Institut für Visualisierung und Interaktive Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3154

Migration und Anpassung von Dialoganwendungen für berührungsempfindliche Bildschirme

Christian Wimmer

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Thomas Ertl
Betreuer	Jun.-Prof. Dr. Thomas Schlegel, Dipl.-Phys. Michael Raschke
begonnen am:	1. März 2011
beendet am:	31. August 2011
CR-Klassifikation:	D.2.2 Design Tools and Techniques D.2.3 Coding Tools and Techniques D.2.10 Design D.2.11 Software Architectures H.5.2 User Interfaces

Abstract

Berührungsempfindliche Bildschirme wurden in den letzten Jahren immer günstiger und beliebter. Die grafischen Benutzeroberflächen von Anwendungen für Desktop- und Laptopcomputer sind jedoch für eine solche Eingabeart nicht ausgelegt und neue Programmversionen ändern diesen Umstand auch kaum. In dieser Diplomarbeit wird daher eine Methode vorgestellt, um Entwicklern die Möglichkeit zu geben, ihre Dialoge automatisch auf die Benutzung für Touch anzupassen. Dafür wurde ein Prototyp entwickelt, der die automatische Transformation von Dialogen in der Dialogbeschreibungssprache XAML erlaubt. Die Migration der Dialoge wird durch eine parametrisierbare Transformationsvorschrift in XSLT durchgeführt. Zusätzlich kann mit einer beliebigen .NET Sprache in den Migrationsprozess eingegriffen werden. Um die Funktionsfähigkeit der beschriebenen Transformation zu zeigen, wurde im Rahmen dieser Diplomarbeit eine Studie auf einem berührungsempfindlichen Bildschirm durchgeführt. Dazu wurden zwei Dialoge mehrmals transformiert, indem die Steuerelemente automatisch verändert und ersetzt wurden. Außerdem wurden einige Steuerelemente für die Studie analysiert und entsprechend von Gestaltungsregeln für die Benutzung mit den Fingern angepasst, um in der Studie bewertet zu werden.

Touch-sensitive screens became cheaper and more popular over the last few years. But the graphical user interfaces of applications for desktop and also laptop computers do not support such a type of entry by default. Therefore, in this diploma thesis a new method is proposed that supports software engineers in adapting their dialogs automatically to touch input. To implement this method, a prototype application was developed to migrate dialogs that were created in the description language XAML. The migration itself was performed by using a mapping process in XSLT. In addition, .NET programming languages can be integrated into the process to influence the migration. To show the functionality of the migration process a user study was conducted on a touch-sensitive monitor screen. Two dialogs were transformed several times by changing and replacing control elements automatically. Also some control elements were analyzed and adapted to conform to design rules of touch input. In the study users had to test and evaluate the transformed dialogs.

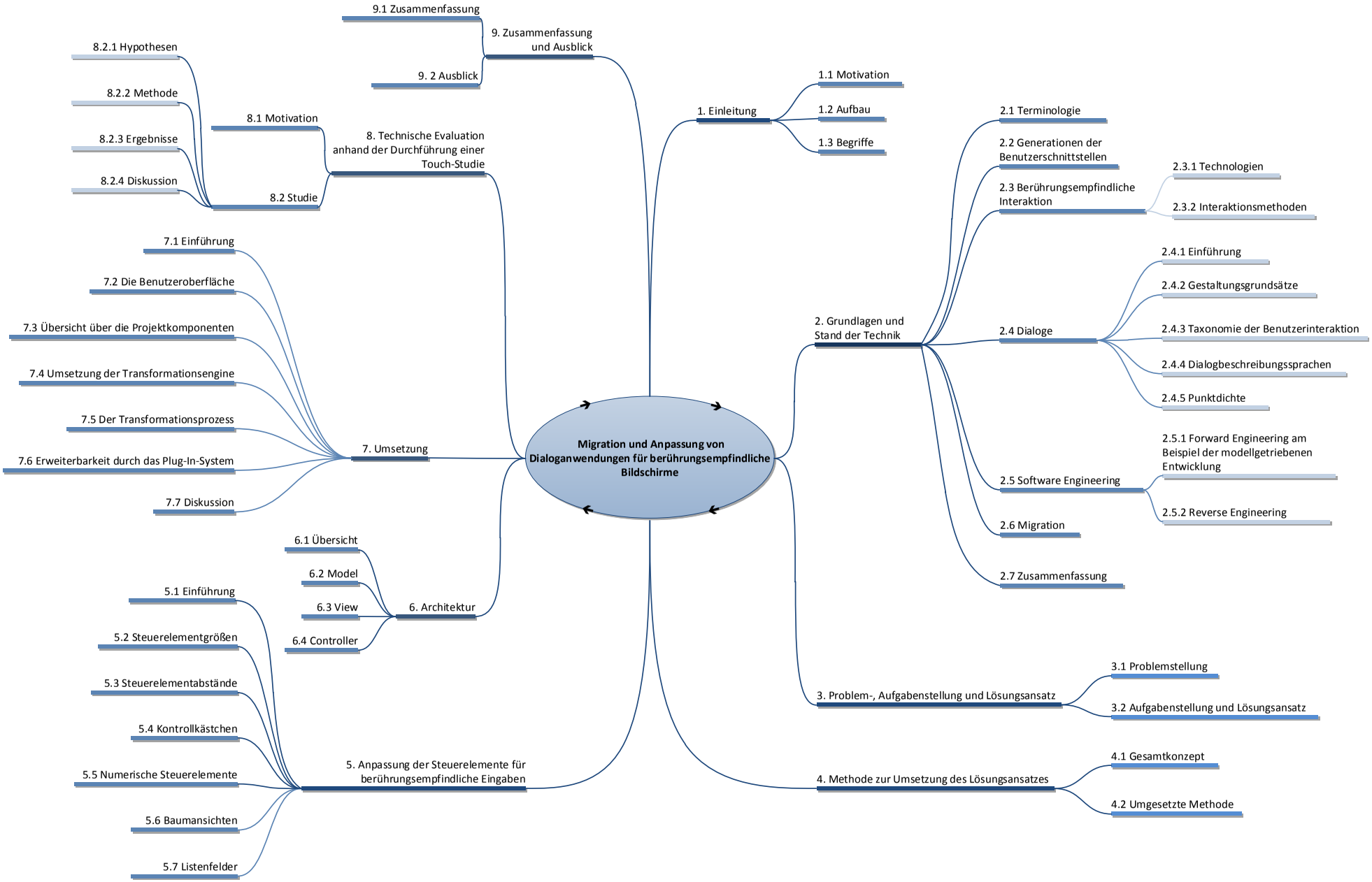
Danksagung

Auch wenn Johann Wolfgang von Goethe einst sagte: „Leider läßt sich eine wahrhafte Dankbarkeit mit Worten nicht ausdrücken.“, so will ich dennoch meine Dankbarkeit hier niederschreiben, damit sie nicht vergessen wird. Zu allererst ist diese Diplomarbeit meiner Familie gewidmet, die mich immer und ausnahmslos bis zum Abschluss begleitete und unterstützte. Danke an meine Eltern! Danke an meine kleine Schwester; am Montag ist wieder Training! Vielen Dank möchte ich den Teilnehmern der Studie zukommen lassen. Außerdem geht ein großes Dankeschön an die Korrekturleser, die sich durch den vielen Text gearbeitet haben. Danke Onur und Alina! Weiterhin danke ich auch meinen Betreuern, die mir immer mit Rat zur Seite gestanden haben, als ich mal wieder ratlos war. Danke dir Thomas, ohne dich wäre dieses Thema nie zu einem Diplomarbeitsthema gereift! Und auch Dankeschön an euch Michael und Florian, für eure konstruktive Kritik und eure Mühen; unsere Treffen haben mir immer viel Freude bereitet! Sollte ich noch jemanden vergessen haben, dann möchte ich mich hiermit entschuldigen und meinen Dank nachreichen. War doch keine Absicht!

Zuletzt möchte ich auch an Sie, den Leser, meinen Dank richten. Eine Diplomarbeit zu schreiben, ohne dass jemand sie liest, ist wie einen guten Wein zu keltern, ohne dass ihn jemand trinkt. Beide verstauben nur im Regal und geraten in Vergessenheit. Lassen Sie mir daher Ihre Meinung zukommen.
Schreiben Sie mir: chriswimmer@gmx.de

Inhaltsverzeichnis

Inhaltsverzeichnis als Mind-Map



Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Diese Seite wurde absichtlich leer gelassen

1 Einleitung

In Mitten der Schwierigkeiten liegt die Möglichkeit.

Albert Einstein
theoretischer Physiker

1.1 Motivation

Berührungsempfindliche Bildschirme (Neudeutsch auch Touchbildschirm oder engl. touchscreen) werden von Jahr zu Jahr attraktiver [Schöning, et al., 2008]. Besonders im Bereich der mobilen Geräte steigt die Attraktivität von PDAs, Tablets, Smartphones und E-Book-Reader mit berührungsempfindlicher Oberfläche. Kiosk- und Informationssysteme mit Fingereingabe sind mittlerweile unter Anderem schon in vielen Kaufhäusern und Banken zum Standard für Kunden geworden. Die Vorteile liegen auf der Hand. Die Benutzer können direkt mit den Elementen (insbesondere Schaltflächen) interagieren und zusätzlicher Platz und Kosten für Hardware wie Tastatur und Maus entfallen.
Doch zu Hause und bei der Arbeit sind LCD- und TFT-Monitore noch berührungs**un**empfindlich. Für Softwarehersteller bedeutet dies keinen Aufwand in die Benutzbarkeit ihrer Software für Finger stecken zu müssen. Dies lässt wiederum Benutzer keinen Sinn darin sehen berührungsempfindliche Bildschirme zu kaufen, die keinen wirklichen Mehrwert bringen und stattdessen Frustration erzeugen, etwa durch zu kleine Schaltflächen. Wenn der Lustaspekt nicht vorhanden sei, so [Novak, et al., 2009], dann führe dies zur Abneigung und schließlich zum Nichtnutzen der Technologie.

Mittlerweile unterstützt das neuste Betriebssystem Windows 7 aus dem Hause Microsoft standardmäßig die berührungsempfindliche Eingabe zur Bedienung von Anwendungen. Dies geschieht seit Windows Vista auch dann, wenn die Anwendung selbst nicht für die Bedienung mit einem Finger entworfen wurde. Doch diese Software ist nicht immer einfach mit den Fingern zu bedienen, da Steuerelemente und das gesamte Bedienkonzept nur für die Maus- und Tastatureingabe funktionieren (z.B. kleine Schalterflächen). Daher bietet [Microsoft] eine Programmierschnittstelle an, das sogenannte Windows Touch SDK für Softwareentwickler, um Multi-Touch sowie Gesten zu unterstützen. Auch wurden von [Microsoft] einige Gestaltungsrichtlinien für die Fingerbedienung unter Windows 7 herausgegeben. Solche Richtlinien gibt es auch von anderen Herstellern für deren jeweils eigene Plattform (eine Übersicht gibt es bei [Experience Dynamics]). Doch eine Umsetzung der Gestaltungsrichtlinien kann sich schwierig gestalten, da die Regeln oftmals nur in einer unscharfen Schriftform vorliegen und dadurch keine automatische Anwendung z.B. in der Programmierung erlauben. Außerdem kann bereits eine einzelne Anwendung viele unterschiedliche Dialoge enthalten, so dass ein großer Migrationsaufwand entstehen würde, der eine Anpassung für berührungsempfindliche Bildschirme nicht rechtfertigt.

Ziel dieser Diplomarbeit ist es daher die Bedienung von Anwendungen mit den Fingern zu verbessern. Dazu wird ein Konzept aufgezeigt, das den Anwendungsentwickler bei der Migration seiner vorhandenen Dialoge unterstützen soll, indem es die Migration zu Teilen automatisiert.

1.2 Aufbau

Die Arbeit führt mit Kapitel 2 in die Grundlagen der berührungsempfindlichen Bildschirme und Generationen von Benutzerschnittstellen ein. Zudem werden grundlegende Technologien und Methoden für eine berührungsempfindliche Bedienung erläutert. Der Interaktion folgt ein Kapitel über Dialoge, deren grundsätzliche Gestaltung sowie Sprachen, um Dialoge zu entwickeln. Das Kapitel 2 schließt mit dem aktuellen Stand der Technik bei der Migration von Dialogschnittstellen ab.

Das Kapitel 3 führt in die aktuellen Probleme der berührungsempfindlichen Interaktion ein und stellt die Aufgaben der Diplomarbeit in einem Überblick dar. Zu den vorgestellten Problemen und Aufgaben wird außerdem bereits ein Lösungsansatz gezeigt und erläutert.

Um die Probleme und Schwächen der Dialoge für berührungsempfindliche Bildschirme aus Kapitel 3 zu überwinden, wird in Kapitel 4 eine generische Methode vorgestellt. Diese Methode wird zudem konkretisiert, um eine umsetzbare Grundlage für die folgenden Kapitel zu erhalten.

Basierend auf den Grundlagen der Dialoggestaltung werden in Kapitel 5 mögliche Anpassungen der Steuerelemente in Dialoge für berührungsempfindliche Bildschirme analysiert und formuliert. Außerdem werden neue Arten von Steuerelementen vorgestellt, welche die vorhandene Elemente für die berührungsempfindliche Bedienung ersetzen und dadurch die Bedienbarkeit verbessern sollen.

Für die aus Kapitel 4 bekannte Methode wird in Kapitel 6 eine Architektur präsentiert. Die Architektur dient als Grundlage für eine prototypische Umsetzung. Dieser Prototyp wird in dem darauf folgenden Kapitel 7 eingehend erläutert. Darin wird außerdem der Transformationsprozess vorgestellt und wie dieser erweitert werden kann.

Die Umsetzung wird anschließend in einer Studie mit mehreren Benutzern evaluiert. Dazu wurden zwei transformierte Dialoge mit den aus Kapitel 5 angepassten Steuerelementen angepasst und auf deren Benutzbarkeit auf einem berührungsempfindlichen Bildschirm überprüft. Im Kapitel 8 werden dafür Hypothesen aufgestellt, um die angepassten Dialoge in der Studie auf die Probe stellen zu können. Die daraus gewonnenen Ergebnisse werden zudem präsentiert und diskutiert.

Kapitel 9 schließt die Diplomarbeit mit einer Zusammenfassung aller Themen ab. Außerdem wird ein Ausblick auf mögliche Erweiterungen sowie Verbesserungen der entwickelten Methode und des Prototyps gegeben.

1.3 Begriffe

In dieser Diplomarbeit werden durchgehend alle Begriffe in der deutschen Sprache wiedergegeben, sofern dies möglich ist und der Kontext dies zulässt. Dadurch soll eine bessere Lesbarkeit der Diplomarbeitsschrift ermöglicht werden, denn Softwareentwickler sollten stets ihre Arbeitsdomäne auch Außenstehenden, sprich Nichtentwicklern oder Kunden, begreiflich machen können. Für viele verwendete, aber englische Begriffe existieren deutsche Entsprechungen. In der Anwendungsentwicklung trifft dies insbesondere auf die Namen von Steuerelementen zu. ComboBox, Edit und Button mögen vielleicht Programmierern wohl bekannt sein, doch sind diese außerhalb der Entwicklung für viele Menschen (und Leser) nicht unbedingt aussagekräftig. Für einen Softwareentwickler sollte es daher nicht zu viel verlangt sein, beide Welten zu kennen und zwischen ihnen eine Brücke schlagen zu können. In diesem Sinne nutzt der Entwickler die englischen Begriffe für die Entwicklung und auf der anderen Seite die deutschen Begriffe für die Erklärung. Für diejenigen, die die deutschen Namen der Steuerelemente noch nicht kennengelernt haben, wurde eine Vergleichstabelle (Tabelle 17) im Anhang erstellt. Diese beinhaltet neben einem Bild sowohl die deutschen als auch die englischen Bezeichner.

2 Grundlagen und Stand der Technik

Sag es mir – und ich werde es vergessen.
Zeige es mir – und ich werde mich daran erinnern.
Beteilige mich – und ich werde es verstehen.

Lao Tse
chinesischer Philosoph

Das Grundlagenkapitel beginnt mit der Einführung von grundlegenden Begriffen, die im weiteren Verlauf der Arbeit verwendet werden (Kapitel 2.1). Darauf folgt eine Einführung in die Evolution der Benutzerschnittstellen, von den Schnittstellen mit Kommandozeile bis hin zu den organischen Benutzerschnittstellen (Kapitel 2.2). Wie berührungsempfindliche Bildschirme funktionieren und welche Interaktionsmöglichkeiten dazu existieren, wird im Kapitel 2.3 präsentiert. Das Kapitel 2.4 beschäftigt sich mit dem Thema Dialoge. Darin wird erläutert, welche grundlegenden Gestaltungsgesetze für Dialoge existieren, wie die Taxonomie für Benutzerinteraktionen aussieht und welche Arten von Benutzungsschnittstellen vorkommen können. Das Kapitel 2.5 behandelt die modellgetriebenen Entwicklung. Darin werden Ansätze zur Entwicklung sowie Rückentwicklung von Oberflächen mit Modellen aufgezeigt. Das Grundlagenkapitel schließt mit dem Kapitel 2.6 ab, welches weitere Ansätze und Lösungen präsentiert, um Oberflächen von einer Plattform auf eine andere zu überführen.

2.1 Terminologie

In der heutigen Zeit und Welt ist es mittlerweile unmöglich geworden jeden Begriff genau zu kennen. Dies gilt besonders für eine sich rasant verändernde IT Welt, die von englischen Begriffen wie Touch, Smartphone, Middleware, Tablet, E-Book oder VPN regen Gebrauch macht und auch immer wieder neue Wörter (z.B. *Twitter*) erfindet. Doch auch deutsche Begriffe benötigen oftmals eine (erneute) Erklärung, um Unklarheiten oder Verwechslungen beim Lesen zu vermeiden, weil Autor und Leser von unterschiedlichen Definitionen ausgehen. Daher sind im Folgenden alle Begriffe aufgelistet, die in dieser Arbeit verwendet und damit als bekannt vorausgesetzt werden.

Benutzbarkeit (Usability)

Usability, übersetzt Bedienbarkeit oder Benutzbarkeit, beschreibt wie gut sich ein interaktives System von einem Benutzer erschließen und erlenen lässt [Machate, 2003].
Usability wurde auch unter dem Begriff der Gebrauchstauglichkeit eines Produkts in der Norm ISO 9241-11 definiert. Darin ist die Gebrauchstauglichkeit *„das Ausmaß, in dem ein Produkt durch spezifische Benutzer in einem spezifischen Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen.“*.

Interaktion

Die *Interaktion* beschreibt ein wechselseitiges Handeln zwischen Menschen, Prozessen und Geräten [Fischer, et al., 2008]. Dabei beschränkt sich die Interaktion zwischen Mensch und Gerät auf die Manipulation von Benutzungsschnittstellen (siehe Kapitel 2.2), da Maschinen nicht interagieren, sondern auf Eingaben reagieren. Systeme, die über Benutzungsschnittstellen gesteuert werden, werden interaktive Systeme genannt. Besitzt ein

interaktives Computersystem eine berührungsempfindliche Benutzungsschnittstelle und wird es durch Berührung oder Anfassen gesteuert, so handelt es sich bei diesem Vorgang um eine *berührungsempfindliche Interaktion*.

Modell

Ein *Modell* ist ein unvollkommenes und interpretiertes Abbild eines komplexeren Systems oder der realen Welt. Bei der Erschaffung von Modellen beschränkt man sich auf eine Untermenge der Vorgänge und Gesetze eines Systems und versucht durch die Interpretation des Modells das System besser verstehen zu können [Ludewig, et al., 2007]. Der Prozess der Modellbildung wird Abstraktion genannt und besteht aus dem Erkennen und Weglassen von Eigenschaften, die nicht von Interesse sind. Durch diesen Vorgang wird aus etwas Konkretem etwas Abstraktes.

Touch (Multi-Touch)

Der Begriff *Touch* ist ein englisches Substantiv und bedeutet Kontakt oder Berührung. Touch wird oftmals in Verbindung mit anderen Begriffen zu einer neuen Wortkombination vereint, um auszudrücken, dass eine Maschine mit einer Berührung durch ein Werkzeug oder Finger bedient werden kann. Zum Beispiel Touchpad, ein Mausersatz oder Touchbildschirm, ein berührungsempfindlicher Bildschirm. Die Interaktion kann aktiv oder passiv erfolgen. Die passive Form, beispielsweise auf einem Fingerpad am Laptop, unterscheidet sich kaum von der Eingabe mit einer Maus, die Elemente auf dem Bildschirm über einen Zeiger manipuliert. Erst mit berührungsempfindlichen Bildschirmen können Elemente mit dem Finger direkt auf dem Bildschirm anvisiert und bedient werden. Zudem wird die Interaktion durch neue Technologien (siehe Kapitel 2.3.1) und Formen von Benutzungsschnittstellen (siehe Kapitel 2.2) auch mit mehreren Fingern (*Multi-Touch*) oder sogar mit mehreren Benutzern gleichzeitig (*Multi User-Touch*) möglich.
Gängige Touch-Geräte sind heutzutage: Tablet-PCs, Smartphones, E-Books, Touchpads und Bildschirme aller Arten.

2.2 Generationen der Benutzerschnittstellen

Eine Benutzerschnittstelle (eigentlich Benutzungsschnittstelle) ist der Teil eines Computersystems, der es dem Benutzer ermöglicht mit dem Computer zu interagieren. Die ersten Benutzerschnittstellen bestanden aus Schaltern und Lampen, die den Zustand des Computers ändern und anzeigen konnten. Diese Art von Schnittstelle war jedoch nicht einfach zu benutzen, daher kamen bald zeichen- bzw. befehlsorientierte Benutzerschnittstellen auf.

Seit den Anfangszeiten der Arbeit mit Computern wird daher versucht die Arbeit mit dem Computer einfachen zu gestalten, indem die Schnittstelle den Bedürfnissen der Nutzer angepasst wird. Der Computer soll einfach, effizient und angenehm zu bedienen sein [Rädle, 2009]. Aus diesem Grund wurden in den ISO Normen die Norm **ISO 9241-110** definiert, welche Benutzungsschnittstelle wie folgt definiert:

„*all components of an interactive system (software or hardware) that provide information and controls for the user to accomplish specific tasks with the interactive system*" (entnommen aus [Geis, 2006])

Mittlerweile gibt es verschiedenste Arten von computergestützten Schnittstellen. In ihrer Art und Weise wie sie vom Benutzer bedient werden, können nach [Rädle, 2009] und [Chapman, 2008] die folgenden Schnittstellen unterschieden werden:

- Kommandozeilenorientierte (Benutzer-)Schnittstelle (Command Line Interface - CLI)
- Grafische Benutzerschnittstelle (Graphical User Interface - GUI)
- Natürliche Benutzerschnittstelle (Natural User Interface - NUI)
- Organische Benutzerschnittstelle (Organic User Interface - OUI)

Kommandozeilenorientierte Schnittstelle

Eine der ersten Benutzerschnittstellen, die mit Software verwirklicht wurden, sind die kommandoorientierten Schnittstellen. Ihre Hauptaufgabe besteht darin Kommandobefehle in Form von Texteingaben über eine Tastatur anzunehmen, zu interpretieren und das Resultat auf dem Bildschirm auszugeben.

Obwohl die Kommandozeile (auch Eingabeaufforderung oder engl. command prompt) einer der ersten Schnittstellen war, ist sie heute noch in allen PC-Betriebssystemen verfügbar. Ihr Vorteil liegt in der schnelleren Abarbeitung von Befehlen sowie einem größeren Befehlsumfang als der der grafischen Schnittstellen. Diese Vorteile macht die Kommandozeile deshalb sehr attraktiv für Experten, die immer wiederkehrende Aufgaben effizient erledigen wollen. Die Einstiegshürde für Anfänger ist jedoch entsprechend hoch und erfordert zudem nicht selten ein intensives Studium der möglichen Befehle.

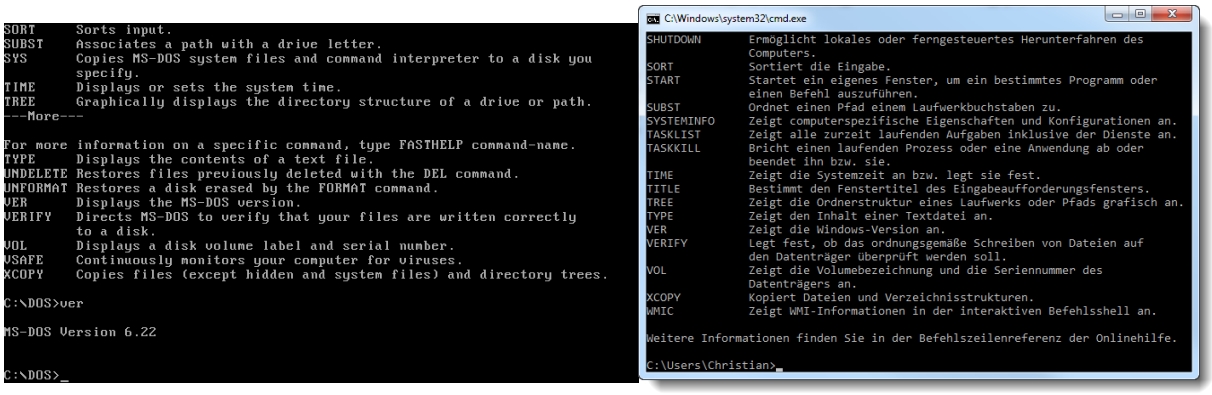


Abbildung 1 Die meisten Befehle aus MS DOS, wie copy, dir, echo, format, goto, mkdir, more, path und viele mehr werden noch heute in der Kommandozeile unter Windows unterstützt.

Grafische Schnittstelle

Um die Einstiegshürde zu senken und auch einer breiteren Bevölkerungsschicht den Umgang mit Computern zu erleichtern wurden grafische Benutzerschnittstellen eingeführt. Beispielsweise kamen recht schnell für das kommandozeilenorientiere Microsoft DOS grafische Dateimanager zum Einsatz wie die MS DOS Shell oder die ersten Versionen von Microsoft Windows. Die Oberflächengrafik bestand dabei oftmals nur aus geschickt zusammengesetzten (farbigen) Zeichen, die u.a. Fensterrahmen bildeten. Die eingesetzte Bildschirmauflösung war jedoch immer auf die verwendete Bildschirmauflösung der Kommandozeile beschränkt, d.h. gewöhnlich 80 Zeichen in der Breite und 25 Zeichen in der Höhe für MS-DOS Systeme. Diese Auflösung wurde auch als Textmodus bezeichnet.

Mit der Einführung von Apples Lisa OS Anfang der 1980er Jahre wurden Betriebssysteme dann zunehmend mit echten grafischen Schnittstellen ausgestattet. Zudem bildeten sich die ersten Begriffe wie „Desktop“, um die Einstiegshürde für die Benutzung von Computern zu reduzieren [Rädle, 2009].

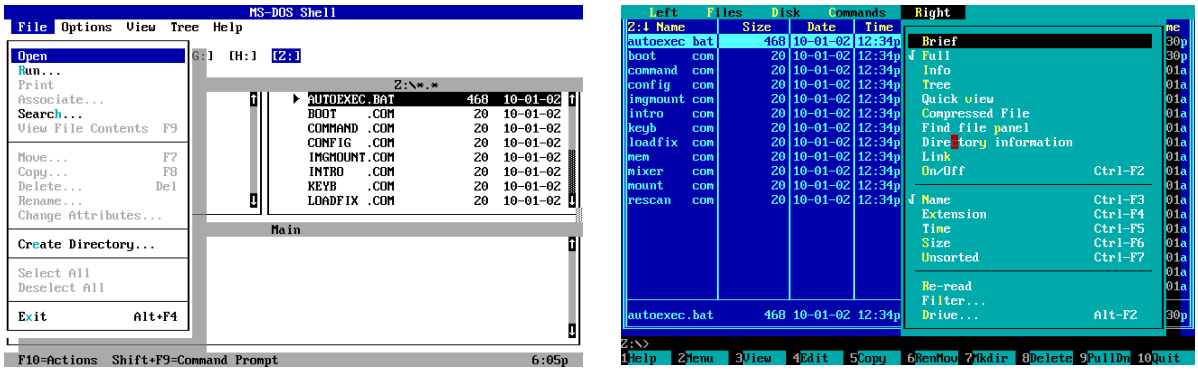


Abbildung 2 Der Dateisystemmanager MS DOS Shell und Norton Commander nutzten Textzeichen für Fenster und Menüs.

Die neuen grafischen Betriebssysteme nutzen die Metapher des Fensters (engl. window), um Texte oder Grafiken darzustellen. Anwendungen können nicht mehr den gesamten Bildschirm für sich alleine nutzen, wie es auf der Kommandozeile üblich war, sondern müssen ihre Inhalte und Benutzerinteraktion auf das Innere des Fensterbereichs beschränken. Zudem werden bei grafischen Betriebssystemen nicht mehr alle Steuerbefehle über die Tastatur eingegeben, sondern durch anklickbare Symbole (engl. icons) präsentiert. Diese Symbole stellen den dahinterliegenden Befehl durch eine stark vereinfachte Grafik dar und befinden sich z.B. auf Schaltflächen, in der neuen Multifunktionsleiste (genannt Ribbon) aus Microsoft Office oder am bekanntesten Ort: auf dem Desktop.

Ein weiteres wichtiges Mittel zur Interaktion bei grafischen Oberflächen sind Menüs (deutsch für Befehlsübersicht). Eine große Anzahl von Befehlen kann auf diese Weise übersichtlich aufgelistet werden. Dazu werden die Befehle in einer Baumstruktur mit Menüeintragen und Untermenüs (oder Untermenüeinträgen) strukturiert. Gleichartige Befehle können so in ein Untermenü positioniert und durch einen Überbegriff präsentiert werden.

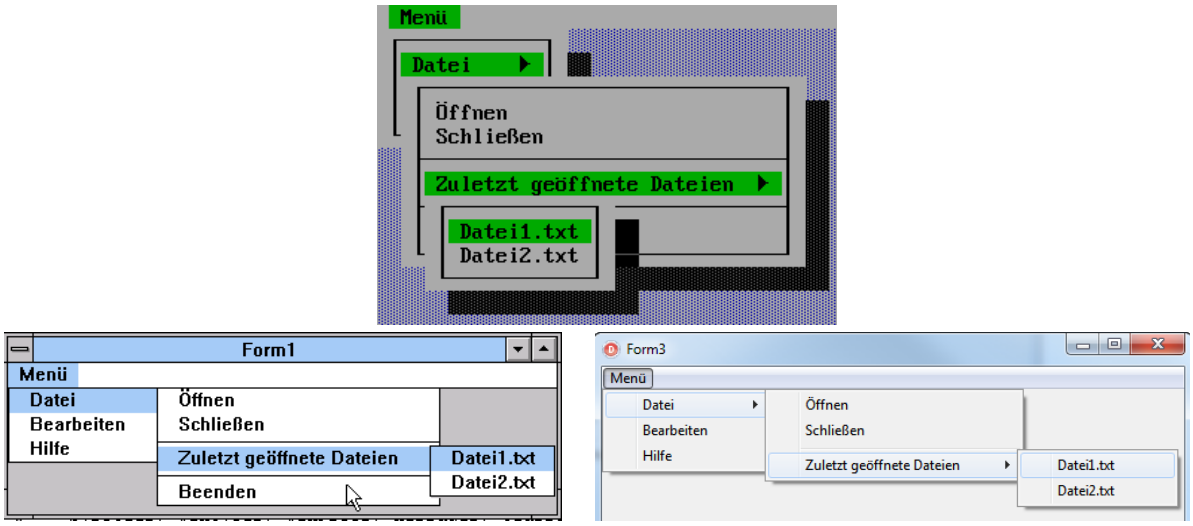


Abbildung 3 Menüs können in einer Baumstruktur dargestellt werden. Die Struktur hat sich nicht geändert, jedoch die Darstellung: oben ein Menü im Textmodus unter DOS, unten links in Windows 3.1 und rechts Windows 7.

Eng verwandt mit grafischen Benutzerschnittstellen ist die Nutzung eines Zeigegeräts oder auch Maus genannt. Der Benutzer manipuliert damit die Steuerelemente der grafischen Oberfläche indirekt über einen Mauszeiger (engl. pointer), ohne dass er dafür lange Befehlsketten auf der Tastatur eingeben müsste. Die Maus ist daher ein einfaches Werkzeug, um Steuerelemente anzuklicken oder zu verschieben. Ihr Erfolg besteht seit den 1970ern und ist bis heute ungebrochen.

Natürliche Benutzerschnittstelle

Natürliche Benutzerschnittstellen ergeben sich aus dem Nachteil der grafischen Benutzeroberflächen, die dem WIMP (Window, Icon, Menu und Pointer) Paradigma folgen. Die Nutzung von Maus und Tastatur hat zwar Vorteile, trotzdem werden mit diesen Werkzeugen nur die Werte innerhalb des unsichtbaren Speichers manipuliert, indem Daten in Textfelder eingegeben oder Schiebregler bewegt werden. Das wichtigste Werkzeug des Menschen, die Hand, wird dadurch zu einem passiven Hilfsmittel. Sie wird durch die Technik unnötig eingeschränkt.
Mit den natürlichen Benutzerschnittstellen wird deshalb das Objekt selbst in den Vordergrund gerückt und direkt durch Fingereingaben manipulierbar gemacht, indem es berührt, vergrößert, verkleinert oder verschoben wird. Die natürlichen Oberflächen nutzen die menschlichen Bewegungen und Gesten mit einem oder mehreren Fingern (Multi-Touch), um virtuelle Elemente auf dem Bildschirm zu manipulieren als wären es physikalische Objekte. Der Benutzer kann dadurch die Oberfläche intuitiv und ohne bewusstes Vorwissen bedienen [Koller, et al., 2010]. Die größte Änderung dabei, so schreibt [Rädle, 2009 S. 12], ist das Fehlen einer physikalischen Tastatur und Maus, wo sie unnötig geworden sind (z.B. beim Internetseiten lesen). Derzeit lässt sich nur schwer abschätzen, ob und in wieweit natürliche Benutzerschnittstellen die herkömmlichen GUIs mit Fenstern und Menüs verdrängen können. Dies mag auch daran liegen, dass bis jetzt (2011) nur spielerische oder demonstrative Umsetzungen (siehe Abbildung 4) von NUI vorhanden sind. Dabei sehen die Umsetzungen vielversprechend aus, doch sie besitzen laut einer Studie des Fraunhofer Instituts kaum einen Anwendungsnutzen [Fraunhofer IAO, 2009 S. 70 und 78]. Zudem kann bereits die Anschaffung eines solchen Gerätes am Preis scheitern – das Microsoft Surface kostet ca. 15.000 Euro.

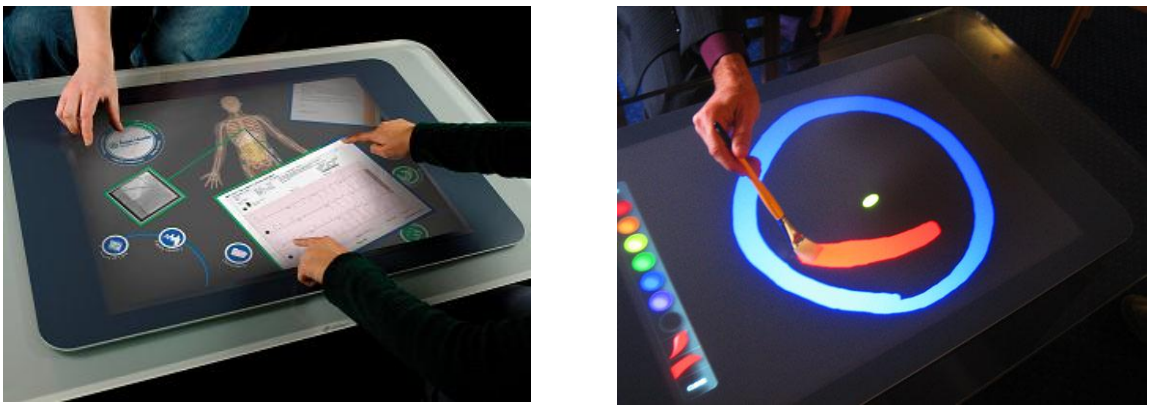


Abbildung 4 Demonstrative und spielerische Anwendungen auf dem MS Surface für 15.000 Euro. (Quelle: links [Microsoft, 2009], rechts [CNET News.com])

Organische Benutzerschnittstelle

Die Möglichkeiten der natürlichen Benutzerschnittstelle sind weiterhin beschränkt auf die virtuellen Elemente, die vom Computer auf einem Bildschirm dargestellt werden. Sie bleiben daher in den zweidimensionalen Bereichen und ermöglichen keine direkte Manipulation, wie wir es gewohnt sind, wenn wir z.B. eine Kaffeemaschine bedienen oder eine Zeitung lesen.

[Holman, et al., 2008] definiert organische Benutzerschnittstellen wie folgt:

“An Organic User Interface is a computer interface that uses a non-planar display as a primary means of output, as well as input. When flexible, OUIs have the ability to become the data on display through deformation, either via manipulation or actuation. Their fluid physics-based graphics are shaped through multitouch and bimanual gestures.”

Wenn die gesamte Maschine mit einer biegsamen Benutzeroberfläche ausgestattet wird, so wird damit auch der gesamte Computer zum Ein- und Ausgabemedium. Neue Interaktionsformen, die sonst nur mit realen Objekten möglich waren, können so auch mit einem Computer verwendet werden. Eine recht eindrucksvolle Vorstellung sind biegsame Papiercomputer, die eine Eingabe durch die Interaktionsformen Verbiegen oder Stapeln von mehreren Geräten erkennen (Abbildung 5).

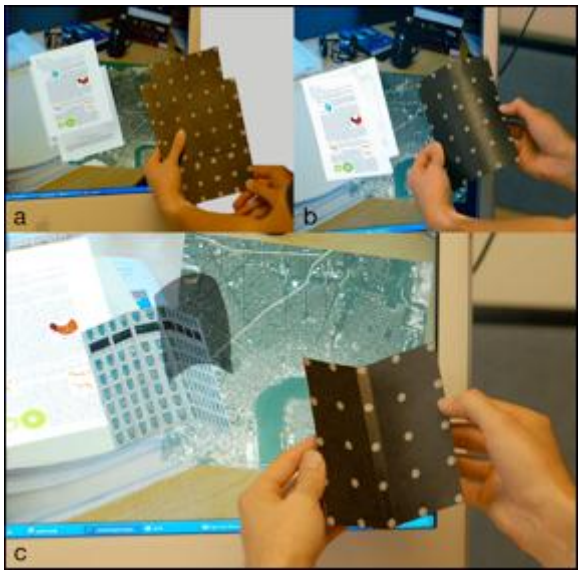


Abbildung 5 Prototypen von interaktiven Papiercomputern. Mehrere solcher Geräte lassen sich stapeln. Das Durchblättern von mehreren solcher Computer ändert ihren Inhalt analog zu einem Stapel Papier. [Holman, et al., 2008]

Die „Natürlichkeit“ der neuen organischen Benutzerschnittstellen hat mehrere Vorteile gegenüber den heutigen Maschinen wie PC, Laptop oder Mobiltelefon. So ist die Zeitung oder das Buch aus Papier immer noch sehr beliebt, da es nach [Holman, et al., 2008] auf viele verschiedene Arten mit beiden Händen gegriffen werden kann. Eine Zeitung oder Buch kann zum Beispiel schnell oder langsam durchgeblättert, auf einen Tisch gelegt und gestapelt werden. Neue Technologien wie flexible OLED-Bildschirme oder die Papiercomputer können bereits heute einen Einblick geben, wie organische Benutzerschnittstellen in Zukunft aussehen könnten. Eine ungelöste Frage besteht jedoch noch: Wie muss die Software für organische Benutzerschnittstellen gestaltet werden?

2.3 Berührungsempfindliche Interaktion

Es existiert eine Vielzahl von verschiedenen Technologien, um eine direkte Interaktion auf einem Bildschirm zuzulassen. Zudem können verschiedene Arten von Methoden der Interaktion auf einen solchen Bildschirm angewendet werden. Die folgenden Kapitel stellen daher zuerst unterschiedliche Technologien für berührungsempfindliche Bildschirme vor (Kapitel 2.3.1), um danach Interaktionsmethoden für die berührungsempfindliche Eingabe zu beschreiben (Kapitel 2.3.2).

2.3.1 Technologien

Obwohl berührungsempfindliche Bildschirme schon seit den 1960er oder den 1970er Jahren existieren (nach [Breier, 2010] bzw. [Schöning, et al., 2008]) wurden sie doch erst durch die Einführung von mobilen Geräte bei einer breiten Bevölkerungsschicht populär. Mittlerweile existieren verschiedene Technologien, die je nach Anforderung und Preisklasse eingesetzt werden können:

- Druckempfindliche Bildschirmoberflächen
- Ladungsempfindliche Bildschirmoberflächen
- Akustische Bildschirmoberflächen
- Optische Bildschirmoberflächen

Druckempfindliche Bildschirmoberflächen

Die am häufigsten verwendete Touch-Technologie nach [Breier, 2010] nutzt zwei in Abstand gehaltene, stromleitende Schichten, die sich durch Druck auf eine Stelle verbinden können. Dazu wird an den Rand einer der Schichten eine Spannung angelegt, die zur anderen Seite gleichmäßig abfällt. Wird nun Druck, z.B. durch einen Finger, auf die Bildschirmoberfläche ausgeübt verbinden sich beide Schichten an dieser Stelle. Die Schichten bilden damit einen sogenannten Spannungsteiler. An den Rändern der zweiten Schicht können so zwei verschiedene Spannungen gemessen werden, deren Verhältnis eine Dimension der Positionskoordinaten ist. Um die zweite Dimension zu erhalten wird die Messung mit vertauschten Rollen der Schichten wiederholt (siehe Abbildung 6).

Die Technik der druckempfindlichen Bildschirme erlaubt daher auch den Einsatz von beliebigen Werkzeugen wie Stiften (z.B. dem *Stylus*) oder sogar Fingernägel. Doch lassen die Schichten nur 75 Prozent der Bildschirmhelligkeit durch, so dass das Bild insgesamt dunkler wirkt und Menschen mit einer Sichtbehinderung das Lesen erschwert wird [tiresias.org, 2009]. Zudem erschwert der Umstand, dass die leitenden Schichten durch Druck sich berühren müssen, die Bedienung solcher Bildschirme. Ein zu gering ausgeübter Druck kann daher zu Eingabefehlern, wie eine nicht erkannte Eingabe, führen und den Nutzen schmälern.

1. Widerstandsfähige Beschichtung
2. Leitende Oberschicht
3. Abstandshalter
4. Resistive Beschichtung
5. Glasscheibe
6. Bildschirmfläche

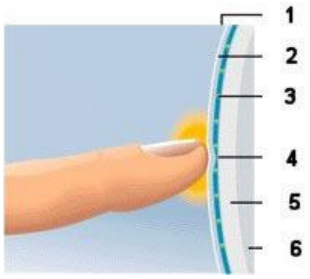


Abbildung 6 Druckempfindliche Bildschirmoberflächen bestehen aus mehreren Schichten. Durch den Druck des Fingers berühren sich die leitendenden Schichten und ermöglichen so eine Bestimmung der Position. Bild: [Tyco Electronics, 2010]

Ladungsempfindliche Bildschirmoberflächen

Im Gegensatz zu den druckempfindlichen Bildschirmoberflächen funktionieren die ladungsempfindlichen Bildschirmoberflächen ohne starken Druck auf die leitende Oberschicht. Stattdessen funktioniert der Bildschirm wie ein Kondensator, dessen Ladung durch eine Berührung mit dem Finger abfließt. Dieser Spannungsabfall wird von einem integrierten Mikroprozessor erkannt und in Positionswerte umgerechnet. Gegenüber druckempfindlichen Bildschirmen ist die ladungsempfindliche Technik resistenter gegenüber mechanischen Beschädigungen, da die Bildschirmoberfläche nicht verformbar sein muss. Daher werden die ladungsempfindlichen Bildschirme auch gerne in öffentlichen Automaten wie Fahrkartenschaltern eingesetzt [Schöning, et al., 2008]. Zudem besitzt diese Art von Bildschirmen eine höhere optische Transparenz, was sie zwar heller, jedoch auch teurer als die druckempfindlichen Bildschirme macht.

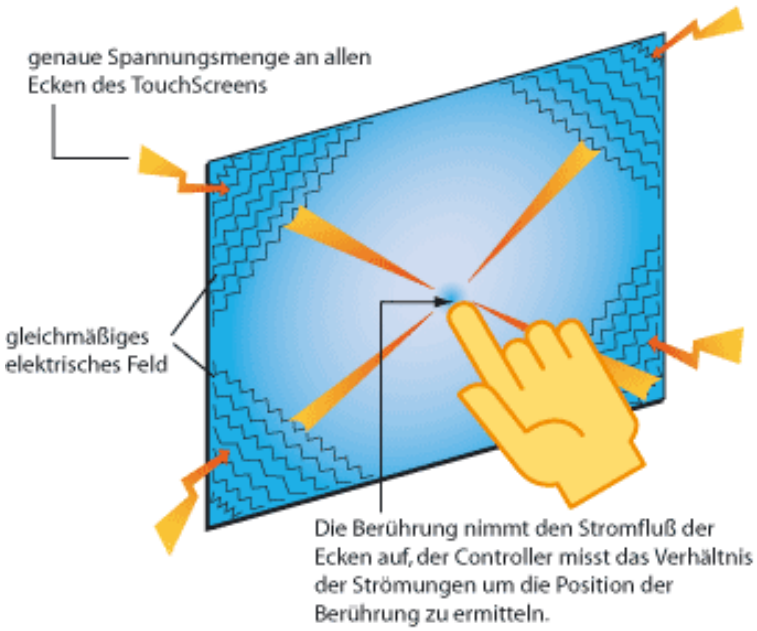


Abbildung 7 Ladungstransport bei Berührung der geladenen Bildschirmoberfläche. Der Finger lässt bei einer Berührung der Oberfläche die Spannung abfallen, so dass die Position der Berührung bestimmt werden kann. Bild: [VISAM]

Akustische Bildschirmoberflächen

Akustischen Bildschirmoberflächen nutzen Mikrofone an den Bildschirmrändern einer Glasscheibe. Durch Antippen der Glasoberfläche wird Schall erzeugt, der von den Mikrofonen erkannt wird. Die Laufzeitunterschiede zu den verschiedenen Mikrofonen ermöglichen dabei die Position der Quelle zu ermitteln.

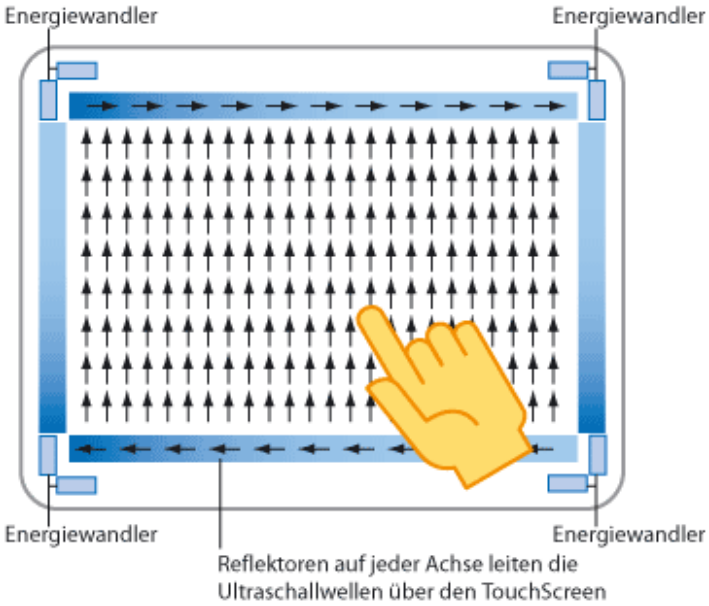


Abbildung 8 Surface Acoustic Wave verwendet ein Ultraschallwellenraster, das bei einer Berührung verändert wird. Eine Berührung der Scheibe absorbiert einen Teil der Wellen, so dass die Position der Berührung bestimmt werden kann. Bild: [VISAM]

Die Technik **Surface Acoustic Wave** (SAW) setzt Ultraschallwellen in einer Glasscheibe ein. Die Schallwellen werden von Sendern und Empfängern an den horizontalen Bildschirmrändern abgegeben und aufgenommen. Dadurch ergibt sich ein Raster (siehe Abbildung 8), welches zur Positionsbestimmung genutzt wird. Bei einer Berührung der Oberfläche wird ein Teil der ausgesendeten Wellen absorbiert und ermöglicht beim Empfänger nicht nur eine Erkennung des Antippens, sondern auch die Kraft des Antippdrucks, je nachdem wie stark die Welle gedämpft wurde [VISAM].

Die Vorteile der akustischen Technik liegen in der hohen Lichttransparenz durch den Einsatz von durchsichtigem Glas. Die Robustheit gegenüber Vandalismus wird zudem erhöht, wenn verstärktes Glas, sogenanntes „Gorilla-Glas“ eingesetzt wird. Die Bedienbarkeit ist jedoch auf die Finger mit oder ohne Handschuh beschränkt und erfordert je nach Empfindlichkeit der Mikrofone mehr ein Klopfen anstatt eines Antippens der Glasplatte. Durch diesen Umstand entfällt auch die Möglichkeit ziehenden Bewegungen auf der Oberfläche (z.B. für Drag & Drop) durchzuführen.

Optische Bildschirmoberflächen

Es existieren verschiedene Umsetzungen von optischen Bildschirmen. Alle gemeinsam nutzen Licht (meistens im Infrarotbereich) und dessen Brechung und Zerstreuung an einem fast durchsichtigen Material (z.B. Acrylglas). Liegt ein Gegenstand auf dem Glas oder berührt ein Benutzer das Glas, so ändert sich die Lichtbrechung, die dann durch eine Kamera unterhalb des Bildschirms erkannt werden kann. Die Techniken

- Diffuse Surface Illumination (DSI),
- Front Diffuse Illumination (FDI) und
- Frustrated Total Internal Reflection (FTIR) (vorgestellt in [Han, 2005])

sind Umsetzungen, die dieses Prinzip nutzen. Sie sind in Abbildung 9 dargestellt. Die darin eingesetzten Kameras erzeugen Bilder, die in Echtzeit von einem Computer ausgewertet werden und dadurch eine flüssige Bedienung der Benutzeroberfläche ermöglichen.

Vorteil dieser Techniken ist die hohe Robustheit nach [Schöning, et al., 2008] und die zusätzliche Erkennung von abgelegten Objekten an Hand ihrer Form (vgl. [Breier, 2010 S. 8]).

Eine weitere optische Technik nutzen auch Bildschirme, deren Oberfläche selbst nicht berührungsempfindlich ist. Stattdessen wird ein Raster aus Infrarotstrahlen knapp über der Bildschirmoberfläche durch Dioden erzeugt. Diese Dioden bilden mit Fotozellen als Empfänger ein Gitter, welches um den gesamten Rand des Bildschirms geht und das durch einen Finger oder Stylus unterbrochen werden kann. Die Unterbrechung von einzelnen Infrarotstrahlen ermöglicht so die Bestimmung des Berührungsortes auf der Bildschirmoberfläche.

Diese Technik ist relativ billig und zudem unabhängig von der notwendigen Leitfähigkeit des Fingers, wie bei ladungsempfindlichen Bildschirmoberflächen. Doch mit der Zeit können Schmutz und Staub, die sich an den Bildschirmrändern abgelagert haben, die Infrarotstrahlen unterbrechen und so eine Nutzung erschweren oder gar verhindern [Porteck, 2011 S. 129].

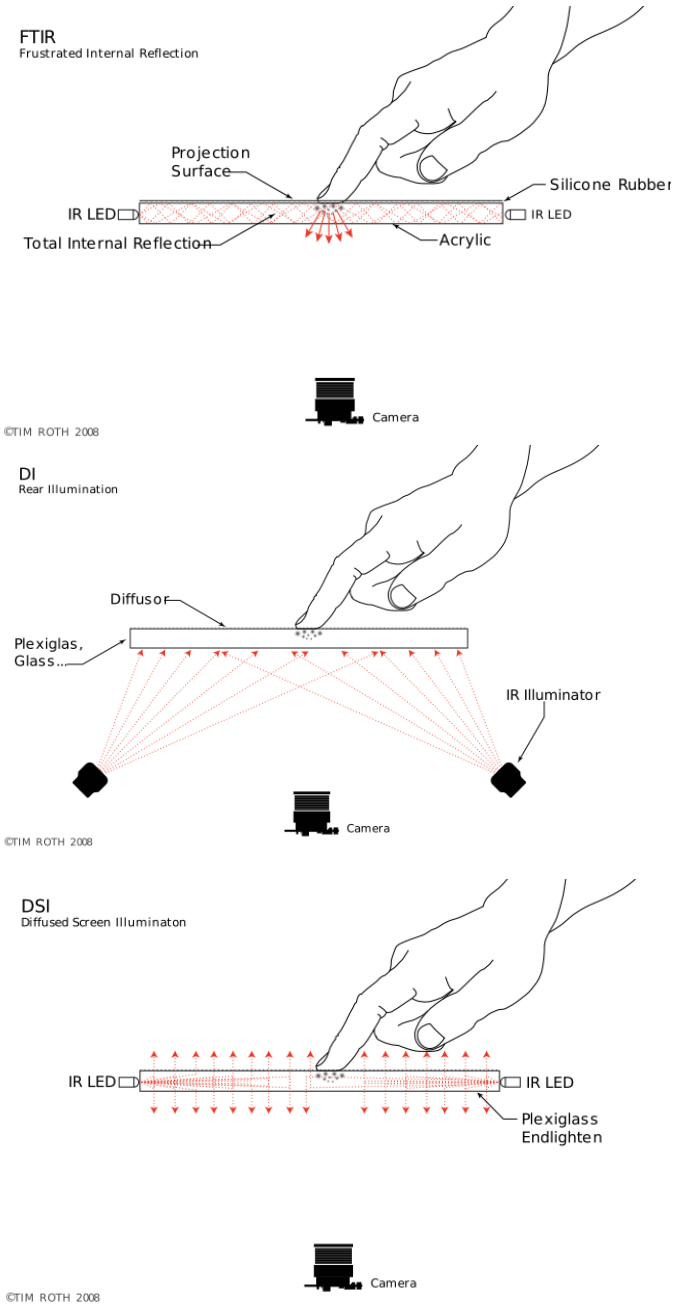


Abbildung 9 Verschiedene optische Bildschirmaufbauten (v.o.n.u. FTIR, DI, DSI). Eine Kamera nimmt unterhalb der Bildschirmoberfläche Veränderungen auf der Oberfläche durch Lichtstreuung wahr. Bilder: [Roth, 2008]

2.3.2 Interaktionsmethoden

Eine Methode ist nach [Fischer, et al., 2008] ein „[…] *Weg bzw. Art, wie man zu einem angestrebten Ziel gelangt.*“ Damit beschreiben berührungsempfindliche Interaktionsmethoden die Art und Weise, wie durch Berührung eine Aufgabe erledigt werden kann. Die Interaktionsmöglichkeiten mit einem oder mehreren Fingern sind in Touchumgebungen auf vier Arten beschränkt:

1. An-/Tippen (oder Klopfen)
2. Halten
3. Bewegen
4. Gestik

Dabei besteht die Gestik aus einer Kombination der drei ersten Interaktionsmöglichkeiten. Gesten sind daher keine unabhängige Eingabeart, sondern bestehen immer aus Antippen, Halten und Bewegen der Finger bzw.

Hand. In einer Studie von [Blascheck, et al., 2010/2011] werden Handgesten basierend auf [Pavlovic, 1997] wie folgt definiert: „*Eine Handgeste ist eine kontinuierliche, zeitliche Folge von Handposen über ein bestimmtes Zeitintervall. Dabei ist eine Handpose die Form, Position und Orientierung der Hand und der Finger zu einem bestimmten Zeitpunkt.*“ Erst der Einsatz von Gesten ermöglicht die effiziente Bedienung einer Benutzerschnittstelle. Beispielsweise werden die bereits vorgestellte *natürlichen Benutzerschnittstellen* aus Kapitel 2.2 ausschließlich über Finger bedient. Doch auch grafische Schnittstellen sind auf diese Weise zu manipulieren, wenn die Interaktionsmethoden der Fingereingabe die Methoden von Maus und Tastatur ersetzen können. So stellen [Matejka, et al., 2009] eine Lösung vor, um die Maussteuerung durch eine Fingergestik zu ersetzen. Dadurch kann jede Maustaste durch Tippen von zwei oder mehreren Fingern emuliert werden (Abbildung 10).

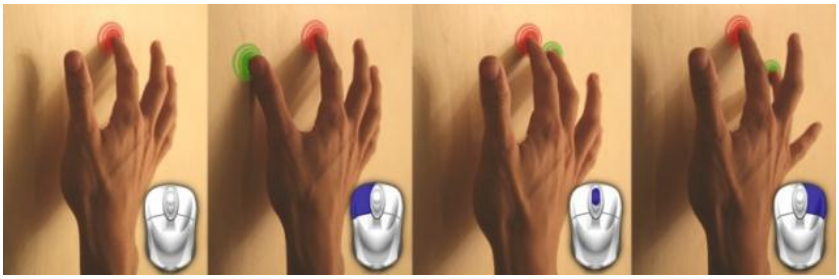


Abbildung 10 Maustasten und das Mausrad können durch Finger simuliert werden [Matejka, et al., 2009]. Die rechte Maustaste und Scrollrad werden durch eine Fingerkombination ersetzt.

Neben Fingergesten können auch andere Arten von Gesten verwendet werden, um Elementen einer Benutzeroberfläche zu steuern. In einem Dokument über *Touch* von [Microsoft] werden fünf Gesten vorgestellt, die von Anwendungen mit dem Windows Entwicklungstoolkit (Windows SDK) angewendet werden können, um berührungsempfindliche Eingaben zu unterstützen:

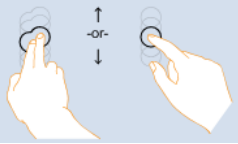
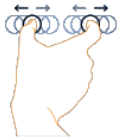
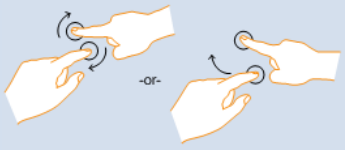
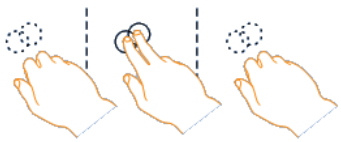
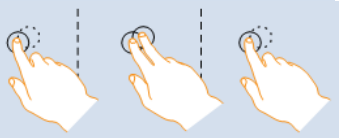
Geste	Beschreibung	Darstellung
Verschieben, Schwenken (engl. panning)	Ein Objekt oder eine Ansicht wird durch einen oder zwei Finger von einer Position zu einer anderen bewegt. Durch Loslassen wird die Geste beendet.	
Größenänderung (engl. zoom)	Ein Objekt oder eine Ansicht wird durch zwei Finger, die sich gleichzeitig auf dem Bildschirm befinden, vergrößert oder verkleinert, indem die Finger sich voneinander weg- oder aufeinander zubewegen.	
Rotieren (engl. rotate)	Ein Objekt oder eine Ansicht wird durch zwei Finger, die sich gleichzeitig auf dem Bildschirm befinden, um einen Finger rotiert, indem der zweite Finger im Kreis um den ersten bewegt wird.	
Antippen mit zwei Fingern (engl. two-finger tap)	Diese Geste soll den voreingestellten Zustand der Größe oder Anzeige eines Objekts oder Ansicht wiederherstellen. Die Geste wird ausgeführt, indem zwei Finger gleichzeitig die Oberfläche antippen.	
Halten und Tippen (engl. press and tap)	Diese Geste simuliert einen Rechtsklick mit der Maus am Ort des ersten Fingers. Die Geste beginnt mit dem Antippen und Halten eines Fingers auf dem Objekt, gefolgt von einem zweiten Finger, der die Geste durch ein einmaliges Klopfen beendet und den Rechtsklick damit auslöst.	

Tabelle 1 Gesten mit einem oder mehreren Fingern lösen vielfältige Aktionen aus.

2.4 Dialoge

Dialoge stellen ein zentrales Thema dieser Arbeit dar. Daher ist es notwendig, zuerst einmal zu wissen was Dialoge sind und wie sie beschrieben werden können. Die folgenden Kapitel führen in die Grundlagen der Dialoge ein.

Zuerst wird in Kapitel 2.4.1 die grundlegendste Aufgabe von Dialogen erläutert und welche weiteren Formen von Dialogen existieren. In den darauf folgenden Kapiteln wird erklärt, wie Dialoge grundsätzlich gestaltet werden sollten, wie Benutzer mit Dialogen interagieren können und wie diese Interaktionsformen genannt werden. Für die Entwicklung von Dialogen ist ein Modell oder eine Sprache erforderlich, die den Dialog für Computer beschreiben. Diese Arten von Dialogbeschreibungssprachen werden in Kapitel 2.4.4 erläutert. Abgeschlossen wird das Kapitel über Dialoge mit einer Demonstration über den Einfluss der Punktdichte von Bildschirmen auf Dialoge (Kapitel 2.4.5).

2.4.1 Einführung

Dialogfenster dienen dem Benutzer zur Eingabe und Abfrage von Daten sowie zur Bestätigung derselben. Dialoge beinhalten dazu Steuerelemente (Textfelder, Schiebregler usw.), die es dem Benutzer ermöglichen die angezeigten Werte zu ändern, zu bestätigen („OK“) oder auf voreingestellte Werte zurückzusetzen („Abbrechen“).

Dialogfenster können modal oder nichtmodal betrieben werden. Modal bedeutet, dass die Fortsetzung des Programmablaufs mit dem Schließen des Dialogs verknüpft ist. So können Falscheingaben unterbunden und Folgefehler verhindert werden. Nichtmodale Dialoge können vom Benutzer auch während das Hauptprogramm läuft angesprochen werden (natürlich auch andersherum) und zeigen zum Beispiel Detailinformationen für im Hauptprogramm ausgewählte Objekte an.

Betriebssystem APIs (z.B. *MessageBox* mit dem Windows SDK) und Frameworks (z.B. *MessageDlg* in Delphi) bieten unterschiedliche Arten von vordefinierten Dialogen an. So erlauben Nachrichtendialoge kurze Textausgaben zum aktuellen Status der Anwendung (d.h. Fehler-, Warn- oder Informationsmeldung). Meistens kann der Benutzer den Dialog nur durch einen Klick auf einen Bestätigungsschaltknopf („OK“) beenden. Es gibt aber auch Dialoge, die dem Benutzer durch weitere Druckschalter („Ja“, „Nein“, „Ignorieren“) eine gewisse Einflussnahme auf den weiteren Programmablauf bieten.

Weitere Dialoge werden für das Öffnen und Speichern von Dateien sowie zur Auswahl von Druckern angeboten. Diese sogenannten Standarddialoge einer Benutzeroberfläche ermöglichen für alle Anwendungen eines Betriebssystems ein einheitliches Erscheinungsbild und eine gleichartige Benutzbarkeit. Zudem haben Standarddialoge den Vorteil, dass Entwickler sich besser auf die eigentliche Programmentwicklung konzentrieren können, ohne eigene Dialoge für diese Art von Standardfunktionen erstellen zu müssen.

Eine neue Art von Standarddialogen, der sogenannte Aufgabendialog (engl. task dialog, siehe Abbildung 11 rechtes Bild) wurde mit Windows Vista eingeführt [Microsoft, 2011]. Sie sind weit aus flexibler als die Nachrichtendialoge, denn sie enthalten neben Steuerelementen auch große und selbsterklärende Befehlsschalter (engl. command button). Die Aufgabendialoge können vom Anwendungsentwickler sehr flexibel gestaltet werden. Trotzdem bleiben ihr Aufbau und ihre Bedienung auch über Anwendungsgrenzen hinweg gleich, weil alle Aufgabendialoge aus einer Vorlage stammen.

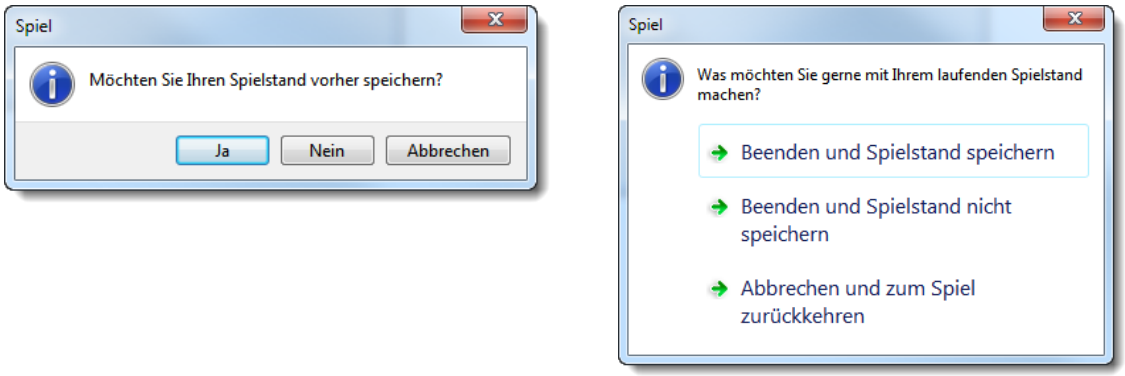


Abbildung 11 Links eine Standardabfragedialog mit den bekannten Schaltknöpfen. Der rechte Aufgabendialog setzt die Abfrage des linken Dialogs mit großflächigen und selbsterklärenden Befehlsschaltern um.

