Sebastian Feuerstack, Mauro dos Santos Anjo und Ednaldo Brigante Pizzolato

Modellierung und Ausführung von multimodalen Anwendungen auf Basis von Zustandsdiagrammen

Modeling and Execution of Multimodal User Interfaces Based on State-Charts

Multimodale Benutzungsschnittstellen, Modellbasierte Entwicklung, Interaktoren

Zusammenfassung. Neuartige Interaktionsformen, wie die Verwendung von Gesten zur Steuerung von Benutzungsschnittstellen ermöglichen eine Anwendungsbedienung auch in Situationen wo hardwarebasierte Steuerungen nicht verwendbar sind oder können Personen mit körperlichen Beeinträchtigungen unterstützen. Das große Spektrum an denkbaren Gesten bietet zahlreiche Interaktionsmöglichkeiten, erfordert aber auch umfangreiche Nutzertests um eine intuitive, fehlertolerante, sowie zielgruppengerechte Steuerung zu entwickeln.
Dieser Beitrag beschreibt wie mittels der modellbasierten Entwicklung basierend auf Interaktoren, die mit Zustandsdiagrammen beschrieben und Mappings verbunden werden, multimodale Benutzungsschnittstellen entworfen und direkt ausgeführt werden können. Dabei steht die schnelle Entwicklung von verschieden prototypischen Interaktionsvarianten im Vordergrund.

Summary. New forms of interactions such as the gesture-based control of interfaces could enable interaction in situations where hardware controls are missing and support impaired people where other controls fails. The rich spectrum of combining hand postures with movements offers great interaction possibilities but requires extensive user testing to figure out a user interface navigation control with a sufficient performance and a low error rate.
In this paper we describe a model-based interface design based on assembling interactors and multimodal mappings to design multimodal interfaces. We propose to use state charts for the rapid generation of different user interface controls to accelerate user testing.

1. Einleitung

Die Steuerung von grafischen Benutzungsschnittstellen mittels Gesten erlaubt eine Interaktion in Situationen in denen physische Bedienungselemente fehlen, wie beispielsweise bei der Bedienung wandfüllender Bildschirme (Fikkert, 2010) und bieten den Nutzern eine Bedienungsalternative, die die klassische Steuerungsgeräte wie Tastatur und Maus aufgrund von körperlichen Einschränkungen nicht nutzen können. Im Unterschied zu der klassischen Computerbedienung ist der Freiheitsgrad einer Steuerung mittels Gesten nur durch die Kreativität der Nutzer, nicht aber durch ein physisches Steuerungsgerät eingeschränkt.

DOI 10.1524/icom.2011.0031

Dieser Beitrag beschreibt die modellbasierte Entwicklung von multimodalen Schnittstellen auf Basis von Zustandsdiagrammen. Im Gegensatz zu proprietären Modellierungssprachen sind Zustandsdiagramme als Bestandteil von UML oder durch Standards wie beispielsweise Statechart XML schon weitläufig bekannt und durch die geringe Anzahl von wenigen grundlegenden Konzepten (Zustände, Transitionen und Nachrichten) schnell erlernbar. Im Gegensatz zu anderen Arbeiten zur modellbasierten Entwicklung von Benutzungsschnittstellen steht nicht ein Entwurf eines strukturierten Prozesses im Vordergrund dieser Arbeit, sondern eine Modellierungstechnik mit der auf Basis von Zustandsdiagrammen Komponenten einer Schnittstelle, sowie die Eigenschaften der an sie angeschlossenen Modalitäten entworfen werden können. Diese Komponenten nennen wir Interaktoren. Interaktoren lassen sich direkt ausführen. Dazu werden sie zur Laufzeit dynamisch in Zustandsmaschinen übersetzt und können während des Betriebs oder bei Nutzertests inspiziert und auch manipuliert werden.

Der nächste Abschnitt widmet sich der Modellierung der Interaktoren und beschreibt, wie deren Verhalten über Zustandsdiagramme spezifiziert wird. In Abschnitt 3 wird ausgeführt, wie sich mittels Mappings Interaktoren zu multimodalen Benutzungsschnittstellen zusammenfügen lassen. Im darauf folgenden Abschnitt 4 wird gezeigt, wie mittels neu definierter Interaktoren und deren Kombi-

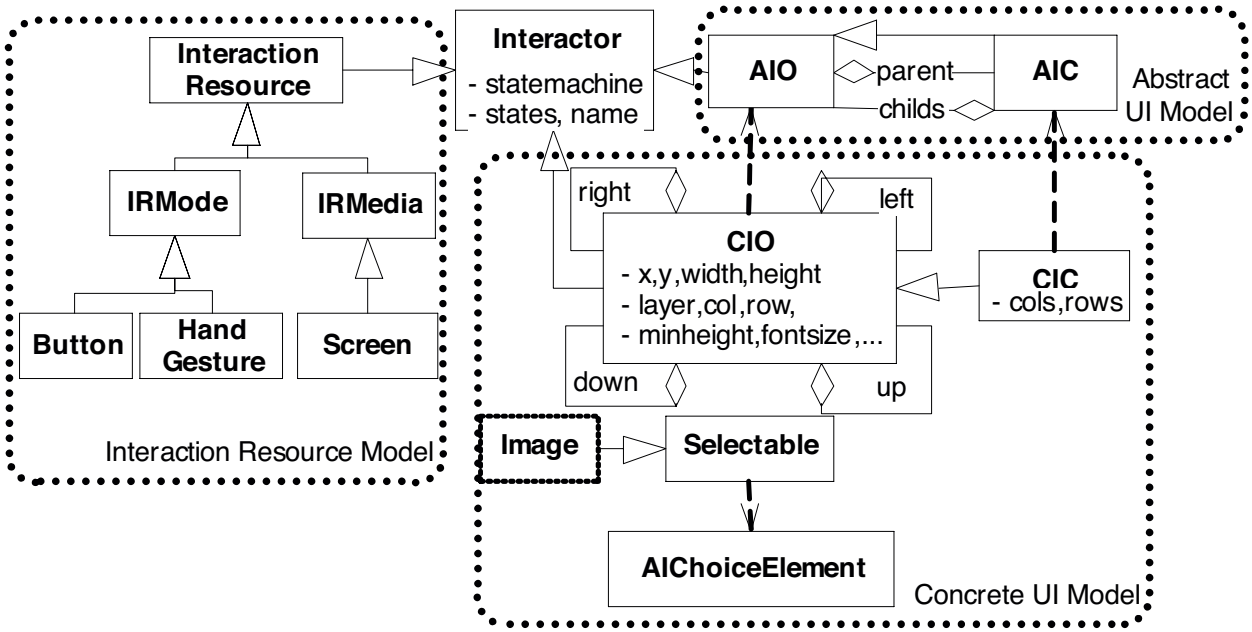


Bild 1: Beziehungen der Interaktoren auf den verschiedenen Modellebenen.

nation durch Mappings verschiedene Varianten einer auf Gesten basierten Schnittstellenavigation umgesetzt wurden. Im Abschnitt 5 wird der Beitrag im Kontext verwandter Arbeiten diskutiert und abschließend in Abschnitt 6 ein Fazit gezogen.

