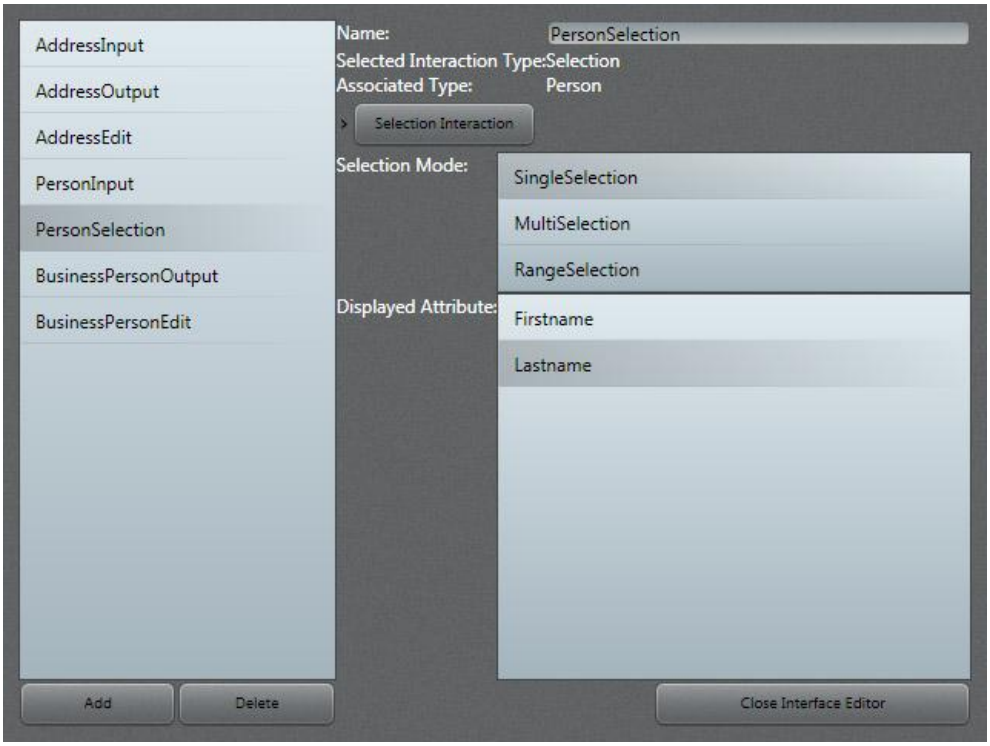


Abbildung 9.4 Anpassung des Benutzersteuerelements *PersonSelection*

Für das *PersonSelection*-Element, das in Abbildung 9.4 dargestellt ist, wird ebenfalls ein neues Benutzersteuerelement erstellt, dessen assoziierter Datentyp auf den

Wert *Person* und der Interaktionstyp auf *Selection* gesetzt wird. Anschließend wählt der Benutzer *SingleSelection* als *Selection Mode* und *Lastname* als *Displayed Attribute* aus. Eine *Command Interaction* kann nicht erstellt werden, sondern wird automatisch hinzugefügt, ebenso wie eine Command Interaction zur zufälligen Auswahl eines Elements aus der Auswahlliste.

Nachdem damit die Oberflächenmodellierung ebenfalls abgeschlossen ist, folgt im nächsten Schritt die Modellierung des Ablaufs.

Schritt 3: Ablaufmodellierung

Der letzte Schritt innerhalb der Modellierung ist die Ablaufmodellierung. Dabei wird als Basis wiederum die Problemstellung verwendet. Zunächst müssen daher die Teilnehmerdaten ermittelt werden. Diese können entweder direkt durch den Benutzer angegeben oder über eine Datenquelle bezogen werden. Innerhalb der Implementierung steht eine Datenquelle zur Verfügung, welche eine Menge von *Person*- und *BusinessPerson*-Objekten bereitstellt.

