Jürgen Falb, Hermann Kaindl, Roman Popp und David Raneburger

Automatiserte Generierung von WIMP-UIs basierend auf Kommunikationsmodellen

Automated WIMP-UI Generation Based on Communication Models

WIMP-UI Generierung, Kommunikationsmodell

Zusammenfassung. Manuelles Erstellen von User Interfaces (UIs) ist teuer und zeitaufwändig, insbesondere für UIs, die auf mehreren Endgeräten mit verschiedenen Eigenschaften (z.B. Größe bzw. Auflösung des Bildschirms) ablaufen sollen. Etwa sind Webseiten, die für PCs konzipiert wurden, oft nur schlecht auf einem Smartphone zu bedienen. Daher wird es im Allgemeinen für gute Bedienbarkeit nötig sein, verschiedene UIs für verschiedene Geräte zu erstellen.
Eine Möglichkeit zur Verbesserung ist die automatisierte Generierung von UIs aus Modellen mit hoher Abstraktion. Dies sind meist Task-basierte Modelle. Wir definieren hingegen Kommunikationsmodelle zur Spezifikation kommunikativer Interaktion durch Diskurse im Sinne von (Klassen von) Dialogen. Ähnlich wie bei einigen Task-basierten Ansätzen verwenden auch wir automatisierte Modell-Transformationen in immer konkretere Modelle und letztlich Code für die UIs. Unser Ansatz zeichnet sich allerdings durch Optimierungen aus, die automatisch für gegebene Geräte-Spezifikationen erfolgen. Konkret ergeben sich dadurch sogar spezielle Anpassungen für verschiedene Smartphones (iPhone vs. Android-basierte Geräte).

Summary. Manual creation of user interfaces (UIs) is expensive and time-consuming, especially for UIs supposed to run on several devices with different properties (e.g., size and resolution of their screens). For instance, Web pages designed for PCs are often only poorly readable on a Smartphone. So, it will be generally necessary for good usability to create different UIs for different devices.
One possibility for improvement is the automated generation of UIs from models on a high level of abstraction. These are usually task-based models. In contrast, we define communication models for the specification of communicative interaction based on discourses in the sense of (classes of) dialogues. Analogously to some of the task-based approaches, we also use automated model transformations to more and more concrete models and finally code for the UIs. Our approach, however, is unique through optimizations, which automatically apply given device specifications. In fact, even specific adaptations result for different Smartphones (iPhone vs. Android-based devices).

1. Einleitung

In der wissenschaftlichen Literatur wurde automatisierte Generierung von User Interfaces (UIs) schon vielfach behandelt. Die folgende kleine Auswahl soll hier nur einen Überblick über andere relevante Ansätze vermitteln. Eine Vielzahl dieser Ansätze verwendet Task-Modelle als Ausgangspunkt. Task-Modelle modellieren die Tasks (Aufgaben), welche ein Benutzer erfüllen muss um an ein bestimmtes Ziel zu kommen, d.h. einen übergeordneten Task zu erfüllen. Da unterschiedliche Task-Modelle und die dazugehörenden Generierungsansätze unterschiedliche Schwerpunkte haben, ist es wichtig im Vorfeld der zu lösenden Aufgabe das richtige Modell zu wählen. Limbourg und Vanderdonckt geben einen guten Überblick über etablierte Task-Modelle (Limbourg und Vanderdonckt 2003). Meixner und Seissler haben eine Taxonomie entwickelt, um den Vergleich von Task-Modellen zu erleichtern (Meixner und Seissler 2011). Die hier erwähnten Ansätze verwenden Modelle mit ConcurTaskTrees (CTT) (Paterno et al. 1997) auf ihrem höchsten Abstraktionslevel. Ausgehend von solchen CTT-Modellen unterstützt Paternos MARIA Environment die Entwicklung von UIs für verschiedene Endgeräte und Modalitäten (Paternò et al. 2009). UI-Generierung basierend auf CTT-Modellen wird auch von verschiedenen Tools des UsiXML-Frameworks durchgeführt (Vanderdonckt 2008). Beide Ansätze unterstützen den Designer auf allen Ebenen des Cameleon Reference Frameworks (Calvary et al. 2003). Im Gegensatz zu diesen beiden eher allgemein gehaltenen Ansätzen, stellt die OO-Methode von Pastor einen speziell für Informationssystem-UIs entwickelten Ansatz dar (Pastor et al. 2008).

DOI 10.1524/icom.2011.0032

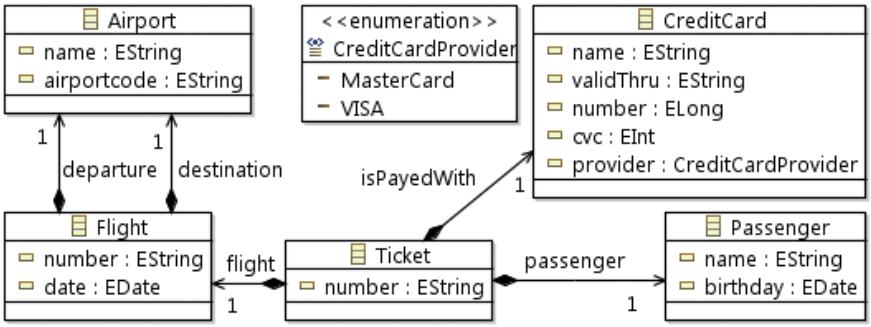

Bild 1: Domain-of-Discourse Model für Flugbuchung.