2. Modellierung von Interaktoren

Die Verwendung von Interaktoren zum Entwurf oder der Analyse von interaktiven Systemen ist ein gängiger und ausgereifter Ansatz und wurde umfassend diskutiert (Markopoulos, 1997). Interaktoren können, ähnlich wie Objekte in der Objekt-orientierten Programmierung, als eine abstrakte Architekturabstraktion verstanden werden. In früheren Arbeiten wurden zahlreiche Definitionen erarbeitet. So beispielsweise die PISA (Paterno, 1994) oder die York (Duke et al., 1994) Interaktoren. Unsere Arbeit basiert auf der Definition der York Interaktoren, welche einen Interaktor

„als eine Komponente in der Beschreibung eines interaktiven Systems, welche eine Zustand, die Ereignisse die diesen Zustand manipulieren können, sowie einen Mechanismus beinhaltet, der den Zustand wahrnehmbar für den Nutzer des Systems macht."

(übersetzt aus Duke und Harrison, 1993).

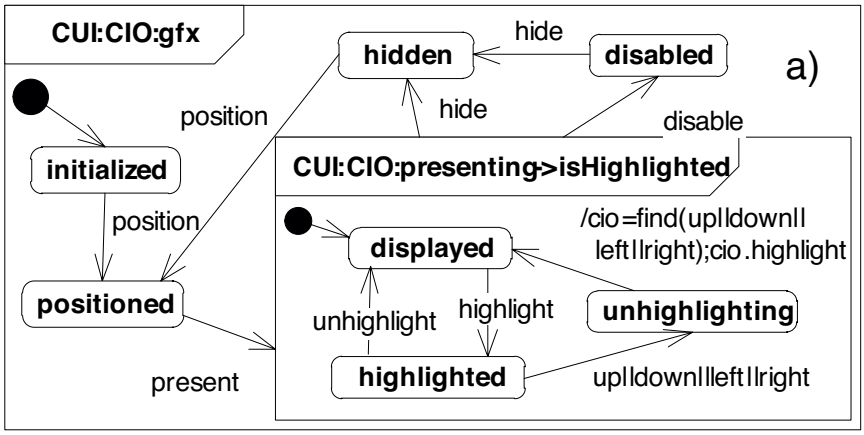
Im Unterschied zu früheren Arbeiten, die Interaktoren zur Beschreibung von grafischen Schnittstellen verwendet haben, konzentriert sich unsere Arbeit auf den Entwurf von multimodalen Schnittstellen.

Zur Spezifikation dieser Datenstrukturen verwenden wir Klassendiagramme. Jeder Interaktor ist mit einer Klasse assoziiert, die den Namen des Interaktors trägt und von der abstrakten „Interaktor" Klasse abgeleitet ist. Wie in Bild 1 dargestellt, beinhaltet die „Interaktor"-Klasse grundlegende Daten, die eine persistente Speicherung eines Interaktors, sowie seine Instanziierung als Zustandsmaschine erlauben. Bild 1 stellt weiterhin die generellen Beziehungen der drei grundlegenden Interaktortypen untereinander dar: Wir unterscheiden zwischen abstrakten Interaktoren (AIO), konkreten Interaktoren (CIO) sowie Interaktionsressourcen (Interaction Resources). Letztere beschreiben die Eigenschaften von Eingabemodalitäten und Ausgabemedien, wohingegen AIOs eine modalitätsunabhängige Beschreibung des grundlegenden Verhaltens und der hierfür benötigten Daten und CIOs die spezifischen Eigenschaften eines konkreten Mediums (bspw. grafische oder natürlich sprachliche Ausgabe) beschreiben. Beim Start einer Anwendung wird jedes Element der Benutzungsschnittstelle über jeweils eine eigene AIO Interaktorinstanz repräsentiert. Weiterhin wird für jede berücksichtigte Modalität und jedes Medium jeweils ein CIO Interaktor (z.B. ein Button) instanziiert. Jedes, dem Nutzer verfügbares Interaktionsgerät wird ebenfalls durch einen eigenen Interaktor zur Systemlaufzeit dargestellt. Interaktionsgeräte können physische Geräte wie beispielsweise ein Wii-Controller oder eine Maus, aber auch komplexe Kombinationen aus Hard- und Software, wie zum Beispiel eine Gestenerkennung mittels farbigen Handschuhen und einer Webcam, darstellen.

2.1 Konkrete Interaktoren für grafische Schnittstellen

Beim Entwurf einer grafischen Benutzerschnittstelle werden Bildschirm oder Fensterinhalte mit verschiedenen CIO Interaktoren wie beispielsweise Buttons, Menüleisten, Comboboxen oder Listenansichten gefüllt und positioniert. Bild 2a stellt das Zustandsdiagramm des grundlegenden Interaktors für all diese Elemente dar.

Der Lebenszyklus eines CIO Interaktors beinhaltet nach seiner Initialisierung eine Berechnung seiner aktuellen Position in Bezug auf andere, gleichzeitig dargestellte Interaktoren mit dem Ergebnis, dass genaue Koordinaten für jeden Interaktor bekannt sind („positioned"). Danach folgt die eigentliche Darstellung des Interaktors („presenting"), die Deaktivierung („disabled") oder die Ausblendung des Inter-



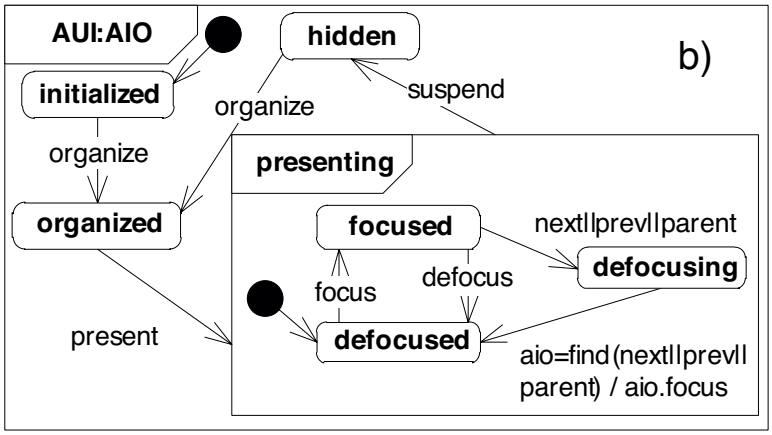


Bild 2: Das Zustandsdiagramm eines konkreten (a) und eines abstrakten (b) Interaktors.

aktors („hidden"). Jeder CIO Interaktor (dessen Verhalten alle abgeleiteten Interaktoren erben) kann außerdem während seiner Darstellung visuell hervorgehoben werden („highlighted") um die Navigation durch die Benutzungsschnittstelle zu erleichtern.