2.4.2 Gestaltungsgrundsätze

Die Norm DIN EN ISO 9241-10 definiert sieben Grundsätze für die Gestaltung von Dialogen. Sie wurden unabhängig von der Art des Dialogs oder des verwendeten Interaktionsstils entworfen. Die Grundsätze sind Aufgabenangemessenheit, Selbstbeschreibungsfähigkeit, Steuerbarkeit, Erwartungskonformität, Fehlertoleranz, Individualisierbarkeit und Lernförderlichkeit. Im Folgenden sollen diese kurz vorgestellt werden. Für eine ausführliche Behandlung können u.a. die folgenden Quellen genutzt werden, auf die sich auch die Übersicht stützt: [Heinecke, 2004 S. 168ff.], [Wessel, 2002 S. 38ff.] oder [Köth, 2001].

Aufgabenangemessenheit

Die Norm ISO 9241-10 definiert die Aufgabenangemessenheit von Dialogen wie folgt:
„*Ein Dialog ist aufgabenangemessen, wenn er den Benutzer unterstützt, seine Arbeitsaufgabe effektiv und effizient zu erledigen*."
Dialoge sollten nach dieser Empfehlung nur die notwendigen Informationen enthalten, die für den Benutzer zum aktuellen Zeitpunkt für die Aufgabe nützlich sind (z.B. keine Hardwareinformationen bei einem Kundenformular). Aufgaben, die automatisch ausgeführt werden können, dürfen den Benutzer nicht mehr belasten („Der Ordner Z existiert nicht. Soll er angelegt werden?"). Zudem hat das Dialogsystem den Benutzer bei wiederkehrenden Aufgaben zu unterstützen, indem z.B. Eingabemasken bereits mit voreingestellten Werten ausgefüllt sind.

Selbstbeschreibungsfähigkeit

„Ein Dialog ist selbstbeschreibungsfähig, wenn jeder einzelne Dialogschritt durch Rückmeldung des Dialogsystems unmittelbar verständlich ist oder dem Benutzer auf Anfrage erklärt wird."
Dialogsysteme sollten die Benutzer durch Rückmeldungen über die Konsequenzen ihrer Handlung informieren, beispielsweise wenn dadurch Daten unwiederbringlich verloren gehen könnten („Möchten Sie alle Daten aus dem Formular durch voreingestellte ersetzen?"). Über Fehler bei der Eingabe von Daten sollte nicht nur der Benutzer informiert werden, sondern es sollte auch der Ursprung (z.B. ein bestimmtes Eingabefeld) und ein Korrekturvorschlag angegeben werden (z.B. für Datum TT.MM.JJJJ). Zudem sollte die Rückmeldung entsprechend den Kenntnissen des Benutzers angepasst sein, indem z.B. technische Informationen (Speicheradresse, Ausnahmebezeichner) für Experten nur auf Anfrage (ein Schalter „Mehr technische Informationen") ausgegeben werden. Alle anderen Benutzer sollten dagegen nicht nur über den Fehler, sondern auch über eine mögli-

che Lösung aufgeklärt werden („Der Suchbegriff ist zu kurz. Bitte geben Sie einen Suchbegriff mit mindestens 3 Zeichen ein und wiederholen Sie die Suche anschließend.“).

Steuerbarkeit

„Ein Dialog ist steuerbar, wenn der Benutzer in der Lage ist, den Dialogablauf zu starten sowie seine Richtung und Geschwindigkeit zu beeinflussen, bis das Ziel erreicht ist.“
Die Geschwindigkeit, in der ein oder mehrere Dialoge bearbeitet werden, sollte immer vom Benutzer abhängig sein. D.h. ein Dialogfeld verschwindet erst, nachdem der Benutzer es abgeschlossen hat. Zur Steuerbarkeit zählt auch dem Benutzer die Möglichkeit zu geben, wie ein Dialog fortgesetzt (z.B. mit Eingabe- oder Tabulatortaste) oder wie er jederzeit abgebrochen werden kann (z.B. mit Esc Taste). Eine weitere Empfehlung sieht zudem vor bei mehreren Eingabegeräten (u.a. Maus, Stift oder Finger) dem Benutzer die Wahl zu lassen, welche Eingabeart verwendet wird.

Erwartungskonformität

„Ein Dialog ist erwartungskonform, wenn er konsistent ist und den Merkmalen des Benutzers entspricht, z. B. seinen Kenntnissen aus dem Arbeitsgebiet, seiner Ausbildung und seiner Erfahrung sowie den allgemein anerkannten Konventionen.“
Eine gleichbleibende Bedienung eines Dialogsystems ist notwendig, um dem Benutzer die Arbeit so effizient wie möglich erledigen zu lassen, ohne dass er sich auf die Bedienung konzentrieren muss. Dazu zählen, dass Dialoge bei ähnlichen Aufgaben entsprechend ähnlich aussehen und sich gleich bedienen lassen (z.B. zwei ähnliche Dialoge für Kunden- und Mitarbeiterdatenerfassung). Die Bedienung des Dialogs sollte konsistent bleiben. Beispielsweise möchte der Benutzer eine Hilfe mit F1 erhalten oder zwischen Steuerelementen mit der Tabulator Taste springen. Benötigt das System für die Bearbeitung einer Aufgabe außerdem eine längere Zeitspanne, dann sollte während dieser Dauer ein Fortschrittsdialog angezeigt werden, welcher dem Benutzer den aktuellen Arbeitsverlauf mitteilt.

Fehlertoleranz

„Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand seitens des Benutzers erreicht werden kann.“
Die Eingaben des Benutzers sollten vom Dialogsystem geprüft und Fehler sowie eine Beschreibung dem Benutzer zurückgemeldet werden. Es ist jedoch genauso möglich Fehler automatisch korrigieren zu lassen. In diesem Fall sollte der Benutzer über die Korrektur informiert und eine Gelegenheit geboten werden, den Korrekturvorschlag zu überschreiben.

Individualisierbarkeit

„Ein Dialog ist individualisierbar, wenn das Dialogsystem Anpassungen an die Erfordernisse der Arbeitsaufgabe sowie an die individuellen Fähigkeiten und Vorlieben des Benutzers zulässt.“
Kann ein Dialog individualisiert werden, bedeutet dies für den Benutzer den Dialog nach seinen Vorlieben und Vorstellungen anzupassen. Dies kann z.B. die Sprache der Texte sein oder wie der Dialog und dessen Steuerelemente dargestellt werden (z.B. Layout, Größe und Position). Während Einsteiger durch zusätzliche Informationen angeleitet werden können, wollen Experten möglichst effizient durch den Dialog gelangen, z.B. indem sie Abkürzungen (Shortcuts) verwenden.

Lernförderlichkeit

„Ein Dialog ist lernförderlich, wenn er den Benutzer beim Erlernen des Dialogsystems unterstützt und anleitet.“ Dialoge sollten den Benutzer unterstützen, sein Wissen selbständig zu erweitern. Dies kann z.B. durch ein umfangreiches Hilfesystem oder durch kurze Hilfestellungen (engl. tooltips) zum jeweiligen Problembereich (z.B. fokussiertes Steuerelement) erreicht werden. Zudem kann ein Benutzer auch beim Ausprobieren („Learning by doing“) unterstützt werden, indem jeder Schritt rückgängig gemacht werden kann.

2.4.3 Taxonomie der Benutzerinteraktionen

Neben sensorischen Eingabearten (z.B. Gesichts-, Sprach- und Berührungserkennung) werden bei grafischen Dialogsystemen größtenteils einfache Benutzerinteraktionen eingesetzt (siehe Abbildung 12). [Paulenz, 2010] beschreibt diese grundlegenden Interaktionen basierend auf [Barclay, et al., 1999] und [Meixner, et al., 2008] und teilt sie in fünf Kategorien ein:

Ausführung

Die Ausführung von Funktionen einer Anwendung ist die grundlegendste Interaktion zwischen einem Benutzer und einem System. Die Ausführung wird durch ein Kommando, das als Schaltknopf (siehe Tabelle 17 Steuerelemente: Bezeichnung, Symbol und Kurzbeschreibung im Anhang auf Seite 134) oder Menüeintrag zur Verfügung steht, ausgelöst und ermöglicht verschiedene Aktionen wie das Speichern eines Dokuments, das Wiederherstellen eines voreingestellten Wertes sowie das Bestätigen oder Abbrechen eines Dialogs.

Ausgabe

Die Ausgabe gehört strenggenommen auch zu den Benutzerinteraktionen. Mit der Ausgabe werden Rückmeldungen (engl. feedback) und die aktuellen Daten der Anwendung bezeichnet, die auf eine Benutzereingabe folgen. Beispielsweise werden die zuletzt in einem Dialog eingegebenen Werte durch das Klicken eines Schaltknopfes geprüft und bei möglichen aufgetretenen Fehlern eine Meldung ausgegeben.

Eingabe

Die Eingabe wird bestimmt durch die Übermittlung von neuen Werten an ein System. Bei der Eingabe sind keine Werte vorgegeben, sondern müssen vollständig neu vom Benutzer eingegeben werden.

Bearbeitung

Die Bearbeitung ermöglicht die Änderung oder das Anpassen von im System bereits vorhandenen Werten. Statt leere Eingabefelder für Texte oder Zahlen dem Benutzer zu präsentieren, sind die Eingabefelder bereits mit durch das System voreingestellten Werten gefüllt. Diese Werte können vom Hersteller vordefiniert worden sein oder aus vorangegangenen Eingaben stammen. Die Bearbeitung ist daher ein Spezialfall der Eingabe und vermeidet zusätzlichen Arbeitsaufwand für den Benutzer.

Auswahl

Mit der Auswahl kann der Benutzer ein oder mehrere Elemente aus einer Menge von Elementen auswählen. Es können drei Klassen unterschieden werden:

1. Die Einzelauswahl von Werten, wie es bei einem Dropdown-Listenfeld möglich ist.
2. Die Mehrfachauswahl von verschiedenen Werten, wie es bei Listen (z.B. Dateien im Windows Explorer) möglich ist.
3. Die Bereichsauswahl von Werten, d.h. auch die Auswahl von Werten, die innerhalb eines bestimmten Bereichs liegen.

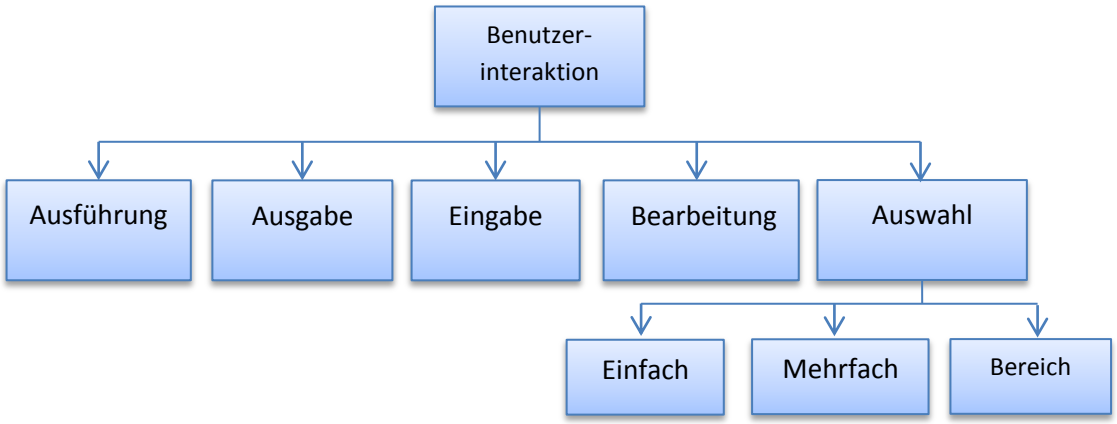


Abbildung 12 Benutzerinteraktionen in Dialogen können in verschiedene Kategorien aufgeteilt werden

2.4.4 Dialogbeschreibungssprachen

Dialoge sind Benutzungsschnittstellen, die durch ein Modell oder eine Dialogbeschreibungssprache spezifiziert werden können. Die eigentliche Darstellung für den Benutzer erfolgt durch eine von dem eingesetzten Framework abhängige Dialogbeschreibungssprache. Aus der modellgetriebenen Entwicklung (Model Driven Engineering) sind eine Menge von sogenannten Dialoggeneratoren bekannt (vgl. [Schlegel, et al., 2010]), die aus abstrakten Modellen interaktive Dialoge erstellen. Die Dialogbeschreibungssprachen können in abstrakte und konkrete Sprachen unterteilt werden, die im Folgenden vorgestellt werden.

Abstrakte Benutzungsschnittstellen

Abstrakte Benutzungsschnittstellen (AUI, engl. Abstract User Interface) kennzeichnen sich durch ihre Unabhängigkeit von einem bestimmten Framework aus. Ihre Inhalte werden durch Modelle definiert, die die eigentlichen Daten sowie ihren Nutzen (z.B. Uhrzeit, Datum oder Währung) beschreiben. Diese Beschreibungen enthalten daher keine Details, die von Frameworks gefordert werden, um die Darstellung auf dem Bildschirm oder anderen Ausgabegeräten zu erreichen. Als Beispiel für ein abstraktes Steuerelement liefert [Paternò, 2005] :

"*[…] at a given point there is a need for a selection object without indicating whether the selection is performed graphically or vocally or through a gesture or some other modality.*"

AUIs wurden durch die Entwicklung von modellbasierten Ansätzen bekannt. Die Ansätze nutzen Modelle, um Dialoge unabhängig von der einzusetzenden Plattform zu beschreiben. Nach [Paternò, et al., 2009] war eine der ersten Entwicklungen das UIDE (User Interface Design Environment) von [Foley, et al., 1995], welches Dia-

logoberflächen durch Schemata aus Objekten und Aktionen erstellte. Dieses Projekt wird jedoch nicht mehr weiterverfolgt.

Die nachfolgenden Ansätze verließen schließlich den objektorientierten Ansatz von [Foley, et al., 1995] und arbeiten stattdessen mit einer semantischen Beschreibung der Interaktion. Dazu werden Aufgabenmodelle (engl. task model) definiert, um Aktionen ausdrücken zu können, die der Benutzer in der Oberfläche ausführen kann. Ein Vertreter dieser Lösung ist ConcurTaskTrees (CTT) von [Paternò, et al., 1997] (siehe auch Kapitel 2.5.2 Reverse Engineering und [W3C, 2009]). CTT ist eine grafische Notation, die eine hierarchische Struktur besitzt und Aktivitäten einsetzt, um einen Aktion zu beschreiben. Eine zusätzliche Eigenschaft der CTT bildet die Unterstützung von Fehlerbehandlungen, die durch sogenante zeitliche Operatoren umgesetzt werden. Paternò et al. argumentieren, dass CTT daher flexibler als andere Modelle ist, dennoch aber leicht verständlich bleibt.

Aktuell werden gerade auch wegen neuer Geräteplattformen im Mobilbereich verstärkt Modellierungssprachen eingesetzt, die an die Geräte und ihren unterschiedlichen Merkmalen (Leistung, Auflösung, Eingabeart usw.) angepasste Dialogformen erlauben. Diese Sprachen werden deshalb nicht mehr als „abstrakt“ bezeichnet, sondern beschreiben konkrete Benutzungsschnittstellen. Sie werden im nächsten Abschnitt beschrieben.

Konkrete & finale Benutzungsschnittstellen

Konkrete Benutzungsschnittstellen hängen nach [Paternò, 2005] vom Typ der Plattform und vom Ausgabegerät ab. Sie besitzen eine Vielzahl von Eigenschaften, die das Aussehen und Verhalten von Dialogen beeinflussen können, jedoch immer von der Zielplattform abhängig sind. Weiterhin definiert [Paternò, 2005] die finale Benutzungsschnittstelle, die nicht nur von der eingesetzten Plattform abhängig ist, sondern auch von der verwendeten Softwareumgebung wie beispielsweise C# und XAML.

Im Folgenden werden zwei Arten, wie Dialoge für eine Plattform beschrieben werden können, vorgestellt.

1. Ein Dialog wird durch eine Programmiersprache (Java, C++) realisiert.
2. Ein Dialog wird in einer externen, vom Quelltext unabhängigen Form beschrieben : XAML

Programmiersprachen

Die offensichtlichste Art und Weise, um Oberflächen zu erstellen, sind die Klassen und Methoden eines Frameworks direkt zu nutzen, indem Instanzen erstellt und Attribute gesetzt werden. Ohne einen Oberflächendesigner ist dies meist auch die einzige Möglichkeit Oberflächen zu gestalten. Dabei vermischt sich der Quelltext für die Erstellung der Oberfläche mit anderen Quelltextbestandteilen wie der Programmlogik und dem Ereignismanagement.

Es ist nicht verwunderlich, dass Frameworks zuerst keine grafischen Dialogeditoren besaßen. Sie wurden erst später oftmals durch Dritthersteller geliefert. So wurde Borlands Turbo Pascal 6.0 (1990) mit einer objektorientierten GUI mit dem Namen Turbo Vision ausgestattet. Die Oberfläche wurde dabei ausschließlich mit Objekten im Quelltext erzeugt (Abbildung 13). Erst nachträglich gab es von Hobbyprogrammierer erstellte Dialogdesigner, die den Quelltext generierten.

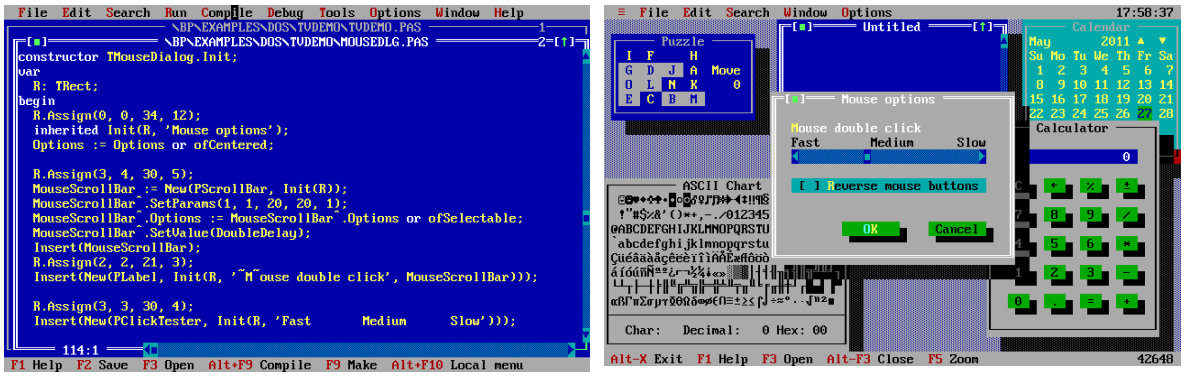


Abbildung 13 Der Quelltext eines mit Turbo Vision erstellten Dialogs und die Darstellung des Dialogs in einer Turbo Vision Anwendung für MS DOS.

Ein weiteres Beispiel ist Netbeans, eine von Oracle [Oracle]) entwickelte Entwicklungsumgebung. Sie bietet seit langem einen Dialogdesigner für das Framework Swing, um Dialoge grafisch mit der Maus bearbeiten zu können.

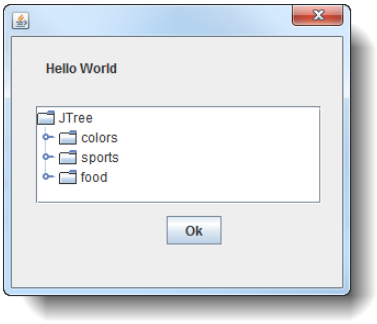


Abbildung 14 Ein Dialog in Java mit dem Framework Swing erstellt

Abbildung 14 zeigt einen mit Java Swing in Netbeans erstellten Dialog. Bei Änderungen wird der Quelltext 1 automatisch erstellt und die manuellen Änderungen des Entwicklers werden verworfen. Die Elemente werden als Instanzen in der Klasse abgelegt, um auch später noch darauf einen Zugriff zu ermöglichen. Das bedeutet, der Entwickler kann jederzeit alle Attribute der Steuerelemente ändern oder sie sogar löschen. Das Layout der Elemente wird durch Panel-Elemente vorgenommen, die durch ihre Attribute das Aussehen und Ausrichtung der in ihnen enthaltenen Kindelemente bestimmen.

```
/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    jButton1 = new javax.swing.JButton();
    jLabel1 = new javax.swing.JLabel();
    jScrollPane1 = new javax.swing.JScrollPane();
    jTree1 = new javax.swing.JTree();

    setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);

    jButton1.setText("Ok");

    jLabel1.setText("Hello World");

    jScrollPane1.setViewportView(jTree1);
```

```
        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(31, 31, 31)
                        .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 213,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(layout.createSequentialGroup()
                        .addGap(141, 141, 141)
                        .addComponent(jButton1)))
                .addContainerGap(57, Short.MAX_VALUE))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addContainerGap(22, Short.MAX_VALUE)
                .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 260,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(19, 19, 19))
        );
 […]  // snip
        pack();
    }// </editor-fold>
```

Quelltext 1 Dialoge können aus komplizierten Strukturen bestehen. Hier der Quelltext für den Dialog aus Abbildung 14.

Um Anwendungen mit Oberflächen, die vollständig oder zum größten Teil mit Quelltext erstellt wurden auch wirklich für die Fingerbedienung nutzbar zu machen, müssen große Teile des Quelltexts gelesen und verstanden werden. Es ist wohl zweifelhaft, ob dies überhaupt lohnenswert ist. Stattdessen würde man einfach entscheiden, eine solche Anwendung nicht für die Fingerbedienung zu überarbeiten.

Denn letztendlich ist es nur sehr aufwändig aus einem bestehenden Quelltext Informationen über den Dialog zu extrahieren. Auch wenn Dialoge teilweise als Ressource und Quelltext vorliegen macht dies die Sache nicht einfacher. Denn die Verknüpfung zwischen Dialog und Quelltext maschinell zu erkennen erfordert einen zusätzlichen Entwicklungsaufwand für einen Interpreter. Womöglich wäre es in solch einem Fall einfacher die Oberfläche mit einem moderneren Framework neu zu gestalten, als den Quelltext mit einem eigenen Parser zu lesen und versuchen zu interpretieren. Weitere Nachteile, die Dialoge in Quelltextform mit sich bringen, sind in [Draheim, et al., 2006] beschrieben.

XAML

Die E**x**tensible **A**pplication **M**arkup **L**anguage (XAML, ausgesprochen „xemml“ (siehe [Doberenz, et al., 2008])) ist eine Oberflächenbeschreibungssprache in XML Syntax. Sie wurde zuerst von Microsoft für das WPF-Framework entwickelt, wird mittlerweile jedoch auch in der Windows Workflow Foundation (siehe [WF]) verwendet. Mit XAML wird eine strikte Trennung von Layout und Logik erreicht, so dass Entwickler und Designer unabhängig voneinander an einer Anwendung arbeiten können. Während der Entwickler sich um die Logik unterhalb der Oberfläche kümmert, kann der Designer das Aussehen mit eigenen Grafiken und Themen (Aussehen der Elemente) gestalten.

Ein einfaches Beispiel, für einen in XAML beschriebenen Dialog, zeigt der Quelltext 3. Die Beschreibung beginnt mit dem Fenster-Element, welches XML-Attribute (in XAML Eigenschaften genannt) für das Fenster selbst (unter anderem Fenstertitel und -größe) und Namensräume für externe Eigenschaften (z.B. Elementname „*x:Name*“) aus XAML enthält. Darin sind weitere Elemente, auch verschachtelt, enthalten.

Laut XAML Definition kann nur ein Element unterhalb des Fenster-Elements platziert werden. Daher muss ein Container-Element, hier das Element Grid, verwendet werden. Innerhalb des Grid-Elements werden dann die Steuerelemente automatisch angeordnet und ausgerichtet. Um die Abstände zwischen den einzelnen Elemen-

ten zu ändern, wird die Eigenschaft Margin (Links, Oben, Rechts, Unten) eingesetzt. Ihre Längeneinheit ist dabei standardmäßig geräteunabhängig. Trotzdem wird sie Pixel genannt, obwohl dies nichts mit den Bildschirmpixeln zu tun hat, sondern nur abhängig von der aktuellen PPI-Einstellung (siehe Kapitel 2.4.5) des Systems ist. Ein Pixel entspricht dabei standardmäßig dem 96-sten Teil eines Zolls (ca. 0,3 Millimeter). Somit sind Dialoge in XAML nicht abhängig von einer eingestellten Punktdichte (siehe Kapitel 2.4.5), sondern lassen sich unabhängig von der Bildschirmeinstellung beschreiben.

Jedes Steuerelement ist in XAML eine Objektinstanz, die zur Laufzeit über den optionalen Namen der Eigenschaft *x:Name* angesprochen werden kann. Die Elemente können auch absolut platziert werden, indem die Eigenschaften Top, Left, Width und Height gesetzt werden. Dies ist jedoch nur über die Eigenschaften eines Canvas-Elements möglich, das eine absolute Platzierung erlaubt. Die Eigenschaften werden dabei durch Anhängen an das Element gesetzt, denn das Element unterstützt diese standardmäßig nicht. In XAML werden diese externen Eigenschaften auch „attached properties“ (angehängte Eigenschaften) genannt. Im Quelltext 2 wird beispielhaft ein Schalterelement innerhalb eines Canvas-Elements mit den angehängten Eigenschaften Left und Top positioniert.

```
<Canvas Height="73" Name="canvas1" Width="227">
   <Button Canvas.Left="150" Canvas.Top="19" Content="Button" Height="34" Width="62" />
</Canvas>
```

Quelltext 2 Angehängte Eigenschaften Canvas.Left und Canvas.Top bei einem Druckschalter

```
<Window
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   x:Class="WpfApplication1.MainWindow"
   x:Name="Window"
   Title="MainWindow"
   Width="374" Height="370"mc:Ignorable="d" ResizeMode="NoResize">

   <Grid x:Name="LayoutRoot">
      <Border Margin="8,8,8,55" BorderBrush="Black" BorderThickness="1">
         <StackPanel Margin="-1,-1,-1,7" Name="Stack">
            <GroupBox Margin="14,0,12,0" Height="133" Header="" >
                     <TextBlock Text="Hello World"
                                       TextWrapping="Wrap" HorizontalAlignment="Center"
                                       VerticalAlignment="Center"/>
            </GroupBox>
            <Button HorizontalAlignment="Right" Margin="0,0,134,19" VerticalAlignment="Bottom"
                           Width="118" Height="23" Content="Ok"/>
            <Button HorizontalAlignment="Right" Margin="0,0,8,19" VerticalAlignment="Bottom"
                            Width="122" Height="23" Content="Abbrechen"/>
   </Grid>
</Window>
```

Quelltext 3 Ein in XAML beschriebener Dialog beginnt immer mit dem Fenster-Element und enthält alle weiteren Elemente in einer Baumstruktur

Es existieren zwei komfortable Oberflächeneditoren für XAML: Der Dialogeditor in Visual Studio Designer 2008 und 2010 (Abbildung 15) sowie Expression Blend (Abbildung 16). Während ersterer für Entwickler gedacht ist, die auch mit dem Quelltext arbeiten, kann mit Expression Blend das Aussehen der Schnittstelle vollständig geändert werden. Dazu unterstützt es die Grafikbearbeitungsprogramme Photoshop und Illustrator von Adobe. Ein einfaches aber nützliches Werkzeug stellt XMLPad (Abbildung 17) dar. Das Programm ermöglicht wie Visual Studio und Blend eine sofortige Vorschau während die XAML-Syntax in einem Editor geändert wird. Im Gegensatz zu Visual Studio und Expression Blend ist es kostenlos über das Windows SDK von [Microsoft] erhältlich und ermöglicht ein schnelles Ausprobieren von Designänderungen.

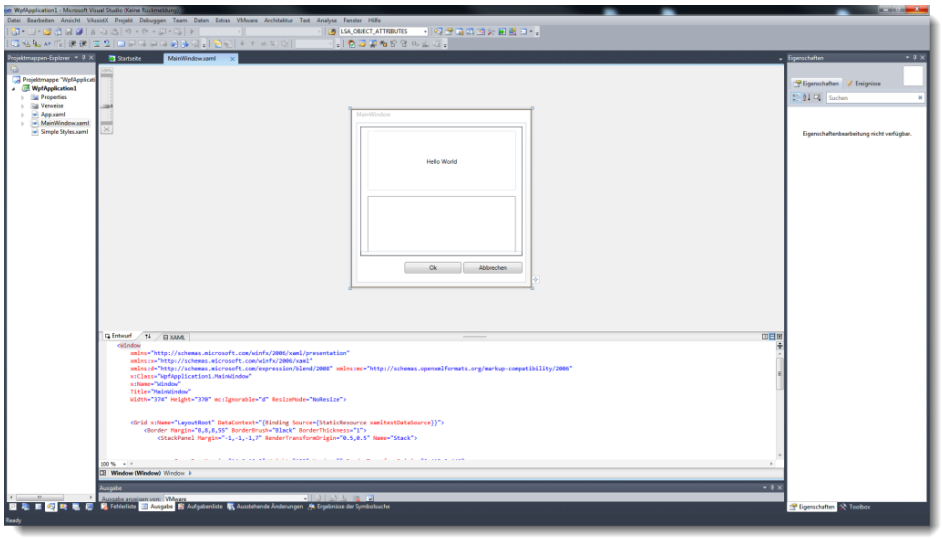


Abbildung 15 Mit Microsoft Visual Studio 2010 können Dialoge sowohl als XAML Quelltext als auch in einem Designer erstellt und bearbeitet werden. Im Gegensatz zu Expression Blend kann der Dialog mit Ereignissen ausgestattet werden, die in einer .NET Sprache geschrieben wurden.

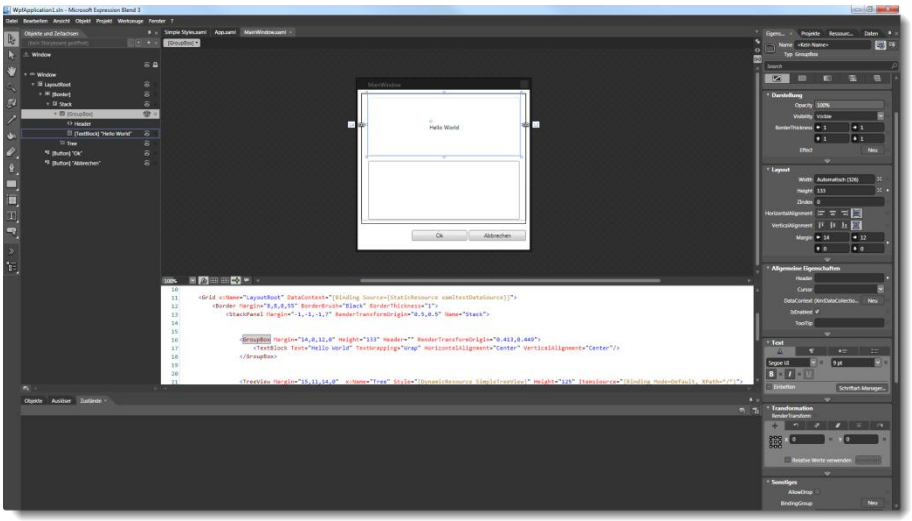


Abbildung 16 Für Designer und Nicht-Programmierer hat Microsoft den Expression Blend XAML Editor entwickelt. Er lässt sich ähnlich bedienen wie bekannte Bildbearbeitungsprogramme.

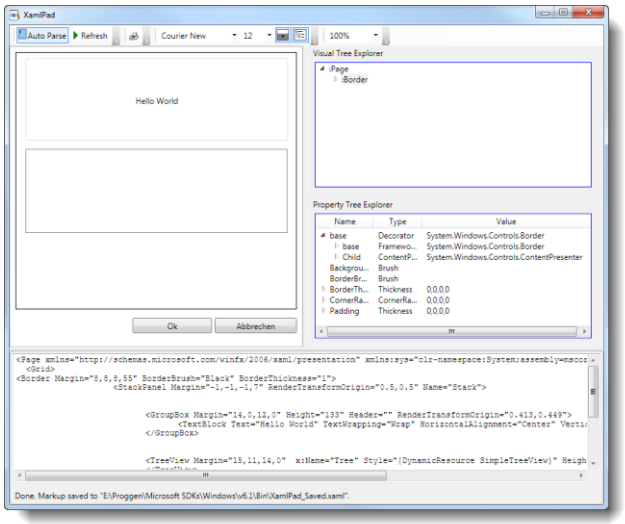


Abbildung 17 XMLPad - Ein MS SDK Werkzeug zum schnellen Ausprobieren von XAML-Syntax.

Die strikte Trennung von Logik und Oberfläche ermöglicht Entwicklern und Designern sich nur in ihrem jeweiligen Aufgabengebiet zu bewegen. Der Entwickler benötigt keinerlei Kenntnisse in der Erstellung und Gestaltung von Schnittstellen mit hoher Benutzbarkeit, kann aber erst einmal mit einer einfachen Oberfläche die Programmlogik testen. Auf der anderen Seite benötigt der Designer kaum Programmierkenntnisse und mit Expression Blend auch keine funktionierende Programmlogik. Der Dialog kann mit seinen Inhalten ohne Quelltext über Datenanbindungen (Dateien oder eine Datenbank) sofort angezeigt werden. Zusätzlich ermöglichen Stilvorlagen ein neues Aussehen aller Steuerelemente, ohne dass diese manuell angepasst werden müssten.

Die XML Struktur von XAML bietet einen guten Angriffspunkt für XML Parser über XPath oder XQuery und ist damit einfacher zu handhaben als beispielsweise ein selbstgeschriebener Parser. Mit C# findet XAML mittlerweile in Microsoft Produkten Anwendung. Dazu zählen u.a. Visual Studio 2010, Expression Blend und Microsoft Touch Pack für Windows 7.

Die Möglichkeiten der Dialogbeschreibungssprache XAML sind groß. Besonders die Datenanbindung über XPath oder XQuery lässt kaum Wünsche offen, ist jedoch auch recht kompliziert und daher für Programmieranfänger weniger geeignet. Alle Steuerelemente, d.h. Eigenschaften der Elemente sind vollständig über die XAML-Syntax anpassbar; damit entfällt oftmals die Notwendigkeit zusätzlichen Quelltext einzusetzen.

2.4.5 Punktdichte

Mit Punktdichte wird die Anzahl der Punkte in einem Längenabschnitt bezeichnet (siehe Abbildung 18). Gebräuchliche Verhältnisse für Drucker sind Rasterpunkte pro Zoll oder engl. Dots Per Inch (DPI). Bei der Punktdichte von Bildschirmen spricht man sinngemäß aber nicht von Dots sondern von Pixeln, weshalb die Dichte dort in Pixel per Inch (PPI) angegeben wird [Watson, 2011]. Der häufig als Synonym verwendete Begriff Auflösung hat jedoch nichts mit dieser Definition zu tun, denn eine Auflösung i.e.S. bezeichnet das Produkt der Zahl von horizontalen und vertikalen Punkten eines gerasterten Bildes.

$$d_{punkt} = \sqrt{Breite^2 + Höhe^2}$$

$$PPI = \frac{d_{punkt}}{d_{phys}}$$

d_{punkt} Diagonale Punkteanzahl in Pixeln

d_{phys} Diagonale Länge des Bildschirms in Zoll

Abbildung 18 Die Formel zur Berechnung der Pixel Pro Zoll (Pixel Per Inch, kurz PPI) für einen Bildschirm

Heutzutage ist die am meisten verwendete Punktdichte 96 Pixel pro Zoll. Diese Punktdichte wird seit der ersten Version von Microsoft Windows verwendet, um Schriften größer darzustellen als sie der Entwickler angedacht hat. Der Xerox Alto und Macintosh nutzten zuerst 72 PPI und konnten damit eine Schriftgröße im gleichen Verhältnis auf dem Monitor ausgeben. Mit einer 10 Punkte Schrift konnten daher Zeichen mit genau 10 Pixeln ausgegeben werden. Diese Größe war für viele Printmedien ausreichend, konnte jedoch bei Monitoren zu klein sein, weil die Entfernung zwischen Benutzer und Bildschirm tendenziell größer ist [Hitchcock, 2005]. Microsoft behob das Problem, indem Windows seitdem 96-PPI nutzt, was die Schriften um ein Drittel größer macht. Der Nachteil liegt nun in den unterschiedlichen Verhältnissen, d.h. die Punkt- und Pixelanzahl von Druck- und Bildschirmschriften sind nicht mehr im Verhältnis eins zu eins.

Seit der Einführung von 96 PPI wurden viele Anwendungen entwickelt, die Annahmen über die Pixelgröße machen oder diesen Wert als konstant betrachten. Besonders bei Bildschirmen mit hohen Auflösungen (bspw. 1920x1080, das ist Full-HD Standard) sind Symbole und Schriften mit 96 PPI für viele Menschen mit einer Sehschwäche zu klein. Daher kann die Punktdichte softwaretechnisch erhöht werden, um Elemente größer darzustellen.

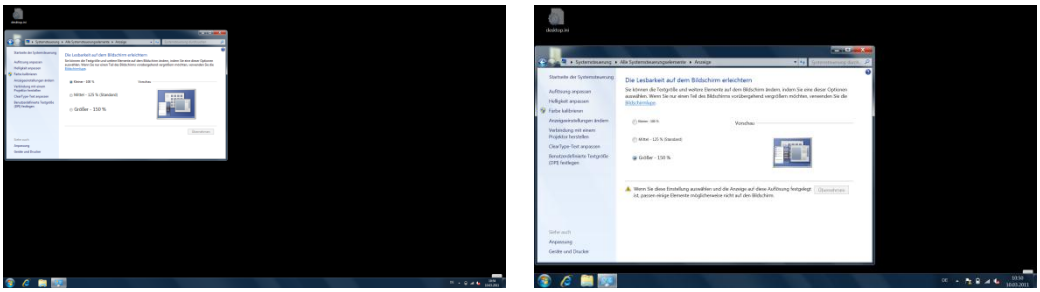


Abbildung 19 Unterschiedliche Punktdichten (hier 96 und 150 PPI) bei gleicher Auflösung und Fenstergröße. Steuerelemente werden bei 150 PPI größer dargestellt als bei 96 PPI.

Daneben ermöglicht die vergrößerte Darstellung der Bildschirmelemente auch eine einfachere Bedienung mit berührungsempfindlichen Bildschirmen. Sie sind einfach leichter zu treffen. Doch das führt auch zu Darstellungsproblemen (siehe Abbildung 20), wie [Hitchcock, 2005] anmerkt:

"[…] these applications have made assumptions about the size of a pixel and many dialog boxes and web pages have been designed around 96 PPI. As newer displays have come along, some, especially laptops have higher pixel densities. Unfortunately if one adjusts for this by using PPIs besides 96, then there is a risk of some applications or web pages not working properly."

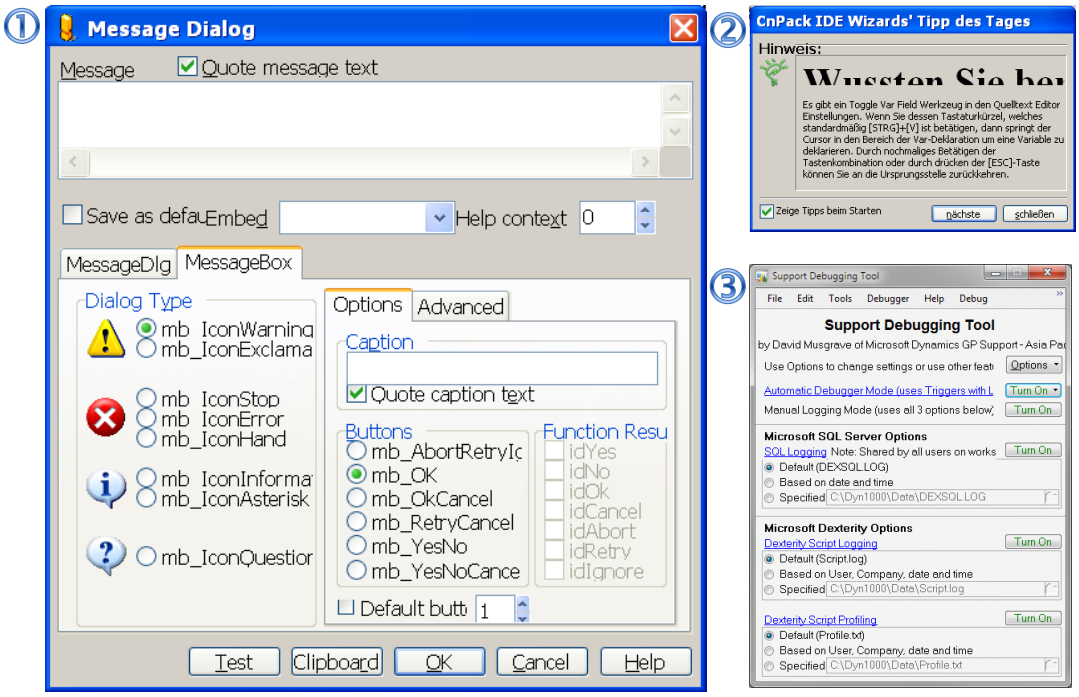


Abbildung 20 Typische Darstellungsprobleme bei höheren PPI Einstellungen
① [GExperts] (Bildschirmfoto), ② [CnPack] (Bildschirmfoto), ③ [Musgrave, 2009])

Bei den immer größeren Bildschirmauflösungen und gleichbleibender Pixelgröße von Kontrollelementen wird einmal auch jede noch so gut bedienbare Anwendung mit den Fingern nicht mehr zu benutzen sein. Es bleibt also nichts anderes mehr übrig, als die Anwendungen, wie es Microsoft nennt, „DPI-Aware" zu entwickeln, um der Miniaturisierung entgegenzuwirken. Denn letztendlich sollte ein Benutzer die globale Einstellung der PPI nicht erhöhen müssen, damit der Benutzer auch noch kleine Kontrollelemente mit seinen Fingern bedienen kann.

2.5 Software Engineering

Software Engineering *ist jede Aktivität, bei der es um die Erstellung oder Veränderung von Software geht, soweit mit der Software Ziele verfolgt werden, die über die Software selbst hinausgehen.*

[Ludewig, et al., 2007]

Die Softwareentwicklung unterlag schon immer stetiger Veränderung. Mit traditionellen Prozessmodellen wie dem Wasserfall- und dem V-Modell und den agilen Vorgehensmodellen wie dem Extreme Programming (XP) wurden immer wieder neue Ansätze und Verfahren benutzt, um die Software nicht nur in der gegebenen Zeit zum Abschluss zu bringen, sondern auch die Wartbarkeit nach dem Projektende auf einem hohen Niveau gewährleisten zu können. Außerdem muss die Software nicht nur korrekt funktionieren, sondern auch an neue, nachträgliche Anforderungen angepasst werden wie beispielsweise einer Portierung auf eine neue Plattform. Doch Prozess- und Vorgehensmodelle sagen nur etwas über den Ablauf des Projekts aus und stellen natürlich kaum oder keine Forderungen an die Architektur und Datenverarbeitung der Software. Mit dem Anspruch auch später noch Änderungen an der Software vornehmen zu können, müssen andere Ansätze gewählt werden.

Die Entwicklung von Software mit der Hilfe von Modellen ist mittlerweile in vielen Sprachen (insbesondere Java) angekommen und wird überall dort eingesetzt, wo die Software auch später noch schnell und einfach angepasst werden muss. Modelle beschreiben eine Untermenge der betrachteten Wirklichkeit. Da nicht immer alle Eigenschaften interessant sind, werden sie bei der Beschreibung weggelassen („abstrahiert“). Ein wesentlicher Effekt davon ist, dass die Beschreibung einfacher und weniger komplex ist als die Wirklichkeit. Modelle helfen also nicht nur die Realität zu erfassen, sondern sie auch verstehen zu lernen (vgl. [Ludewig, et al., 2007] und [Seidewitz, 2003]). Zur Beschreibung von Modellen werden Modellierungssprachen wie DSL (Domain Specific Language) oder UML (Unified Modeling Language) eingesetzt.

Mit dem vorwärts gerichteten Erzeugen (engl. Forward Engineering) von Software aus abstrakten Modellbeschreibungen [Chikofsky, et al., 1990] beschäftigt sich auch die Object Management Group (OMG). Die OMG hat daher eine modellgetriebene Architektur vorgestellt, die die Wartbarkeit von Software erheblich verbessern kann. Im folgenden Kapitel 2.5.1 werden dieses Prinzip der OMG sowie beispielhafte Softwareumsetzungen gezeigt.

Unglücklicherweise wird oder kann nicht jede Software mit einem modellgetriebenen Ansatz entwickelt werden und trotzdem ist auch in so einem Fall eine Anpassung notwendig, um eine teure Neuentwicklung zu vermeiden. Daher gibt es auch hier Ansätze, um Software zu verstehen, d.h. aus dem Design und der Kodierung die Datenmodelle und Programmabläufe zurückzuentwickeln. Das Kapitel 2.5.2 Reverse Engineering beschäftigt sich mit der Rückgewinnung von abstrahierten Informationen aus dem Design von Anwendungsoberflächen. Vorhandene Lösungen und Ansätze für eine GUI Erkennung und automatische Modellierung werden dort vorgestellt.

2.5.1 Forward Engineering am Beispiel der modellgetriebenen Entwicklung

Seit Ende des 20. Jahrhunderts beschleunigte sich die Entwicklung von Softwareprodukten dramatisch. Software wurde als Lösung für alles und jeden gesehen, denn sie half die Unmengen von Informationen zu bewältigen, die das Informationszeitalter mit sich brachte. Doch auch wenn solch eine Software streng nach Plan zusammengesetzt wurde, war sie nicht für die Ewigkeit bestimmt (“[…]this application will only be needed for the next few years” [Miller, et al., 2003]). Sobald sich die Gesetzeslage oder der Wirtschaftsprozess änderte, musste auch die gesamte Software an diese Umstände angepasst werden. Daher wurde jede neue Technik als Lösung für die aktuellen Probleme angepriesen und verkauft [Miller, et al., 2003]. Jedoch war und ist es immer noch schwierig, Informationen für verschiedene, meist inkompatible Anwendungen zur Verfügung zu stellen

und synchron zu halten. Es stehen eine Vielzahl von verteilten Daten, Objekten und Komponenten sowie Webdienste zur Verfügung, die nicht miteinander kommunizieren können, weil sie streng nach Vorschrift – und oftmals mit Scheuklappen – gebaut wurden. Aber was davon überlebt den stetigen Wandel in der IT? Es ist immer wieder überraschend wie kurz- oder langlebig Software sein kann.

Es wurde daher notwendig die Verwaltung der Geschäftslogik nicht vollständig mit der Programmlogik zu verknüpfen und stattdessen so viel Unabhängigkeit zu bewahren wie möglich. Erst dadurch ist es möglich, dass Geschäft und Software sich separat entwickeln können. Denn sollte die Software eines Tages veraltet sein, kann eine neue deren Platz einnehmen [Leymann, 2008].

Eine Technik, welche die modellgetriebene Entwicklung (engl. Model Driven Engineering, kurz MDE) umsetzt, ist die modellgetriebene Architektur (engl. Model Driven Architecture, kurz MDA). Sie wurde von der OMG entwickelt und ermöglicht Abläufe und Regeln formal zu spezifizieren und in Modelle zu packen, die unabhängig von der eingesetzten Plattform sind. Denn verschiedene Plattformen (z.B. CORBA, J2EE, Microsoft .NET) bieten oftmals Schnittstellen an, die nicht nur unterschiedlich definiert sein können, sondern sich auch anders verhalten und damit separate Entwicklungen erfordern (für Windows, Linux, Java, .NET, EJB usw.).

MDA unterstützt nach [Miller, et al., 2003] dabei die folgenden Ansätze:

- Ein Modell zu spezifizieren, das unabhängig von der eingesetzten Plattform ist.
- Die Plattform auszuwählen oder selbst zu spezifizieren …
- … und die Modellspezifikation in eine Entsprechung für eine bestimmte Plattform zu übertragen.

Durch die Trennung von Modell (Prozesse, Regeln) und Plattform gelingt es MDA plattformunabhängig (auch portabel), plattformübergreifend (auch interoperabel) und wiederverwendbar zu sein. Dazu durchläuft ein plattformunabhängiges Modell (engl. Platform Independent Model, kurz PIM) mehrere Transformationsschritte bis es in einer Anwendung umgesetzt wird. Dieser Prozess wird im Englischen auch mit Forward Engineering bezeichnet, um ihn, dort wo notwendig, vom Reverse Engineering zu unterscheiden [Chikofsky, et al., 1990].

Der Prozessablauf ist in Abbildung 21 dargestellt. Zuerst einmal wird aus Anwendungsfällen (engl. Use Cases) und Beschreibungen für Problemlösungen (Domänenmodell) in einer Textform oder grafischen Notation (z.B. Unified Modeling Language, kurz UML) per Hand ein plattformunabhängiges Modell (Platform Independent Model, kurz PIM) entworfen. Beispielsweise möchte ein Kunde sein Geld abheben. Diese Art von Beschreibungen gibt normalerweise keine Auskunft über die notwendigen Details mit denen gearbeitet werden soll. So benötigt ein Kunde u.a. einen Namen und ein Konto bei der Bank, um Geld abheben zu können. Doch diese Informationen sind eigentlich nur für das System notwendig, denn wie der Kunde sein Geld bekommt ist ihm nicht wichtig. Geschäftsmodelle werden daher in MDA auch als berechnungsunabhängige Modelle bezeichnet (Computation Independent Model, kurz CIM).

Aus der Transformation von CIM nach PIM entsteht ein erster Entwurf, der noch unabhängig von den eingesetzten Technologien ist. Dies sind oftmals in UML notierte Diagramme, wie Klassendiagramme, die Klassen, Attribute und Methoden spezifizieren. Mit solch einem Modell kann durch eine automatische Transformation bereits ein plattformspezifisches Modell (kurz PSM) generiert werden. Bei Klassendiagrammen werden dazu u.a. allgemeine Typen, wie *Ganzzahl* oder *Text* durch die jeweilige Sprache des Zielmodells ersetzt, also *Integer* oder *String*. Das Modell ist daher an die jeweilige Plattform, die diese Typen unterstützt, gebunden.

Die Generierung des PSM kann in MDA noch durch zusätzliche Faktoren beeinflusst werden, die von der eingesetzten Plattform abhängen. So kann mit Optionen die Transformation beeinflusst werden, um beispielsweise einen bestimmten Architekturstil zu erreichen oder anzupassen [Miller, et al., 2003 S. 13]. Mit bereits eingesetzten Erfolgsmethoden (Best Practices) und Entwurfsmustern kann PSM um Funktionen erweitert werden, die nicht zum PIM gehören. So werden dadurch die Typen der PIM Klassen auf die entsprechenden Typen der Zielsprache des PSM abgebildet oder Getter- und Setter-Methoden für den Zugriff auf Attribute generiert. Mit Markierungen können Elemente im PIM auch auf einer semantischen Ebene situationsbedingt transformiert

werden. Dies ist besonders hilfreich, wenn es mehrere Entsprechungen im PSM gibt. So gibt es auf einer Plattform oftmals verschiedene Typen für Gleitpunktzahlen (Float, Double, Extended) und GUI Generatoren können ein Listenelement in ein einfaches Listenfeld oder Dropdown-Listenfeld (siehe auch Tabelle 17 Steuerelemente: Bezeichnung, Symbol und Kurzbeschreibung) transformieren.

Der letzte Schritt besteht in der Umsetzung des plattformspezifischen Modells in Quelltextform, so dass die Anwendung in einer gewählten Programmiersprache (Java, C# usw.) vervollständigt werden kann. Dazu werden Klassendiagramme, die vorher noch sprachunabhängig waren, in Quelltext transformiert. Methoden, die durch das PIM und PSM definiert wurden sind nun nicht nur definiert, sondern auch implementiert, d.h. ausführbar, auch wenn sie womöglich noch keinen Code enthalten. Die letzte Transformation kann auch Oberflächen erzeugen, wenn die Modelle diese vorher beschrieben haben. Dadurch ist es leicht möglich, ganze Anwendungsgerüste in kurzer Zeit zu erstellen.
Ein wichtiger Aspekt von MDA ist, dass die Generierung des plattformspezifischen Modells auch übersprungen werden kann [Miller, et al., 2003 S. 7]. Aus dem plattformunabhängigen Modell entsteht dabei ohne Umweg sofort die Anwendung. Das PSM könnte nach MDA trotzdem abseits des normalen Anwendungsablaufs, z.B. für die Fehlersuche, generiert und verwendet werden.

Neben der Trennung von Modellen innerhalb der modellgetriebenen Architektur, definiert OMG auch die Modell-Transformationen. Dabei können Modelle selbst wieder in Modelle abgebildet (Model to Model, kurz M2M) oder daraus Code erzeugt werden (Model to Text, kurz M2T). Eine „Modell nach Modell Transformation“ kann dabei vertikal oder horizontal geschehen, d.h. die vertikale Transformation ändert die Abstraktionsstufe (z.B. von PSM nach PIM) während in der horizontalen Transformation üblicherweise nur die Darstellung des Modells ändert (z.B. Refactoring, grafische Umgestaltung). Die Modelltransformation ist dabei nicht auf einzelne Modelle beschränkt, sondern kann auch mehrere Quell- und Zielmodelle einbeziehen, die bei der Transformation benötigt werden. OMG hat die M2M und M2T Transformationssprachen QVT (Query/View/Transformation, [OMG, 2011]) und MOF (siehe [OMG, 2008]) spezifiziert, die von Dritten, mehr oder weniger genau, umgesetzt wurden. Dazu zählen beispielsweise M2M, M2T und ATL von [The Eclipse Foundation], dem proprietären Werkzeug ModelMorf von [TCS] und dem Open Source Projekt [Tefkat].

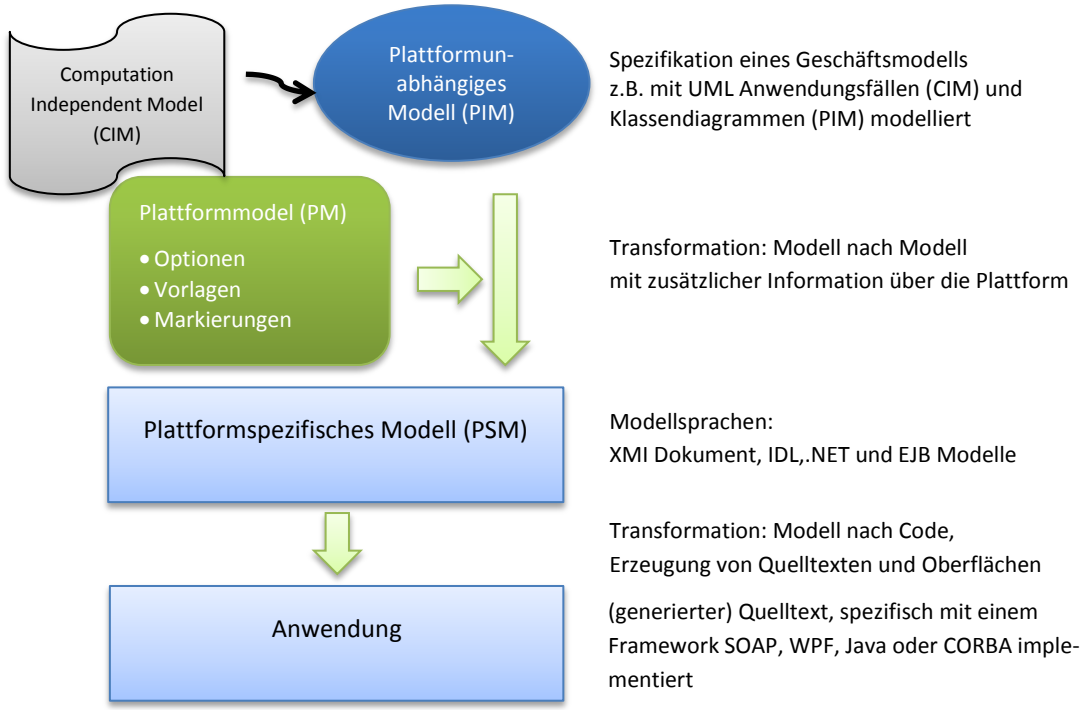


Abbildung 21 Die Modellgetriebene Architektur ermöglicht aus Geschäftsmodellen automatisch Quelltext zu generieren

Frameworks und Werkzeuge, die eine modellgetriebene Architektur bereitstellen oder sie nutzen, gibt es einige. Eine Auswahl davon wurde in einer Studie von [Schlegel, et al., 2010] untersucht. Darin wurden Werkzeuge und Frameworks zur Generierung von Benutzungsschnittstellen aus Modellen getestet, u.a. OpenXava, JAXFron, Wicket RAD, JMatter und das Roma Framework. Zwei weitere Frameworks, die jedoch nicht in der Studie getestet wurden, sind AndroMDA und MARIA, die im Folgenden vorgestellt werden.

MARIA

MARIA, ein Mischkurzwort für **M**odel-based l**A**nguage fo**R** Interactive **A**pplications, ist, wie der Name schon sagt, eine modellbasierte Sprache für interaktive Anwendungen. MARIA wurde von [Paternò, et al., 2009] entworfen und entstand als eine Weiterentwicklung von TERESA. Mit MARIA sollte u.a. mehr Kontrolle und Flexibilität für den Entwickler möglich sein und die Transformationen nicht mehr in Quelltextform kodiert vorliegen. Damit soll die Entwicklung in Richtung dienstorientierte Architekturen (engl. Service Oriented Architectures, kurz SOA) in ubiquitären Umgebungen erleichtert und gefördert werden. Das MARIA Framework besteht daher aus einer Vielzahl von Beschreibungssprachen: für abstrakte, konkrete und vokale UIs sowie für mobile und multimodale Oberflächen. MARIA ermöglicht dies durch eine eigene plattformunabhängige Sprache, der MARIA Universal Declarative Language (siehe Abbildung 22). Die Sprache wurde in XML (MARIA XML) definiert und besitzt einen modularen Aufbau, der die folgenden Haupteigenschaften ermöglicht:

- Ein Datenmodell für Eingabewerte und die Verknüpfung von Datenobjekte mit der Oberfläche.
- Ereignisse für Änderungen an Eigenschaften in den abstrakten und konkreten Stufen
- Ein erweitertes Dialogmodell, das Bedingungen und Operatoren, beschrieben durch die CTT Notation, definiert.
- Eine dynamisch anpassbare Menge von Benutzerelementen, um das Layout und die Navigation zwischen den Darstellungen den Gegebenheiten anzupassen.

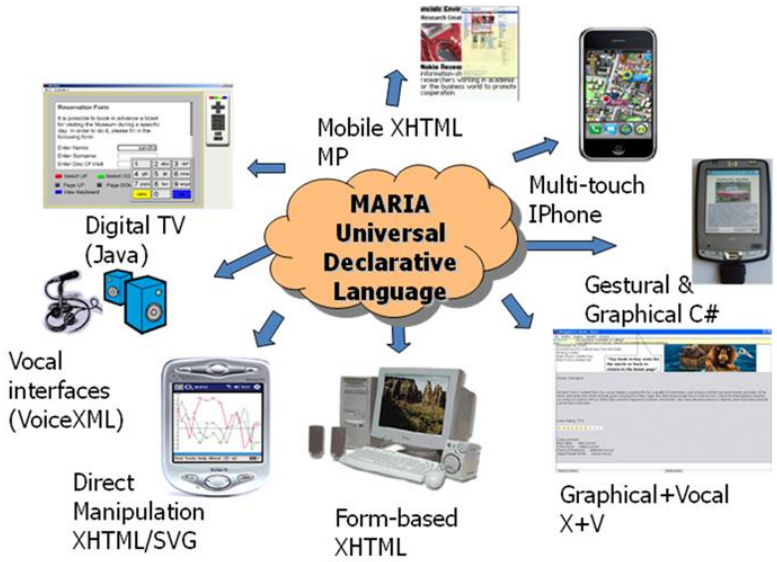


Abbildung 22 Die Möglichkeiten von MARIA [HIIS Laboratory] erlauben eine Anwendung auf verschiedensten Plattformen zu betreiben

Um das MARIA Framework herum wurden einige Werkzeuge entworfen. So wurde eine Entwicklungsumgebung MARIAE (siehe [HIIS Laboratory] und Maria Environment) gebaut, die Aufgabenmodelle in der CTT Notation beschreiben lässt. Weiterhin wurden mit ReverseMARIA und WebRevEng Werkzeuge entworfen, die aus Webseiten Aufgabenmodelle als CTT generieren. Sie können in MARIA eingesetzt und weiterentwickelt werden.

AndroMDA

AndroMDA (ausgesprochen nach IPA: [anˈdʀɔmeˌdaː], [AndroMDA, 2011]) ist ein Open Source Framework mit modellgetriebener Architektur, das durch eine Vielzahl von Erweiterungsmöglichkeiten profitiert. Modelle können mit Hilfe von externen CASE Werkzeugen (z.B. UML Designer) erstellt und über XMI importiert werden. Zusätzlich können Plug-Ins dazwischen geschaltet werden, um die Verwaltung und Generierung von benutzerdefinierten Ausgabekomponenten in Quellformaten wie Java, C# .NET, HTML und weitere zu ermöglichen. Derzeit bietet AndroMDA die größte Unterstützung für J2EE mit Projekt- und Codeschablonen. Um AndroMDA nutzen zu können muss Java mit Maven oder Ant installiert sein. Der generierte Code läuft dann auf einem Applikationsserver wie JBoss oder Tomcat.
Der erste Schritt beim Aufsetzen eines Projektes besteht darin Schablonen zu nutzen und anzupassen, die für die Erzeugung von Quellcode verantwortlich sind. Sammlungen von Schablonen, sogenannte „cartridges“ (engl. für auswechselbare Einsätze), werden durch die „AndroMDA Code Generation Engine“ zu Quelltext transformiert. Es existieren bereits eine Menge solcher Sammlungen von Drittherstellern, die an die eigenen Bedürfnisse angepasst werden können. Sobald die Schablonen vorbereitet sind, können Modelle verarbeitet werden. Dabei unterstützt AndroMDA den XMI Standard von OMG, so dass ein beliebiges UML Werkzeug verwendet werden kann.
Die Hauptaufgaben von AndroMDA bestehen darin, das UML-Modell (PIM) umzuwandeln, d.h. daraus einen Klassenbaum zu erstellen und Quelltexte zu generieren. Der Objektbaum wird dazu nach den Stereotypen der aufgelisteten Klassen untersucht und für jeden gefundenen Stereotyp ein entsprechendes „cartridge“ zugeordnet. Sobald dieser Vorgang abgeschlossen wurde, werden die anfangs ausgewählten Schablonen für jedes „cartridge“ aufgerufen. Sie analysieren das PIM und erzeugen daraus ein plattformspezifisches Modell (PSM), das in den Schablonen genutzt wird, um letztendlich den Quelltext zu erzeugen.
AndroMDA ist durch den modularen Aufbau bestrebt eine hohe Erweiterbarkeit und Flexibilität zu erreichen. Es können eigene Schablonen erstellt und genutzt werden oder aber auch eigene Einsätze. Außerdem können Quelltexte für andere Sprachen erzeugt werden, die standardmäßig nicht von AndroMDA unterstützt werden. Wenn sich Schablonen und Einsätze für das eigene Projekt finden lassen, kann so eine modellgetriebene Entwicklung im Softwareprojekt genutzt werden. Dies kann besonders schnell gelingen, wenn Java eingesetzt wird. Für andere Sprachen mag der Einstieg jedoch nicht so einfach sein, weil es an den notwendigen Code-Schablonen fehlt.

Der modellbasierte Ansatz besitzt, wie jede andere Technologie Stärken (✓) und Schwächen (✗). Einige davon sind die Folgenden (nach [Leymann, 2008], [Schmid, 2010], [Wikipedia, 2011]):

- ✓ Jedes Modell ist unabhängig von dem anderen und bildet eine eigene Einheit mit wohldefinierter Abstraktionsstufe (Modularität). Während Aufgabenmodelle die Geschäftsprozesse definieren, implementieren konkrete Modelle die einzelnen Aufgaben über Klassenmodelle und Klassenbeziehungen bis hin zur Codeebene.
- ✓ MDA ermöglicht unabhängig von Plattformen und Frameworks zu arbeiten, d.h. der Entwicklungs- und Codierungsprozess ist streng getrennt. Ändert sich ein Modell kann daraus immer wieder plattformspezifischer Code generiert werden. Aufwändige Codeanpassungen entfallen dadurch. Dies kann besonders bei großen und komplexen Anwendungssystemen häufig eine Fehlerquelle sein (Flexibilität).
- ✓ Die Modelle können geändert und die Änderungen durch übergeordnete Modelle verifiziert werden (Sicherung der Korrektheit).
- ✓ Durch die Automatisierung wird die Produktivität gesteigert.

- ✗ Negativ kann sich der Aufwand bei der Erzeugung von geeigneten Modellen erweisen, wenn die Anwendung nicht groß genug ist wie bei einmaligen Projektarbeiten. Stattdessen könnte in der gleichen Zeit bereits die Anwendung umgesetzt werden.

