H. Reiterer & O. Deussen (Hrsg.): Mensch & Computer 2012
München: Oldenbourg Verlag, 2012, S. 93-102

Migration und Anpassung von Benutzeroberflächen für Touchscreens

Christian Wimmer[1], Steffen Lohmann[1], Michael Raschke[1], Thomas Schlegel[2]

Institut für Visualisierung und Interaktive Systeme, Universität Stuttgart[1]
Institut für Software- und Multimediatechnik, Technische Universität Dresden[2]

Zusammenfassung

Obwohl sich Touchscreens zunehmender Beliebtheit erfreuen, sind die grafischen Benutzeroberflächen vieler Anwendungen für die Eingabe per Finger nicht geeignet. In diesem Beitrag wird deshalb ein Ansatz präsentiert, der die teilautomatisierte Migration und Anpassung von grafischen Benutzeroberflächen für Touchscreens ermöglicht. Eine Umsetzung für Benutzeroberflächen, die in einer XML-basierten Beschreibungssprache definiert wurden, demonstriert die Anwendbarkeit des Ansatzes. Die Migration wird mittels einer parametrisierbaren Transformationsvorschrift in XSLT durchgeführt und kann über Plugins flexibel erweitert werden. Darüber hinaus werden Anpassungsmöglichkeiten für häufig verwendete Steuerelemente diskutiert und in einer qualitativen Nutzerstudie im Kontext des Ansatzes erprobt.

1 Einleitung

Die Verbreitung von Touchscreens hat in den letzten Jahren stark zugenommen. Besonders im Bereich mobiler Geräte ist die Eingabe per Finger inzwischen allgegenwärtig, aber auch Notebooks und Desktop-Computer werden zunehmend mit Touchscreens ausgestattet. Demgegenüber steht eine Vielzahl von Anwendungen, die für die Bedienung mit Fingern nicht geeignet sind, da sie für Eingaben per Maus und Tastatur konzipiert wurden. Eine Anpassung dieser Anwendungen für die Benutzung auf Touchscreens wäre zwar oftmals wünschenswert, doch werden Aufwand und Kosten hierfür gescheut, insbesondere bei komplexen Anwendungen mit vielen Benutzerdialogen.

Ein Hauptproblem bei der Migration und Anpassung von Benutzeroberflächen für Touchscreens ist die fehlende Unterstützung für Entwickler. Zwar beinhalten die Gestaltungsrichtlinien moderner Betriebssysteme zunehmend Empfehlungen für eine erfolgreiche Touch-Interaktion, doch sind diese v.a. auf die Neugestaltung von Benutzeroberflächen ausgerichtet, nicht auf die Umgestaltung vorhandener Oberflächen. Da es meist eine Reihe von alter-

nativen Umsetzungsmöglichkeiten gibt, können zudem leicht Inkonsistenzen entstehen, insbesondere wenn mehrere Entwickler die Benutzeroberflächen für Touchscreens anpassen.

Um diesen Problemen zu begegnen, haben wir einen Ansatz entwickelt, der die Migration und Anpassung von Benutzeroberflächen für Touchscreens unterstützt. Durch parametrisierbare Transformationsvorschriften wird eine teilautomatisierte Migration ermöglicht, die sich über Plugins auf die spezifischen Besonderheiten der Benutzeroberfläche anpassen lässt. Auf diese Weise wird der Migrationsprozess geleitet, was den Entwickler entlastet und gerade bei komplexen Anwendungen mit vielen Benutzerdialogen zu einer Reduzierung des Aufwands beitragen und eine konsistente Transformation von Steuerelementen ermöglichen kann.

In Abschnitt 2 gehen wir zunächst allgemein auf die Migration von Benutzungsschnittstellen ein und fassen wesentliche Arbeiten in diesem Bereich zusammen. Anschließend stellen wir in Abschnitt 3 grundlegende Transformationsstrategien und spezifische Regeln vor, die wir bei der Migration verwenden. In Abschnitt 4 präsentieren wir die Architektur und in Abschnitt 5 die Umsetzung unseres Ansatzes. Schließlich geben wir in Abschnitt 6 die Ergebnisse einer ersten Nutzerstudie wieder, in der wir den Ansatz getestet haben.