Wie in Bild 2a dargestellt, wird die Navigation über den Empfang von Nachrichten (vor – next, zurück – prev, hoch – parent) gesteuert. Dabei verarbeitet jeder Interaktor autonom die ihn betreffenden Nachrichten: Empfängt beispielsweise der aktuell hervorgehobene Interaktor ein „vor" Befehl, sucht er den folgenden Interaktor aus dem Speicher, deaktiviert die eigene Hervorhebung und sendet an den folgende Interaktor die Nachricht, dass er sich hervorheben soll. Jeder konkrete Interaktor ist an einen abstrakten Interaktor gekoppelt, welcher das grundlegende Verhalten des jeweiligen Elements der Benutzerschnittstelle bestimmt. Bild 2b stellt diesen abstrakten Interaktor dar. Er hat eine ähnliche Struktur, aber eine andere Semantik, da er medienunabhängig ist und somit z.B. auf kein Koordinatensystem zurückgreifen kann. Somit gibt es keine „Positionierung", aber eine Berücksichtigung einer grundlegenden Navigation (vor und zurück), da Sequentialisierung eine Anforderung ist, die alle Medien unterstützen müssen.

2.2 Abstrakte, modalitätsunabhängige Interaktoren

Eine multimodale Anwendung verbindet mindestens ein Ausgabemedium mit mehreren Eingabemodalitäten. Daher sind ein Teil der Interaktoren ausgabebezogen (wie beispielsweise eine Sprachausgabe), während der andere Teil für die Verarbeitung der Nutzereingaben über eine angeschlossenen Modalität zuständig ist (beispielsweise ein Steuerknüppel, eine Maus oder eine Gestensteuerung). Für die abstrakte, modalitätsunabhängige Modellebene führt dies zu zwei Implikationen: 1. die Eingabe wird von der Ausgabe getrennt und 2. eine kontinuierliche Ein- oder Ausgabe wird von einer diskreten Ein- oder Ausgabe unterschieden. In den folgenden zwei Unterabschnitten wird beschrieben, wie sich diese beiden Implikationen in der Gestaltung des abstrakten Interaktorenmodells berücksichtigen lassen: Zum einen durch eine geeignete Strukturierung der Interaktoren in einem Klassendiagramm und zum anderen durch die Spezifikation ihres Verhaltens (der dynamische Aspekt) mittels Zustandsdiagrammen.

Statische Strukturierung der abstrakten Interaktoren

Bild 3 stellt das Klassendiagramm des abstrakten Interaktorenmodells (AUI) und die Vererbungsbeziehungen aller beinhalteten abstrakten Interaktoren dar. Der schon in den vorherigen Abschnitten erwähnte grundlegende AIO Interaktor erbt von der „Interactor" Klasse die Eigenschaft eines eindeutigen Namens sowie die Fähigkeit, einen Gesamtzustand in dem globalen Speicher zu schreiben.

Das AUI Modell unterscheidet generell zwischen eingabebezogenen (AIIN) und ausgabebezogenen (AIOUT) Interaktoren, welchen wiederum beide jeweils eine kontinuierliche oder eine diskrete Verarbeitung unterstützen können. Kontinuierliche ausgabebezogene Interaktoren sind z.B. ein grafischer Fortschrittsbalken, oder ein sich aktualisierendes Diagramm, ein Ton, der sich in der Höhe ändert oder ein Licht, welches gedimmt werden kann. Für den Nutzer ist die aktuelle Information oder der Zustand die von dem Interaktor ausgegeben wird aufgrund Ihrer kontinuierlichen Aktualisierung oder durch seine unscharfen Sinne zwar beschreibbar, aber nicht exakt wiedergebbar. So kann der Nutzer zwar beispielsweise die Lichtstärke eines gedimmten Lichtes in Relation zu anderen (heller, dunkler als…) erfassen, aber nicht exakt in einer diskreten Einheit (Lux) wiedergeben. Eine diskrete Ausgabe lässt sich dahingegen vom Nutzer komplett erfassen und auch wiedergeben (z.B. ein gesprochener oder geschriebener Text).

Die Gruppierung von Interaktoren (AIC) ist ebenfalls diskret sowie ausgabeorientiert und kann bei Bedarf in eine Einfach – (AISingleChoice) oder Mehrfachauswahl (AIMultiChoice) spezialisiert werden. Zu jedem Interaktor kann