- Der modellbasierte Ansatz erfordert neue Denk- und Herangehensweisen für die Entwickler. Nicht jeder Entwickler ist damit vertraut oder kann gut mit abstrakten Modellbeschreibungen umgehen.

2.5.2 Reverse Engineering

Reverse engineering *is the process of analyzing a subject system to [a)] identifies the system's components and their interrelationships and [b)] create representations of the system in another form or at a higher level of abstraction.*

[Chikofsky, et al., 1990]

Eine Software zurückzuentwickeln wird notwendig, sobald Quellen und Dokumentation für das System nicht mehr vorhanden sind und dessen Wert erhalten bleiben soll. Dazu werden nach Chikofsky zuerst einmal die Systemkomponenten erkannt und in Beziehung gebracht (Teil a). Dann können Quelltexte aus Binärcode entwickelt und Aufgabenmodelle, z.B. in UML, durch eine Systemanalyse erstellt werden (Teil b). Reverse Engineering (RE) bedeutet also den Entwicklungsprozess rückwärts durchzuführen und das System zu verstehen. Mit RE wird jedoch keine Änderung oder Ergänzung am System durchgeführt. Dieser Schritt, das Anpassen der Software an eine neue Umgebung oder Plattform, wird daher Migration oder Portierung genannt [Moore, et al., 1993]. Im Migrationsprozess wird das Wissen des Reverse Engineerings verwendet, um schneller und kostengünstiger, d.h. abgekürzt Software Engineering betreiben zu können. Da die Migration die vorhandene Funktionalität möglichst zu 100% übernehmen soll, sind große Teile zu verstehen und neu zu entwickeln („*[Information systems] also tend to be very large systems, hundreds of thousands, even millions, of lines of code.*“, [Moore, 1995]). Das bedeutet, dass die manuelle Migration nicht ohne große Kosten und Fehler zu bewerkstelligen ist. Es müssen daher neue Methoden und Techniken gefunden werden einen Großteil der Arbeit zumindest zu halb-automatisieren.

Technisch wird Reverse Engineering auf zwei Arten durchgeführt: dynamisch oder statisch (vgl. [Grilo, et al., 2007]). Im ersten Fall wird das System analysiert, indem die vorhandenen Bestandteile (u.a. Variablen und Klassen) erkannt und deren Beziehung (z.B. Vererbung und Methodenaufrufe) verstanden wird. Im Gegensatz dazu wird beim dynamischen RE das System analysiert während es ausgeführt wird. Dadurch können u.a. Speicherwerte, Methodenaufrufe und Nebenläufigkeit zur Laufzeit beobachtet oder sogar geändert werden. Zudem bleibt das dynamische RE die einzige Möglichkeit, die Software zu verstehen, wenn Quelltext und Dokumentation verloren gegangen sind. Doch ein System kann nur dann vollständig verstanden werden, wenn es sowohl statisch als auch dynamisch zurückentwickelt wird [Grilo, et al., 2007].

Mit Reverse Engineering können nicht nur Klassenstrukturen und Programmabläufe verstanden werden, sondern auch Benutzungsschnittstellen. Bei neuen Plattformen müssen dann insbesondere auch die Oberflächen angepasst werden. Diese Anpassung stößt jedoch auf eine Vielzahl von Problemen, wie [Moore, et al., 1993] anmerken:

1. Bildschirme bieten heutzutage sehr unterschiedliche Auflösungen und Interaktionsmöglichkeiten („Touch“) an. („*Display technologies*“)
2. Es hat sich eine Vielzahl von unterschiedlichen Entwicklungsschnittstellen für Oberflächen etabliert. („*Lack of standards*“)
3. Das Aussehen der Oberflächen kann sich zwischen Plattformen drastisch unterscheiden. Bei der Migration muss daher entschieden werden, ob das originale Aussehen der Oberfläche beibehalten oder an die neue Plattform angepasst wird. („*Look and Feel*”)
4. Funktionsunterschiede (z.B. Vorhandensein von bestimmten Steuerelementen) der Plattformen müssen bei der Migration berücksichtigt werden. („*Functionality changes*“)

5. Die Art und Weise, wie bei einer Software die Benutzungsschnittstelle umgesetzt wurde, d.h. vollständig getrennt, per Modul oder verteilt, bestimmt den Aufwand, der betrieben werden muss, um die plattformabhängigen Oberflächenkomponenten zu migrieren. („*Integration of the user interface*“)

Zum Reverse Engineering von bestehenden Dialoganwendungen existieren bereits einige Konzepte und Lösungen. So wurde für die MDA Projekte MARIA und MARIAE von [HIIS Laboratory, 2010] das Werkzeug Reverse-MARIA entwickelt, das die Modelle aus vorhandenen Webseiten zurückentwickeln kann. Von derselben Stelle stammt auch WebRevEng. Es kann Aufgabenmodelle in der ConcurTaskTrees-Notation aus HTML Quellen generieren und sie für die Weiterverarbeitung speichern. Eine ähnliche Lösung bieten auch [Bandelloni, et al., 2008] mit ReverseAllUI. Damit können fast beliebige Dialogbeschreibungssprachen (z.B. HTML, XHTML, aber auch TERESA XML, siehe [Berti, et al., 2004]) in ein einheitliches Aufgabenmodell zurückentwickelt werden. Modellgetrieben arbeiten auch die Autoren [Sánchez Ramón, et al., 2010], um Oberflächen aus Rapid Application Development Softwareentwicklungen (RAD) zurück zu gewinnen. Sie überführen die Oberfläche der Anwendung in ein plattformunabhängiges Zwischenmodell („RAD model“) und generieren daraus wieder ein neues konkretes UI Modell („Concrete User Interface Model“), welches als Ausgangsmodell für neue Oberflächen dient.

Zwei weitere Lösungen befassen sich mehr mit der Interpretation der Steuerelemente von Dialogen und ihrer Anordnung. Der Algorithmus des Auckland Layout Model (siehe unten) sorgt beispielsweise dafür, dass Steuerelemente entsprechend ihrer von Entwickler zugewiesenen Bedeutung mehr oder weniger Platz in einem Dialog erhalten. Ein bestehendes Layout einer Altanwendung kann durch den Algorithmus zurückentwickelt werden, um eine spätere Neuanordnung der Steuerelemente zu ermöglichen. Weiterhin ist es für Reverse Engineering von Oberflächen wichtig, die Steuerelemente zuverlässig erkennen zu können, um, je nach Typ des Elements, eine Abbildungsregel anwenden zu können. Da oftmals für viele alte Anwendungen keine Quelltexte mehr existieren, wurde für die Migration dieser Anwendungen ein neuer Ansatz entwickelt. Unter Windows wurden daher die Dialoge mit Funktionen der Windows API zur Laufzeit analysiert und kopiert. Dazu wurde jedes Steuerelement erkannt und dessen Eigenschaften kopiert wie Typ und Größe. In einer Studie wurde diese Methode an mehreren Windows Anwendungen durchgeführt und überprüft.

Die letzten beiden Lösungen befassen sich direkt mit Dialoganwendungen, die auch in dieser Arbeit das Thema sind. Daher soll im Folgenden etwas näher auf die Umsetzung dieser Lösungen eingegangen werden.

Automated Reverse Engineering of Hard-Coded GUI Layouts

Mit dem **Auckland Layout Model** (ALM), benannt nach der Universität des Autors [Lutteroth, 2008], beschreibt der selbige eine mathematische Technik, die die Anordnung („Layout“) von Oberflächenelementen spezifiziert. Das ALM beschreibe dabei das Layout so exakt und formal, dass die Spezifikation auch ohne ein „Layout Manager“ sinnvoll sei.

$$area = {}_{def} (x_1, y_1, x_2, y_2, content, size_{min}, size_{pref}, size_{max})$$

def	definiert durch
content	Inhalt des Bereichs (area)
$size_{min}$	Minimale Größe des Bereichs
$size_{pref}$	Bevorzugte Größe des Bereichs
$size_{max}$	Maximale Größe des Bereichs

Abbildung 23 Nach [Lutteroth, 2008] besteht ein Bereich („Area“) aus den Koordinaten, Inhalt sowie Grenzen für eine Änderung der Größe.

Denn Luttheroth bemängelt die absolute Positionierung von Steuerelementen im Quelltext („[…] *the lack of dynamic layout.*“) und die daraus ergebenden Nachteile für den Benutzer der Anwendung („[…] *users may want to change the amount of screen real-estate that is allocated to a GUI by changing the size of its window*“, “[…] *the content shown in a control may change so that its size needs to be adjusted*”). Zudem sei das Mischen von generierten und handgeschriebenem Quelltext für Oberflächen problematisch, weil Werkzeuge wie Ober-

flächeneditoren mit selbstgeschriebenen Algorithmen nicht umzugehen verstehen („*it is effectively not possible anymore to use a GUI design tool*“).

Das ALM kehrt der statischen Positionierung von Steuerelementen den Rücken und spezifiziert stattdessen Invarianten für die Oberfläche. Diese Invarianten werden als Bedingung für eine lineare Optimierung des Layouts benutzt, sobald Anpassungen der Oberfläche notwendig sind. Um die Invarianten für die lineare Optimierung einfacher zu spezifizieren, führt Lutteroth zusätzliche Abstraktionen („*Features*“), wie „*tabstops*“, „*areas*“ und „*linear constraints*“ ein. Mit Tabstopps wird ein virtuelles Raster für die Ausrichtung von Steuerelementen definiert. Die Positionen der Elemente werden durch diese Tabstopps (x- und y-Richtung) ausgedrückt und erst im Layout-Prozess mit konkreten Werten belegt. Mit Areas (siehe Abbildung 23) werden Flächen auf der Oberfläche definiert, d.h. deren Position und Größe (x_1 , y_1 , x_2 und y_2 in Tabstopps), das beinhaltete Steuerelement (*content*) sowie minimale ($size_{min}$), maximale ($size_{max}$) und bevorzugte ($size_{pref}$) Größenänderungen beschreiben. Die Gleichung wird im Layout-Prozess eingesetzt, um die dadurch spezifizierten Invarianten beim dynamischen Berechnen der Größenveränderung der Oberfläche zu berücksichtigen.

Da Tabstopps keine konkreten Werte (z.B. in Pixeln) enthalten, werden sie über lineare Beschränkungen („linear constraints“) spezifiziert. So können absolute Positionen ($x_1 = 100$), Größen ($x_1 – x_2 = 100$) oder komplizierte Abhängigkeiten (auch mit Ungleichungen) von Steuerelementen definiert werden. Doch nicht immer können alle Beschränkungen gemeinsam erfüllt werden (keine Lösung), so dass [Lutteroth, 2008] zwischen harten und weichen Beschränkungen unterscheidet. Während die harten Beschränkungen immer erfüllt sein müssen, können weiche Beschränkungen in ALM mit Kostenparametern versehen werden, die Durchsetzungsfähigkeit zu anderen weichen Beschränkungen angeben.

Für die Herstellung einer Dialogspezifikation im ALM nutzt Lutteroth einen Algorithmus, der die Anordnung der Steuerelemente zurückentwickelt. Dabei werden absolute Positions- und Größenangaben in Tabstopps überführt und zueinander in Beziehung gesetzt (Reihenfolge der Elemente). Der Algorithmus ist laut dem Autor schnell und kann daher beispielsweise mit wenig Aufwand auch nachträglich das Layout während der Laufzeit vergrößern und verkleinern. Mit Hilfe der Heuristik kann der Algorithmus noch weiter verfeinert werden. So können etwa vom Entwickler schlecht ausgerichtete Elemente (*„A control might just be misplaced by a single pixel, and the deviation might only be visible under closer observation.* “) durch Definieren einer Epsilon Umgebung (z.B. für eine horizontale Ausrichtung:$x_{min} < \varepsilon < x_{max}$) immer noch auf dieselbe Tabstopp-Position gebracht werden („*Fuzzy Alignment*“). Weiterhin können gleichartige Elemente einer Spalte auf dieselbe Größe gebracht werden („*Standardized Sizes*“) oder die Oberflächengröße relativ zur Bildschirmgröße gewählt werden, indem die verwendete Einheit umgerechnet wird („*Adjustment of Units*“).

Eine Umsetzung des ALM Algorithmus fand mit C# in Visual Studio statt. Die Implementierung von [Lutteroth, 2008] führt zur Laufzeit das obige Reverse Engineering mit dem Algorithmus durch und ermöglicht so eine nachträgliche Layoutveränderung bei Windows Forms .NET Anwendungen. Die Dialogelemente werden dort zur Entwicklungszeit als Quelltext statisch definiert und ermöglichen daher kein flexibles Layout.

User Interface Migration of Microsoft Windows Applications

Mit der Zurückentwicklung und Portierung von Altanwendungen auf neue Plattformen beschäftigt sich [John, 2009]. Durch das Fehlen von Quelltexten geht er den Weg des dynamischen Zurückentwickelns auf Windows Betriebssystemen. Er erstellte dazu eine Anwendung in Visual Basic, welche die Elemente einer beliebigen laufenden Dialoganwendung mit Hilfe der Low-Level Windows API (EnumWindows, EnumChildWindows, GetWindowInfo) extrahiert und deren Elementeigenschaften in einer Datenbank speichert. D.h. es werden Position, Größe, Text und einiges mehr ausgelesen, um später eine neue Oberfläche rekonstruieren zu können. Zusätzlich fügt [John, 2009] noch ein Bildschirmfoto des Dialogs bei, um, wie er schreibt, bei der Identifizierung von Steuerelementtypen und deren Abbildung auf andere entsprechende Elemente zu unterstützen. Er ver-

sucht dabei durch Vergleiche von vorhandenen Elementbildern unbekannte Steuerelemente zu erkennen und neu zuzuordnen. Da dies nicht immer automatisch funktioniert, ist es möglich eine manuelle Anpassung vorzunehmen. Der automatische Teil verwendet dagegen Abbildungsregeln, d.h. Elementtypen werden mit den Typen aus einer Liste verglichen und entsprechenden neuen Elementen automatisch zugeordnet: Zum Beispiel werden die Elemente TCheckBox und ThunderRTCheckBox zu CheckBox. Das Werkzeug wurde in einer Studie an 20 bekannten Windows Anwendungen geprüft und konnte fast 76% der Steuerelemente erkennen und neuen zuordnen. Doch einige Elemente wurden auch dann erkannt, obwohl sie normalerweise nur bei bestimmten Aktionen sichtbar werden (Hover-Effekt). Andere Elemente konnten zwar erkannt werden, deren Sinn oder Zweck war jedoch durch ihre Generizität (z.B. benutzerdefinierte Steuerelemente, Toolbar-Buttons) nicht zu ermitteln. Zudem liefert die benutzte API keine Information über die Farbgestaltung der Elemente, so dass die Farben nur über ein Foto erkannt werden können. Die Erkennung lief dabei immer mit derselben Bildschirmauflösung (und vermutlich auch Punktdichte-Einstellung) und bildet die Dialoge daher immer im Original ab. Nach dem Autor seien daher zusätzliche Kodierungen für Skalierung und Positionsänderung notwendig.

Die beschriebene Lösung besitzt den Vorteil, dass damit fast jede beliebige GUI kopiert werden kann, deren Quelltext nicht mehr vorhanden ist. Dabei werden alle Elemente, ob sichtbar oder nicht, gefunden und Entwickler können diese in einem separaten Prozess neu mit dem Programmablauf verknüpfen. Doch nicht alle Oberflächen lassen sich mit der Low-Level API durchforsten. Sobald GUI Elemente nicht mit dem Windows API Befehl *CreateWindow* erstellt wurden, können auch API Funktionen wie EnumChildWindows oder GetWindowInfo diese nicht ermitteln. Dies ist beispielsweise in Anwendungen der Fall, die ihre Oberfläche mit DirectX oder OpenGL zeichnen, wie dem Windows Presentation Foundation. Für die Windows API besteht dann eine solche Anwendung nur aus einem Fenster mit unbekanntem Inhalt.

2.6 Migration

Die *Migration* (lat. Wanderung) beschreibt die Umrüstung einer Anwendung in eine neue Systemumgebung [Fischer, et al., 2008]. Dieser Vorgang kann von einem Menschen oder einer Maschine vollständig oder nur zum Teil durchgeführt werden. Im besten Fall erzeugt die Umrüstung eine funktionale Kopie der Anwendung auf der neuen Plattform. Eine Anwendung wird migriert, indem deren Komponenten analysiert, kopiert und angepasst werden. Die einfachste Art der Migration ist das Kopieren von Komponenten (z.B. GUI, Programmlogik oder Teile davon), d.h. die ursprüngliche Funktion der Altanwendung wird vollständig erhalten. Dies ist jedoch nur möglich, falls die Zielplattform dies auch unterstützt (z.B. Prozessorleistung, Bildschirmauflösung usw.). Ist dies nicht der Fall, dann müssen die Komponenten der Zielplattform angepasst werden und zwar so, dass sie unter dem neuen System genutzt werden können. Im schlechtesten Fall unterstützt die Zielplattform eine oder mehrere Funktionen nicht (z.B. 3D GUI auf Mobilgeräten oder biometrische Identifikation). Die Komponenten müssen dann entweder ersetzt oder entfernt werden, um die Migration vervollständigen zu können.

Bei der Migration von Benutzungsschnittstellen werden die Steuerelemente einer Plattform auf die entsprechenden Steuerelemente einer anderen Plattform abgebildet [Moore, et al., 1993]. Diese Aufgabe kann vereinfacht werden, indem Detailinformationen weggelassen und die Abbildungen damit abstrakter werden. So wird das spezifische Element „Schalter“ zum generischen Eingabeelement oder das „Dropdown-Listenfeld“ wird mit dem „Listenfeld“ (vgl. Tabelle 17 auf Seite 134) zum allgemeinen Auswahlelement.

Die Durchführung der Migration teilt Moore in drei Phasen ein:

1. Erkennung („*detection*“) der Funktionalität der Benutzungsschnittstellen.
2. Präsentation („*representation*“) und Dokumentation derselben.
3. Abbildung („*transformation*“) in die neue Umgebung.

In der Erkennungsphase werden das System analysiert und die Bestandteile erkannt, die für die Benutzeroberfläche zuständig sind. Dies kann per Hand geschehen oder durch Mustererkennung und syntakti-

sche/semantische Analyse. Dabei bezieht sich Moore nur auf die Benutzungsschnittstellen, die im Quelltext integriert vorliegen. Die Analyse von Oberflächenbeschreibungssprachen gehört daher auch in die Erkennungsphase, d.h. die Steuerelemente sowie deren Anordnung und Beziehungen zueinander werden analysiert und für die Präsentationsphase vorbereitet.

Mit der Präsentation werden die gewonnenen Erkenntnisse in eine bestimmte Darstellung gebracht. Diese können im Falle von Oberflächen abstrakte Beschreibungssprachen („*Abstract User Interface Design*“), Zustandsautomaten („*Since most user interfaces involve system states and transitions […]*“) oder auch andere Modellformen (z.B. Objekte, Graphen, XML usw.) sein.

Die dritte und letzte Phase der Migration ist die Abbildung der Benutzungsschnittstellen auf die neue Plattform. Diese Phase gehört nicht zum Reverse Engineering (vgl. Kapitel 2.5.2), sondern baut stattdessen darauf auf und nutzt die daraus gewonnen Erkenntnisse für die Migration. Dabei werden die zurückentwickelten Elemente der Oberflächen auf die neue Plattform abgebildet und gegebenenfalls an die neue Umgebung angepasst. So können Auswahlelemente als Listenfeld oder Dropdown-Listenfeld (vgl. Tabelle 17) platziert werden, je nachdem, ob ein Dropdown-Listenfeld von der Zielplattform unterstützt wird oder nicht. Zudem kann das Dropdown-Listenfeld vorgezogen werden, wenn der Platz für die Oberfläche knapp ist und eine Liste zu viel Raum beanspruchen würde. Da dieser Vorgang sich auf vorhandene Abbildungsvorschriften stützt, wird er dementsprechend als wissens- oder regelbasierte Transformation (vgl. [Moore, 1995]) bezeichnet.

Die Probleme beim Migrieren können vielfältig sein. [Moore, et al., 1993] zählen dazu die Unterschiede in den Funktionen und den Architekturen der verschiedenen Plattformen sowie die Integration der Benutzeroberfläche in der Anwendung. Zudem sehen die Autoren im Fehlen von Standards bei Benutzeroberflächen eine Hürde bei der Migration von Anwendungen. Ihre Empfehlung lautet daher den Migrationsprozess methodisch durchzuführen und in Phasen (wie bereits erwähnt: Erkennung, Repräsentation und Transformation) einzuteilen. Diese Vorgehensweise wurde auch in zwei Studien (dem *TRANSOPEN* Projekt und der Studie *Knowledge Worker Platform Analysis*) angewendet und überprüft [Moore, et al., 1993].

Die Migration hängt offensichtlich stark mit dem Reverse und Software Engineering zusammen. Eine vorhandene Anwendung muss zuerst verstanden sein, bevor sie für eine neue Plattform erstellt werden kann. Einige Ansätze zur Migration setzen daher auf plattformunabhängige Modelle, die Anwendung und deren Interaktion mit dem Benutzer beschreiben und aus denen Oberflächen generiert werden können (Eine Menge von GUI Generatoren wurden in [Schlegel, et al., 2010] evaluiert). Diese Ansätze nutzen dazu üblicherweise eine geräteunabhängige Beschreibungssprache wie beispielsweise [USIXML], UIML (siehe [Abrams, et al., 2004]) oder XIML (siehe [Di Santo, et al., 2004]). Diesen Ansätzen ist allen gemein, dass die Migration zur Entwurfszeit passiert. Dagegen erlauben andere Ansätze auch die Migration zur Laufzeit. Wie beispielsweise [Bandelloni, et al., 2007] oder von [Bandelloni, et al., 2004] , die einen modellgestützten Ansatz beschreiben, der zur Laufzeit Webseiten für verschiedene Plattformen wie Mobilgeräte oder auch als Sprachausgabe generieren kann.

Mit dem Aufkommen immer neuerer Mobilgeräte und Bildschirmauflösungen kam auch die Notwendigkeit auf, vorhandene Anwendungen auf die neuen Plattformen zu migrieren. Viele Ansätze befassen sich daher mit der Migration von grafischen Oberflächen auf die Bedürfnisse von Geräten mit kleiner Anzeigefläche (z.B. Handhelds, Smartphone, Tablets). Darunter sind auch die bereits erwähnten Arbeiten von Bandelloni et al. mit einer weiteren Arbeit *Flexible Interface Migration* von denselben Autoren. Weitere Beiträge stammen von [Grolaux, 2004], [Canfora, et al., 2004] und [Wong, et al., 2002]. Letztere präsentieren ein Framework für Java, welche „Scalable Graphical User Interfaces“ nutzt, um Oberflächen zwischen verschiedenen Plattformen (hauptsächlich Mobilgeräte) zu migrieren. Der Ansatz nutzt eine eigene Layoutspezifikation für jede Plattform und eine Menge von Regeln für die Migration von Steuerelementen. Zur Entwurfszeit wird ein Dialog entwickelt, der auf verschiedene Plattformen migriert werden kann.

Im Gegensatz zu den bereits oben genannten Ansätzen gehen [Salminen, et al., 2007] einen gegensätzlichen Weg und beschreiben eine Lösung auf der Basis einer zwischengeschalteten Software (Middleware), die Diens-

te für Verbindungsaufbau, Suche und Kommunikation für Mobilgeräte auf Desktopcomputern ermöglicht. Die Autoren erstellen den Softwareprototyp, der die Kommunikation zwischen Mobilanwendung und migrierter Desktopanwendung ermöglicht und evaluieren ihn, indem sie Benutzern die Möglichkeit geben SMS auf einer Computertastatur schreiben und sofort senden zu lassen. Eine weitere Lösung, die eine Middleware-Lösung nutzt, um grafische Benutzerschnittstellen auf verschiedenen Geräten nutzen zu können, ist unter [Paternò, et al., 2008] zu finden. Die darin vorgestellte Migration unterstützt sogar die Darstellung von Benutzerschnittstellen im Fernseher, wie in der Abbildung 24 zu sehen ist.

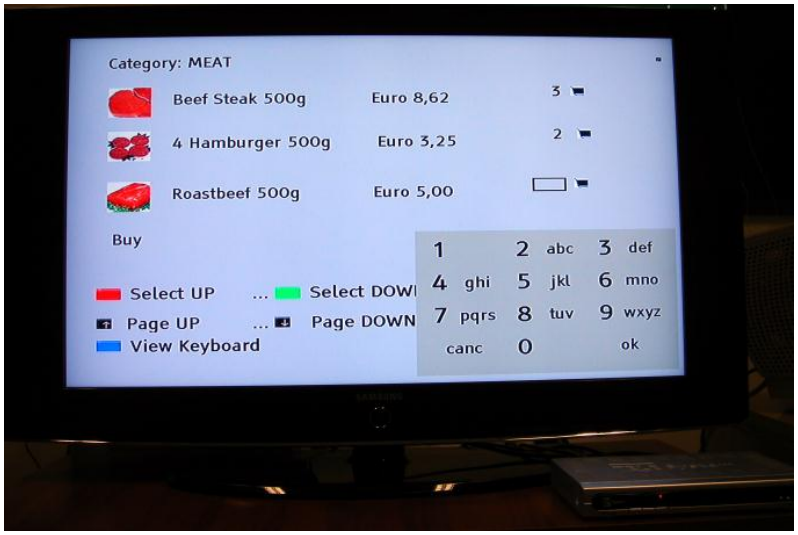


Abbildung 24 Eine GUI migriert für ein Fernsehgerät [Paternò, et al., 2008]. Die Elemente können durch die Fernbedienung bedient werden.

Die Migration kann besonders aufwändig sein, wenn eine Anwendung auf ein anderes Betriebssystem umgerüstet werden soll. Für die Migration von UNIX Anwendungen auf Windows gibt es dazu von Microsoft eine 600 Seiten starke Anleitung (siehe [Microsoft, 2003]). Weitere Strategien sowie Ansätze und Lösungen für die Migration von Windows Anwendungen gibt es unter [Gerdes, 2009], [Abramson, et al., 2002] und [Antoniol, et al., 1995].

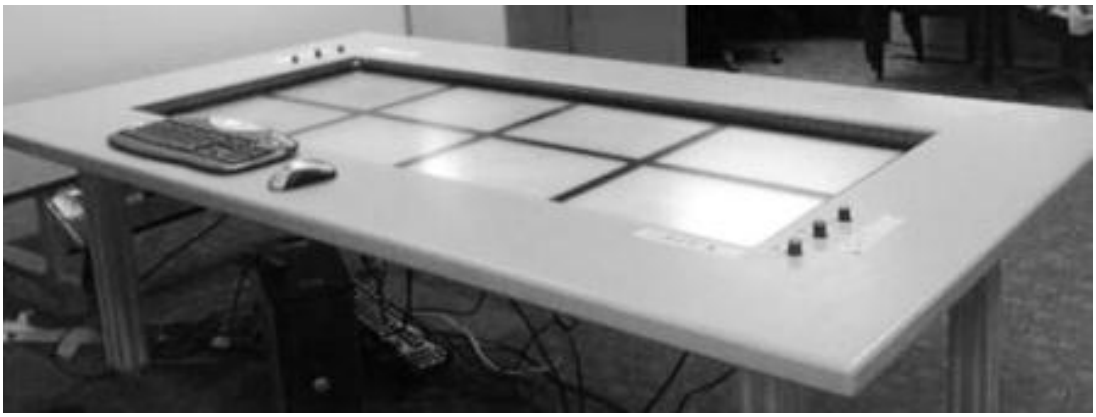


Abbildung 25 Die Migration von AgilePlanner auf ein Multi-User „Touchtisch“ mit 10 Mega Pixel Auflösung [Wang, et al., 2008]

Weiterhin sollen Anwendungen für neue, populär werdende Technologie, wie die berührungsempfindlichen Bildschirme, angepasst werden. An der Universität Calgary in Kanada wurde bereits eine Anwendung *Agile-Planner* für agile Projektsitzungen entwickelt, um den Teilnehmern ihre Projektplanung zu erleichtern. Die Sitzungsteilnehmer nutzten *AgilePlanner* bereits auf ihren portablen Computern, ohne jedoch einen Vorteil bei der Zusammenarbeit während eines Meetings zu erhalten, da jeder Teilnehmer nur den eigenen Bildschirm betrachtete. Deshalb wurde die Software von [Wang, et al., 2008] auf einen 183cm x 122 cm großen Multi-

User „Touchtisch“, ein sogenanntes Tabletop, migriert. Der Tisch erlaubt die Nutzung der Software von mehreren Benutzern gleichzeitig und insbesondere gemeinsam.

Die Entwickler berichten über ihre Erfahrungen bei der Migration der Anwendung und der Umsetzung von Fingereingaben, Gesten und multiplen Simultaneingaben von mehreren Benutzern am Tisch. Aus ihren Evaluationsergebnissen der migrierten Anwendung leiten sie u.a. die folgenden Richtlinien ab:

1. Komponenten der Oberfläche sollten beweglich und drehbar sein
2. Fingergesten sollten gegenüber traditionellen Menüs vorgezogen werden
3. Texteingaben sollten über Handschriftenerkennung geschehen, statt eine Tastatur zu verwenden
4. Die Größe von Steuerelemente sollte für Fingereingabe angepasst werden

2.7 Zusammenfassung

Das Grundlagenkapitel befasste sich mit dem aktuellen Wissensstand bei der Oberflächenentwicklung. Zuerst wurden wichtige Begriffe für diese Arbeit definiert, u.a. die Begriffe Benutzbarkeit und Touch. Daraufhin wurde die Geschichte der Benutzerschnittstellen vorgestellt und gezeigt, was in Zukunft mit organischen Benutzerschnittstellen möglich sein wird. Es folgte ein Kapitel über berührungsempfindliche Technologien und Methoden. Dazu wurden der aktuelle Stand der Technik für berührungsempfindliche Bildschirme präsentiert und die einzelnen Techniken wie druckempfindliche, ladungsempfindliche und optische Oberflächen erklärt sowie die Vor- und Nachteile dargelegt. Weiterhin wurde auf die Interaktionsmethoden bei einer „Toucheingabe“ eingegangen und veranschaulicht wie Finger für verschiedene Bedienaktionen eingesetzt werden können.

Das nächste Kapitel behandelte Dialoge und welchen Zweck – bekanntlich zur Eingabe und Abfrage von Daten sowie zur Bestätigung derselben – diese erfüllen. Es wurden außerdem sieben Gestaltungsgrundsätze aus der ISO Norm für die Entwicklung von Dialogen genannt, welche die Bedienbarkeit von Dialogen verbessern sollen. Neben den Gestaltungsgrundsätzen wurden außerdem fünf mögliche Arten von Bedienaktionen gezeigt, die ein Benutzer in einem grafischen Dialogsystem ausführen kann. Ein weiteres Kapitel über Dialoge führte anschließend in die Beschreibungssprachen von Dialogen ein. Dazu wurde zwischen zwei Arten von Benutzungsschnittstellen unterschieden: den plattformabhängigen und –unabhängigen Dialogbeschreibungssprachen. Als Beispiele kamen ConcurTaskTrees als plattformunabhängige Dialogbeschreibung sowie Programmiersprachen und XAML als plattformabhängige Sprachen zum Einsatz. Das Kapitel über Dialoge wurde schließt mit einem Exkurs über die Punktdichte, und welche wichtige Rolle diese bei der Darstellung von Dialogen auf Bildschirmen spielt, abgeschlossen.

Im Anschluss wurde in die Softwareentwicklung, insbesondere die modellgetriebene Entwicklung und das Reverse Engineering, eingeführt. Die modellgetriebene Entwicklung wurde erläutert und gezeigt welche Möglichkeiten sie für die gleichzeitige Entwicklung auf verschiedenen Plattformen bietet. Als Beispiele wurden MARIA als Modellsprache für interaktive Anwendungen sowie das Framework AndroMDA vorgestellt. Überdies wurde das Thema Reverse Engineering vorgestellt und Konzepte und Lösungen vorgebracht, um bereits bestehende Dialogoberflächen zu analysieren und zurückzuentwickeln, d.h. die Inhalte und Darstellung der Dialoginhalte aus dem Programmquelltext oder einer Dialogbeschreibungssprache zu erlangen.

Zum Schluss wurde die Migration von Dialogsystemen behandelt. Verschiedene Lösungen wurden dazu vorgestellt, die Dialogoberflächen für unterschiedliche Plattformen umrüsten können. Außerdem wurde ein Artikel vorgestellt, bei dem die Autoren über ihre Erfahrungen mit der Migration einer bestehenden Desktop-Anwendung zur Konferenzverwaltung auf einen großen Multi-Touch-Tisch berichteten.

3 Problem-, Aufgabenstellung und Lösungsansatz

Wer aufhört, besser sein zu wollen, hat aufgehört, gut zu sein.

Oliver Cromwell
englischer Militärführer und Politiker

Das folgende Kapitel 3.1 gibt einen Überblick über die Aufgaben und Ziele dieser Diplomarbeit. Zuerst wird die aktuelle Situation beschrieben und welche Nachteile dies mit sich bringt. Daraus ergeben sich die Aufgaben dieser Diplomarbeit. Den Problemen und Aufgaben folgt zuletzt ein Überblick des gewählten Lösungsansatzes in Kapitel 3.2.

3.1 Problemstellung

Mit dem Erscheinen von Smartphones und Tablets PC ist es mittlerweile üblich geworden, Mobilgeräte mit den Fingern zu bedienen. Aber auch für Desktopcomputer existieren schon lange berührungsempfindliche Bildschirme, für die es häufig jedoch an geeigneter Software mangelt. Um welche Probleme es sich dabei handelt (Fehlende Touch- und Entwickler-Unterstützung), soll in den folgenden Abschnitten näher erläutert werden.

Fehlende Touch-Unterstützung

Man muss zugegeben, dass bei Desktopsystemen keine übermäßig hohe Priorität für den Einsatz von berührungsempfindlicher Bedienung besteht. Einerseits sind Touchbildschirme nicht weit verbreitet, obwohl sie mittlerweile kaum noch im Preis von normalen Flachbildschirmen zu unterscheiden sind. Und andererseits sind die Nutzer an die Bedienung mit Maus und Tastatur gewohnt. Nur in speziellen Einsatzgebieten, wie Fahrkartenautomaten überwiegt die Bedienung mit den Fingern, nicht zuletzt weil die Betreiber die Instandhaltungskosten sparen wollen.

Diese Systeme sind jedoch geschlossen, d.h. nur von einem Anbieter. Zudem wird die Software dazu direkt für die eingesetzte Hardware entwickelt und darauf optimiert. Im Gegensatz dazu sind Anwendungen für Desktopcomputer nicht für ein spezielles System entwickelt worden. Sie sollen, wenn möglich, jahrelang funktionieren. Viele Anwendungen, insbesondere aus dem Bankensektor, stammen daher aus dem letzten Jahrtausend und sind damit mehrere Jahrzehnte alt. Aber auch Anwendungen, die immer wieder in neue Versionen entwickelt wurden, sind nicht für Touch ausgelegt, da oftmals die Funktionalität im Vordergrund steht und weniger die Benutzbarkeit (obwohl wir hier auch Verbesserungen sehen können). Diese lange etablierten Anwendungen für Desktopsysteme unterscheiden sich grundlegend von den Anwendungen, welche für die relative neue Mobilgerätegeneration von iPhone, iPad und Co entwickelt werden (Neudeutsch: „App“). Die Bedienung mit den Fingern ist für den Benutzer standardmäßig die einzige Möglichkeit zur Interaktion mit dem Mobilgerät. Entsprechend hoch sind auch die Anforderungen für mobile Anwendungen (oder „App“), denn wenn eine solche Anwendung nicht gut genug mit den Fingern bedient werden kann, dann wird sie einfach nicht genutzt.

Entsprechend groß sind auch die Bestrebungen, den Entwicklern bei der Anwendungsentwicklung unter die Arme zu greifen, indem Touch bereits von Anfang an von den Entwicklungswerkzeugen unterstützt wird. Mit Windows Vista und Windows 7 ist mittlerweile auch für Windowsanwendungen die Fingerbedienung möglich. Windows selbst bietet jedoch standardmäßig nur grundlegende Touchfähigkeiten an. Es unterscheidet daher zwischen Altanwendungen, die selbst nicht auf Berührungen reagieren können und Neuanwendungen, die aktiv auf die Fingereingabe antworten, d.h. dafür entwickelt wurden. Die alten Anwendungen werden mit den üblichen Interaktionen versorgt, indem ihnen vorgespielt wird sie würden mit einer Maus bedient. Dazu zählen

z.B. das Klicken und der Bildlauf mit ein oder zwei Fingern. Windows bietet jedoch keine Möglichkeiten an, fremde Anwendungen gezielt einfacher und präziser mit den Fingern zu bedienen, zum Beispiel durch vergrößerte Bedienelemente. Daher besteht in diesem Fall die einzige Möglichkeit darin, die Punktdichte des gesamten Systems (und damit aller Anwendungen) zu erhöhen und mit den Nachteilen zu leben, welche durch Anwendung entstehen, die damit nicht umgehen können (vgl. Kapitel 2.4.5).

Fehlende Entwickler-Unterstützung

Die meisten Anwendungsentwickler werden sich kaum um die Fingerbedienung ihrer Anwendungen kümmern, wenn sie damit keine oder nur wenig Erfahrungen gemacht haben oder sie keinen berührungsempfindlichen Bildschirm besitzen. Mit Einführung von Windows Vista gibt es zwar auch für die Entwicklerseite ein Entwicklungswerkzeug, welches Touchinteraktionen für Anwendungen ermöglicht, jedoch erfordert dies immer einen Anpassungsaufwand für die Entwickler.

Denn eine beträchtliche Anzahl von Dialogen wurde bereits entwickelt und befindet sich damit im Einsatz. Selbst in einer Firma, in der jahrelang Software entwickelt wurde, wäre ein einzelner Entwickler wohl überfordert mit der Anpassung der Dialoge für die Touchbedienung. Allerdings ist es fraglich, ob es für kleine Firmen oder selbstständige Entwickler wirtschaftlich sinnvoll ist, eine Arbeitskraft alleine mit der Anpassung zu betrauen. Und sogar für Firmen, die mehrere Arbeitskräfte einsetzen könnten, um ihre Dialoge entsprechend anzupassen, würden mit Probleme konfrontiert. Zuerst wäre eine Art von Regelwerk erforderlich, das die Entwickler anleitet, wie die Anpassung zu erfolgen hat. Jedoch ist kein Werk beliebig präzise, so dass darin immer Lücken enthalten sind. Diese Lücken werden daher vom Entwickler selbst geschlossen, indem er seine eigene Erfahrung nutzt. Dies bedeutet allerdings, dass die Anpassung der Dialoge zwischen verschiedenen Entwicklern (und besonders bei unterschiedlichen Kulturen, z.B. aus Europa und Asien) stark variieren kann und dadurch die Bedienkonsistenz über alle angepassten Dialoge leidet. Und selbst wenn es keine Lücken im Regelwerk gäbe, kann die Konsistenz leiden, wenn der Anpassungsaufwand nur hoch genug ist. So kann die Qualität der angepassten Dialoge sich verändern, indem die Entwickler während ihrer Arbeit dazulernen. Fehlt zu Beginn der Arbeit die Erfahrung mit dem Regelwerk und Anpassungsprozess, so ändert sich dies bis zum Schluss.

Ein weiteres Problem betrifft die eingesetzten Steuerelemente selbst. Denn nur einige sind für die Fingerbedienung geeignet und auch dann müssen sie richtig eingesetzt werden. Während Schalterknöpfe (Buttons) für Touch bestens geeignet sein können, sind sie in sogenannten Drehfeldern (siehe Tabelle 17 im Anhang) durch ihr geringes Maße unbrauchbar. Die Entwickler können ohne ausreichend Erfahrung und Wissen nicht erkennen, welche Elemente wie eingesetzt werden, um gut bedienbar zu sein (Andernfalls wäre die Benutzbarkeit kein so großes Forschungsfeld in der Wissenschaft und Industrie). Ein Entwickler muss bei der Anpassung außerdem immer wieder viele gleichartige problematische Stellen innerhalb des Dialogs erkennen und anpassen. Er muss die Maße der Elemente korrigieren, die Abstände vergrößern oder Steuerelemente austauschen. Diese monotone Arbeit kann daher zu Fehlern führen, die nachträglich korrigiert werden müssen.

3.2 Aufgabenstellung und Lösungsansatz

Die folgenden Abschnitte beschreiben die Aufgaben der Diplomarbeit sowie den Lösungsansatz.

Aufgabenstellung

Um die aktuelle Situation und die erläuterten Probleme bei vorhandenen Desktopanwendungen zu verbessern, wurden die folgenden Aufgaben für diese Diplomarbeit konzipiert (siehe auch Abbildung 26).

1. Recherche zum Thema, zu Beschreibungsmethoden, Frameworks und Transformatoren für die Anpassung der Dialoge (Kapitel 2).
2. Recherche und Konzeptbildung zu Interaktionselementen und Gestaltungsregeln für die Fingerbedienung (Kapitel 2 und 5).
3. Konzeption eines regel-/modellbasierten Ansatzes für die Transformation von klassischen Dialogen in Touchanwendungen (Kapitel 4).
4. Erstellung einer Architektur und eines in die Implementierung überführbaren Konzepts (Kapitel 6).
5. Erstellung einer Implementierung auf der Basis der Windows Presentation Foundation und für XAML nutzbaren Transformationsanwendung (Kapitel 7).
6. Durchführung einer Nutzerstudie, in der entsprechende Bedienelemente und Regeln evaluiert werden, um eine Basis für die Gestaltungsregeln und die Transformation zu erhalten (Kapitel 7.6).
7. Evaluation der Anwendung anhand der Erkenntnisse aus der Studie und mit Hilfe der Implementierung anhand von realistischen Beispielen für Dialoganwendungen (Kapitel 7.6).

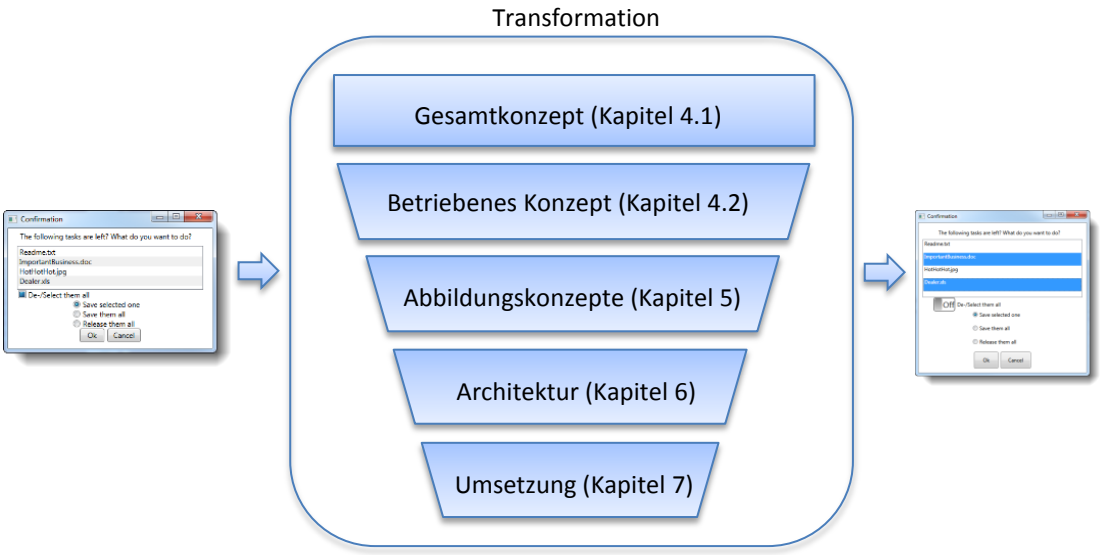


Abbildung 26 Schematische Darstellung der Lösung. Die Transformation von Dialogen besteht aus einem Gesamtkonzept, welches im Laufe dieser Arbeit konkretisiert und schließlich umgesetzt wird.

Lösungsansatz

Den Aufgaben und deren Bearbeitung ging zuerst eine Idee voraus, Dialogelemente mit der Hilfe von Regeln so zu verändern, dass deren Bedienung auf berührungsempfindlichen Bildschirmen für Benutzer leichter fällt. Die Idee wurde erweitert und es kam ein modellgetriebener Ansatz dazu, der sich durch die Möglichkeiten in der modellgetriebenen Entwicklung inspirieren ließ. Denn die Idee bestand auch darin jede Art von Dialogbeschreibungssprache transformieren zu können. Dies hätte jedoch bedeutet, die Transformationsregeln auf jede die-

ser Sprachen anwenden zu müssen. Doch der Ansatz sollte eine einheitliche Transformationssprache für mehrere Dialogsprachen sein, um den Wartungs- und Portieraufwand zu minimieren. Das Konzept bestand also darin erst einmal verschiedene Dialogsprachen zu vereinheitlichen, um die Regeln auf dieser einheitlichen Sprache anwenden zu können. Zum Schluss würde die Einheitssprache dann wieder in die ursprüngliche Dialogsprache zurückkonvertiert.

Wie es sich herausstellte war das ursprüngliche Konzept bereits zu groß, um vollständig umgesetzt werden zu können. Auf Grund dieses Umstandes wurde der zentrale und wichtige Teil, die Abbildung der Dialogelemente in einer einheitlichen Sprache, in dieser Arbeit angegangen. Das Gesamtkonzept ist daher als Ansatz beschrieben, der für weitere Entwicklungen als Grundlage dienen kann.

Weiterhin war es für die Abbildung von Steuerelementen erforderlich zu wissen, wie diese Abbildungen überhaupt aussehen könnten. Daher wurden Konzepte entwickelt, welche die Steuerelemente für die Bedienung mit Fingern verbessern sollten. Diese konzeptionellen Regeln bilden damit die Grundlage für die Abbildungsregeln des Transformationsprozesses und können mit Hilfe der entwickelten Anwendung für Dialoge eingesetzt werden.

4 Methode zur Umsetzung des Lösungsansatzes

Software is a thought process.
To patent it is comparable to patenting induction or deduction.

Tom DeMarco
amerikanischer Softwareentwickler

Im vorherigen Kapitel wurde bereits die Aufgabenstellung genannt, die aus der Problemstellung hergeleitet wurde. Dazu wurde außerdem ein Lösungsansatz entwickelt, um die Umsetzung zu ermöglichen. Das folgende Kapitel konkretisiert diesen Lösungsansatz und stellt eine grundlegende Vorgehensweise für die Umsetzung bereit. Es beginnt daher mit der Beschreibung eines großen Konzepts in Kapitel 4.1 und erläutert die möglichen Wege und Methoden zur Transformation einer Oberfläche. Für die Umsetzung selbst wird in Kapitel 4.2 ein Teil dieses Konzepts herausgenommen und näher erläutert, um eine umsetzbare Methode für die Kapitel 6 und 7 zu erhalten.

4.1 Gesamtkonzept

Die Herausforderung, Altanwendungen auch für berührungsempfindliche Bildschirme anzupassen, macht es erst einmal erforderlich ein Grundkonzept, d.h. eine Vorgehensweise zu entwickeln, die hier vorgestellt werden soll. Eine solche Methode gibt die Abbildung 27 wieder.

Die Abbildung zeigt eine modellgetriebene Entwicklung aufgeteilt in zwei Seiten. Die linke Seite stellt die Dialoganwendung ohne Unterstützung und Anpassung für berührungsempfindliche Bildschirme dar (Legacy Seite), während die rechte Seite diese Anpassungen bereits besitzt (Touch Seite). Die Aufgabe besteht nun darin dialogbasierte Altanwendungen (linke Seite) über eine Transformation für die Benutzung auf berührungsempfindlichen Bildschirmen (rechte Seite) anzupassen.

Für die meisten Dialoganwendungen wird entweder eine Dialogbeschreibungssprache verwendet oder sie sind durch ein mehr oder weniger abstraktes Modell spezifiziert. Die Vielzahl von Dialogbeschreibungssprachen und Dialogmodellen macht die Aufgabe der Migration nicht einfach. Daher besteht die Lösung darin, nicht die Dialogsprachen selbst zu migrieren, sondern einen Dialog erst einmal in ein einfacher zu migrierendes Modell zurückzuentwickeln (Reverse Engineering, Symbol ① in der Abbildung). Der Reverse Engineering Schritt muss für verschiedene Arten von Beschreibungssprachen einmalig unternommen werden. Danach kann die Migration (③) jedoch stets auf demselben Modell stattfinden. In dieser Arbeit soll das XAML Modell verwendet werden, da es auf XML aufsetzt und daher leicht automatisch zu interpretieren und anzupassen ist. Der Weg über XAML ist aber nicht der einzige. Die modellgetriebene Softwareentwicklung (MDE) nutzt Modelle für Dialoge und Anwendungen, die noch unabhängig von der eingesetzten Plattform sind. Ein abstraktes Dialogmodell in XML Notation, wie es UsiXML (siehe [USIXML]) implementiert, kann zum Einsatz kommen, wenn kein .NET, C# oder XAML möglich ist wie z.B. bei Webseiten mit HTML. Es ist daher genauso möglich Abbildungsregeln (②) auf der Modellebene der Stufe 3 (PIM) und höher einzusetzen. Wie im MDE üblich, durchläuft die Codegenerierung nicht alle untergeordneten Stufen, sondern kann sofort die notwendige Zielsprache erzeugen. D.h. XAML ist nicht für die Transformation mit höheren Modellen notwendig (①).

Die Abbildung der Dialogelemente in eine Modellsprache und wieder zurück (vertikale Transformation, ① und ④) wurde in dieser Arbeit nicht untersucht, weil die Abbildung von Alt- auf Neudialogen (horizontale Transformation, ③) im Blickpunkt stehen sollte. Ansätze und Umsetzungen für die Konvertierung von Dialogsprachen in Modelle und zurück wurden jedoch bereits in den Kapiteln 2.5.1 und 2.5.2 genannt.

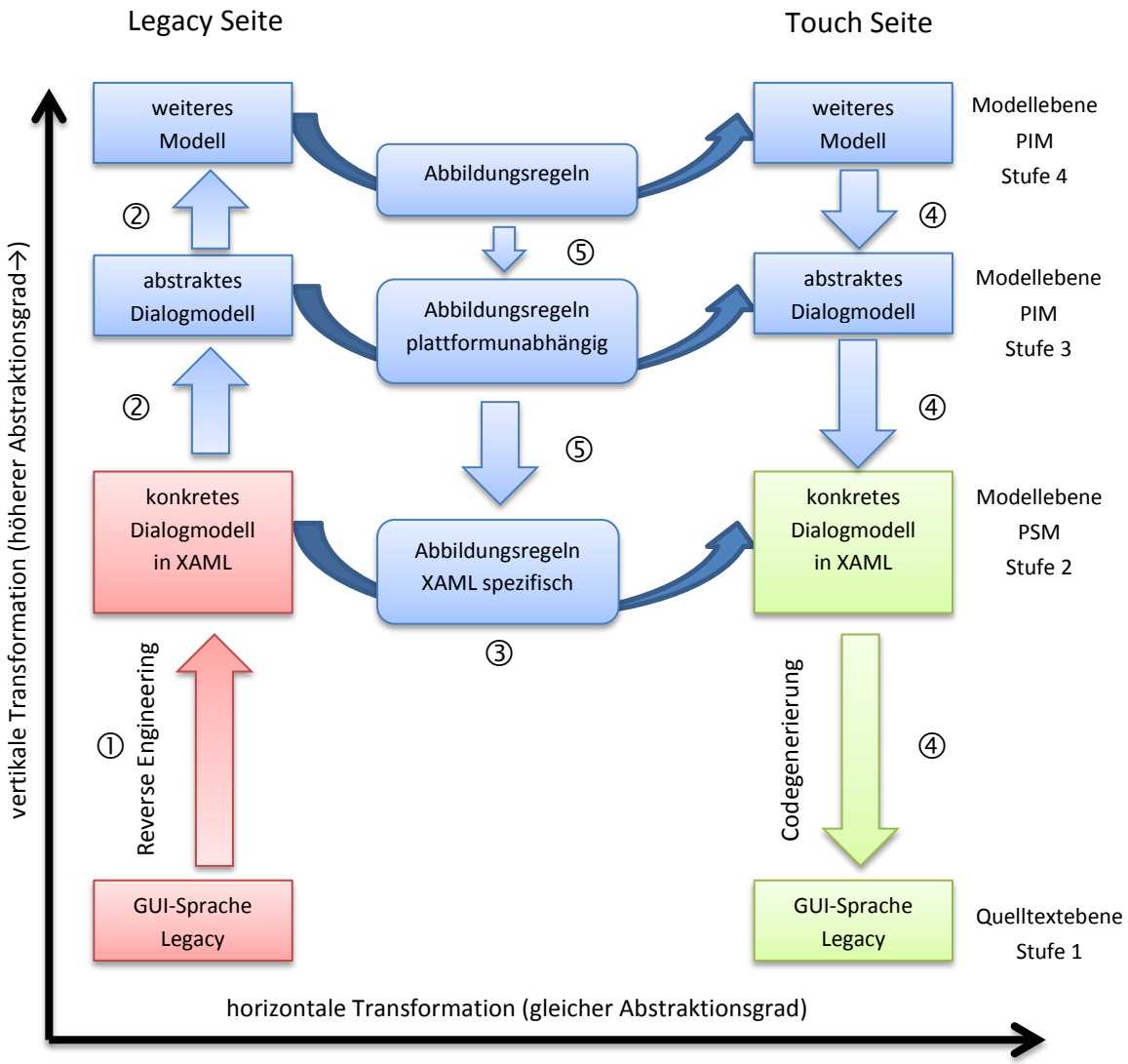


Abbildung 27 Modellgetriebene Transformation zur Migration auf berührungsempfindliche Dialoge im allgemeinen Fall bestehend aus Reverse Engineering, Transformation (auch Abbildung) und Codegenerierung

Wird XAML als Zielmodell gewählt, hat dies jedoch drei Vorteile:

1. XAML ist erstens anpassbar durch völlig neuartige und selbst erstellte Steuerelemente und kann dadurch sehr aufwändig gestaltete Oberflächen produzieren.
2. XAML beruht auf dem XML Standard und ist somit mit XML DOM Parsern nutzbar.
3. Die XAML Dialoge können in einem Designer wie Visual Studio betrachtet und bearbeitet werden, um beispielsweise ihr Aussehen zu optimieren.

Ob nun XAML oder ein anderes Modell verwendet wird, der Abbildungsprozess (③) bleibt der gleiche. Kann das Modell jedoch mit XML beschrieben werden, ist letztlich nur ein XML Parser notwendig und zudem das „Wissen“ wie die Steuerelemente migriert werden können. Das Lesen von XML gestaltet sich mit Hilfe von XQuery und XPath außerdem sehr einfach und flexibel. Und viele Programmiersprachen unterstützen XML standardmäßig durch eigene Klassen und Methoden. Zur Transformation von XML hat sich mit der Sprache XSLT (Extensible Stylesheet Language Transformation) ein weiterer Standard gebildet. XSLT ist eine Turing-vollständige Sprache, die in dieser Arbeit zur Migration von Dialogen in XAML verwendet wird (Eine in XSL gebaute Turingmaschine ist unter [Unidex Inc., 2001] zu finden).

Ein letzter Lösungsansatz betrifft die Abbildungsregeln selbst. Abbildung 27 zeigt Regeln auf verschiedenen Modelebenen, die eine vertikale Transformation (⑤) durchlaufen. Die Idee dahinter ist eine einfache Sprache für den Benutzer zu haben, in der sie/er die Abbildungsregeln beschreibt. Die Regeln werden schließlich in einem vertikalen Transformationsprozess auf die gewünschte Modellstufe (z.B. Stufe 2 mit XSLT) abgebildet, so dass beispielsweise der Benutzer sich nicht direkt mit XSLT, XPATH (von XSLT verwendet) und den Eigenschaften von XAML auseinander setzen muss. Dazu müssen die Abbildungsregeln in einer einfach gestalteten Sprache spezifiziert werden. Dies kann z.B. eine Untermenge der natürlichen Sprache sein, die unabhängig von der eingesetzten Plattform und Dialogbeschreibungssprache die Abbildungsregeln bestimmt. Das folgende Beispiel einer einfachen Abbildungsvorschrift bildet die Steuerelemente eines Dialogs ab:

> Set *size* of all elements where type is Button to *minimum 1 centimeter*.
> Replace all elements where type contains ListBox with element ListBoxTouch.

Das Beispiel erinnert entfernt an SQL Anweisungen bei Datenbanksystemen. SQL wird seit vielen Jahren erfolgreich eingesetzt und ist auch für Anfänger einfach zu lernen und zu verwenden. Doch in den Beispielen existiert noch eine gewisse Ungenauigkeit, die durch eine vertikale Transformation (⑤) entfernt werden muss. Einerseits ist die Eigenschaft *size* sicher nicht in allen Dialogsprachen vorhanden, sondern wird beispielsweise durch *width* und *height* ausgedrückt. Und andererseits benutzen die wenigsten Dialogsprachen metrische Längeneinheiten oder nennen ihre Listenelemente ListBox (vgl. Tabelle 18 *Abbildungstabelle für Steuerelemente von Delphi, Dialog Ressource und XAML* im Anhang). Vor der Transformation müssen daher die Abbildungsregeln von allen Ungenauigkeiten befreit werden, d.h. die korrekten Bezeichner und Größen müssen für die jeweils eingesetzte Dialogbeschreibungssprache konvertiert werden.

4.2 Umgesetzte Methode

Im Rahmen dieser Diplomarbeit wird der im Kern der Abbildung 27 sitzende Transformationsprozess bearbeitet (Ein Auszug ist in Abbildung 28). Darin wird die Migration von Dialogen für den Gebrauch auf berührungsempfindlichen Bildschirmen behandelt. Der Transformationsprozess wird in Kapitel 7.5 beschrieben und stellt nur einen Teil der Hauptaufgabe dar. Die andere Aufgabe besteht darin Abbildungen zu finden, die Dialoge für berührungsempfindliche Bildschirme verbessern.

Für den Transformationsprozess selbst wurde eine Anwendung in C# geschrieben, die einem Benutzer die Möglichkeit bietet einen Dialog, geschrieben in der Sprache XAML, zu transformieren. Es ist möglich, die meisten Dialoge bereits in der Anwendung zu betrachten und Änderungen am Transformationsprozess vorzunehmen. Die Transformation wird dabei über einen XSLT-Text-Editor beschrieben und gesteuert. Ein Plug-In-System ermöglicht während des Betriebs den Einsatz von externen in .NET geschriebenen Algorithmen. Die Anwendung und ihre Architektur werden in Kapitel 7.1 und 6 behandelt.

Abbildung 28 Ausschnitt des Transformationsprozesses aus Abbildung 27

Die Transformation von Dialogen wird mit der Hilfe von XSLT, der Extensible Stylesheet Language Transformationssprache, durchgeführt. Dazu werden XAML Elemente ersetzt oder entfernt bzw. neue Elemente hinzugefügt, indem in XSLT geschriebene Abbildungsregeln darauf angewendet werden. Zudem können auch Eigenschaften bzw. Attribute der Elemente in XSLT angepasst werden, um die Darstellung und das Verhalten des Elements zu beeinflussen. Parameter in XSLT werden außerdem die Transformation vor jeder Durchführung beeinflussen können, so dass Transformationen nicht statisch bleiben, sondern ohne größeren Aufwand für die eigene Bedürfnisse angepasst werden können.

5 Anpassung der Steuerelemente für berührungsempfindliche Eingaben

A common mistake that people make
when trying to design something completely foolproof
is to underestimate the ingenuity of complete fools.

Douglas Adams
britischer Schriftsteller

Dieses Kapitel behandelt die Abbildung von Steuerelementen für die Nutzung auf berührungsempfindlichen Bildschirmen. Zuerst werden die Eigenschaften von Steuerelementen für die berührungsempfindliche Eingabe in den Kapitel 5.2 und 5.3 analysiert. Darauf folgt eine Analyse der am meisten benutzten Steuerelemente wie Druckschalter, Kontrollkästchen, Textfelder, Baumansichten und Listenfelder in den Kapiteln 5.4 bis 5.7. Für diese Steuerelemente werden zudem eigene Ersatzelemente vorgestellt, die eine bessere Touchbedienung ermöglichen sollen.

5.1 Einführung

Ein Entwickler, der eine Anwendung berührungssensitiv auslegen oder sie dafür optimieren soll, wird sich die Frage stellen, wie seine Dialoge und deren Elemente gestaltet werden müssen, um einfach mit den Fingern bedient werden zu können. Neben den Größenanpassungen, Abständen und Interaktionskonzepten wie Gesten, könnte zudem die Frage aufkommen, ob für die eingesetzten Steuerelemente überhaupt eine verbesserte Version für die berührungsempfindliche Eingabe vorhanden ist. Daher werden im Folgenden in dieser Hinsicht verbesserte Steuerelemente vorgeschlagen. Jedoch sind die folgenden Empfehlungen für den berührungsempfindlichen Betrieb von Steuerelementen nur eine Auswahl von mehreren. Mit dieser Auswahl wird eine beispielhafte Umsetzung demonstriert. Einige der vorgeschlagenen Steuerelemente sollen zudem im Rahmen einer Studie (siehe Kapitel 8) untersucht werden.

5.2 Steuerelementgrößen

Die Fläche einzelner Steuerelemente hat einen entscheidenden Einfluss auf die Bedienbarkeit. Je größer die Trefferfläche ist, desto einfacher ist die Bedienung des Elements. Dies wird von verschiedenen Quellen bestätigt (siehe [Sears, et al., 1991], *Touch* in [Microsoft], [Parhi, et al., 2006], [Kwon, et al., 2009] und [Rozlog, 2009]). Die in Abbildung 29 unten dargestellten Schaltflächen verdeutlichen die Unterschiede zwischen verschiedenen Flächengrößen. Empfehlungen aus den verschiedenen Quellen nennen eine minimale Seitenlänge zwischen 10 und 20 Millimeter (38 und 75 Pixel bei 96 PPI). In der Studie aus Kapitel 8 wurde die Auswirkung verschiedener Größen innerhalb eines Dialogs untersucht.

Abbildung 29 Verschiedene Größen von Schaltern: 5, 7, 10, 15 und 20 Millimeter Seitenlänge. Die Größe hat einen Einfluss auf die Bedienung mit den Fingern.

5.3 Steuerelementabstände

Wie die Größe von Steuerelementen trägt ein Abstand zwischen Steuerelementen zur besseren Bedienbarkeit bei. Bei nahe beieinanderliegenden Steuerelementen besteht die Gefahr, dass der Benutzer ungewollt ein anderes Element aktiviert. Der Abstand soll mindestens 5 Pixel betragen [Microsoft]. Liegen die Steuerelemente zu weit auseinander besteht jedoch die Gefahr, dass der Benutzer diese nicht mehr als zusammengehörig erachtet und Bedienfehler auftreten (z.B. bei Optionsfeldern oder Kontrollkästchen).
Weiterhin können zu große Abstände zu einem aufgeblähten Dialog führen. Besonders bei vielen Steuerelementen innerhalb eines Dialoges führt dies zu einem hohen Platzverbrauch auf dem Bildschirm. Dies wiederum hat zur Folge, dass der Benutzer größere Strecken mit dem Finger zurücklegen muss. Daher wurden verschiedene Abstände von Steuerelementen in der Studie aus Kapitel 8 untersucht.

5.4 Kontrollkästchen

Anwendungsentwickler nutzen gerne Kontrollkästchen und Optionsfelder zur Konfiguration von Einstellungen. Die Größe und Ausrichtung dieser Elemente kann jedoch den Bedienkomfort auf berührungsempfindlichen Bildschirmen schmälern, wenn diese Elemente zu nahe beieinander liegen. Zudem lässt sich der aktuelle Zustand (an, aus) nicht erkennen, wenn der Finger das Steuerelement überdeckt. Daher sollten Kontrollkästchen und Optionsfelder so gestaltet werden, dass sie leicht zu treffen, jedoch nicht unbeabsichtigt aktiviert werden können. Für den Einsatz auf berührungsempfindlichen Bildschirmen bietet sich daher ein Ersatzelement an, das diese Aufgaben erfüllt. Abbildung 30 zeigt Ersatzelemente für ein Kontrollkästchen, die sich aufgrund ihrer Größe und Zustandsanzeige gut für die Fingerbedienung eignen können.

Abbildung 30 Mögliche Ersatzelemente von Kontrollkästchen. Links: Alte GUI-Elemente aus [Plaisant, et al., 1992]. Rechts: Beispielhafte Umsetzung mit WPF als Widgets. Die Bedienung von Kontrollkästchen wurde in der Studie aus Kapitel 8 untersucht.