2 Migration von Benutzungsschnittstellen

Die Migration von Benutzungsschnittstellen hat mit der zunehmenden Vielfalt an interaktiven Geräten (wie Smartphone, Desktop-PC, Tablet-PC, interaktiver Fernseher, etc.) und Anwendungskontexten in den letzten Jahren stark an Bedeutung gewonnen. Das übergeordnete Ziel ist häufig, die grundsätzliche Bedienbarkeit der Anwendungen vom jeweiligen Gerät abzukoppeln. Bestenfalls soll der Nutzer nach einem Gerätewechsel die Interaktion dort fortsetzen können, wo er sie zuvor unterbrochen hat. Da die verwendeten Geräte unterschiedliche Anforderungen haben (bzgl. Bildschirmgröße, Rechenleistung, etc.), müssen die Benutzungsschnittstellen entsprechend migriert und angepasst werden.

Ein Forschungsgebiet, das diese Herausforderungen adressiert, ist die modellbasierte Entwicklung von Benutzungsschnittstellen (Meixner et al. 2011). Die Benutzungsschnittstellen werden hierbei auf Basis von abstrakten Modellen entwickelt, die verschiedene Aspekte der Interaktion beschreiben (z.B. Aufgaben-, Dialog- und Präsentationsmodell). Die Modelle sind in einer plattform- und geräteunabhängigen Beschreibungssprache wie UsiXML (Limbourg & Vanderdonckt 2004), UIML (Abrams et al. 1999) oder XIML (Puerta & Eisenstein 2002) definiert, aus denen dann die konkreten Benutzungsschnittstellen erstellt werden. Hierbei unterscheidet man typischerweise zwischen Ansätzen, die die Modelle zur Entwurfszeit nutzen (z.B. Paternò et al. 2009) und Ansätzen, bei denen die Benutzungsschnittstellen zur Laufzeit erzeugt werden (z.B. Lohmann et al. 2006).

Im Gegensatz zu diesen Ansätzen liegen bei dem von uns betrachteten Problem meist keine umfassenden Modelle der Benutzungsschnittstelle vor. Der Regelfall ist eher, dass die zu migrierenden Benutzungsschnittstellen mit unterschiedlichen Entwicklungsmethoden erstellt wurden und häufig kaum mehr als das Endprodukt für die Migration zur Verfügung steht.

Zwar gibt es Arbeiten, die die Erstellung von abstrakten Modellen aus bestehenden Benutzungsschnittstellen (sog. Reverse Engineering) unterstützen (z.B. Di Santo & Zimeo 2007, Paganelli & Paternò 2002), doch bedeutet dies zusätzlichen Migrationsaufwand. Wir haben uns deshalb entschieden, nicht von umfassenden Modellen auszugehen, sondern lediglich eine XML-basierte Beschreibung der Benutzungsschnittstellen vorauszusetzen. Für die konkrete Umsetzung haben wir letztlich XAML verwendet, wobei der Ansatz grundsätzlich auch für andere XML-basierte Beschreibungssprachen wie z.B. XUL oder XHTML funktioniert.

3 Transformationsstrategien

Die Idee unseres Ansatzes ist es, grafische Benutzeroberflächen mit Hilfe von Regeln so zu verändern, dass eine Interaktion per Finger ermöglicht wird. Dabei werden drei grundlegende Transformationsstrategien angewandt, die in Abb. 1 beispielhaft illustriert sind:

1. **Vergrößerung:** Häufig sind Steuerelemente zu klein für die Bedienung mit Fingern oder liegen so nah beieinander, dass eine eindeutige Auswahl schwierig ist. Die erste Transformationsstrategie ist deshalb die Vergrößerung dieser Elemente und/oder ihrer Abstände zueinander (z.B. Vergrößerung einer Schaltfläche, um mehr Oberfläche für die Berührung mit dem Finger zu erhalten, Abb. 1a).

2. **Erweiterung:** Einige Steuerelemente sind generell nicht sonderlich geeignet für die Touch-Interaktion. Allerdings sind sie dem Nutzer vertraut, so dass man sie manchmal nicht komplett ersetzen möchte. Diese Steuerelemente werden in der zweiten Transformationsstrategie um zusätzliche Interaktionsbausteine ergänzt, die sich besser für eine Touch-Interaktion eignen (z.B. Erweiterung einer Bildlaufleiste um Schaltflächen, so dass nicht zwangsläufig eine ziehende Bewegung durchgeführt werden muss, Abb. 1b).

