T. Gross (Hrsg.): Mensch & Computer 2007: Konferenz für interaktive und kooperative Medien. München: Oldenbourg Verlag. 2007, S. 29

Evolutionäre Generierung von Applikationen aus OOA-Modellen

Steffen Büder

Fakultät Informatik, Technische Universität Dresden

Zusammenfassung

Im vorliegenden Beitrag wird eine evolutionäre Methode zur Generierung von Anwendungssoftware aus objektorientierten Analysemodellen aufgezeigt, bei der Fehler im UI-Design wie beispielsweise nicht erwartungskonforme Dialoge durch Analyse des Nutzerverhaltens automatisch erkannt und im jeweils folgenden Generierungszyklus behoben werden. Hierzu werden Nutzertypen mit ihrem Erfahrungs- und Wissensstand und ihren Präferenzen definiert. Anhand der Nutzung von Dialogen und Eingabekomponenten werden fehlplatzierte oder falsch dimensionierte Eingabekomponenten erkannt und entsprechend korrigiert. Auf diese Weise können für jeden Nutzertyp eigene speziell angepasste Dialoge für dieselbe Sicht auf das Datenmodell ohne zusätzlichen Entwicklungsaufwand generiert werden.

1 Einleitung

Applikationen werden in vielen Fällen von Anwendungsentwicklern erstellt, die sich in die fachspezifische Domäne, in der die Applikation zum Einsatz kommen soll, einarbeiten und auf Basis dieses Wissens versuchen, eine erwartungskonforme Dialogschicht zu entwerfen. Die Anforderungen und Vorstellungen der zukünftigen Anwender und die der Entwickler an die neue Applikation sind dabei trotz intensiver Gespräche in den allermeisten Fällen nicht identisch, da ein Entwickler in der Regel einen anderen Blick auf die Applikation hat als ein Sachbearbeiter, der sie nur verwendet.

Die in diesem Beitrag beschriebene Methode greift nun diesen Ansatz auf und geht davon aus, dass viele Designfehler erst durch den Anwender bemerkt werden, wenn dieser die Applikation im praktischen Einsatz hat. Ziel ist es diese Fehler automatisch zu erkennen und (halb-)automatisch in der nächsten Softwareversion zu beheben.

Das grundsätzliche Vorgehen baut auf dem JANUS-System[1] auf und gestaltet sich folgendermaßen. Der Entwickler erstellt ein fachspezifisches Analysemodell (OOA-Modell), aus dem vollautomatisch die Datenschicht und ein erster Entwurf der Dialogschicht generiert wird. Anschließend werden die nichtgenerierbaren Programmteile programmiert und offensichtliche UI-Designfehler vom Entwickler korrigiert. Der Anwender erhält nun diese erste Programmversion und kann sie bereits praktisch einsetzen. Im Hintergrund wird das Verhalten des Nutzer bei der Arbeit mit der Applikation aufgezeichnet und später analysiert. Aus diesen Ergebnissen werden anhand vordefinierter Regeln Entwurfsverbesserungen abgeleitet und beim nächsten Generierungszyklus automatisch umgesetzt. Die entstehende folgende Softwareversion ist dadurch bereits deutlich besser an das vom Nutzer gewünschte bzw. erwartete Programmverhalten angepasst.

Die Kombination der Softwaregenerierung mit einer automatisierten Nutzeranalyse und Entwurfskorrektur stellt eine deutliche Verbesserung gegenüber anderen Forschungsansätzen zur Generierung von Applikationen dar. Das JANUS-System wird hierzu um ein Nutzertypen-, ein Komponenten- und das Sichtenmodell erweitert, die in den nachfolgenden Abschnitten detailliert vorgestellt werden.

2 Übersicht

Abbildung 1 gibt einen Überblick über die Methode und ihre Teilmodelle. Zentraler Bestandteil ist das fachspezifische objektorientierte Analysemodell (OOA-Modell), welches vom Entwickler nach Analyse der Anwendungsdomäne erstellt wird. Hierbei werden alle fachspezifischen Objekttypen[2] mit ihren Eigenschaften und zu erwartenden Wertebereichen modelliert und Assoziationen zwischen den Objekttypen bewertet mit Kardinalitäten hergestellt (Niemann 2000). Dieses OOA-Modell bildet die Grundlage für alle folgenden Generierungsschritte. Bei Erweiterungen bzw. Anpassungen der Applikation für zukünftige Versionen werden die Änderungen wieder zuerst in das OOA-Modell eingepflegt.

[1] JANUS wurde als Forschungsprojekt an der Ruhr-Universität-Bochum entwickelt und wird von der otris software AG (www.otris.de) kommerziell weiterentwickelt und vertrieben.

[2] Fachspezifische Objekttypen einer Personalverwaltung sind beispielsweise „Mitarbeiter“ und „Fortbildung“, die mit einer N:M-Assoziation verknüpft sind.