5.5 Numerische Steuerelemente

Die berührungsempfindliche Eingabe mit den Fingern ist nur sehr bedingt geeignet für die Eingabe von Text- oder Zahlenwerten, denn dazu wird offensichtlich eine Tastatur benötigt. Mit Windows kommen zwar Bedienwerkzeuge wie virtuelle Tastatur oder Schriftenerkennung automatisch zum Einsatz, trotzdem erfordert diese Art der Bedienung zusätzliche Handgriffe (das Tastatursymbol) und Einarbeitungszeit. Besonders Felder für die Eingabe von Zahlen können einfach durch Steuerelemente ersetzt werden, die eine Änderung durch Klicken, Ziehen oder Schieben ermöglichen. Schiebregler erlauben diese Art von Bewegungen mit den Fingern durchzuführen, ohne dass eine Texterkennung notwendig wäre. Allerdings besitzen Schiebregler immer eine obere und untere Grenze, so dass nicht jede beliebige Zahl vom Benutzer eingeben werden kann. Der Entwickler muss dafür sorgen, dass der Schiebregler zumindest die am meisten verwendeten Werte ansteuern kann. Die restlichen, selten verwendeten Werte können dann direkt im Zahlenfeld über Tastatur oder Schrifterkennung eingegeben werden. Aus diesen Gründen wurde das folgende Zahlenelement entwickelt, welches aus einem Eingabefeld und einem Schiebregler besteht. Der Schiebregler kann durch Anklicken des Feldes dargestellt werden. Der Benutzer hat in diesem Moment die Wahl, ob er den Schiebregler bedient oder eine Tastatur darstellen lässt.

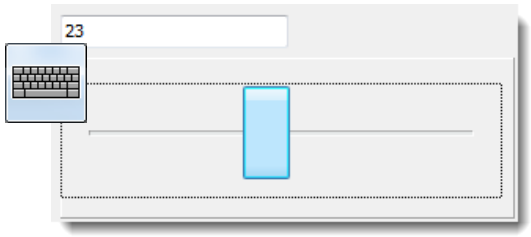


Abbildung 31 Beispielhafte Erweiterung eines Zahlenfeldes mit einem Schiebregler, der beim Antippen des Feldes aufspringt. Das Tastatursymbol wird von Windows automatisch eingeblendet.

Des Weiteren sollen auch die sogenannten Drehfelder (Siehe Tabelle 17) ersetzt werden. Drehfelder ermöglichen durch das Anklicken von Auf- und Abschaltern, den Wert zu erhöhen oder zu verringern. Diese Art von Zahleneingabe ist durch die geringe Größe der daneben platzierten Druckschalter jedoch schwer mit den Fingern zu bedienen. Zudem muss für jede Änderung des Wertes der Auf- oder Abschalter präzise angetippt werden. Denn ein Antippen und Halten funktioniert standardmäßig nicht, sondern das Drehfeld muss mit einem RepeatButton-Element (siehe MSDN) selbst hergestellt werden. Außerdem sind größere Werte mit den Fingern nicht bequem zu erreichen, da entweder oft der Schalter angetippt werden muss oder man lange auf die gewünschte Zahl beim Halten warten muss. Daher soll auch dieses Steuerelement für die Bedienung mit den Fingern angepasst werden.

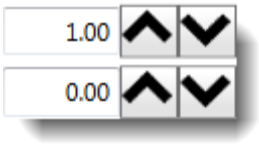


Abbildung 32 Drehfelder mit größeren Schaltflächen können einfacher mit dem Finger bedient werden

Für ein Drehfeld wurde dazu eine größere Variante gewählt, bei der die Schalter einfacher zu treffen sind (siehe Abbildung 32). Zudem wurde ein virtuelles Tastenfeld entwickelt, das die Eingabe einer Zahl über ein Druckschalter erlaubt. Insbesondere Zahlen mit Nachkommastellen sollen so besser eingegeben werden können. Beide Varianten wurden in der Studie in Kapitel 8 untersucht.

Abbildung 33 Ein Zahlenfeld für die präzise Eingabe von Kommazahlen, das sogenannte numerische Tastenfeld. Die Schaltfläche „Def“ setzt den Wert im Textfeld auf die ursprüngliche Eingabe zurück. „Clear“ belegt die Eingabe mit dem Wert 0. „X“ schließt die Eingabe ab. Die Eingabe wird abgebrochen, indem der Bereich außerhalb des Tastenfeldes berührt wird.

5.6 Baumansichten

Baumansichten oder Hierarchien sind ein beliebtes Mittel in Dialogen, um Werte zu gliedern. Ein bekanntes Beispiel ist die Dateiordneransicht im Öffnen- und Speichern-Dialog. Für die Nutzung mit dem Fingern besitzt das Element der Baumansicht jedoch mehrere Unzulänglichkeiten. Als erstes ist die Fläche der einzelnen Elemente vertikal häufig zu gering, um mit Fingern getroffen werden zu können. Zudem besitzen diese Elemente häufig kleine Symbole in Form eines Plus- oder Minuszeichens, welche die Möglichkeit bieten die untergeordneten Werte ein- oder auszublenden. Als Lösung könnten Baumansichten einfach vergrößert dargestellt werden, jedoch kann es dann leicht passieren, dass durch die Untergliederung der seitliche Platz nicht ausreicht. Untergeordnete Werte werden dadurch nicht mehr vollständig angezeigt, d.h. durch den Rand abgeschnitten. Der Benutzer muss deshalb zusätzlich seitlich scrollen, um die abgeschnittenen Texte lesen zu können.

Für Elemente, wie Baumansichten, kann daher der folgende Ersatz angewendet werden. Dazu stellt das Steuerelement in einer einfachen Liste immer nur die untergeordneten Werte des aktuell ausgewählten Wertes dar. Zum Beispiel werden in dem Steuerelement zuerst alle Ordner des Laufwerks C aufgelistet. Tippt der Benutzer einen angezeigten Ordner an, dann werden dessen Unterordner dargestellt. Damit der Benutzer jederzeit in eine übergeordnete Ebene wechseln kann, wird eine Brotkrumennavigation zusätzlich benötigt. Diese Baumansicht mit Brotkrumennavigation wurde in der durchgeführten Studie (Kapitel 8) eingesetzt und dabei deren Bedienbarkeit überprüft.

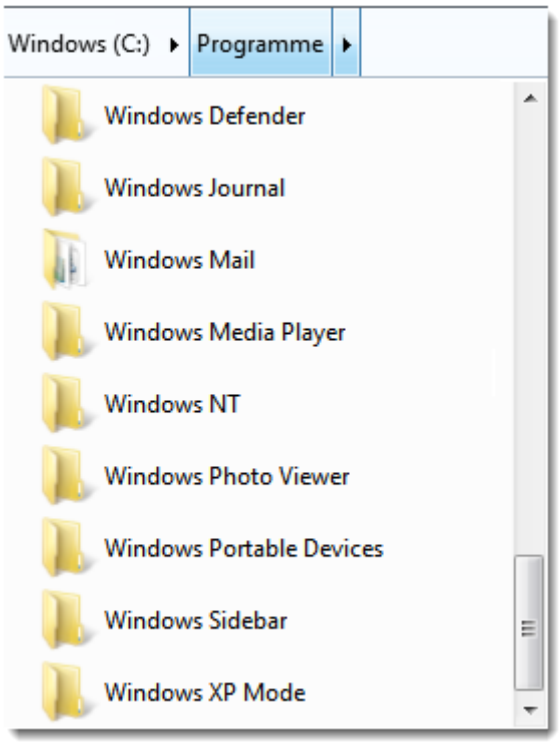


Abbildung 34 Eine illustrierte, flache Baumansicht mit Brotkrumennavigation erleichtert die Fingerbedienung

5.7 Listenfelder

Listenfelder oder Listenansichten (für den Unterschied siehe Tabelle 17) werden genutzt, eine Menge von Werten gleichzeitig darzustellen. Außerdem ermöglichen sie die Auswahl von mehreren Werten, indem die Werte markiert werden. Die Listenfelder stehen somit im Gegensatz zu Dropdown-Listenfeldern, die nur ein Element, nämlich das ausgewählte, ständig darstellen können.

Für die Fingerbedienung sind Listenfelder, genauso wie die Baumansichten, nur dann gut geeignet, sofern die Fläche der einzelnen Werte groß genug ist. Allerdings gilt auch hier: Je größer die angezeigten Werte werden, desto weniger Werte können im sichtbaren Bereich angezeigt werden. Besonders bei kleinen Listenfeldern wird so schnell erreicht, dass nur ein paar wenige Werte angezeigt werden können und die restlichen dadurch nur über einen ausgiebigen Bildlauf (oder neudeutsch Scrollen) erreicht werden. Allerdings kann der Bildlauf mit Gesten durchgeführt werden (siehe weiter unten). Die Nutzung von zusätzlichen Bildlaufleisten ist dadurch nicht notwendig, denn diese würden den bereits knappen Platz weiter verringern. Als Lösung kann entweder ein vernünftiger Bereich für die Größe der Werte gefunden werden (abhängig von der Listengröße) oder die Liste kann dynamisch vom Benutzer verkleinert und vergrößert werden. Letzteres bietet sich immer dann an, wenn der Dialog selbst in seiner Größe verändert werden kann.

Ein weiterer Lösungsansatz – der hier nur der Vollständigkeit genannt werden soll – sind Listenansichten, die ihre Werte auf eine spezielle Art und Weise darstellen und somit mehr Platz gewinnen. Sogenannte Trommellisten besitzen ständig einen ausgewählten Wert, der in der Mitte der Liste dargestellt wird. Es wird angenommen, dass Werte, die sich weiter weg befinden für den Benutzer weniger interessant sind. Solche Werte werden daher in Richtung der oberen und unteren Ränder kleiner dargestellt. Der dadurch erzeugte visuelle Effekt ähnelt einer Trommel auf deren länglichen Seite die Werte aufgedruckt sind.

Abbildung 35 Ein Trommellistenelement für das Android Betriebssystem. Die Werte oben und unten scheinen nach hinten gezogen zu werden.

Ein wichtiger Aspekt bei Listenansichten ist der Bildlauf. Wie bereits erwähnt, können Gesten einfach benutzt werden, um Bildlaufleisten zu vermeiden und das Mausrad zu simulieren. Allerdings sind Gesten wie Tastenkürzel oftmals nur mit Vorwissen zu verwenden und können ohne Übung frustrierend sein, wenn sie von der Anwendung nicht, wie vom Benutzer gewollt, erkannt werden. Aus dem Bereich der natürlichen Benutzerschnittstellen stammen die sogenannten kinetischen Gesten (Kinetik: Änderung der Bewegung). Sie werden bereits in Listenfeldern für Smartphones eingesetzt. Durch die Geschwindigkeit der Fingerbewegung innerhalb einer Liste werden die Werte entsprechend langsamer oder schneller geblättert. Dieser Vorgang ähnelt dem Drehen der oben vorgestellten Trommelliste. Das Scrollen der Liste mit den Fingern simuliert eine physikalische Beschleunigung sowie Reibung. Der Bildlauf einer Liste wird durch die wiederholte Einwirkung des Fingers zum Beschleunigen gebracht. Ohne weitere Einwirkung kommt der Bildlauf nach einer kurzen Verzögerung durch die Simulation von Reibung zum Stehen.

In Kioskanwendungen wie Bahnschaltern wird gewöhnlich auf Gesten verzichtet, da die eingesetzte Technologie – es sind meistens akustische Bildschirmoberflächen (siehe Seite 20) – dies nicht erlaubt. Stattdessen wird zu jeder Liste eine Reihe von Navigationsflächen angeboten. Die Art und Anzahl von möglichen Schaltern kann sich je nach Aufgabengebiet unterscheiden. Beispielsweise ist es sinnlos eine seitenweise Navigation zu erlauben, wenn die Werte nur eine Seite beanspruchen. Die Abbildung 36 zeigt eine Liste mit seitlich angeordneter Navigation, die sogenannte Navigationsleiste. Die mittleren Schalter (①) ermöglichen zum vorherigen bzw. nächsten Element zu springen. Die Schalter mit dem Symbol ② setzen den Auswahlbalken entweder um meh-

rere Elemente oder eine Seite weiter bzw. zurück. An das Ende bzw. den Anfang der Liste gelangt man mit den Schaltern③. Die Navigationsleiste wurde in der durchgeführten Studie (Kapitel 8) eingesetzt und untersucht.

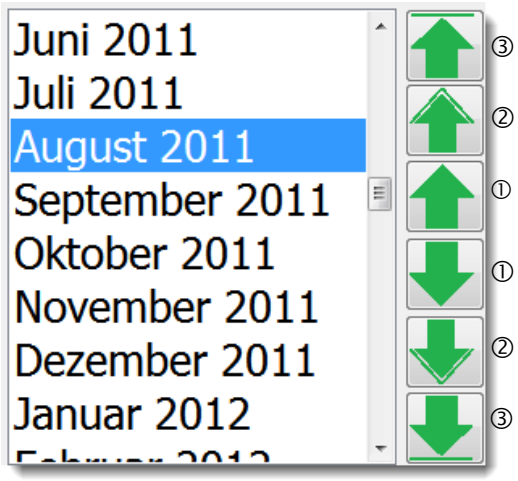


Abbildung 36 Ein Listenfeld mit seitlicher Navigationsleiste, wie man es aus diversen Kioskanwendungen kennt

Zuletzt bieten Listenfelder die Möglichkeit, mehrere Werte gleichzeitig auszuwählen. Dies wird normalerweise durch das Halten der Steuerungstaste und Anklicken mit der Maus erreicht. Leider ist jedoch die Mehrfachauswahl bei einer reinen Fingerbedienung so nicht möglich. Stattdessen besteht eine Lösung darin, jedem Wert ein Kontrollkästchen anzufügen, so dass der Benutzer einen Wert markieren kann, indem er das Kontrollkästchen antippt. Natürlich muss auch das Kontrollkästchen entsprechend fingerfreundlich angepasst werden.

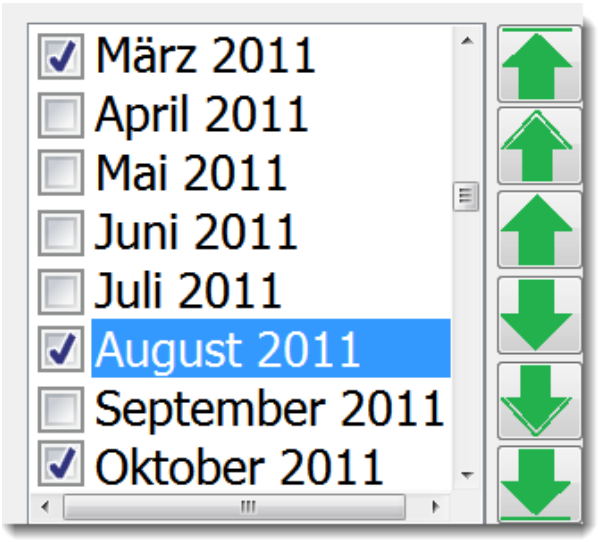


Abbildung 37 Beispielhafte Erweiterung einer Liste für die Mehrfachauswahl mit Kontrollkästchen

6 Architektur

Jede neue Situation erfordert eine neue Architektur.

Jean Nouvel
französischer Architekt

In dem vorangegangenen Kapitel 4 wurde bereits die Methode für die Umsetzung für den Lösungsansatz beschrieben. Die in Kapitel 5 vorgestellten Steuerelemente ersetzen durch die besprochene Methode andere Elemente in einem Transformationsprozess. Im aktuellen Kapitel soll daher eine grundlegende Architektur für die zu entwickelnden Transformationsanwendung aus Kapitel 7 vorgestellt werden. Dazu wird zuerst das verwendete Architekturmuster MVC beschrieben (Kapitel 6.1), um daraufhin die einzelnen Komponenten Model, View und Controller (Kapitel 6.2, 6.3 und 6.46.3) sowie ihr Zusammenspiel zu erläutern.

6.1 Übersicht

Die hier vorgestellte Architektur wurde nach dem Model View Controller Paradigma (MVC, [Krasner, et al., 1988]) entwickelt. Sie besteht daher aus drei Teilen, denen verschiedene Komponenten zugeordnet sind. Abbildung 38 illustriert die Aufteilung der Komponenten in den verschiedenen Bereichen von MVC. Im Folgenden werden die einzelnen Komponenten des MVC Musters beschrieben.

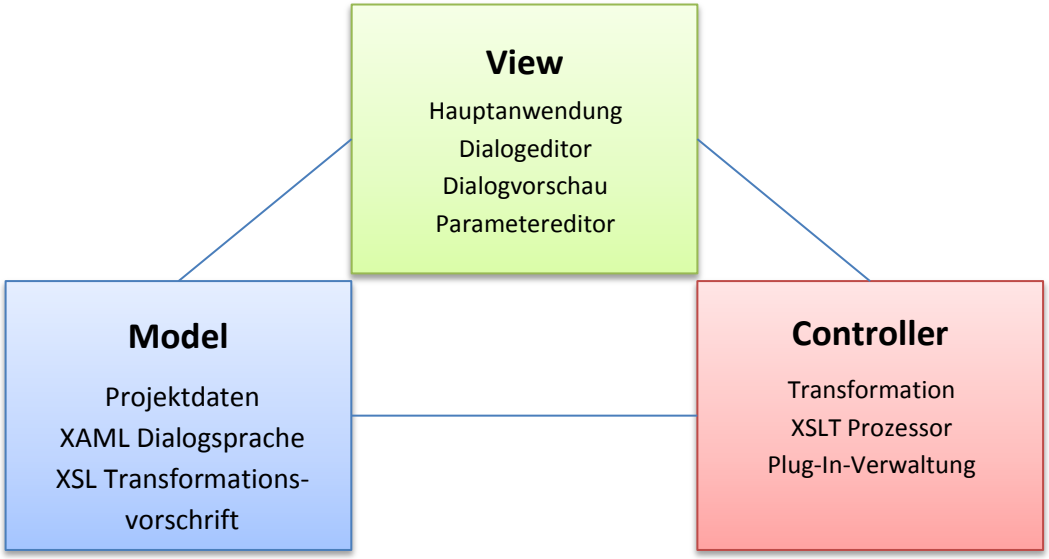


Abbildung 38 Die Komponenten aufgeteilt nach dem MVC Muster

6.2 Model

Das Modell enthält die darzustellenden Daten für die Anwendung. Diese werden in einer Projektklasse verwaltet, wie sie in Abbildung 39 zu sehen ist. Zu den wichtigsten Daten darin gehören die Dialogbeschreibungssprache XAML und die Transformationsvorschrift XSL. Diese Daten enthalten den Quelltext für den zu transformierenden Dialog sowie den Quelltext für die Transformation selbst. Im Projekt sind beide durch einen Verweis auf eine externe Datei gegeben, um einen Export und Import in und aus einer Entwicklungsumgebung (z.B. Visual Studio) zu ermöglichen.

Der Dialogquelltext besteht, wie bereits erwähnt, in dieser Umsetzung aus der XAML Syntax. Der Benutzer bearbeitet dabei den XAML Quelltext und sieht eine Vorschau des Dialogs. Für die Vorschau muss der Quelltext in ein Klassenmodell, d.h. in eine ausführbare Form, umgewandelt werden. Dies geschieht innerhalb der Komponente XAMLEditor im Abschnitt 6.3 („View“). Für das Modell besteht der Dialog daher aus einem Quelltext und einer Klassenstruktur, die synchron zu halten sind.

Weiterhin enthält ein Projekt zusätzliche Daten für Verwaltungszwecke. Darunter fallen der Projektname, eine Beschreibung und eine Projektversion. Diese Daten dienen in erster Linie zur Identifizierung eines Projekts (für den Benutzer) und Vermeidung von Fehlern (z.B. durch das Laden von nicht unterstützen Versionen).

Die Transformation soll durch Parameter (oder auch Eigenschaften) beeinflusst werden können. Diese werden für Plug-Ins (*PluginProperties*) und die XSL Transformation (*XsltProperties*) separat verwaltet. Dadurch ist ein Austausch des XSLT Quelltexts zusammen mit den dafür definierten Parametern einfacher möglich, als wenn diese unter den Plug-In Parametern verstreut liegen würden.

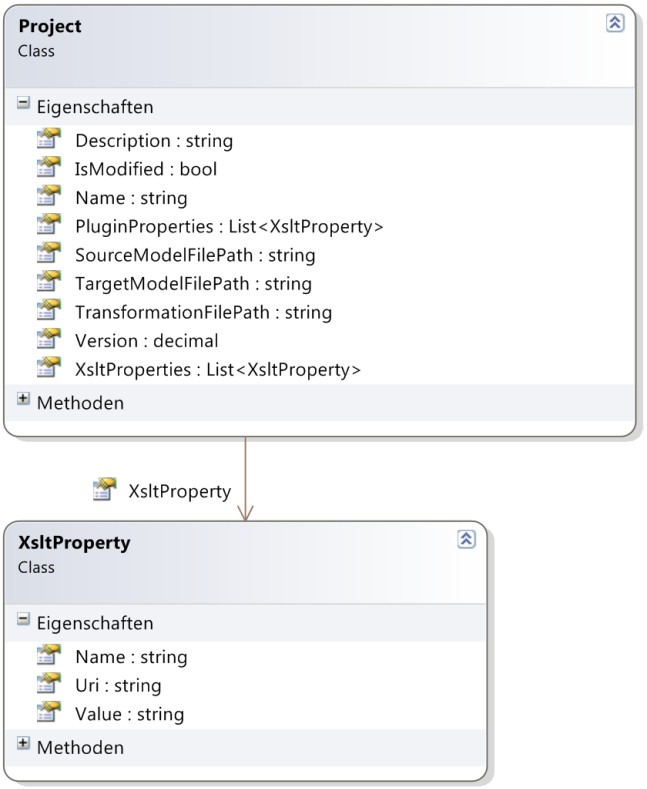


Abbildung 39 Klassendiagramm der Komponente Model

6.3 View

Die Ansicht definiert die Oberfläche der später umzusetzenden Anwendung. Der Benutzer soll in der Lage sein eine Dialogbeschreibungssprache (hier XAML) in einen Editor zu laden und bearbeiten zu können. Weiterhin soll der Benutzer die Transformation durch eine weitere Sprache, der „Extensible Stylesheet Language Transformations“ (kurz XSLT), zur Laufzeit steuern können. Dies geschieht durch den Einsatz von Texteditoren. Für die Dialogdarstellung soll zudem eine grafische Vorschau auf den Dialog möglich sein. Abbildung 40 verdeutlicht die Komponente View und die darin zu realisierenden Klassen. Im Folgenden werden diese Klassen näher beschrieben.

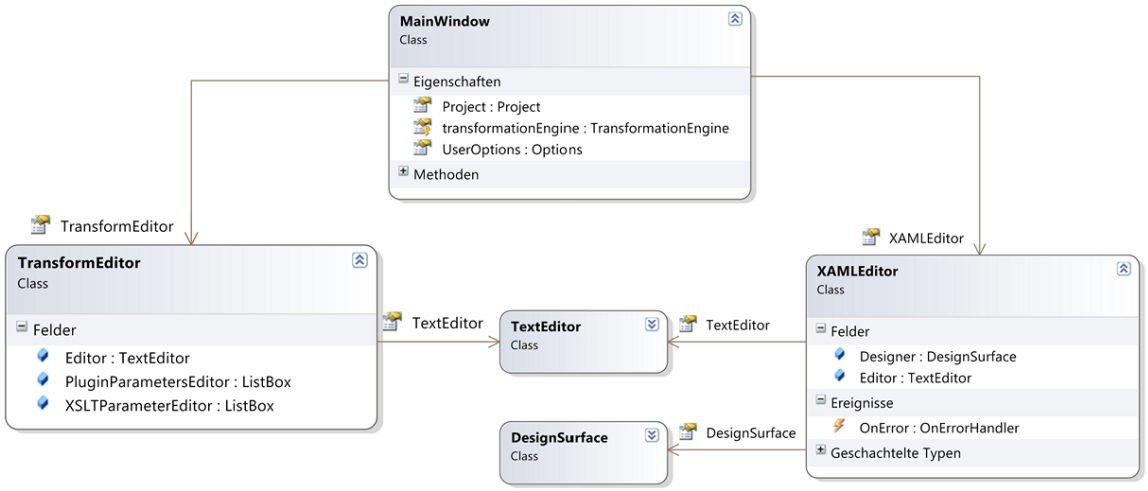


Abbildung 40 Klassendiagramm der View Komponente

MainWindow

Das Hauptfenster ist die zentrale Benutzerschnittstelle für den Benutzer. Darin können Projekte, Dialoge und Transformationsbeschreibungen geladen, bearbeiten und die Transformation durchgeführt werden, um die Ergebnisse weiterzuverwenden. Außerdem kann ein Benutzer ein neues Transformationsprojekt erstellen. Das Projekt enthält, wie bereits besprochen, alle notwendigen Daten für eine Transformation. Projekte können daher vom Benutzer gespeichert und wieder nach dem Programmstart geladen werden. Das Hauptfenster stellt dafür entsprechende Interaktionsmöglichkeiten zur Verfügung (d.h. ein Menü und Werkzeugleiste).

TransformEditor

Der Transformationseditor stellt die Oberfläche für die Bearbeitung (TextEditor) der Transformationssprache XSLT sowie der Beeinflussung der Transformation zur Verfügung. Der Benutzer soll XSLT bequem betrachten und verändern können (Syntaxhervorhebung). Weiterhin stellt der Transformationseditor zur Beeinflussung der Transformation Parametereditoren zur Verfügung. Darin können Parameter für XSLT und Plug-Ins hinzugefügt und bearbeitet werden.

XAMLEditor

Der XAMLEditor stellt die Oberfläche für die Bearbeitung (TextEditor) sowie grafischen Vorschau (DesignSurface) der Dialogbeschreibungssprache zur Verfügung. Der Benutzer soll XAML bequem bearbeiten können (Syntaxhervorhebung) und die Änderung zudem auch grafisch, d.h. in einer Art von Dialogvorschau, dargestellt bekommen. Die Vorschau wird durch eine Transformation des XAML Quelltextes in die Klassenstruktur von WPF ermöglicht. WPF stellt dazu bereits eine entsprechende Lösung (die Klasse *XAMLReader*) zur Verfügung.

6.4 Controller

Der Controller enthält die zentrale Komponente für die zu entwickelnde Anwendung: die Transformationsengine (oder auch Transformationseinheit). Diese Klasse soll alle notwendigen Methoden und Eigenschaften ent-

halten, um eine Transformation durchführen zu können. Zu den wichtigsten Eigenschaften zählen die Dialogbeschreibungssprache XAML und die Transformationssprache XSLT.

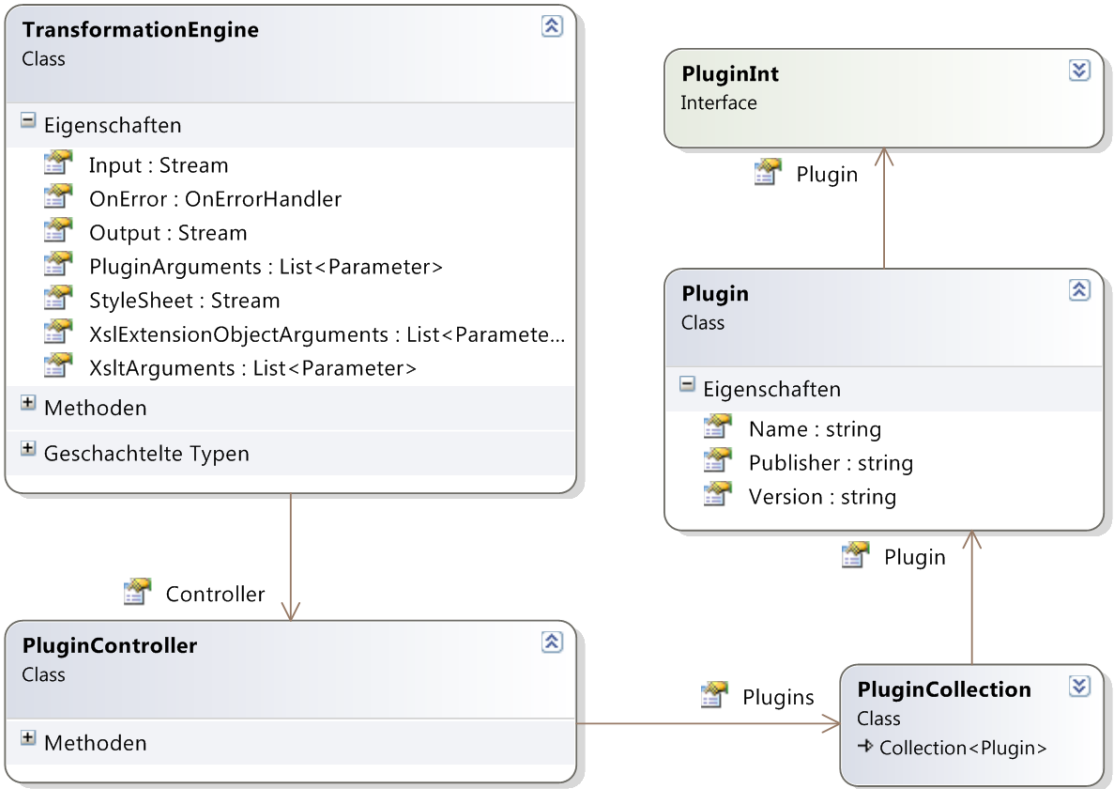


Abbildung 41 Klassendiagramm der Controller Komponente

Ein weiterer wichtiger Bestandteil der Controller Komponente soll darin bestehen, Plug-Ins zu laden und zu verwalten, um diese während der Transformation auszuführen. Die Abbildung 42 stellt das Zusammenspiel zwischen Controller und Plug-In als ein Komponentendiagramm dar. Jedes Plug-In kann durch die Implementierung zweier Schnittstellen *PluginInt* und *ProcessingIntf* an den Controller „andocken“. Dadurch wird der Klasse TransformationEngine ermöglicht auf die Plug-In Implementierung (*PluginImpl* und *ProcessingImpl*) zuzugreifen.

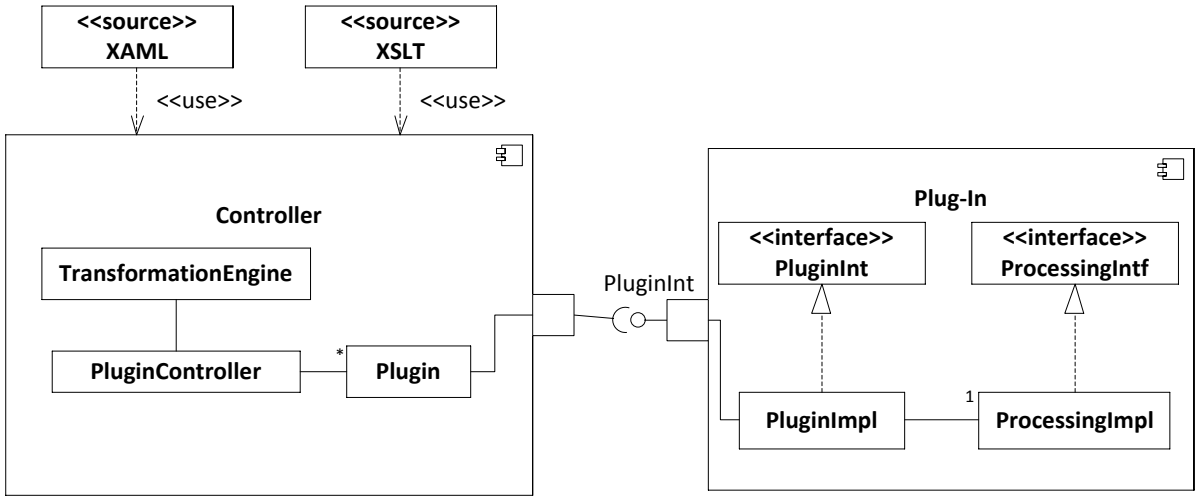


Abbildung 42 Komponentendiagramm mit Controller und Plug-In Komponenten. Plug-Ins „docken“ an den Co

TransformationEngine

Die Klasse TransformationEngine bietet Methoden und Eigenschaften, um eine Transformation eines XAML Quelltexts durchzuführen. Sie führt dazu den XSLT Prozessor auf den XAML und XSLT Daten aus und liefert das Ergebnis zurück. Weiterhin lädt die Klasse alle verfügbaren Plug-Ins, die durch die Klasse PluginController gefunden wurden und führt sie nacheinander aus. Eine detaillierte Beschreibung der Umsetzung der Klasse TransformationEngine wird in Kapitel 7.4 durchgeführt.

PluginController

Die Verwaltung der Plug-Ins wird durch die Klasse PluginController übernommen, welche sie Plug-Ins in den Speicher lädt und nach Gebrauch wieder entlädt. Die Plug-In Moduldateien können so auch zur Laufzeit ersetzt werden, während die umzusetzende Anwendung läuft. Dazu wird das „Managed Extension Framework“ (kurz MAF) eingesetzt. Eine detaillierte Beschreibung der Funktionalität von MAF und wie dieses verwendet wird, werden im Kapitel 7.6 geliefert.

Plugin

Die Klasse Plugin verwaltet ein geladenes Plug-In. Sie enthält den Name sowie den Hersteller und die Version des Plug-Ins. Ferner enthält die Klasse Plugin einen Verweis (Eigenschaft Plugin, siehe Abbildung 42) zur Implementierung des geladenen Plug-Ins in Form der Schnittstellendefinitionen *PluginIntf* (Abbildung 43). Das Plug-In implementiert diese Schnittstellen und zusätzlich *ProcessingIntf*, deren Methoden von der Klasse TransformationEngine zu Kommunikationszwecke aufgerufen werden. Die Schnittstellen sind damit aufgeteilt nach Plug-In-Verwaltung (*PluginIntf*) und Methoden für den Transformationsprozess (*ProcessingIntf*).

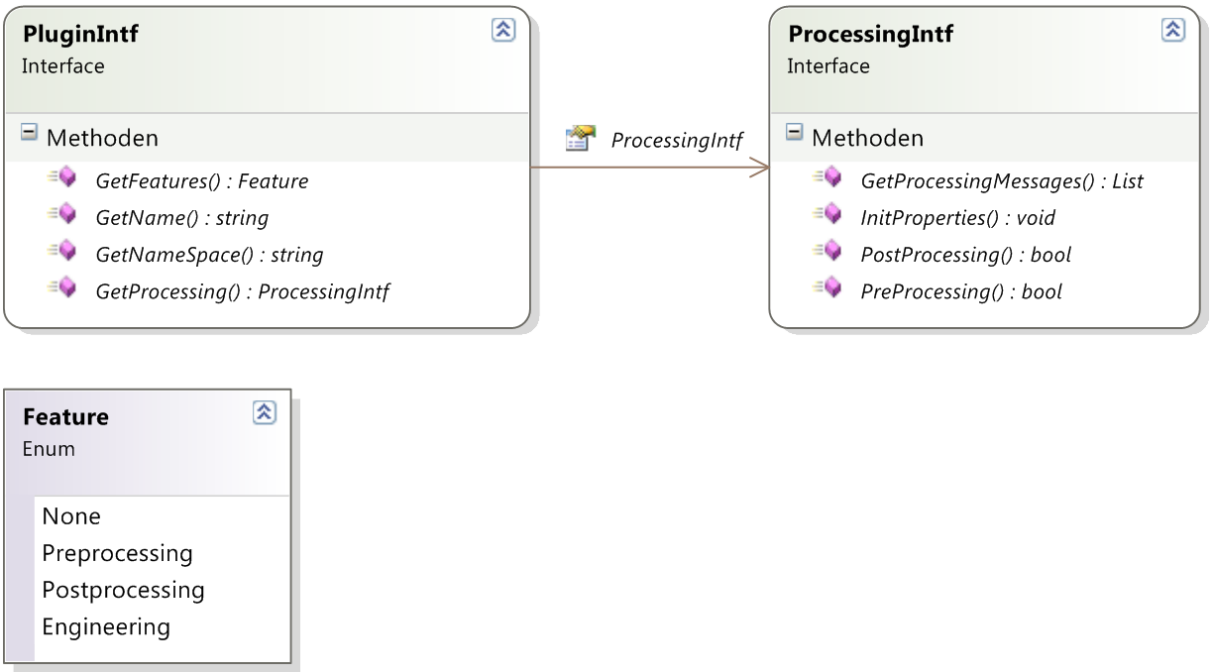


Abbildung 43 Klassendiagramm für ein Plug-In. Die Schnittstellen sind aufgeteilt nach Plug-In-Verwaltung (PluginIntf) und Methoden für den Transformationsprozess (ProcessingIntf).

7 Umsetzung

Every wall is a door.

Ralph Waldo Emerson
amerikanischer Philosoph

Ein Ziel der Diplomarbeit war es, einen Prototyp zu erstellen, der das Konzept der Transformation von in XAML geschriebenen Quelltexten umsetzt. Die folgenden Kapitel befassen sich mit dieser Anwendung, die den Namen LATTE trägt. LATTE ist ein Apronym und steht für die englische Bezeichnung **L**egacy **A**pplication **T**ransformation to **T**ouch **E**nvironments. Die Anwendung stellt die Oberfläche und die Grundlagen für die Durchführung des Transformationsprozesses zur Verfügung. Als Grundlage für die Anwendung dient die im vorangegangenen Kapitel 6 beschrieben Architektur.

In Kapitel 7.1 wird die Umsetzung von LATTE beschrieben. Das nächste Kapitel 7.2 beginnt mit der Einführung in die Oberfläche. Darauf folgend wird im Kapitel 7.3 die umgesetzten Komponenten von LATTE erläutert, um dann den Transformationsprozess, dessen Umsetzung und Anwendung in den Kapiteln 7.4 und 7.5 zu beschreiben. Die Umsetzung und Nutzung des Plug-In Systems wird in Kapitel 7.6 erläutert. Das letzte Kapitel 7.7 diskutiert die Vor- und Nachteile des Transformationsprozesses.

7.1 Einführung

Die Hauptbestandteile von LATTE sind die Oberfläche, die Transformationseinheit und das Plug-In System. Die gesamte Anwendung wurde mit dem .NET Framework 4.0 entwickelt. Die Oberfläche von LATTE nutzt das Framework Windows Presentation Foundation (WPF) und das darin zu Grunde liegende ModelView ViewModel Architekturmuster (MVVM, *MSDN Magazin Februar 2009* [Microsoft, 2009]). Es handelt sich dabei um ein erweitertes MVC Muster, welches den Controller insoweit ersetzt, dass die Datenbindungen bereits im View (d.h. in XAML Notation) umgesetzt werden. Trotzdem ist es auch weiterhin unumgänglich einen Controller in der Form einer Programmlogik zu implementieren, da Daten zur Laufzeit erzeugt werden müssen (Transformation). LATTE nutzt daher beide Ansätze MVC (siehe Kapitel 6) und MVVM, um die Dialogtransformation für den Benutzer zur Verfügung zu stellen. Mit der Hilfe des Plug-In Systems werden außerdem Plug-Ins eingebunden und ausgeführt. Abbildung 44 illustriert den schematischen Aufbau der Anwendung LATTE sowie die darin verwendeten Komponenten.

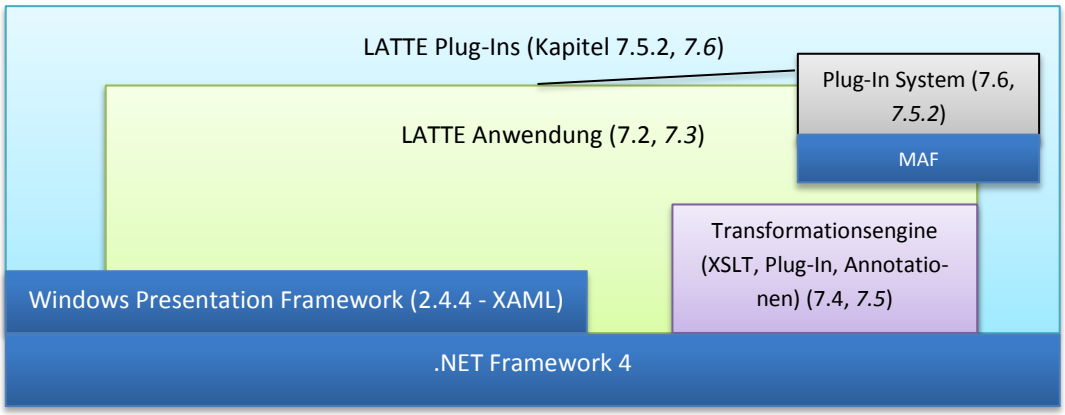


Abbildung 44 Übersicht über die verwendeten und umgesetzten Bestandteile von LATTE

7.2 Die Benutzeroberfläche

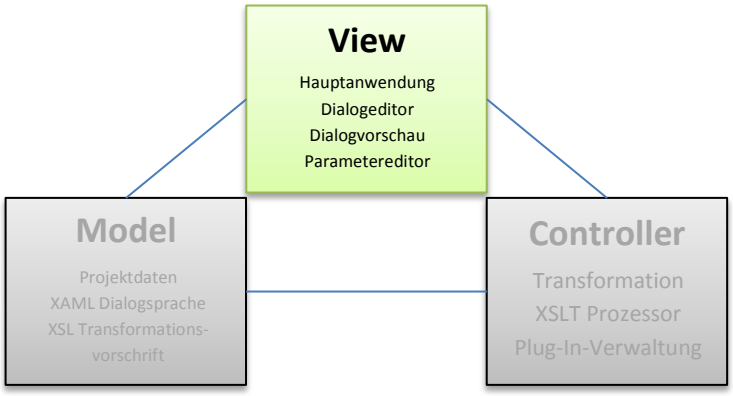


Abbildung 45 Die View Komponente aus Kapitel 6 bildet die Schnittstelle zum Benutzer

In der Benutzeroberfläche von LATTE wurde die View Komponente der Architektur (Kapitel 6) umgesetzt. LATTE besteht aus einer Oberfläche (Abbildung 46), die beinahe frei gestalten werden kann. D.h. die Fensteraufteilung ist beliebig und wird neben anderen Benutzereinstellungen für spätere Sitzungen gespeichert. Transformationsprojekte, welche die XAML Dialog- und XSLT-Dateien verwalten, können erstellt, gespeichert und geladen werden. Die Dialoge werden in einen Editor geladen, der die Schlüsselworte im Text farbig darstellt. Außerdem können die Dialoge grafisch dargestellt werden, um eine Vorschau zu erhalten und die Anpassung zu unterstützen. Der Transformationsprozess wird innerhalb eines XML Texteditors in der Sprache XSLT gesteuert. Auch dieser Editor kann den Text in unterschiedlichen Farben darstellen. Für die Transformation unterstützt LATTE externe ausführbare Steckmodule, sogenannte Cartridges oder Plug-Ins. Sie werden kurz vor der Transformation geladen und dann entsprechend ausgeführt. Meldungen, die während dieser Transformation auftreten, können in einer Liste betrachtet werden. Sie stellt die Meldungen mit verschiedenen Kriterien dar wie Typ (z.B. Warnung, Fehler) und Ursprung der Nachricht (d.h. XSLT oder Plug-In).

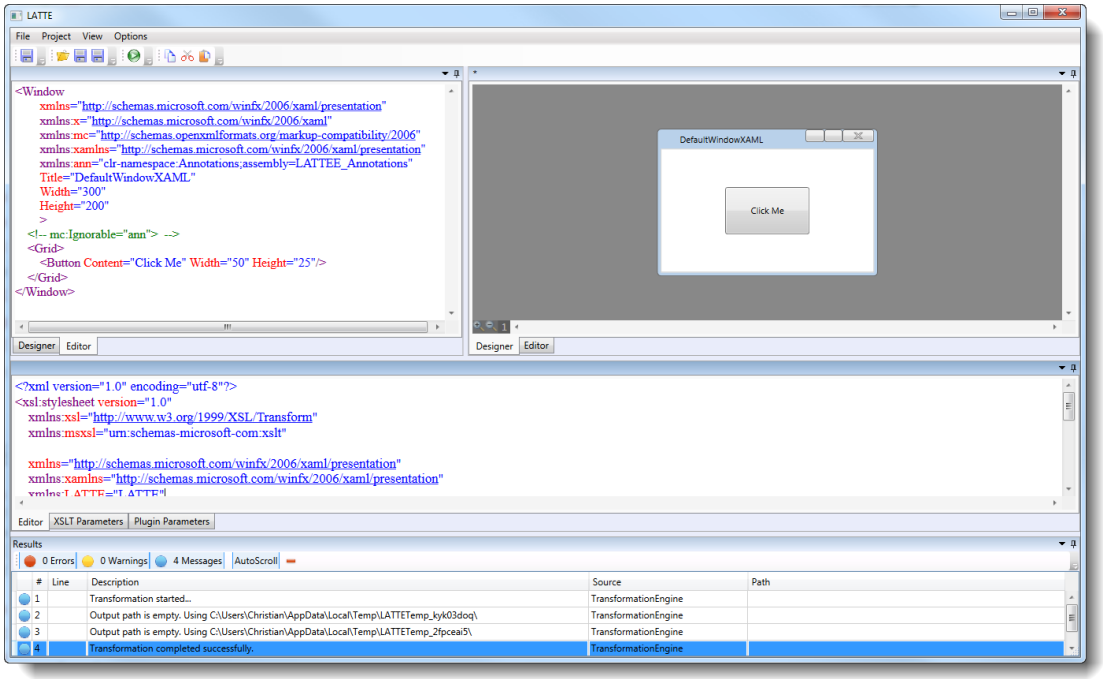


Abbildung 46 Die LATTE Anwendung zur Transformation von Dialogen

XSLT Editor

Der XSLT Editor ermöglicht die Eingabe und Korrektur von XSLT Befehlen für den Transformationsprozess. Er befindet sich in einem Register unterhalb der Dialogeditoren und neben den weiteren Editoren für XSLT und Plug-In Parameter.

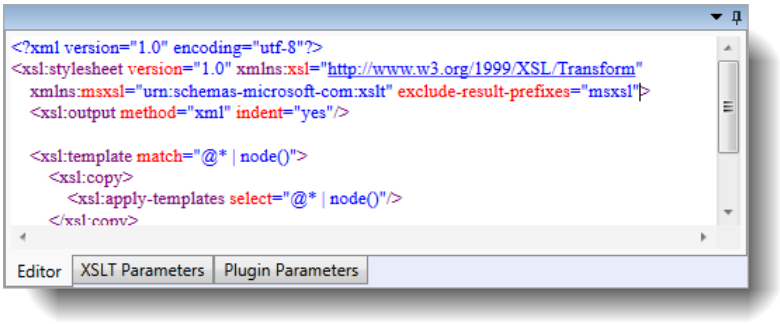


Abbildung 47 Der XSLT Editor für die Eingabe von Abbildungsvorschriften

XSLT Parameter Editor

Der Parameter Editor für XSLT (Abbildung 48) ermöglicht die Erstellung und Änderung von Parametern sowie deren Werte, die im XSLT Quelltext verwendet werden sollen. Der Editor befindet sich in einem Register unterhalb der Dialogeditoren und neben dem XSLT Editor.
Ein Parameter besteht aus einem Namen, einen Wert und einen Namensraum. Neue Parameter können hinzugefügt werden, indem das Symbol angeklickt wird. Ein oder mehrere ausgewählte Parameter können durch das Schaltersymbol entfernt werden. Um die Parameterwerte zu bearbeiten, kann einfach ein Listeneintrag ausgewählt werden. Dadurch werden die Werte in Textfeldern zum Ändern angezeigt.
Weitere Informationen zu XSLT Parametern können im Abschnitt *XSLT Prozessor* des Kapitels 7.5.1 nachgelesen werden.

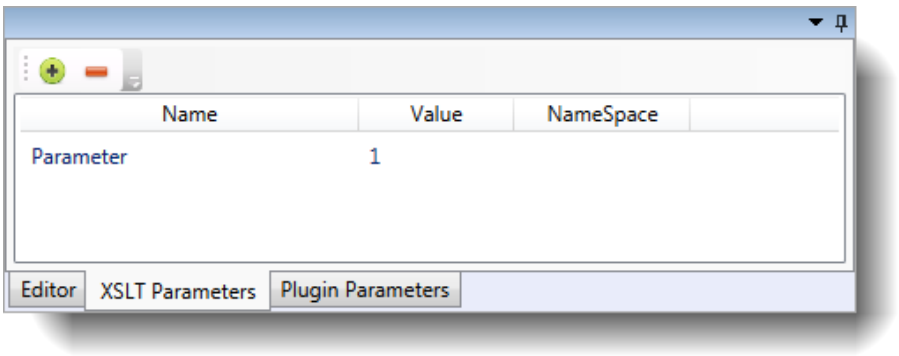


Abbildung 48 XSLT Parameter Editor für die Steuerung der Abbildungsvorschriften

Plug-Ins Parameter Editor

Mit der Hilfe des Parameter Editors können Parameter für Plug-Ins festgelegt werden. Auf diese Art können Plug-Ins abhängig vom aktuellen Projekt zusätzlich zur XSLT gesteuert werden (z.B. um Plug-Ins generell zu deaktivieren). Der Editor befindet sich in einem Register unterhalb der Dialogeditoren und neben dem XSLT Parameter Editor. Parameter bestehen aus einem Namen, Wert und Namensraum. Letzterer bestimmt welches Plug-In den Parameter zu sehen bekommt. Der Parameternamensraum muss dazu mit dem Namen des Plug-Ins

übereinstimmen. Ein leerer Namensraum ermöglicht allen Plug-Ins den Parameter zu nutzen.
Die Bedienung gestaltet sich gleich der Benutzung des XSLT Parameter Editors (Abbildung 48).

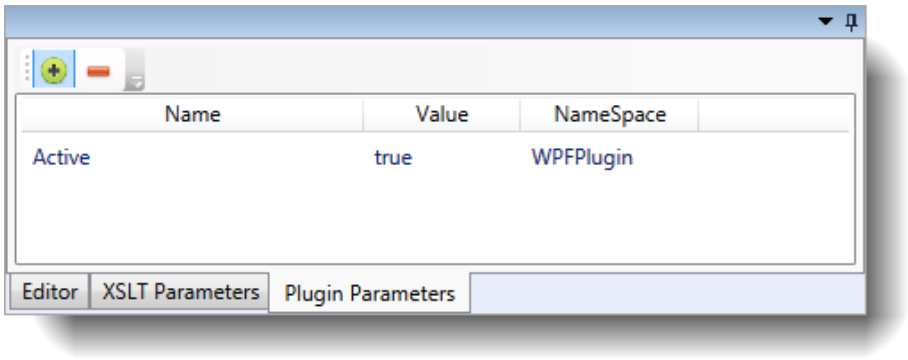


Abbildung 49 Parameterliste für Plug-Ins für die Steuerung von Plug-Ins

Meldungsfenster

Das Meldungsfenster (Abbildung 50) befindet sich am unteren Rand der LATTE Anwendung. Es enthält Nachrichten, Warnungen und Fehler aus allen Teilen der Anwendung. Dazu zählen die Dialogeditoren, der XSLT Editor und die Plug-Ins. Meldungen besitzen mehrere Kategorien, die in der Liste angezeigt werden. Dazu zählen:

- der Typ der Meldung: Nachricht, Warnung und Fehler als Kreissymbole in blau, gelb und rot.
- die Nummer der Meldung (#), damit die Reihenfolge des Auftretens ermittelt werden kann
- eine Zeilennummer, wo der Fehler auftrat (nur für Dialog- und XSLT Editoren)
- eine Beschreibung (engl. description) der Meldung, die von der Quelle geliefert wird
- die Quelle (engl. source) der Meldung, welche den Autor identifiziert (z.B. ein Plug-In)
- ein Ordnerpfad für Nachrichten, die eine Datei betreffen.

Weiterhin kann mit der Werkzeugleiste die Darstellung des Listenfelds beeinflusst werden. Dazu können die verschiedenen Arten von Meldungen ein- und ausgeblendet und die neuste Meldung immer (Schalter *Autoscroll*) ins Blickfeld gerückt werden. Veraltete Meldungen werden nicht automatisch entfernt, so dass das Betätigen des Schalters − alle Meldungen löscht. Außerdem können ein oder mehrere ausgewählte Meldungen mit der Tastenkombination *Strg + C* in die Zwischenablage kopiert werden.

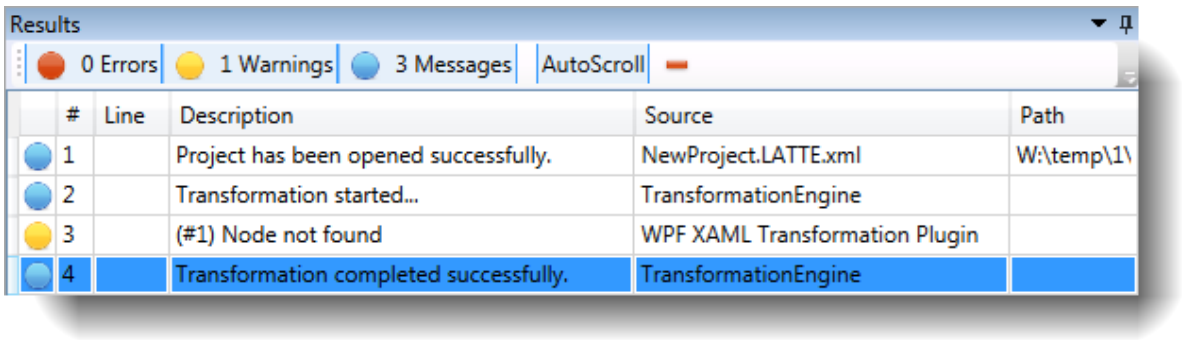


Abbildung 50 Fehler, Warnungen und Nachrichten während der Transformation werden im Meldungsfenster dargestellt

7.3 Übersicht über die Projektkomponenten

Die Anwendung LATTE wurde mit Visual Studio 2010 C# .NET 4 und der WPF entwickelt. Das Hauptprojekt besteht aus der LATTE Anwendung und elf Assemblyprojekten (Abbildung 51), welche die Oberfläche und die Transformationspipeline stellen. Die Aufteilung richtet sich nach der Aufgabe der jeweiligen Komponente (LATTE für die Oberfläche, LATTEE für den Transformationsprozess usw.). Alle Komponenten der Hauptanwendung befinden sich auf der sogenannten *Host Side*, während die Komponenten der Plug-Ins sich auf der „*AddIn Side*„ befinden. Die für die Plug-In-Kommunikation benötigten Komponenten befinden sich unter *AddIn*. Mit *LATTEC_WPF* im Ordner *AddIns* ist bereits ein Plug-In als Beispiel implementiert worden.

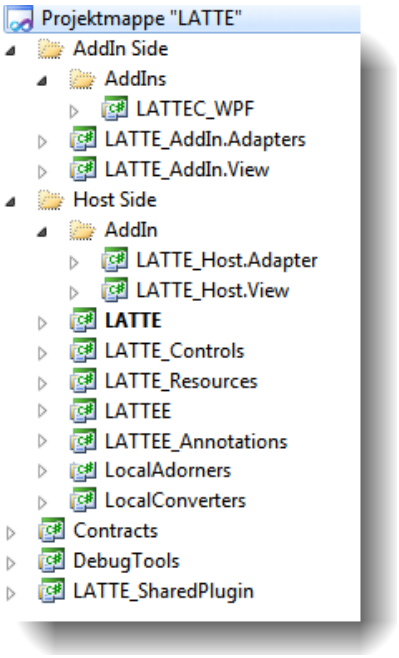


Abbildung 51 Die LATTE Projekte aufgeteilt in Hauptanwendung und Plug-In (AddIns)

LATTE

Die Anwendung wird durch das Projekt LATTE mit Visual Studio zu Binärdateien übersetzt. Die Übersetzung erzeugt die notwendige Ordnerstruktur und alle Assemblydateien. Zu den Ordnern zählen die Verzeichnisse AddIns, AddInSideAdapters, AddInViews, Contracts und HostSideAdapters für die Plug-In-Verwaltung. Die Anwendung LATTE kann durch die ausführbare Datei LATTE.exe gestartet werden.

LATTE_Controls

Die Assembly LATTE_Controls beinhaltet alle Steuerelemente, die in LATTE verwendet werden. Dazu zählen die XAML, XSLT und Parameter Editoren. Zudem enthalten sie die Vorlagendateien für ein neues Projekt mit LATTE.

LATTE_Resources

LATTE_Resources stellt einige gemeinsam benutzte Ressourcen wie XAML Styles zur Verfügung. Zudem enthält es einige neu erstellte Elemente für die Touchbedienung (siehe Kapitel 7.5.2).

LATTEE

Die Transformationspipeline (siehe Kapitel 7.5.1) wurde in der Assembly LATTEE implementiert. Das Apronym steht für **L**egacy **A**pp **T**ransformation to **T**ouch **E**nvironments **E**ngine. LATTE**E** stellt die folgenden Klassen zur Verfügung:

Name der Klasse	Quelltextdatei	Beschreibung
PluginData	PluginController.cs	PluginData enthält die Instanz eines geladenen Plug-Ins sowie weitere Information wie Name, Version oder Hersteller.
PluginController	PluginController.cs	PluginController ist zuständig für das Laden und Beenden von Plug-Ins.
TransformationEngine	TransformationEngine.cs	Die Klasse TransformationEngine stellt Methoden bereit, um Eingaben mit der Hilfe der Transformationspipeline zu verarbeiten.
LATTEE_XsltExtensionObject	TransformationEngine.cs	LATTEE_XsltExtensionObject stellt Methoden zur Verfügung, die in XSLT als Erweiterung genutzt werden können.

Tabelle 2 Die Klassen von LATTEE

LATTEE_Annotations

Die Assembly LATTEE_Annotations enthält die Implementierung der Annotationen. Diese werden genutzt, um in XAML Steuerelemente mit zusätzlichen Informationen auszustatten (siehe Kapitel 7.5.3). Projekte, die Annotationen erweitern müssen auf LATTEE_Annotations verweisen.

LocalAdordner

Diese Assembly stellt dekorative Elemente zur Verfügung. Diese Elemente erweitern Steuerelemente in XAML in ihrem Aussehen (z.B. zur Anzeige eines Dreiecks, um der Sortierrichtung im Listenkopf anzuzeigen).

LocalConverters

XAML erfordert für die Konvertierung von Datenwerten sogenannte Converter, die eine bestimmte Schnittstelle implementieren (`IValueConverter` oder `IMultiValueConverter`). In dieser Assembly werden einige Converter-Klassen implementiert, die für LATTE notwendig sind.

LATTE_Host.Adapter und LATTE_Host.View ↔ LATTE_AddIn.Adapters und LATTE_AddIn.View

Diese Projekte sind die Kommunikationsschnittstelle für die Anwendung LATTE und Plug-Ins. Die Klassen in den Adaptorprojekten (LATTE_Host.Adapter und LATTE_AddIn.Adapters) übernehmen das Marshalling (De-/Serialisieren von Daten) von Methodenaufrufen zwischen der Anwendung und den Plug-Ins. Im Gegensatz dazu stellen die View-Projekte die eigentlichen Klassen bereit, um die Methodenaufrufe zu ermöglichen. Sie konvertieren intern alle Methodenparameter in Streamobjekte und wieder zurück.
Weitere Informationen zum Plug-In System finden sich in Kapitel 7.6.

Contracts

Die Contracts Assembly definiert Schnittstellen, welche die Kommunikation zwischen LATTE und den Plug-Ins vereinbaren. Die Adaptorenklassen von LATTE und Plug-Ins müssen diese Schnittstellen implementieren.

DebugTools und LATTE_SharedPlugin

Die Assembly DebugTools implementiert Methoden zur Erfassung von Ausnahmefehlern und ihres Ursprungs. In LATTE_SharedPlugin wurden Klassen und Methoden implementiert, die von LATTE und Plug-Ins gemeinsam genutzt werden. Dazu zählen Klassen für Plug-In Parameter und Übertragung von Meldungen (Fehler, Warnungen). Zudem stehen mit der Klasse `XAMLUtility` Methoden zur Verfügung, um XAML Dialoge aus XML oder Texten (in C# *String*) zu laden.

7.4 Umsetzung der Transformationsengine

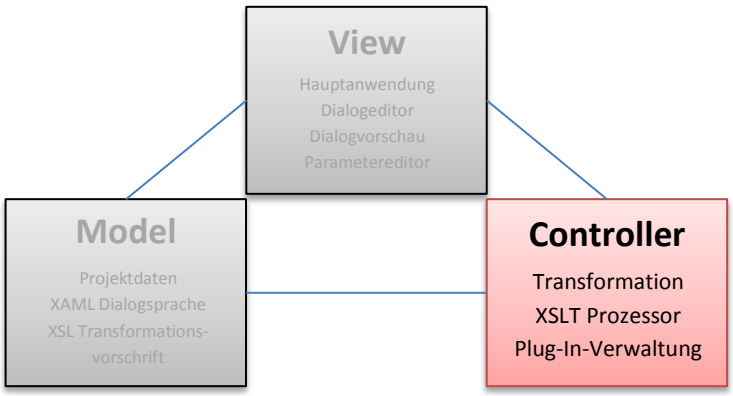


Abbildung 52 Die Klasse TransformationEngine von LATTE ist der zentrale Bestandteil der Controller Komponente aus der MVC Architektur

Die Transformationsengine oder auch Transformationseinheit genannt, ist ein Teil von LATTE, der für die Überführung der Dialoge mit Hilfe von XSLT und den Plug-Ins zuständig ist. Die Einheit wird durch die Klasse TransformationEngine (siehe Abbildung 53) implementiert und bildet die Grundlage für den gesamten Transformationsprozess. Die Durchführung des Prozesses findet anhand der Transformationspipeline (siehe Abbildung 55 und Kapitel 7.5.1) statt, welche die Abfolge und Ausführung der XSLT und Plug-In Prozessoren bestimmen.

Nach der Initialisierung der Klasseninstanz kann die Transformation durch die Methode Transform() gestartet werden. Der Ablauf folgt der Transformationspipeline. Wie in Abbildung 84 des Sequenzdiagrammes im Anhang abgebildet, werden zuerst alle Plug-Ins geladen und auf ihre Unterstützung von Prä- und Postprozessor geprüft. Zudem erhalten die Plug-Ins alle vom Benutzer eingegeben Parameter aus dem *Plug-Ins Parameter Editor* (vgl. Abbildung 48). Zusätzlich wird für die Plug-Ins ein Speicherort für temporäre Dateien als Eigenschaft WorkingPath der Parameterliste hinzugefügt.

Nach dieser Vorbereitung folgt die erste Phase der Transformationspipeline. Alle Plug-Ins mit Präprozessor werden ausgeführt und erhalten den Dialogquelltext aus der Klasseneigenschaft Input als XmlDocument. Das erste Plug-In bearbeitet das XML Dokument und liefert es an die Transformationseinheit (d.h. Transform()) zurück, worauf diese es dem nächsten Präprozessor übergibt. Dieser Vorgang wiederholt sich bis entweder kein Plug-In mit Präprozessor mehr vorhanden ist oder bis ein Plug-In einen Ausnahmefehler erzeugt.

Nachdem die erste Phase erfolgreich abgeschlossen wurde, kann die Transformation mit XSL stattfinden. Dazu werden für die Transformation notwendigen XSLT Argumente aus dem XSLT Parameter Editor geladen und zusammen mit dem XSLT Quelltext (Eigenschaft StyleSheet) an den XSLT Transformator übergeben. Die XSLT Komponente prüft den Quelltext und führt ihn anschließend mit der Ausgabe des letzten Präprozessors aus. Der letztgenannte Schritt entfällt, wenn der XSLT Quelltext Fehler enthält. Dadurch wird jedoch auch der Transformationsvorgang abgebrochen und dem Benutzer der Fehler gemeldet.

Mit dem dritten und letzten Schritt der Transformationseinheit wird die XML Ausgabe der XSLT Transformation an die Postprozessoren übergeben. Die letzte Phase läuft analog zur ersten ab, indem alle Plug-Ins mit einem Postprozessor hintereinander ausgeführt werden. Die Ausgabe des letzten Plug-Ins wird im Klassenattribut Output gespeichert und stellt das Ergebnis der Transformation dar.

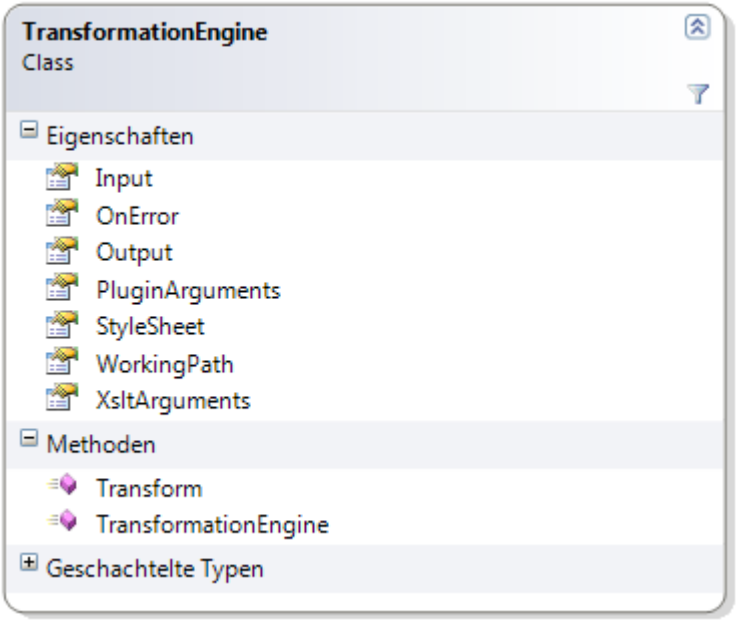


Abbildung 53 Das Klassendiagramm der Klasse TransformationEngine. Jede Transformation erhält eine Vielzahl von Eingaben.