Die OO-Methode transformiert Task-Modelle nicht direkt, sondern verwendet sie um „Interaction Requirements" zu spezifizieren. Diese Requirements werden dann im „Conceptional Model" abgebildet, welches direkt zu Quellcode transformiert werden kann.

Im Gegensatz zu diesen meist Task-basierten Modellierungsansätzen haben wir in (Falb et al. 2006) Modelle von Diskursen zur Modellierung vorgeschlagen. Dies sind Diskurse im Sinne von (Klassen von) Dialogen zwischen zwei Dialogpartnern. Bei der Spezifikation von kommunikativer Interaktion mit Software (oder auch einem System mit spezieller Hardware wie etwa einem Roboter) ist ein Dialogpartner eben dieses System und der andere ein menschlicher Benutzer. Grundsätzlich können wir mit diesem Ansatz aber auch Diskurse zwischen solchen Systemen modellieren und sogar das systematische Zusammenspiel solcher verschiedener Diskurse nutzen (Kaindl et al. 2011).

Der Weg von solchen Modellen zum Code von User Interfaces (genauer genommen von Windows/Icons/Menues/Pointers – WIMP UIs) führt über Modell-Transformationen, wie wir sie etwa in (Kavaldjian et al. 2008) und (Raneburger et al. 2011a) für unseren Ansatz genauer beschrieben haben. Diese Transformationen erfordern die Spezifikation von Metamodellen und von Transformationsregeln, welche Modelle entsprechend dieser Metamodelle in einander überführen. Zur Automatisierung war auch die Erstellung eines Transformations-Werkzeugs nötig, wobei sich unseres dadurch auszeichnet, dass es das „Feuern" von mehreren Regeln für gleiche Modell-Teile erlaubt. Dies ist wiederum auch eine wichtige Voraussetzung für die Optimierung eines UIs für ein spezielles Endgerät, welche unseren Ansatz ebenfalls auszeichnet, siehe (Raneburger et al. 2011b).

Für die folgenden genaueren Erklärungen bedienen wir uns eines stark vereinfachten Beispiels für das Suchen und Buchen von Flügen mittels Software. Wir präsentieren insbesondere unserem Ansatz entsprechende konkrete Modelle dafür. Ebenfalls zeigen wir Screenshots des automatisch erzeugten UIs für iPhones, das gemäß einer einfachen Geräte-Spezifikation (genauer der Bildschirmauflösung) aus genau diesen Modellen speziell dafür optimiert wurde.

Der Rest dieses Beitrags ist wie folgt strukturiert: Zuerst präsentieren wir kurz die von uns definierten Diskurs-basierten Kommunikationsmodelle. Danach erklären wir die Grundzüge der automatisierten Generierung von WIMP-UIs für verschiedene Endgeräte, die wir konzipiert und implementiert haben. Zuletzt beschreiben wir die Anbindung der generierten UIs an die Applikationslogik des interaktiven Programms, für das sie Verwendung finden sollen.

2. Diskurs-basierte Kommunikationsmodelle

Die Hauptidee des hier präsentierten Ansatzes ist ein Diskurs-basiertes Kommunikationsmodell (Popp und Raneburger 2011). Dieses Kommunikationsmodell besteht aus drei Teilmodellen, dem „Domain-of-Discourse Model", dem „Action-Notification Model" und dem „Discourse Model".

Das Domain-of-Discourse Model spezifiziert diejenigen Objekte der Domäne, über welche Kommunikation stattfinden kann. Zur Beschreibung kann ein UML-Klassendiagramm verwendet werden. Bild 1 zeigt das Domain-of-Discourse Model für unser Beispiel. In der linken unteren Ecke sieht man die Abbildung eines Fluges. Die definierte Abbildung eines Fluges (die Klasse Flight) hat die Flugnummer und das Datum als Attribute und jeweils einen Abflugs- und Ankunftsflughafen. Die Abbildung eines Flughafens (die Klasse Airport) hat wiederum einen Namen und den Flughafencode als Attribute. Bild 1 zeigt noch Abbildungen für Tickets, Passagiere und für Kreditkarten.

Das zweite Teilmodell ist das Action-Notification Model. Dieses spezifiziert Aktionen, welche ausgeführt werden können. Ein Beispiel für solche Aktionen ist das Auswählen eines Objektes aus einer Liste. Diese spezielle Aktion wird in dem Beispiel öfters verwendet, z.B. bei der Auswahl von Abflugs- und Ankunftsflughafen, sowie für die Auswahl eines Fluges. Eine andere Aktion ist zum Beispiel das Setzen einer Variablen. Es gibt eine Menge von vordefinierten Aktionen, welche speziell auch für die Anbindung an die Applikationslogik eine wichtige Rolle spielen. Zusätzlich hat der Designer die Möglichkeit eigene domänenspezifische Aktionen hinzuzufügen. (Popp und Raneburger 2011) beschreiben das Action-Notification Model im Detail.