3. **Ersetzung:** Letztlich ist es jedoch häufig am sinnvollsten, ungeeignete Steuerelemente komplett durch solche zu ersetzen, die für die Touch-Interaktion besser geeignet sind (z.B. Kalenderauswahl durch Trommelliste, Abb. 1c). Allerdings kann diese dritte Transformationsstrategie dazu führen, dass sich die Benutzeroberfläche in ihrem Aussehen stark verändert. Gelernte Interaktion geht dadurch leicht verloren, so dass sich der Nutzer neu einlernen muss, was nicht immer erwünscht sein wird.

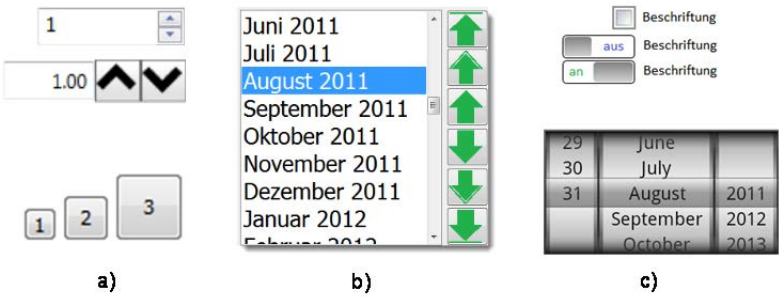


Abbildung 1: Beispiele für die drei verwendeten Transformationsstrategien bei der Migration von Steuerelementen für Touchscreens: a) Vergrößerung, b) Erweiterung, c) Ersetzung

Ausgehend von diesen drei Transformationsstrategien haben wir eine Reihe detaillierter Regeln für die Migration häufig verwendeter Steuerelemente definiert. Die Regeln sind als parametrisierbare Transformationsvorschriften formuliert, die sich auf den jeweiligen Anwendungsfall und Touchscreen-Typ anpassen lassen. Typischerweise wird etwa der Vergrößerungsfaktor für Schaltflächen durch die Größe und Auflösung des Touchscreens bestimmt. Kontrollkästchen können entweder ebenfalls um einen entsprechenden Faktor vergrößert oder aber durch den von Plaisant & Wallace (1992) inspirierten Kippschalter ersetzt werden, wie in Abb. 1c dargestellt.

Andere Transformationsvorschriften berücksichtigen die Touch-Technologie. Beispielsweise ist die bereits erwähnte Erweiterung der Bildlaufleiste um Schaltflächen für solche Touchscreens gedacht, die keine ziehenden Bewegungen auf der Oberfläche erkennen können. Auch weitere Konfigurationen des Anwendungsfalls, wie das Vorhandensein einer physischen Tastatur, sind in den Transformationsvorschriften berücksichtigt. Falls keine Tastatur vorhanden ist, können Textfelder je nach Datentyp um eine virtuelle Tastatur (bei Buchstaben) oder einen virtuellen Ziffernblock (bei Zahlen) ergänzt werden, die automatisch erscheinen, sobald das jeweilige Textfeld aktiviert wird.

Natürlich decken die Transformationsstrategien und -vorschriften nicht alle möglichen Anwendungsfälle ab. Will man etwa die Benutzeroberfläche einer Desktop-Anwendung nicht nur Touchscreen-tauglich machen, sondern auch an die kleinere Bildschirmgröße von Mobilgeräten anpassen, werden im Regelfall weitere Transformationen nötig (Watters & MacKay 2004). Solche zusätzlichen Migrationsanforderungen sollen in diesem Beitrag jedoch nicht weiter betrachtet werden.

4 Architektur

Der von uns entwickelte Ansatz eignet sich für grafische Benutzeroberflächen, die in einer XML-basierten Beschreibungssprache (wie z.B. XAML, XUL, XHTML) vorliegen oder in eine solche übersetzt werden können. Technisch gesehen reduziert sich das zu lösende Problem damit auf die Überführung eines XML-Dokuments in ein anderes, so dass sich für die Umsetzung eine Pipeline-Architektur wie in Abb. 2 dargestellt anbietet.

Zentrale Steuereinheit der Transformations-Pipeline ist der XSLT-Prozessor, der den XML-Quellcode des Ausgangsdialogs in den des Zieldialogs überführt. Dabei können über XSLT-Befehle beliebige XML-Elemente entsprechend den oben genannten Transformationsstrategien vergrößert, erweitert oder ersetzt werden. Die Verwendung von XSLT bietet sich in diesem Fall an, da es speziell für die Transformation von XML-Dokumenten entwickelt wurde. XSLT basiert selbst auch auf XML, so dass es leicht lesbar ist und Transformationsvorschriften entsprechend einfach erstellt und angepasst werden können.