Innerhalb des Ablaufmodells werden daher zunächst zwei *Task*-Elemente erstellt, denen das Benutzersteuerelement *PersonInput* zugeordnet wird, sowie eine Datenquelle. Da im Rahmen dieser Implementierung nur eine Datenquelle bereitgestellt wird, müssen für diese keine weiteren Einstellungen getroffen werden. Nachdem also die Teilnehmerdaten erfasst wurden, kann nun der Gewinner ermittelt werden. Dazu wird ein weiteres *Task*-Element erstellt, das mit dem Benutzersteuerelement *PersonSelection* verknüpft wird. Anschließend werden drei Verbindungen erstellt, welche die Eingabe-Elemente mit dem Auswahl-Element verbinden.

Diesem Vorgehen entsprechend, werden zwei weitere *Task*-Elemente für die Bearbeitung und Ausgabe von *BusinessPerson*-Objekten erstellt und so verbunden, dass die der Darstellung in Abbildung 9.5 entsprechen.

Schritt 4: Ausführung

Als letzter Schritt erfolgt die Ausführung des Modells. Dabei kann der Benutzer zunächst zwei *Person*-Objekte eingeben, anschließend aus einer Menge von *Person*-Objekten, die neben den erstellten auch die aus der Datenquelle enthält, eines auswählen oder zufällig auswählen lassen und dieses bearbeiten. Zum Ende der Ausführung wird das bearbeitete Element schließlich ausgeben.

Um die Ausführung real nachvollziehen zu können, wird beim Starten der Anwendung EDT4MT automatisch das oben beschriebene Modell erstellt. Der Benutzer kann dieses genauer erkunden und anpassen und die Auswirkungen innerhalb der Ausführung begutachten.

