Ein UML-basierter Ansatz für die modellgetriebene Generierung grafischer Benutzerschnittstellen

Verena Kluge, Frank Honold, Felix Schüssel, Michael Weber
Institut für Medieninformatik
Universität Ulm
89081 Ulm

verena.kluge@alumni.uni-ulm.de
{frank.honold, felix.schuessel, michael.weber}@uni-ulm.de

Abstract: Die Erstellung der Benutzerschnittstelle nimmt mittlerweile den größten Teil im Entwicklungsprozess ein. Je komplexer Anwendungen werden, desto zeit- und kostenintensiver wird es, eine Oberfläche zu erstellen. Eine automatisierte Lösung könnte hier Abhilfe schaffen. Vorhandene Techniken bieten maximal eine Teilautomatisierung und sind zudem nicht in den Arbeitsprozess der Entwickler ohne Störungen einzubinden, da sie oftmals eigene Modellierungssprachen voraussetzen.

Der hier vorgestellte Ansatz basiert auf dem Cameleon Reference Framework und verwendet als Ausgangsbasis für die Generierung die UML. So kann er ohne aufwendiges Einarbeiten angewendet werden. Er ermöglicht automatisch aus UML-Diagrammen ein GUI für unterschiedliche Zielplattformen zu erzeugen. Durch den Einsatz von Beautifications ist es zudem möglich, wiederverwendbare Anpassungen der Oberfläche zu erstellen und so Expertenwissen bezüglich Interaktion und Gestaltung einzubringen.

1 Einleitung

Durch die Erkenntnis, welch wesentlichen Beitrag die Benutzerschnittstelle (kurz: UI) für die Bedienbarkeit, die Akzeptanz und schlussendlich für den Erfolg einer Anwendung leistet, nimmt die Erstellung des UIs mittlerweile den größten Teil im Entwicklungsprozess ein [MR92].

Zur Unterstützung der UI-Erstellung existieren verschiedene Ansätze. So erleichtern Toolkits, Patterns und WYSIWYG-Editoren die Entwicklungsarbeit. Sie bieten jedoch meist noch keine integrierte automatisierte Lösung, welche gerade bei großen und komplexen Anwendungen eine Kosten- und Zeitersparnis gegenüber der herkömmlichen Erstellung bieten würde. Die Bereitstellung ein und derselben Anwendung für unterschiedliche Zielplattformen – mit teils unterschiedlichen Interaktionskonzepten – steigert den Implementierungsaufwand abermals. Es fehlt an einer Lösung, die es ermöglicht, selbst für komplexe Anwendungen aus einer leicht verständlichen, abstrakten und plattformunabhängigen Interaktionsbeschreibung Benutzerschnittstellen für unterschiedliche Zielplattformen abzuleiten.

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4

Mittels einer automatisierten Lösung könnten nicht nur bei der Implementierung Zeit und Kosten gespart werden, es bestünde darüber hinaus auch die Möglichkeit, frühzeitig und kostengünstig Einblicke in die späteren Interaktionsmöglichkeiten mit einem System zu erhalten. Denn gerade in frühen Phasen der Software-Entwicklung können bspw. im Bezug auf Kundenanforderungen Missverständnisse auftreten, welche anhand eines UIs aufgedeckt und ausgeräumt werden könnten.

Die Grundlage für dieses Vorgehen bildet das Cameleon Reference Framework[1] [CCT+03]. Es definiert eine Richtlinie für die Strukturierung des UI-Entwurfs. Hierbei werden verschiedene Kontexte in der Erstellung berücksichtigt. Das Framework besteht dabei unabhängig vom eigentlichen Software-Entwicklungsprozess.

Trotzt eines hohen Grades an Automatisierung muss es dem Entwickler möglich sein, seine eigenen Vorstellungen in das entstandene UI einzubringen. So soll es möglich sein, Erfahrungen aus der Ergonomie und Interaktionspsychologie mit in die erzeugte Oberfläche einzubringen. Auch Änderungen am Design, wie Form und Farbänderungen der Anwendung sowie ihrer Elemente sollen nachträglich möglich sein. Hierdurch kann die Benutzerschnittstelle an die spezifischen Bedürfnisse der jeweiligen Benutzergruppe angepasst werden. Bei allem soll sich die abstrakte Beschreibung durch bestehende und etablierte Konzepte der Informatik formulieren lassen.

2 Aktueller Stand

Mit steigendem Komplexitätsgrad einer Software empfiehlt sich eine strukturierte Softwareentwicklung. Der Einsatz von Modellen kann den Erstellungsaufwand von Software verringern. Abhängig vom Entwicklungsschritt werden unterschiedliche Modelle verwendet. Jedes Modell beschreibt einen bestimmten Aspekt der Software. Durch die Kombination von Modellen erhält man eine Modellsicht des Gesamtsystems.

Für die Beschreibung werden heute unterschiedliche Modelle herangezogen. Die verbreitetsten sind CTT, XIML, UIML, UsiXML und UMLi [Pat99, SV03, TZV03, VLM+04, SP00]. Hierbei muss jedoch i. d. R. die gesamte Beschreibung von Hand erstellt werden; eine teilweise Automatisierung bieten Teresa [MPS04], Tadeus [SE96] und der Wisdom Ansatz [NeC02].

Das ConcurTaskTree (CTT) Modell von Paternò dient der Aufgabenbeschreibung [Pat99]. Es besteht aus grafischen Modellen, welche hierarchisch zerlegte Aufgaben darstellen. Mit ihm werden Benutzeraufgaben hierarchisch definiert, jedoch wird durch den hierarchischen Aufbau die Entwicklung von kleinen Aufgaben besonders aufwändig. Weitere Einschränkungen sind die fehlende Möglichkeit zur Darstellung mehrerer parallel aktiver Aufgaben und die fehlende History Funktionalität.