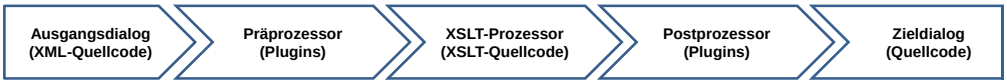


Abbildung 2: Pipeline-Architektur des Ansatzes

Die Definition von komplexen Transformationen in XSLT ist allerdings recht umständlich. Deshalb haben wir die Pipeline-Architektur um eine Möglichkeit ergänzt, in anderen Programmiersprachen geschriebene Transformationsvorschriften als Plugins einzubinden (vgl. Abb. 2). Die Plugins können entweder als Präprozessoren oder als Postprozessoren fungieren: Präprozessoren bereiten das XML-Dokument für die Transformation mit XSLT vor, indem sie z.B. Datenbindungen (wie WPF Bindings) durch ihre eigentlichen Werte ersetzen. Komplementär dazu dienen die als Postprozessoren eingebundenen Plugins der Nachbearbeitung des mit XSLT transformierten XML-Dokuments. Hier werden beispielsweise Datenbindungen wiederhergestellt, die mittels eines Präprozessors ersetzt wurden.

Präprozessoren und XSLT-Transformationsvorschriften können die Postprozessoren steuern, indem sie dem XML-Dokument Annotationen hinzufügen. Annotationen verändern nicht die Darstellung der Benutzeroberfläche, sondern dienen lediglich der Kommunikation entlang der Transformations-Pipeline. Das Ergebnis der Pipeline ist ein für Touchscreens angepasster Dialog, der entweder auch als XML-Quellcode oder aber in einem anderen Ausgabeformat erstellt wird.

5 Umsetzung

Die beschriebene Pipeline-Architektur haben wir im System *LATTE (Legacy Application Transformation to Touch Environments)* für XML-basierte Benutzeroberflächen umgesetzt. LATTE ist im .NET-Framework implementiert und bietet eine integrierte Entwicklungsumgebung zur Erstellung und Bearbeitung von Migrationsprojekten. Die Entwicklungsumgebung besteht im Wesentlichen aus drei Komponenten: Der Benutzeroberfläche, der Transformationseinheit und einem Plugin-System. Aufbau und Funktionen orientieren sich an gängigen Entwicklungsumgebungen, so dass sich Entwickler schnell in die Anwendung einfinden sollten.

Abb. 3 zeigt die Benutzeroberfläche von LATTE, die in drei Bereiche unterteilt ist. Im oberen Bereich wird der XAML-Quellcode von Ausgangs- und Zieldialog angezeigt. Er kann in den Editoren bearbeitet und in einer Vorschau angezeigt werden. Über einen weiteren Editor im mittleren Bereich lassen sich mittels XSLT die Transformationsvorschriften definieren. Die XSLT-Parameter können zusätzlich in einer Listenansicht bearbeitet werden, was ein schnelles Ausprobieren verschiedener Konfigurationen und Anpassen auf den jeweiligen Anwendungsfall ohne Veränderung des XSLT-Quellcodes ermöglicht. Darüber hinaus lassen sich gegebenenfalls eingebundene Plugins aktivieren und deaktivieren und mit XSLT-Parametern verknüpfen. Ein Nachrichtenfenster im unteren Teil rundet die Entwicklungsumgebung ab. Es hilft beim Debugging, indem es Hinweise, Warnungen und Fehler aus allen Teilen der Anwendung (z.B. vom XSLT-Prozessor oder den Plugins) anzeigt.