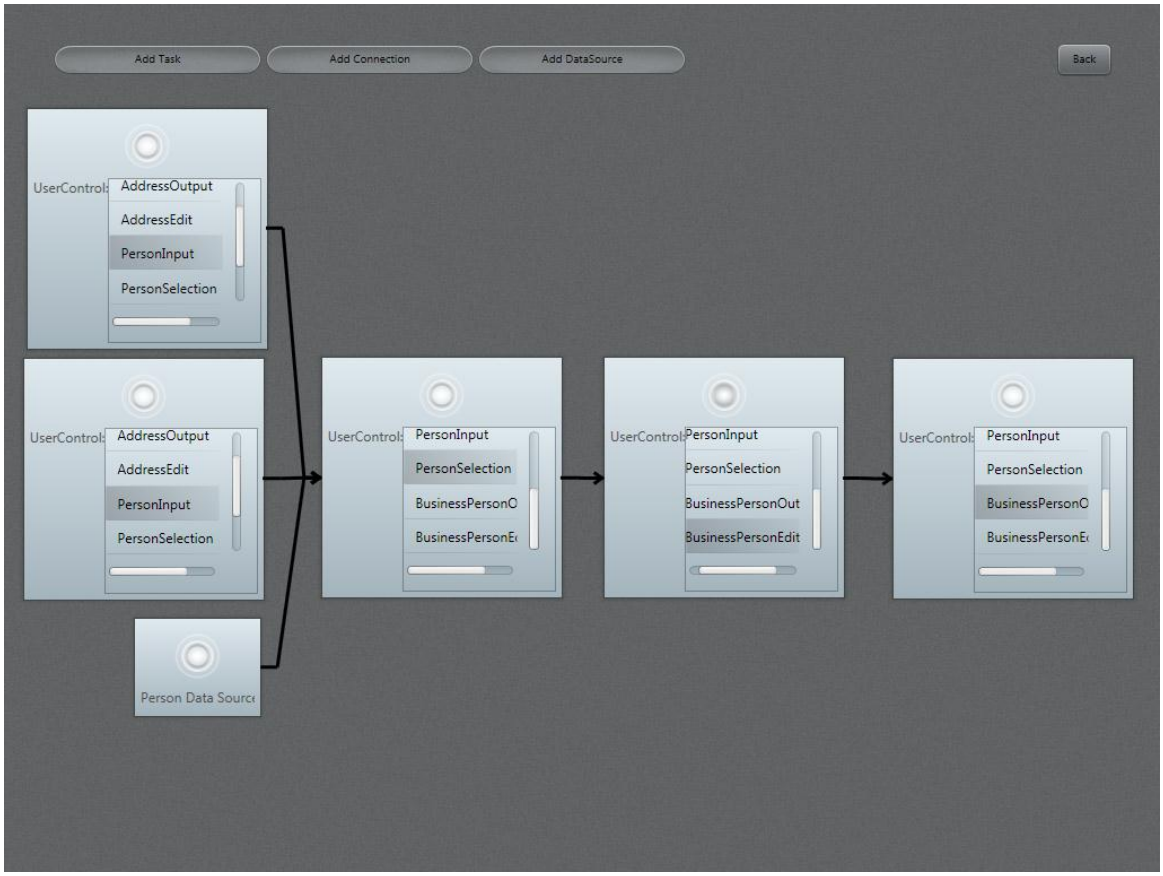


Abbildung 9.5 Ablaufmodell der Gewinnspiel-Anwendung

Neben dem Aspekt, dass die modellgestützte Erstellung von Benutzungsschnittstellen auf Basis des vorgestellten Modells funktioniert, müssen noch zwei weitere Aspekte evaluiert werden. Zum einen ist dies die Eigenschaft, dass auch Endbenutzer mit der Anwendung und dem Modell umgehen können, zum anderen, dass auch Multi-Touch-Benutzungsschnittstellen unterstützt werden.

9.2. Endbenutzertauglichkeit

Eine wesentliche Anforderung sowohl an das Modell, als auch an die darauf basierende Anwendung zur Bearbeitung und Ausführung des Modells, war die Endbenutzertauglichkeit, wie sie in Kapitel 3 gefordert wurde.

Der im Rahmen des vorherigen Abschnitts vorgestellte Ablauf zur Modellierung einer Anwendung und die Bedienung der vorgestellten Editoren wurden absichtlich so konzipiert, dass auch Endbenutzer damit zurechtkommen und sie nach einer Einführung bedienen können.

Dabei wurden die Dialoge und Einstellungsmöglichkeiten so einfach wie möglich aufgebaut, um den Benutzer nicht mit unnötigen oder zu komplexen Oberflächen zu konfrontieren. Selbstverständlich benötigt der Benutzer dennoch eine gewisse Einarbeitungszeit, damit er sich an das Konzept des Modell und dessen Bearbeitung gewöhnen kann.

Durch diese Maßnahmen konnte die Komplexität des Modells und der darauf basierenden Anwendung relativ gering gehalten, wodurch es auch Endbenutzern ohne Programmierkenntnissen möglich ist diese zu verwenden.

Um zu belegen, dass die genannten Schritte auch das gewünschte Ergebnis liefern, dass Endbenutzer die erstellte Anwendung verwenden und damit eigene Anwendungen modellieren und ausführen können, ist eine weitere Untersuchung, beispielsweise durch eine Benutzer- oder Usability-Studie, nötig, die im Rahmen dieser Arbeit allerdings nicht durchgeführt wurde.

Eine weitere in Kapitel 3 genannter Anforderung ist die Einflussnahme auf die erzeugte Oberfläche der modellierten Anwendung, wodurch der Benutzer die erstellte Anwendung an seine Bedürfnisse anpassen kann. Dadurch, dass der Benutzer neben den einzelnen Interaktionselementen auch deren Anordnung mit Hilfe von Layoutelementen festlegen kann, konnte die genannte Anforderung erfüllt werden.

9.3. Multi-Touch-Unterstützung

Der letzte Abschnitt der Evaluation betrifft die Unterstützung von Multi-Touch-Benutzungsschnittstellen, wie sie in Abschnitt 2.4 beschrieben wurden. Da bisherige MB-UIDEs diese nicht unterstützten, war dies ein weiteres wichtiges Ziel dieser Arbeit, wie es beispielsweise in Kapitel 3 genannt wurde.