Die eXtensible Interactive Markup Language (XIML) wurde von RealWhale Inc. als Sprache für interaktionsbezogene Daten entwickelt [SV03]. Mittels XIML soll der gesamte Entwicklungszyklus berücksichtigt werden. Dieser beinhaltet Gestaltung, Organisation,

[1] Cameleon Project: http://giove.isti.cnr.it/projects/cameleon.html (Zugriff am: 01.03.2011)

Test und Ausführung. Die UIs werden vom abstrakten zum konkreten Modell beschrieben. Durch die Trennung von abstrakter Beschreibung und Präsentation kann XIML verschiedene Plattformen unterstützen. XIML kann mehrere Modelle unterstützen und lässt sich erweitern. Es existiert für XIML lediglich eine geringe Werkzeugunterstützung.

Die User Interface Markup Language (UIML) ist eine auf XML basierende Markupsprache [TZV03]. Sie unterstützt die Beschreibung von UIs, welche durch einen Renderer auf eine bekannte UI Sprache abgebildet wird. Entwickelt wurde UIML ab 1997 vom Center for Human Computer Interaction an der Virginia Polytechnic Institute and State University und der Harmonia Inc. Die Trennung von UI Code und Anwendungslogik ist ein Ziel der UIML. Da es in UIML jedoch lediglich eine Sprache gibt um UIs zu beschreiben, muss für jedes Gerät ein separates UI definiert werden. Da keine abstrakten Modelle für die frühe Phase der Modellierung erstellt werden können, sind die Daten direkt an das Interface gekoppelt und müssen so ständig synchronisiert werden, um die Konsistenz sicher zu stellen. Bei mehreren Zielplattformen skaliert dieser Ansatz aufgrund der Datenbindung an das Interface eher schlecht.

Die USer Interface eXtensible Markup Language (UsiXML) ist eine plattformunabhängige Sprache zur Beschreibung von UIs [VLM+04]. Für die Modellierung von UIs stellt UsiXML verschiedene Modelle bereit, die gemäß dem Cameleon Reference Framework strukturiert sind und unterstützt deren Transformation. UsiXML unterstützt zudem unterschiedliche Kontexte wie verschiedene Benutzer-, Plattform- und Umgebungskontexte. Jedes in UsiXML beschriebene Modell kann separiert werden. Dadurch kann der Designer entscheiden, was in der UI Beschreibung enthalten sein soll und was nicht. Für UsiXML existieren verschiedene Werkzeuge, die die Gestaltung unterstützen. UsiXMLCode kann mit verschiedenen Toolkits erzeugt werden. Ein Beispiel hierfür ist GrafiXML, mit dem die Oberfläche definiert wird. Als Aufgabenmodell wird CTT verwendet, welches um Benutzerstereotypen erweitert wurde. Bei Daske [Das09] wird gezeigt, wie UML-Diagramme als Aufgabenmodell in UsiXML verwendet werden können. Dies erfordert jedoch einen Zwischenschritt über State Web Charts, in die die UML-Diagramme zunächst umgeformt werden müssen, was aufwendig ist.

Die Unified Modeling Language for Interactive applications (UMLi) entstand 1998 in einem Forschungsprojekt an der Universität Manchester [SP00]. Durch UMLi soll die Erstellung von grafischen Benutzerschnittstellen (kurz: GUIs) vereinfacht werden. In der Modellierung mit UMLi werden die Ein/Ausgabemöglichkeiten sowie mögliche Interaktionsabläufe beschrieben. Hierbei wird nicht das detaillierte Aussehen dargestellt, sondern die Semantik des GUIs. UMLi unterstützt jedoch nur formularbasierte Interfaces. Die Gestaltung bleibt jedoch abstrakt und vermeidet eine Konkretisierung der Oberfläche.

Das Teresa Projekt ist eine modellbasierte Entwicklungsumgebung für die Erstellung von UIs für unterschiedliche Plattformen [MPS04] und dient der Beschreibung von abstrakten UIs. Zu Beginn wird ein plattformunabhängiges Aufgabenmodell mittels CTT erzeugt. Es enthält alle Aufgaben, die mit der Anwendung umgesetzt werden können. Die Transformationslogik von Teresa kann von einem Entwickler nur schwer dahingehend beeinflusst werden, dass mit individuellen Modelltransformationen Einfluss auf die GUI-Generierung genommen werden kann.

ISBN 978-3-88579-286-4

Tadeus unterstützt die UI Entwicklung im gesamten Entwicklungsprozess [SE96]. Es ist ein modellbasierter Ansatz und besteht aus verschiedenen Modellen. Sie beschreiben Systemaktionen, wichtige Entitäten der Anwendung und die unterschiedlichen Rollen der Benutzer. Bis zur automatisierten Generierung muss bei Tadeus selbst bei einem Minimalbeispiel viel Vorarbeit in Bezug auf die Erstellung der benötigten Modelle (CTT, DomainModel, userModel . . .) geleistet werden.

Wisdom ist eine auf UML basierende Notation mit deren Hilfe sog. Präsentationsmodelle erstellt und anschließend mittels XSLT in eine GUI-Beschreibung in AUIML (Abstract User Interface Markup Language) transformiert werden können [NeC02]. Das Wisdom-Modell enthält dabei schon konkrete Aktionen und Ein- und Ausgaben, die für den Benutzer bestimmt sind. Dies wird durch eigene UML-Notationselemente ermöglicht.

Die vorgestellten Ansätze sind aufwändig in ihrer Anwendung. So ist CTT durch den hierarchischen Aufbau schon für kleine Aufgaben relativ aufwändig und nicht in der Lage parallel aktive Aufgaben darzustellen. Dies spricht gleichsam gegen den Einsatz von UsiXML, dessen Aufgabenmodell auf CTT basiert. In UIML können keine abstrakten Modelle in der frühen Phase der Modellierung erzeugt werden.