Das dritte Teilmodel ist das Discourse Model. Es spezifiziert den Ablauf der Kommunikation. Dieses Modell basiert auf drei Theorien aus den Sprachwissenschaften bzw. der Soziologie. Die für unseren Ansatz wichtigste Theorie aus diesem Bereich ist die Sprechakt-Theorie (Searle 1969), welche besagt, dass man jede Kommunikation als Austausch von sogenannten Sprechakten sehen kann. Jeder Sprechakt hat eine bestimmte Intention und einen propositionalen Inhalt. Beispiele für eine Intention sind eine Frage und eine Aufforderung. Eine andere wichtige Theorie aus dem Bereich der Soziologie ist die Konversationsanalyse (Luff et al. 1990). Diese Theorie beschreibt den zeitlichen Zusammenhang zwischen Nachrichten. Sie besagt, dass eine Antwort erst dann erfolgen kann, wenn die dazugehörige Frage gestellt wurde. Dieses Konzept wird „Adjacency Pair" genannt. Ein weiteres Konzept dieser Theorie ist die „Inserted Sequence". Eine Inserted Sequence erlaubt einem Dialogpartner Rückfragen zu stellen, bevor dieser die ursprüngliche

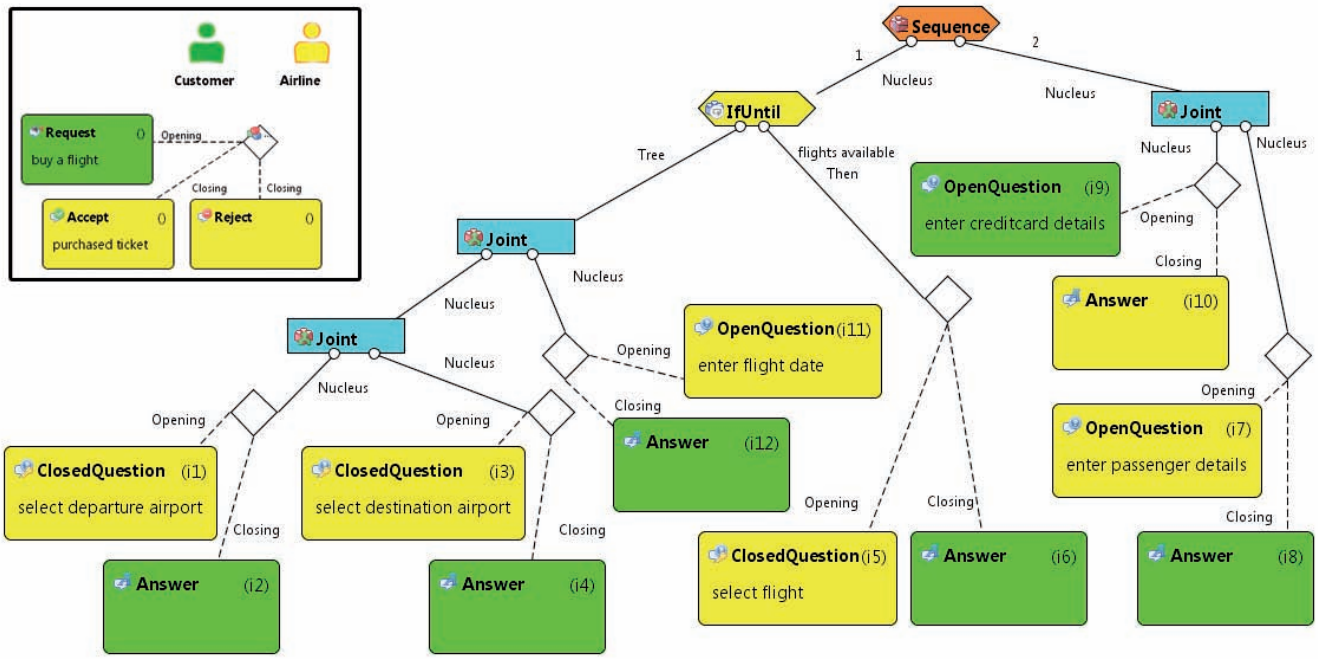

Bild 2: Discourse Model für Flugbuchung.

Nachricht beantwortet. Die dritte wichtige Theorie aus dem Bereich der Sprachwissenschaften ist die Rhetorical Structure Theory (Mann und Thompson 1988). Diese Theorie wird meist zur automatischen Textgenerierung genutzt und beschreibt den inhaltlichen Zusammenhang zwischen Textbausteinen. Das Discourse Model kombiniert einige wichtige Erkenntnisse aus diesen Theorien (Falb et al. 2006).