Um zu zeigen, dass diese Unterstützung durch das Modell und dessen Implementierung möglich ist und die gestellten Anforderungen erfüllt wurden, werden die folgenden Aspekte betrachtet:

Verwendung der Microsoft Surface Plattform

Ein wichtiger Punkt ist die Tatsache, dass die Implementierung auf Basis der Microsoft Surface Plattform durchgeführt wurde und die erstellten Anwendungen auch auf dieser Plattform ausgeführt werden können. Dadurch ist die grundsätzliche Unterstützung der Multi-Touch-Technologie sichergestellt.

Unterstützung von Multi-Touch- und Multi-User-Funktionalitäten

Neben der grundlegenden Unterstützung von Multi-Touch-Technologien werden durch das vorgestellte Modell auch besondere Funktionalitäten dieser Technologie unterstützt. Neben der, durch die Microsoft Surface Plattform gegebenen Unterstützung für Multi-Touch-Interaktionen, beispielsweise bei der Verarbeitung von absolut visuellen Werten, umfasst dies auch die folgenden Multi-Touch- und Multi-User-Funktionalitäten, die in Abschnitt 2.4.5 vorgestellt wurden:

- Geselligkeit
  Die Geselligkeit beziehungsweise Multi-User-Unterstützung wird im Rahmen der vorgestellten Lösung sowohl durch das Modell und damit verbunden auch die erzeugten Anwendungen, als auch durch die Editoren unterstützt.

Innerhalb des Modells ist es möglich im Rahmen der Ablaufmodellierung Aufgaben zu erstellen, die parallel ausgeführt werden. Dadurch können gleichzeitig mehrere Benutzer mit der Anwendung arbeiten.
Die Implementierung des Modells, *EDT4MT*, bietet ebenfalls im Rahmen der Ablaufmodellierung eine Möglichkeit mit mehreren Benutzern gleichzeitig zu arbeiten. Alle Elemente der Ablaufmodellierung, also Aufgaben, Datenquellen und die Verbindungen zwischen diesen, können gleichzeitig bearbeitet werden. So ist es möglich, dass ein Benutzer die Einstellungen verschiedener Aufgaben ändert, während ein anderer die Verbindungen zwischen diesen bearbeitet.

- Übernatürliche Interaktion
  Die erstellten Anwendungen stellen die verschiedenen Benutzersteuerelemente mit Hilfe von *ScatterViewItem*-Elementen dar, welche neben natürlichen Interaktionen, wie dem Verschieben, auch übernatürliche Interaktionen wie die Skalierung der Elemente ermöglichen.
- Scaffolding
  Auch das beschriebene Scaffolding wird durch die vorgestellten Ansätze realisiert. Zum einen wird dies innerhalb der vorgestellten Teilmodelle verwendet, beispielsweise wenn ein Benutzer zunächst lernt, wie er einen eigenen Datentyp zusammenbaut, und anschließend diesen wieder mit anderen Datentypen kombinieren kann. Ähnlich verhält es sich mit den Benutzersteuerelementen.
  Zudem bauen auch die Teilmodelle aufeinander auf. Nachdem der Benutzer weiß, wie er Datentypen modelliert, kann er anschließend darauf aufbauen und Benutzersteuerelemente definieren, welche die verschiedenen Benutzerinteraktionen für die Datentypen bereitstellen.

10. Zusammenfassung und Ausblick

10.1. Zusammenfassung

Ziel dieser Arbeit war es Endbenutzern die Entwicklung von Anwendungen mit Multi-Touch-Benutzungsschnittstellen zu ermöglichen. Um dies zu erreichen wurden zunächst die verschiedenen Interaktionsformen zwischen einem Benutzer und einem Multi-Touch-System, sowie die verwendeten Werttypen analysiert und in Form einer Taxonomie aufbereitet. Zudem wurde eine beispielhafte Abbildung der identifizierten Interaktionselemente in Kombination mit den Werttypen auf reale Steuerelemente innerhalb der Micrsoft Surface Plattform beschrieben.