Die Modellierungsmethode XIML verlangt einen hohen Aufwand für den Entwickler und es existiert kaum Werkzeugunterstützung für diese Methode. UMLi ist vor allem für den Einsatz bei formularbasierten Interfaces geeignet und somit für den hier gewählten Ansatz nicht geeignet. Auch die beiden Entwicklungsumgebungen Teresa und Tadeus eignen sich nicht, da der Entwickler auf die GUI-Generierung kaum Einfluss nehmen kann und der Erstellungsaufwand erhöht ist. Der Ansatz mittels Wisdom kommt dem hier vorgestellten Ansatz recht nahe, erfordert jedoch schon eine sehr konkrete Vorstellung der späteren Oberfläche, da Ein- und Ausgaben und mögliche Aktionen bereits spezifiziert sein müssen. Zudem beinhaltet das Wisdom-Präsentationsmodell eigene Notationselemente und erfordert so eine gesonderte Einarbeitung. Auf das Design der generierten Oberfläche kann keinerlei Einfluss genommen werden.

3 Ansatz

Wie das vorhergehende Kapitel zeigt, gibt es noch keinen Ansatz, der eine automatische Generierung grafischer Oberflächen erlaubt, ohne dass sich Entwickler in neue Modellierungssprachen einarbeiten müssen. Ziel unseres Ansatzes ist es, eine Generierung von Oberflächen basierend auf verbreiteten Modellierungssprachen zu ermöglichen, um so den zusätzlichen Aufwand für eine solche Generierung zu minimieren. Da sich UML für diese Art der Modellierung als geeignet herausgestellt hat [dM10] und als verbreiteter Standard in vielen Softwareprojekten eingesetzt wird, dient sie als Ausgangspunkt des hier vorgestellten Frameworks. Für eine umfassende Beschreibung des Ansatzes sei auf [Klu10] verwiesen.

Als konzeptuelle Basis unseres Ansatzes wurde das Cameleon Reference Framework (kurz: CRF) gewählt. Wie dieses besteht es aus Modellen, welche von einer abstrakten zu einer konkreten Beschreibung führen. Die vier Abstraktionsebenen des Cameleon Reference

Frameworks (*Tasks&Concepts*, *Abstract UI*, *Concrete UI* und *Final UI*), werden in dem hier vorgestellten Framework noch um eine fünfte Ebene zwischen abstraktem und konkretem UI ergänzt: die gerätespezifische Ebene. Mit dieser Erweiterung wird eine Beschreibung für verschiedene Gerätearten möglich.

Abbildung 1 zeigt den strukturellen Aufbau des Frameworks. Im folgenden wird zunächst ein grober Überblick über den Ablauf des Frameworks gegeben. Eine detaillierte Beschreibung der verwendeten Modelle und Transformationen an einem konkreten Beispiel folgt in Kapitel 4.

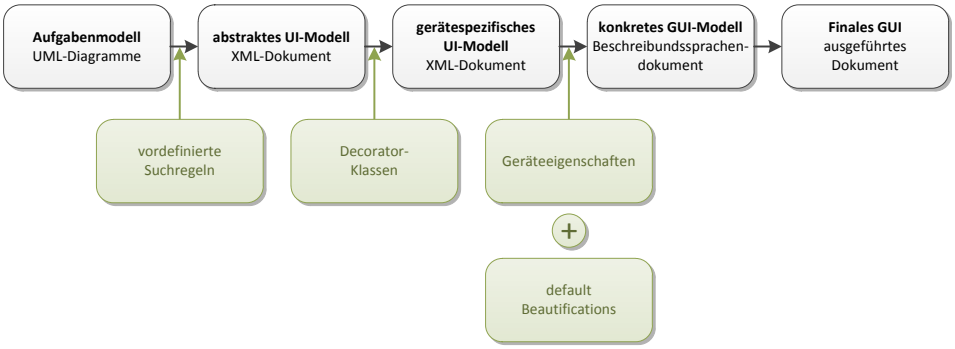

Abbildung 1: Konzeptueller Ablauf des erstellten Transformations-Tools

Ausgangspunkt für den hier vorgestellten Ansatz ist ein in UML erstelltes *Aufgabenmodell*. Mit Hilfe *vordefinierter Suchregeln* wird aus dem UML-Modell ein *Abstraktes UI-Modell* in XML erstellt, indem im UML-Diagramm Strukturen gesucht werden, die bestimmten Interaktionsschemata entsprechen. Diese Interaktionsschemata werden im späteren Verlauf auf konkrete Widgets abgebildet.

Das abstrakte UI-Modell wird anschließend mit Hilfe von *Decorator-Klassen* in ein oder mehrere *gerätespezifische UI-Modelle* überführt. *Geräteeigenschaften* und *default Beautifications* dienen nun dazu, das *konkrete GUI-Modell* zu erzeugen, das in einer konkreten Oberflächen-Beschreibungssprache vorliegt. Die *finale GUI* resultiert aus der Ausführung des konkreten GUI-Modells in einer Laufzeitumgebung.

4 Der Generierungsprozess

Zur Veranschaulichung des motivierten Frameworks wird im Weiteren der Prozess der Generierung unterschiedlicher Benutzerschnittstellen beschrieben. Als Szenario dient eine Radioanwendung. Als repräsentative Interaktionen wurden Ein-/Ausschalten, Lautstärkeregelung, Ton an/aus, Frequenzsuche (automatisch und manuell), Senderspeichern, Senderanzeige und Senderliste umgesetzt. Als Zielpattform für die Oberflächengestaltung wurde ein Desktop und ein PDA gewählt. Bei der angewandten UI-Beschreibungssprache

handelt es sich in beiden Fällen um XAML. Bei dem hier präsentierten Ansatz stand die Machbarkeit der automatischen Generierung und nicht das endgültige Aussehen der Oberflächen im Vordergrund. Ausgehend von einer abstrakten Beschreibung in UML werden zwei unterschiedliche und plattformspezifische XAML-Dateien erstellt.