Bild 2 zeigt ein solches Modell. Die grünen und gelben abgerundeten Rechtecke symbolisieren die „Communicative Acts", welche eine Verallgemeinerung der Sprechakte darstellen. Die Farbe symbolisiert den Dialogpartner, welcher den Communicative Act äußert. Eine Frage wird mit der dazugehörigen Antwort mittels eines Adjacency Pairs verbunden, welches als Raute dargestellt wird. Diese Adjacency Pairs werden dann über sogenannte Discourse-Relationen verbunden, welche im Wesentlichen von den Relationen der Rhetorical Structure Theory abgeleitet sind. Jeder Communicative Act hat einen propositionalen Inhalt, welcher mit Hilfe des Domain-of-Discourse Models und des Action-Notification Models modelliert wird. In Adjacency Pairs wird normalerweise nur beim eröffnenden Communicative Act (z.B. der Frage) der propositionale Inhalt explizit spezifiziert, da der Inhalt des abschließenden Communicative Acts (z.B. der Antwort) daraus abgeleitet werden kann. Im linken oberen Bereich von Bild 2 sind die beiden Kommunikationsteilnehmer (Customer und Airline) und das Adjacency Pair zum Kauf eines Flugtickets dargestellt. Dieses Adjacency Pair beinhaltet eine Inserted Sequence, welche der Airline erlaubt, die dafür notwendigen Informationen abzufragen, bevor der Request beantwortet wird. In der Inserted Sequence werden zuerst die Informationen von Abflug- und Ankunftsflughafen sowie das Flugdatum abgefragt. Diese sind mit einer „Joint Relation" verbunden, welche besagt, dass diese Fragen gleichzeitig gestellt werden können und jede von ihnen beantwortet werden muss. Dieser Teildiskurs wird dann mittels „IfUntil" mit der Frage nach dem passenden Flug verbunden. Dieses IfUntil-Konstrukt besagt, dass der erste Teildiskurs wiederholt wird, bis eine Bedingung erfüllt ist. In unserem Beispiel ist diese Bedingung, dass es zum gewünschten Reisetermin einen passenden Flug gibt. Ist diese Bedingung erfüllt, wird dem Customer eine Liste der verfügbaren Flüge angeboten. Sobald dieser dann den gewünscht Flug ausgewählt hat, wird noch eine Frage bezüglich der Fluggastinformation und eine weitere Frage bezüglich der Bezahlung gestellt. Diese beiden Fragen sind wieder mittels einer Joint Relation verbunden. Diese ist mittels einer Sequence Relation mit dem restlichen Diskurs verbunden. Die Sequence Relation definiert die Reihenfolge, in der die einzelnen Teildiskurse ausgeführt werden müssen.

Das Kommunikationsmodell definiert mit Hilfe seiner drei Teilmodelle die kommunikative Interaktion zwischen zwei Dialogpartnern. Diese Definition ist eindeutig, sowohl was den Ablauf der Interaktion als auch den Inhalt der ausgetauschten Nachrichten angeht. Des Weiteren ist dieses Modell unabhängig vom Endgerät und der schlussendlich verwendeten UI-Bibliothek.

3. Automatisierte WIMP-UI Generierung für verschiedene Endgeräte

In diesem Abschnitt stellen wir unseren modellgetriebenen Ansatz vor, der die automatische Transformation von Kommunikationsmodellen in WIMP-UIs für verschiedene Endgeräte unterstützt. Diese Transformation findet zur Designzeit und nicht zur Laufzeit des UIs statt. Unser Ansatz nützt die Tatsache, dass Kommunikationsmodelle per se vom Endgerät unabhängig sind. Das zu erzeugende UI ist allerdings sehr wohl von physischen Einschränkungen, wie zum Beispiel der Bildschirmgröße und -auflösung des konkreten Endgerätes, und dem darauf verwendeten UI-Toolkit abhängig. Diese