Basierend auf der erstellten Taxonomie wurde anschließend ein Modell vorgestellt, das für die Modellierung von Multi-Touch-Benutzungsschnittstellen und daraus erzeugten Anwendungen durch Endbenutzer geeignet ist. Das erstellte Modell orientiert sich dabei an bereits vorhandenen Modellen zur Beschreibung von Benutzungsschnittstellen und Anwendungen und erweitert beziehungsweise vereinfacht diese, damit auch Endbenutzer das Modell verwenden können.

Das beschriebene Modell besteht aus drei Komponenten um die Datentypen, die Benutzersteuerelemente zur Verarbeitung dieser Datentypen, sowie die Abfolge der einzelnen Benutzersteuerelemente zu modellieren.

Um zu zeigen, dass dieses Modell die gewünschten Anforderungen erfüllt, wurde zunächst eine Architektur vorgestellt, die zum einen das beschriebene Modell abbildet und zum anderen eine Ausführung einer modellierten Anwendung ermöglicht. Basierend auf dieser Architektur wurde die Anwendung *EDT4MT* implementiert, welche zusätzlich über Editoren verfügt, mit denen die verschiedenen Modellkomponenten bearbeitet werden können.

10.2. Die Zukunft von *EDT4MT*

Da es sich bei der implementierten Anwendung um einen Prototypen handelt, der zum einen nur einen Teil des beschriebenen Modells realisiert und zum anderen nur eine grundlegende Funktionalität ermöglicht, ist eine Weiterentwicklung über diese Arbeit heraus notwendig. Einige Erweiterungsmöglichkeiten der Anwendung *EDT4MT* sind im Folgenden beschrieben.

Vollständige Unterstützung des Modells

Die offensichtlichste Möglichkeit die Anwendung zu erweitern wäre eine vollständige Unterstützung des vorgestellten Modells. Beispielsweise durch die Möglichkeit alle vorgestellten Interaktion-Werttyp-Kombinationen oder Layoutelemente zu verwenden.

Vereinfachte Bedienung

Auch wenn die erstellte Anwendung bereits mit dem Ziel entwickelt wurde die Komplexität in Grenzen zu halten, damit auch unerfahrene Endbenutzer die Anwendung verwenden können, gibt es einige Bestandteile, die verbessert werden können. Dazu zählt beispielsweise die Definition von Formaten, die als Grundlage für formatierte Textwerte dienen. Momentan ist der Benutzer gezwungen die Formate selber zu erstellen, wobei er die Syntax der Implementierung regulären Ausdrücke innerhalb des .NET Frameworks verwenden muss. Ein Editor zur Erstellung dieser Formate mit einer intuitiven Bedienung wäre sehr wünschenswert.

Ebenso sind verschiedene Überarbeitungen im Bereich der Editoren möglich. So könnte die Darstellung von Datentypen sowie der Vererbungshierarchien zwischen diesen geändert werden, beispielsweise durch die Verwendung einer *TreeView*-Komponente, die allerdings erst für die Microsoft Surface Plattform implementiert werden muss.

Des Weiteren besteht im Rahmen des Ablaufeditors die Möglichkeit mehr Rückmeldungen an den Benutzer zu geben, zum Beispiel könnte beim Anlegen einer neuen Verbindung mit Hilfe einer Linie oder Ähnlichem angezeigt werden, von welchem Element die Verbindung ausgeht.

Typsicherheit herstellen

Besonders im Bereich der Ablaufmodellierung wäre es sinnvoll zu überprüfen, ob die Ein- und Ausgangsparameter der verschiedenen Aufgaben kompatibel zueinander sind und falls dies nicht der Fall ist den Benutzer entsprechend darauf hinzuweisen.