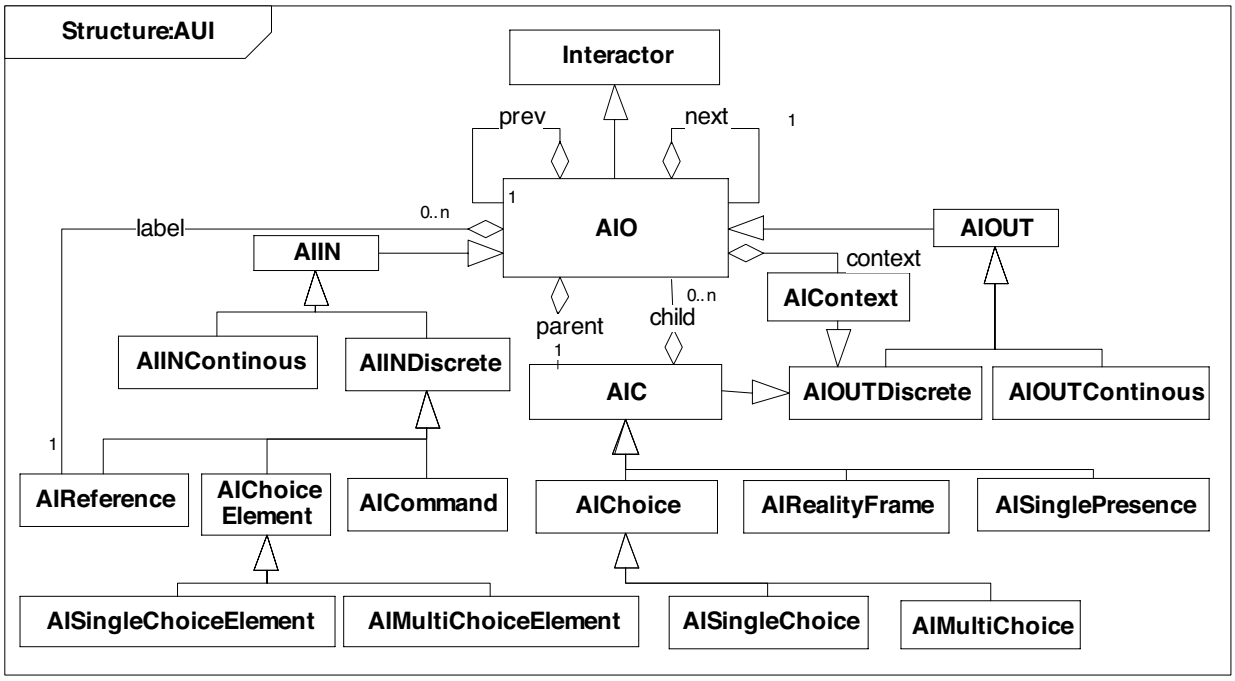


Bild 3: Statisches, abstraktes Interaktorenmodell.

ein Kontext (AIContext) definiert werden. Dieser ist ebenfalls eine diskrete Ausgabe, wie beispielsweise ein Bild, ein Text in einem Hilfsfenster (Tooltip) oder ein gesprochener Text. Ein Kontext macht einen Interaktor besser für den Nutzer verständlich und hilft bei dessen Bedienung.

Auch Nutzereingaben die von eingabebezogenen Interaktoren verarbeitet werden, können kontinuierlich (AIINContinous) oder diskret (AIINDiscrete) vorgenommen werden. Beispiele für eine kontinuierliche Eingabe sind z.B. ein Scrollbalken oder das Darstellen einer Längenangabe durch den Abstand zwischen zwei Händen. Diskrete Eingaben sind beispielsweise Sprachkommandos, das Betätigen eines realen oder virtuellen (mittels Gestik) Knopfes. Die Auswahl eines Elements (AISingleChoiceElement) oder mehrerer Elemente (AIMultichoiceElement) aus einer Liste wird ebenfalls als eine eingabezogen Interaktion verstanden.

Die grundsätzliche Trennung zwischen eingabe- und ausgabebezogenen Interaktoren ermöglicht die Verteilung von Interaktoren auf unterschiedlichen Modalitäten und Medien, hat aber auch Implikationen auf die statische Interaktorenstruktur. So führt sie beispielsweise zu einer Trennung von Listen in eine ausgabebezogene Interaktion (die Darstellung der Liste als solche) und in eine eingabebezogene Interaktion (die Möglichkeit der Selektion eines oder mehrerer Elemente, die über das Element einer Liste durchgeführt wird).

Zu guter Letzt können zu jedem abstrakten Interaktor Referenzen angegeben werden, die z.B. als Sprungmarken bei der Navigation zwischen Elementen, als ein Tastaturkürzel, eine kategorisierende Farbe oder auch als ein Bereich definiert werden können, auf den der Nutzer zeigen kann, um ein Element eindeutig zu referenzieren.

Verhaltensspezifikation von abstrakten Interaktoren

Auf Basis der im letzten Unterabschnitt entworfenen statischen Strukturierung der Interaktoren, wird jeder Interaktor mit einer Beschreibung seines Verhaltens mittels eines Zustandsdiagrammes vervollständigt, wobei ein Interaktor grundsätzlich das Verhalten an die von ihm abgeleiteten Kinder weitergibt, die dieses dann ergänzen.

Das Verhalten des grundlegenden abstrakten Interaktors (AIO) wurde bereits in Bild 2b dargestellt und in Abschnitt 2.1 erläutert. Aus Platzgründen beschreiben wir im Folgenden beispielhaft nur zwei weitere Interaktoren, die für die Auswahl von Elementen in einer Liste zuständig sind. Die Spezifikation aller weiteren Interaktoren kann online auf der Projektseite abgerufen werden.

Bild 4a stellt das Zustandsdiagramm eines Elements (AISingleChoiceElement) einer Einfachauswahlliste (AISingleChoice) dar. Der Interaktor erbt sein grundlegendes Verhalten vom AIO Interaktor und ergänzt dieses durch eine verfeinerte Spezifikation des „presenting" Zustands. Dazu wird der ursprüngliche „presenting" Zustand durch zwei parallel laufende komplexe Zustände ersetzt: Einer realisiert die ursprüngliche AIO Funktionalität die eine Fokussierung des Interaktors ermöglicht, wohingegen der andere komplexe Zustand das neue Verhalten des Interaktors beschreibt: er kann auch ausgewählt, aufgegriffen (drag) und losgelassen (drop) werden. In einer Einfachauswahl kann immer nur ein Element ausgewählt sein. Dieses Verhalten ist in Bild 4a durch den „choosing" Zustand sichergestellt, der bedingt, dass zuvor ein Element ausgewählt wurde und alle anderen Elemente der Liste de-selektiert werden. Durch die „History" Funktion „H" wird weiterhin sichergestellt, dass die Auswahl persistent bleibt, auch wenn der Interaktor ausgeblendet oder deaktiviert ist.

Bild 4b stellt den Listeninteraktor einer Einfachauswahl dar, der Listenelemente verwaltet. Auch hier wird auf die gleiche

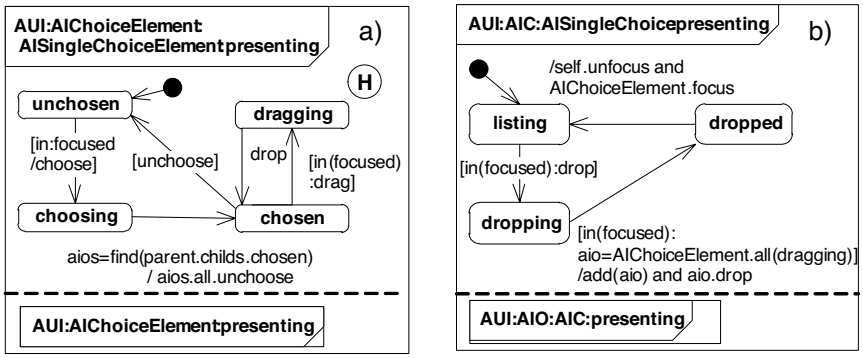


Bild 4: Zustandsdiagramme eines Listenelement-Interaktors (a) und eine Einfachauswahlliste (b).