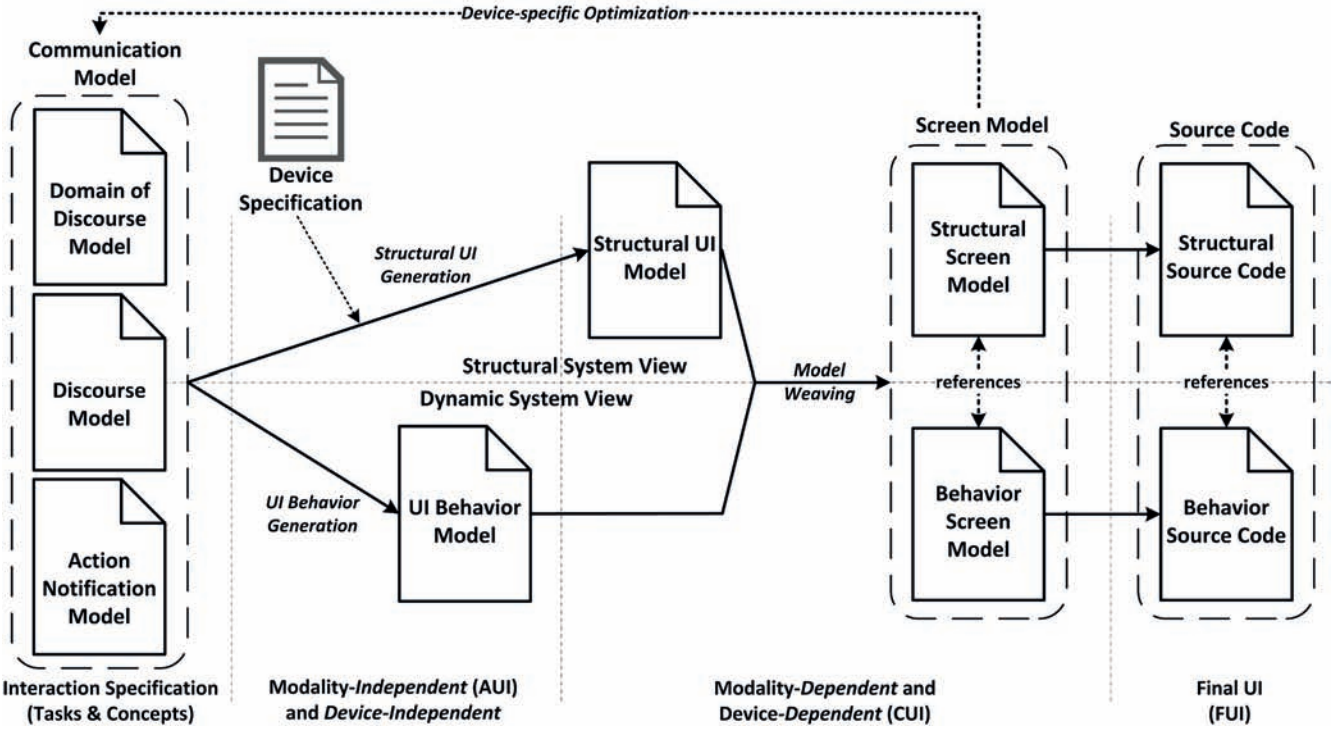

Bild 3: Generierungsprozess.

Zusatzinformation wird in unserem Ansatz durch eine Gerätespezifikation bzw. durch die ausgewählte graphische UI-Bibliothek während des Generierungsvorgangs eingebracht.

Bild 3 zeigt ein schematisches Abbild des Generierungsprozesses und seiner Artefakte. Am unteren Rand von Abbildung 3 kategorisieren wir die von uns verwendeten Modelle anhand des Cameleon Reference Frameworks, um den Vergleich mit anderen Ansätzen zu erleichtern.

Unser Kommunikationsmodell befindet sich am „Task&Concepts Level". Um eine vollständige UI-Spezifikation im Sinne von automatischer Quellcodegenerierung zu unterstützen, leiten wir ein strukturelles und ein dynamische UI-Modell aus einem Kommunikationsmodell ab. Alle Modelle, die wir der strukturellen Systemsicht zurechnen, finden sich in der oberen Hälfte von Bild 3 und alle Modelle der dynamischen Systemsicht in der unteren Hälfte.

Die Transformation vom Kommunikationsmodell zum strukturellen UI-Modell („Structural UI Model") ist regelbasiert. Die Transformationsregeln bilden Elemente aus dem Kommunikationsmodell auf UI-Widgets ab. Ein Beispiel hierfür ist die Abbildung des Frage-Antwort Adjacency Pairs nach einem Startflughafen auf eine Liste zur Flughafenauswahl und einen Knopf um die Antwort abzuschicken. Die konkrete Darstellungsform der Liste, d.h. ob sie als Liste mit Radiobuttons oder als Drop-down Liste dargestellt wird, hängt vom Endgerät ab. Einerseits spielen die physikalischen Randbedingungen (z.B. Bildschirmgröße und -auflösung) bei der Auswahl eine Rolle und andererseits die zu verwendende UI-Bibliothek (Verfügbarkeit von Widgets). Die Geräte-Spezifikation („Device Specification") definiert diese und andere Charakteristika, die während des Generierungsprozesses berücksichtigt werden. Außerdem referenziert diese Geräte-Spezifikation ein „Cascading Style Sheet", um die Beeinflussung von Layout- und Stil-Parametern zu ermöglichen. Zusätzlich zu diesen Einschränkungen haben wir drei weitere Kriterien definiert, um die Benutzbarkeit der generierten UIs zu verbessern (Raneburger et al. 2011b). Diese Optimierungskriterien sind:

a. maximale Ausnützung des verfügbaren Platzes,
b. minimale Anzahl an Navigationsklicks und
c. minimales Scrolling.

Unser Transformationsframework unterstützt das Feuern mehrerer Regeln für dieselben Kommunikationsmodellelemente, um mehr Flexibilität während des Generierungsprozesses zu bieten. Die drei Optimierungskriterien legen fest, in welcher Reihenfolge die Regeln feuern müssen um zu einem „optimalen" Ergebnis zu gelangen. Es gibt zwei Abbruchbedingungen der Optimierungsschleife („Device-specific Optimization"), welche am oberen Rand von Bild 3 zu sehen sind. Diese Abbruchbedingungen sind entweder ein UI, welches ohne Scrollen auf dem Endgerät dargestellt werden kann, oder der Zustand, dass keine andere Darstellungsmöglichkeit (d.h. Regelkombination) mehr verfügbar ist. Da unsere Optimierungskriterien eine Größenberechnung der UI Widgets erfordern, welche erst auf dem „Concrete User Interface (CUI) Level" verfügbar ist, verzichten wir auf ein modalitäten- und endgerätunabhängiges UI-Modell auf dem „Abstract User Interface (AUI) Level" und transformieren direkt zwischen Tasks&Concepts Level und Concrete User Interface (CUI) Level. Diese Tatsache und die Optimierung unterscheidet uns von allen anderen Transformationsansätzen.