Automatische Generierung von Benutzersteuerelementen

Wie bereits in Abschnitt 7.3 beschrieben, wäre eine automatische Generierung von Benutzersteuerelementen auf Basis der modellierten Datentypen möglich. Dadurch könnten im Rahmen der Oberflächenmodellierung zunächst automatisch ein Steuerelement erstellt werden, das anschließend vom Benutzer angepasst werden kann.

Verbesserung der erzeugten Benutzeroberflächenelemente

Die während der Ausführung erzeugten Oberflächen bieten eine ideale Erweiterungsmöglichkeit für softwareergonomische Anpassungen. Wodurch die Akzeptant und Verbreitung der erzeugten Anwendungen erhöht werden könnte.

Unterstützung weiterer Multi-Touch-Plattformen

Neben der in dieser Implementierung verwendeten Microsoft Surface Plattform gibt es einige weitere Plattformen, die auf Basis des vorgestellten Modells unterstützt werden können. Neben PC-Anwendungen können beispielsweise auch Anwendungen für mobile Plattformen wie Apples iPhone entwickelt werden.

10.3. Die Zukunft des Modells

Neben diese Verbesserungen der Anwendung *EDT4MT* bietet auch das vorgestellte Modell Möglichkeiten zur Erweiterung:

Definition weiterer Basisdatentypen

Die im Rahmen des Modells vorgestellten Basisdatentypen sind für viele Anwendungen ausreichend, allerdings können in einigen Anwendungsgebieten auch weitere Datentypen wie Videos oder akustische Werte angebracht sein, welche in das Modell eingepflegt werden müssen.

Trennung von Ablauf und Datenfluss

Eine weitere Erweiterungsmöglichkeit ist die Trennung von Ablauf und Datenfluss, wie es beispielsweise bei einigen Workflow-Systemen der Fall ist, im Rahmen der Ablaufmodellierung. Dadurch wird die Flexibilität in diesem Bereich vergrößert und dem Benutzer stehen größere Freiheiten zur Modellierung von Anwendungen zur Verfügung.

10.4. Die Zukunft von MB-UIDEs

Wie bereits in der Einleitung dieser Arbeit erwähnt, stellt die Modellierung stets ein aktuelles Thema innerhalb der Softwareentwicklung dar. Zusammen mit den Themengebieten End-User-Development und Multi-Touch, die immer weiter an Bedeutung gewinnen, bilden diese drei behandelten Themen ein äußerst spannendes Gebiet innerhalb der Informatik, die auch in Zukunft weiteres Interesse wecken werden.

Nachdem die meisten MB-UIDEs bisher keine Endbenutzerorientierung haben erkennen lassen, könnte diese Arbeit ein erster Schritt in diese Richtung sein und es Endbenutzern ermöglichen ihre eigenen Anwendungen zu erstellen, wodurch die Zufriedenheit der Benutzer mit ihren Anwendungen steigen könnte.

In Zukunft könnte es möglich sein, dass Anwendungen zum Großteil nicht mehr von professionellen Softwareentwicklern erstellt werden, sondern von Endbenutzern, die sich ihre Anwendungen mit ein paar Mausklicks beziehungsweise Berührungen zusammenstellen und anschließend generieren lassen.

Somit bleibt das genannte Ziel dieser Arbeit auch gleichzeitig die Vision folgender Arbeiten. Die vorgestellten Ansätze sind ein erster Schritt in Richtung Softwareentwicklung durch Endbenutzer. Basierend auf dieser Arbeit können weitere Entwicklungen durchgeführt werden, die es letztendlich jedem Benutzer ermöglichen selber professionelle Anwendungen für die verschiedensten Plattformen und Technologien zu entwickeln, ohne jemals programmieren zu müssen.

Literaturverzeichnis

Acevedo Zambrano, Cristián. 2009. *Concept, Interface and Interaction design with moultitouch-gestures for a portfolio application.* Stuttgart, Deutschland : s.n., Oktober 2, 2009.