4.1 Das Aufgabenmodell

Das Aufgabenmodell entspricht dem Task Model des CRFs. Es besteht aus den im Systementwurf erstellten UML-Zustands- und Klassendiagrammen (vgl. Abb. 2 bis 5). Die Erstellung erfolgt wie gewohnt nach den Regeln der UML-Diagramme, welche um einige Regeln erweitert werden. Zu diesen Regeln gehören die Stereotypen `<<range>>` und `<<decorator>>` sowie unter anderem der Parametertyp *List*, hierzu später mehr. Für eine

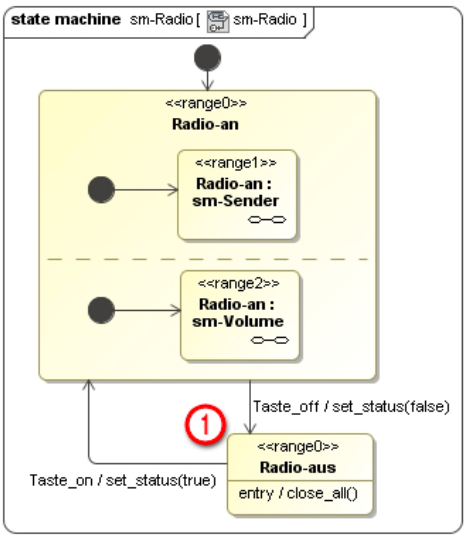

Abbildung 2: UML-Zustandsdiagramm einer einfachen Anwendung als Aufgabenmodell. Die einzelnen integrierten Zustandsautomaten zur Sender- und Lautstärkesteuerung (vgl. Abb. 3 und 4) werden durch entsprechende Range-Stereotypen abstrakt strukturiert.

bessere Weiterverarbeitung des Aufgabenmodells wird es im XML Metadata Interchange (XMI) Format abgespeichert und in das Framework eingelesen. Das XML-Dokument wird mittels einer in Java implementierten Lösung von Schmitzl und de Melo [dM10, SdM08] bereinigt. Dies erleichtert die Lesbarkeit sowie den Speicherbedarf des Dokuments erheblich. Nach dieser ersten Bearbeitung erfolgt der Übergang zum abstrakten Modell.

4.2 Das abstrakte UI-Modell

Für die Erstellung des abstrakten UI-Modells sind vordefinierte Suchregeln notwendig, welche es dem Framework ermöglichen aus dem Aufgabenmodell die Struktur verschiedener Interaktionsschemata zu identifizieren. Zu den bisher unterstützten Interaktionsschemata gehören:

- das Auslösen einer Funktion
- das Anzeigen von Ereignissen
- das Setzen von Werten aus einem Bereich
- das Ein-/Ausschalten von Zuständen sowie
- die Auswahl aus einer Liste.

Die Suchregeln bilden die Grundlage zur automatischen GUI-Erzeugung. Die Abstraktion des Modells bleibt dabei jedoch gewahrt, denn durch die Suchregeln findet noch keine Zuordnung zu konkreten Interaktionselementen statt. Es wird lediglich nach Strukturen in den UML-Zustands- und Klassendiagrammen gesucht. Zu den erwähnten Interaktionsschemata sind bisher folgende Regelnd definiert:

Regel 1 (Auslösen einer Funktion) sucht nach Transitionen mit einem *Effect*, welche keine weiteren Parameter besitzen. Bei Transitionen mit identischen Effekten wird nur eine Transition verwendet. So wird eine doppelte Anwendung vermieden (vgl. Abb. 4, ①).

Regel 2 (Anzeigen von Ereignissen) sucht nach Zuständen mit einer *do-Activity* ohne ausgehende Transitionen (vgl. Abb. 3, ①).

Regel 3 (Setzen von Werten aus einem Bereich) sucht nach Transitionen, in denen ein *Guard* enthalten ist. Im *Guard* sind die Minimal- und Maximalwerte definiert. Des Weiteren enthält die Transition den Parametertyp Integer (vgl. Abb. 4, ②).

Regel 4 (Ein-/Ausschalten von Zuständen) sucht nach zwei Transitionen mit identischer Operationensignatur. Die Transitionen enthalten den Parametertyp Boolean (vgl. Abb. 2, ①).

Regel 5 (Auswahl aus einer Liste) sucht nach Transitionen, welche den Parametertyp List besitzt (vgl. Abb. 3, ② in Kombination mit Abb. 5, ①).

Wird durch die Suchregeln eine Struktur gefunden, werden die zugehörigen Informationen aus dem Aufgabenmodell in ein neues XML-Dokument geschrieben, das als abstraktes UI-Modell fungiert. Je nach Regel sind unterschiedliche Informationen aus dem Aufgabenmodell für das damit identifizierte Interaktionsschema relevant. So extrahiert z.B. Regel 3, die das Setzen von Werten aus einem Bereich identifiziert, die Minimal- und Maximalwerte des Bereichs aus dem Guard der Transition und fügt sie als Attribute in das XML-Dokument des abstrakten UI-Modells ein. Zu den Regeln ist anzumerken, dass die Reihenfolge ihrer Anwendung nicht von Bedeutung ist. Für die im XML-Dokument

vorkommenden Zustände und Transitionen werden stets alle aufgestellten Regeln durchlaufen und auf Übereinstimmung geprüft. So ist es möglich, dass mehrere Regeln auf eine Struktur zutreffen. Ist dies der Fall, werden alle passenden Regeln angewandt. Das dadurch erstellte, abstrakte UI-Modell enthält somit alle gefundenen Strukturen und bleibt offen für Anpassungen des Entwicklers.