Die Transformation vom Kommunikationsmodell zum dynamischen UI-Modell („UI Behavior Model") ist ebenfalls regelbasiert. Wir definieren das Verhalten für jedes Kommunikationsmodellelement

durch endliche Zustandsautomaten (Popp et al. 2009). Diese Zustandsautomaten setzen wir basierend auf dem zu transformierenden Kommunikationsmodell zu einem kombinierten endlichen Zustandsautomaten zusammen. Dieser kombinierte Zustandsautomat, welcher das UI Behavior Model darstellt, bildet den Ablauf der im Kommunikationsmodell spezifizierten Interaktion ab und definiert damit das Verhalten des UIs. Nachdem das Kommunikationsmodell Endgeräte- und Modalitäten- bzw. UI-Bibliotheks-unabhängig ist, ist dies auch das UI Behavior Model.

Durch die Einordnung unserer Modelle in die Ebenen des Cameleon Reference Framework wird in Bild 3 sofort ersichtlich, dass sich das strukturelle UI-Modell (Structural UI Model) und das dynamische UI-Modell (UI Behavior Model) auf unterschiedlichen Abstraktionsebenen befinden. Konkret berücksichtigt das strukturelle UI-Modell Endgerät-Charakteristika, wohingegen das dynamische UI-Modell Endgeräte-unabhängig ist. Um diese beiden Modelle zu einem konsistenten Modell, welches die Struktur und die Dynamik des UIs berücksichtigt, zusammenzuführen, haben wir einen Modellverknüpfungsprozess („Model Weaving") entwickelt (Raneburger et al. 2011a). Um dies zu erreichen nutzt dieser Prozess im Wesentlichen die einander ergänzenden Informationen aus dem strukturellen und dem dynamischen Modell. Das strukturelle Modell definiert, welche Communicative Acts zum selben Schirm (Screen) des UIs gehören. Anhand dieser Information werden die Zustände des dynamischen Modells gegebenenfalls in Unterzustände unterteilt. Umgekehrt wird die Information, welche Communicative Acts zu einem Zustandswechsel führen, genutzt, um die minimale Anzahl von „Submit" Knöpfen zu bestimmen. Dies verbessert die Bedienbarkeit. Das sogenannte „Screen Model" ist das Resultat unseres Verknüpfungsprozesses. Genau genommen besteht dieses Modell ebenfalls wieder aus einem strukturellen Screen Modell und einem dynamischen Screen Modell, diese Modelle sind jedoch eng gekoppelt. Ein Screen entspricht genau einem Zustand im dynamischen Modell und umgekehrt. Die Trennung dieser beiden Modelle wurde beibehalten um die Quellcodegenerierung zu erleichtern. Der Quellcode für das strukturelle UI wird für jede UI-Bibliothek bzw. -Notation (z.B. Java Swing, HTML) generiert. Der Quellcode für das dynamische Modell hingegen ist unabhängig von der konkreten UI-Bibliothek und muss nur einmal erzeugt werden.

Bild 4 zeigt drei Schirme (Screens) unseres Flugbuchungsbeispiels, die für das iPhone generiert wurden. Der linke Schirm zeigt die Darstellung der Frage-Antwort Adjacency Pairs nach Start- bzw. Zielflughafen und dem Flugdatum. Die Fragen nach Start- bzw. Zielflughafen werden jeweils als „Radio Button" Liste dargestellt. Da nicht beide Listen gleichzeitig auf dem iPhone-Schirm Platz haben, wurde von unserer Optimierung die Darstellung mittels Tabs gewählt. Damit ist zwar ein Navigationsklick mehr notwendig, aber Scrollen wurde vermieden. Die drei „Antwort Communicative Acts" zu den jeweiligen Fragen werden zugleich durch den „Submit" Knopf abgeschickt. Der mittlere Schirm entspricht der geschlossenen Frage nach einem Flug und bietet dem Benutzer die Möglichkeit einen auszuwählen. Der letzte Schirm entspricht den offenen Fragen nach der Kreditkarten-Information und den Passagierdaten. An dieser Stelle muss erwähnt werden, dass diese Schirme vollautomatisch generiert wurden, die Texte allerdings manuell angepasst wurden, da diese Informationen nicht im Kommunikationsmodell spezifiziert sind. Ebenfalls ist erwähnenswert, dass generierte UIs sogar für andere Smartphones, etwa Android-basierte Geräte mit anderer Schirmauflösung, anders aussehen können.