Weise ein parallel laufender komplexer Zustand eingeführt, wie er schon bei den Listenelementen ergänzt wurde. Im Unterschied zu einem Listenelement, welches sich bei einem Drag und Drop selbständig aus seiner Liste entfernen kann, muss das Einfügen in eine neue Liste über die Liste selbst geprüft werden (nicht jede Liste nimmt jedes Element). Daher wird im Listeninteraktor lediglich das Verhalten bei einem fallenlassen (drop) eines Elementes ergänzt. Dazu wird beschrieben, dass die Liste selbst im „fokus" stehen muss (z.B. indem der Mauszeiger auf sie zeigt) und dann mittels des „dropping" Zustands alle Elemente die gegenwärtig aufgegriffen sind, in die Liste aufgenommen und über die erfolgreiche Aufnahme informiert werden (sie erhalten eine „drop" Nachricht).

3. Modellsynchronisierung und Multimodale Mappings

Um die zahlreichen Interaktoren aus dem abstrakten, konkreten und Gerätemodellen zur Entwurfszeit zu verbinden und zur Laufzeit zu synchronisieren, werden sogenannte Mappings verwendet. Sie basieren auf Eigenschaften der Zustandsdiagramme, die Nachrichten empfangen und daraufhin Zustandswechsel durchführen. Jedes Mapping kann zur Laufzeit diese Zustandswechsel beobachten und Nachrichten an die Zustandsmaschinen schicken um Zustandswechsel auszulösen.

Es wird zwischen zwei Arten von Mappings unterschieden: Synchronisationsmappings und multimodale Mappings. Erstere synchronisieren zur Systemlaufzeit die Zustandsmaschinen der instanziierten Interaktoren auf den unterschiedlichen Modellebenen (insbesondere zwischen abstrakten und konkreten Interaktoren). Synchronisationsmappings werden gemeinsam mit den abstrakten und konkreten Interaktoren definiert und sind daher beim Design einer multimodalen Anwendung vorgegeben und werden automatisch berücksichtigt, sobald der jeweilige Interaktor für die Gestaltung der Benutzerschnittstelle verwendet wird. Mittels Synchronisationsmappings wird z.B. sichergestellt, dass ein hervorgehobener CIO Interaktor auf abstrakter Ebene als fokussiert interpretiert wird (vergl. Bild 2).

Multimodale Mappings werden zum Anschluss der Geräte und Modalitätsinteraktoren an die abstrakten oder konkreten Interaktoren verwendet. Sie können ebenfalls schon vordefiniert beim Anwendungsdesign vorliegen, wenn eine bestimmte Interaktionsform für einem Gerät vorgeben ist oder ein bestimmter Interaktionsparadigma (wie z.B. ein Drag-and-Drop) verwendet werden soll. Aber der weitaus häufigere Anwendungsfall ist die Definition eines anwendungsspezifischen multimodalen Interaktionsschrittes, wie beispielsweise der Anforderung einer über mehrere Modalitäten redundanten Bestätigung eines Kommandos für sicherheitskritische Anwendungen (z.B. mittels Sprache und Mausklick).

Die Notation der multimodalen Mappings unterscheidet zwischen Rechtecken mit scharfen oder runden Kanten. Runde Kanten bezeichnen die Beobachtung von Zustandswechseln, wohingegen Rechtecke mit scharfen Ecken Methoden aus dem funktionalen Anwendungsbackend aufrufen oder Nachrichten an die Zustandsmaschinen der Interaktoren senden.

Bild 5 zeigt, wie ein Interaktionsparadigma, das Drag-and-Drop, auf abstrakter Ebene mit einem multimodale Mapping modelliert werden kann. Es spezifiziert die Drag-and-Drop Eigenschaft für Elemente (AIChoiceElement) einer Liste (AIChoice), welche im vorherigen Abschnitt eingeführt worden sind. Dieses Mapping wird gestartet sobald (1.) eine Maustaste gedrückt, (2.) eine Liste im Fokus des Nutzers steht (sie ist dann hervorgehoben, was impliziert das der Mauszeiger über der Liste steht und angehalten wurde) und (3.) mindestens ein Element der Liste ausgewählt mit dem Tastendruck oder schon zuvor ausgewählt worden ist.

Im folgenden Abschnitt verwenden wir nun die in diesem Abschnitt vorgestellten Interaktoren und Mappings, um eine gestenbasierte Navigation zu modellieren und erläutern dann anhand einer Fallstudie die prototypische Umsetzung mehrerer Interaktionsalternativen zur Navigation.

4. Modellierung von gestenbasierter Interaktion

Gestik als eine Form der der natürlichen Interaktion wird immer häufiger auch als ein Weg der Kommunikation zwischen Mensch und Computer genutzt (Bernandes Jr. Et al., 2009). Aber die wenigsten Gesten sind interkulturell oder auch anwendungsübergreifend akzeptiert. Nutzertests und insbesondere Vergleiche zwischen verschiedenen Alternativen sind daher häufig unumgänglich. Faktoren, die beim Design einer auf Gestik basierten Anwendungssteuerung eine Rolle spielen sind beispielsweise: Die Zuordnung einzelner Gesten zu konkreten Kommandos auf der Anwendungsebene,

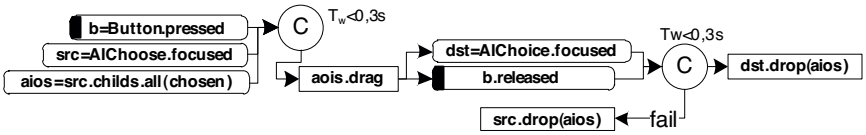
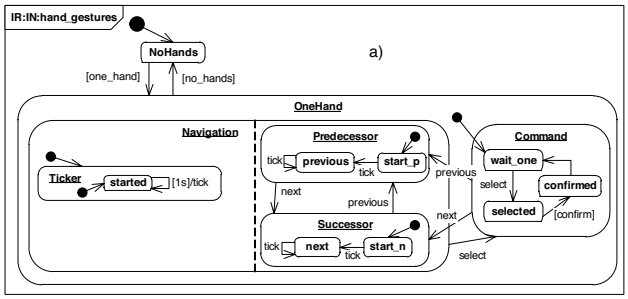


Bild 5: Grundlegendes Drag-and-Drop Mapping auf abstrakter Modellebene.