Barclay, Peter J., et al. 1999. *The Teallach Tool: Using Models for Flexible User Interface Design.* [Dokument] Glasgow, Großbritannien : Department of Computer Studies, University of Manchester; Department of Computer Studies, Napier University; Department of Computer Sciende, University of Glasgow, 1999.

Boehm, Barry W., et al. 2000. *Software Cost Estimation With COCOMO II.* New Jersey, USA : Prentice Hall PTR, 2000.

Browne, Thomas, et al. 1996. *The MASTERMIND User Interface Generation Project.* [Dokument] Georgia, USA : Graphics, Visualization, and Usability Center; Georgia Institute of Technology, 1996.

Costabile, M. F. 2002. End-User Development - Empowering people to flexibly employ advanced information and communication technology. *EUD-Net: Network of Excellence on End-User Development.* [Online] September 23./24, 2002. [Cited: September 17, 2010.] http://giove.isti.cnr.it/projects/EUD-NET/doc/EUD-Net%20D1.3.doc.

Costabile, Maria Francesca, et al. 2004. *Software Environments for End-User Development and Tailoring.* [Dokument] Bari, Italien : Università di Bari, 2004.

Costanza, E. and Robinson, J. 2003. *A Region Adjacency Tree Approach to the Detection and Design of Fiducials.* York : s.n., 2003.

Deutsches Institut für Normung. 2000. *DIN EN ISO 13407: Benutzer-orientierte Gestaltung interaktiver Systeme (ISO 13407:1999).* Berlin : Beuth Verlag GmbH, 2000.

Downs, Rick. 2005. *Using resistive touch screens for human/machine interface.* [Paper] Dallas : Texas Instruments, 2005.

Erler, Thomas and Ricken, Michael. 2005. *UML 2 GE-PACKT.* Bonn : mitp-Verlag, 2005.

Habbe, Christoph. 2005. *GAME: Ein grafischer Editor für Aufgabenmodelle - Konzeption und Implementierung.* [Dokument] Paderborn : Universität Paderborn, 2005.

Han, Jefferson. 2006. Jeff Han demos his breakthrough touchscreen. *TED - Ideas worth spreading.* [Online] The Sapling Foundation, August 2006. [Cited: Mai 31, 2010.] http://www.ted.com/talks/jeff_han_demos_his_breakthrough_touchscreen.html.

—. 2005. *Low-Cost Multi-Touch Sensing through Frustrated Total Internal Reflection.* New York : s.n., 2005.

Janssen, C., Weisbecker, A. and Ziegler, J. 1993. *Generating User Interfaces from Data Models and Dialogue Net Specifications.* [Dokument] Amsterdam : s.n., 1993.

Johnson, Peter, et al. 1988. *Task-Related Knowledge Structures: Analysis, Modelling and Application.* [Dokument] London : s.n., 1988.

Kim, Won Chul and Foley, James D. 1990. *DON: user interface presentation assistant.* [Dokument] Snowbird : s.n., 1990.

Klann, Markus. 2002. *EUD-Net's Roadmap to End-User Development.* [Dokument] Sankt Augustin, Deutschland : Fraunhofer Institute for Applied Information Technology (FhG/FIT), 2002.

Krasner, Glenn E. and Pope, Stephen T. 1988. *A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System.* [Dokument] Mountain View : ParcPlace Systems, Inc., 1988.

Lonczewski, Frank and Schreiber, Siegfried. 1996. *The FUSE-System: an Integrated User Interface Design Environment.* [Dokument] München : s.n., 1996.

Ludewig, Jochen and Lichter, Horst. 2007. *Software Engineering - Grundlagen, Menschen, Prozesse, Techniken.* Heidelberg : dpunkt.verlag GmbH, 2007. 3-89864-268-2.

Markopoulos, P., et al. 1992. *Adept - A task based design environment.* [Dokument] Queen Mary and Westfield College : s.n., 1992.

Marquardt, Bernd. 2007. *Microsoft Windows Presentation Foundation Crashkurs.* Unterschleißheim : Microsoft Press Deutschland, 2007.