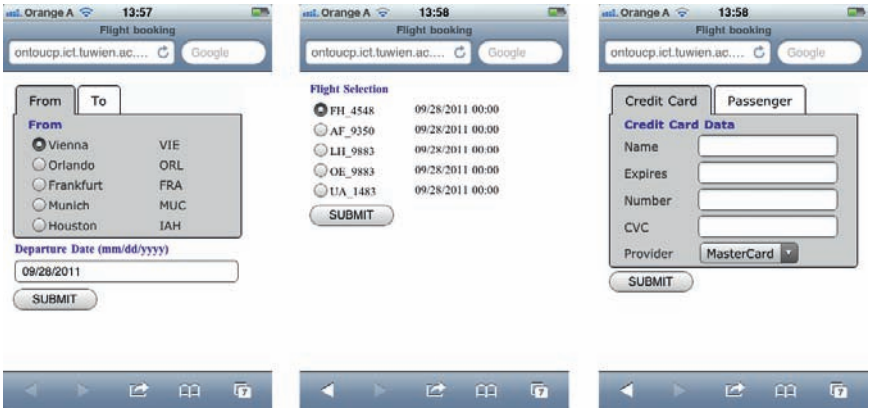

Bild 4: Flugbuchungs-UI für iPhone.

4. Anbindung des UI an die Applikationslogik

Ein wichtiger Punkt bei generierten UIs ist die Anbindung des generierten Codes an eine Applikationslogik. Der hier vorgestellte Ansatz bietet dafür eine über alle Applikationen einheitliche Schnittstelle. Für diese Schnittstelle wird das Kommunikationsmodel genutzt. Die Communicative Acts dienen zur Laufzeit als Nachrichtendefinition zwischen UI und Applikation. Diese Nachrichten beinhalten alle notwendigen Informationen, um die beiden Teile miteinander kommunizieren zu lassen, was auch eine Verteilung auf mehrere Geräte erlaubt. Beide Teile müssen dazu eine einheitliche Schnittstelle implementieren, welche nur eine Methode zum Empfangen von Communicative Acts bietet. Dadurch ist es leicht möglich, mit mehreren UIs ein und dieselbe Applikationslogik zu nutzen, oder ein UI für verschiedene Applikationslogiken anzubieten, welche aber ein und dasselbe Kommunikationsmodell unterstützen müssen (Popp 2009).

Aus dem Kommunikationsmodel können auch die einzelnen Aufrufe für die Applikation abgeleitet werden. Um die Aufrufe in ein standardisiertes Format zu bringen, werden das Action-Notification Model und das Domain-of-Discourse Model herangezogen. Mit diesen beiden Modellen gemeinsam können alle notwendigen Applikationsaufrufe spezifiziert werden. (Popp und Raneburger 2011) beschreiben dies im Detail.

5. Conclusio und Ausblick

In unserem Ansatz ist es möglich, mit Diskurs-basierten Kommunikationsmodellen Klassen von Dialogen zu spezifizieren. Aus einem solchen Modell kann unser Transformations-Werkzeug für Modell-Transformationen automatisiert konkretere Modelle und zuletzt den Code für WIMP-UIs generieren. Im Rahmen dieser Transformationen werden sogar Gerätespezifische Optimierungen vorgenommen, was unseren Ansatz gegenüber den anderen in der Literatur auszeichnet.

Eine vollautomatische Generierung führt aber insbesondere für Geräte mit größeren Bildschirmen wie etwa PCs im Allgemeinen (noch) nicht zu benutzerfreundlichen UIs. Dies liegt insbesondere daran, dass man die ästhetischen Kriterien menschlicher Designer (noch) nicht ausreichend modellieren kann. Daher verfolgen wir auch den Ansatz semiautomatischer Generierung, bei der sich ein menschlicher Designer interaktiv bei der Generierung einbringen kann, indem er das generierte Screen Model geeignet modifiziert, siehe (Raneburger 2010).

Danksagung

Die Entwicklung unseres Ansatzes wurde zu großen Teilen aus den folgenden Projekten finanziert: OntoUCP (Projektnummer 809254/9312), FIT-IT Programm der Österreichischen FFG; CommRob (IST-045441 im Rahmen des 6. Rahmenprogramms), gefördertes EU-Projekt. Wir danken der Firma Points Management GmbH, die Teile unserer Forschung in einem von der Österreichischen FFG geförderten Projekt finanziert hat. Weiters danken wir Edin Arnautovic, Christian Bogdan, Dominik Ertl, Helmut Horacek, Sevan Kavaldjian, Michael Leitner und Alexander Szep für ihre Mitarbeit an einzelnen Teilen des hier präsentierten Ansatzes.

Literatur

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

Falb, J.; Kaindl, H.; Horacek, H.; Bogdan, C.; Popp, R.; Arnautovic, E.: A discourse model for interaction design based on theories of human communication. In CHI ,06 Extended Abstracts on Human Factors in Computing Systems (CHI '06), 2006.

Kaindl, H.; Popp, R.; Raneburger, D.; Ertl, D.; Falb, J.; Szep, A.; Bogdan, C.: Robot-supported cooperative work: A shared-shopping scenario. In Proceedings of the 44th Hawaii International Conference on System Sciences (HICSS), 2011.

Kavaldjian, S.; Bogdan, C.; Falb, J.; Kaindl, H.: Transforming discourse models to structural user interface models. In Models in Software Engineering, LNCS 5002, Volume 5002/2008. Springer, Berlin / Heidelberg, 2008.