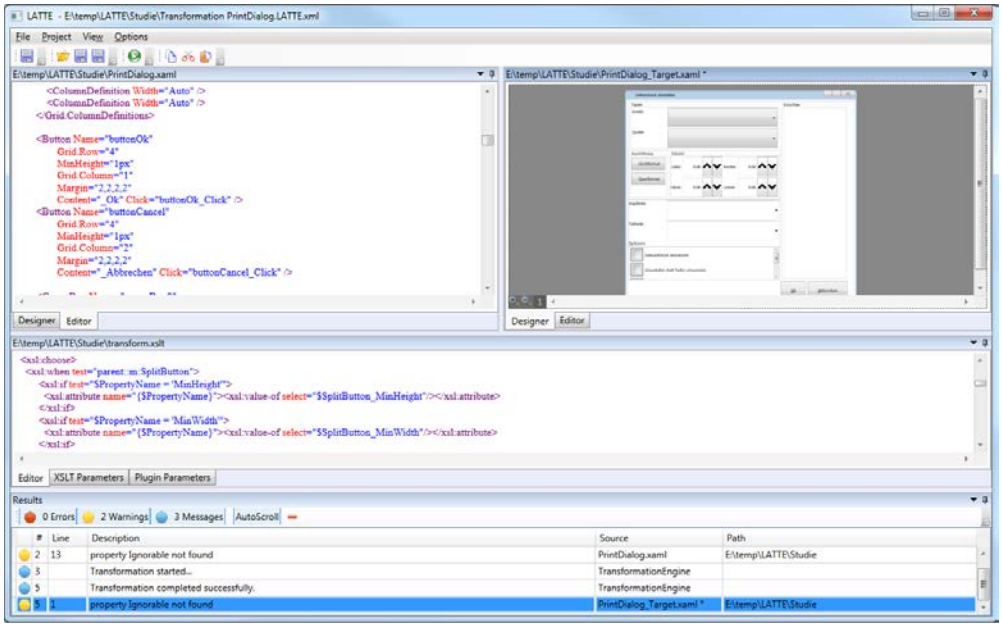
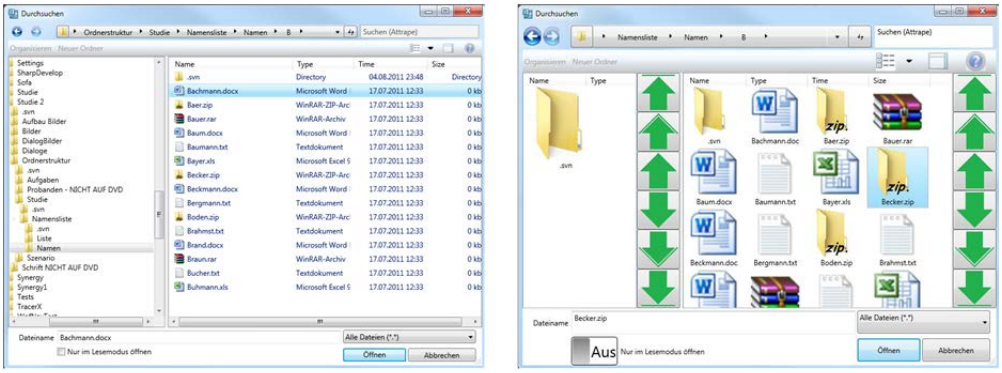


Abbildung 3: Die Benutzeroberfläche von LATTE integriert zwei XAML-Editoren mit Vorschaufunktion, einen XSLT-Editor zur Definition von Transformationsvorschriften, eine Plugin-Verwaltung und ein Nachrichtenfenster.

Die zentrale Komponente von LATTE ist die Transformationseinheit, die den Migrationsprozess entsprechend der Pipeline-Architektur steuert. Sie lädt die Plugins, kompiliert und validiert den XSLT-Quellcode und sendet Fehler an das Nachrichtenfenster. Die Plugins können in einer beliebigen .NET-Programmiersprache (wie C#, VB.NET, etc.) implementiert sein. Sie lassen sich über das Plugin-System jederzeit einbinden oder austauschen. Zudem kann LATTE Transformationsvorschriften als parametrisierbare XSLT-Vorlagen bereitstellen, die lediglich auf den jeweiligen Anwendungsfall angepasst werden müssen.

Abb. 4 zeigt ein Beispiel für eine mit LATTE transformierte Benutzeroberfläche. Es handelt sich um einen ‚Öffnen'-Dialog, wie er in vielen Programmen für das Windows-Betriebssystem in ähnlicher Form zu finden ist. Der Zieldialog (Abb. 4b) wurde mittels der Transformationsstrategien aus dem Ausgangsdialog (Abb. 4a) erstellt. Während die meisten Elemente einfach vergrößert wurden, wurden die Bildlaufleisten, die Ordnerhierarchie und das Optionsfeld ausgetauscht. Diese Ersetzungen fanden auf Basis der spezifischen Merkmale des Touchscreens entsprechend den Regeln aus Abschnitt 3 statt.