7.5 Der Transformationsprozess

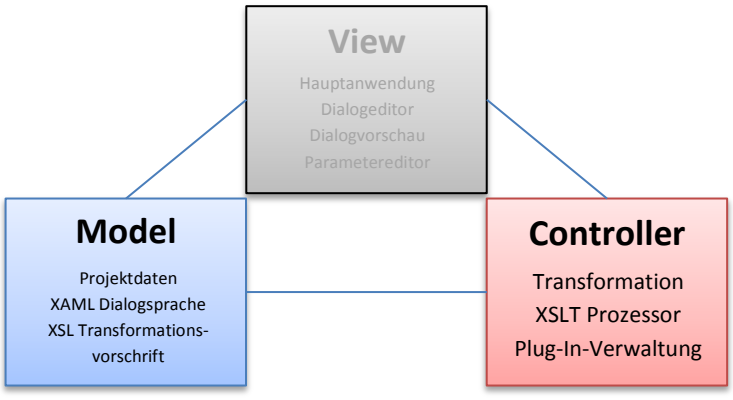


Abbildung 54 Die Model und Controller Komponenten von LATTE bilden den Transformationsprozess

Der Transformationsprozess ist der Kern der Anwendung LATTE. Ein Prozess ist eine Folge von Schritten, um einem bestimmten Zweck zu dienen (übersetzt aus IEEE Std 610.12 (1990)). In diesem Fall werden die XAML Dialogbeschreibungssprache eingelesen, die Plug-Ins geladen, die XSLT Befehle kompiliert und aufgetretene Fehler und andere Meldungen zurück an den Benutzer gemeldet. Am Ende des Prozesses entsteht ein transformierter Dialog.
Die folgenden Kapitel behandeln diesen Prozess und geben Beispiele, wie die sogenannte Transformationspipeline genutzt werden kann, um Dialoge zu transformieren.

7.5.1 Die Transformationspipeline

Die Transformationspipeline ist der Teil der LATTE Architektur, der die Transformation ausführt. Die Pipeline ist, wie in Abbildung 55 unten zu sehen, in fünf Abschnitte (Eingabe, Präprozessor, XSLT Prozessor, PostProzessor und Ausgabe) eingeteilt, die durch Leitungen (blaue Pfeile) verbunden sind. Der Transformationsprozess beginnt bei der Eingabe eines Dialogs in XAML und endet mit der Ausgabe eines angepassten Dialogs in XAML. Die eigentliche Durchführung der Transformation findet dabei in den Zwischenschritten Präprozessor, XSLT Prozessor und Postprozessor statt. Die XAML Daten werden in jedem Schritt bearbeitet und zum nächsten Plug-In oder Prozessor weitergereicht, wo weitere Änderungen durchgeführt werden können.

Alle Plug-Ins werden vor dem Transformationsprozess geladen und bei der Registrierung gefragt, welche Prozessortypen (Prä- und/oder Postprozessor) sie implementieren. So ist es möglich, dass ein Plug-In keinen Präprozessor implementiert, sondern nur einen Postprozessor. Dies ist beispielsweise bei der Erstellung eines neuen Plug-Ins hilfreich, wenn ein Postprozessor getestet werden soll und dieser nicht durch den Präprozessor beeinflusst werden darf.

Jedes Plug-In bearbeitet die XAML Struktur und liefert sie im Erfolgsfall an das nächste Plug-In in der Reihe. Dies geschieht solange bis entweder kein Plug-In mehr vorhanden ist, ein Plug-In den Prozess als gescheitert signalisiert oder eine Ausnahme (Exception) geworfen wird. Der Vorgang wiederholt sich nach der XSL Transformation schließlich im Postprozessor und führt zu einem transformierten XAML Dialog als Ausgabe.

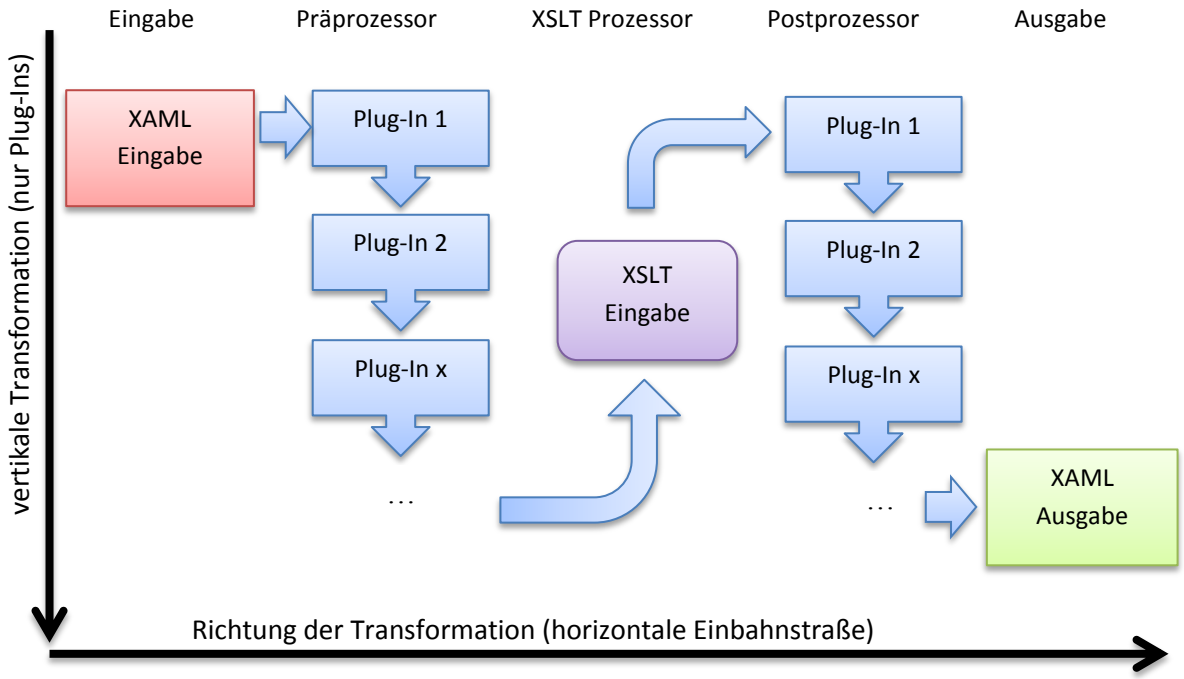


Abbildung 55 Die Transformationspipeline, wie sie umgesetzt wurde

Die XSL Transformation im Mittelpunkt ist die einzige Möglichkeit die Transformation ohne einen externen Compiler (hier C#) durchführen zu können. Die Transformation wird dabei mit Hilfe einer Skriptsprache, der XSLT (Extensible Stylesheet Language Transformation), realisiert. LATTE stellt dafür einen Editor zur Verfügung. Bei jeder Transformation wird der XSLT Quelltext validiert und auf den aktuellen Dialog ausgeführt.

Die Umsetzung und Schnittstellen wurden recht einfach gehalten. Jedoch steigerte das eingesetzte und zwingend notwendige Managed Add-in Framework (MAF, siehe Kapitel 7.6) die Komplexität. Daher wird MAF in den folgenden vier Hauptabschnitten

1. Prozessstart
2. Präprozessor,
3. Postprozessor und
4. XSLT Prozessor

ausgeblendet und nur die Implementierung ohne MAF erläutert.

Prozessstart

Der Benutzer startet den Transformationsprozess, indem sie/er F6 drückt oder im Projektmenü bzw. in der Werkzeugleiste das Symbol klickt. Wie in der Abbildung 84 zu sehen wird dadurch in der Klasse TransformationEngine die Methode Transform() aufgerufen. Neben einigen Verwaltungsaufgaben veranlasst diese Methode alle Plug-Ins aus dem *AddIns* Ordner zu laden, indem sie die Methode LoadPlugins() der statischen Klasse PluginController aufruft. Die Kommunikation zwischen Anwendung und Plug-Ins läuft dann über die ProcessingAddInView Klasse (Quelltext 6), die von dem jeweiligen Plug-In implementiert werden muss.
Nachdem alle Plug-Ins schließlich geladen wurden, kann der eigentliche Transformationsprozess mit dem ersten Schritt, dem Präprozessor, beginnen.

Präprozessor

Der erste Schritt bei der Transformation des Dialogs dient der Vorverarbeitung des XAML Dokuments, bevor überhaupt eine XSL Transformation angewendet wird. Dies kann zur Vereinfachung der XAML Struktur geschehen, indem z.B. Datenbindungen (WPF Bindings) durch ihre eigentlichen Werte ersetzt werden. Es ist dadurch möglich, diese auch dann mit XSL auszuwerten, wenn die Bindung ihren Wert erst durch eine Programmlogik erhält.

Ein Plug-In wird als Präprozessor genutzt, wenn es über seine Schnittstellenmethode ProcessingAddInView::GetFeatures() das Bit Feature.Preprocessing im Rückgabewert setzt. Daraufhin wird die durch das Plug-In implementierte Methode ProcessingAddInView::PreProcessing() aufgerufen, welche den Algorithmus enthält. Wie im Quelltext 6 an der Methodendeklaration zu sehen ist, empfängt die Methode zwei Parameter. Der erste Parameter document enthält die eigentliche XAML Struktur als XmlDocument Klasse, die durch das .NET Framework (im Namensraum System.Xml) definiert wird. Die Instanz enthält die XAML Struktur so wie sie im Editor angezeigt wird oder von einem vorangegangenen Plug-In bearbeitet wurde.

Der zweite Parameter nameSpace der Methode PreProcessing() enthält weitere Namensräume, die verwendet werden können. Derzeit wird dort nur ein Namensraum mit dem Bezeichner **xamlns** (für xaml namespace) definiert, der den voreingestellten Namensraum (engl. default namespace, gewöhnlich ist dies in XAML *"http://schemas.microsoft.com/winfx/2006/xaml/presentation")* kopiert. Dies ist eine Besonderheit von XPath, die zu beachten ist, da in XAML alle XML Knoten ohne einen Namensraum (<Button>, <ListBox> usw.) diesen voreingestellten Namensraum implizit zugewiesen bekommen (z.B. <"http://schemas.microsoft.com/winfx/2006/xaml/presentation":Button>). Eine Suche mit XPath oder über die Methoden XmlDocument::SelectSingleNode() und SelectNodes() ohne vorangestellten Namensraum (z.B.

//Button) nutzt jedoch immer den leeren Namensraum („“). Dieser stimmt aber bei XAML nicht mit dem voreingestellten überein, so dass die Suche keine Knoten liefert. Stattdessen muss der Namensraum **xamlns** bei jedem Zugriff mit der XmlDocument Klasse angegeben werden. Der Quelltext 4 zeigt wie ein Aufruf auszusehen hat.

```
public override bool PreProcessing(ref System.Xml.XmlDocument document, XmlNamespaceManager nameSpace)
{
     XmlNode node = document.SelectSingleNode("//xamlns:Button", nameSpace);
     …
```

Quelltext 4 Zugriff auf XML Knoten mit der Klasse XmlDocument

Neben den Parametern der PreProcessing() Methode kann der Algorithmus auch durch den Benutzer beeinflusst werden. Dazu werden dem Plug-In Eigenschaften als textuelle Werte über die Methode InitProperties() übergeben (Abbildung 84). Eine Eigenschaft wird durch ihren Namen und einen Wert vom Typ String im LATTE Plug-In Parameter Editor (siehe Abbildung 49) definiert. Sie gelten generell für alle Plug-Ins, so dass eine Namensraumnotation notwendig wird, um keine Kollision mit neuen Plug-Ins zu erzeugen. Einer Eigenschaft wird dazu ein Namensraum zugewiesen, wie z.B. „WPFPlugin“. Diese Notation wird von der Anwendung aufgezwungen, d.h. jedes Plug-In bekommt über InitProperties() nur diejenigen Eigenschaftswerte zu sehen, deren Namensraum mit dem von PluginAddInView::GetNameSpace() zurückgelieferten Namensraum (Quelltext 6) übereinstimmt. Eine Ausnahme bilden leere Namensräume, die alle Plug-Ins zu sehen bekommen.

Während der Ausführung des Prozessors kann es notwendig sein, den aktuellen Status oder die Fehlermeldungen des Plug-Ins an den Benutzer zurückzumelden. Dazu fragt die Anwendung das Plug-In über die Methode GetProcessingMessages() nach einer Liste von Meldungen, die im Meldungsfenster von LATTE (siehe Abbildung 50) angezeigt werden. Die Meldungen können Fehler, Warnungen oder Nachrichten sein und werden durch den Klassentyp ProcessingMessage noch weiter beschrieben. So können Zeilennummer und Spaltenposition sowie eine Bezeichner Nummer (ID) und auch ein Meldungstext zurückgegeben werden. Dies kann die Fehlerkorrektur erheblich erleichtern. Da GetProcessingMessages() erst nach PreProcessing() aufgerufen wird, müssen die Meldungen zunächst in einem Variablenfeld zwischengespeichert werden. Dieses Feld kann dann von GetProcessingMessages() zurückgegeben werden. Der Quelltext 5 zeigt, wie Meldungen erzeugt werden können.

```
//innerhalb der Plug-In Klasse
private readonly ProcessingMessageList messageList = new ProcessingMessageList();

public override bool PreProcessing(ref System.Xml.XmlDocument document, XmlNamespaceManager namespace)
{
    messageList.Clear();  //alte Nachrichten löschen

    //… arbeiten
    messageList.Add(new ProcessingMessage(MessageType.Warning,
        null/*no line*/, null/*no column*/, 1/*message ID*/, “Node not found”));
    …
}

public override ProcessingMessageList GetProcessingMessages()
{
   return messageList;
}
```

Quelltext 5 Eine Plug-In Meldung erzeugen

Es muss beachtet werden, dass eine erzeugte Fehlermeldung den Transformationsprozess nicht abbricht. Ein Prozessor kann die Pipeline nur auf zwei Arten vorzeitig beenden:

1. Die Prozessormethode (PreProcessing(), PostProcessing()) wird mit dem booleschen Wert **FALSE** beendet.
2. Der Parameter document, der die XAML Struktur enthält, wurde auf **null** gesetzt.

Es ist daher möglich, dass ein Plug-In einen Fehler in der Verarbeitung erzeugt, dies meldet, aber die XAML Struktur unverändert lässt, so dass die Transformation fortgesetzt werden kann.

Eine weitere Fehlerquelle sind unbehandelte Ausnahmen. Wird eine Ausnahme erzeugt und nicht im Plug-In abgefangen, wertet LATTE dies als Totalversagen und schließt das Plug-In für den aktuellen Transformationsprozess aus. Das Plug-In wird dann erst wieder in einer neuen Transformation berücksichtigt.

Postprozessor

Ein Postprozessor wird analog zum Präprozessor ausgeführt. Die Unterschiede in der Implementierung sind daher marginal:

1. Der Methodenname lautet PostProcessing() ist jedoch sonst gleich (Quelltext 6) in ihrer Spezifikation.
2. Die Ausführung liegt nach der Transformation durch den XSLT Prozessor (Abbildung 55 und Abbildung 84)

Im Gegensatz zum Präprozessor empfängt der Postprozessor eine bereits veränderte XAML Struktur. Sie wurde durch vorangegangene Präprozessoren oder durch die XSL Transformation verändert und benötigt möglicherweise nun noch eine letzte Anpassung. Zum Beispiel können Datenbindungen, die im Präprozessor durch Werte ersetzt wurden, wiederhergestellt oder das Layout noch angepasst werden.

Während der Präprozessor am Anfang der Transformation steht und daher nur durch Plug-In-Eigenschaften beeinflusst werden kann, ist der Postprozessor das letzte Glied in der Transformationspipeline. D.h. Präprozessoren sowie die XSL Transformation können dem XAML Dokument Informationen beifügen, die den Postprozessor lenken. Diese Informationen werden als sogenannten Annotationen in die XAML Struktur eingebettet. Ein Postprozessor kann daher entweder durch vorangegangene Prozessoren oder durch die Eingabe von XSLT Befehlen flexibel gesteuert werden. Annotationen werden im Kapitel 7.5.3 besprochen.

```
public abstract class ProcessingAddInView
{
    public abstract void InitProperties(PropertyList properties);

    public bool PreProcessing(ref XmlDocument document, XmlNamespaceManager namespace);
    public bool PostProcessing(ref XmlDocument document, XmlNamespaceManager namespace);

    public ProcessingMessageList GetProcessingMessages();
}

public abstract class PluginAddInView
{
    public String GetName();
    public String GetNameSpace();
    public Feature GetFeatures();
    public ProcessingAddInView GetProcessing();
}
```

Quelltext 6 Plug-In Kommunikationsvertrag. Diese Methoden müssen implementiert werden.

XSLT Prozessor

Der XSLT Prozessor ist die zentrale Steuereinheit der Transformationspipeline (Abbildung 55). Er wird durch die Programmiersprache XSLT [W3C, 1999] implementiert, die in LATTE mit Hilfe des XSLT Editors eingegeben werden kann (vgl. Abbildung 47). Die XSLT Sprache transformiert mit der Hilfe von Regeln den XML Baum in eine neue Struktur, die nicht unbedingt wieder XML sein muss. In dieser Arbeit wird der Transformationsprozess jedoch auf die Ausgabe von XAML beschränkt. D.h. XSLT wird benutzt, um die XAML Struktur zu transformieren und den Postprozessor zu steuern.

Der XSLT Quelltext wird in der Methode TransformationEngine::Transform() eingelesen, validiert und schließlich auf die XAML Struktur angewendet. Die eigentliche Transformation mit XSLT findet dabei über die .NET Klasse XslCompiledTransform statt, deren Methode Transform() die Umwandlung ausführt. Fehler, die dabei

auftreten, oder Meldungen, die im Quelltext über <xsl:message> eingebettet sind, werden im Meldungsfenster von LATTE (vgl. Abbildung 50) angezeigt. Fehler führen dabei immer zum Abbruch der gesamten Transformation.

Die XSLT Sprache als eine vollständige Turingmaschine (siehe [Unidex Inc., 2001]) bietet eine Vielzahl von XML Befehlen, die über den Namensraum **xsl** aufgerufen werden können. Die Möglichkeiten können sogar mit externen Bibliotheken erweitert werden. Diese stellen dann eigene Elemente und Funktionen zur Nutzung zur Verfügung. Die Transformation ist daher nicht auf die mit XSLT gelieferten Elemente und Funktionen beschränkt, sondern kann bei Bedarf erweitert werden. Eine Auswahl der Standardelemente bietet unten Tabelle 3 an, die aus der XSLT Spezifikation [W3C, 1999] erstellt wurde. Dort oder unter [W3Schools, 2011] können weitergehende Informationen zu der Vielzahl von Befehlen bezogen werden.

Elementname	Beschreibung
xsl:stylesheet	Wurzelelement eines XSLT Dokuments
xsl:include/xsl:import	Zusätzliche Style-Regeln importieren
xsl:template	Definiert eine Regel, die bei einem positiven Mustervergleich angewendet wird.
xsl:apply-templates	Regeln auf den aktuellen Knoten neu anwenden.
xsl:element	Erstellt ein XML Knoten im Ausgabedokument.
xsl:attribute	Erstellt ein XML Attribut im Ausgabedokument.
xsl:comment	Erstellt ein XML Kommentar im Ausgabedokument.
xsl:copy	Kopiert den aktuellen Knoten ins Ausgabedokument.
xsl:value-of	Ermittelt den Wert eines XML Elements.
xsl:for-each	Eine Schleife für XML Elemente.
xsl:if/xsl:choose	Bedingung auf ein XML Element anwenden.

Tabelle 3 Einige XSLT Standardelemente zur Verwendung für die Abbildungsvorschrift

Ein XSLT Dokument besteht aus dem Wurzelelement <xsl:stylesheet>, welches alle Regeln enthält, die bei einer XML Transformation angewendet werden sollen. Jede Regel wird durch das Element <xsl:template> definiert und enthält neben dem zu erkennenden Muster (match Attribut) den Inhalt, der in die Ausgabe geschrieben werden soll. Dabei können (fast) alle XSL Befehle verwendet werden, um Werte zu verändern, XML Knoten zu kopieren und damit die Ausgabe anzupassen.

```
1  <xsl:stylesheet version="1.0"
2      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3      xmlns:msxsl="urn:schemas-microsoft-com:xslt"
4
5      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
6      xmlns:xamlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
7      xmlns:LATTE="LATTE"
8
9      exclude-result-prefixes="msxsl LATTE xamlns">
10   <xsl:namespace-alias stylesheet-prefix="xamlns" result-prefix="#default"/>
11     <!-- Parameter value by LATTE, it is set to xaml namespace -->
12   <xsl:param  name="LATTE:xamlns"/>
13
14   <!-- Begin here -->
15   <xsl:output method="xml" indent="yes"/>
16     <xsl:template match="@* | node()">
17         <xsl:copy>
18             <xsl:apply-templates select="@* | node()"/>
19         </xsl:copy>
20     </xsl:template>
21 </xsl:stylesheet>
```

Quelltext 7 Ein XSLT Dokument. Diese Vorlage wird von LATTE für ein neues Projekt erzeugt.

Quelltext 7 zeigt eine XSL Transformation, die mit einer Regel (Zeile 16) alle XML Knoten und Attribute unverändert ins Ausgabedokument kopiert. Diese Vorlage wird von LATTE für neue Projekte automatisch generiert. Damit können schon sofort alle XML Elemente über das XSL Element *template* (Vorlage) ohne Änderung transformiert werden. Die Zeilen eins bis zwölf werden in Tabelle 4 unten erläutert:

Zeile	Bedeutung
1-3	Diese Zeilen definieren die Standardnamensräume für XSLT Elemente sowie Elemente aus der XSLT Bibliothek von Microsoft. Damit können zusätzliche Funktionen verwendet werden. Ein Beispiel ist die Funktion node-set(), die in Quelltext 17 verwendet wird.
5	WPF definiert den voreingestellten Namensraum für XAML auf die in Zeile 5 definierte URL. Damit XSLT Elemente ohne Namensraum in der Ausgabe als XAML Elemente erkannt werden, wird der voreingestellte Namensraum in der Zeile 5 definiert. Der Quelltext 14 zeigt ein Beispiel mit ListBox.ItemTemplate und DataTemplate dazu. Ohne diese Einstellung würden alle Elemente ohne Namensraum ein Namensraumattribut (xmlns="") in der Ausgabe erhalten, was bei ListBox.ItemTemplate als XAML Eigenschaft einen Fehler in XAML auslöst.
6	Bei XPath Abfragen muss dieser Namensraum für alle XAML Elemente angegeben werden. Zusätzlich muss er aber auch für XSLT definiert werden. Für ein Beispiel siehe dazu Quelltext 10.
7	Der Namensraum LATTE wird definiert, um die Parameter, Elemente und Funktionen von LATTE nutzen zu können.
9	In dieser Zeile werden die durch Leerzeichen getrennten Namensräume aus dem Ausgabedokument ausgeschlossen. Normalerweise würden sie sonst in der XAML Struktur unnötigerweise auftauchen.
10	Der Befehl `namespace-alias` ersetzt den Namensraum im Ausgabedokument durch den voreingestellten Namensraum (wie bei XAML üblich). Damit enthält die neue XAML Struktur keinen Namensraum `xamlns` mehr, der nur für XSLT gebraucht wird.
12	In dieser Zeile wird ein Eingabeparameter für die XSLT Transformation definiert. Der Parameter `LATTE:xamlns` wird von LATTE automatisch zugewiesen und enthält den Namensraum von XAML.

Tabelle 4 Erklärung zu den einzelnen Zeilen von Quelltext 7

Im Hauptteil (ab Zeile 15) wird jedes XML Element mit dem im Attribut match definierten Ausdruck durch den XSLT Prozessor verglichen und bei Gleichheit auf die Vorlage angewendet. Der Ausdruck wird in der Sprache XPath (siehe [W3C, 1999]) geschrieben, die auch bei der Navigation in XML Dokumenten verwendet wird. Der Ausdruck bedeutet, dass alle Attribute (@*) sowie der aktuelle XML Knoten (node()) von der Regel genutzt werden soll. Tabelle 5 listet einige der XPath Funktionen auf. Eine vollständige Beschreibung kann unter [W3C, 1999] oder [W3Schools, 2011] eingesehen werden.

Funktionsname	Beschreibung
last()	Liefert die Nummer des letzten Elements zurück.
count()	Liefert die Anzahl der Elemente zurück.
name() / local-name()	Liefert den Namen mit oder ohne Namensraum zurück.
node()	Liefert den aktuellen Knoten zurück.

Tabelle 5 Einige XPath Standardfunktionen

Oft ist es notwendig, dass XSLT Quelltexte erst während des Transformationsprozesses Werte benutzen sollen, die beim Schreiben des Codes noch nicht bekannt waren (z.B. um wie viel Prozent der neue Dialog größer sein soll). Dazu bietet XSLT sogenannte Parameter an, die in XSLT wie Variablen verwendet werden können. LATTE ermöglicht das Setzen von Parametern über den XSLT Parameter Editor (siehe Abbildung 48) an. Die dort eingegebenen Werte können im XSLT Quelltext geladen und verwendet werden. Der Quelltext 8 unten zeigt, wie Parameter genutzt werden können, um in allen XAML Elementen eine Eigenschaft sowie einen Wert zu setzen. Zu beachten ist, dass Parameternamen mit einem Dollarzeichen ($) beginnen. Sollen Parameter jedoch in XSLT Attributen verwendet werden, die keinen XPath Ausdruck (name) enthalten, so müssen sie noch zusätzlich in geschweiften Klammern („{“ und „}“) eingebettet werden, andernfalls wird ein Fehler vom XSLT Prozessor erzeugt.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt" exclude-result-prefixes="msxsl">
  <xsl:output method="xml" indent="yes"/>
  <xsl:param name="AttributeName"/>
  <xsl:param name="AttributeValue"/>
  <xsl:template match="@* | node()">
    <xsl:copy>
      <xsl:if test="$AttributeName != ''">
        <xsl:attribute name="{$AttributeName}">
          <xsl:value-of select="$AttributeValue"/>
        </xsl:attribute>
      </xsl:if>
      <xsl:apply-templates select="@* | node()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Quelltext 8 Einige Parameter in XSLT verwendet. xsl:param definiert einen Parameter, der durch $Name angewendet wird.

Bei der Programmierung von XSLT ist, wie auch schon beim Präprozessor beschrieben, zu beachten, dass XPath nur mit XAML genutzt werden kann, wenn der Namensraum **xamlns** vorangestellt wird. Die Transformationsengine erzeugt diesen Namensraum in einem XSLT Dokument automatisch, so dass er ohne Umstände genutzt werden kann (Quelltext 9).

```
<xsl:template match="xamlns:Button">
...
</xsl:template>
```

Quelltext 9 Die Besonderheit des XAML Namensraums machen es erforderlich, in XSLT den Namensraum explizit zu deklarieren

7.5.2 Grundlagen der Transformation

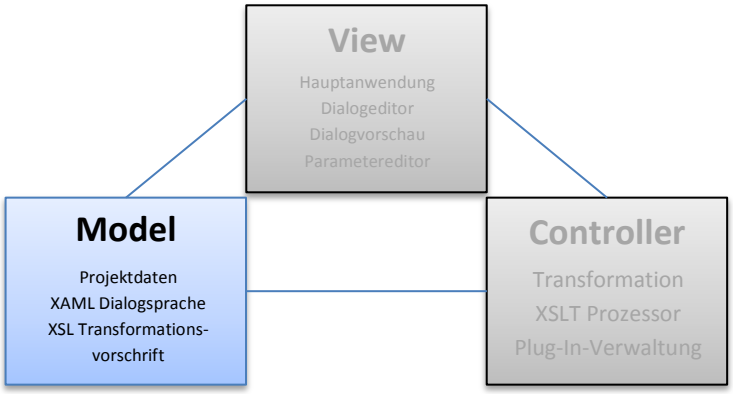


Abbildung 56 Die Modellkomponente bildet die Grundlage für die Transformation von Dialogen

Wie bereits erläutert, kann die Transformation entweder über eine Programmiersprache in .NET oder mit der Hilfe von XSLT durchgeführt werden. In diesem Kapitel werden beide Ansätze erläutert und Beispiele gezeigt, wie eine XAML Struktur transformiert werden kann. Eine Transformation besteht dabei aus vier grundlegenden Aufgaben: *Löschen, Kopieren, Einfügen* und *Ersetzen*. Durch Kombinationen, Bedingungen und Wiederholung jeder Aufgabe können komplizierte Transformationen entstehen, die helfen sollen Dialoge für neue Eingabearten anzupassen. Im Folgenden werden daher die Grundlagen der Transformation besprochen und wie diese umgesetzt werden können. Das darauf folgende Kapitel 7.5.3 geht auf eine Erweiterung der Transformation ein, die in einem kombinierten Einsatz von XSLT und Plug-In die Kommunikation vereinfacht und standardisiert.

Als Beispiel soll ein Dialog für die Nutzung auf berührungsempfindlichen Bildschirmen angepasst werden. Der Dialog in Abbildung 57 unten ist zwar einfach mit der Maus zu bedienen, doch für die Nutzung mit einem Finger ist er offensichtlich nicht ausgelegt. Dazu sind die Elemente zu klein und liegen zu nahe beieinander. So ist die

Gefahr groß, dass ein falsches Steuerelement berührt wird oder der Benutzer einfach die Maus statt den Fingern benutzt.

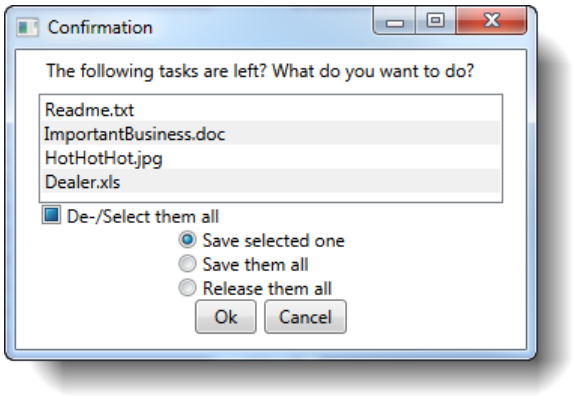


Abbildung 57 Anzupassender Dialog (Quelltext 45 siehe im Anhang)

Löschen und Kopieren

Das Kopieren von Elementen ist die am meisten ausgeführte Aufgabe bei der Transformation. Viele XML Knoten und Attribute müssen mit XSLT manuell kopiert werden. Existiert keine Regel oder wurde kein Befehl zum Kopieren in XSLT erzeugt, wird das Element nicht ins Ausgabedokument übernommen (=Löschen). Dies kann ein Vorteil sein, wenn man die Transformation z.B. in Schritten durchführen will, um die Elemente einzeln anzupassen.

```
1 <xsl:template match="xamlns:Button">
2    <xsl:copy>
4       <xsl:for-each select=" attribute::*">
5          <xsl:copy/>
6       </xsl:for-each>
7    </xsl:copy>
8 </xsl:template>
```

Quelltext 10 Knoten mit Attribute in XSLT kopieren

Eine Standardtransformation, die alle XML Knoten und Attribute kopiert, wurde bereits mit dem Quelltext 7 vorgestellt. Soll nur eine bestimmte Art von XML Knoten kopiert werden, kann das Template Element angepasst werden. Quelltext 10 zeigt eine Möglichkeit ein XAML Druckschalter zu kopieren. Dabei werden keine Unterelemente mitkopiert, sondern nur der aktuelle Knoten (Zeile 2) und dessen Attribute (Zeile 4 und 5). Mit attribute::* (oder auch @*) werden Attribute mit beliebigen Namen im Element ausgewählt und kopiert. Wie bereits erwähnt, müssen XAML Elemente, damit diese erkannt werden, mit dem xamlns Namensraum versehen und die Groß- und Kleinschreibung beachtet werden.

```
1 <xsl:template match="xamlns:Window | xamlns:Canvas | xamlns:Button">
2    <xsl:copy>
3       <xsl:for-each select="@*">
4          <xsl:copy/>
5       </xsl:for-each>
6       <xsl:apply-templates select="child::*"/>
7    </xsl:copy>
8 </xsl:template>
```

Quelltext 11 Kopieren einer gesamten XAML Struktur

Die Ausgabe genügt jedoch nicht den Anforderungen von XAML, da eine Struktur entsteht, die nur zwei Schalter enthält, aber kein übergeordnetes Fenster. Um die Struktur korrekt abzubilden, ist es daher erforderlich auch die anderen notwendigen Elemente zu kopieren. Dies wird in Quelltext 11 bewerkstelligt. Darin werden alle Fenster-, Canvas- und Schalter-Elemente erfasst (Zeile 1), deren Attribute kopiert (Zeile 3-5) sowie die Vorlage erneut auf alle Kinderelemente des aktuellen Knotens ausgeführt (Zeile 6).

Einfügen

Auf die vorgestellte Art und Weise ist es sehr aufwändig alle Arten von Knoten zu kopieren. Dazu müsste man alle Namen der auftretenden Elemente kennen und für diese für jeden Dialog neu anpassen. Es wäre wünschenswert auch unbekannte Steuerelemente unbeachtet zu kopieren und dabei neue Elemente in die XAML Struktur einfügen zu können. Daher wird Quelltext 7 im Folgenden als Grundlage verwendet. D.h. alle folgenden Quelltexte setzen die darin enthaltene Vorlage (siehe Quelltext 12) ein, um alle XAML Elemente einschließlich deren Eigenschaften zu kopieren. Da die neuen Vorlagen vor die Zeile 16 eingefügt werden, können einige Elemente speziell behandelt werden, während die restlichen Elemente einfach kopiert werden.

```
…
16 <xsl:template match="@* | node()">
17    <xsl:copy>
18       <xsl:apply-templates select="@* | node()"/>
19    </xsl:copy>
20 </xsl:template>
```

Quelltext 12 XSLT Vorlagen, um alle Attribute und Knoten zu kopieren

Eine mögliche Aufgabe bei der Transformation kann darin bestehen, neue XAML Elemente zu erstellen. Im Beispieldialog soll dies genutzt werden, um die Liste einfacher mit einem Finger bedienen zu können. Dazu kann in XAML die sogenannte ItemTemplate Eigenschaft des Listenfelds (das ListBox-Element) geändert werden. Diese Eigenschaft definiert das Aussehen jedes Eintrags innerhalb des Listenfelds. XML Knoten können in XSLT entweder durch <xsl:element > erzeugt werden oder durch die direkt Deklaration der Elemente innerhalb von XSLT. Daher sind die Beispiele in Quelltext 13 und Quelltext 14 gleichbedeutend. In beiden Fällen wird die Höhe eines Listeneintrags auf feste 30 Dialogeinheiten gesetzt, so dass die Einträge besser mit den Fingern zu treffen sind.

```
1  <xsl:attribute-set name="ItemTemplate">
2    <xsl:attribute name="Text">{Binding Path=.}</xsl:attribute>
3    <xsl:attribute name="Height">30</xsl:attribute>
4  </xsl:attribute-set>
5
6  <xsl:template match="xamlns:ListBox">
7    <xsl:copy>
8      <xsl:apply-templates select="@*"/>
9        <xsl:element name="ListBox.ItemTemplate" >
10     <xsl:element name="DataTemplate">
11       <xsl:element name="TextBlock" use-attribute-sets="ItemTemplate"/>
12     </xsl:element>
13       </xsl:element>
14     <xsl:apply-templates select="child::*"/>
15   </xsl:copy>
16 </xsl:template>
```

Quelltext 13 Elemente mit XSLT hinzufügen

Die Erstellung von Knoten und Attribute in XSLT ist, so wie in Quelltext 13 gezeigt, jedoch sehr aufwändig und nicht unbedingt einfach zu verstehen oder nachträglich anzupassen. Mit XSLT ist es daher auch möglich den bereits vorhandenen XAML Quelltext in die Transformation einzubetten. (siehe Quelltext 14). Der Nachteil ist jedoch, dass die geschweiften Klammern besonders behandelt werden müssen, da sie von XSLT für spezielle

Ausdrücke genutzt werden. In Zeile 6 ist daher die XAML Datenbindung mit doppelten Klammern versehen worden. In der Ausgabe steht dann natürlich nur jeweils eine geschweifte Klammer.

Bei beiden Quelltexten muss zusätzlich der voreingestellte Namensraum **xmlns** korrekt gesetzt werden. Sonst bindet XSLT zusätzliche Namensräume in die XML Knoten ein, was bei der angehängten Eigenschaft <ListBox.ItemTemplate> einen Fehler im XAML Editor erzeugt, da Eigenschaften in XAML selbst keine XML Attribute besitzen dürfen. Die von LATTE erzeugte XSLT Vorlage (Quelltext 7) setzt diese Bedingung bereits korrekt um, so dass nichts angepasst werden muss.

```
1  <xsl:template match="xamlns:ListBox">
2    <xsl:copy>
3      <xsl:apply-templates select="@*"/>
4      <ListBox.ItemTemplate>
5        <DataTemplate>
6          <TextBlock Text="{{Binding Path=.}}" Height="30" />
7        </DataTemplate>
8      </ListBox.ItemTemplate>
9      <xsl:apply-templates select="child::*"/>
10   </xsl:copy>
11 </xsl:template>
```

Quelltext 14 Elemente mit XSLT hinzufügen (2.Teil)

Der so erzeugte Dialogquelltext kann direkt in einer Anwendung verwendet werden. Die Abbildung 58 zeigt den Dialog mit vergrößerten Einträgen. Alle anderen Elemente wurden vorerst ohne Änderung übernommen.

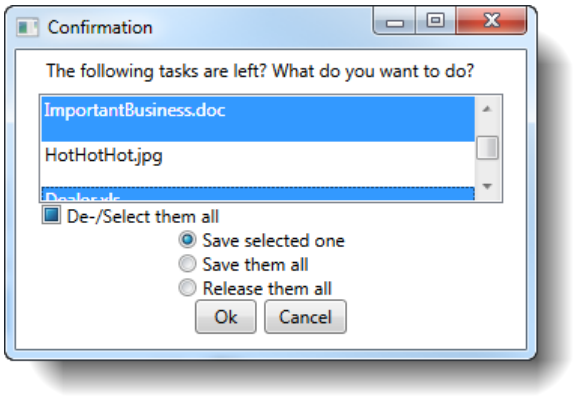


Abbildung 58 Dialog mit vergrößerten Listeneinträgen

Ersetzen

Oftmals reichen die vorhandenen Steuerelemente nicht aus, um alle Bedürfnisse zu befriedigen. Daher können Steuerelemente auch vollständig ersetzt werden. Dies geschieht im Quelltext 15, indem der XML Knoten nicht mit <xsl:copy> kopiert wird, sondern einfach ein neues Element über XSLT eingefügt wird (Zeile 7). Die restlichen Attribute des alten Steuerelements werden wieder mit <xsl:copy-of > (Zeile 8) übernommen. Dies ist natürlich nur dann möglich, wenn das neue Steuerelement diese Eigenschaften zur Verfügung stellt. Hier wurde jedoch ein neu eingeführtes Element CheckBoxTouchSwitcher verwendet, welches alle Eigenschaften einer XAML CheckBox unterstützt.

Das neue Kontrollkästchen wird über die Assembly **LATTE_Resources** (siehe auch Seite 71) zur Verfügung gestellt, die nicht nur im transformierten XAML Quelltext bekannt sein muss, sondern auch dem XSLT Prozessor. Das bedeutet den Namensraum auch im XSLT Quelltext zu definieren (Zeile 3). So definiert, wird der Namensraum dann automatisch dem Element im erzeugten XAML Dokument angehängt (Quelltext 16).

```
1  <xsl:stylesheet version="1.0"
2   …
3    xmlns:touch="clr-namespace:LATTE_Resources;assembly=LATTE_Resources"
4  >
5  …
6  <xsl:template match="xamlns:CheckBox">
7     <touch:CheckBoxTouchSwitcher>
8     <xsl:copy-of select="@*" />
9       <xsl:apply-templates select="@*|node()"/>
10    </touch:CheckBoxTouchSwitcher>
11 </xsl:template>
```

Quelltext 15 Transformation eines Kontrollkästchens

Dies ist jedoch nicht immer vorteilhaft, insbesondere dann, wenn mehrere Steuerelemente in demselben Namensraum im Dokument verteilt liegen. Denn die Deklaration des Namensraums kann auch in einem übergeordneten Element erfolgen, so dass alle Unterelemente diesen Namensraum verwenden können. In XAML definiert man üblicherweise alle benutzten Namensräume im Wurzelelement, das normalerweise ein <Window> Element ist. D.h. es wäre notwendig ein Attribut als Namensraum dort einzufügen. Unglücklicherweise ist dies in XSLT 1.0 nicht so einfach umzusetzen, da ein Name wie xmlns:touch von XSLT als qualifizierter Bezeichner ausgewertet wird und damit nicht ins Ausgabedokument gelangt. Die Lösung kann aber mit externen Skripten umgesetzt werden. Microsoft stellt dazu bereits eine Bibliothek zur Verfügung. Aber auch andere Hersteller bieten Skripte dafür an (siehe [Stewart, et al., 2006] und [Mangano, 2006 S. 631ff.]).

```
<touch:CheckBoxTouchSwitcher
    Name="checkBox4"  […]  xmlns:touch="clr-namespace:LATTE_Resources;assembly=LATTE_Resources" />
```

Quelltext 16 Erzeugtes Steuerelement CheckBoxTouchSwitcher

Die verwendete Lösung stammt ursprünglich aus [Georges, 2007] und wurde für den Quelltext 17 gekürzt. Sie wird auf ein Fenster-Element angewendet (Zeile 1) und ruft eine weitere Vorlage auf (Zeile 3-6), die für die Erzeugung des Namensraums zuständig ist.

```
1  <xsl:template match="xamlns:Window">
2     <xsl:copy>
3        <xsl:call-template name="make-namespace-node">
4           <xsl:with-param name="prefix">touch</xsl:with-param>
5           <xsl:with-param name="uri">clr-namespace:LATTE_Resources;assembly=LATTE_Resources</xsl:with-param>
6        </xsl:call-template>
7        <xsl:apply-templates select="@*|node()"/>
8     </xsl:copy>
9  </xsl:template>
10
11 <xsl:template name="make-namespace-node">
12    <xsl:param name="prefix"/>
13    <xsl:param name="uri"/>
14       <xsl:variable name="dummy">
15          <xsl:element name="{ $prefix }:e" namespace="{ $uri }"/>
16       </xsl:variable>
17    <xsl:copy-of select="msxsl:node-set($dummy)/*/namespace::*"/>
18 </xsl:template>
```

Quelltext 17 Erzeugen eines Namensraum innerhalb eines Fenster-Elements

Die Vorlage definiert neben den Parametern *prefix* und *uri* (Zeile 12 und 13) auch eine Variable *dummy* (Zeile 14), die als Wert einen XML Knoten enthält. Dessen Name besteht aus dem übergebenen Präfix (hier „touch“) und einem beliebig gewählten Namen (hier „e“). Der Namensraum des Elements (Zeile 15) wird schließlich noch auf den Parameter *prefix* gesetzt, der später den Namen der Assembly **LATTE_Resources** zugewiesen bekommt. Der so erzeugte Dummy-Knoten wird in Zeile 17 verwendet, um dessen Namensraum in den aktuel-

len Knoten (hier Window, Zeile 1) einzufügen. Der Skriptbefehl msxsl:node-set in Zeile 17 konvertiert dazu den Wert der Variable *dummy* in eine Knotenmenge, die dann mit XPath durchsucht werden kann. Der Umweg ist notwendig, da in XSLT der Wert der Variable *dummy* nicht automatisch für XPath in eine Knotenmenge konvertiert wird. Letztendlich werden in der Knotenmenge die Namensräume (/namespace::*) aller Elemente (/*) in den aktuelle Knoten des Aufrufers kopiert (Zeile 3 bis 6). Mit dem Aufruf der Vorlage in Zeile 3 können so dem Fenster-Element beliebige neue Namensräume zugewiesen werden.

```
<Window Title="Confirmation" […]
        xmlns:touch="clr-namespace:LATTE_Resources;assembly=LATTE_Resources">
```

Quelltext 18 Window-Element mit manuell eingefügtem Namensraum

Das Ergebnis der Bemühungen ist ein <Window> Element mit dem vorgegebenen Namensraum (Quelltext 18) und ein Dialog mit einem veränderten Kontrollkästchen (Abbildung 59).

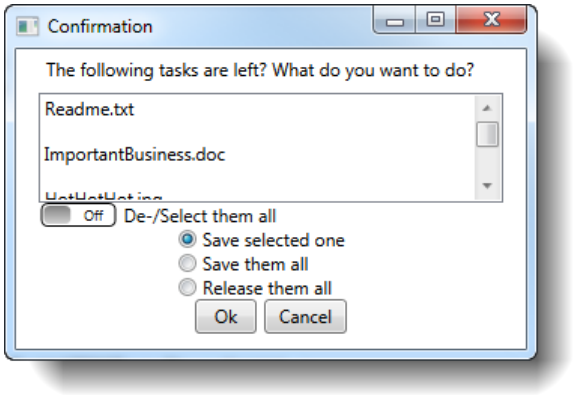


Abbildung 59 Dialog mit ersetztem Kontrollkästchen

Zuletzt bleibt im Dialog noch die Abstände zwischen den Steuerelemente zu vergrößern. Dies kann in XAML auf mehrere Arten geschehen:

- Die Höhe und Breite des Steuerelements kann vergrößert werden.
- Ein Rand kann um das Steuerelement gesetzt werden, der es von anderen Elementen räumlich abtrennt.

Die Umsetzung hängt sehr von der eingesetzten Layout Technik ab. Wird ein Canvas eingesetzt, also die Steuerelemente absolut positioniert, müssen alle Elemente abhängig von der Größe und Position neu ausgerichtet werden. Dazu existieren bereits umfangreiche Arbeiten, die im Kapitel 2.5.2 Reverse Engineering besprochen wurden wie z.B. das Auckland Layout Model (siehe Seite 42).

Für dieses Beispiel wird ein einfacherer Ansatz gewählt. Die Steuerelemente wurden dazu im Dialog bereits mit einem Layout-Panel versehen (siehe Quelltext 45 im Anhang), so dass Größenänderungen automatisch umgesetzt werden.

In XAML werden Ränder durch die Eigenschaft Margin definiert. Margin wird von XAML als eine Folge von Pixelabständen interpretiert. Vier durch Komma getrennte Zahlen geben den Abstand des Steuerelements in alle vier Richtungen an (z.B. „5, 5, 5, 5“). Weiterhin kann auch nur eine Zahl für alle vier Abstände verwendet werden. Da die Interpretation dieses Formats in XSLT schwer umzusetzen ist, wird stattdessen die Höheneigenschaft der Elemente angepasst. Dazu sollen alle Steuerelemente in ihrer Höhe um einen bestimmten Wert, der von der Art des Elements abhängt, vergrößert werden.

Die erste Idee wäre eine einzelne, neue Vorlage zu erstellen, die einfach alle Steuerelemente enthält, deren Höhe angepasst werden soll. Dieser Ansatz ist in XSLT so nicht umzusetzen, da XML Elemente nur immer **einmal** auf eine Vorlage angewendet werden können (die erste passende Vorlage). Damit wäre es nicht mehr

möglich das Kontrollkästchen (CheckBox) oder das Listenfeld (ListBox) zu konvertieren, weil sie ja bereits durch eine andere Vorlage in ihrer Höhe transformiert wurden. Jedoch wäre es sehr wohl möglich die Größe des Kontrollkästchens oder des Listenfelds in den bereits existierenden Vorlagen zusätzlich zu behandeln und alle restlichen Elemente gemeinsam in einer Vorlage zu transformieren. Allerdings gibt es einen noch besseren Ansatz.

Mit XSLT können Vorlagen erstellt werden, die nicht nur Knoten behandeln, sondern auch Attribute. Auf diese Weise ist es möglich die XAML Eigenschaft *Height* für jedes Steuerelement separat anzupassen, indem das Elternelement mit dem gewünschten Name verglichen wird. Der Quelltext 19 definiert ein Vorlage setSize, die als Parameter den Attributnamen (@Width oder @Height) sowie den Namen der XAML Eigenschaft (Width oder Height) übergeben bekommt.

Damit nicht alle Steuerelemente in ihrer Größe angepasst werden, wird in der Vorlage geprüft, zu welchem Steuerelement das Attribut gehört (Zeilen 9, 12 und 15). Alle anderen Elemente werden ohne Änderungen weitergereicht (Zeile 6 und 19), darunter fällt auch das Fenster-Element.

Der Wert, um den die Größe verändert wird, ist natürlich abhängig von dem eingesetzten Element. In den Zeilen 10, 13 und 16 werden die Größendifferenzen als Konstanten zur aktuellen Größe des Elements addiert. Das ist für den Beispieldialog in Ordnung, könnte aber in anderen Fällen zu groß oder zu klein sein, so dass man hier überlegen sollte Parameter zu nutzen, die vor der Transformation im XSLT Parameter Editor (vgl. Abbildung 48) angegeben werden.

```
1  <xsl:template name="setSize">
2     <xsl:param name="AttributeName" select="@Height"/>
3     <xsl:param name="PropertyName" select="Height" />
4
5     <xsl:choose>
6        <xsl:when test="parent::xamlns:Window|parent::xamlns:StackPanel">
7           <xsl:copy/>
8        </xsl:when>
9        <xsl:when test="parent::xamlns:ListBox">
10          <xsl:attribute name="{$PropertyName}"><xsl:value-of select="$AttributeName + 64"/></xsl:attribute>
11       </xsl:when>
12       <xsl:when test="parent::xamlns:RadioButton | parent::xamlns:CheckBox">
13          <xsl:attribute name="{$PropertyName}"><xsl:value-of select="$AttributeName + 16"/></xsl:attribute>
14       </xsl:when>
15       <xsl:when test="parent::xamlns:Button">
16          <xsl:attribute name="{$PropertyName}"><xsl:value-of select="$AttributeName + 14"/></xsl:attribute>
17       </xsl:when>
18       <xsl:otherwise>
19          <xsl:copy/>
20       </xsl:otherwise>
21    </xsl:choose>
22 </xsl:template>
23
24 <xsl:template match="@Height | @Width">
25    <xsl:call-template name="setSize">
26       <xsl:with-param name="AttributeName"><xsl:value-of select="."/></xsl:with-param>
27       <xsl:with-param name="PropertyName"><xsl:value-of  select="local-name()"/></xsl:with-param>
28    </xsl:call-template>
29 </xsl:template>
```

Quelltext 19 Transformation zum Verändern von Höhe und Breite

Schließlich wird die Vorlage setSize durch die Vorlage in Zeile 24 auf die Höhe und Breite aller Attribute angewendet. Dazu werden die Vorlagenparameter auf das Attribut (@Height oder @Width) und den Name der Eigenschaft (Height oder Width) gesetzt. Da im Dialog alle Steuerelemente diese beiden Eigenschaften besitzen, erzeugt die Transformation den rechten Dialog der Abbildung 60. Wird in Zeile 24 nur die Eigenschaft Höhe

angepasst, stellt sich der Dialog langgezogen dar, wie auf der Abbildung 60 links zu sehen ist. Der vollständige Quelltext (Quelltext 46) des abgebildeten Dialogs kann im Anhang eingesehen werden.

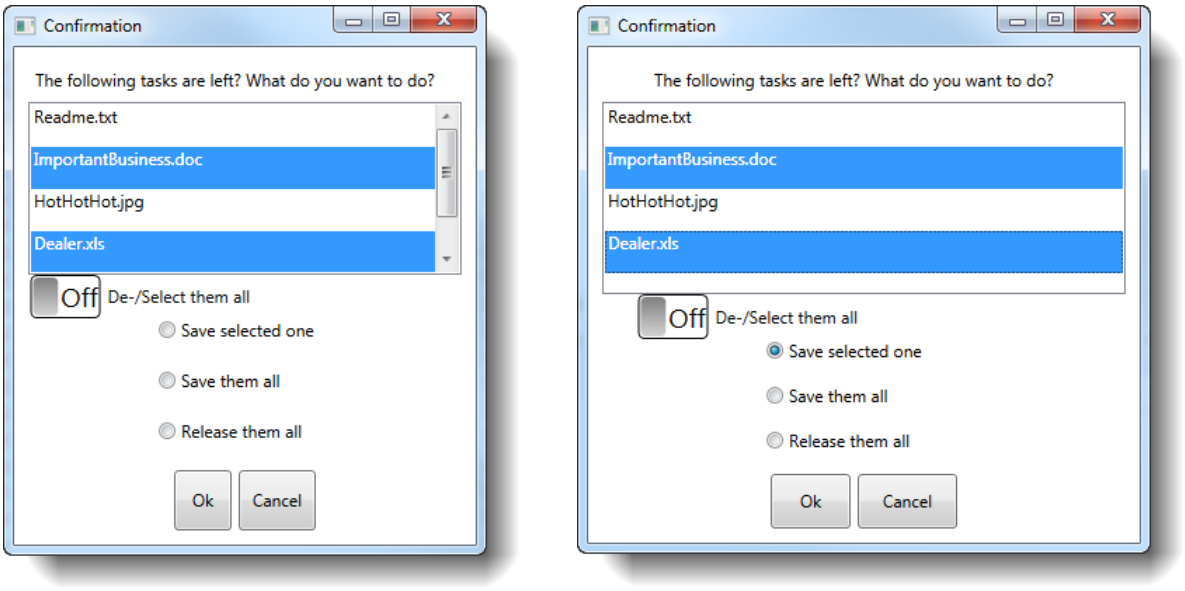


Abbildung 60 Links: Höhe angepasst, Rechts: Höhe und Breite angepasst

Transformation mit Plug-Ins

Die Transformation mit reinem XSLT mag besonders anfangs große Probleme bereiten, weil das Verhalten des XSLT Prozessors nicht immer den Erwartungen entspricht. Aber obwohl XSLT turing-vollständig ist, so bedeutet dies nicht, dass jedes Problem einfach gelöst werden kann. Letztendlich soll ein Problem so einfach wie möglich gelöst werden, daher ist es notwendig auch moderne Programmiersprachen im Transformationsprozess einzusetzen. Einige Transformationen, wie die der Größe, können so einfacher geprüft, zwischengespeichert und auch mehrmals verändert werden.

Der Einstiegspunkt für Plug-Ins ist mit den Methoden PreProcessing und PostProcessing (siehe Quelltext 6 auf Seite 78) definiert. Sie empfangen die XAML Struktur, die angepasst werden soll. Mit dem Projekt LATTEC_WPF (das C steht für Cartdrige, zu dtsch. Steckmodul) steht eine Vorlage bereit, die für weitere Plug-Ins genutzt werden kann. Die eigentliche Implementierung steckt in der Datei *PluginImpl.cs* und muss entsprechend angepasst werden. Die Beispiele Quelltext 19 und folgende zeigen, wie dies geschehen kann.

Mit dem Quelltext 20 wird dieselbe Transformation durchgeführt wie in Quelltext 19. Mit SelectNodes() (Zeile 7) werden alle Attribute Height und Width des XML Baums abgefragt und versucht, deren Inhaltswert als Double zu interpretieren (TryParse()). Während in XSLT ungültige, numerische Werte, wie z.B. * oder *„Auto“* für automatische Größe, ohne jede Änderung ins Ausgabedokument übernommen werden, können diese Werte mit .NET etwas einfacher interpretiert und ausgewertet werden (Zeile 6). Es ist jedoch trotzdem möglich in XSLT die Erkennung und Transformation vorzunehmen, wenn auch mit mehr Aufwand. Ein weiterer Vorteil von PostProcessing besteht darin, dass Meldungen, Warnungen und Fehler direkt als solche mit zusätzlichen Informationen an LATTE gesendet werden können. Die eingefügte Warnung in Zeile 40 wird durch LATTE nach dem Methodenaufruf PostProcessing ausgewertet und im Meldungsfenster angezeigt.

Während PostProcessing nach der XSLT Transformation ausgeführt wird und alle Werte aus dieser Transformation bereits enthält, könnte ein Beispiel für die Methode PreProcessing sein, bestimmte Werte in eine einfachere Form für XSLT umzuformen. Die bereits erwähnte Eigenschaft Margin kann so in mehrere XML Attribute aufgeteilt werden und ist in XSLT damit einfacher lesbar. Zusammengefasst kann der Präprozessor damit die

XSLT Verarbeitung beeinflussen, indem XML oder XAML Elemente zusätzlich eingefügt, verändert oder gelöscht werden.

```
public override bool PostProcessing(
        ref System.Xml.XmlDocument document,
            XmlNamespaceManager nameSpace)
{
    messageList.Clear();

    XmlNodeList nodes = document.SelectNodes("//@Height | //@Width");

    if (nodes != null)
    {
        foreach (XmlAttribute item in nodes)
        {
            String strSize = item.Value;
            double size = 0;

            if (Double.TryParse(strSize, out size))
            {
                switch (item.OwnerElement.LocalName)
                {
                    case "Window": break;
                    case "ListBox":
                        size += 64;
                        break;
                    case "CheckBoxTouchSwitcher":
                    case "CheckBox":
                    case "RadioButton":
                        size += 16;
                        break;
                    case "Button":
                        size += 14;
                        break;
                }

                item.Value = size.ToString();
            }
        }
    }
    else
    {
        messageList.Add(new ProcessingMessage(MessageType.Warning,
            /*LineNumber*/null, /*LinePosition*/null, /*MessageID*/1001,
            "No nodes found"));
    }
}
```

Quelltext 20 PostProcessing mit Höhe und Breite

Diese Art der Kommunikation zwischen den Prozessoren besitzt jedoch den Nachteil, dass zusätzliche Elemente die Prozessoren anderer Hersteller stören könnten. Daher wird in Kapitel 7.5.3 die Annotation eingeführt. Sie definiert die Kommunikationsschnittstelle als XAML Element und unterstützt auch mehrere Plug-Ins gleichzeitig im Prozess.

Nutzen eines XAML Laders

WPF besitzt die Möglichkeit eine XAML Struktur in eine Klassenstruktur transformieren zu können. Die entsprechende .NET Klasse nennt sich XamlLoader und ermöglicht durch den Aufruf von statischen Methoden eine

Fensterinstanz zu erzeugen, die alle Dialogsteuerelemente enthält. Damit können die Elemente und ihre Werte über Klasseninstanzen und Eigenschaften zugegriffen werden, ohne den Syntax von XPath nutzen zu müssen.

```
1 XmlTextReader reader = new XmlTextReader("DialogFile.xaml");
2 Window x = XamlReader.Load(reader) as Window;
3
4 x.Margin = new Thickness(10, 20, 20, 10);
```

Quelltext 21 XAML Struktur laden

Eine *perfekte* Lösung wäre gewesen, wenn die geladene XAML Struktur auch wieder als XAML Quelltext hätte geschrieben werden können. Die Rückwärtstransformation kann zwar mit der Methode Save der Klasse XamlWriter durchgeführt werden, jedoch ist die Rückumwandlung in XAML Code aus verschiedenen Gründen nicht vollständig. Sie ist daher nur von sehr begrenztem Nutzen. Die Autoren von *Serialization Limitations of XamlWriter.Save* in [Microsoft, 2011] und [Hillberg, 2006]) nennen die folgenden Einschränkungen für XamlWriter:

- Das Steuerelement wird in dem Zustand, den es zur Laufzeit besitzt, als XAML Repräsentation gespeichert. Die erzeugte XAML Struktur kann sich daher vom Original unterscheiden.
- Alle verwendeten Ressourcen werden in die XAML Struktur eingebettet. Ressourcen wie *Style* oder *DataProvider* werden direkt in XAML gespeichert, d.h. ein aus mehreren Dateien bestehender XAML Dialog wird zur einer Quelldatei zusammengefasst.
- Datenbindungen werden nicht in ihrer ursprünglichen Form in die XAML Struktur zurückgeschrieben, sondern nur deren Werte. D.h. Datenbindungen wie Binding, StaticResource und DynamicResource werden zur Laufzeit (d.h. beim Laden) ausgewertet und beim Speichern durch den aktuellen Wert ersetzt.
- Ereignisroutinen werden nicht in die XAML Struktur übernommen, sondern ignoriert.

Zwei Quellen bieten dazu mehr oder weniger funktionierende Lösungen an (siehe [Richter, 2007] und [AlexDov, 2008]). Doch eine Überprüfung ergab immer wieder Schwierigkeiten mit verschiedenen XAML Strukturen (Templates, Datenbindungen, Stile), so dass auch hier keine vollständige Lösung präsentiert werden kann.

7.5.3 Erweiterte Transformation mit Annotationen

Annotationen sind aus den Hochsprachen wie Java und C# schon lange Zeit bekannt. Annotationen, auch „Attribute“ genannt, sind Anmerkungen, die Klassen, Methoden oder anderen Deklaration durch zusätzliche Informationen, sogenannte Metadaten, erweitern. Diese Metadaten können zur Laufzeit oder von externen Werkzeugen (auch der Kompiler) eingelesen und interpretiert werden.

Die Frameworks von Java und C# stellen eigene Klassen und Methoden zur Verfügung, um zur Laufzeit auf Annotationen (in Java [Sun, 2005 S. 281]) oder Attribute (in C# [Microsoft, 2011]) zugreifen zu können. Durch den Einsatz von externen Werkzeugen ist es aber genauso möglich, in Sprachen Annotationen zu verwenden, die sonst nicht Bestandteil der Sprache sind, beispielsweise als Kommentar. Diese sind jedoch nur vor und während der Übersetzung vorhanden, um von den externen Werkzeugen erkannt zu werden; nicht jedoch zur Laufzeit.

Für XML oder genauer gesagt XAML wurde eine Klassenstruktur entwickelt, die in XAML eingebettete Annotationen erlaubt. Annotationen in XAML können als Mittel zur Ablaufsteuerung innerhalb des Transformationsprozesses benutzt werden. Somit ist es möglich, dass entlang der Transformationspipeline Informationen an XAML Elemente angehängt werden, um in einer späteren Phase der Pipeline durch einen Prozessor (Prä-, XSLT- oder Postprozessor) ausgewertet werden zu können. Die Auswertung ist dabei nicht auf die horizontale Transformation (siehe Abbildung 55) beschränkt, sondern kann auch zwischen Plug-Ins erfolgen (vertikale Transformation).

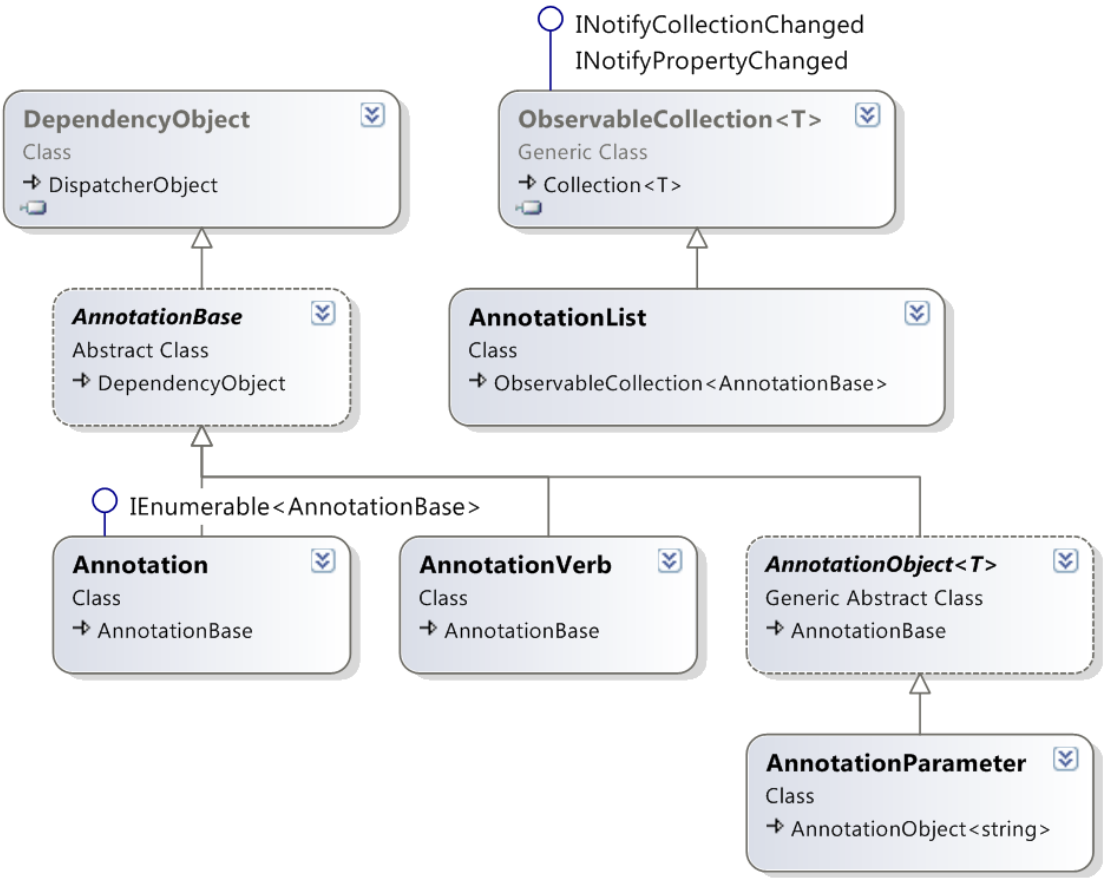


Abbildung 61 Klassendiagramm der erstellten Annotationen für XAML

Das Klassendiagramm in Abbildung 61 gibt einen Überblick über die entwickelte Klassenstruktur für Annotationen. Eine Beschreibung der Klassen kann man aus Tabelle 6 entnehmen.