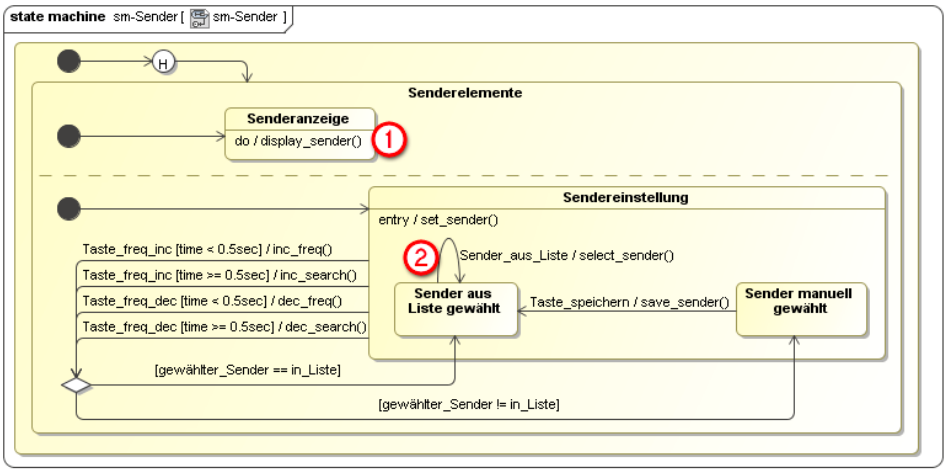

Abbildung 3: Modell eines abstrakten Sender-Interaktionskonzepts mit Senderanzeige und unterschiedlichen Methoden der Senderauswahl.

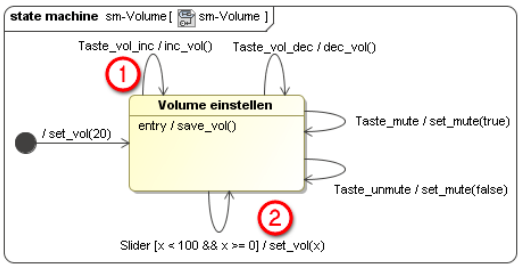

Abbildung 4: Modell eines abstrakten Lautstärke-Interaktionskonzepts mit unterschiedlichen Interaktionsmöglichkeiten.

Zusätzlich zu den aufgestellten Regeln kann über einen `<<range>>` Stereotypen die Platzierung der Interaktionselemente im späteren final GUI-Modell beeinflusst werden (vgl. Abb. 2). Der Stereotyp kann mehrere Unterstereotypen besitzen. Diese werden mit einer Nummer versehen, beginnend mit null. Je nach Höhe der Zahl wird damit ein Bereich im späteren UI festgelegt. So kann eine Range mit der Nummer null bspw. als „der obere Bereich eines GUI-Fensters“ interpretiert werden. Die anderen folgen entsprechend in aufsteigender Reihenfolge. Hierfür wird eine `<<rangeX>>` an einen Zustand gebunden (s.

Abb. 2). Alle zu diesem Zustand gehörenden Unterzustände gehören zu der Range des Oberzustands. Innerhalb der angegebenen Range werden die Transitionen und Zustände in interne Gruppen angeordnet. Zu einer internen Gruppe gehören alle von einem Zustand ausgehenden Transitionen.

4.3 Gerätespezifische Modelle

Damit das UI für verschiedene Geräte erstellt werden kann, wird aus dem abstrakten UI-Modell für jeden gewünschten Gerätetyp ein gerätespezifisches Modell erstellt. Dieses Modell ist notwendig, da sich die Funktionalitäten von Geräten unterscheiden können. Mittels des gerätespezifischen Modells können diese Funktionalitäten für das jeweilige Gerät festgelegt werden.

Um das gerätespezifische Modell erstellen zu können, werden benötigte Informationen aus den vorher erstellten Modellen ausgelesen. So enthält das Aufgabenmodell für jeden Gerätetyp im Klassendiagramm eine Decorator-Klasse (vgl. Abb. 5). Die Zuordnung von Decorator-Klasse zu Gerätetyp erfolgt durch den entsprechenden Klassennamen. Dabei enthalten diese Klassen alle Operationen, die in dem entsprechenden Gerät dargestellt werden sollen.

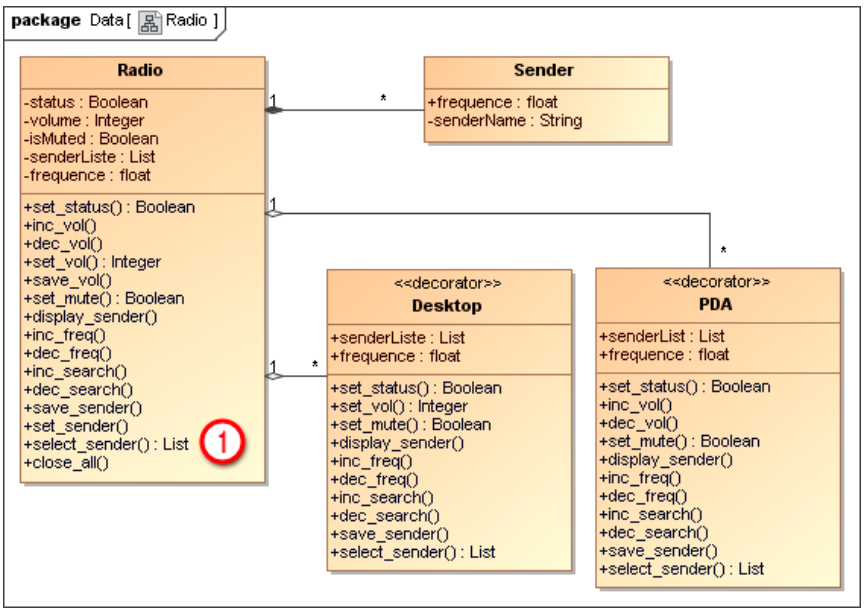

Abbildung 5: UML-Klassendiagramm der Beispielanwendung. Unterschiede in den plattformspezifischen Decorator-Spezifikationen bedingen unterschiedliche Benutzerschnittstellen (vgl. `set_vol():Integer` vs. `inc_vol()` und `dec_vol()`).

Für jede Decorator-Klasse werden nun die im abstrakten UI-Modell stehenden Operationen der Interaktionsschemata (zuvor über die Regeln identifiziert) mit denen der Decorator-Klassen verglichen. Bei einer Übereinstimmung werden die Interaktionsschemata in das entsprechende gerätespezifische UI-Modell geschrieben. Am Ende dieses Prozesses existiert für jedes spezifizierte Gerät, respektive jede Plattform, ein eigenes XML-Dokument welches alle zu realisierenden Funktionalitäten enthält.