Wie in der Abbildung zu erkennen ist, wurden die grundsätzliche Gestaltung der Benutzeroberfläche und deren Funktionalität nicht verändert, sondern lediglich die Steuerelemente angepasst. Durch diese Anpassungen können zwar insgesamt weniger Informationen dargestellt werden, dafür ist der Zieldialog jedoch wesentlich geeigneter für die Touch-Interaktion.



a) b)

Abbildung 4: Beispiel für eine mit LATTE transformierte Benutzeroberfläche: a) Ausgangsdialog, b) Zieldialog

6 Nutzerstudie

In einer ersten qualitativen Nutzerstudie haben wir die Effektivität und Akzeptanz unseres Ansatzes erprobt. Insbesondere wollten wir wissen, ob sich die migrierten Benutzeroberflächen tatsächlich besser für die Touch-Interaktion eignen. Darüber hinaus interessierte uns die Reaktion der Probanden auf die neu eingeführten Steuerelemente.

Für die Studie haben wir zwei Benutzeroberflächen in XAML erstellt und mit LATTE für Touchscreens angepasst. Bei der ersten handelte es sich um den in Abschnitt 5 vorgestellten ‚Öffnen'-Dialog (siehe Abb. 4), die zweite simulierte einen ‚Drucken'-Dialog, wie er im Vorschaufenster in Abb. 3 klein dargestellt ist. Wir entschieden uns für diese zwei Dialogarten, da sie häufig in Programmen vorkommen und gute Möglichkeiten bieten, die neu eingeführten Steuerelemente zu erproben. Während der ‚Öffnen'-Dialog v.a. Listen und Navigationselemente verwendet, besteht der ‚Drucken'-Dialog vorrangig aus Listenfeldern, Drehfeldern und Kontrollkästchen. Zusätzlich setzten wir in zwei Aufgaben den in Abschnitt 3 erwähnten virtuellen Ziffernblock ein. Die Parameter der Transformationsvorschriften haben wir so variiert, dass wir verschiedene Versionen des Zieldialogs mit unterschiedlich großen Steuerelementen und Abständen testen konnten.

An der Studie nahmen 6 Probanden im Alter von 22 bis 61 Jahren teil. Alle waren mit dem Windows-Betriebssystem sehr vertraut und hatten sowohl Erfahrungen mit der Interaktion per Maus und Tastatur als auch mit Touchscreens (v.a. mit Smartphone, Fahrkartenautomat und Navigationssystem). Für jeden Dialog haben wir eine Reihe von Aufgaben erstellt, die die Teilnehmer durchzuführen hatten, wie z.B. eine vorgegebene Datei zu öffnen oder bestimmte Seiteneinstellungen für den Druck auszuwählen. Die Aufgaben wurden zuerst mit Maus und Tastatur und dann per Touch-Interaktion bearbeitet, wobei wir die Variablen der Aufgaben (z.B. die zu öffnende Datei oder die zu wählenden Seiteneinstellungen) jeweils variiert haben.

Vor Beginn der Aufgabendurchführung durften sich die Teilnehmer mit dem verwendeten 23"-Touchscreen vertraut machen. Hierzu wurde die Anwendung ‚Surface Collage' aus dem Microsoft Touch Pack für 5-10 Minuten zum Testen gestartet. Anschließend wurden die Dialoge mittig auf dem Touchscreen bei einer Auflösung von 1920x1080 Pixeln präsentiert. Die Interaktion wurde mit einer Software zur Bildschirmaufnahme als Video für die nachträgliche Auswertung festgehalten. Außerdem sollten die Teilnehmer ihre Aktionen und Gedanken mittels der Methode des lauten Denkens verbalisieren. Die Audiokommentare wurden mit dem Video synchronisiert und gespeichert. Im Anschluss an jede Aufgabendurchführung sollten die Teilnehmer den gezeigten Dialog abschließend bewerten.

Die Ergebnisse der Nutzerstudie unterstrichen zunächst das eingangs genannte Problem: Die Bedienung der Ausgangsdialoge per Finger war wesentlich umständlicher und von mehr Fehleingaben begleitet als die Bedienung derselben Dialoge mit der Maus. Wie vermutet hatten die Probanden Schwierigkeiten, die Elemente aufgrund ihrer geringen Größe korrekt auszuwählen. Hierauf deuten nicht nur die im Vergleich zur Mausinteraktion höheren Fehlerraten hin, sondern auch die Kommentare der Probanden. Während ein Mauszeiger zunächst in Ruhe positioniert werden kann und die Auswahl erst per Tastenklick erfolgt, wird jede Berührung des Touchscreens mit dem Finger direkt als Auswahl gewertet. Zwei Probanden hatten bei der Touch-Interaktion mit den Ausgangsdialogen solche Probleme, dass sie die Aufgaben als „fast nicht machbar" bewerteten. Besonders die Auswahl der kleinen Schalter in den Drehfeldern des ‚Drucken'-Dialogs bereitete Schwierigkeiten. Entsprechend führte die erste Transformationsstrategie der Vergrößerung von Steuerelementen und Abständen bereits zu deutlich besseren Ergebnissen. Die durchschnittliche Fehlerrate sank und die Kommentare der Probanden waren bei den transformierten Dialogen positiver.