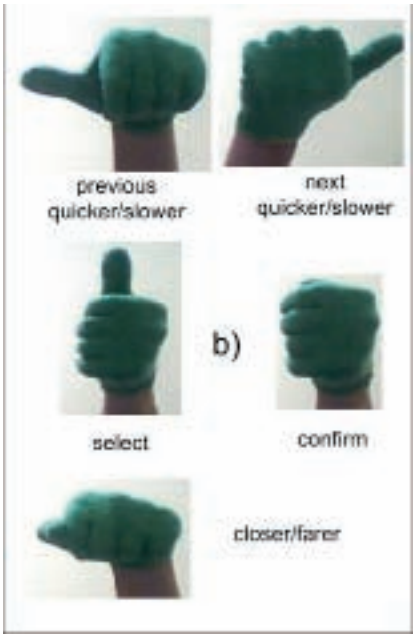


Bild 6: Die erste Variante einer auf Gestik basierenden Navigationssteuerung (a) und die fünf grundlegenden Gesten für die Navigationssteuerung (b).

die Entscheidung, ob ein oder beide Hände zur Steuerung berücksichtigt werden sollen, ergonomische Kriterien, wie auch die Intuitivität der gewählten Gesten oder deren leichte Erlernbarkeit. In unterschiedlichen Anwendungsfällen haben wir bereits auf Gesten basierte Anwendungssteuerungen umgesetzt (Feuerstack et al., 2011).

Die Umsetzung des hier beschriebenen Interaktors basiert auf einem System zur Erkennung von Gebärdensprache (Pizzolato et al. 2010).

Bild 6a stellt das Zustandsdiagramm für die erste Variante einer auf Gestik basierenden Navigation dar. Die Navigation beschränkt sich auf die grundlegende Möglichkeit, mittels zwei Handzeichen das jeweils vorherige bzw. nächste Element zu erreichen. Dabei werden die beiden in der oberen Zeile in Bild 6b) dargestellten Handzeichen verwendet. Es wird weiterhin ein fester Taktgeber verwendet, der während das Handzeichen vom Nutzer gezeigt wird, pro Sekunde einen Navigationssprung auf das nächste oder vorherige Benutzungsschnittstellenelement ausführt. In dem Zustandsdiagramm des Interaktors wird dieses Verhalten über die beiden „Predecessor" und „Successor" Zustände ausgedrückt, die in Abhängigkeit zu dem parallel laufenden Taktgeber die Vorwärts- und Rückwärtsnavigation ausführen. Weiterhin wird durch das Zustandsdiagramm ein Kommando definiert, welches zur Auswahl eines Elementes genutzt wird. Sobald die „select" Geste (vgl. Bild 6b) gezeigt wird, hält daher der Taktgeber wie auch die Navigation an und der „Command" Zustand wartet auf eine Bestätigung des Nutzers durch ein „Confirm" Handzeichen.

Die zwei Zustandsdiagramme von Bild 7 stellen zwei alternative Navigationsformen dar. Aus Platzgründen werden lediglich die Änderungen in Bezug auf die erste Variante aus Abbildung 6a dargestellt. Daher bleiben die Zustände „Predecessor", „Successor" und „Command" gleich. Bild 7a stellt eine Variante der Navigationssteuerung mit einem veränderbaren Taktgeber dar. Dazu kann die Taktzeit t in drei Stufen (1200ms, 1000ms und 800ms) beeinflusst werden, indem der Nutzer die Hand näher zu der am Bildschirm montierten Kamera (schnellerer Takt) bzw. weiter von der Kamera wegbewegt (langsamerer Takt). In der dritten Variante (Bild 7b) wird die Navigation zwischen den Elementen direkt über die Handbewegung zum Bildschirm gesteuert. Unabhängig von einer Vorwärts- oder Rückwärtsbewegung wird jede 15 cm ein Navigationsschritt ausgelöst. In den zusätzlichen Taktgeber, der im

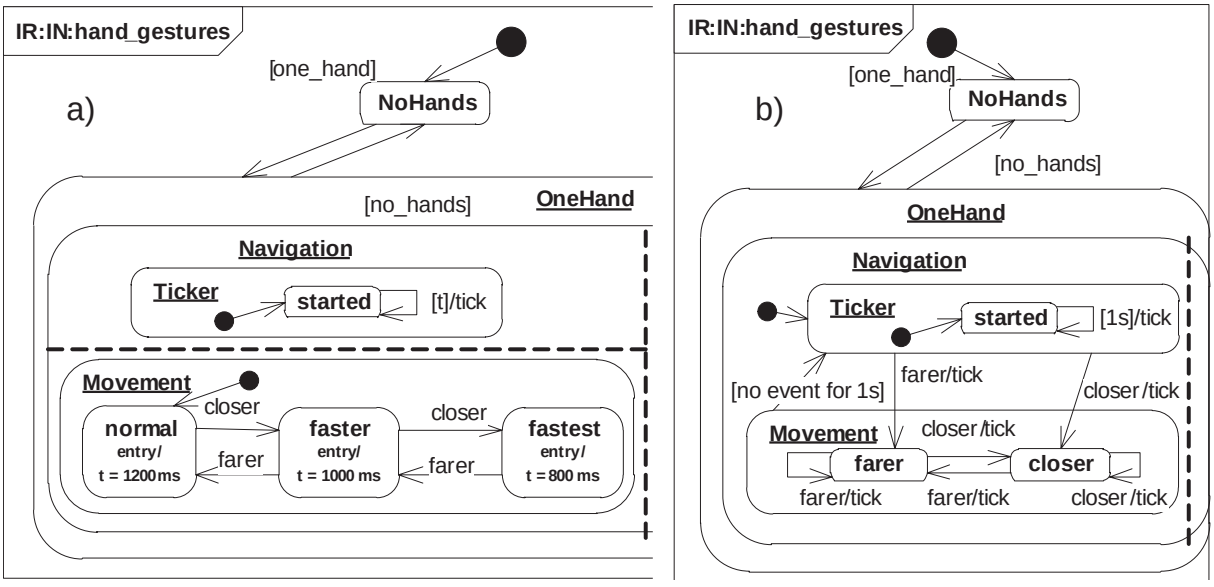


Bild 7 Zweite Variante: Zur Erhöhung der Navigationsgeschwindigkeit bewegt der Nutzer seine Hand näher zur Kamera (a). sowie die dritte Variante: Anstelle eines Taktgebers wird durch jede Handbewegung in Richtung Kamera (oder zurück) ein Navigationsschritt ausgelöst (b).

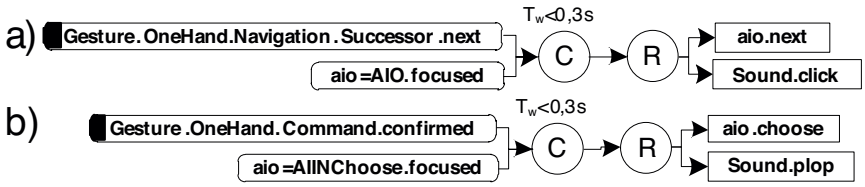


Bild 8: Zwei Mappings, die die Anbindung eines Gesteninteraktors an eine Benutzungsschnittstellen beschreiben.