Die Klassen für Annotationen sind mit WPF entwickelt worden und können daher direkt als Objekte in einem XAML Quelltext eingesetzt werden, ohne die visuelle Darstellung zu ändern. Annotiert werden können alle Objekte in XAML, die von der Klasse DependencyObject abgeleitet wurden. Das sind in erster Linie alle sichtbaren Steuerelemente (u.a. die WPF Klassen *Control, Label, Button, ListBox*, *ListBoxItem, Window, Text* oder auch *ComboBox*), jedoch keine Datenbindungen oder vordefinierte Objekte wie Farben (z.B. Colors.Green), Pinsel (z.B. Brushes.Blue) usw. Eine grafische Übersicht zu den XAML Elementen und welche von DependencyObject abgeleitet sind findet sich unter [Chattopadhyay, 2010].

Aus den Elementen erzeugt ein XAML Parser automatisch die entsprechenden Instanzen. Zusätzlich wurde für diese Arbeit eine Deserialisierung und Serialisierung den Klassen hinzugefügt, so dass eine XAML Struktur und deren Annotationen auch über einen XML Parser gelesen und geschrieben werden können. Sind die vorhandenen Klassen für die eigenen Bedürfnisse nicht ausreichend, können zudem einfach neue abgeleitete Annotationsklassen erstellt und verwendet werden.

XAML Annotationen sind in erster Linie nicht sichtbare Elemente in der XAML Struktur. Eine Annotation wird als eine angehängte Eigenschaft (engl. attached property) definiert und lässt sich daher an beliebige XAML Steuerelemente anhängen. Diese Art von Eigenschaften verhält sich dabei wie jede normale Eigenschaft eines XAML Elements, jedoch wird sie einem Element ohne dessen Wissen hinzugefügt. Das mit WPF eingeführte Konzept erweitert damit die Liste der zugreifbaren Eigenschaften eines Elements ohne den Quelltext des Elements anpassen zu müssen. Im Folgenden wird gezeigt, wie Annotationen genutzt werden können, um zusätzliche Informationen in XAML Quelltexte einzubetten.

Klasse/XAML Element	Beschreibung
AnnotationList	AnnotationList ist selbst kein XAML Element, sondern definiert eine Hilfsklasse für eine Sammlung von Annotationen. Die Klasse Annotation nutzt AnnotationList zur Verwaltung von Annotationen.
AnnotationBase	*AnnotationBase* definiert eine *abstrakte* Klasse, von der alle Annotationsklassen abgeleitet werden müssen. Sie definiert eine **Eigenschaft Name** sowie den Namensraum (**Prefix, Uri**) der Annotation für die Serialisierung. Annotationsklassen, die innerhalb einer anderen Annotation (Annotation oder AnnotationList) auftreten sollen, jedoch nicht an einem XAML Element angehängt sein dürfen, können von dieser Klasse abgeleitet werden.
Annotation	Annotation ist die Basisklasse aller Annotations und bietet die angehängte Eigenschaft **Attach** als Einstiegspunkt für XAML Annotationen an. Sie erlaubt als Unterelemente weitere Annotationen zu definieren. Neue Annotationsklassen, die an XAML Elemente angehängt werden dürfen, können von dieser Klasse abgeleitet werden.
AnnotationVerb	AnnotationVerb ist eine Annotation, die eine auszuführende Funktion definiert. Die Funktion wird über die Eigenschaft **Verb** benannt und kann beliebig viele **Parameter** beinhalten. Der Name und Wert jedes Parameters werden dabei mit AnnotationParameter übergeben. Die Semantik wird durch den jeweiligen Nutzer bestimmt. Plug-Ins können AnnotationVerb mit einer eigenen Klassenmethode verknüpfen, die dadurch ausgeführt werden kann.
AnnotationObject	*AnnotationObject* ist eine *abstrakte* Klasse für Annotationen mit XML Inhalt (in XAML **Content** genannt). Abgeleitete Klassen können so XAML Elemente definieren, die einen Inhalt präsentieren. Neue Annotationsklassen können von dieser Klasse abgeleitet werden.
AnnotationParameter	AnnotationParameter definiert einen Parameter mit **Name** und Wert. Die Klasse wird aber nicht ausschließlich für AnnotationVerb verwendet, sondern kann auch für eigene Annotationen wiederverwendet werden.

Tabelle 6 Beschreibung der Annotationsklassen für XAML

Annotationen in XAML verwenden

Damit ein XAML Parser die Annotationen erkennen kann und keinen Fehler erzeugt, muss die Assembly **LATTE_Annotations** durch einen Namensraum mit einem Präfix bekannt gemacht werden. Dies passiert üblicherweise im Fenster-Element kann aber auch an jedem, der Annotation übergeordneten Elementen vorgenommen werden. Es ist jedoch nicht möglich den Namensraum direkt in der Annotation zu setzen, weil die Syntax von XAML keine XML Attribute an angehängten Eigenschaften erlaubt.

```
1  <Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
2          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
3          xmlns:ann="clr-namespace:Annotations;assembly=LATTEE_Annotations">
4      <StackPanel>
5          <ListBox Name="listBox1">
6              <ann:Annotation.Attach>
7                  ...
8              </ann:Annotation.Attach>
9          </ListBox>
10     </StackPanel>
11 </Window>
```

Quelltext 22 Definieren einer Annotation in XAML. Die Annotation wird als eine fremde Eigenschaft Attach an das Listenfeld angehängt.

Der Quelltext 22 zeigt wie eine Annotation bekannt gemacht (Zeile 3) und an ein Listenelement gehängt wird (Zeile 6). Zu beachten ist dabei, dass im Gegensatz zu den Annotationen in Programmiersprachen, die Annotation in XAML innerhalb eines XAML Steuerelements deklariert werden und nicht vor dem zu annotierenden Element. Die Annotation selbst beginnt mit der Deklaration der angehängten Eigenschaft Attach (Zeile 6) im Element ListBox. In XAML ist somit dieses Steuerelement um die Eigenschaft Attach erweitert worden und kann daher auch als Objekteigenschaft benutzt werden. Eine explizite Verknüpfung (z.B. durch Nennung des Ele-

mentbezeichners in der Annotation) zwischen Annotation und Steuerelement ist somit nicht notwendig, da die angehängte Eigenschaft bereits dem Steuerelement angehört. Außerdem spielt es keine Rolle an welcher Stelle die Annotation innerhalb des Steuerelements gesetzt wird.

Zwischen den Zeilen 6 und 8 (Quelltext 22) können beliebige Annotationselemente eingefügt werden. Die derzeit möglichen Kombinationen sind in einer angepassten Version der erweiterten Backus-Naur-Form in Tabelle 7 dargestellt. Symbolnamen sind dabei in den Produktionsregeln mit geschweiften Klammern dargestellt, da die Zeichen < und > bereits für XML verwendet werden.

Symbolname	Produktionsregel
S	<{**Präfix**}:Annotation.Attach>{**Annotation**}</{**Präfix**}:Annotation.Attach>
Annotation	{***AnnotationBase***}{**Annotation**} | <{**Präfix**}:Annotation [Name="{**Name**}"]>{**Annotation**}</{**Präfix**}:Annotation> | {**AnnotationVerb**}{**Annotation**} | {}
AnnotationVerb	<{**Präfix**}:AnnotationVerb Verb={**Name**}>{**Parameter**}</{**Präfix**}:AnnotationVerb>
Parameter	<{**Präfix**}:AnnotationParameter [Name="{**Name**}"]>{**Wert**}</{**Präfix**}:AnnotationParameter>{**Parameter**} | {}
***AnnotationBase* (abstrakt)**	{*AnnotationBase* kann nicht direkt verwendet werden, aber jede von *AnnotationBase* abgeleitete und nicht abstrakte Klasse. Das Präfix muss nicht mit dem Präfix in S übereinstimmen, sondern kann auch auf eine Assembly eines Drittherstellers verweisen.}
Präfix	{Beliebiger Name für XML Namensräume. Diese müssen jedoch als Namensraum definiert worden sein.}
Name	{Beliebiger Name für XML Attribute}
Wert	{Beliebiger Wert für XML Inhalte}

Tabelle 7 EBNF Produktionsregel für die entwickelten Annotationen in XAML

Annotationen in XSLT verwenden

Annotationen wurden entwickelt, um in XSLT eine Steuerung des Postprozessors zu ermöglichen. Der Quelltext des XSLT Prozessors kann direkt in LATTE bearbeitet werden. Daher ist es normalerweise nicht notwendig, den XSLT Prozessor mit dem Präprozessor zu steuern. Manchmal ist es dennoch nützlich bestimmte XSLT Aktionen zu ignorieren und ein Steuerelement direkt an den Postprozessor, ohne Änderung, übergeben zu lassen.

Im folgenden Beispiel wurde dazu eine Annotation (Quelltext 23) erstellt, um den XSLT Prozessor (Quelltext 24) zu veranlassen einen Vorlagen-Befehl zu ignorieren. Die Annotation, die auch von einem Präprozessor stammen könnte, definiert dazu einen Parameter mit dem Namen Ignore. Da der Name beliebig gewählt werden kann, wurde noch ein zusätzlicher Bezeichner im Attribut CUri (Custom Unified Resource Identifier) definiert. Damit können unterschiedliche Plug-Ins identische Bezeichner nutzen, ohne dass eine Namenskollision zu befürchten wäre.

```
1 <ListBox Name="listBox1" Width="309" Height="73" SelectionMode="Extended">
2     <ann:Annotation.Attach>
3         <ann:AnnotationParameter Name="Ignore" CUri="WPF_Plugin">true</ann:AnnotationParameter>
4     </ann:Annotation.Attach>
5 </ListBox>
```

Quelltext 23 Eine annotierte Listendefinition in XAML. Die Annotation AnnotationParameter kann nur innerhalb der Attach Eigenschaft definiert warden.

Annotationen zu nutzen kann insbesondere mit XSLT 1.0 aufwändig sein. Wie im Quelltext 24 zu sehen wurde eine Bedingung erstellt, welche die oben genannte Annotation prüft. In Zeile 3 wurde dazu ein Ausdruck in XPath erstellt, der innerhalb der Vorlage (hier xamlns:ListBox) nach einem Element AnnotationParameter sucht, dessen Attribute Name und CUri die dargestellten Werte besitzen. Zusätzlich muss AnnotationParameter

innerhalb der angehängten Eigenschaft Annotation.Attach definiert worden sein, um andere, später definierte Annotationen mit denselben Eigenschaften auszuschließen.

```
1  <xsl:template match="xamlns:ListBox">
2    <xsl:choose>
3      <xsl:when test="LATTE:ConvertBoolean(./ann:Annotation.Attach/ann:AnnotationParameter
                                          [@Name='Ignore' and @CUri='WPF_Plugin'])">
4        <xsl:copy><xsl:apply-templates select="child::*"/></xsl:copy>
5      </xsl:when>
6      <xsl:otherwise>
7        <!-- ListBox konvertieren -->
8      </xsl:otherwise>
9    </xsl:choose>
10 </xsl:template>
```

Quelltext 24 Eine XSLT Transformation mit Prüfung einer Annotation

Damit eine Prüfung mit verschiedenen Wahrheitswerten (z.B. True, true, TRUE oder 1) ohne zu großen Aufwand durchgeführt werden kann, wurde ein Erweiterungsobjekt im Namensraum **LATTE** zur Verfügung gestellt. So kann mit der Methode ConvertBoolean ein Wahrheitswert leichter aus einem Text erzeugt werden. Der Nachteil jedoch besteht darin, dass das XSLT nur mit LATTE ausgeführt werden kann.

Annotationen mit XSLT verwenden

Der Einsatz von Annotationen zur Steuerung des Postprozessors gestaltet sich einfach, wenn der Präprozessor keine eigenen Annotationen einsetzt. Um eigene Annotationen in XSLT zu erzeugen, können die aus Kapitels 7.5.2 bekannten Möglichkeiten zum Einfügen von Elementen genutzt werden. In Quelltext 25 wird mit XSLT eine Annotation an ein Listenfeld gehängt. Es kann später (siehe Abschnitt *Verwendung in Plug-Ins (C#)*) von einem Postprozessor abgefragt werden. Die im Beispiel angegeben Parameter sind nur von der Implementierung des Plug-Ins abhängig.

```
1  <xsl:template match="xamlns:ListBox">
2     <ann:Annotation.Attach>
3         <ann:AnnotationVerb Name="Convert" CUri="WPF_Plugin">
4            <ann:AnnotationParameter Name="TargetTemplate">ListBoxTouch</ann:AnnotationParameter>
5            <ann:AnnotationParameter Name="ItemHeight">50</ann:AnnotationParameter>
6         </ann:AnnotationVerb>
7     </ann:Annotation.Attach>
8  </xsl:template>
```

Quelltext 25 Ein Listenelement wurde in XSLT annotiert

Weitere Annotationen innerhalb eines Elements zu erstellen ist, wie gesehen, denkbar einfach. Jedoch kann es vorkommen, dass bereits der Präprozessor Annotationen für ein Element gesetzt hat. Da man nicht wissen kann, welche Annotationen im Element eingesetzt werden, müssen alle Annotationen – zusätzlich zu den neu eingeführten – kopiert werden. Das bedeutet, Annotationen können nicht nur in der Ausgabe doppelt vorkommen, sondern sie können auch unterschiedliche Werte besitzen. Dieses Problem ist allerdings dadurch zu umgehen, dass Annotation, die als erstes definiert wurden, Vorrang haben. Denn eine Abfrage mit XPath resultiert immer in einer Menge von XML Knoten, deren Reihenfolge mit der Reihenfolge im XML Baum übereinstimmt. Doppelt auftretende Annotationen werden mit XPath dadurch ignoriert, indem nur das erste XML Element in der Ergebnismenge verwendet wird.
Zu beachten ist allerdings, dass diese Regel nicht für die Deklaration der Annotation mit Annotation.Attach gilt. Denn XAML lässt eine angehängte Eigenschaft nur einmal zu. Jede weitere Deklaration einer angehängten Eigenschaft wird mit einem Fehler bestraft. Daher müssen alle im Präprozessor definierten Annotationen mit XSLT kopiert und erweitert werden.

Der Quelltext 26 zeigt, wie Annotationen aus dem Präprozessor innerhalb einer Annotation.Attach kopiert werden. Wenn die in der XSLT definierten Annotationen eine hohe Priorität genießen sollen, müssen sie zuerst innerhalb von Annotation.Attach definiert werden. Andernfalls können die Zeilen 5 und 6 vertauscht werden.

```
1 <xsl:template match="xamlns:ListBox">
2   <xsl:copy>
3     <xsl:apply-templates select="@*|child::*[local-name() != 'Annotation.Attach']"/>
4     <ann:Annotation.Attach>
5       <ann:AnnotationParameter Name="Ignore" CUri="WPF_Plugin">true</ann:AnnotationParameter>
6       <xsl:apply-templates select="./ann:Annotation.Attach/child::*"/>
7    </ann:Annotation.Attach>
8   </xsl:copy>
9 </xsl:template>
```

Quelltext 26 Annotationen eines Listenelements kopieren und erweitern

Eine große Bedeutung besitzt der Befehl xsl:apply-templates in Zeile 3. Darin werden alle Attribute (@*) und alle Unterelemente (child::*), außer der Annotation selbst, kopiert (local-name() != 'Annotation.Attach'). D.h. die gesamte Struktur des Listenelements (Zeile 1) bleibt erhalten, ohne jedoch die Annotationen zu kopieren. Denn diese müsse erst noch angepasst werden. Stattdessen wird ein neues Annotations-Element in Zeile 4 erstellt und mit zusätzlichen Metadaten (Zeile 5) erweitert. In Zeile 6 werden letztendlich die bereits vorhandenen Annotationen kopiert. Wie bereits erwähnt ist die Reihenfolge wichtig, da später die Annotationen von vorne nach hinten durchsucht werden und nur der erste Treffer zählt.

Das Ergebnis aus der Transformation sieht man im Quelltext 28. Die mit AnnotationParameter definierten Metadaten sind durch den XSLT Prozessor und einem Präprozessor entstanden. In einer späteren Auswertung mit dem Postprozessor wird nur das erste Metadatum (Zeile 3) ausgewertet. D.h. der Postprozessor, der auf den Namensraumbezeichner **WPF_Plugin** hört, wird angewiesen das Element nicht zu ignorieren, obwohl zwei gegensätzliche Anweisungen existieren.

```
1 <ListBox Name="listBox1" Width="373" Height="137" SelectionMode="Extended">
2   <ann:Annotation.Attach>
3     <ann:AnnotationParameter Name="Ignore" CUri="WPF_Plugin">false</ann:AnnotationParameter>
4     <ann:AnnotationParameter Name="Ignore" CUri="WPF_Plugin">true</ann:AnnotationParameter>
5   </ann:Annotation.Attach>
6 </ListBox>
```

Quelltext 27 Vereinigung von Annotationen bei einer Transformation (mit doppelten Metadaten AnnotationParameter)

Das Präfix der Annotationen

In den Beispielen wurden Annotationen mit einem „ann“-Präfix versehen. Es ist zu beachten, dass die Präfixnamen für XAML und XSLT gleich sein sollten. Der Grund liegt im XSLT Prozessor. Unterscheiden sich Präfixe in XSLT und XAML, wird bei der Transformation das Präfix am Element Annotation.Attach definiert (siehe Quelltext 28). Dies ist jedoch nach den Regeln von XAML ein Fehler, so dass das Ergebnis nicht ohne Anpassung genutzt werden kann.

```
<annx:Annotation.Attach xmlns:annx="clr-namespace:Annotations;assembly=LATTEE_Annotations">
```

Quelltext 28 Notation mit Namensraumdefinition: Ungültiger Syntax für XAML

Ist eine Änderung des Präfixnamens für Annotationen notwendig (z.B. weil es bereits von dritter Seite verwendet wird), dann muss das neue Präfix in einem übergeordneten Element der Annotation definiert werden. Das kann z.B. das Steuerelement der angehängten Annotation oder auch das Fensterelement sein. Zum Einfügen des Präfixes mit Namensraum kann der Quelltext 17 aus Kapitel 7.5.2 genutzt werden. Natürlich ist es auch möglich den Präprozessor dafür einzusetzen. Soll der Präfix lediglich im transformierten XAML Quelltext anders lauten (z.B. newAnn), kann mit XSLT der Name automatisch geändert werden, indem ein Alias am Anfang

des XSLT Codes definiert wird. Dabei muss das neue Präfix im XSLT und im XAML Quelltext definiert werden. Dieser Aufwand ist jedoch nicht notwendig, wenn einfach das Standardpräfix verwendet wird, denn LATTE erzeugt automatisch das Präfix „ann“ für alle neuen Projekte.

```
<xsl:namespace-alias stylesheet-prefix="ann" result-prefix="newAnn"/>
```

Quelltext 29 Automatische Konvertierung von Präfixe in XSLT

Verwendung in Plug-Ins (C#) mit XmlDocument

Plug-Ins verwenden, wie bereits erwähnt, die Klasse XmlDocument, um Elemente zu transformieren. Annotationen unterscheiden sich bei der Nutzung mit XmlDocument nicht von anderen XML Elementen und sind daher genauso zu verwenden. Die folgenden Quelltexte geben einen Einblick, wie bestimmte Annotationen aus der XML Dokument gelesen werden können. Dazu wird jedes Beispiel innerhalb der `PostProcessing` Methode eines beispielhaften Plug-Ins ausgeführt.

```
public override bool PostProcessing(ref System.Xml.XmlDocument document,
                                    XmlNamespaceManager nameSpace)
```

Quelltext 30 Die Methode PostProcessing empfängt den vollständigen XML Baum sowie den XAML Namensraum. In der Methode können so die Anpassungen direkt am XML Dokument durchgeführt werden.

Die folgenden Beispiele erzeugen ein oder mehrere Annotationen als XML Knotenmenge. Die Ergebnisse sind dazu im Quelltext 35 entsprechend kenntlich gemacht worden, um den jeweiligen XPath Ausdruck besser verstehen zu können.

Beispiel 1: Die Annotation eines bestimmten Steuerelements auswählen

Häufig kann es notwendig sein, einfach alle Annotationen eines bestimmten, einzelnen Steuerelementes zu erfahren. Um z.B. die Annotation des Fensterelements in XAML als XmlElement zu erhalten, kann der folgende Quelltext verwendet werden. Die Ergebnismenge ist im Quelltext 35 mit dem Symbol ① (für Beispiel 1) gekennzeichnet und umfasst alle Zeilen mit dem gleichen Symbol.

```
XmlNodeList nodes = xmlDoc.SelectNodes(
                    "//xamlns:Window/*[local-name()='Annotation.Attach']", nameSpace);
```

Quelltext 31 Die Annotation eines bestimmten Elementes auswählen (Beispiel 1)

Beispiel 2: Die Annotationen aller Steuerelemente einer Art auswählen

Um die Annotationen zu erhalten, die innerhalb eines `StackPanel`s von bestimmten XAML Elementen getragen werden, kann der Ausdruck aus Quelltext 32 verwendet werden. Dies ist zum Beispiel nützlich, wenn mehrere Steuerelemente derselben Art transformiert werden sollen und der Vorgang von der jeweiligen Annotation abhängt.

```
"//xamlns:Window/xamlns:StackPanel/*[local-name()='RadioButton']
                                    /*[local-name()='Annotation.Attach']"
```

Quelltext 32 Die Annotationen aller Elemente einer Art auswählen (Beispiel 2)

Beispiel 3: Alle Annotationen innerhalb eines XAML Steuerelements ermitteln

Das folgende Beispiel ermittelt alle Annotationen innerhalb des Fenster-Elements. Damit werden nicht nur die Annotationen des Fenster-Elements, sondern auch die der untergeordneten Steuerelemente gefunden. Alle verfügbaren Annotationen einer XAML Struktur können so auf einen Schlag ermittelt werden.

```
//xamlns:Window/descendant::*[local-name()='Annotation.Attach']
```

Quelltext 33 Alle Annotationen innerhalb eines XAML Elements ermitteln (Beispiel 3)

Beispiel 4: Eine bestimmte Art von Annotation innerhalb von bestimmten Steuerelementen ermitteln

Kompliziertere XPath Ausdrücke ermöglichen auch direkt nach bestimmten Annotationen innerhalb von Annotation.Attach zu suchen.
Das folgende Beispiel ermittelt alle Annotationen vom Typ AnnotationVerb, die den Wert „Convert“ durch das Attribut Verb definiert haben und innerhalb eines Optionsfeldes (RadioButton) oder Labels stehen.

```
//xamlns:Window/xamlns:StackPanel/*[local-name()='RadioButton'
                or local-name()='Label']/child::*[local-name()='Annotation.Attach']
                /*[local-name()='AnnotationVerb' and @Verb='Convert']
```

Quelltext 34 Eine bestimmte Art von Annotation innerhalb von bestimmten XAML Elementen ermitteln (Beispiel 4)

Die ermittelten Annotationen müssen nicht weiter mit Hilfe von XmlNode oder anderen XML Klassen in Attribute und Inhalte zerlegt werden. Die Annotationen von LATTE unterstützen die Deserialisierung von XML Elementen durch die Methoden Deserialize und DeserializeByElement. Damit kann ein Element aus einem XPath Ergebnis direkt in eine Klasseninstanz verwandelt und eingesetzt werden, ohne weitere XPath Ausdrücke zum Ermitteln der anderen Werte (d.h. Typ, Name usw.) nutzen zu müssen.

Der Unterschied zwischen den Methoden Deserialize und DeserializeByElement liegt darin, welche Art von XML Elemente verwendet werden können. Während die Methode Deserialize einen angegebenen XML Knoten direkt als Annotation deserialisiert, kann bei DeserializeByElement der übergeordnete Elternknoten angegeben werden, um die Annotation zu erhalten. Dadurch ist es nicht notwendig, selbst den Typ des Annotationselements innerhalb eines XML Knotens zu suchen und zu deserialiseren. Neben dem Aufruf von Deserialize kann auch einfach das XML Element dem Konstruktor der Annotationsklasse übergeben werden.

Auf diese Weise kann das Ergebnis des ersten Beispiels „Die Annotation eines bestimmten Elementes auswählen“ mit der Hilfe der Klassenmethode AnnotationList.DeserializeByElement und allen darin eingeschlossenen Annotation in entsprechende Klasseninstanzen konvertiert werden. Es ist dabei zu beachten, dass die Annotation Annotation.Attach nicht vom Typ Annotation ist, sondern die Klasse AnnotationList darstellt, da die angehängte Eigenschaft Attach diesen Klassentyp besitzt. Daher muss DeserializeByElement aus der Klasse AnnotationList verwendet werden und nicht aus Annotation.

Der Quelltext 36 nutzt das Beispiel aus Quelltext 35, indem darin die Annotationsliste geladen wird (Zeile 4) und alle Annotationen des Typs AnnotationVerb aufgelistet werden (Zeile 6). Weiterhin unterstützt AnnotationVerb die automatische Ausführung des angegeben Verbs durch eine benutzerdefinierte Methode in .NET. Die Verben können im Plug-In ausgeführt werden, dadurch dass eine im Plug-In implementierte Methode mit AnnotationVerb.Invoke aufgerufen wird (Zeile 9). Die Methode erhält dann die in XML eingegebenen Parameter (hier Para1 und Para2 mit den Werten *A Text* und *1,00*). Der Methode Invoke wird dazu ein Objekt zugewiesen, welches die Methode mit dem im Quelltext angegebenen Namen Verb1 enthält. Ist keine Methode dieses Namens vorhanden oder können die Parameter nicht korrekt zugeordnet werden, d.h. die Parameterliste ist zu klein oder enthält unpassende Datentypen, so wird eine Ausnahme NotSupportedException erzeugt.

Treffermenge Nr. des Beispiels	Quelltext

```
          <Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                  xmlns:ann="clr-namespace:Annotations;assembly=LATTEE_Annotations"
                  xmlns:annc="clr-namespace:AnnotationsEx;assembly=AnnotationsEx">
① ③           <ann:Annotation.Attach>
① ③               <ann:AnnotationVerb Verb="Verb1" CUri="Example">
① ③                   <ann:AnnotationParameter Name="Para1" >A Text</ann:AnnotationParameter>
① ③                   <ann:AnnotationParameter Name="Para2"
                                               DataType="Double">1,00</ann:AnnotationParameter>
① ③               </ann:AnnotationVerb>
① ③           </ann:Annotation.Attach>
              <StackPanel>
                  <Label Name="label1"
                         Content="The following tasks are left? What do you want to do?" >
②③                    <ann:Annotation.Attach>
②③④                    <ann:AnnotationVerb Verb="Verb2">
②③④                     <ann:AnnotationParameter Name="Parameter1">Value1</ann:AnnotationParameter>
②③④                    </ann:AnnotationVerb>
②③                    </ann:Annotation.Attach>
                  </Label>
                  <RadioButton Name="Box1" Content="Save selected one">
②③                    <ann:Annotation.Attach>
②③                        <annc:AnnotationGroup GroupName="Radios" CUri="PlugEx"/>
②③                    </ann:Annotation.Attach>
                  </RadioButton>
                  <RadioButton Name="Box2" Content="Save them all">
②③                    <ann:Annotation.Attach>
②③                        <annc:AnnotationGroup GroupName="Radios" CUri="PlugEx"/>
②③                    </ann:Annotation.Attach>
                  </RadioButton>
              </StackPanel>
          </Window>
```

Quelltext 35 XAML Beispielquelltext mit markierten Ergebnismenge der XPath Ausdrücke aus den Beispielen 1 bis 4.

Die Standardeinstellung der Methode AnnotationVerb.Invoke ignoriert die Namen der Parameter, d.h. die Reihenfolge der Parameter im Quelltext muss mit der Reihenfolge der Methode (hier Verb1) in der angegebenen Klasseninstanz (Zeile 9, Invoke(this)) übereinstimmen. Durch die Eigenschaft AnnotationVerb.InvokeIgnoresParameterNames kann jedoch dieses Verhalten umgekehrt werden. Die Parameternamen im XAML Quelltext müssen dann mit den Namen der Methodenparameter (im Beispiel unten v1 und v2) übereinstimmen.

```
 1 if (nodes[0] != null)
 2 {
 3     AnnotationList ann = new AnnotationList();
 4     ann.Deserialize(nodes[0] as XmlElement); //oder dem Konstruktor übergeben
 5
 6     List<AnnotationVerb> verbs = ann.ListByType<AnnotationVerb>("Example");
 7     if (verbs.Count > 0)
 8     {
 9         String s = (verbs[0]).Invoke(this) as String;
10     }
11 }
```

Zusätzlich wurde diese Methode in derselben Klasse implementiert, um von Invoke aufgerufen werden zu können.

```
1 public String Verb1(String v1, Double v2)
2 {
3     return String.Format("{0} = {1}", v1, v2);
4 }
```

Quelltext 36 Annotationen können aus XML Dokument geladen werden. AnnotationVerb unterstützt die Ausführung von Methoden.

Das Laden von angehängten Annotationen aus XML gestaltet sich jedes Mal gleich. Das vorgestellte vierte Beispiel von Seite 97 erfordert jedoch einen anderen Typ von Annotation, weil das Ergebnis nun nicht mehr vom Typ AnnotationList (Annotation.Attach) ist, sondern von AnnotationVerb. Entsprechend muss die Deserialisierungsmethode Deserialize der AnnotationVerb Klasse aufgerufen werden, um eine Instanz erhalten zu können.

```
if (nodes[0] != null)
{
    AnnotationVerb annVerb = new AnnotationVerb ();
    annVerb.Deserialize(nodes[0] as XmlElement); //oder dem Konstruktor übergeben
    …
}
```

Quelltext 37 Entsprechend dem Typ des XML Elements müssen die richtigen Annotationsklassen verwendet werden.

Ein letzter Schritt bei der Nutzung von Annotationen mit XmlDocument besteht darin die Annotationen wieder zurück in die XAML Struktur zu schreiben. Die Annotationsklassen bieten dazu die Methode Serialize und SerializeByElement an. Die Instanzen der Annotationen zu verändern erfordert daher jedes Mal die Serialisierung derselben. Der folgende Quelltext 38 zeigt beispielhaft, wie eine Serialisierung von neuen, gelöschten oder angepassten Annotationen durchgeführt werden kann. Er basiert auf dem Quelltext 36, entfernt nun jedoch die Annotation Verb (Zeile 9) und fügt eine neue hinzu (Zeilen 11 bis 16). Genauso gut hätte die vorhandene Verb Annotation aber auch abgeändert werden können. Zuletzt wird SerializeByElement aufgerufen (Zeile 18), welches dafür sorgt die Annotation des Fenster-Elements (siehe Quelltext 35) zu entfernen, bevor eine neue hinzugefügt wird. So befindet sich immer genau eine angehängte Eigenschaft Annotation.Attach im übergeordneten Steuerelement (nodes[0].ParentNode). Serialize wird in diesem Fall intern von SerializeByElement aufgerufen, um die Annotationen zu speichern.

```
1  if (nodes[0] != null)
2  {
3      AnnotationList ann = new AnnotationList();
4      ann.Deserialize(nodes[0] as XmlElement);
5
6      List<AnnotationVerb> verbs = ann.ListByType<AnnotationVerb>("Example");
7      if (verbs.Count > 0)
8      {
9          ann.Remove(verbs[0]);
10     }
11     var verb = new AnnotationVerb();
12     verb.CUri = "Example";
13     verb.Verb = "Verb1";
14     verb.Parameters.Add(new AnnotationParameter { Content = "Value1" });
15     verb.Parameters.Add(new AnnotationParameter { Content = "Value2" });
16     ann.Add(verb);
17
18     ann.SerializeByElement(nodes[0].ParentNode as XmlElement);
19 }
```

Quelltext 38 Neue, wie auch veränderte Annotationen können zurück ins XML Dokument serialisert werden.

Da SerializeByElement sich um alles kümmert, sollte es bevorzugt für die Serialisierung von Annotationen genutzt werden. Die hier gezeigten Beispiele (1-4) funktionieren ohne Unterscheidung mit SerializeByElement, so dass sich jede Art von Annotationsklasse (auch selbst erstellte) wieder zurück in das XmlDocument schreiben lässt.

Verwendung des XAML Laders in Plug-Ins (C#)

Annotationen in XAML können, wie jedes andere XAML Element, durch den XAML Lader zur Klasseninstanzen transformiert werden. Da angehängte Eigenschaften nicht direkt einem Steuerelement angehören, müssen sie über die Methode GetAttach der Klasse Annotation aus einem XAML Steuerelement gelesen werden.

```
AnnotationList annotations = Annotation.GetAttach(anElement);
```

Quelltext 39 Mit Annotation.GetAttach können die Annotationen eines XAML Steuerelements ausgelesen werden.

Auf diese Art können Annotationen vollständig aus der XAML Struktur gelesen und zurück in Klasseninstanzen konvertiert werden. Existiert keine solch angehängte Annotation, liefert GetAttach **null** zurück. Es ist zu beachten, dass spezielle Annotationen, wie AnnotationVerb, nur über den oben beschriebenen Weg mit AnnotationList eingelesen werden können. Es ist z.B. nicht möglich, wie mit XPath, direkt ein AnnotationVerb Element zu erhalten. AnnotationList bietet jedoch Methoden an, um jede Art von Annotation zu finden. Neben den Methoden All, Any, AsEnumarble, ElementAt, First und Last, die von der Basisklasse ObservableCollection (siehe [MSDN, 2011]) bereitgestellt werden, wurden zusätzlich ListByHandler und ListByType in AnnotationList implementiert. ListByType wurde bereits in vorangegangenen Quelltexten vorgestellt. Daher wird, um die Vielseitigkeit von ListByHandler darzulegen, als Beispiel die Implementierung der Methode ListByType gezeigt.

```
public List<T> ListByHandler<T>(FindItemHandler<T> itemHandler)
where T : AnnotationBase;

public List<T> ListByType<T>(String CUri = null) where T : AnnotationBase
{
    return ListByHandler<T>((item) =>
        {
            return ((CUri != null) && (CUri == item.CUri)) || (CUri == null);
        });
}
```

Quelltext 40 Die Implementierung von AnnotationList.ListByType nutzt die generische Methode ListByHandler. Selbstgestaltete Methoden können genauso verfahren.

Zum Schluss muss erwähnt werden, dass Annotationen nicht mit der WPF Klasse XamlWriter zurück in den XAML Quelltext geschrieben werden können. Die Annotationen bleiben nicht erhalten, stattdessen bleibt einfach das Steuerelement in XAML zurück. Die genauen Gründe konnten auch nach gründlicher Recherche nicht herausgefunden werden. Vermutlich wirken sich die im Abschnitt „Nutzen eines XAML Laders“ (Seite 89) besprochenen Nachteile der Klasse XamlWriter auch bei den Annotationen aus. Darum unterstützen die Annotationsklassen die Serialisierung mit XmlDocument und setzen nicht alleine auf XamlReader und XamlWriter.

Annotationen mit benutzerdefinierten Eigenschaften erweitern

Falls die vorhandenen Annotationsklassen (Tabelle 6) nicht ausreichen sollten, können diese erweitert werden. Es ist möglich von jeder der vorhandenen Annotationsklassen eine neue Klasse abzuleiten, zu ergänzen und mit LATTE einzusetzen. Die Basis aller Klassen bleibt jedoch immer AnnotationBase, welche die notwendigen Eigenschaften (Namensraum, Uri und Name) implementiert.

Vor der Erstellung einer Annotation sollte entschieden werden, welche Attribute die Annotation unterstützt und ob das XAML Element einen einzelnen oder mehrere Inhalte umfassen kann. Der Inhalt ist dabei der innere Text eines XML Elements, d.h. zwischen dem öffnenden und schließenden XML Tag (z.B. <tag>Inhalt</tag>). Dieser kann entweder aus einem reinen Text bestehen oder weitere XAML Elemente enthalten. Beispielsweise kann die Annotationsklasse AnnotationParameter nur einen Text enthalten, während AnnotationList weitere Annotationen als Inhalt unterstützt.

Im folgenden Beispiel soll eine neue Annotation erstellt werden, die ein Steuerelement mit anderen Steuerelementen zu einer Gruppe zusammenfügt. Dies kann z.B. die Migration von bestimmten Elementen erleichtern, die zusammengefasst ein neues Element bilden sollen. Der folgende Quelltext 41 zeigt die Anwendung von Annotationen an mehreren Optionsfeldern, die später in einem Plug-In erkannt und entsprechend den

vorherigen Abschnitten dieses Kapitels verarbeitet werden könnte. Auf die Umsetzung des Plug-Ins wurde jedoch verzichtet.

```
1  <RadioButton Name="Box1" Content="Save selected one">
2    <ann:Annotation.Attach>
3      <annc:AnnotationGroup GroupName="Radios" CUri="PlugEx"/>
4    </ann:Annotation.Attach>
5  </RadioButton>
6  <RadioButton Name="Box2" Content="Save them all">
7    <ann:Annotation.Attach>
8      <annc:AnnotationGroup GroupName="Radios" CUri="PlugEx"/>
9    </ann:Annotation.Attach>
10 </RadioButton>
11 <RadioButton Name="Box3" Content="Release them all">
12   <ann:Annotation.Attach>
13     <annc:AnnotationGroup GroupName="Radios" CUri="PlugEx"/>
14   </ann:Annotation.Attach>
15 </RadioButton>
```

Quelltext 41 Mehrere Optionsfelder werden durch eine selbsterstellte Annotation in einer Gruppe zusammengefasst.

Um neue Annotationen für LATTE zu erstellen, ist als erster Schritt erforderlich eine neue .NET Assembly zu erstellen, welche die Annotationsklassen aus Assembly LATTEE_Annotations nutzt. Außerdem muss der .NET Namensraum Annotations für den Zugriff auf Klassen eingebunden werden (mittels *using* Direktive). Im Beispiel wurde dazu in XAML der Namensraum annc eingeführt, der auf die Assembly und deren Namensraum wie gewohnt verweist.

Die neue Annotationsklasse AnnotationGroup wird von der Basisklasse AnnotationBase abgeleitet und um die Eigenschaft GroupName erweitert. Wie in Quelltext 41 zu sehen ist, wurde die Annotation auch mit der Eigenschaft CUri ausgestattet. Es ist jedoch zu beachten, dass CUri aus der verwendeten Basisklasse stammt und nicht implementiert werden muss.

Die neue Eigenschaft GroupName kann, wie jede .NET Eigenschaft, mit Setter und Getter definiert werden. Objektinstanzen wie z.B. String und andere Klassen müssen jedoch von Anfang an mit einem Wert initialisiert werden, da sonst die Deserialisierung fehlschlagen könnte (Verwendung eines null Wertes). Dies geschieht üblicherweise im Konstruktor der Annotationsklasse, jedoch wird mit der in WPF eingeführten Abhängigkeitseigenschaft DependencyProperty ein mächtigeres Mittel für XAML zur Verfügung gestellt. Denn diese Art von Eigenschaft lässt sich schon bei ihrer Deklaration initialisieren und auf Änderungen überwachen. Weitere Informationen dazu bietet das Microsoft Developer Network unter [Microsoft, 2011] (siehe *Dependency Properties Overview*). In Quelltext 42 wird eine Instanz der Klasse DependencyProperty in Kombination mit FrameworkPropertyMetadata erstellt (Zeile 9), um die Eigenschaft GroupName mit leerem Inhalt zu erstellen. Die Deklaration von GroupNameProperty ist zwar statisch, also für alle Klassen gleich, doch intern bekommt jede Klasseninstanz ihren eigenen Eigenschaftswert zugeteilt. Der Wert kann folglich über die Setter und Getter in Zeile 3 und 4 gelesen und verändert werden.

```
1  public String GroupName
2  {
3    get { return (String)GetValue(GroupNameProperty); }
4    set { SetValue(GroupNameProperty, value); }
5  }
6  public static DependencyProperty GroupNameProperty =
7                  DependencyProperty.Register("GroupName",
8                     typeof(String), typeof(AnnotationGroup),
9                     new FrameworkPropertyMetadata(""));
```

Quelltext 42 Attribute von Annotationen können gewöhnliche Eigenschaften sein oder über das WPF System mit Dependency Eigenschaften verwirklicht werden.

Weiterhin muss die neue Klasse eine Serialisierung und Deserialisierung für die neuen Eigenschaften implementieren. Dies gilt jedoch nur, wenn die Annotation in einem Plug-In mit der Klasse XmlDocument gelesen und geschrieben werden soll. Ein XAML Lader verwendet den hier vorgestellten Weg nicht.

Die neue Annotationsklasse kann mit XmlDocument verwendet werden, indem die Methoden Serialize und Deserialize der Basisklasse AnnotationBase überschrieben werden. AnnotationBase bietet die zusätzlichen Hilfsmethoden SerializeAttribute und DeserializeAttribute an, um Eigenschaften in oder aus einem XML Knoten zu schreiben oder zu lesen. Es ist außerdem immer notwendig die gleichnamigen Vorgängermethoden der Basisklasse aufzurufen, damit die De-/Serialisierung vollständig durchgeführt werden kann. Um die korrekten Namensräume und Präfixe in der XAML Struktur kümmert sich letztendlich AnnotationBase (siehe Abschnitt *Das Präfix der Annotationen*). Die Umsetzung ist daher, wie der untenstehende Quelltext zeigt, kurz.

```
1  public override XmlElement Serialize(XmlElement parent)
2  {
3      var node = base.Serialize(parent);
4      SerializeAttribute(node, GroupNameProperty);
5      return node;
6  }
7  public override void Deserialize(XmlElement node)
8  {
9      base.Deserialize(node);
10     DeserializeAttribute(node, GroupNameProperty);
11 }
```

Quelltext 43 Die Methoden Serialize und Deserialize müssen überschrieben werden, um die Eigenschaften der Annotation zu speichern.

Die Annotationsklasse AnnotationGroup ist nun bereit, in einem XAML Dokument verwendet werden zu können. Auf diese Art können auch weitere Eigenschaften hinzugefügt werden, um noch mehr Funktionalität zu erhalten. Weiterhin kann eine einzelne Eigenschaft des Elements als Inhaltseigenschaft des XML Elements definiert werden. Dadurch wird der Wert (z.B. eine weitere Annotation) zwischen den Start- und Endetags des XML Elements automatisch dieser Eigenschaft zugewiesen. Dies ist schnell realisiert, indem das Attribut ContentPropertyAttribute an die neue Annotationsklasse angehängt wird. Die Klasse AnnotationVerb in LATTE besitzt beispielsweise die Eigenschaft Parameters als Inhaltseigenschaft, um weitere Annotationselemente aufnehmen zu können.

```
[ContentPropertyAttribute("Parameters")]
public class AnnotationVerb : AnnotationBase
{
 …
    public ObservableCollection<AnnotationParameter> Parameters
    {
        get { … }
        set { … }
    }
    public static readonly DependencyProperty ParameterProperty =
        DependencyProperty.Register("Parameters",
        typeof(ObservableCollection<AnnotationParameter>),typeof(AnnotationVerb),
        new PropertyMetadata());
 …
```

Quelltext 44 AnnotationVerb unterstützt Inhalte durch das Attribut ContentPropertyAttribut

7.6 Erweiterbarkeit durch das Plug-In-System

Mit LATTE wurde ein Plug-In System entwickelt, das auf dem Managed Addin Framework (MAF) von Microsoft (*Add-ins and Extensibility* [Microsoft, 2011]) aufsetzt. MAF ermöglicht das Spezifizieren von Schnittstellen zur

Kommunikation zwischen der Anwendung LATTE und Plug-Ins. Plug-Ins für LATTE werden dabei in sogenannten Assemblys untergebracht, welche DLL Dateien aus Windows ähneln. LATTE lädt diese Dateien jedoch nicht direkt, sondern kommuniziert über die Grenze der eigenen Anwendungsdomäne (*Application Domain* [Microsoft, 2011]) mit den Plug-Ins. Die Ausführung der Plug-Ins erfolgt daher in einem separaten Prozess, um die Hauptanwendung nicht zu beeinflussen (z.B. durch unerwünschte Speichermanipulationen) oder gar zum Absturz zu bringen.

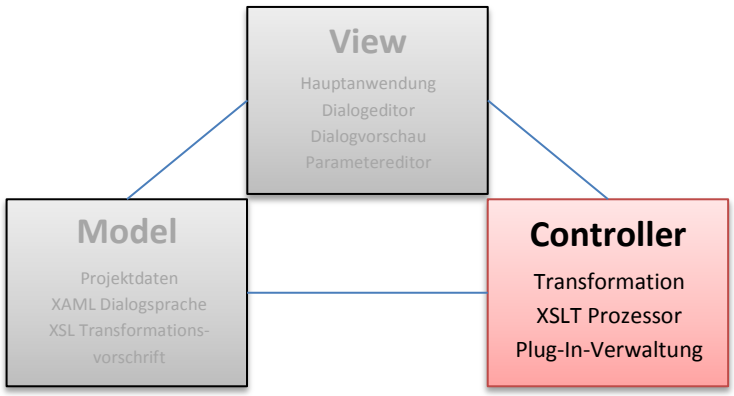


Abbildung 62 Die Controller Komponente der MVC Architektur wird durch Plug-Ins erweitert

Neben dem Sicherheitsaspekt besitzt MAF den für LATTE wichtigen Vorteil, dass Plug-Ins geladen und auch wieder entladen werden können. Normalerweise können Assemblys, die mit .NET geladen wurden nicht mehr entladen werden; d.h. der Entladezeitpunkt kann nicht selbst festgelegt werden, sondern wird von der .NET Laufzeitumgebung bestimmt. Da die Transformationseinheit jedoch die Plug-In Dateien nach einer Transformation freigeben muss – der Benutzer könnte sich entscheiden, eine Änderung am Plug-In vorzunehmen – ist es notwendig, dass die Plug-Ins auf jeden Fall nicht mehr im Speicher vorhanden sind. Mit MAF können die DLL Dateien ersetzt, kopiert oder gelöscht werden, weil MAF jedes Plug-In in einen eigenständigen .NET Prozess lädt. Der eigenständige Prozess wird nach der Transformation beendet und die Plug-In Dateien dadurch freigegeben.

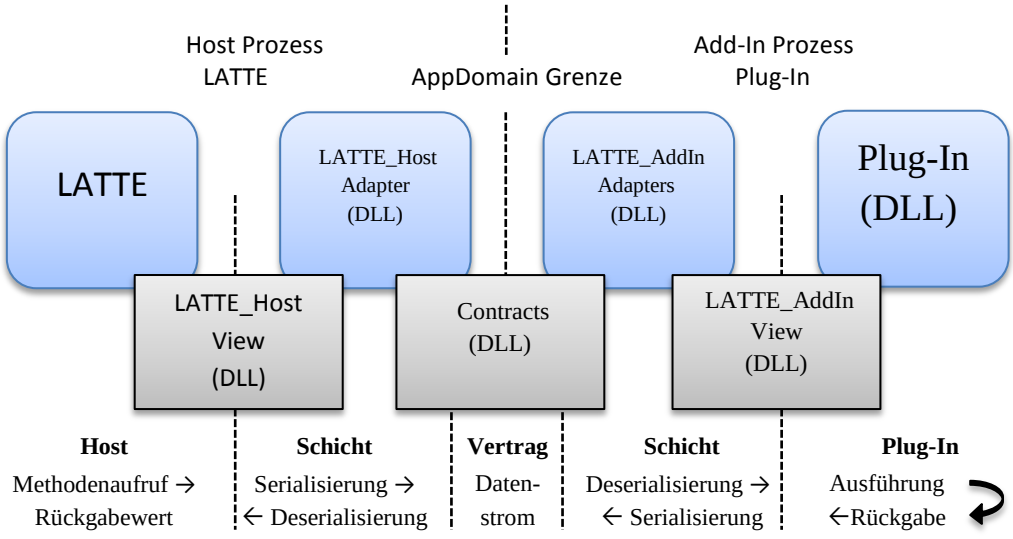


Abbildung 63 Die Add-In Pipeline in LATTE (angepasst aus [MacDonald, 2010]) und die Ausführung eines Methodenaufrufs durch die Schichten

Der Nachteil dieser Vorgehensweise besteht in der aufwändigen Umsetzung der Kommunikation. Die Methodenaufrufe können nicht mehr einfach umgesetzt werden, weil sie nun über eine Prozessgrenze hinweg stattfinden. Die Parameter und auch der Rückgabewert müssen daher zuerst serialisiert, d.h. in einen Datenstrom umgewandelt werden. In der anderen Richtung wird aus dem Datenstrom wieder ein Objekt mit Typinformation (z.B. ein Integer) erstellt, d.h. die Information wurde deserialisiert (Abbildung 63). Zu diesem Zweck setzt MAF auf ein System aus Komponenten, das die Kommunikation zwischen der Anwendung und den Plug-Ins übernimmt. Die möglichen Methoden für Kommunikation werden dazu in sogenannten Verträgen (engl. contracts) festgelegt, die einfache Schnittstellenbeschreibungen (in C# *interface*) sind. Die Kommunikationspartner benutzen diese Schnittstellen jedoch nicht direkt, sondern kommunizieren über eine jeweils eigene Schicht zur Anwendungsgrenze hin. LATTE und Plug-Ins implementieren in ihrer jeweiligen Schicht die sogenannten View- und Adaptorenklassen. Während die View-Klassen dem Benutzer den Schein eines normalen Objekts vorspielen, konvertieren die Adaptorenklassen die Methodenparameter in oder aus einem Datenstrom.

In LATTE wird MAF durch die Visual Studio Projekte LATTE_Host.View und LATTE_Host.Adapter auf der Host-Anwendungsseite umgesetzt und durch LATTE_AddIn.View sowie LATTE_AddIn.Adapters für Plug-Ins zur Verfügung gestellt. Die Schnittstellendeklarationen der Verträge sind im Projekt Contracts definiert.

7.6.1 Neues Plug-In erstellen

Ein neues Plug-In wird schnell und einfach hergestellt, indem das Visual Studio Projekt LATTEC_WPF als Vorlage verwendet wird. Die Kopie kann dann nach Belieben angepasst werden. Dazu wird die Implementierung der Datei **PluginImpl.cs** verändert. Zudem sollten die Assemblyinformationen in den Projekteinstellungen sowie die Annotation `AddIn` der Klasse `PluginImpl`, d.h. Name, Version, Beschreibung und Autor geändert werden.

Jedes neue Plug-In muss drei Vorrausetzungen erfüllen, damit es mit LATTE verwendet werden kann:

1. Die Add-In Assembly muss auf die folgenden Assemblys verweisen:
   - **LATTE_AddIn.View** und
   - **LATTE_AddIn.Adapters** für die Kommunikation
   - **LATTE_SharedPlugin** zur Nutzung und Austausch von gemeinsam genutzten Datenstrukturen (ProcessingMessage, Properties)
   - **System.Addin** zur Einbindung des MAF
2. Die folgenden Standardklassen müssen abgeleitet und implementiert werden:
   - *ProcessingAddInView* (AddInView.cs)
   - *PluginAddInView* (AddInView.cs)

3. Das Plug-In muss einen eindeutigen und von anderen Plug-Ins unterscheidbaren Namen und Namensraum über die Methoden GetName() und GetNameSpace() der Klasse PluginAddInView zurückliefern.

7.6.2 Anpassung der Kommunikation

Das Hinzufügen oder Ändern von Methoden oder Schnittstellen im Vertrag erfordert die Anpassung aller genannten Projekte, also Contracts, LATTE_Host.View, LATTE_Host.Adapter, LATTE_AddIn.View und LATTE_AddIn.Adapters. Außerdem ist es notwendig, dass die Typen der Methodenparameter, die komplexe Objekte darstellen (z.B. `XmlDocument`) in den Adaptoren sowie in der Contracts Assembly als .NET Klasse Stream deklariert werden. Einfache Klassentypen, wie Integer oder String können von MAF automatisch serialisiert und deserialisert werden und müssen daher nicht als Stream deklariert werden. Nur die View Klassen erhalten die tatsächlichen Parametertypen wie `XmlDocument`.
Die Adaptorenklassen übernehmen die Serialisierung und Deserialisierung der Parameter. Für eigene komplexe Objekte muss entsprechend die Datenstruktur in einen Datenstrom konvertiert werden, um für die Kommuni-

kation eingesetzt werden zu können. Für die Klassen XmlDocument, XmlNamesapceManager und PropertyList (benutzt in InitProperties(), siehe Quelltext 6) wurden dazu bereits Routinen geschrieben, die sich in der Klasse Converters in der Assembly **Contracts** befinden (Namensraum Contracts.Shared). Die darin enthaltenen statischen Methoden konvertieren u.a. XML Objekte in einen Datenstrom und wieder zurück. Die Konvertierungsmethoden unterstützen auch den **null** Zustand eines Objektes und können diesen über die Grenze der Anwendungsdomäne erhalten. Dieser Anwendungsfall ist zu beachten, da **null** sonst nicht serialisiert werden kann.

7.7 Diskussion

In den vorangegangenen Kapiteln wurde die Umsetzung der Anwendung LATTE und insbesondere der Transformationsprozess vorgestellt. Darin wurde auch der Prototyp LATTE erläutert, der aus drei Bestandteile nach dem MVC Muster besteht: der Oberfläche, der Transformationseinheit und dem Plug-In System. Diese Umsetzung ermöglicht die Transformationen von Dialogen mit XSLT sowie mit einer .NET Programmiersprache, z.B. in C#. Für die Kommunikation entlang der Transformationspipeline werden Annotationen in den XAML Quelltext eingefügt, welche die Darstellung des Dialogs nicht beeinträchtigen. Stattdessen können so Präprozessoren und XSLT die nachfolgenden Prozessoren XSLT und Postprozessor beeinflussen und sogar steuern. Das Ergebnis ist ein für berührungsempfindliche Bildschirme angepasster Dialog.

Die Kapitel der Umsetzung konnten allerdings nur einen Teil der Grundlagen über den Transformationsprozess behandeln. Der bestehende Prototyp kann in einigen Bereichen verbessert werden, um die Transformation noch mächtiger werden zu lassen. Diese Verbesserungen sollen hier geschildert werden.

Eine erste Verbesserung ist die Unterstützung für die hinter den Dialogen liegende Programmierlogik oder Quelltexte. Für neue oder ausgetauschte Dialogelemente kann es manchmal erforderlich sein, dass auch eine Programmlogik eingebaut werden muss. Mit dem aktuellen Prototyp ist das noch nicht möglich. Doch mit den Plug-Ins sollte dies kaum ein Problem darstellen. Aber auch mit XSLT ist es denkbar, dass externe Funktionen die Möglichkeit bereitstellen, die Programmlogik anzupassen. Besonders mit partiellen Klassen, d.h. Klassen, die über mehrere Quelldateien verstreut liegen können, ist diese Umsetzung möglicherweise leicht zu bewerkstelligen.

Ein weiterer Ansatzpunkt für Verbesserungen ist die eingesetzte XSLT Sprache. Für die Durchführung wurde das XSLT Framework von Microsoft eingesetzt, welches die XSLT Version 1.0 von 1999 unterstützt. Diese Version ist bereits ausreichend für den Transformationsprozess, da sie Turin-vollständig ist. Doch mit der neuen Version kommen auch zahlreiche Erleichterungen für den Entwickler wie beispielsweise mehrere Ausgabedokumente und benutzerdefinierte Funktionen für XPath Ausdrücke. Jedoch sieht Microsoft es bis heute nicht für notwendig an, die neue Version 2.0 von 2007 (siehe [W3C, 2007]) zu implementieren: *„As for XSLT 2.0 - we've heard from customers and understand the improvements in XSLT 2.0 over XSLT 1.0, but right now we're in the middle of a big strategic investment in LINQ and EDM for the future of the data programming platform [..]. But we are always re-evaluating our technology investments so if your readers want to ramp up their volume on XSLT 2.0 please ask them to drop us a line with their comments. "* [Lovett, 2006]. Stattdessen wird auf andere Hersteller verwiesen wie z.B. das teilweise proprietäre SAXON.NET [Saxonica Limited, 2010]. Der Einsatz von SAXON.NET in LATTE wurde jedoch verworfen, da der Implementierungsaufwand mit SAXON höher eingeschätzt wurde als der mit dem Microsoft .NET Framework. Außerdem setzt SAXON vollständig auf JAVA, so dass für Aufrufe aus einer .NET Umgebung (wie C# Anwendungen) zuerst eine Java Laufzeitumgebung gestartet werden muss, um die Methoden und Klassen von SAXON nutzen zu können.

Auch wenn XSLT in Version 1.0 verwendet wird, ist es möglich die Sprache durch eigene Funktionen zu erweitern, wie es im Quelltext 24 mit LATTE:ConvertBoolean demonstriert wurde. Dadurch können komplizierte Vergleich oder String-Manipulationen vereinfacht werden. Wem das nicht reicht, der kann mit EXSLT (siehe [Stewart, et al., 2006]) weitere XSLT Elemente einsetzen, die noch mehr Möglichkeiten bieten. Natürlich können auch eigene XSLT Elemente erstellt und verwendet werden.

Eine weitere Möglichkeit besteht darin, Plug-In als Ersatz für XSLT zu verwenden, z.B. wenn man sich noch nicht gut genug mit XSLT auskennt. Dies ist jedoch auf lange Sicht nicht zu empfehlen, da Änderungen am Transformationsprozess in LATTE nur mühsam über eine separate Entwicklungsumgebung wie Visual Studio erreicht werden können. Stattdessen sollte XSLT erst einmal zum Experimentieren verwendet werden und einige wenige Abbildungsvorschriften ausprobiert oder die Vielzahl von Möglichkeiten von XSLT und XPath ausgelotet werden. Für besonders komplizierte oder aufwändige Transformationen kann zuletzt immer noch auf reinen .NET Code (z.B. C#, VB# oder Delphi Prism) zurückgegriffen werden und XSLT zur Steuerung der Plug-Ins verwendet werden.

Annotationen sind ein mächtiges Mittel, um Plug-Ins und damit den Transformationsprozess steuern zu können. Die in dieser Arbeit vorgestellten Annotationen bilden die Grundlage für zukünftige Transformationen und können durch Erweiterungen, d.h. neue Annotationsklassen, für Plug-Ins vielseitig einsetzbar gemacht werden. Dies gilt besonders, weil Plug-Ins spezielle Aufgaben übernehmen können, um beispielsweise nur eine Art von Steuerelement zu transformieren. Für diese Spezialfälle könnten auch benutzerdefinierte Annotationen eingesetzt werden, welche die Transformation innerhalb des Plug-Ins regeln. Doch Plug-Ins können mit der Hilfe von Annotationen als eine Art Bibliothek für XSLT dienen. Mit AnnotationVerb.Invoke sind so beliebige Methoden eines Plug-Ins über die Annotation AnnotationVerb innerhalb des XSLT Quelltexts aufrufbar.

Wie bereits erwähnt gestaltet sich das Schreiben von XAML Strukturen schwierig. Microsoft unterstützt zwar das Einlesen von XAML, das Schreiben in XAML wird jedoch nur oberflächlich unterstützt (siehe Abschnitt *Nutzen eines XAML Laders* auf Seite 89). Daher wurde mehr Wert auf den Umgang mit XmlDocument innerhalb von Plug-Ins gelegt als auf den XAML Lader. Trotzdem bietet XAML den Vorteil, die Dialoge ohne großen Aufwand grafisch darzustellen. In LATTE wurde dies benutzt, um eine Vorschau des transformierten Dialogs zu ermöglichen. Dazu wurde der Dialogdesigner des quelloffenen SharpDevelop Projekts verwendet, welcher zusätzliche Funktionen besitzt, um die Steuerelemente zu verschieben und deren Größe anzupassen. Für das Projekt wurden diese Möglichkeiten jedoch deaktiviert, da eine vollständige Integration des Designers zu viel Zeit gekostet hätte. Ein bedeutender Vorteil des SharpDevelop Editors für LATTE ist jedoch, dass er Fenster- und XAML Seitenelemente (<Window> und <Page>) auf die gleiche Art und Weise darstellt. Der XAML Lader von Microsoft kann zwar Seitenelemente innerhalb eines Fensters darstellen, dagegen können Fensterelemente selbst nur als eigenständige Fenster außerhalb der LATTE Umgebung dargestellt werden.

Letztendlich kann man erkennen, dass die vorgestellten Umsetzungen noch etwas Feinschliff erfordern, um in der Produktion eingesetzt werden zu können. Doch der Autor dieser Zeilen denkt, dass die vorgestellten Grundlagen und Konzepte die künftigen Aufgaben gut lösen können. Zum Schluss und weil die Prä-, Post- und XSLT Prozessoren das Herz der Transformationspipeline darstellen, soll hier noch eine kurze Übersicht zu deren Vor- und Nachteilen (Tabelle 8) gegeben werden.

	Vorteile	Nachteile
Prä- und Post-prozessor (Plug-Ins)	• Kann mit beliebiger .NET Sprache geschrieben werden • Syntax von C# oder VB.NET ist einfach zu verwenden	• Prozessor muss bei Änderungen neu in eine DLL-Datei kompiliert und in den Plug-In Ordner kopiert werden. • Das Laden und Ausführen von Plug-Ins vor dem Transformationsprozess dauert einige Zeit.
XSLT Prozessor	• Mächtige Transformationssprache • Turing-vollständig • Kann direkt im Editor erstellt und in der Transformation verwendet werden. Änderungen sind daher schnell möglich.	• Mit XPath muss eine weitere Sprache gelernt werden • Einige Probleme sind nur umständlich zu lösen, dadurch dass XSLT regelbasiert ist. • Durch .NET Implementierung beschränkt auf XSLT 1.0 Syntax.

Tabelle 8 Vor- und Nachteile der Prozessoren

8 Technische Evaluation anhand der Durchführung einer Touch-Studie

To measure is to know.

James Clerk Maxwell
schottischer Physiker

Dieses Kapitel beschäftigt sich mit einer im Rahmen der Diplomarbeit durchgeführten Benutzerstudie, um die Effektivität der beschriebenen Lösungsmethode und Umsetzung zu untersuchen. Über die Durchführung der Studie und deren Ergebnisse wird im Kapitel 8.2.2 und 8.2.3 berichtet. Eine Diskussion der Ergebnisse hält das Kapitel 8.2.4 bereit.

8.1 Motivation

Es existiert eine Vielzahl von Experimenten, die den Einsatz von berührungsempfindlichen Bildschirmen in verschiedenen Umgebungen und Situationen prüfen. Eines der ersten experimentellen Vergleiche zwischen Maus und berührungsempfindlichen Bildschirm wurde von Ben Shneidermann (vgl. [Sears, et al., 1991]) durchgeführt. Seitdem befassen sich viele Untersuchungen auch mit Themen wie der Präzision der Fingereingabe (u.a. [Gleeson, et al., 2004], [Schmidt, 2008 S. 34ff.] und [Holz, et al., 2011]), mit Gesteninteraktion (vgl. [Matejka, et al., 2009] und [Mauney, 2010]) oder mit der Bewertung, wie Formen auf einem berührungsempfindlichen Bildschirm präsentiert und manipuliert werden können (u.a. [Anslow, 2010] und [Raschke, et al., 2010]). Es konnte jedoch keine Studie gefunden werden, die ganze Dialoge (insbesondere Standarddialoge wie Öffnen und Drucken) von handelsüblichen Desktopsystemen wie Windows für die Touchbedienung untersucht.

Zudem sollte die in dieser Diplomarbeit gezeigte Lösung sowie Umsetzung, d.h. die Transformationsmethode (Kapitel 4) und der Prototyp (Kapitel 7), überprüft werden. Dass die Umsetzung bereits Dialoge transformieren kann, wurde schon während der Umsetzung in Kapitel 7.5.2 („Grundlagen der Transformation“) gezeigt. Es war jedoch zusätzlich notwendig zu prüfen, ob auf diese Art die Dialoge mit den Fingern überhaupt bedienbar werden.

8.2 Studie

In einer Benutzerstudie sollte zuerst die Bedienung von Standarddialogen wie „Datei öffnen“ oder Druckseiteneinstellung mit den Fingern untersucht werden. Diese Dialoge werden für gewöhnlich sehr oft im Alltag genutzt, so dass man bei einem Einsatz von berührungsempfindlichen Bildschirmen dort ansetzen sollte. Außerdem wurde eine Auswahl von den vorgeschlagenen Steuerelementen aus Kapitel 5 für die Transformation verwendet. Für jeden Dialog wurde eine Reihe von üblichen Tätigkeiten erstellt, die die Teilnehmer durchzuführen hatten, wie z.B. eine Datei zu öffnen oder eine Einstellung vorzunehmen. Anschließend bewerteten die Teilnehmer die Dialoge und Steuerelemente mit einem Fragebogen.

8.2.1 Hypothesen

Vor der Durchführung der Studie wurden die folgenden Hypothesen aufgestellt, um prüfen zu können, ob eine Verbesserung durch die vorgestellte Lösung (Kapitel 4, 5 und 7) aufgetreten ist:

1. **Es ist ausreichend die Eigenschaften von einzelnen Elementen zu verändern, um die Elemente mit den Fingern besser bedienen zu können.** Mit dieser These soll geprüft werden, dass die Änderung von Eigenschaften wie Größe und Abstände von bestehenden Steuerelementen eine einfache und schnelle

Verbesserung für die Fingerbedienung darstellt. Für eine Fingerbedienung muss es also nicht unbedingt erforderlich sein, Dialoge vollständig neu zu gestalten oder gar NUI zu generieren.