Allerdings gilt die Regel ‚je größer, desto besser' nicht pauschal, sondern wurden Vergrößerungen von den Probanden nur bis zu einem gewissen Grad positiv bewertet. Drei Probanden merkten an, dass eine zu starke Vergrößerung der Steuerelemente zu Lasten der Orientierung und Übersicht gehe, was insbesondere in der Auflistung mehrerer Kontrollkästchen im ‚Drucken'-Dialog zu Problemen beim Lösen der Aufgaben führe.

Der virtuelle Ziffernblock als Erweiterung der Textfelder entsprechend der zweiten Transformationsstrategie kam bei fünf der sechs Teilnehmer gut an. Sie favorisierten diese Eingabeform gegenüber den Drehfeldern mit vergrößerten Schaltern (vgl. Abb. 1a). Allerdings kritisierten zwei Teilnehmer, dass die Verknüpfung zwischen Ziffernblock und Textfeld zunächst unklar war und optisch deutlicher hervorgehoben werden sollte, was sich jedoch leicht umsetzen lässt.

Entsprechend der dritten Transformationsstrategie hatten wir in der Nutzerstudie die Bildlaufleiste des ‚Öffnen'-Dialogs durch das Mehrfach-Schaltflächen-Element aus Abb. 1b ersetzt. Die Meinungen hierzu waren geteilt: Während zwei Probanden mit vergleichsweise wenig Touchscreen-Erfahrung dieses Element positiv bewerteten, bevorzugten die vier erfahreneren Probanden Wischgesten, um in der Liste zu blättern, was mit dem verwendeten Touchscreen ebenfalls möglich war. Die Ersetzung des Kontrollkästchens durch den Kippschalter im ‚Öffnen'-Dialog wurde nach anfänglicher Skepsis von allen Teilnehmern positiv bewertet.

Insgesamt zeigte die Nutzerstudie, dass die Migration der Benutzeroberflächen entsprechend der genannten Transformationsstrategien die Bedienbarkeit auf Touchscreens überwiegend verbesserte. Wie zu erwarten verdeutlichte die Studie allerdings auch, dass die migrierten Dialoge nicht in jeder Hinsicht ideal waren und vermutlich kaum gegen Dialoge bestehen könnten, die durch einen erfahrenen Interface-Designer migriert werden.

7 Diskussion und Ausblick

In diesem Beitrag haben wir einen Ansatz zur Migration von Benutzeroberflächen für Touchscreens vorgestellt. Ausgehend von drei grundlegenden Transformationsstrategien haben wir Regeln für die Anpassung von Steuerelementen definiert. Anschließend haben wir diese Regeln als parametrisierbare Transformationsvorschriften in der Entwicklungsumgebung LATTE auf XML-basierte Beschreibungen von Benutzeroberflächen angewandt. Die Ausgangsdialoge werden hierbei unter Verwendung von XSLT und optionalen .NET-Plugins teilautomatisiert in Touchscreen-taugliche Zieldialoge umgewandelt. Die Ergebnisse einer ersten qualitativen Nutzerstudie deuten darauf hin, dass die so migrierten Benutzeroberflächen auf Touchscreens besser zu bedienen sind.

Allerdings ist zu beachten, dass die mit unserem Ansatz migrierten Dialoge nicht in jeder Hinsicht ideal für die Verwendung mit Touchscreens sind. Wir haben in dieser Arbeit weder ästhetische Aspekte betrachtet noch alle möglichen Migrationsformen diskutiert. Im Idealfall müsste man die Benutzeroberflächen für jede einzelne Touchscreen-Konfiguration separat gestalten. Da dies jedoch sehr aufwändig sein kann und leicht zu Inkonsistenten zwischen verschiedenen Systemen führt, war es das Ziel dieser Arbeit, einen Alternativansatz zu entwickeln. Wir sehen den Ansatz besonders dann als hilfreich an, wenn mehrere Dialoge aus einer oder mehreren Anwendungen einheitlich für Touchscreens migriert werden sollen. Dies schließt nicht aus, dass die migrierten Dialoge anschließend noch durch einen Interface-Designer optimiert werden.