Mehta, Nimish. 1982. *A Flexible Machine Interface.* Toronto, Kanada : s.n., 1982.

Meixner, Gerrit and Görlich, Daniel. 2008. *Aufgabenmodellierung als Kernelement eines nutzerzentrierten Entwicklungsprozesses für Bedienoberflächen.* [Dokument] Kaiserslautern : s.n., 2008.

Microsoft Corporation. Microsoft Surface Architecture. *Microsoft Developer Network.* [Online] Microsoft Corporation. [Cited: Juni 9, 2010.] http://msdn.microsoft.com/en-us/library/ee804755(v=Surface.10).aspx.

—. 2009. *Microsoft Surface User Experience Guidelines.* [Dokument] s.l. : Microsoft Corporation, 2009.

NUI Group. 2008. Diffused Illumination. *NUI Group Community Wiki.* [Online] September 5, 2008. [Cited: Juni 7, 2010.] http://wiki.nuigroup.com/images/a/a0/Rearditouch.jpg.

—. 2009. NUI Group Community Wiki. *Hardware.* [Online] Oktober 22, 2009. [Cited: Juni 7, 2010.] http://wiki.nuigroup.com/Hardware.

Object Management Group, Inc. 2008. *Common Object Request Broker Architecture (CORBA) Specification, Version 3.1, Part 1: CORBA Interfaces.* [Dokument] Needham, USA : Object Management Group, Inc., 2008.

Paternò, Fabio. 2000. *Model–based design and evaluation of interactive.* [Dokument] Pisa : Springer-Verlag, 2000.

Platt, David S. 2006. *Why Software SUCKS... and what you can do about it.* Boston, USA : Pearson Education, INC., 2006. ISBN 0-321-46675-6.

Schlegel, Thomas. 2002. *Entwurf und Erprobung eines software-gestützten Verfahrens zur Anwendung software-ergonomischer Methoden in frühen Phasen der Anwendungsentwicklung.* Stuttgart, Deutschland : s.n., Juli 30, 2002.

Schlungbaum, Egbert and Elwert, Thomas. 1996. *Automatic User Interface Generation from Declarative Models.* [Dokument] Rostock, Deutschland : Universität Rostock, Fachbereich Informatik, 1996.

Schlungbaum, Egbert. 1996. *Model-based User Interface Software Tools - Current state of declarative models.* [Dokument] Atlanta : Georgia Institute of Technology, 1996.

Schöning, Johannes, et al. 2008. *Multi-Touch Surfaces: A Technical Guide.* München, Deutschland : s.n., 2008.

Surface HQ. 2009. Surface HQ :: Information :: Hardware and Specs. *Surface HQ.* [Online] November 27, 2009. [Cited: Juni 9, 2010.] http://www.surfacehq.com/content/microsoft-surfaces-hardware.

Szekely, P., et al. 1996. *Declarative interface models for user interace construction tools: the MASTERMIND approach.* [Dokument] s.l. : University of Southern California, Information Sciences Institute; Georgia Institute of Technology; Universidad Autonoma de Madrid; University of Technology, Graz;, 1996.

Trætteberg, Hallvard. 2002. *Model-based User Interface Design.* Trondheim, Norwegen : s.n., Mai 14, 2002.

Trümper, Jonas. 2007. *Multi-Touch-Systeme und interaktive Oberflächen.* Berlin, Deutschland : s.n., 2007.

Uhr, Holger. 2001. *Die Definition und Ausführung von benutzerspezifischen Webprozessen mit TOMBOLA.* [Dokument] Paderborn : s.n., 2001.

White, Stephen A. 2004. *Introduction to BPMN.* [Dokument] s.l. : IBM Corporation, 2004.

Wobbrock, Jacob O., Morris, Meredith Ringel and Wilson, Andrew D. 2009. *User-Defined Gestures for Surface Computing.* [Dokument] Boston : ACM, 2009.

Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und nur die angegebenen Hilfsmittel verwendet habe.

(Marc Dennis Paulenz)