2. **Es ist ausreichend Steuerelemente in einem Dialog auszutauschen oder zu erweitern (mit zusätzlichen Elementen), um diesen Dialog besser mit den Fingern bedienen zu können.** Mit dieser These soll überprüft werden, dass überall dort, wo es nicht ausreicht Eigenschaften von Steuerelementen zu ändern, diese Elemente durch neue oder verbesserte Versionen ersetzt werden können.
3. **Die eingeführten Steuerelemente, als Ersatz vorhandener Elemente, sind für die Nutzung mit den Fingern geeignet.** Damit soll gezeigt werden, dass die in Kapitel 5 eingeführten Steuerelemente als Ersatz für die Standardelemente in Dialogen dienen können und so die Bedienung des Dialogs mit den Fingern erleichtern. In dieser Studie wurden allerdings aus Zeitgründen nur die folgenden Steuerelemente verwendet: das vergrößerte Kontrollkästchen (Kapitel 5.4), das numerische Tastenfeld (5.5), die Brotkrumennavigation (5.6) und die Navigationsleiste für Listenfelder (5.7).

Daraus lässt sich folgern, dass wenn diese Hypothesen bestätigt werden, die in der Umsetzung vorgestellte Transformation ausreichend ist, um Steuerelemente in Dialogen semi-automatisch an berührungsempfindliche Bildschirme anzupassen.

8.2.2 Methode

Die Studie wurde als ein between-subjects-Design mit sechs Teilnehmern entworfen. Die Teilnehmer mussten die ihnen gestellten Aufgaben an einem berührungsempfindlichen Bildschirm erledigen und darauf folgend Fragen zu ihren Präferenzen beantworten. Zudem wurden während der Durchführung die Tätigkeiten sowie Gesprochenes durch den Studienleiter (d.h. der Autor) aufgenommen.

Experimenteller Aufbau und Umgebungsbedingungen

Die Studie wurde in einem Privatraum durchgeführt, der während den Aufgaben von äußerlichen Ablenkungen abgeschottet wurde. Die Dialoge wurden für die Bedienung auf einem 23 Zoll großen und berührungsempfindlichen TFT Bildschirm mit einer Auflösung von 1920x1080 Pixeln durchgeführt. Es handelte sich dabei um einen Bildschirm der Marke Acer, den T230H Breitbildschirm mit Multi-Touch (zwei Berührungen gleichzeitig). Die im Gerät verbaute Technik nutzt ein Raster aus Infrarotstrahlen, das über die Bildschirmscheibe gelegt ist. Diese Technik wurde bereits im Kapitel 2.3.1 („Technologien“) im Unterabschnitt „Optische Bildschirmoberflächen“ erläutert. Weiterhin wurde der Bildschirm von seinem mitgelieferten Standfuß getrennt, um eine bessere Bedienung zu ermöglichen. Dazu wurde der Bildschirm auf einem Untergrund aus Polystyrol gebettet und auf einen Neigungswinkel von knapp 35 Grad gebracht. Dieser Winkel wurde als für den Aufbau ideal befunden, da ein kleinerer Winkel die Lichtspiegelung an der Bildschirmscheibe verstärkt und ein größerer Winkel die Integrität des Polystyrols gefährdet hätte. Für die Hand- und Armablage wurde eine handelsübliche Tastaturmatte den Teilnehmern zur Nutzung überlassen. Zudem wurde für zwei Aufgaben eine Maus benötigt, die deshalb entsprechend gestellt wurde.

Während die Teilnehmer die Aufgaben bearbeiteten, wurde der Bildschirm mit einem Videoprogramm aufgenommen, um später die Berührungen zu zählen und die Treffer von den „Vertippern“ unterscheiden zu können. Jede Berührung konnte in der Aufnahme als kleiner werdender Kreis wahrgenommen werden. Zudem wurde den Teilnehmern vorgegeben, dass sie ihre Tätigkeit kommentieren und Erfolge, Probleme, Unstimmigkeiten oder Ähnliches mitteilen. Die Kommentare der Teilnehmer wurden während der Aufgabe in das Bildschirmvideo integriert, so dass sie mit den Tätigkeiten auf dem Bildschirm direkt in Beziehung gebracht werden konnten.

Abbildung 64 Der Aufbau des berührungsempfindlichen Bildschirms für die Benutzerstudie. Hinten rechts kam ein Bildschirm zur Darstellung des aktuellen Aufgabentextes und zur Ablaufkontrolle zum Einsatz. Daneben liegt ein Fragenkatalog, den die Teilnehmer für die Bearbeitung beantworteten.

Die Aufnahme wurde durch den Leiter gestartet, nachdem der Teilnehmer die Aufgabe gelesen und dies bestätigt hatte. Die Aufnahme wurde nach der Aufgabe manuell beendet, als der Teilnehmer den Dialog, wie von der Aufgabenstellung verlangt, durch Betätigen des Ok Schalters beenden wollte. Der Dialog wurde dadurch jedoch nicht geschlossen, sondern es wurde nur die Aktion durch einen Nachrichtendialog bestätigt. Der Teilnehmer konnte so auch noch im Anschluss der Aufgabe den Dialog betrachten und Erinnerungen für die Befragung wachrufen.

Teilnehmer

Für die Evaluation wurde eine Studie durchgeführt, die sechs Teilnehmer zwischen 20 und 60 Jahren absolvierten. Der Altersdurchschnitt lag bei 36 Jahren. Von den Teilnehmern waren zwei weiblich und vier männlich. Dies wurde jedoch nicht als Einflussfaktor gesehen. Die Studienteilnehmer gaben an, dass sie keinen berührungsempfindlichen Bildschirm besäßen, sondern alle nur Mobilgeräte (Smartphone, Navigationsgeräte) oder Kiosksysteme mit den Fingern bedient hätten. Jedoch waren alle gut mit der Maus und Tastatur vertraut und kannten sich mit der Oberfläche von Windows 7 aus. Als Dankeschön für die Absolvierung der Studie bekamen die Teilnehmer am Ende eine süße Aufmerksamkeit geschenkt.

Durchführung

Zuerst mussten die Teilnehmer einen Fragebogen ausfüllen, der sie zu ihrem Alter und ihren Mediengewohnheiten wie Nutzungsdauer von PC sowie dem Vorwissen zu berührungsempfindlichen Technologien (Smartphone, Tablet usw.) befragte. Außerdem wurden die Teilnehmer bereits vorher befragt, was sie von der Bedienung eines berührungsempfindlichen Bildschirms erwarteten.

Vor dem Beginn der Aufgabendurchführung durften sich die Teilnehmer mit dem Bildschirm vertraut machen. Dazu wurde die Anwendung Surface Collage aus dem Microsoft Touch Pack für Windows 7 gestartet. Mit dieser Software konnten die Teilnehmer sich auf die kommenden Aufgaben einstimmen, so dass bereits bei der ersten Aufgabe ein Lerneffekt auftreten konnte. Die Anwendung Surface Collage hatte jedoch nichts mit den Aufgaben zu tun, so dass die Teilnehmer nicht wissen konnten, was sie erwartet. Die Übungsdauer lag zwischen fünf und zehn Minuten.

Anschließend wurden sie über den anstehenden Aufgabenverlauf informiert. Dazu wurde ihnen auf einem zweiten Bildschirm ein Browserfenster in ausreichender Größe dargestellt, das sie über den Studienablauf aufklärte. Dieses Fenster wurde außerdem über die gesamte Dauer der Studie für die zu bearbeitende Aufgabenbeschreibung verwendet. Die Teilnehmer konnten die Aufgabe daher wiederholt lesen, während sie die Aufgabe bearbeiteten. Während der Aufgaben und der Beantwortung der Fragen konnten die Teilnehmer zu jeder Zeit zum aktuellen Fragebogen, zur Aufgabenstellung oder zum Aufgabenablauf Fragen stellen, um Probleme oder Fehldeutungen zu vermeiden. Die Teilnehmer konnten jederzeit eine Pause einlegen oder sogar die Studie abbrechen, was sie jedoch beides nicht taten.

Die Aufgaben liefen stets gleich ab. Zuerst wurde auf dem Webseitenfenster die neue Aufgabe präsentiert, die die Teilnehmer sofort lesen durften. Dazu wurde auf dem berührungsempfindlichen Bildschirm der zur Aufgabe gehörende Dialog positioniert. Damit sich die anfängliche Position niemals änderte, wurde der Desktophintergrund von Windows durch ein Bild mit weißen Rechtecken ersetzt. Die Dialoge wurden mit ihrer linken, oberen Ecke näherungsweise in diese Rechtecke mit der Maus positioniert. Die Rechtecke besaßen dazu die Ausmaße der entsprechenden Dialoge. Den Teilnehmern war es zwar erlaubt, die Dialoge zu verschieben, jedoch machte keiner davon Gebrauch. Weiterhin waren die Rechtecke auf dem Desktop so platziert, dass die Dialoge die rechte untere Hälfte des Bildschirms beanspruchten, so dass sie mit der rechten Hand auf der Ablage benutzt werden konnten. Für Linkshänder wurden die Dialoge an der Mittelachse auf die andere Bildschirmseite gespiegelt.

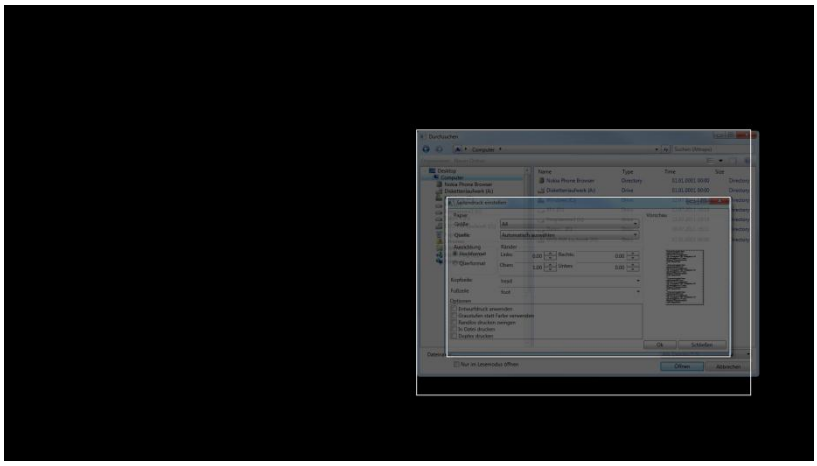


Abbildung 65 In der Studie verwendeter Desktophintergrund für die einheitliche Positionierung der Dialoge an weißen Rechtecken.

Nachdem die Teilnehmer Bereitschaft meldeten, konnten sie die eigentlichen Aufgaben angehen. Die ersten Aufgaben nutzten einen Nachbau des Standarddialogs „Datei öffnen“ aus Windows, der mit WPF und XAML erstellt wurde. Der Nachbau ermöglichte Änderungen einzelner Steuerelemente, die in den Aufgaben entsprechend eingesetzt wurden. Im Folgenden werden die für die Studie entwickelten Aufgaben erläutert. Die Aufgaben wurden dazu in zwei Teile aufgeteilt, in denen zwei unterschiedliche Arten von Dialoge untersucht wurden. In den ersten sieben Aufgaben sollten die Teilnehmer die Bedienung eines Standard Öffnen-Dialogs von Windows 7 bewerten, während im zweiten Teil die Teilnehmer in neun Aufgaben einen Seiteneinstellungsdialog für Drucker bedienen sollten. Beide Dialoge enthielten unterschiedliche Arten von Steuerelementen. Während der Öffnen-Dialog mit Listen und Navigationsleisten ausgestattet war, wurde der Seiteneinstellungsdialog mit Elementen wie Dropdown-Listenfeld, Drehfeldern und Kontrollkästchen aufgebaut. Einzelne Dialoge wurden entsprechend der Aufgabenstellung angepasst, um verschiedene Steuerelemente und Eigenschaften zu prüfen. Außerdem wurde gemäß den Hypothesen verschiedene Schaltergrößen, Listenelementgrößen und Steuerelementgrößen eingesetzt.

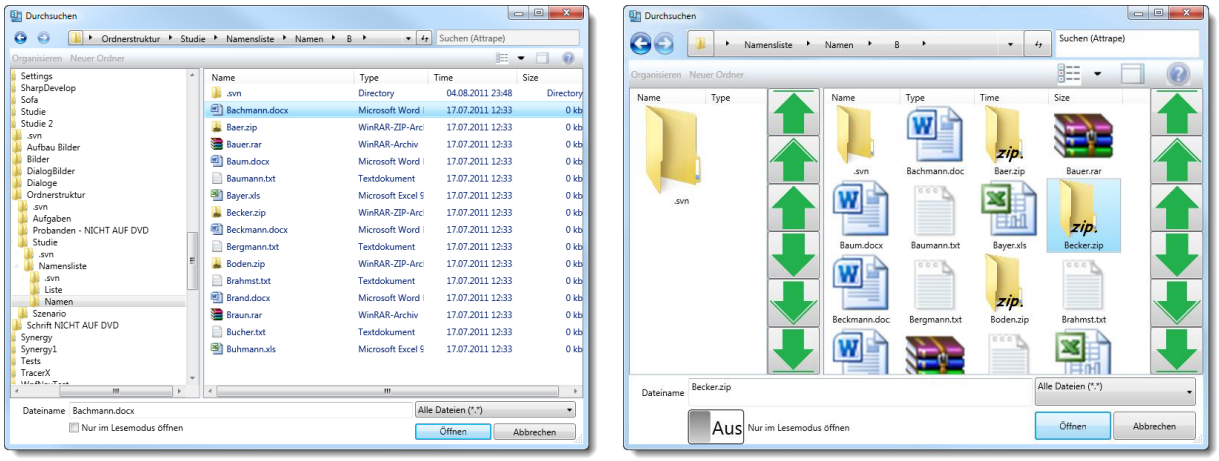


Abbildung 66 Vergleich der in der Studie eingesetzten Dialoge. Links: eine Nachbildung des originalen Öffnen-Dialogs für Ö1 und Ö2; Rechts: der transformierte Öffnen-Dialog für die Aufgaben Ö3 bis Ö7.

Die **Aufgabe 1** (in der Studie Ö1 genannt) und **Aufgabe 2 (Ö2)** unterschieden sich nicht in der Aufgabenstellung, sondern nur in der Bedienung (Abbildung 66). Die Teilnehmer sollten dazu jeweils eine Datei öffnen, die sich unterschiedlich tief in der Ordnerhierarchie befand. Die Aufgabe konnte dadurch gelöst werden, indem mit der Maus und später mit den Fingern die Ordnernamen angeklickt bzw. getippt wurden, um so die Datei sichtbar zu machen. Ein Vergleich bei gleichem Dialog mit unterschiedlichen Eingabearten sollte prüfen, ob und in wie weit die aktuellen Dialoge eine Schwierigkeit für die Fingereingabe darstellen.

Die **Aufgabe Ö3** präsentierte den Teilnehmern das erste Mal eine angepasste Version des Öffnen-Dialogs. Die Teilnehmer sollten dazu eine bereits sichtbare Datei im Listenfenster markieren und öffnen (durch Doppeltippen oder durch den Öffnen-Schalter). Die Aufgabe wurde für den Einstieg sehr einfach gehalten. So konnten sich die Teilnehmer an die ungewöhnlich großen Dateisymbole (2,5 x 2,5 Zentimeter) und Schaltflächen gewöhnen.

Die nächste **Aufgabe Ö4** erweiterte die vorangegangene Aufgabe, indem eine Datei geöffnet werden sollte, die nicht sichtbar war. Dazu mussten die Teilnehmer ohne das Vorhandensein einer Bildlaufleiste die Datei in den sichtbaren Bereich der Liste bringen, um diese zu öffnen. Diese Aufgabe sollte einen Vergleich mit der nächsten **Aufgabe Ö5** erbringen, in der für das Blättern in der Liste eine Navigationsleiste vorgeschrieben war. Den Teilnehmern war es freigestellt, welche Pfeile sie von der Navigationsleiste nutzen konnten.

Ein weiterer Vergleich wurde mit den **Aufgaben Ö6** und **Ö7** angestrebt. Die Teilnehmer sollten dazu zu einem bestimmten Ordner navigieren, der oberhalb des aktuellen Ordners in der Hierarchie lag. Dafür sollte die Brotkrumennavigation genutzt werden. Jeder übergeordneter Ordner wurde dabei als Schalter in der Leiste der Brotkrumennavigation angezeigt und konnte so durch einfaches Antippen erreicht werden. Der Unterschied zwischen Ö6 und Ö7 bestand darin, dass in Ö6 die Teilnehmer nicht über die Brotkrumennavigation aufgeklärt wurden, im Gegensatz zu Ö7. Außerdem wurde in beiden Aufgaben die Baumansicht der Ordner durch eine einfache Liste mit den Unterordner des aktuellen Ordners ersetzt, um eine einfachere Fingerbedienung durch eine Liste zu erhalten (vgl. Kapitel 5.6).

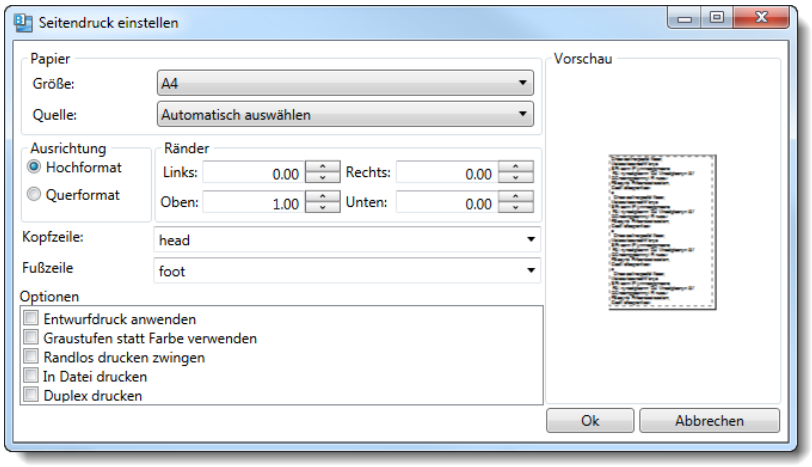


Abbildung 67 Standarddialog für die Einstellung einer Druckerseite, wie er unter Windows eingesetzt werden kann.

Mit den **Aufgaben S1** und **S2** begann der zweite Teil der Studie, in der ein selbst gebauter Dialog zur Einstellung der Druckseite eingesetzt wurde. Die beiden Aufgaben sollten wie in Ö1 und Ö2 prüfen, ob der Dialog mit den Fingern besser oder schlechter zu bedienen ist. Die Aufgabenstellung war entsprechend ähnlich und unterschied sich, um einen Lerneffekt zu verhindern, nur in Einzelheiten wie z.B. den Papiergrößen (A2, COM-10 usw.), den Größen für die Ränder (z.B. 1,5 Zentimeter) und die Kombination von aktivierten Kontrollkästchen.

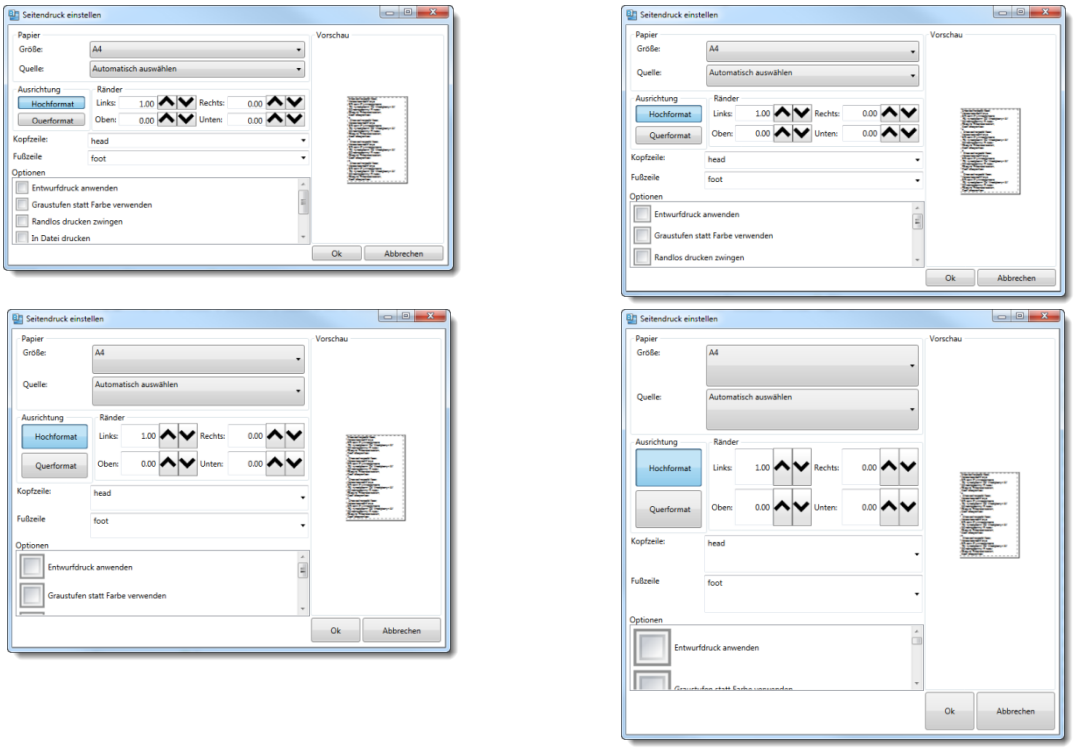


Abbildung 68 Vergleich der transformierten Dialoge aus den Aufgaben S3 bis S6 (v.l.n.r. und v.o.n.u.)

Die folgenden **Aufgaben S3 bis S6** stellten den Teilnehmer Steuerelemente von verschiedenen Größen bereit. Die Dialoge unterschieden sich in größer werdenden Kontrollkästchen, Dropdown-Listenfeldern, Options- sowie Drehfeldern. Die Ausmaße betrugen dabei jeweils 0,5cm, 0,7 cm, 1,0cm und zuletzt 1,5cm. Außerdem wurden wie in S2 die einzugebenden Werte (Papiergröße, Ränder und Optionen) geändert, um einen Lerneffekt zu verhindern.

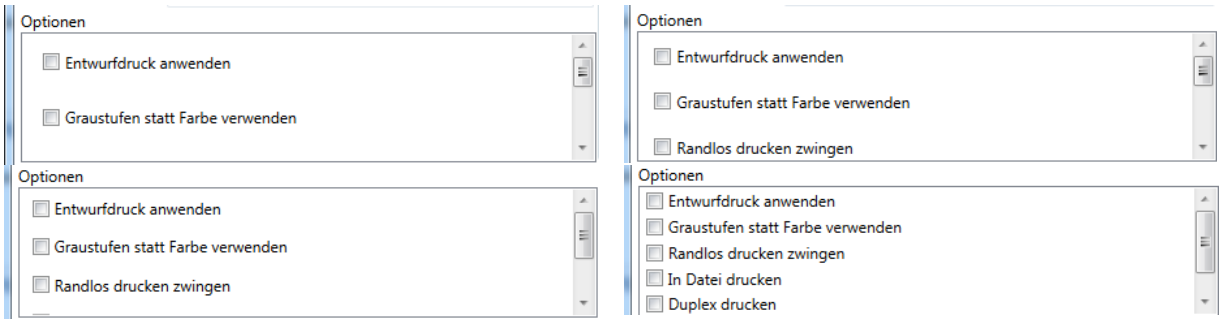


Abbildung 69 Vergleich der Abstände zwischen den Kontrollkästchen in Aufgabe S7 Teil 1 bis 4 (v.l.n.r. und v.o.n.u.)

Als nächste **Aufgabe S7** wurden die Abstände zwischen Kontrollkästchen der Optionenliste verändert, um deren Einfluss auf die Bedienbarkeit zu überprüfen. Die Abstände der Kontrollkästchen untereinander wurden dazu in vier Teilaufgaben beginnend bei 35mm über 25mm und 17,5mm auf 5mm verkleinert. Zusätzlich mussten die Teilnehmer in jeder Aufgabe andere Kontrollkästchen aktivieren.

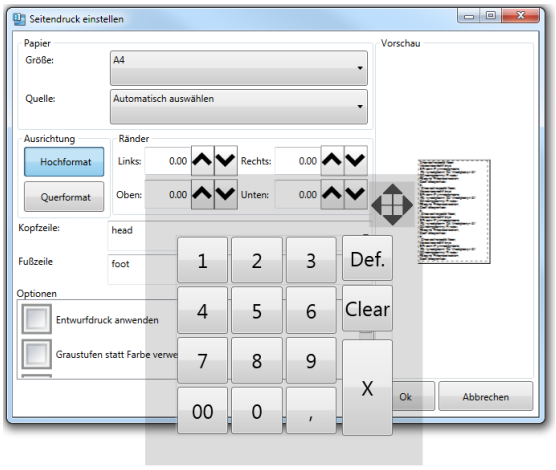


Abbildung 70 Eingesetzter Prototyp eines numerischen Tastenfeld für die Aufgaben S8 und S9

Die letzten **Aufgaben S8** und **S9** nutzten das in Kapitel 5.5 vorgestellte numerische Tastenfeld. Die Teilnehmer nutzten dieses Feld statt den Drehfeldern, um die Ränder einer Druckseite einzustellen. Die Aufgaben unterschieden sich nur durch verschiedene Werte der Ränder. Auf diese Weise sollte überprüft werden, ob ein Lerneffekt bessere Ergebnisse bei dem Tastenfeld erreichen kann oder ob das Tastenfeld auch intuitiv gut zu bedienen ist.

Zuletzt wurden zu jeder Aufgabe zwei bis vier Fragen gestellt, um einen subjektiven Eindruck der Bedienung des Dialogs und der eingesetzten Steuerelemente zu bekommen. Die Fragen konnten die Teilnehmer durch das Vergeben von einem bis acht Punkten auf einer Likert-Skala beantworten. Jeder Teilnehmer konnte so jeweils bis zu acht Punkte in sechs Kategorien verteilen. Die Fragen können im Anhang eingesehen werden. Die Ergebnisse der Fragen werden auf Seite 119 besprochen.

8.2.3 Ergebnisse

Für die durchgeführte Studie wurden sechs Teilnehmer eingeladen. Die Zahl der Teilnehmer lässt daher nur eine qualitative Auswertung der Ergebnisse zu. Trotzdem kann die Studie einen Einblick in die Welt der Fingerbedienung an Desktop- und Laptopsystemen geben.

Weiterhin werden nicht alle Ergebnisse dargestellt, um den Rahmen dieser Arbeit nicht zu sprengen. Deshalb werden die Ergebnisse entweder zusammengefasst oder nur die ergiebigsten und interessantesten aufgezählt.

Die vollständigen Daten können aus dem Studiendokument auf der beiliegenden CD entnommen werden.

Aufgabenausführungsdauer

Von jedem Teilnehmer wurde die Ausführungsdauer gemessen. Der Startzeitpunkt war dabei die erste Bedienung, d.h. Klick oder Berührung, eines Steuerelements. Das Ende der Zeitmessung wurde durch das Bestätigen des Dialogs (Schalter OK oder Abbrechen) ausgelöst.

Die Zeitdauer über alle Aufgaben bieten die Tabellen 9 und 10 unten. Für jede Aufgabe wurden zudem der Mittelwert und Median angegeben. Es zeigte sich, dass die Teilnehmer sehr unterschiedlich an die Aufgaben herangingen und deshalb die Ausführungszeiten innerhalb derselben Aufgabe stark variieren. Die Tabellen zeigen außerdem, dass die älteren Teilnehmer allgemein sich etwas länger Zeit gelassen haben oder benötigten, um die Aufgabe zu erledigen.

Aufgaben / Probanden (Alter)	Ö1	Ö2	Ö3	Ö4	Ö5	Ö6	Ö7
1 (20-30 Jahre)	12	23	2	22	39	23	49
2 (20-30 Jahre)	19	10	17	12	24	55	90
3 (20-30 Jahre)	6	24	1	20	26	63	46
4 (50-60 Jahre)	29	23	9	18	21	239	77
5 (20-30 Jahre)	20	28	4	15	15	31	44
6 (50-60 Jahre)	22	32	6	40	35	39	62
Mittelwert	18,0	23,3	6,5	21,2	26,7	75,0	61,3
Median	19,5	23,5	5,0	19,0	25,0	47,0	55,5
Min/Max	6/29	10/32	1/17	12/40	15/39	23/239	44/90

Tabelle 9 Benötigte Ausführungsdauer der Teilnehmer für die Aufgaben Ö1-Ö7 in Sekunden

Aufgaben / Probanden	1	2	3	4	5	6	7	8	9
1 (20-30 Jahre)	35	112	66	36	33	53	15	47	37
2 (20-30 Jahre)	35	46	39	52	44	34	22	18	37
3 (20-30 Jahre)	37	61	62	39	70	60	18	44	44
4 (50-60 Jahre)	53	74	91	51	54	84	18	53	43
5 (20-30 Jahre)	34	55	45	37	51	40	16	20	31
6 (50-60 Jahre)	44	89	73	48	55	73	25	55	38
Mittelwert	39,7	72,8	62,7	43,8	51,2	57,3	18,7	39,5	38,3
Median	36,0	67,5	64,0	43,5	52,5	56,5	18,1	45,5	37,5
Min/Max	34/53	46/112	39/91	36/52	33/70	34/84	14,5/24,5	18/55	31/44

Tabelle 10 Benötigte Ausführungsdauer der Teilnehmer für die Aufgaben S1-S9 in Sekunden

Einige Aufgaben wurden außerdem so konzipiert, dass deren Ausführungsdauer direkt miteinander verglichen werden konnten. Dazu zählen jeweils die Aufgaben 1 und 2 des Öffnen- und Seiteneinstellungs-Dialogs. Zusätzlich wurden die Aufgaben 3 bis 6, die vier Teile der Aufgabe 7 und 8 bis 9 des Seiteneinstellungsdialogs für einen Vergleich erstellt.

Die Ergebnisse zeigen, dass die Bedienung von Standarddialogen mit den Fingern länger dauern kann. Während die Ausführungsdauer der Aufgabe Ö2 (mit den Fingern) noch nahe an der Dauer der Aufgabe Ö1 (mit der Maus) liegt, ist der Zeitunterschied zwischen den Aufgaben S1 und S2 bereits deutlicher.

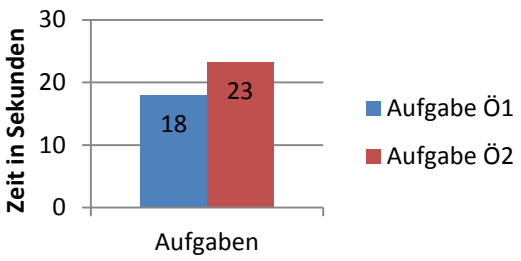


Abbildung 71 Vergleich der durchschnittlich benötigten Zeit der Aufgaben Ö1 (Maus) und Ö2 (Touch)

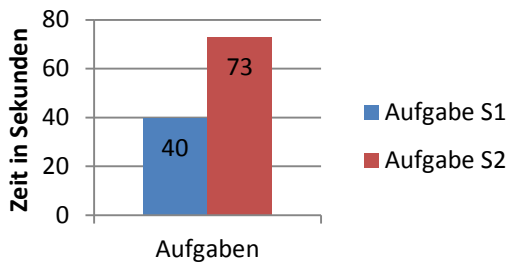


Abbildung 72 Vergleich der durchschnittlich benötigten Zeit der Aufgaben S1 (Maus) und S2 (Touch)

Die Aufgaben S3 bis S4 unterschieden sich nur durch die Größe der einzelnen Elemente (0,5cm; 0,7cm; 1,0cm und 1,5cm). Dadurch entstanden teils stark unterschiedliche Ausführungszeiten der Teilnehmer wie es in Abbildung 73 unten zu sehen ist. In Aufgabe 7 wurden, wie bereits erläutert, unterschiedliche Abstände zwischen Kontrollkästchen untersucht. Die Abbildung 74 zeigt, dass die Teilnehmer die Einstellung in der Liste, umso schneller vornehmen konnten, je kleiner der Abstand wurde (35mm, 25mm, 17,5mm und 5mm). Dies endete erst mit der vierten und letzten Aufgabe, als die Abstände so gering waren, dass die Teilnehmer benachbarte Kontrollkästchen versehentlich berühren konnten.

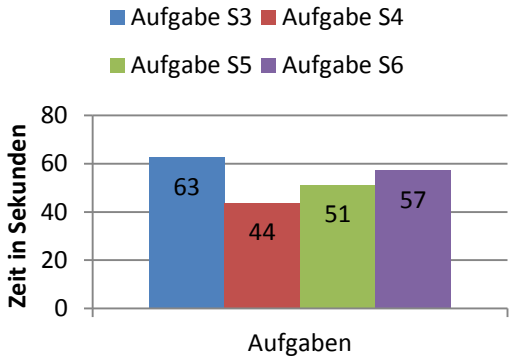


Abbildung 73 Durchschnittlich benötigte Zeit der Aufgaben S3 bis S6

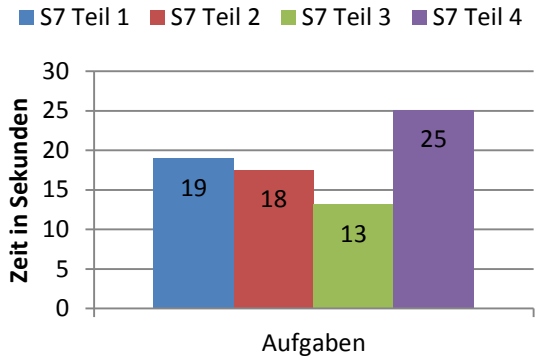


Abbildung 74 Durchschnittlich benötigte Zeit der Aufgaben S7 Teil 1 bis Teil 4

Genauigkeit

Die Genauigkeit der Teilnehmer wurde bestimmt, indem die aufgetretenen Fehler gezählt wurden. Dazu wurden die gemachten Bildschirm- und Audioaufnahmen der Teilnehmer zur Beurteilung herangezogen. Ein Fehler wurde dazu folgendermaßen definiert:

1. **Ein Steuerelement wird nicht getroffen.** Stattdessen wird der Dialoghintergrund oder ein benachbartes Steuerelement getroffen. Die Absicht ein bestimmtes Element zu treffen ergibt sich dabei aus der Aufgabenstellung. Beispielsweise sollte ein bestimmter Wert mit den beiden Schaltern des Drehfelds eingestellt werden. Die Teilnehmer mussten dazu den Aufwärtsschalter betätigen. Eine Berührung wurde als Fehler gezählt, wenn stattdessen der Abwärtsschalter getroffen wurde. Die Audioaufnahmen halfen zudem leichter bestimmen zu können, ob eine bestimmte Bedienungstätigkeit absichtlich durchgeführt wurde. Die Teilnehmer kommentierten solche Fehlaktionen in vielen Fällen mit entsprechenden Bemerkungen.
2. **Ein Doppeltippen wird nicht erkannt.** Die Aufgaben (insbesondere Ö1 bis Ö7) konnten alle ohne ein doppeltes Tippen abgeschlossen werden. Trotzdem nutzten alle Teilnehmer ein Doppeltippen beispielsweise um eine Datei zu öffnen. Wurde das Tippen zu schnell oder zu langsam ausgeführt oder

fanden beide Berührungen nicht innerhalb eines 1 Zentimeter Radius statt, dann wurde dies als ein Fehler gewertet. In diesem Fall wurde das Doppeltippen als zwei einzelne Berührungen bewertet und von der UI erkannt.

Die Fehlerrate ist der Quotient aus den gesamten Berührungen und den Fehlern. Entsprechend ist die Trefferrate der Kehrwert des Quotienten Q_F.

$$Q_F = \frac{Fehlerzahl}{Zahl\ der\ Berührungen}$$

Abbildung 75 Der Fehlerquotient berechnet sich aus der Anzahl der gemachten Fehlern und der Zahl der Berührungen

Ein Vergleich der Aufgaben Ö1 mit Ö2 sowie S1 mit S2 zeigt deutlich die Probleme aktueller Dialoge, wenn sie mit den Fingern bedient werden. Während in den jeweils ersten Aufgaben Ö1 und S1 noch die Maus benutzt werden durfte, wurde derselbe Dialog im zweiten Teil Ö2 und S2 nur noch mit den Fingern bedient. Dabei stiegen nicht nur die gemachten Fehler, sondern auch die Gesamtzahl der Berührungen, da die Teilnehmer die Fehler korrigieren mussten. Während die meisten Teilnehmer bei der Bedienung des Öffnen-Dialogs noch die wenigsten Probleme hatten, änderte sich dies beim Bedienen des Seiteneinstellungsdialogs mit Standardsteuerelementen. Die Abbildung 76 sowie Abbildung 77 stellen den Fehlerquotient aufgeschlüsselt in Fehler- und Berührungsgesamtzahl dar. Ö1 und S1 wurden jeweils mit der Maus durchgeführt; Ö2 und S2 jeweils mit dem Finger.

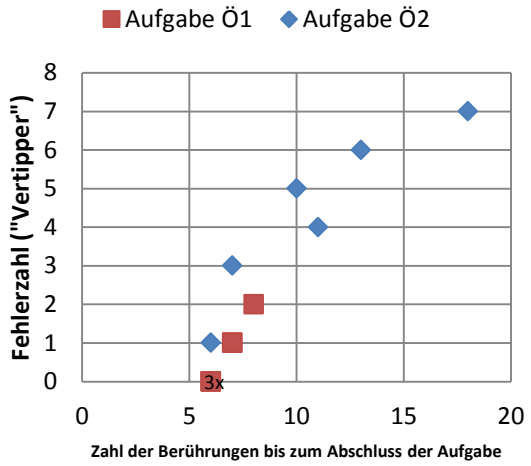


Abbildung 76 Fehler pro Gesamtberührungen der Aufgaben Ö1 (mit Maus) und Ö2 (mit Finger)

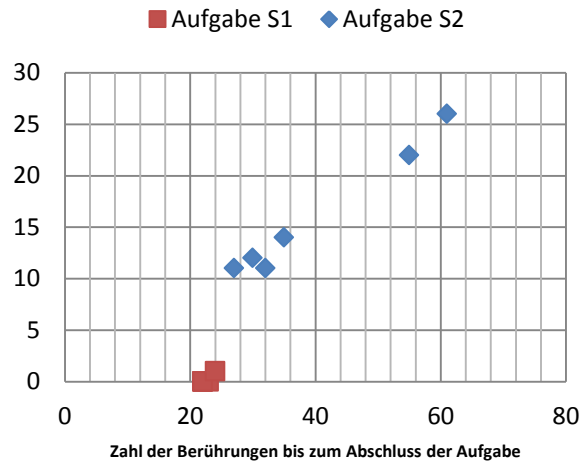


Abbildung 77 Fehler pro Gesamtberührungen der Aufgaben S1 (mit Maus) und S2 (mit Finger)

Die nachfolgenden Aufgaben wurden nur noch mit den Fingern durchgeführt. Trotzdem kam kein Teilnehmer mehr an die hohen Fehlerraten der Aufgaben Ö2 (◆) heran (Abbildung 78). In Ö4 (■) und Ö5 (▲) sollten die Teilnehmer eine Datei analog zu Ö2 (◆) in einem Unterordner öffnen. Während sie in Ö4 (■) die Liste direkt mit den Fingern verschieben durften, um die entsprechenden Ordnereinträge sichtbar zu machen, sollten die Teilnehmer in Ö5 (▲) die Navigationsleiste aus Kapitel 5.7 benutzen. Dabei traten gehäuft zwischen einem und zwei Fehler auf. Außerdem gab es auch fehlerlose Durchgänge, die nicht auf dem direkten Weg zum Ziel kamen, sondern z.B. länger nach der geforderten Datei suchen mussten. Daher sind die Symbole ▲ und ■ auf der X-Achse entsprechend verteilt.

Einen weiteren Vergleich der Genauigkeit zeigt die Abbildung 79 für die Aufgaben S2 bis S6. Man kann erkennen, dass die Fehlerzahl ab der Aufgabe S2 (◆) bis zur Aufgabe S5 (✕) zuerst zurückgeht, um dann bei der letzten Aufgabe S6 (✳) wieder einen Sprung nach oben macht. Die Aufgaben S4 (▲) und S5 (✕) sind dabei kaum voneinander zu unterscheiden. Die meisten fehlerlosen Durchgänge (3) wurden in Aufgabe S5 (✕) erreicht. Dagegen machten die Teilnehmer bei Aufgabe S4 (▲) noch mindestens einen Fehler.

	Fehlerrate Ø
Aufgabe Ö2	38%
Aufgabe Ö4	19%
Aufgabe Ö5	9%

	Fehlerrate Ø
Aufgabe S2	40%
Aufgabe S3	17%
Aufgabe S4	8%
Aufgabe S5	2%
Aufgabe S6	11%

Tabelle 11 Die Fehlerraten der Aufgaben Ö2, Ö4 und Ö4 sowie S2 bis S6 im direkten Vergleich

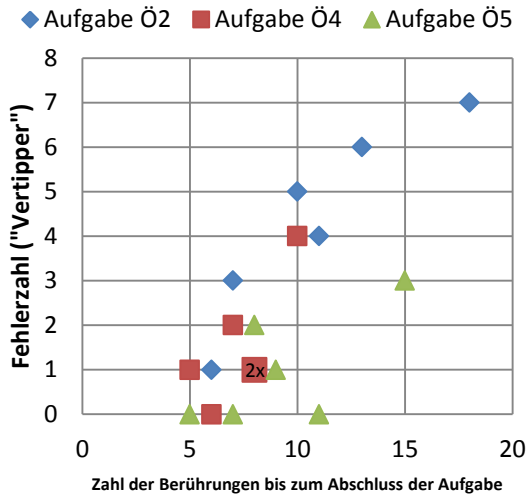


Abbildung 78 Vergleich der Fehlerquotienten der Aufgaben Ö2, Ö4 und Ö5

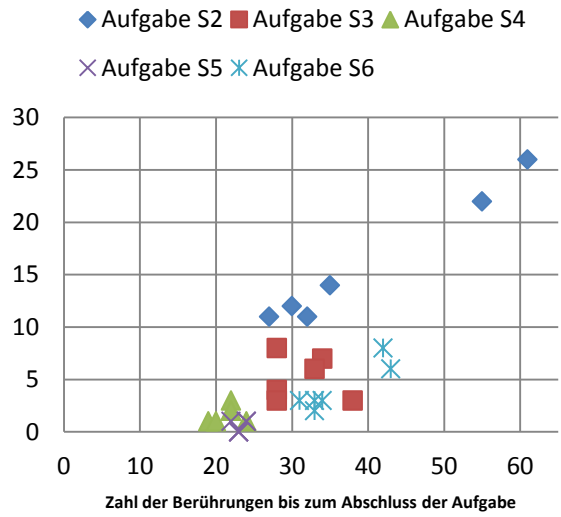


Abbildung 79 Vergleich der Fehlerquotienten der Aufgaben S2 bis S6

Noch deutlicher beschreiben die Werte der Tabelle 11 den Fehlerverlauf der einzelnen Aufgaben. Sie zeigen für den Öffnen-Dialog eine leicht bessere Bedienbarkeit mit der Navigationsleiste (Aufgabe Ö5). Auf der anderen Seite führen im Seiteneinstellungsdialog größere Elemente und ein ersetztes Drehfeld zu weniger Fehlern. Als Ausnahme steht, wie bereits aus den vorherigen Abbildung 79 bekannt, die Aufgabe S6. Die Steuerelemente sind darin 1,5 Zentimeter groß.

Für numerische Eingabeelemente wurde bereits das numerische Tastenfeld im Kapitel 5.7 vorgestellt. In den letzten beiden **Aufgaben S8** und **S9** sollte dieses Steuerelement auf dessen Bedienbarkeit getestet werden. Es ist anzumerken, dass auf eine genauere Untersuchung verzichtet wurde, da die Bewertung dieses einzelnen Steuerelements nicht im Vordergrund stand. Die Aufgaben unterschieden sich ausschließlich durch die einzugebenden Zahlen, so dass ein Unterschied nur durch einen Lerneffekt bestehen sollte. Die Ergebnisse der wiederholten Aufgabe zeigten jedoch kaum nennenswerte Unterschiede. Es konnte beobachtet werden, dass die große Menge an Berührungen dadurch möglich wurde, dass die Teilnehmer sich beim Ablesen der Zahlen aus der Szenarienbeschreibung irrten. Die Teilnehmer nutzten jedoch das numerische Tastenfeld trotz der höheren Eingabezahl mit nur maximal zwei Fehlern.

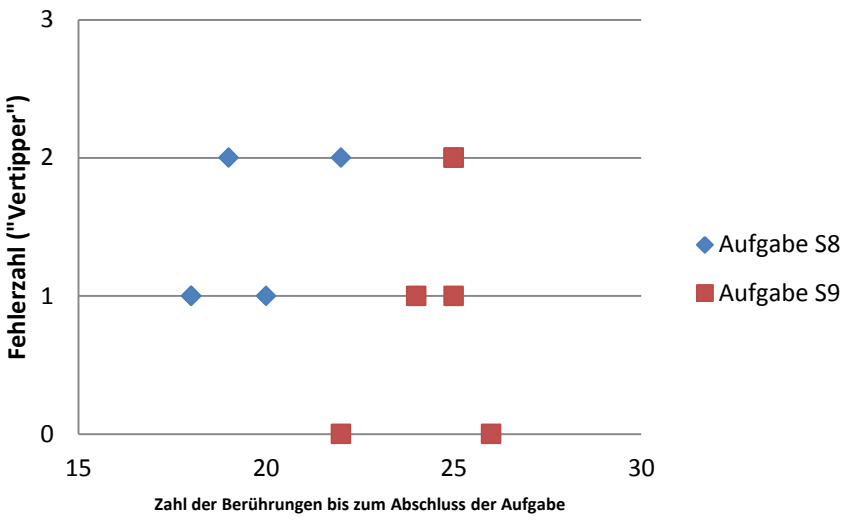


Abbildung 80 Vergleich der Fehlerquotienten der Aufgaben S8 und S9

Subjektive Präferenzen

Vor Beginn der Aufgaben wurden alle Teilnehmer gefragt, was sie bei der Bedienung von Oberflächen mit der Hilfe eines berührungsempfindlichen Bildschirms erwarten. Die Antworten waren:

- 3 x „einfach“ (zu bedienen)
- 2 x „schnelle Reaktion“ (der UI)
- „intuitiv“ (zu bedienen)
- „leichtgängig“
- „effektiv“
- „haptisch“ (als Rückmeldung)

Es ist deutlich, dass die einfache Bedienung ein K.O.-Kriterium darstellt, gefolgt von einer schnellen Reaktionsfähigkeit der Oberfläche.

Am Ende der Studie wurden die Teilnehmer gefragt, ob ihre Erwartungen erfüllt wurden und wenn nicht warum. Die Teilnehmer waren darüber allerdings geteilter Meinung wie Abbildung 81 zeigt. Zudem wurden sie gefragt, ob und in welcher Kombination (mit Tastatur und/oder Maus) sie einen berührungsempfindlichen Bildschirm am PC verwenden würden (Abbildung 82). Die meisten entschieden sich dafür den Bildschirm nur als zusätzliches Eingabegerät neben Maus und Tastatur verwenden zu wollen. Eine Person hätte den Bildschirm auch einzeln für bestimmte Anwendungen benutzt, während ein anderer Teilnehmer von Touch Abstand nahm.

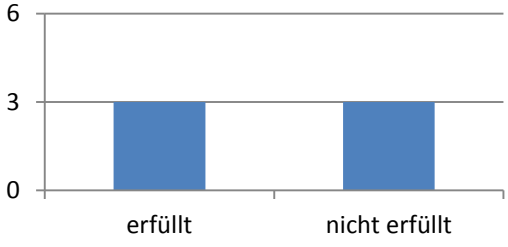


Abbildung 81 Wurden Ihre Erwartungen an die Bedienung des berührungsempfindlichen Bildschirms erfüllt?

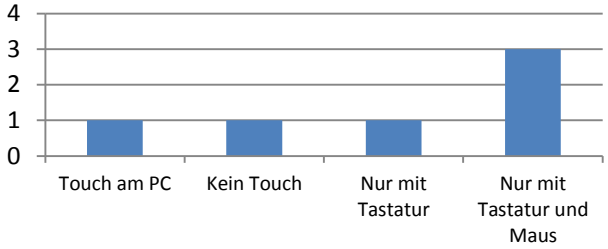


Abbildung 82 Können Sie sich vorstellen einen berührungsempfindlichen Bildschirm am PC oder Laptop zu verwenden?

Eine weitere Frage wurde gestellt, um herauszufinden, welche Art von Anwendung die Teilnehmer sich am PC als Touchanwendung vorstellen könnten. Die meisten entschieden sich für Spieleanwendungen oder Browser

(Internet), aber auch zum Zeichnen (Paint) oder zur Grafikmodellierung (3D). Dagegen wurde die Textverarbeitung für die Fingerbedienung ausgeschlossen.

In den Studienaufgaben selbst wurde jede Aufgabe durch zwei bis vier Fragen begleitet, bei denen der Teilnehmer seine Präferenz zum Ausdruck bringen sollte. Eine Frage konnte in sechs Unterscheidungsmerkmalen (A bis F) mit jeweils einem bis acht Punkten auf einer Likert-Skala bewertet werden (Tabelle 12). Zusätzlich sollten die Teilnehmer drei weitere Merkmale (A bis C) an Hand der gleichen Skala mit eins bis acht Punkten bewerten (Tabelle 13). Insgesamt kann die Summe der Bewertungen aller Teilnehmer damit zwischen sechs und 48 Punkten liegen, wobei letztere Punktzahl die beste mögliche Bewertung darstellt. Für die folgenden Ergebnisse werden die Kategorien A bis F bzw. A bis C mit der Nummer der jeweiligen Fragen verwendet, um einen Ergebniswert zu erhalten. Beispielsweise gibt F14.C an, wie viele Punkte die Teilnehmer bei der Frage nach der Bedienung des Dialogs (F14) im Kriterium C (schwierig bis leicht) vergeben haben.

Kategorie	Min. Kriterium 1 Punkt	1-8	Max. Kriterium 8 Punkte
A	**Frustrierend**	…	Motivierend
B	**Langweilig**	…	Stimulierend / Spannend
C	**Schwierig**	…	Leicht
D	**Reagiert langsam**	…	Reagiert schnell
E	**Ungewohnt**	…	Gewohnt
F	**Körperlich sehr anstrengend**	…	Körperlich kaum anstrengend

Tabelle 12 Kriterien und Gewichtung für die Aufgabenbewertungen. Bei allen Fragen (außer F15) eingesetzt.

Kat.	Kriterium	1 Punkt	1-8	8 Punkte
A	**Die optische Darstellung des Dialogs gefiel mir.**	Trifft nicht zu	…	Trifft zu
B	**Ich konnte Steuerelemente einfach treffen/anklicken/antippen.**	Trifft nicht zu	…	Trifft zu
C	**Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.**	Trifft nicht zu	…	Trifft zu

Tabelle 13 Spezielle Kriterien mit Gewichtungen für die Aufgabenbewertungen. Nur für den Aufgabentyp F15.

Aufgaben-nummer	Frage zur jeweiligen Aufgabe
F14	Wie beurteilen Sie die Bedienung des Dialogs während der gesamten Aufgabe? (Kriterien in Tabelle 12).
F15	Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien? (Kriterien in Tabelle 13)
F20	Wie beurteilen Sie die Bedienung des Steuerelements Drehfeld während der Aufgabe? (Kriterien in Tabelle 12).
F21	Wie beurteilen Sie die Bedienung des Steuerelements Tastenfeld für Zahleneingabe (am Drehfeld) während der Aufgabe? (Kriterien in Tabelle 12).
F22	Wie beurteilen Sie die Bedienung des Steuerelements Liste mit Kontrollkästchen während der Aufgabe? (Kriterien in Tabelle 12).

Tabelle 14 Einige Fragen zu den Studienaufgaben

Die Ergebnisse aus Tabelle 15 und Tabelle 16 spiegeln zum großen Teil die Ergebnisse der Abschnitte Aufgabenausführungsdauer und Genauigkeit wider. Diese Tabellen enthalten die vergebene Gesamtpunktzahl der Teilnehmer pro Frage und Kategorie. Die vorherigen Ergebnisse zeigten hohe Fehlerraten bei der Bedienung der Dialoge von Ö2 und S2. Die Teilnehmer bewerteten die Bedienung des Dialogs Ö2 entsprechend schlecht. Die Bedienung mit dem Finger sind nach Tabelle 15 schwerer (F14.C: 28 Punkte) und ungewohnter (F14.E: 19 Punkte) zu bedienen als mit der Maus (Ö1: C: 46 und E:39 Punkte). Die Steuerelemente der Aufgaben Ö2 und S2 waren den Teilnehmern nach Tabelle 16 damit zu klein (F15.B: 14 Punkte) und konnten nur schwer mit den Fingern getroffen werden (F15.C: 19 Punkte). Analog wurden die Aufgaben S1 und S2 bewertet. Auch dort sahen die Teilnehmer eine Zunahme des Schwierigkeitsgrades (F14.C: 14 Punkte) bei der Bedienung. Die kleinen Steuerelemente in S2 straften sie mit niedrigen Bewertungen in F15.B und F15.C (14 und 13 Punkte) ab. Insbesondere die kleinen Schalter der Drehfelder wurden als schwer zu treffen bewertet (F20.C: 14 von 48 Punkten). Die Aufgabe wurde zudem als „fast nicht machbar“ kommentiert.

Die Bedienung der Dialoge in den Aufgaben Ö3 bis Ö7 wurde von den Teilnehmern wieder besser als Ö2 gewertet. Der allgemeine Schwierigkeitsgrad F14.C ging den Teilnehmern zufolge deutlich zurück und die vergebenen Punkte erhöhten sich bis auf das Reihenmaximum von 46 in Ö3. Die Teilnehmer befanden außerdem, dass sie bei jeder neuen Aufgabe die Steuerelemente besser treffen würden (F15.B) und der Dialog einfacher mit dem Finger zu bedienen sei (F15.C).

Im Seiteneinstellungsdialog der Aufgaben S3 bis S6 sahen die Teilnehmer eine sichtbare Verbesserung gegenüber dem Dialog aus S1 und S2. Die größer werdenden Flächen der Steuerelemente fanden entsprechende Punktzahlen beim Schwierigkeitsgrad (F14.D): Die vergebenen Punkte (40, 46, 47 und 45) sind höher als beim Dialog in Originalgröße (29 Punkte). Zudem konnten die Steuerelemente einfacher getroffen werden (F15.B, von 14 auf maximal 44 Punkte). Die letzte Aufgabe S6 verursachten jedoch einen leichten Rückgang der hohen Punktzahl von 47 auf 45 Punkte. Dies liegt an den Kontrollkästchen, die von den Teilnehmern in einer separaten Bewertung F22 evaluiert wurden. Der Schwierigkeitsgrad der Liste mit Kontrollkästchen wurde in F22.C (Tabelle 15) festgehalten. Die 27 Punkte der Aufgabe S6 (F22.C) reflektieren deutlich, welche Mühen die Teilnehmer hatten, um die Kontrollkästchenliste zu bedienen. Und das, obwohl die Kontrollkästchen selbst gut zu treffen waren, wie F15.B mit 40 Punkten zeigt. Weiterhin befanden die meisten Teilnehmer, dass die Bedienung der Dialoge in den Aufgabe S4 und S5 am wenigsten gewöhnungsbedürftig sei, indem sie 45 Punkte in F22.E vergaben. Das sind jeweils 9 und 10 Punkte mehr als noch in den Aufgaben S3 und S6.

Aufgabe	F14.C	F14.D	F14.E	F20.C	F20.D	F20.E	F21.C	F21.D	F21.E	F22.C	F22.D	F22.E
Ö1	46	39	41									
Ö2	28	19	26									
Ö3	46	31	30									
Ö4	40	36	30									
Ö5	34	34	29									
Ö6	40	38	34									
Ö7	42	41	34									
S1	38	39	38	45	45	45				47	48	46
S2	29	44	34	14	46	15				33	45	31
S3	40	43	41	35	45	37				35	46	36
S4	46	45	43	47	46	43				43	46	45
S5	47	44	43	46	45	40				42	46	45
S6	45	45	44	44	47	39				27	45	35
S7	29	45	34									
S7.1										16	37	25
S7.2										25	44	35
S7.3										33	42	34
S7.4										33	45	37
S8	43	43	35				45	41	38			
S9	45	46	40				45	45	41			

Tabelle 15 Ergebnisse der Befragung aus den Fragentypen F14, F20, F21 und F22 in den Kategorien C (schwierig/leicht), D ([UI] reagiert schnell/langsam) und E (ungewohnt/gewohnt). Minimale/Maximale zu vergebende Punktzahl: 6/48.

Aufgabe	F15.B	F15.C
Ö1	40	48
Ö2	14	19
Ö3	29	28
Ö4	44	40
Ö5	42	39
Ö6	44	45
Ö7	47	44

Aufgabe	F15.B	F15.C
S1	46	48
S2	14	13
S3	29	32
S4	44	46
S5	43	44
S6	40	40

Aufgabe	F15.B	F15.C
S7.1	16	29
S7.2	22	35
S7.3	29	39
S7.4	32	35
S8	43	44
S9	44	43

Tabelle 16 Ergebnisse der Befragung des Fragentyps F15 in den Kategorien B (Ich konnte Steuerelemente einfach treffen/anklicken/antippen.) und C (Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.). Minimale/Maximale zu vergebende Punktzahl: 6/48.

Mit den Aufgaben Ö4 und Ö5 wurde ein Vergleich durchgeführt, der das Blättern in einer Dateiliste mit einem Finger (durch Halten der Berührung und Bewegen des Fingers) oder mit einer Navigationsleiste (grüne Pfeile) bewerten sollten. Die Teilnehmer bewerteten dabei die Bedienung (F14.C) der Navigationsleiste schwerer (Ö5: 34 Punkte) als direkt mit dem Finger in der Liste zu blättern (Ö4: 40 Punkte). Dies liegt allerdings nach Tabelle 16 nicht an zu kleinen Schaltflächen (F15.B) in der Navigationsleiste, denn der Einsatz des Fingers in Aufgabe Ö4 und Ö5 fiel den Teilnehmern in etwa gleich schwer oder einfach (44 zu 42 und 40 zu 39 Punkten). Es muss dazu gesagt werden, dass die beiden ältesten Teilnehmer (50-60 Jahre) die Navigationsleiste immer mit 8 Punkte in den Kategorien F14.C, F15.B und F15.C bewerteten. Die jüngeren Teilnehmer zogen allgemein vor, in der Liste mit dem Finger zu blättern.

Das zuletzt in den Aufgaben S8 und S9 eingesetzte numerische Tastenfeld kam bei den Teilnehmern besser an als die Drehfelder der Aufgaben S3 bis S6. Den Schwierigkeitsgrad bewerteten die Nutzer daher niedrig (F14.C: 43 und 45 Punkte). Zudem wurden die Schalterelemente als einfach zu treffen gewertet (F15.B: 43 und 44 von 48 möglichen Punkten) genauso wie den gesamten Einsatz des Fingers (F15.C: 44 und 43). Die Teilnehmer bemängelten lediglich die Symbole der Schalter und Positionierung des Eingabefeldes nahe dem unteren Bildschirmrand als missglückt. Zudem rüffelten sie auch die fehlende, optische Verknüpfung mit dem Zahlenfeld. Das aktuelle Feld hätte hervorgehoben werden müssen.

8.2.4 Diskussion

Diese durchgeführte Studie produzierte nicht nur eine Menge interessanter Daten, sondern auch eine Menge Erfahrungswerte. So stellte es sich erst im Nachhinein heraus, dass die Datenerhebung mit Fragebogen einen beträchtlichen Aufwand darstellte. Die Nutzung von automatisierten oder Onlinefragebögen wäre doch deutlich schneller gegangen. Außerdem war die Zeitdauer pro Teilnehmer von bis zu 1,5 Stunden zu lang, so dass entweder weniger Fragen oder Aufgaben hätten gestellt werden müssen.

Allgemein lassen die Ergebnisse erkennen, dass die Bediendung von Dialogen mit den Fingern etwas länger dauern und mehr „Vertipper“ hervorbringen kann. Die liegt vor allem an der Positionierung des Fingers auf dem Bildschirm, die weit weniger präzise ist als die Maus und zudem sofort als Klick vom Dialog gewertet wird. Ein Mauszeiger muss dagegen nicht nur positioniert, sondern auch noch explizit zum Klicken gebracht werden. Die Ergebnisse zeigten jedoch, dass eine Änderung der Steuerelementgröße bereits eine deutliche Verbesserung in beiden Dialogen (Öffnen und Seiteneinstellung) für die Bedienung hervorbrachte. Auch die Dialoge in späteren Aufgaben, die mit entsprechend mit größeren Steuerelementen gestaltet worden waren, zeigten eine hohe Trefferrate mit dem Finger. Die erste Hypothese (*„Es ist ausreichend die Eigenschaften von einzelnen Elementen zu verändern, um die Elemente mit den Fingern besser bedienen zu können.“*) sehe ich daher als bestätigt an. Allerdings muss man beachten, dass größer auch nicht immer besser ist. Insbesondere Steuerelemente, die in Listen eingebettet sind wie Kontrollkästchen waren letztendlich durch ihre Größe von 1,5 Zentimeter zu groß, so dass die Teilnehmer keine Übersicht hatten und ständig in der Liste blättern mussten. Dies verursachte jedoch weitere Fehler und Unsicherheiten, ob nun alle verlangten Kontrollkästchen aktiviert waren. Dieser Umstand sollte beachtet werden, wenn Dialoge transformiert werden.

Eine weitere Frage, die in der Studie geklärt werden sollte, bestand darin, ob die vorgestellten Steuerelemente aus dem Kapitel 5 („Anpassung der Steuerelemente für berührungsempfindliche Eingaben“) vorteilhaft für die Fingerbedienung sind. Geprüft wurden dabei die Steuerelemente Kontrollkästchen (Kapitel 5.4), das numerisches Tastenfeld (Kapitel 5.5), die Brotkrumennavigation (Kapitel 5.6) und die Navigationsleiste für Listenfelder (5.7). Die Kontrollkästchen wurden dafür erweitert, um eine Vergrößerung zu erreichen, da die Größe der Box und des Hakens im Standardelement von WPF unveränderlich sind. Stattdessen wurde eine abgeleitete Klasse mit dem Namen *CheckBoxTouch* erstellt, die eine Größenänderung zuließ. Der Aufwand lohnte sich auch, denn die Studienergebnisse zeigten, dass dadurch die Treffersicherheit stieg. Dies galt ebenso für das numerische Tastenfeld, welches außerdem von den Teilnehmern besser aufgenommen wurde als die vergrößerten Drehfelder. Ein weiterer Vorteil gegenüber den Drehfeldern war zudem die Platzersparnis. Die großen Drehfeld-

schaltflächen nehmen immer wertvolle Dialogfläche in Besitz, auch wenn sie nicht benötigt werden. Die Navigationsleiste wurde, wie bereits erwähnt, von den jungen Teilnehmern nicht gut angenommen. Die älteren Teilnehmer dagegen nutzten sie gerne. Daher ist der Einsatz einer solchen Leiste sehr vom Geschmack und Gewohnheitsempfinden des Benutzers abhängig. Ist er oder sie bereits gewohnt an das Blättern mit den Fingern, können die Pfeile eher stören und wertvollen Platz in Anspruch nehmen. Dagegen sind Anfänger und weniger Technik affine Menschen besser mit der Navigationsleiste vertraut, da sie ein offensichtliches Werkzeug zum Blättern darstellt. Nicht zuletzt sind solche Pfeile bereits von vielen Kiosksystemen wie Fahrkartenschalter bekannt. Daher ist der Einsatz eines solchen Steuerelements von der Zielbenutzergruppe abhängig, die vor der Transformation bekannt sein sollte. Ein anderes Steuerelement konnte in der Studie leider nicht genügend bewertet werden. Es handelt sich dabei um die Brotkrumennavigation, die für hierarchische Listen gedacht war. Die Teilnehmer nutzten, um eine Ordnerebene aufwärts zu kommen, nicht die vorgesehene Brotkrumennavigation, sondern den Zurück-Schalter des Verlaufs daneben. Die Aufgaben oder der Dialog hätten also entsprechend anders gestaltet werden müssen. Die gewonnene Erkenntnis war trotzdem aufschlussreich: Für die Teilnehmer kam die Brotkrumenleiste nicht als Navigationswerkzeug in Frage. Womöglich müssen die Schaltelemente innerhalb der Leiste besser als solche erkennbar gestaltet werden. Der bei der Brotkrumennavigation genutzte Stil (oder in Neudeutsch „Theme“) von Windows war so eher kontraproduktiv. Der Stil verbirgt die Schaltelemente innerhalb der Leiste bis der Mauszeiger darüber schwebt. Für eine Fingerbedienung ist dies natürlich hinderlich. Die zweite Hypothese („Es ist ausreichend Steuerelemente in einem Dialog auszutauschen oder zu erweitern (mit zusätzlichen Elementen), um diesen Dialog besser mit den Fingern bedienen zu können. “) kann jedoch trotzdem als erfüllt angesehen werden, denn der Erfolg des neuen Steuerelements hängt von dessen Umsetzung sowie von der Zielbenutzergruppe ab. In diesem Fall waren die vergrößerten Kontrollkästchen und das numerische Tastenfeld ausschlaggebend für eine verbesserte Bedienung mit den Fingern.