Gegensatz zur Variante 2 parallel zur Navigationskontrolle modelliert wurde, wird nur gesprungen, sobald der Nutzer in einer Position für mindestens eine Sekunde verharrt, während er aber weiterhin eine Handpose (z.B. „next") zeigt.

Um nun mit diesen drei Gesteninteraktoren eine konkrete Benutzungsschnittstelle zu steuern, sind Mappings notwendig, die festlegen, wie die Schnittstelle auf eine Geste reagieren soll.

Bild 8 stellt zwei Mappings dar, die die Vorwärtsnavigation (Bild 8a) sowie die Auswahl eines Elementes (Bild 9b), beschreiben. Für die Vorwärtsnavigation wartet das Mapping aus Bild 8a auf die entsprechende Geste und den damit verbundenen Zustandswechsel des Gesteninteraktors in den Zustand „next". Über die komplementäre Relation „C" wird im Mapping definiert, dass es zu dem Zeitpunkt zu dem die Geste gezeigt wird (t<0.3s) ein fokussiertes Element geben muss, damit das Mapping auslöst und einen Navigationssprung zum folgenden Element einleitet („aio.next"). Durch die Relation „R", die eine Redundanz ausdrückt, wird weiterhin definiert, dass redundant zu dem Navigationssprung ein „click" Geräusch ausgegeben werden soll. Das zweite Mapping (Bild 9b) definiert nach dem gleichen Prinzip eine Elementauswahl des aktuell fokussierten Elementes, die ebenfalls mit einem Ton „plop" bestätigt wird.

5. Verwandte Arbeiten

Zahlreiche Arbeiten setzen sich mit der Vereinfachung der Erstellung multimodaler Anwendungen auseinander. Aktuelle Frameworks wie zum Beispiel Squidy (Kaenig et al., 2010) oder das Open Interface Framework (Lawson et al., 2009) vereinfachen das Erstellen von multimodalen Anwendungen indem sie mit Editoren die Zusammenstellung einer multimodalen Anwendung aus vordefinierte Komponenten ermöglichen. Algorithmen zur Signalverarbeitung und Gerätetreiber sind typische Komponenten aus denen mittels grafischer Programmierung ein multimodales System geformt wird. Die Vereinfachung durch grafische Programmierung erfordert die Abkapselung der Komponentenfunktionalität. Zugriffe auf die Komponenten (eingehende Informationen, Ausgabewerte, wie auch Einstellungen) werden über Attribute konfiguriert über die die Komponenten auch verknüpft werden. Der Vorteil der abgekapselten Komponenten liegt in ihrer einfach verständlichen Verwendung, geht aber verloren, sobald eine vorhandene Komponente erweitert werden muss. In diesem Fall kann die Komponente nur auf Quelltextebene analysiert und erweitert werden.

Die komplexe Erweiterbarkeit ist insbesondere dann nachteilig, wenn neuartige Interaktionsformen mit dem Ziel prototypisch implementiert werden, sie miteinander vergleichen zu können. Gerade für multimodale Interaktion, von der durch die Kombination mehrerer Modalitäten und Medien einen Mehrwert gegenüber der Bedienung mittels nur einer Modalität erwartet wird, bieten sich ein Test z.B. durch die Evaluation und den Vergleich verschiedener Prototypen an, um den Mehrwert für den Nutzer z.B. hinsichtlich Robustheit und Geschwindigkeit zu ermitteln. Aktuelle Arbeiten im Bereich der Modalitätentheorie (Bernsen, 2008) entwickeln umfangreiche Klassifizierungen hinsichtlich der Aufschlüsslung der Eigenschaften einzelner Modalitäten, welche eine Basis für eine systematische Analyse des Mehrwerts verschiedener Modalitätenkombinationen verwendet werden können.

Die modelgetriebene Entwicklung von Benutzungsschnittstellen wird schon seit einigen Jahrzehnten betrieben und führte zu zahlreichen, miteinander verbundenen Modellen, die beispielsweise in dem CAMELEON Framework (Calvary et al. 2003) zusammengefasst wurden oder mit Ansätzen wie z.B. USIXML (Limbourg et al., 2004) in Richtung Standardisierung vorangetrieben werden. Es gibt bislang nach unserer Kenntnis nur wenige Ansätze, die die modellgetriebene Entwicklung zur Generierung von multimodalen Benutzungsschnittstellen verwenden, wie beispielsweise (Stanciulescu et al., 2005), mit deren Ansatz sich mit USIXML XHTML+Voice Schnittstellen generieren lassen. Vielmehr werden bei modelgetriebenen Ansätzen, die dem CAMELEON Framework folgen, mittels eines strukturierten Prozesses ausgehend von abstrakten, plattform- und modalitätsunabhängigen Modellen, mehrere plattformspezifische, ausführbare Benutzungsschnittstellen generiert. Diese sind zwar untereinander konsistent, aber nicht mehr miteinander verbunden, wie es in multimodalen und plattformübergreifenden Interaktionsanwendungen erforderlich ist. Die modellgetriebenen Entwicklungsprozesse sind weiterhin für vordefinierte Zielplattformen vorgegeben und beinhalten für diese vordefinierte Modelltransformationsschritte, die semi- oder vollautomatisch ausgehend von abstrakten Modellen, plattformspezifische Benutzungsschnittstellen generieren können.

Während modellgetriebene Ansätze durch ihre vorgegebene Strukturierung und die definierte Abfolge von Prozessschritten die Komplexität bei der Erstellung von Multi-Plattform Anwendungen handhabbar machen, ist es fraglich, ob sie auch für das Prototyping von multimodalen Schnittstellen geeignet sind. Erfordern sie doch häufig den Entwickler eine neue Sprache (wie beispielsweise USIXML) zu erlernen, sowie die Transformationen zwischen den Modellen antizipieren zu können.

Unser Ansatz lässt sich dagegen nur zum Prototyping für den Vergleich von verschiedenen Interaktionen, mangels eines definierten Entwicklungsprozesses aber nicht für die Erstellung komplexerer Anwendungen nutzen. Dadurch, dass der Großteil des Entwurfs auf Zustandsdiagrammen basiert, die in Standards wie UML oder SCXML schon weit verbreitet und damit vielen Entwicklern bekannt sind, sowie auch schon zahlreiche Werkzeuge zu deren Modellierung existieren, sind die Einstiegsbarrieren zur Verwendung unseres Ansatzes eher gering.