Weiterhin ist zu beachten, dass der Ansatz ausschließlich die Migration für Touchscreens behandelt. In Anwendungsfällen, in denen noch weitere Migrationsziele verfolgt werden, wie etwa die Anpassung der Benutzeroberfläche an kleine Bildschirme, werden die angeführten Transformationsstrategien nicht ausreichen. In solchen Fällen müsste der Ansatz entweder um zusätzliche Transformationsstrategien erweitert oder aber mit bestehenden Ansätzen kombiniert werden, was durch die Pipeline-Architektur erleichtert wird.

Zwar basiert unser Ansatz auf XAML, doch lässt er sich grundsätzlich auch auf Benutzeroberflächen anwenden, die in einem anderen Format repräsentiert sind. Das von uns umgesetzte LATTE-System funktioniert für alle XML-basierten Beschreibungssprachen, solange entsprechende Transformationsvorschriften in XSLT und/oder .NET existieren. Neben konkreten Beschreibungssprachen wie XAML, XUL oder XHTML sind damit prinzipiell auch Migrationen zwischen abstrakten Beschreibungssprachen wie XIML, UIML und UsiXML (vgl. Abschnitt 2) möglich.

Der Ansatz lässt sich in verschiedene Richtungen weiterentwickeln. Zum einen kann die Sammlung von Transformationsvorschriften um weitere Regeln und Steuerelemente ergänzt werden. Zum anderen können alternative Eingabeformen (z.B. Stift-Interaktion) und verschiedene Touch-Technologien stärker berücksichtigt werden. Im Idealfall entsteht ein umfassender Katalog mit Anpassungen, Erweiterungen und Ersetzungen für eine Vielzahl von Steuerelementen und Touchscreen-Konfigurationen, so dass eine nahezu automatisierte Migration von großen Teilen der Benutzeroberfläche möglich wird.

Literaturverzeichnis

Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M. & Shuster, J. E. (1999). UIML: An Appliance-Independent XML User Interface Language. *Computer Networks 31*(11-16), 1695-1708.

Di Santo, G. & Zimeo, E. (2007). Reversing GUIs to XIML Descriptions for the Adaptation to Heterogeneous Devices. In: *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC '07)*. New York: ACM, S. 1456-1460.

Limbourg, Q. & Vanderdonckt, J. (2004). UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. In: *Proceedings of Workshops in Connection with the 4th International Conference on Web Engineering (ICWE '04)*. Paramus: Rinton Press, S. 325-338.

Lohmann, S., Kaltz, J. W. & Ziegler, J. (2006). Model-Driven Dynamic Generation of Context-Adaptive Web User Interfaces. In: *Workshops and Symposia at MoDELS 2006*. Berlin/Heidelberg: Springer, S. 116-125.

Meixner, G., Paternò, F. & Vanderdonckt J. (2011). Past, Present, and Future of Model-Based User Interface Development. *i-com, 10*(3), 2-11.

Paganelli, L. & Paternò, F. (2002). Automatic Reconstruction of the Underlying Interaction Design of Web Applications. In: *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*. New York: ACM, S. 439-445.

Paternò, F., Santoro, C. & Spano, L. D. (2009). Model-Based Design of Multi-device Interactive Applications Based on Web Services. In: *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction (INTERACT '09)*. Berlin/Heidelberg: Springer, S. 892-905.

Plaisant, C. & Wallace, D. (1992). Touchscreen Toggle Design. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '92).* New York: ACM, S. 667-668.

Puerta, A. R. & Eisenstein, J. (2002). XIML: A Common Representation for Interaction Data. In: *Proceedings of the 7th International Conference on Intelligent User Interfaces (IUI '02)*. New York: ACM, S. 216-217.

Watters, C. & MacKay, B. (2004). Transformation Volatility and the Gateway Model for Web Page Migration to Small Screen Devices. In: *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS '04)*, Track 4. Washington: IEEE, S. 1530-1605

Kontaktinformationen

Steffen Lohmann, Michael Raschke
{steffen.lohmann,michael.raschke}@vis.uni-stuttgart.de

Jun.-Prof. Dr.-Ing. Thomas Schlegel
thomas.schlegel@tu-dresden.de