Die Bewertung der dritten Hypothese („Die eingeführten Steuerelemente, als Ersatz vorhandener Elemente, sind für die Nutzung mit den Fingern geeignet.“) gestaltet sich nicht so eindeutig. Die Navigationsleiste kam nicht gut bei den jungen Teilnehmern an und die Brotkrumennavigation wurde nicht als Navigationshilfe erkannt. Die schlechtere, subjektive Bewertung der Navigationsleiste steht allerdings der besseren Trefferrate (vgl. Tabelle 11 auf Seite 117) entgegen. Die Brotkrumennavigation dagegen müsste erneut bewertet werden mit besseren, sichtbaren Schaltern. Abgesehen davon sind die vorgestellten Steuerelemente für die Nutzung auf berührungsempfindlichen Bildschirmen geeignet.

Letztendlich sollte mit der durchgeführten Studie gezeigt werden, dass es für eine berührungsempfindliche Oberfläche nicht unbedingt notwendig ist den natürlichen Oberflächen (NUI) zu folgen und NUI damit zum neuen Oberflächenstandard zu definieren. Die Studie lieferte wichtige Hinweise, dass die Umgestaltung von Dialogen eine ausreichende Maßnahme zur Verbesserung der Bedienbarkeit darstellen kann. Die Größe der Steuerelemente spielt letztendlich eine entscheidende Rolle, sie ist jedoch kein alleiniger Faktor für eine bessere Bedienung. Denn es zeigte sich, wie bereits erwähnt, dass Zahlen in Dialogen besser mit einem extra dafür geschaffenen numerischen Tastenfeld bedient werden können als mit Drehfeldern.

Aus allen zuvor genannten Gründen bin ich daher zu der Überzeugung gelangt, dass die zu Anfang genannten Hypothesen bestätigt wurden. Damit sehe ich die Umsetzung LATTE als erfolgreich an. Es muss dabei jedoch beachtet werden, dass dies **nicht** bedeutet, dass die so transformierten Dialoge in jeder Hinsicht ideal für die berührungsempfindliche Eingabe geworden sind. Es scheitert, wie so oft im Leben, am Spezialfall. So sind Dialoge, die als reine Formulare zur Text- und Zahleneingabe dienen kaum für die Fingerbedienung geeignet, sofern die Eingaben nicht überwiegend aus vorgefertigten Auswahllisten entnommen werden können. Für die weniger speziellen Dialoge zeigte sich allerdings, dass bereits durch einfache Änderungen und Ersetzungen von Steuerelementen die Bedienbarkeit mit den Fingern objektiv und subjektiv deutlich verbessern lässt.

9 Zusammenfassung und Ausblick

Wer all seine Ziele erreicht, hat sie zu niedrig gewählt.

Herbert von Karajan
österreichischer Dirigent

Das letzte Kapitel dieser Diplomarbeit widmet sich der Übersicht über die vorangegangenen Themen, um im Rückblick noch einmal die wichtigsten Aspekte hervorzuheben (Kapitel 9.1). Der Abschluss bildet einen Ausblick über die zukünftige Weiterentwicklung der Methode und ihrer Umsetzung (Kapitel 9.2).

9.1 Zusammenfassung

Das Ziel dieser Diplomarbeit war es Dialoge automatisiert für die Nutzung auf berührungsempfindliche Bildschirme anzupassen. Dazu wurde eine Methode entwickelt, die verschiedene Dialogsprachen in eine gemeinsame Sprache überführt, damit darauf eine Transformationsvorschrift ausgeführt werden kann. Auf diese Weise wandelt die Transformation den gewünschten Dialog in eine bessere mit den Fingern bedienbare Darstellung um. Für die Umsetzung wurde die Methode so weit konkretisiert, dass als Ausgangs- und Zielsprache für Dialoge die Sprache XAML festgelegt wurde, um darauf eine Transformation mit der Hilfe von XSL (Extended Stylesheet) ausführen zu können.

Die Transformation sollte in dieser Arbeit durch eine Anwendung erfolgen von der als Erstes eine Architektur erstellt wurde. Mit der gewählten Architektur wurde schließlich eine Anwendung mit WPF und C# entwickelt, welche die Transformation von XAML-Dialogen durch eine flexibel und anpassbare Transformationsvorschrift ermöglicht. Zusätzlich wurden externe Module (oder auch Plug-Ins) zugelassen, um in den Transformationsprozess eingreifen zu können.

Um die Funktionsfähigkeit der entwickelten Methode und Anwendung zu demonstrieren, wurde zuletzt eine Studie durchgeführt. Für die Studie wurden zunächst einmal die notwendigen Anpassungen von Steuerelemente eines Dialoges untersucht, um herauszufinden welche Steuerelemente auf welche Art für die Finger angepasst werden müssen. Diese Erkenntnisse wurden in Form einer parametrisierten Transformationsvorschrift auf zwei unterschiedliche Dialoge angewendet, welche anschließend in der Studie von den Teilnehmern geprüft wurden. Die Studie ergab schlussendlich, dass die Umsetzung und die angepasste Steuerelemente eine Verbesserung für die Bedienung mit den Fingern darstellen.

9.2 Ausblick

Die Bearbeitung des Diplomarbeitsthemas erzeugte eine Menge weiterer Ideen und Möglichkeiten, die leider aus Zeitgründen nicht umgesetzt werden konnten. Diese sollen jedoch nicht unerwähnt bleiben, so dass die Ideen nicht verloren gehen und stattdessen von anderen aufgenommen und weiterentwickelt werden können.

9.2.1 Die Zukunft von LATTE

LATTE wurde als Prototyp entwickelt, so dass nur die grundlegendsten Funktionen enthalten sind. Daher ist eine Weiterentwicklung notwendig. Im Folgenden werden einige Erweiterungsmöglichkeiten beschrieben.

Integrierte Entwicklungsumgebung

Die Oberfläche von LATTE wurde von Grund auf mit WPF entwickelt. Es ist jedoch auch denkbar eine vorhandene Entwicklungsumgebung sowie deren Funktionalität wie Quelltexteditor und Dialogdesigner wiederzuverwenden. Beispielsweise stellt Microsoft Visual Studio für 2010 eine umfangreiche Modulschnittstelle zur Verfügung. Aber auch andere Entwicklungsumgebungen wie Eclipse stellen solche Schnittstellen bereit und können natürlich als Plattform für LATTE genutzt werden. Diese Umgebungen stellen bereits ein ausgeklügeltes Fenstermanagement zur Verfügung und können außerdem den besprochenen Transformationsprozess in die Entwicklung des Programms integrieren.

Erweiterung der Transformationsvorschrift für XAML

Die vorgestellten Abbildungen für Steuerelemente sind bei weitem nicht vollständig. Es existieren viele Steuerelemente, die nicht erkannt werden, obwohl sie in Dialogen auftauchen könnten. Dazu zählen beispielsweise Register oder Kontextmenüs. Zusätzlich müssen dazu Überlegungen und Prüfungen unternommen werden, wie diese Elemente für die berührungsempfindliche Bedienung angepasst werden können. Es wird nie ausreichend sein, Steuerelemente einfach zu transformieren, ohne zu wissen, ob dies ein Vorteil für die Bedienung mit den Fingern sein wird.

Grafischer Designer für die Dialogvorschau

LATTE unterstützt die Vorschau des originalen sowie des transformierten Dialogs. Allerdings ist es nicht möglich, diese Dialoge wie in Visual Studio mit der Maus zu ändern. Der Entwickler kann nur den Quelltext des Dialogs direkt anpassen, jedoch werden Änderungen im transformierten Quelltext immer überschrieben. Eine Verbesserung wäre daher, einen grafischen Designer für die Dialoge einzuführen, der zudem manuelle Änderungen im Zieldialog konserviert, damit diese eine Transformation überstehen können. So können manuelle Änderungen des Entwicklers in LATTE direkt vorgenommen werden, ohne dass ein externer Editor notwendig wäre.

Grafischer Designer für die Transformationsvorschrift

Die Sprache XSL kann sehr aufwändig zu nutzen sein, insbesondere wenn die Abbildungsvorschrift kompliziert ist. Die Idee ist daher, die Sprache zu vereinfachen, indem ein grafischer Editor verwendet wird, der es dem Benutzer erlaubt eine Transformationsvorschrift aus vorgefertigten Bausteinen zusammen bauen zu lassen. Entweder kann dies durch grafische Objekte geschehen, die ähnlich UML in einem Editor zusammengesetzt werden oder die Erstellung einer Transformationsvorschrift wird durch Auswahl von Werten in einem Dialog ermöglicht. Der Dialog stellt dazu entsprechende Steuerelemente zur Verfügung, um Steueranweisungen erstellen zu können (z.B. Wenn-Abfrage, Schleife, Zuweisung, Zahlenwerte und Variablen). Letzteres wurde bereits im Blizzards Warcraft 3 Editor zur Steuerung des Spielgeschehens eingesetzt.

Integration der Plug-Ins in LATTE

Die Erstellung von Plug-Ins kann den Transformationsprozess erleichtern. Allerdings muss dazu ständig eine Entwicklungsumgebung gestartet sein und die notwendigen Dateien müssen in das Plug-In Verzeichnis kopiert werden. Mit dem Kommandozeilenprogramm *msbuild* von Microsoft ist es allerdings möglich Quelltexte automatisiert zu kompilieren. Die quelloffenen Entwicklungsumgebung SharpDevelop für C# demonstriert dies bereits eindrucksvoll. Plug-Ins könnten so direkt innerhalb der LATTE Umgebung geändert und neu erstellt

werden, ohne dass der Entwickler die Umgebung von LATTE verlassen müssten.

Transformation auf der Kommandozeile

Entwicklungsumgebungen für C# und andere Sprachen bieten dem Entwickler häufig die Möglichkeit weitere Prozesse vor und nach der Kompilation des Programms auszuführen, z.B. um die Anwendung automatisch zu signieren. Es ist jedoch genauso denkbar, dass die eingesetzten Dialoge noch vor der Erstellung des Programms transformiert werden müssen, um im Endprodukt ihren Platz zu finden. Ein Kommandozeilenprogramm für die Transformation von Dialogen würde diesen Prozess beschleunigen. Zudem ist es nicht immer notwendig einen Editor zu starten, weil z.B. die Transformationsvorschrift bereits vorliegt und nur ein oder mehrere Dialoge transformiert werden müssen.

9.2.2 Die Zukunft der Methode

Zusätzlich zu den Verbesserungen für LATTE, bietet auch die vorgestellte Methode Spielraum für weitere Ansätze und Verbesserungen. Zwei davon sind nachstehend beschrieben.

Einsatz von plattformunabhängigen Dialogbeschreibungssprachen und Modellen

In der konkreten Umsetzung der Methode wurde XAML verwendet, unter anderem weil für XAML bereits eine Unterstützung für die Quelltextanzeige und Dialogvorschau besteht. Allerdings wäre es denkbar, auch andere Dialogbeschreibungssprachen zu verwenden, beispielsweise UsiXML, UIML oder XIML. Damit kann noch tiefer in die modellgetriebene Entwicklung eingestiegen werden. Ein Dialog existiert dann nur noch in einer abstrakten Beschreibungssprache und kann nach Belieben in verschiedene Darstellungsformen entwickelt werden. In diesem Fall ist nicht mehr die Fingerbedienung im Vordergrund, sondern sie ist nur noch eine Möglichkeit von vielen, einen Dialog darzustellen. Auf diese Weise kann der gleiche Dialog auf einem Computerbildschirm, auf einem Smartphone, in einem Kiosksystem am Bahnhof oder in einem Altenheim benutzt werden. Die Anforderungen aller Plattformen können sich dabei stark unterscheiden. Dazu zählt auch die Bedienung mit den Fingern. Doch mit der modellgetriebenen Entwicklung bestimmt nur noch die Zielplattform das Aussehen des Dialogs, nicht mehr der Entwickler.

Einsatz im Internet

Mit jeder neuen Version von Windows und auch anderen Betriebssystemen, wird man ein Stück näher zur reinen berührungsempfindlichen Bedienung kommen. Mit der Ankündigung in Windows 8 eine komplett neue Benutzerschnittstelle in HTML 5 und JavaScript einzusetzen (siehe [Dow Jones & Company Inc., 2011]), könnte der in dieser Arbeit vorgestellte Ansatz von XAML auf HTML verlegt werden. Dies würde zudem den vielen Webseiten und insbesondere den Oberflächen des Cloud Computing eine bessere Bedienung mit den Fingern verschaffen. Das große Problem stellen jedoch die Unterschiedlichen HTML Standards sowie die vielen Tricks und Kniffe von Webseitenentwicklern dar, um die Oberfläche in den jeweiligen Browsers optimal darstellen zu können. Außerdem ist HTML 5 zum aktuellen Zeitpunkt noch nicht weit verbreitet, Windows 8 noch in der Entwicklung und viele Webseiten setzen auf geschlossene Formate wie Flash und Quick Time. Trotzdem liegt in HTML die Zukunft der Oberflächenentwicklung, denn Tablet-Computer wie das iPad werden immer kostengünstiger und sind daher auf dem Vormarsch. In Zukunft müssen Webdesigner mehr Wert und Aufwand in die berührungsempfindliche Bedienung ihrer Webseiten stecken, damit diese auch mit der neuen Gerätegeneration ohne Probleme genutzt werden können (Abbildung 83).

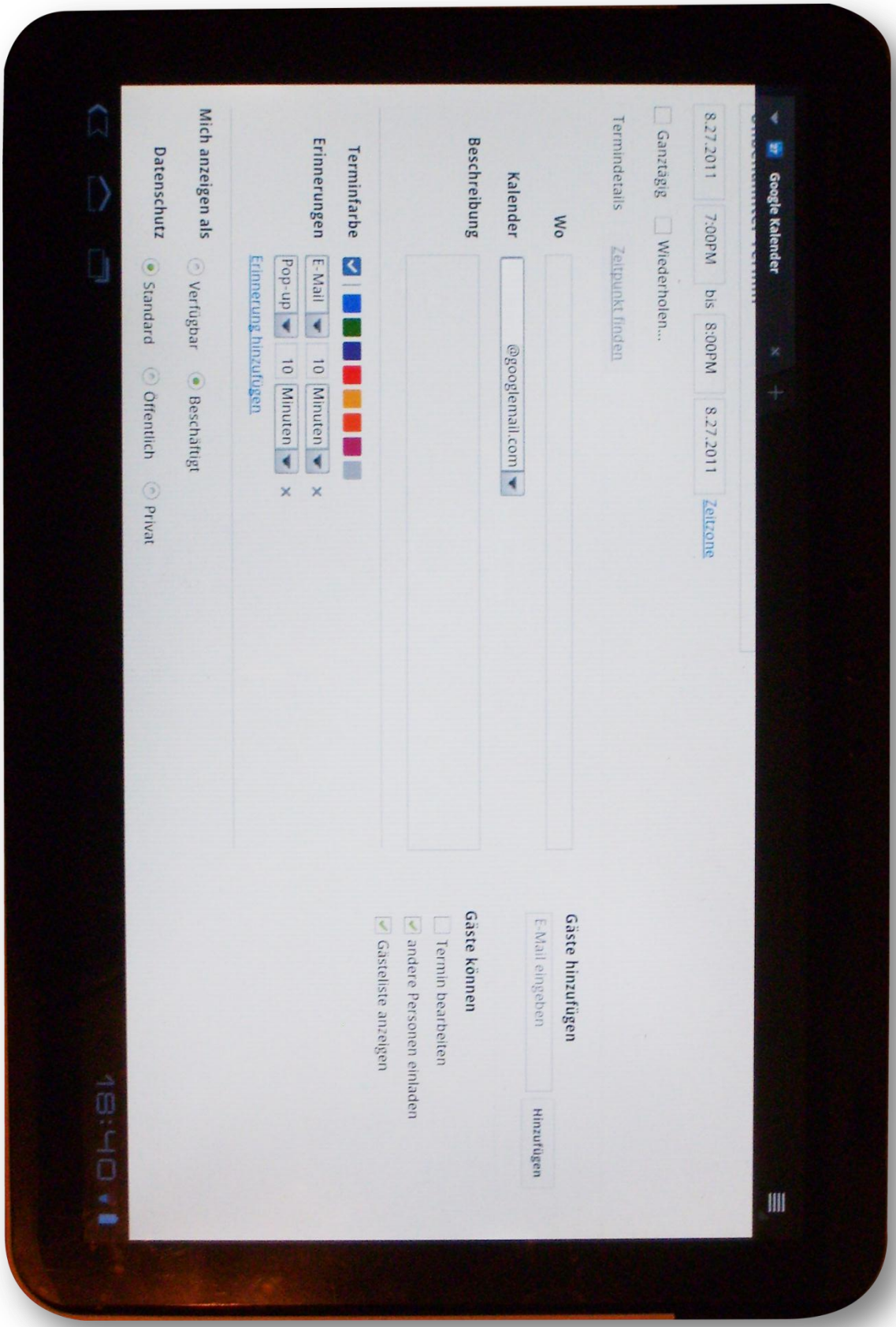


Abbildung 83 Viele Webseiten und Cloud Anwendungen lassen sich nur schwer und nicht ohne weiteres mit den Fingern bedienen. Hier der Google Kalender. Diagonalverhältnis von originalem Tablet zu Bild ist 1 zu ca. 0,93.

Literaturverzeichnis

.NET Framework Developer Center: Windows Workflow Foundation... [Online] http://msdn.microsoft.com/en-us/netframework/aa663328.

Abrams, M. und Helms, J. 2004. User Interface Markup Language Specification Version 3.1. *OASIS.* [Online] 2004. http://www.oasis-open.org.

Abramson, D., Watson, G. und Dung, Le Phu. 2002. *Guard: A Tool for Migrating Scientific Applications to the.NET Framework.* 2002.

AlexDov. 2008. XamlWriter and Bindings Serialization. [Online] 29. Juni 2008. http://www.codeproject.com/KB/WPF/xamlwriterandbinding.aspx.

AndroMDA. 2011. What is AndroMDA. [Online] 2011. http://www.andromda.org/docs/whatisit.html.

Anslow, Craig. 2010. *Multi-touch Table User Interfaces for Collaborative Visual Software Analytics.* Wellington : Victoria University of Wellington, 2010.

Antoniol, G., et al. 1995. Application and user interface migration from BASIC to Visual C++. *11th International Conference on Software Maintenance (ICSM'95).* 1995.

Bandelloni, R., Bert, S. und Paternò., F. 2004. Mixed-Initiative, Trans-Modal Interface Migration. *Proceedings Mobile HCI'04.* 2004, S. 216-227.

Bandelloni, R., Mori, G. und Paternò, F. 2007. Automatic User Interface Generation and Migration in Multi-Device Environments. *ACM Transaction on Computer-Human Interaction.* 2007.

Bandelloni, R., Paternò, F. und Santoro, C. 2008. Reverse Engineering Cross-Modal User Interfaces for Ubiquitous Environments. 2008.

Barclay, P., et al. 1999. *The Teallach Tool: Using Models for Flexible User Interface Design.* Glasgow, Großbritannien : s.n., 1999.

Berti, S., et al. 2004. *TERESA: a transformation-based environment for designing and developing multi-device interfaces.* 2004.

Blascheck, T., Bold, D. und Muhler, D. 2010/2011. *Interaktionskonzepte für Multi-Touch.* Stuttgart : Universität Stuttgart, 2010/2011.

Breier, F. 2010. *Multitouch 3D Interaktion mit 3D Objekten.* Stuttgart : s.n., 2010.

Canfora, G., Di Santo, G. und Zimeo, E. 2004. Toward Seamless Migration of Java AWT-Based Applications to Personal Wireless Devices. *Proceedings of the 11th Working Conference on Reverse Engineering.* 2004, S. 38-47.

Chapman, S. 2008. Windows 7 NUI: Stepping Beyond the GUI. [Online] 4. Juni 2008. http://msftkitchen.com/2008/06/windows-7-nui-stepping-beyond-gui.html.

Chattopadhyay, S. 2010. WPF Simplified Part 10: WPF Framework Class Hierarchy. [Online] 10. Januar 2010. http://soumya.wordpress.com/2010/01/10/wpf-simplified-part-10-wpf-framework-class-hierarchy.

Chikofsky, E.J. und Cross II, J.H. 1990. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Softw.* 1990, Bd. 7, 1, S. 13-17.

CNET News.com. CNET News.com. [Online] http://asia.cnet.com/domino-theories-for-microsofts-surface-pc-62033135.htm.

CnPack. [Online] http://www.cnpack.org.

Di Santo, G. und Zimeo, E. 2004. Reversing GUIs to XIML descriptions for the adaptation to heterogeneous devices. *Proceedings of the 2007 ACM symposium on Applied computing.* 2004, S. 1456-1460.

Doberenz, W. und Gewinnus, T. 2008. *Visual C# 2008.* Frankfurt/Oder : Hanser, 2008.

Dow Jones & Company Inc. 2011. Microsoft Windows President Steven Sinofsky Introduces the New Look of Windows. [Online] 1. Juni 2011. http://allthingsd.com/20110601/up-next-at-d9-microsoft-windows-president-steven-sinofsky-live-at-d9/.

Draheim, D., Lutteroth, C. und Weber, G. 2006. Graphical user interfaces as documents. *Proceedings of the 7th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: design centered HCI.* 2006.

Erlenkötter, H. und Reher, V. 1997. *C++ für Windows 95/NT.* Hamburg : Rowohlt, 1997.

Experience Dynamics. User Interface Style Guides. *http://www.experiencedynamics.com/science-usability/ui-style-guides.* [Online]

Fischer, P. und Hofer, P. 2008. *Lexikon der Informatik.* [Hrsg.] Springer. 14. Luzern : s.n., 2008.

Foley, J. und Sukaviriya, N. 1995. Results, and Bibliography of the User Interface Design Environment (UIDE), an Early Model-Based System for User Interface Design and Implementation. *Interactive Systems: Design, Specification, and Verification.* 1995, S. 3-10.

Fraunhofer IAO. 2009. *Studie Multi-Touch.* [Dokument] 2009.

Geis, T. 2006. ProContext - The new ISO 9241-110 "Dialogue principles". [Online] 11. August 2006. http://www.procontext.com/en/news/2006-08-11.html.

Georges, F. 2007. Nightly thoughts (Blog). [Online] 18. Januar 2007. http://fgeorges.blogspot.com/2007/01/creating-namespace-nodes-in-xslt-10.html.

Gerdes, J. 2009. User Interface Migration of Microsoft Windows Applications. *Journal of Software Maintenance and Evolution: Research and Practice.* 3, 2009, Bd. 21, S. 171–187.

GExperts. Programming Tools For Delphi and C++ Builder. [Online] http://www.gexperts.org.

Gleeson, M., Stanger, N. und Ferguson, E. 2004. Design strategies for GUI items with touch screen based information systems: assessing the ability of a touch screen overlay as a selection device. [Hrsg.] University of Otago. *Information Science Discussion Papers Series.* 2004.

Grilo, A.M.P., Paiva, A.C.R. und Faria, J.P. 2007. Reverse Engineering of GUI Models. *FMICS'07 Proceedings of the 12th international conference on Formal methods for industrial critical systems.* 2007.

Grolaux, D. 2004. Migratable User Interfaces: Beyond Migratory User Interfaces. *MOBIQUITOUS IEEE Computer Society Press, Los Alamitos.* 2004, S. 22-25.

Han, J. Y. 2005. Low-Cost Multi-Touch Sensing through Frustrated Total Internal Reflection. *Proceedings of the 18th annual ACM symposium on User interface software and technology.* 2005.

Heinecke, A. M. 2004. *Mensch- Computer- Interaktion.* s.l. : Fachbuchverlag Leipzig, 2004.

HIIS Laboratory. MARIA. [Online] http://giove.isti.cnr.it/tools/MARIA/home.

—. MARIAE. [Online] http://giove.isti.cnr.it/tools/MARIAE/home.

—. 2010. Tools - MARIA, MARIAE, CTT, ReverseMARIA und andere. [Online] 2010. http://giove.isti.cnr.it/tools.php.

Hillberg, M. 2006. Data See, Data Do - Being written by XamlWriter. [Online] 16. 9 2006. http://blogs.msdn.com/b/mikehillberg/archive/2006/09/16/xamlwriter.aspx.

Hitchcock, G. 2005. Where does 96 DPI come from in Windows? [Online] Where does 96 DPI come from in Windows?, 8. Oktober 2005. http://blogs.msdn.com/b/fontblog/archive/2005/11/08/490490.aspx.

Holman, D. und Vertegaal, R. 2008. Organic user interfaces: designing computers in any way, shape, or form. *Communications of the ACM - Organic user interfaces.* 2008, Bd. 51, 6.

Holz, C. und Baudisch, P. 2011. *Understanding Touch.* Potsdam, Germany : Hasso Plattner Institute, 2011.

IC#Code. 2009. SharpDevelop. [Online] 2009. http://www.icsharpcode.net.

John, Jr., G. 2009. User Interface Migration of Microsoft Windows Applications. *Journal of Software Maintenance and Evolution: Research and Practice.* 2009, Bd. 21, 3, S. 171-187.

Koller, F. und Burmester, M. 2010. *Technik natürlich nutzen – NUI-Design in der Praxis.* s.l. : Usability Professionals, 2010.

Köth, Y. 2001. *User Interface für ein generisches Modellierungswerkzeug.* Dresden : Technische Universität Dresden, 2001.

Krasner, G.E. und Pope, S.T. 1988. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. *Journal of Object-Oriented Programming, SIGS Publication.* 5, 1988, S. 26-49.

Kwon, S., Lee, D. und Chung, Min K. 2009. Effect of key size and activation area on the performance of a regional error correction method in a touch-screen QWERTY keyboard. *International Journal of Industrial Ergonomics.* 2009, Bd. 39, 5, S. 888-893.

Leymann, F. 2008. *MDA.* [PDF] Stuttgart : Institute of Architecture of Application Systems, 2008.

Lovett, C. 2006. Microsoft XML Team's WebLog. [Online] 16. November 2006. http://blogs.msdn.com/b/xmlteam/archive/2007/11/16/chris-lovett-interview.aspx.

Ludewig, J. und Lichter, H. 2007. *Software Engineering.* Stuttgart : dpunkt.verlag, 2007.

Lutteroth, C. 2008. Automated Reverse Engineering of Hard-Coded GUI Layouts. *AUIC '08 Proceedings of the ninth conference on Australasian user interface.* 2008, Bd. 76.

MacDonald, M. 2010. *Pro WPF in C# 2010.* 1. New York : Apress, 2010.

Machate, J. 2003. *User Interface Tuning.* Frankfurt : Software & Support Verlag, GmbH, 2003.

Mangano, S. 2006. *XSLT Kochbuch.* 2. Köln : O'Reilly, 2006.

Matejka, J., et al. 2009. The Design and Evaluation of Multi-Finger Mouse Emulation Techniques. *CHI '09 Proceedings of the 27th international conference on Human factors in computing systems.* 2009.

Mauney, D. 2010. TouchThinking - Gesture Research Part 1 - 4. [Online] 20. Mai 2010. http://www.touchthinking.com.

Meixner, G. und Görlich, D. 2008. *Aufgabenmodellierung als Kernelement eines nutzerzentrierten Entwicklungsprozesses für Bedienoberflächen.* Kaiserslautern : s.n., 2008.

Microsoft. 2011. About Task Dialogs. [Online] 2011. http://msdn.microsoft.com/en-us/library/bb760441%28VS.85%29.aspx.

—. 2011. Add-ins and Extensibility. [Online] 2011. http://msdn.microsoft.com/de-de/library/bb384200.aspx.

—. 2011. Application Domains Overview. [Online] 2011. http://msdn.microsoft.com/en-us/library/2bh4z9hs%28VS.71%29.aspx.

—. 2011. Attributes. [Online] 2011. http://msdn.microsoft.com/en-us/library/z0w1kczw.aspx.

—. 2011. Dependency Properties Overview. [Online] 2011. http://msdn.microsoft.com/en-us/library/ms752914.aspx.

—. 2009. Microsoft News Center. [Online] 2009. http://www.microsoft.com/presspass/features/2009/Apr09/04-06SurfaceHIMSS2.mspx.

—. MSDN Developer Center - CONTROL Control. [Online] http://msdn.microsoft.com/en-us/library/aa380911%28VS.85%29.aspx.

—. MSDN Library: Touch (Gestaltungsrichtlinie). *http://msdn.microsoft.com/en-us/library/cc872774.aspx.* [Online]

—. MSDN Library: Windows Touch SDK. *http://msdn.microsoft.com/en-us/library/dd562197%28VS.85%29.aspx.* [Online]

—. 2011. Resource-Definition Statements. [Online] 19. April 2011. http://msdn.microsoft.com/en-us/library/aa381043%28VS.85%29.aspx.

—. 2011. Serialization Limitations of XamlWriter.Save. [Online] 2011. http://msdn.microsoft.com/de-de/library/ms754193.aspx.

—. 2003. *UNIX Application Migration Guide.* Redmond : s.n., 2003.

—. Windows SDK. [Online] http://msdn.microsoft.com/de-de/windows/bb980924.

—. 2009. WPF-Anwendungen mit dem Model-View-ViewModel-Entwurfsmuster. *MSDN Magazin.* 2009, Februar.

Miller, J. und Mukerji, J. 2003. *MDA Guide Version 1.01.* [PDF] s.l. : OMG, 2003. omg/2003-06-01.

Moore, M. M. und Rugaber, S. 1993. Issues in User Interface Migration. *Proceeding of 3rd Internation Software Engineering Research Forum.* 1993.

Moore, M. 1995. Reverse Engineering User Interfaces - A Technique. *Proceedings of the 1995 Software Developer's Conference.* 1995.

MSDN. 2011. ObservableCollection. [Online] 2011. http://msdn.microsoft.com/de-de/library/ms668604.aspx.

Musgrave, D. 2009. MSDN Blogs: Windows 7, bitmap fonts and Microsoft Dynamics GP. [Online] Microsoft, 24. November 2009. http://blogs.msdn.com/b/developingfordynamicsgp/archive/2009/11/25/windows-7-bitmap-fonts-and-microsoft-dynamics-gp.aspx.

Novak, J. und Schmidt, S. 2009. When joy matters: the importance of hedonic stimulation in collocated collaboration with large-displays. *12th IFIP TC 13 International Conference.* 2009, 5727, S. 618-629.

OMG. 2008. MOF - Model to Text Transformation Language. [Online] Januar 2008. http://www.omg.org/spec/MOFM2T.

—. 2011. QVT - Query/View/Transformation. [Online] Januar 2011. http://www.omg.org/spec/QVT.

Oracle. Netbeans. [Online] http://netbeans.org.

Parhi, P. und Oulu, M. 2006. Target Size Study for One-Handed Thumb Use on Small Touchscreen Devices. *In Proc. MobileHCI 2006.* 2006, S. 203-210.

Paternò, F. 2005. Model-based tools for pervasive usability. *Interacting with Computers.* 2005, Bd. 17, S. 291-315.

Paternò, F., Mancini, C. und Meniconi, S. 1997. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. *INTERACT '97 Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction.* 1997.

Paternò, F., Santoro, C. und Scorcia, A. 2008. User Interface Migration between Mobile Devices and Digital TV. *2nd Conference on Human-Centered Software Engineering.* 2008, S. 287-292.

Paternò, F., Santoro, C. und Spano, Lucio D. 2009. MARIA: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments. [Hrsg.] ACM. *Transactions on Computer-Human Interaction.* November 2009, Bd. 16, 4, S. 30.

Paulenz, M.D. 2010. *Modellgestütztes End-User-Development für Multi-Touch-Benutzungsschnittstellen.* Stuttgart : s.n., 2010.

Pavlovic, V.I., Sharma, R., Huang, T.S. 1997. Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review. *IEEE TPAMI.* 1997, S. 677–695.

Petzold, C. 1999. *Windows Programmierung.* 5. s.l. : Microsoft Press, 1999.

Plaisant, C. und Wallace, D. 1992. Touchscreen toggle design. *Proceedings of the SIGCHI conference on Human factors in computing systems.* 1992, S. 667-668.

Porteck, S. 2011. Anfassbar - Sieben Touchscreen-Monitore ab 22 Zoll. *c't - Magazin für Computer Technik.* 2011, 5.

Rädle, R. 2009. Squidy - A Zoomable Design Environment for Natural User Interfaces. *In CHI EA '09: Proceedings of the 27th international conference extended abstracts on Human factors in computing systems.* 2009, S. 4561-4566.

Raschke, M., et al. 2010. *Evaluation of different interaction techniques for touch devices.* Stuttgart : Universität Stuttgart, 2010.

Richter, B. 2007. An XAML Serializer Preserving Bindings. [Online] 1. November 2007. http://www.codeproject.com/KB/WPF/XamlSerializer.aspx.

Roth, Tim. 2008. MultiTouch Dev Blog. [Online] 9. Juni 2008. http://iad.projects.zhdk.ch/multitouch/?p=90.

Rozlog, M. 2009. DevX.com - Top Five Touch UI-Related Design Guidelines. [Online] 2. November 2009. http://www.devx.com/enterprise/Article/43185.

Salminen, T., Hosio, S. und Riekki, J. 2007. Middleware based user interface migration: implementation and evaluation. *4th International Conference on Mobile Technology.* 2007, S. 358-363.

Sánchez Ramón, Ó., Sánchez Cuadrado, J. und García Molina, J. 2010. Model-Driven Reverse Engineering of Legacy Graphical User Interfaces. *Proceedings of the IEEE/ACM international conference on Automated software engineering.* 2010, S. 147-150.

Saxonica Limited. 2010. The XSLT and XQuery Processor. [Online] 30. Oktober 2010. http://saxon.sourceforge.net/.

Schlegel, T., et al. 2010. Evaluation of current User Interface Generator Frameworks for Graphical Interactive Systems. *IADIS International Conferences Interfaces and Human Computer Interaction and Game and Entertainment Technologies.* 2010, S. 385-390.

Schmid, D. 2010. *Modellgetriebene generative Softwareentwicklung.* [Vorlesung] Karlsruhe : s.n., 2010. “Entwurf eingebetteter Systeme”.

Schmidt, T. 2008. *Interaction Concepts for Multi-Touch User Interfaces: Design and Implementations.* Konstanz : Universität Konstanz, 2008.

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

Sears, A. und Shneiderman, B. 1991. High Precision Touchscreens: Design Strategies and Comparisons with a Mouse. *International Journal of Man-Machine Studies.* 1991, 34, S. 593-613.

Seidewitz, E. 2003. What models mean. *Software IEEE.* Sept.-Okt. 2003, Bd. 20, 5, S. 26-32.

Stackoverflow. 2009. What is the current state of XSLT 2.0 availability within .NET. [Online] 6. Mai 2009. http://stackoverflow.com/questions/831300/what-is-the-current-state-of-xslt-2-0-availability-within-net/831321#831321.

Stewart, C., et al. 2006. EXSLT Downloads. [Online] 2006.

Sun. 2005. *The Java Language Specification.* 3. California, U.S.A. : Addison-Wesley, 2005.

TCS. ModelMorf - A Model Transformer. [Online] http://www.tcs-trddc.com/trddc_website/ModelMorf/ModelMorf.htm.

Tefkat. Tefkat - The EMF Transformation Engine. [Online] http://tefkat.sourceforge.net.

The Eclipse Foundation. Eclipse Modeling Project. [Online] http://www.eclipse.org/modeling.

tiresias.org. 2009. Touchscreens. [Online] 20. November 2009. http://www.tiresias.org/research/guidelines/touch.htm.

Tyco Electronics. 2010. Funktionsweise von AccuTouch-Touchscreens. [Online] 2010. http://www.elotouch.de/Produkte/Touchscreens/AccuTouch/accworks.asp.

Unidex Inc. 2001. Universal Turing Machine in XSLT. [Online] Unidex, 2001. http://www.unidex.com/turing/utm.htm.

USIXML. USIXML - USer Interface eXtensible Markup Language. [Online] http://www.usixml.org.

VISAM. Touchscreen Technik. [Online] http://www.visam.de/04_service/touch.php.

W3C. 2009. W3C Wiki. [Online] 2009. schreibgeschütztes Wiki. http://www.w3.org/2005/Incubator/model-based-ui/wiki/ConcurTaskTrees.

—. 1999. XPath 1.0 Specification. [Online] 1999. http://www.w3.org/TR/xpath/.

—. 1999. XSL Transformations (XSLT) 1.0. [Online] 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.

—. 2007. XSL Transformations (XSLT) Version 2.0. [Online] 2007. http://www.w3.org/TR/xslt20/.

W3Schools. 2011. XPath. [Online] 2011. http://www.w3schools.com/XPath.

—. 2011. XSLT. [Online] 2011. http://www.w3schools.com/xsl/.

Wang, X., Ghanam, Y. und Maurer, F. 2008. *From Desktop to Tabletop: Migrating the User Interface of AgilePlanner.* Pisa, Italien : Springer-Verlag, 2008. S. 263-270.

Watson, K. 2011. All About Digital Photos - The Myth of DPI. [Online] 2011. http://www.rideau-info.com/photos/mythdpi.html.

Wessel, I. 2002. *GUI-Design.* 2. Berlin : Hanser, 2002.

Wikipedia. 2011. Model-driven architecture. [Online] Permanent Revision ID: 417932142, 2011. http://en.wikipedia.org/w/index.php?title=Model-driven_architecture&oldid=417932142.

Wong, C., Chu, H. und Katagiri, M. 2002. GUI Migration across Heterogenous Java Profiles. *In Proceeding of the ACM SIGCHI-NZ'02.* 2002.

Alle Onlinequellen wurden zuletzt am 27. August 2011 geprüft.

Anhang

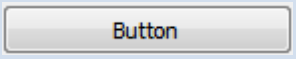
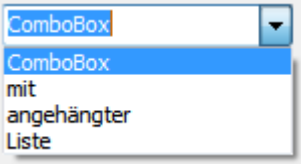
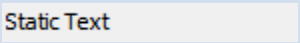
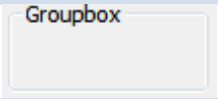
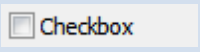
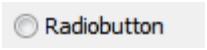
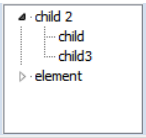
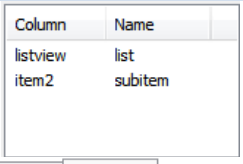
Steuerelementname	Engl. Bezeichner	Symbol	Beschreibung
Befehlsschaltfläche, Schaltknopf, (Druck-)Schalter	button		Ein Schalter zum Auslösen einer Aktion.
Textfeld, Texteingabe, Eingabefeld	edit, edit box, textbox		Ein Feld zur Eingabe von beliebigen textuellen Werten.
Dropdown-Listenfeld	combobox		Ein spezielles Eingabefeld mit Auswahlmöglichkeit aus einer Menge von vorgegebenen Werten. Es kann auch nur auf diese beschränkt sein.
Listenfeld, Liste	listbox		Eine Menge von Elementen, die zur Auswahl stehen. Auch Mehrfachauswahl ist möglich.
Statisches Textfeld, Label	statictext, textblock, label		Ein Anzeigeelement zur Ausgabe von Texten, Werten und Beschriftungen.
Rahmen	groupbox, frame		Ein visuelles Element zur Gruppierung von Elementen.
Kontrollkästchen	checkbox		Ein boolesches Eingabeelement.
Optionsfeld	radiobutton		Ein boolesches Eingabeelement, das abhängig von weiteren Optionsfeldern geschaltet wird.
Schiebregler	slider		Ein Regler zur Eingabe von Werten innerhalb bestimmter Grenzen.
Bildlaufleiste	scrollbar		Ein Steuerelement zur Änderung des sichtbaren Ausschnitts.
Baumansicht	treeview		Ein Steuerelement, dessen Werte in einer hierarchischen Baumansicht dargestellt sind.
Listenansicht	listview		Ein Steuerelement, das Objekte mit deren Eigenschaften darstellen kann.
Register	tabcontrol		Ein Containerelement, das Inhalte durch Karteireiter trennt.
Drehfeld	spinedit		Ein Textfeld zur Eingabe eines Zahlenwertes, der neben der direkten Eingabe auch durch Auf- und Abwärtsschalter erhöht oder verkleinert werden kann.

Tabelle 17 Steuerelemente: Bezeichnung, Symbol und Kurzbeschreibung basierend auf [Petzold, 1999], [Erlenkötter, et al., 1997] und [Wessel, 2002]

Steuerelementname	Embarcadero Delphi Element	Ressource Elemente① (MFC Klasse)	XAML Element
Befehlsschaltfläche, Schaltknopf, (Druck-)Schalter	TButton	PUSHBUTTON (CButton)	Button
Textfeld, Texteingabe, Eingabefeld	TEdit/TMemo	EDITTEXT (CEdit)	TextBox
Dropdown-Listenfeld	TComboBox	COMBOBOX (CComboBox)	ComboBox
Listenfeld, Liste	TListBox	LISTBOX (CListBox)	ListBox
Statisches Textfeld, Label	TStaticText TLabel	CTEXT, LTEXT,RTEXT (CStaticText)	TextBlock Label
Rahmen	TGroupBox	GROUPBOX (CStatic)	GroupBox
Kontrollkästchen	TCheckBox	CONTROL② (CButton)	CheckBox
Optionsfeld	TRadioButton	CONTROL② (int③)	RadioButton
Schiebregler	TSlider	CONTROL② (CSliderCtrl)	Slider
Bildlaufleiste	TScrollBar/ TScrollbox	SCROLLBAR (CScrollBar)	ScrollBar/ScrollViewer
Baumansicht	TTreeView	CONTROL② (CTreeCtrl)	TreeView
Listenansicht	TListView	CONTROL② (CListCtrl)	ListView
Register	TTabControl	CONTROL② (CTabCtrl)	TabControl
Benutzerdefiniertes Element	TControl	CONTROL②	UserControl

① Aus Resource-Definition Statements: **[Microsoft, 2011]**
② Das generische CONTROL Element wird zur Laufzeit durch ein mit Name registriertes Steuerelement ersetzt (CONTROL: **[Microsoft]**)
③ Jedes Optionsfeld wird in MFC als Bitkombinationswert einer Ganzzahl (int) betrachtet.

Tabelle 18 Abbildungstabelle für Steuerelemente von Delphi, Dialog Ressource und XAML

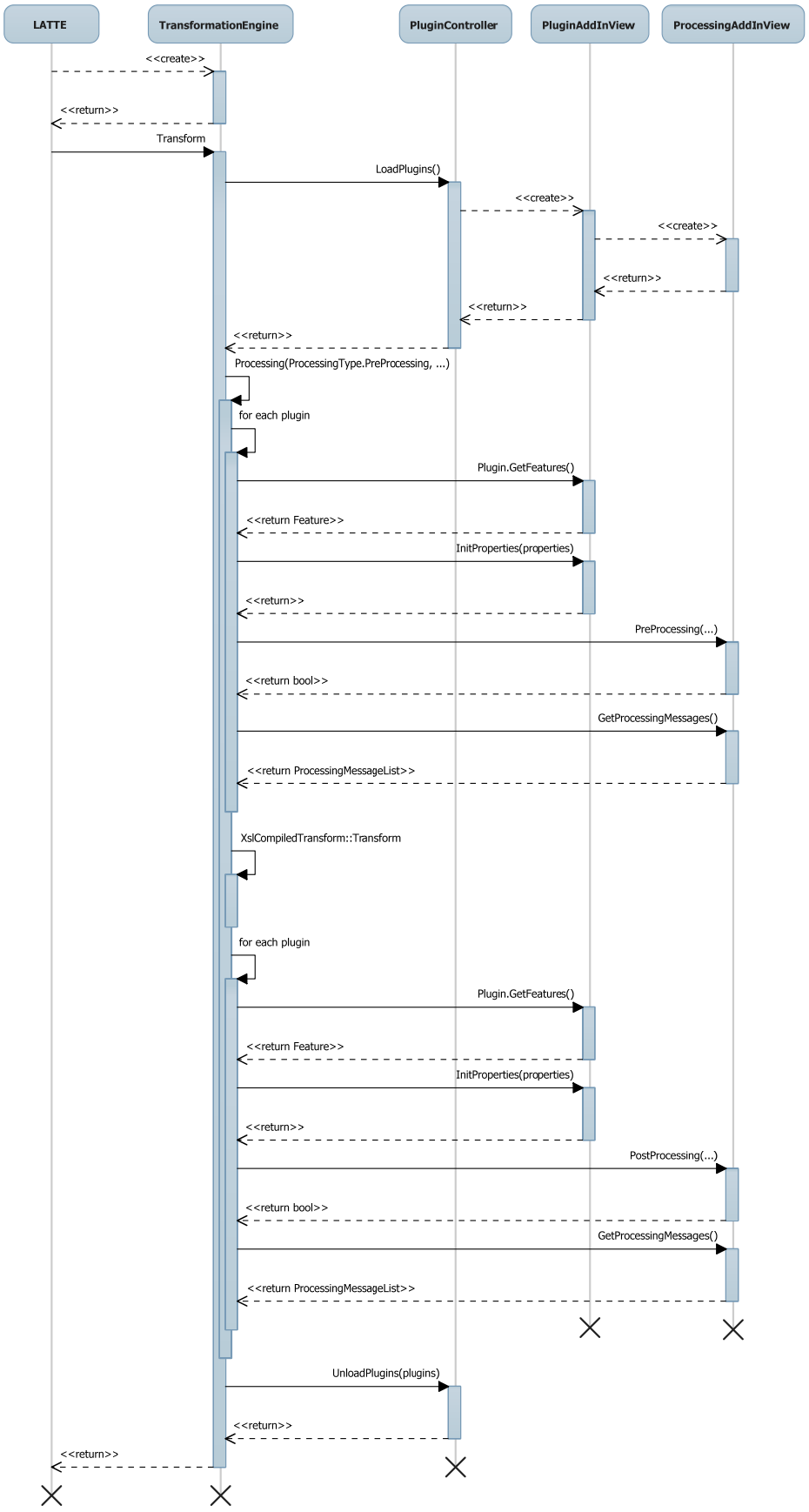


Abbildung 84 Sequenzdiagramm des implementierten Transformationsprozesses

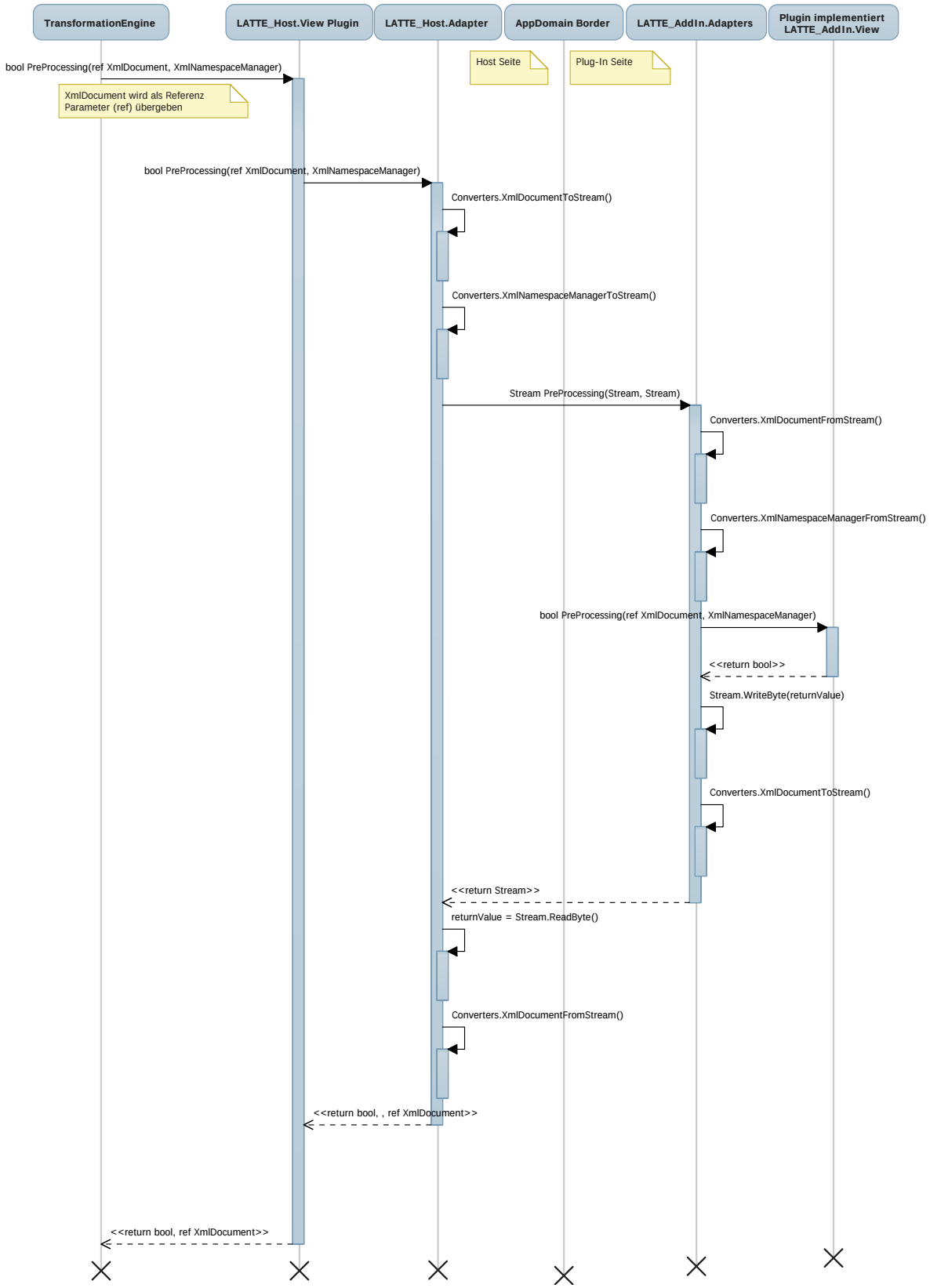


Abbildung 85 Plug-In Methoden Aufruf mit MAF am Beispiel von PreProcessing

```
1  <Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
2      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
3      Title="Confirmation"
4      SizeToContent="WidthAndHeight">
5
6    <Window.Resources>
7      <x:Array xmlns:sys="clr-namespace:System;assembly=mscorlib"
8            x:Key="FileItems"
9            Type="{x:Type sys:String}">
10       <sys:String>Readme.txt</sys:String>
11       <sys:String>ImportantBusiness.doc</sys:String>
12       <sys:String>HotHotHot.jpg</sys:String>
13       <sys:String>Dealer.xls</sys:String>
14     </x:Array>
15   </Window.Resources>
16   <StackPanel Margin="10">
17     <Label Name="label1" Width="309" Height="29"
18         Content="The following tasks are left? What do you want to do?" />
19     <ListBox Name="listBox1" Width="309" Height="73"
20           ItemsSource="{Binding Source={StaticResource FileItems}}"
21           SelectionMode="Extended" />
22     <CheckBox Name="checkBox4" Width="306" Height="16"
23           Content="De-/Select them all"
24           IsChecked="{x:Null}"
25           IsThreeState="True" />
26     <RadioButton Name="checkBox1" Width="123" Height="16"
27           Content="Save selected one" />
28     <RadioButton Name="checkBox2" Width="123" Height="16"
29           Content="Save them all" />
30     <RadioButton Name="checkBox3" Width="123" Height="16"
31           Content="Release them all" />
32     <StackPanel HorizontalAlignment="Center" Orientation="Horizontal">
33       <Button Name="button2" Width="41" Height="23"
34           Margin="0,0,5,0" Content="Ok" />
35       <Button Name="button1" Width="55" Height="23"
36           Content="Cancel" />
37     </StackPanel>
38   </StackPanel>
39 </Window>
```

Quelltext 45 Ausgangsdialog für die Transformation in Kapitel 7.5.2

```
<Window Title="Confirmation" SizeToContent="WidthAndHeight"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 xmlns:touch="clr-namespace:LATTE_Resources;assembly=LATTE_Resources"
 xmlns:xamlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
 <Window.Resources>
  <x:Array x:Key="FileItems" Type="{x:Type sys:String}"
   xmlns:sys="clr-namespace:System;assembly=mscorlib">
   <sys:String>Readme.txt</sys:String>
   <sys:String>ImportantBusiness.doc</sys:String>
   <sys:String>HotHotHot.jpg</sys:String>
   <sys:String>Dealer.xls</sys:String>
  </x:Array>
 </Window.Resources>
 <StackPanel Margin="10">
  <Label Name="label1" Width="309" Height="29"
   Content="The following tasks are left? What do you want to do?" />
  <ListBox Name="listBox1" Width="373" Height="137"
   ItemsSource="{Binding Source={StaticResource FileItems}}"
   SelectionMode="Extended">
   <ListBox.ItemTemplate>
    <DataTemplate>
     <TextBlock Text="{Binding Path=.}" Height="30" />
    </DataTemplate>
   </ListBox.ItemTemplate>
  </ListBox>
  <touch:CheckBoxTouchSwitcher Name="checkBox4"
   Width="322" Height="32" Content="De-/Select them all"
   IsChecked="{x:Null}" IsThreeState="True" />
  <RadioButton Name="checkBox1" Width="139" Height="32"
   Content="Save selected one" />
  <RadioButton Name="checkBox2" Width="139" Height="32"
   Content="Save them all" />
  <RadioButton Name="checkBox3" Width="139" Height="32"
   Content="Release them all" />
  <StackPanel HorizontalAlignment="Center" Orientation="Horizontal">
   <Button Name="button2" Width="57" Height="39"
    Margin="0,0,5,0" Content="Ok" />
   <Button Name="button1" Width="71" Height="39"
    Content="Cancel" />
  </StackPanel>
 </StackPanel>
</Window>
```

Quelltext 46 Quelltext des transformierten Beispieldialogs aus Kapitel 6.3.2

Eingangsfragen

Alter: _____ **F1**

Geschlecht : **F2**

W	Weiblich	M	männlich

Wie lange in der Woche nutzen Sie einen PC zum Arbeiten, im Internet surfen, spielen usw.? **F3**
(Mehrfachnennung möglich)

A	gar nicht	D	3 bis 4 Stunden
B	weniger als eine Stunde	E	4 bis 10 Stunden
C	1 bis 2 Stunden	F	mehr als 10 Stunden

Welche berührungsempfindlichen Geräte haben Sie **bereits einmal** genutzt? **F4**
(Mehrfachnennung möglich)

A	gar keine	F	Öffentliche Systeme (Fahrkartenautomat)
B	Smpartphone / Mobiltelefon	G	Navigationsgeräte
C	Tablet-PC mit Tastatur	H	Kopiergeräte und/oder Drucker
D	Tablet-PC (z.B. iPad) ohne Tastatur	I	Berührungsempfindlicher Monitor mit Standfuß (TFT, LCD usw.) für PC
E	sonstige, bitte unten angeben	J	Laptop mit berührungsempfindlichen Bildschirm

---

Welche berührungsempfindlichen Geräte nutzen Sie regelmäßig? **F5**
(Mehrfachnennung möglich)

A	gar keine	F	Öffentliche Systeme (Fahrkartenautomat)
B	Smpartphone / Mobiltelefon	G	Navigationsgeräte
C	Tablet-PC mit Tastatur	H	Kopiergeräte und/oder Drucker
D	Tablet-PC (z.B. iPad) ohne Tastatur	I	Berührungsempfindlicher Monitor mit Standfuß (TFT, LCD usw.) für PC
E	sonstige, bitte unten angeben	J	Laptop mit berührungsempfindlichen Bildschirm

---

Wie lange nutzen Sie Ihr berührungsempfindliches Gerät pro Woche? (Nur ein Kästchen ausfüllen) **F6**

A	gar nicht	D	3 bis 4 Stunden
B	weniger als eine Stunde	E	4 bis 10 Stunden
C	1 bis 2 Stunden	F	mehr als 10 Stunden

Welche der folgenden Softwaretypen haben Sie bereits mehrmals genutzt? **F7**

A	gar keine	D	Medienwiedergabe (Musik, DVD usw.)
B	Bildbearbeitung / CAD / Videoschnitt	E	Programmierumgebung (IDE)
C	Textverarbeitung / Office-Anwendung	F	Internet-Browser

Welche der folgenden Systeme haben Sie bereits für Spiele benutzt? (Mehrfachnennung möglich) **F8**

A	gar keine	F	XBox ohne Kinect (Kamerasteuerung)
B	Nintendo DS/3DS	G	XBox mit Kinect (Kamerasteuerung)
C	Nintendo Wii	H	Handy
D	Playstation 3 ohne Kamera	I	PC
E	Playstation 3 mit Kamera		

Welche Erwartungen haben Sie an die Bedienung eines berührungsempfindlichen Bildschirms am Desktop-PC / Laptop? **F9**

Aufgabe Ö1

Wie beurteilen Sie die Bedienung des Dialogs während der gesamten Aufgabe? **F14**
(Pro Reihe maximal ein Kreuz)

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Listenfeld (Dateianzeige),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben Ö1, Ö2, Ö3, Ö4, Ö5) **F16**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Wie beurteilen Sie die Bedienung des Steuerelements **Baumansicht** (**Ordneranzeige),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben Ö1, Ö2) **F17**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Aufgabe Ö2

Wie beurteilen Sie die Bedienung des Dialogs während der gesamten Aufgabe? **F14**
(Pro Reihe maximal ein Kreuz)

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Listenfeld (Dateianzeige),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben Ö1, Ö2, Ö3, Ö4, Ö5) **F16**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Wie beurteilen Sie die Bedienung des Steuerelements **Baumansicht** (**Ordneranzeige),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben Ö1, Ö2) **F17**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Aufgabe Ö3

Wie beurteilen Sie die Bedienung des Dialogs während der gesamten Aufgabe? **F14**
(Pro Reihe maximal ein Kreuz)

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Listenfeld (Dateianzeige),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben Ö1, Ö2, Ö3, Ö4, Ö5) **F16**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Aufgabe Ö4

Wie beurteilen Sie die Bedienung des Dialogs während der gesamten Aufgabe? **F14**
(Pro Reihe maximal ein Kreuz)

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Listenfeld (Dateianzeige),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben Ö1, Ö2, Ö3, Ö4, Ö5) **F16**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Aufgabe Ö5

Wie beurteilen Sie die Bedienung des Dialogs während der gesamten Aufgabe? **F14**
(Pro Reihe maximal ein Kreuz)

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Listenfeld (Dateianzeige),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben Ö1, Ö2, Ö3, Ö4, Ö5) **F16**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Wie beurteilen Sie die Bedienung des Steuerelements **Navigationsbar** (**grüne Pfeile),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben Ö5, Ö6, Ö7) **F19**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Aufgabe Ö6

Wie beurteilen Sie die Bedienung des Dialogs während der gesamten Aufgabe? **F14**
(Pro Reihe maximal ein Kreuz)

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Listenfeld** (**Ordneranzeige),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben Ö6, Ö7) **F18**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Wie beurteilen Sie die Bedienung des Steuerelements **Navigationsbar** (**grüne Pfeile),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben Ö5, Ö6, Ö7) **F19**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Aufgabe Ö7

Wie beurteilen Sie die Bedienung des Dialogs während der gesamten Aufgabe? **F14**
(Pro Reihe maximal ein Kreuz)

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Listenfeld** (**Ordneranzeige),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben Ö6, Ö7) **F18**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Wie beurteilen Sie die Bedienung des Steuerelements **Navigationsbar** (**grüne Pfeile),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben Ö5, Ö6, Ö7) **F19**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Aufgabe S1

Wie beurteilen Sie die Bedienung des Dialogs während der gesamten Aufgabe? **F14**
(Pro Reihe maximal ein Kreuz)

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Drehfeld (Ränder-Zahlenfelder),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S1, S2, S3, S4, S5, S6) **F20**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Wie beurteilen Sie die Bedienung des Steuerelements **Liste mit Kontrollkästchen,** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S1, S3, S4, S5, S6, S7) **F22**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Aufgabe S2

Wie beurteilen Sie die Bedienung des Dialogs während der gesamten Aufgabe? **F14**
(Pro Reihe maximal ein Kreuz)

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Drehfeld (Ränder-Zahlenfelder),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S1, S2, S3, S4, S5, S6) **F20**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Wie beurteilen Sie die Bedienung des Steuerelements **Liste mit Kontrollkästchen,** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S1, S2, S3, S4, S5, S6, S7) **F22**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Aufgabe S3

Wie beurteilen Sie die Bedienung des Dialogs während der gesamten Aufgabe? **F14**
(Pro Reihe maximal ein Kreuz)

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Drehfeld (Ränder-Zahlenfelder),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S1, S2, S3, S4, S5, S6) **F20**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Wie beurteilen Sie die Bedienung des Steuerelements **Liste mit Kontrollkästchen,** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S1, S2, S3, S4, S5, S6, S7) **F22**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Aufgabe S4

Wie beurteilen Sie die Bedienung des Dialogs während der gesamten Aufgabe? **F14**
(Pro Reihe maximal ein Kreuz)

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Drehfeld (Ränder-Zahlenfelder),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S1, S2, S3, S4, S5, S6) **F20**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Wie beurteilen Sie die Bedienung des Steuerelements **Liste mit Kontrollkästchen,** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S1, S2, S3, S4, S5, S6, S7) **F22**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Aufgabe S5

Wie beurteilen Sie die Bedienung des Dialogs während der gesamten Aufgabe? **F14**
(Pro Reihe maximal ein Kreuz)

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Drehfeld (Ränder-Zahlenfelder),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S1, S2, S3, S4, S5, S6) **F20**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Wie beurteilen Sie die Bedienung des Steuerelements **Liste mit Kontrollkästchen,** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S1, S2, S3, S4, S5, S6, S7) **F22**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Aufgabe S6

Wie beurteilen Sie die Bedienung des Dialogs während der gesamten Aufgabe? **F14**
(Pro Reihe maximal ein Kreuz)

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Drehfeld (Ränder-Zahlenfelder),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S1, S2, S3, S4, S5, S6) **F20**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Wie beurteilen Sie die Bedienung des Steuerelements **Liste mit Kontrollkästchen,** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S1, S2, S3, S4, S5, S6, S7) **F22**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Aufgabe S7

Teil 1

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Liste mit Kontrollkästchen,** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S1, S2, S3, S4, S5, S6, S7) **F22**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Teil 2

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Liste mit Kontrollkästchen,** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S1, S2, S3, S4, S5, S6, S7) **F22**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Teil 3

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Liste mit Kontrollkästchen,** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S1, S2, S3, S4, S5, S6, S7) **F22**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Teil 4

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Liste mit Kontrollkästchen,** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S1, S2, S3, S4, S5, S6, S7) **F22**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Wie beurteilen Sie die Bedienung des Dialogs während der gesamten Aufgabe? **F14**
(Pro Reihe maximal ein Kreuz)

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Aufgabe S8

Wie beurteilen Sie die Bedienung des Dialogs während der gesamten Aufgabe? **F14**
(Pro Reihe maximal ein Kreuz)

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Tastenfeld für Zahleneingabe (am Drehfeld),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S8, S9) **F21**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Aufgabe S9

Wie beurteilen Sie die Bedienung des Dialogs während der gesamten Aufgabe? **F14**
(Pro Reihe maximal ein Kreuz)

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Bitte bewerten Sie die Aufgabe nach den folgenden Kriterien: **F15**
(Pro Reihe maximal ein Kreuz)

Die optische Darstellung des Dialogs gefiel mir.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Ich konnte Steuerelemente einfach treffen/anklicken/antippen.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu
Der Einsatz von Maus/Finger in dieser Aufgabe fiel mir leicht.	Trifft nicht zu	1	2	3	4	5	6	7	8	Trifft zu

Wie beurteilen Sie die Bedienung des Steuerelements **Tastenfeld für Zahleneingabe (am Drehfeld),** wenn vorhanden, während der Aufgabe? (Pro Reihe maximal ein Kreuz) (Aufgaben S8, S9) **F21**

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Anschlussfragen

Wurden Ihre Erwartungen an die Bedienung des berührungsempfindlichen Bildschirms erfüllt? **F10**
(Nur ein Kästchen ausfüllen)

A	Ja
B	Nein, weil …

Können Sie sich vorstellen einen berührungsempfindlichen Bildschirm am PC oder Laptop zu verwenden? (Nur ein Kästchen ausfüllen) **F11**

A	Ja	B	Nein
C	Nur in Kombination mit Tastatur	D	Nur in Kombination mit Tastatur und Maus

Welche Anwendung(-en) können Sie sich als Touch-Anwendung für PC oder Laptop vorstellen? **F12**

Wie empfanden Sie den Umgang mit dem berührungsempfindlichen Bildschirm **generell**? **F13**
(Pro Reihe maximal ein Kreuz)

Frustrierend	1	2	3	4	5	6	7	8	Motivierend
Langweilig	1	2	3	4	5	6	7	8	Stimulierend / Spannend
Schwierig	1	2	3	4	5	6	7	8	Leicht
Reagiert langsam	1	2	3	4	5	6	7	8	Reagiert schnell
Ungewohnt	1	2	3	4	5	6	7	8	Gewohnt
Körperlich sehr anstrengend	1	2	3	4	5	6	7	8	Körperlich kaum anstrengend

Erklärung

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

Christian Wimmer