4.4 Konkretes GUI-Modell

Durch eine XSL-Transformation wird aus dem gerätespezifischen das konkrete Modell. Hierfür werden nicht nur die gerätespezifischen Modelle herangezogen, sondern auch das abstrakte UI-Modell. Das abstrakte UI-Modell wird verwendet, da es *alle* aus den UML-Diagrammen ausgelesenen Interaktionsschemata enthält. Wird aus ihnen nun ein konkretes Modell abgeleitet, kann der Entwickler sich einen schnellen Überblick über alle realisierte Elemente machen und auf dieser Grundlage entscheiden, ob alle gewünschten Interaktionselemente mittels UML realisiert wurden. Ein weiterer Grund für die Verwendung des abstrakten Modells ist, dass auch dann ein konkretes GUI-Modell entstehen kann, wenn kein gerätespezifisches Modell abgeleitet wurde.

Die für die Transformation verwendete XSLT-Datei konkretisiert die Benutzerschnittstelle. Erst über die XSLT-Datei werden Fenster-, Widgetgröße, konkrete Interaktionselemente sowie deren Farben je nach Geräteart definiert und den gefundenen Strukturen zugeordnet. Dies geschieht über sogenannte Beautifications, die in Abschnitt 5 gesondert betrachtet werden. Bis dato werden lediglich default-Beautifications eingesetzt, die das allgemeine Aussehen einer Oberfläche beschreiben (vgl. Abb. 1).

Anhand der zuvor definierten *Ranges* wird ein Raster definiert. Für die Berechnung dieser Bereiche werden nun die im abstrakten Modell hinzugefügten Informationen über die Anzahl der internen Gruppen verwendet. Abschließend werden den Interaktionsschemata konkrete Widgets zugeordnet. Je nach Gerät können so aus identischen Interaktionsschemata unterschiedliche Widgets entstehen.

Durch dieses späte Festlegen des eigentlichen Aussehens einer GUI ist es möglich, aus den vorhergehenden Modellen Anwendungen in verschiedenen Beschreibungssprachen zu erstellen. Im vorliegenden Fall wurde als Beschreibungssprache XAML eingesetzt. Das konkrete und plattformspezifische GUI-Modell besteht in diesem Fall aus unterschiedlichen XAML-Dateien.[2]

4.5 Finales GUI-Modell

Das finale Modell entsteht durch Ausführung des konkreten Modells. In Abbildung 6 werden die mittels default Beautifications und den Decorator-Klassen generierten Desktop und

[2] An dieser Stelle sind natürlich auch andere UI-Beschreibungssprachen denkbar.

PDA GUIs, die durch die Ausführung der generierten XAML-Beschreibung entstehen.

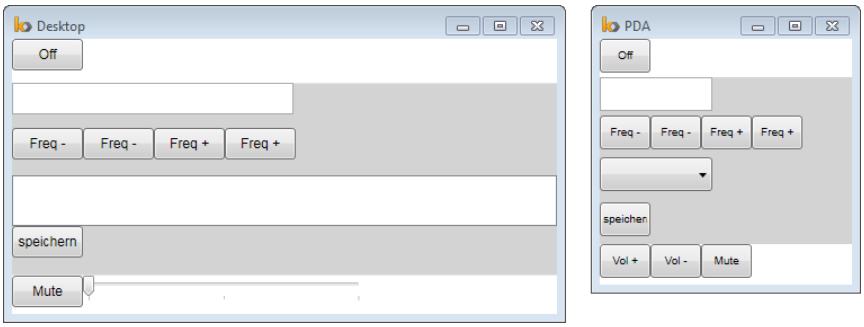

Abbildung 6: Mittels default Beautifications generierte konzeptionelle Desktop- und PDA-Benutzerschnittstellen. Alternierende Hintergrundfarben kennzeichnen die einzelnen Ranges.

Die abgeleiteten GUIs unterscheiden sich im skizzierten Beispiel in Größe und Art einiger Widgets. So wird die Senderliste auf der Desktop-GUI als Liste und im Falle der PDA-GUI als Dropdown-Liste dargestellt. Weiter unterscheiden sich die Widgets für die Lautstärkeregelung (PDA: zwei Button, Desktop: Slider). Die restlichen Widgets wurden in gleicher Form und Position dargestellt, da sich die modellierten Konzepte weder in ihren Decorator-Klassen noch in ihren Beautifications unterscheiden. Zur besseren Unterscheidung wurden die unterschiedlichen Bereiche der Oberfäche (Ranges) durch Beautifications abwechselnd weiß-grau hinterlegt. Das GUI des PDAs wird aus Repräsentationszwecken ebenfalls als Fenster dargestellt.

5 Beautifications

Beautifications sind ein durch Pederiva et. al. motiviertes Konzept [PVE+07]. Es besteht aus optischen Änderungen eines UIs, welche nicht aus den UML-Diagrammen ausgelesen werden können. Zu diesen Änderungen gehören unter anderem Designaspekte wie: Hintergrundfarbe, die Art des Interaktionselements (bei mehreren Möglichkeiten), sowie deren Größe und Farbe.

Eine erste Anwendung finden die Beautifications im Übergang zum konkreten GUI-Modell (siehe Abb. 1). Diese *default Beautifications* werden einmalig definiert und können als Standard auf alle GUIs angewendet werden. Der Vorteil dieses Vorgehens ist es, dass dadurch bspw. ein Corporate Design problemlos auf alle erzeugten Anwendungen angewandt werden kann. Durch die Wiederverwendbarkeit der Beautifications wird ein einheitliches Erscheinungsbild sichergestellt. Nach dieser ersten Gestaltung kann der Entwickler durch weitere Beautifications Feinheiten aus dem GUI herausarbeiten. Dies wird in Kapitel 6 näher erläutert.

Damit für verschiedene Anwendungen geltende Beautifications nicht für jede Verwendung

neu angefertigt werden müssen, können sie separat gespeichert und so wiederverwendet werden. Für eine klarere Strukturierung wurden die Beautifications in zwei Kategorien aufgeteilt. So gibt es Design Beautifications und Structure Beautifications, welche im Folgenden erläutert werden.