6. Conclusio

Mit unserem Ansatz lassen sich Interaktionen basierend auf Zustandsdiagrammen und multimodalen Mappings beschreiben. Nicht ein Generierungsprozess, sondern ein Mittel zur detaillierten Beschreibung einzelner multimodaler Interaktionsszenarien steht im Vordergrund. Dieser Beitrag zeigt, wie sich mit Interaktoren auf Basis von Zustandsdiagrammen, die auch direkt als Automaten ausgeführt werden können, einfach verschiedene Interaktionsvarianten prototypisch entwickeln lassen, um sie zum Beispiel miteinander vergleichen zu können.

Als ein Fallbeispiel beschreiben wir, wie mit unserem Ansatz verschiedene Varianten einer Gestensteuerung zur Navigation entworfen und prototypisch umgesetzt wurden. Dabei konnten die Unterschiede der Varianten direkt in den Zustandsdiagrammen beschrieben werden, ohne dass noch eine umfangreichere Programmierarbeit notwendig wurde. Mit multimodalen Mappings, einer im Gegensatz zu den auf Harel basierten Zustandsdiagrammen noch proprietärer Notation lassen sich einfach weitere Modalitäten und Medien zu einem Prototyp hinzufügen sowie auch Paradigmen, wie hier am Beispiel des Drag-and-Drop gezeigt, modellieren.

Danksagung

Der vorgestellte Ansatz ist Teil des Post-Doc Projektes von Sebastian Feuerstack an der Universidade Federal de São Carlos, São Paulo in Brasilien, welches vom DFG als Forschungsstipendium finanziert wird. Wir danken Jessica Colnago für ihre Mitarbeit an einzelnen Teilen des hier präsentierten Ansatzes.

Literatur

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.

Bernardes Jr, J.-L.; Nakamura R.; Tori, R.: Design and implementation of a flexible hand gesture command interface for games based on computer vision. In Proceedings of the 2009 VIII Brazilian Symposium on Games and Digital Entertainment, SBGAMES '09, pages 64–73, Washington, DC, USA, 2009.

Bernsen, N. O. Multimodality Theory, in Dimitrios Tzovaras, ed., Multimodal User Interfaces', Springer Berlin Heidelberg, pp. 5–29, 2008.

Calvary, G.; Coutaz, J.; Thevenin,D.; Limbourg, Q.; Bouillon, L.; Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interacting with Computers, 15(3):289–308, 2003.

Duke, D.J.; Faconti, G.; Harrison, M.; Paternó, F.: Unifying views of interactors. In AVI '94: Proceedings of the Workshop on Advanced Visual Interfaces, pages 143–152, New York, NY, USA, 1994. ACM, ISBN: 0-89791-733-2.

Duke, D.J.; Harrison, M.D.: Abstract interaction objects. Computer Graphics Forum, 12(3):25–36, 1993.

Feuerstack, S.; Pizzolato, E.B.: Building Multimodal Interfaces out of Executable, Model-based Interactors and Mappings; HCI International 2011; 14th International Conference on Human-Computer Interaction; J.A. Jacko (Ed.): Human-Computer Interaction, Part I, HCII 2011, LNCS 6761, pp. 221–228. Springer, Heidelberg (2011), 9-14 July 2011, Orlando, Florida, USA.

Feuerstack, S.; Oliveira, A. & Araujo, R. (2011), Model-based Design of Interactions that can bridge Realities – The Augmented Drag-and-Drop, in Proceedings of the 13th Symposium on Virtual and Augmented Reality (SVR 2011).

Fikkert, F.: Gesture Interaction at a Distance. PhD thesis, Universiteit Twente, Centre for Telematics and Information Technology, 2010.

Kaenig, W. A.; Raedle, R. ; Reiterer, H.: Interactive design of multimodal user interfaces – reducing technical and visual complexity. Journal on Multimodal User Interfaces, 3(3):197–213, Feb 2010.

Lawson, J.-Y. L.; Al-Akkad, A.-A.; Vanderdonckt, J.; Macq, B.:An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components. In EICS '09: Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems, pages 245–254, New York, NY, USA, 2009. ACM.

Limbourg, Q.; Vanderdonckt, J.; Michotte, B.; Bouillon, L.; Lopez-Jaquero. V.: USIXML: A language supporting multi-path development of user interfaces. In Remi Bastide, Philippe A. Palanque, and Joerg Roth, editors, EHCI/DS-VIS, volume 3425 of Lecture Notes in Computer Science, pages 200–220. Springer, 2004.

Markopoulos, P.: A compositional model for the formal specification of user interface software. PhD thesis, Queen Mary and Westfield College, University of London, 1997.

Paterno, F.: A theory of user-interaction objects. Journal of Visual Languages & Computing, 5(3):227 – 249, 1994.

Pizzolato, E. B. ; Santos Anjo, M.; Pedroso, G. C.: Automatic recognition of finger spelling for libras based on a two-layer architecture. In Proceedings of the 2010 ACM Symposium on Applied Computing, pages 969–973, 2010.

Stanciulescu, A.; Limbourg, Q.; Vanderdonckt, J.; Michotte, B.; Montero, F.: A transformational approach for multimodal web user interfaces based on USIXML. In ICMI '05: Proceedings of the 7th International Conference on Multimodal Interfaces, pages 259–266, New York, NY, USA, 2005. ACM Press.

1 Sebastian Feuerstack beschäftigt sich gegenwärtig an der Universidade Federal de São Carlos (UFSCar) in São Paulo im Rahmen eines DFG Forschungsstipendiums mit der modellbasierten Beschreibung von multimodalen Interaktionen. Seine Doktorarbeit an der TU-Berlin behandelte die modellgetriebene Entwicklung von Multi-Plattform Benutzungsschnittstellen und war Bestandteil des vom BMWi geförderten Service-centric Home Projektes.
E-Mail:Sebastian@Feuerstack.de

2 Mauro dos Santos Anjo ist Student im Masterstudiengang für die digitale Bild- und Signalverarbeitung im Fachbereich Informatik an der Universidade Federal de São Carlos, São Paulo in Brasilien. Seine Forschungsinteressen liegen im Bereich der Bildverarbeitung und Mustererkennung sowie der computergestützten Erkennung von Gebärdensprache. Er arbeitet gegenwärtig als Softwareentwickler bei Ablevision, eine Firma die sich mit Computer Vision Technologie im industriellen Umfeld auseinandersetzt.
E-Mail: mauro_anjo@dc.ufscar.br

3 Ednaldo Brigante Pizzolato ist seit 1999 Professor an der Universidade Federal de São Carlos, São Paulo in Brasilien. Seine Doktorarbeit über Spracherkennung mit Hilfe von Multinet-Systemen verfasste er 1999 an der University of Essex in England. Aktuell forscht er in den Bereichen der Spracherkennung sowie den modularen neuronalen Netzen.
E-Mail: ednaldo@dc.ufscar.br