Limbourg, Q.; Vanderdonckt, J. M.: Comparing task models for user interface design. In D. Diaper and N. Stanton, editors, The Handbook of Task Analysis for Human-Computer Interaction, Kapitel 6. Lawrence Erlbaum Associates, Mahwah, NJ, USA, 2003.

Luff, P.; Frohlich, D.; Gilbert, N.: Computers and Conversation. Academic Press, London, UK, 1990.

Mann, W. C.; Thompson, S.: Rhetorical Structure Theory: Toward a functional theory of text organization. Text, 8(3):243–281, 1988.

Meixner, G.; Seissler, M.: Selecting the right task model for model-based user interface development. In Proc. ACHI 2011, The Fourth International Conference on Advances in Computer-Human Interactions, 2011.

Pastor, O.; España, S.; Panach, J. I.; Aquino N.: Model-driven development. Informatik Spektrum, 31(5), (2008) 394–407.

Paternò, F.; Mancini, C.; Meniconi, S.: ConcurTaskTrees: A diagrammatic notation for specifying task models. In Proceedings of the IFIP TC13 Sixth International Conference on Human-Computer Interaction, 1997.

Paternò, F.; Santoro, C.; Spano, L. D.: Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans. Comput.-Hum. Interact., 16:19:1-19:30, 2009.

Popp, R.: Defining communication in SOA based on discourse models. In Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications (OOPSLA '09), ACM Press: New York, NY, 2009.

Popp, R.; Falb, J.; Arnautovic, E.; Kaindl, H.; Kavaldjian, S.; Ertl, D.; Horacek, H.; Bogdan, C.: Automatic generation of the behavior of a user interface from a high-level discourse model. In Proceedings of the 42nd Annual Hawaii International Conference on System Sciences (HICSS-42), Piscataway, NJ, USA, 2009.

Popp, R.; Raneburger, D.: A high-level agent interaction protocol based on a communication ontology. In E-Commerce and Web Technologies, Volume 85 of Lecture Notes in Business Information Processing, (Editoren Huemer, C.; Setzer, T.; Aalst, W.; Mylopoulos, J.; Sadeh, N. M.; Shaw, M. J.; Szyperski, C.) Springer Berlin Heidelberg, 2011.

Raneburger, D.: Interactive model driven graphical user interface generation. In Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '10), ACM Press: New York, NY, USA, 2010.

Raneburger, D.; Popp, R.; Kaindl, H.; Falb, J.; Ertl, D.: Automated generation of device-specific WIMP UIs: Weaving of structural and behavioral models. In Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS '11, New York, NY, USA, 2011a.

Raneburger, D.; Popp, R.; Kavaldjian, S.; Kaindl, H.; Falb, J.: Optimized GUI generation for small screens. In Model-Driven Development of Advanced User Interfaces, Band 340, Studies in Computational Intelligence, (Editoren Hussmann, H.; Meixner, G.; Zuehlke, D.) Springer Berlin / Heidelberg, 2011b.

Searle, J. R.: Speech Acts: An Essay in the Philosophy of Language. Cambridge University Press, Cambridge, England, 1969.

Vanderdonckt, J. M.: Model-driven engineering of user interfaces: Promises, successes, and failures. In Proceedings of the 5th Annual Romanian Conf. on Human-Computer Interaction, Matrix ROM, Bucaresti, 2008.


1


2


3


4

1 Hermann Kaindl übernahm Anfang 2003 einen Lehrstuhl an der Technischen Universität Wien. Davor war er Senior Consultant im Bereich Programm- und Systementwicklung der Siemens AG Österreich. Er ist Senior Member der IEEE and Distinguished Scientist Member der ACM.
E-Mail: kaindl@ict.tuwien.ac.at

2 Roman Popp wurde 1976 in Wien geboren. Er studierte Elektrotechnik mit Fachrichtung Computertechnik an der Technischen Universität Wien. Das Studium beendete er im März 2003 mit ausgezeichnetem Erfolg. Seit Jänner 2003 ist er als wissenschaftlicher Mitarbeiter am Institut für Computertechnik in den Bereichen „Communication Protocols for SOA" und „User Interfaces" tätig.
E-Mail: popp@ict.tuwien.ac.at

3 David Raneburger arbeitet seit 2008 als Assistent am Institut für Computertechnik. Seinen Dipl.-Ing. erhielt er 2009 für eine Arbeit zum Thema „Automated graphical user interface generation based on an abstract user interface specification", von der Technischen Universität Wien. Im Rahmen seiner Doktorarbeit beschäftigt er sich mit der Verbesserung (semi)automatisch generierter UIs.
E-Mail: raneburger@ict.tuwien.ac.at

4 Jürgen Falb ist seit 1999 als Assistent am Institut für Computertechnik in den Bereichen Kommunikationssysteme und Benutzerschnittstellenentwicklung tätig. Sein Schwerpunkt liegt an der Verbindung von modellgetriebener Softwareentwicklung und Mensch-Maschine-Interaktion. Seine Doktorarbeit behandelte modellgetriebene Entwicklungsmethoden für nomadische Systeme.
E-Mail: falb@ict.tuwien.ac.at