5.1 Design Beautifications

In den Design Beautifications werden Änderungen an Farbe, Form und Größe der Anwendungselemente vorgenommen. So können verschiedene Hintergrundarten ausgewählt werden, zum Beispiel Fenster, Panel usw. Ebenso können Farbe und Größe nach den Wünschen des Entwicklers angepasst werden. Weiter umfassen die Design Beautifications Anpassungen der konkreten Widgets. So kann ein Widget verschiedene Formen besitzen, wie etwa ein Button, der mit eckigen oder abgerundeten Ecken dargestellt werden kann. Auch andere Formen, wie Dreiecke oder Bilder, sind denkbar. Wie beim Hintergrund können auch hier Farbe und Größe angepasst werden. Da ein Interaktionselement auch eine Beschriftung besitzen kann, ist auch dessen Erscheinungsbild für jedes Interationselement separat oder allgemein veränderbar. Geändert werden können Schriftart, -farbe und -größe. Weiter kann ein GUI Logos und Bilder enthalten. Diese werden über Design Beautifications in das generierte GUI integriert. Auch Änderungen der Art des Interakionselements sind möglich. So kann bspw. ein ToggleButton durch zwei RadioButtons ersetzt werden. Sollen aus einer Struktur regelmäßig andere Interaktionselemente abgeleitet werden, so kann dies über eine Änderung der Suchregeln dauerhaft bewerkstelligt werden.

5.2 Structure Beautifications

Für die räumliche Strukturierung einer Anwendung werden die Structure Beautifications verwendet. Durch sie kann das Layout beeinflusst werden. Es gibt an, wo bzw. wie die aus dem UML-Diagramm ausgelesenen Interaktionselemente in der Anwendung platziert werden. Die Platzierung der Interaktionselemente ist für die Bedienbarkeit einer Anwendung von Bedeutung. Liegen Interaktionselemente mit zusammengehörenden Funktionen zu weit auseinander wird die Anwendung unübersichtlich. Hier kann der Entwickler seine Erfahrungen aus dem Bereich Usability einbringen und so die Oberfläche zu einem durchgängig stimmigen Konzept machen.

6 Anwendung von Beautifications

In einem weiteren Anpassungsschritt wird es dem Entwickler ermöglicht eigene Vorstellungen über das Design einfließen zu lassen und Änderungen vorzunehmen. Der Entwickler wählt eine zu bearbeitende XAML-Datei sowie eine XSLT-Datei aus. Die XSLT-Datei enthält alle vom Entwickler gewünschten Änderungen. Hierzu zählen u.a. Elementerwei-

terungen, das Einführen von neuen Attributen, das Ändern von vorhandenen Attributwerten oder das Löschen bzw. Ersetzen von Elementen (Widgets) durch Widgets mit gleicher Grundstruktur. Abbildung 7 zeigt die mittels eigener Beautifications veränderten GUIs.

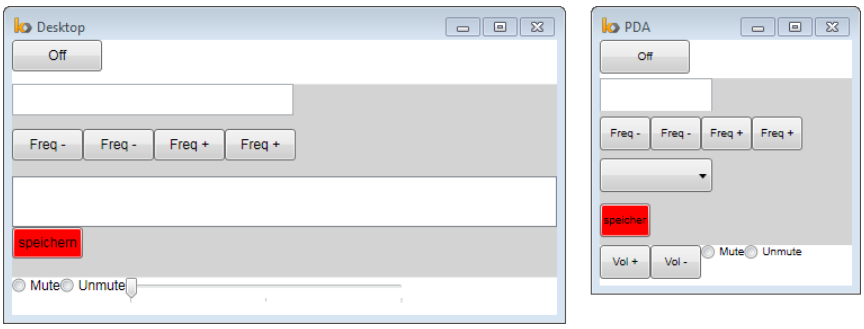

Abbildung 7: Mittels spezifischer Beautifications generierte Desktop- und PDA-GUIs

Im Vergleich zu Abbildung 6 hat sich das UI in der Abbildung 7 folgendermaßen geändert: der ToggleButton „`On/Off`“ wurde verbreitert, die Farbe des Buttons „`speichern`“ wurde geändert und das ToggleButton Konzept des „`Mute/Unmute`“-Buttons wurde in das RadioButton Konzept geändert.

7 Schlussfolgerung

In diesem Beitrag wurde ein Konzept zur vollautomatischen Generierung von grafischen Benutzeroberflächen aus UML-Diagrammen vorgestellt. Gegenüber bestehenden Ansätzen besitzt es den Vorteil, dass für die Beschreibung und Erzeugung einer Benutzerschnittstelle nur noch UML für die abstrakte Interaktionsbeschreibung benötigt wird. UML stellt einen verbreiteten Standard dar und minimiert somit die Einarbeitungszeit für Anwendungsentwickler. Weiter besteht durch den Einsatz von Beautifications für den Entwickler die Möglichkeit, eigene Vorstellungen in die Oberflächengestaltung einfließen zu lassen. Diese können wiederverwendet werden und müssen so nicht für jede Anwendung neu erstellt werden. Mit den decorator-Klassen wird es möglich, eine Oberflächenbeschreibung für mehrere Geräte zu verwenden. Diese Punkte sorgen für eine Zeit- und Kostenersparnis bei der Erstellung von Benutzerschnittstellen. In seiner Gesamtheit ermöglicht der Ansatz die Integration von Expertenwissen bezüglich Interaktion und Gestaltung und dessen modellgetriebene Realisierung.

8 Ausblick

Das hier vorgestellte Framework ist in der Lage, ausgehend von einer in UML-Modellen vorliegenden Beschreibung, automatisch Oberflächen zu generieren. An einigen Stellen besteht aber noch Verbesserungspotential.

Wie in den generierten GUIs (vgl. Abb. 6 und 7) zu erkennen, tauchen die Buttons „`Freq -`“ und „`Freq +`“ zum Einstellen der Sender-Frequenz doppelt auf. Dies liegt daran, dass die Transitionen zum Einstellen der Frequenz im Aufgabenmodell (Abb. 3) tatsächlich jeweils doppelt definiert sind. Im vorliegenden Fall sieht die Spezifikation im Aufgabenmodell vor, dass die Einstellung der Senderfrequenz je nach Dauer des Tastendrucks (`time` < bzw. >= `0.5sec`) unterschiedliche Anwendungsfunktionen auslöst, wie dies auch bei realen Radios üblich ist (manuelle Frequenzeinstellung und automatische Sendersuche). Da diese Semantik nicht erfasst wird, erstellt das Framework für die Transitionen in der späteren GUI jeweils eigene Buttons. Hier fehlt es an einer detaillierteren Auswertung der Guards, welche deren Semantik erfasst und entsprechend umsetzt.

Die Anwendbarkeit und Funktionalität des Frameworks könnte mittels einiger Erweiterungen ausgebaut werden. So könnte eine Anwendung das Einbringen neuer Strukturfindungsregeln erleichtern. Hierdurch könnten weitere Interaktionsschemata durch das Framework erkannt werden. Um auch komplexere GUIs mit mehreren Fenstern generieren zu können, sollten Aktivitätsdiagramme hinzugenommen werden. Ein weiteres Ausbaupotential des Frameworks besteht in einem Frontend für das leichtere Anwenden der Beautifications. Hier wäre bspw. eine graphische Oberfläche zur Veränderung des default-GUIs denkbar, wodurch Änderungen direkt sichtbar würden. Des Weiteren könnte diese Anwendung das umsetzen auf andere Zielplattformen weiter erleichtern. Auch die Umsetzung auf nicht XML-basierte Beschreibungssprachen wäre denkbar. Diese Erweiterungen würden das Gesamtkonzept des vorgestellten Frameworks abrunden und erweitern.

Abgesehen von diesen Verbesserungsvorschlägen wäre die Anwendbarkeit des Ansatzes zu Generierung multimodaler UIs ein interessantes Forschungsfeld. So könnte das abstrakte (bzw. das gerätespezifische) UI-Modell durchaus auch zur Generierung eines Sprachinterfaces herangezogen werden, indem die dort hinterlegten Interaktionsschemata statt in ein XAML Dokument zum Beispiel in ein VoiceXML Dokument überführt werden.

9 Danksagung

Diese Arbeit ist im Rahmen des Sonderforschungsbereich SFB/Transregio 62 “Eine Companion-Technologie für kognitive technische Systeme” entstanden, gefördert durch die Deutsche Forschungsgemeinschaft (DFG).

Literatur

[CCT+03] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon und Jean Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15:289–308, 2003.

[Das09] Lisa Daske. Generierung prototypischer GUIs auf Basis von UML Zustands- und Aktivitätsdiagrammen. Diplomarbeit, Universität Ulm, 2009.

[dM10] Guido M. de Melo. *Modellbasierte Entwicklung von Interaktionsanwendungen*. Dissertation, Ulm University, 2010.

[Klu10] Verena Kluge. Modellgetriebene Generierung graphischer Benutzeroberflächen. Diplomarbeit, Universität Ulm, 2010.

[MPS04] Giulio Mori, Fabio Paterno und Carmen Santoro. Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. *IEEE Trans. Softw. Eng.*, 30:507–520, August 2004.

[MR92] Brad A. Myers und Mary Beth Rosson. Survey on user interface programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '92, Seiten 195–202, New York, 1992. ACM.

[NeC02] Nuno Jardim Nunes und João Falcão e Cunha. Towards Flexible Automatic Generation of User-Interfaces via UML and XMI. In *5th Workshop Iberoamericano de Ingenieria de Requisitos y Ambientes Software, IDEAS 2002*, 2002.

[Pat99] Fabio Paterno. *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, London, UK, 1. Auflage, 1999.

[PVE+07] Inés Pederiva, Jean Vanderdonckt, Sergio España, José I. Panach und Oscar Pastor. The Beautification Process in Model-Driven Engineering of User Interfaces. In C. Baranauskas, P. Palanque, J. Abascal und S. D. J. Barbosa, Hrsg., *Human-Computer Interaction - INTERACT 2007*, Jgg. 4662/2009 of *LNCS*, Seiten 411–425. Springer Verlag. Berlin, Heidelberg, 2007.

[SdM08] Jochen Schmitzl und Gudio M. de Melo. UML 2.0 Diagram Interchange Import für State Charts und Activity Charts. Bachelor-thesis, Ulm University, 2008.

[SE96] E Schlungbaum und T. Elwert. Dialogue Graphs- a Formal and Visual Specification Technique for Dialogue Modelling. In C. R. Roast und J. I. Siddigi, Hrsg., *BCS-FACS Workshop on Formal Aspects of the Human Computer Interface*. Springer-Verlag, 1996.

[SP00] Paulo Pinheiro Da Silva und Norman W. Paton. UMLi: The Unified Modeling Language for Interactive Applications. In *Proceedings of UML2000, volume 1939 of LNCS*, Jgg. 1939 of *LNCS*, Seiten 117–132. Springer, 2000.

[SV03] Nathalie Souchon und Jean Vanderdonckt. A Review of XML-compliant User Interface Description Languages. In *Interactive Systems. Design, Specification, and Verification*, Jgg. 2844 of *LNCS*, Seiten 391–401. Springer Berlin / Heidelberg, 2003.

[TZV03] Shari Trewin, Gottfried Zimmermann und Gregg Vanderheiden. Abstract user interface representations: how well do they support universal access? In *CUU '03: Proceedings of the 2003 conference on Universal usability*, Seiten 77–84, New York, 2003. ACM.

[VLM+04] Jean Vanderdonckt, Quentin Limbourg, Benjamin Michotte, Laurent Buillon, Daniela Trevisan und Murielle Florins. UsiXML: a User Interface Description Language for Specifying Multimodal User Interfaces. In *Proceedings of W3C Workshop on Multimodal Interaction WMI'2004*, Seiten 1–7. W3C, 2004.