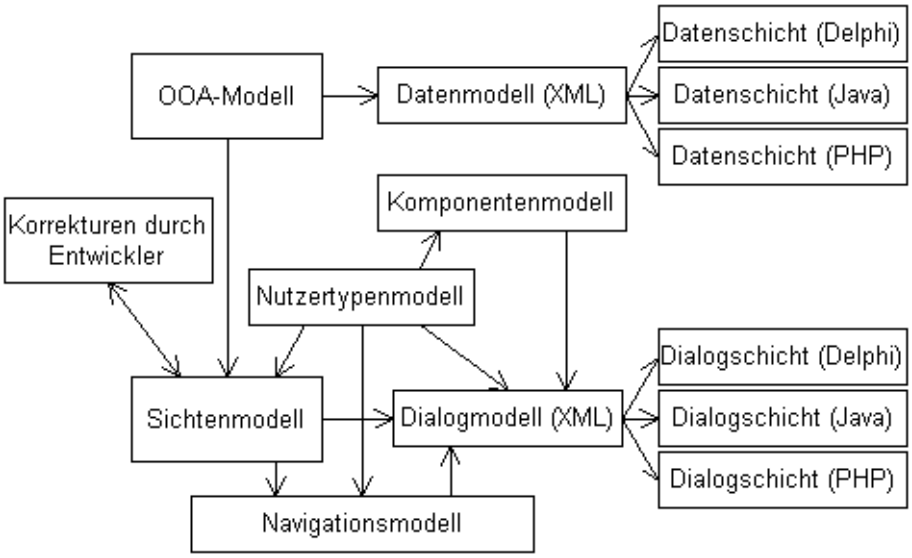

Abbildung 1: Übersicht

Anhand dieser Übersicht werden in den folgenden Kapiteln die Teilmodelle und das Vorgehen detaillierter vorgestellt.

Abbildung 2 veranschaulicht den automatisierten Entwicklungszyklus, der in einer Art Spiralmodell zu einer schrittweisen Verbesserung der Software führt. Während der Nutzer eine Version n der generierten Software praktisch einsetzt, werden im Hintergrund sein Verhalten und seine Eingaben protokolliert. Aus diesen Informationen werden anhand vordefinierter Regeln Entwurfsverbesserungen abgeleitet, die dann in den nächsten Generierungszyklus einfließen und zu einer verbesserten Version n+1 der Software führen.

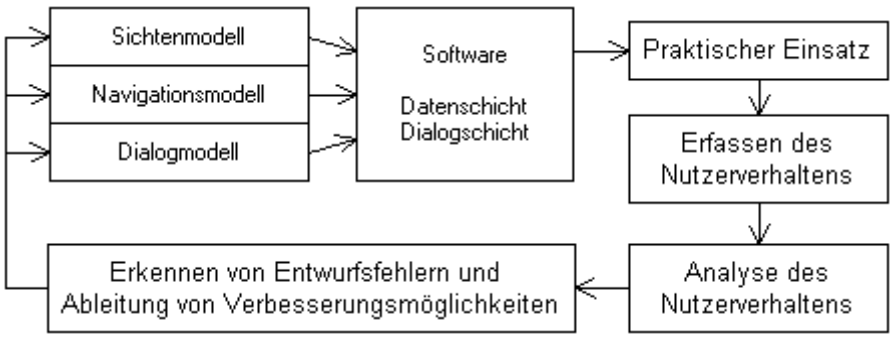

Abbildung 2: Entwicklungszyklus

3 Generierung der Datenschicht

Aus dem OOA-Modell wird vollautomatisch das zielsprachenunabhängige Datenmodell generiert. Dieses Datenmodell enthält neben der Beschreibung der Objekttypen, Attribute und Assoziationen auch deren Abbildung auf eine relationale Datenbank. Die Regeln für diese Abbildung werden in (Niemann 2000) hergeleitet und können für diesen Ansatz übernommen werden.

Das Datenmodell wird in einer XML-Sprache gespeichert und kann anschließend durch Transformation in die Datenschicht der jeweils gewünschten Zielsprache (Java, Delphi, PHP etc.) überführt werden. Die Transformation braucht nur einmalig für jede Zielsprache implementiert zu werden und kann dann auf alle mit dieser Methode generierten Datenmodelle angewendet werden. Durch diesen Zwischenschritt einer XML-Beschreibung des Datenmodells lässt sich der Generator leicht an neue Zielsprachen aber auch an firmeninterne Anforderungen und Frameworks anpassen.

Das Datenmodell lässt sich im Gegensatz zur Dialogschicht vollständig ohne weitere Zusatzinformationen aus dem OOA-Modell generieren und ist logischerweise für alle Nutzertypen identisch.

4 Modellierung und Generierung der Dialogschicht

Für die Generierung der Dialogschicht werden fünf Teilmodelle (siehe Abbildung 1) benötigt, die jeweils bestimmte Aspekte der Anforderungen abbilden:

Das Nutzertypen- und das Komponentenmodell sind nicht anwendungspezifisch, können allerdings auch für jede Applikation speziell angepasst werden.

Im Sichtenmodell werden die darzustellenden Inhalte des Datenmodells für konkrete Aufgaben definiert und das Navigationsmodell beschreibt die möglichen Navigationspfade zwischen diesen Sichten.

Das Dialogmodell umfasst alle Dialoge der Applikation und ist das Ergebnis des Generierungsprozesses.

Alle Teilmodelle werden jeweils in einer XML-Sprache beschrieben, um einen einfachen Austausch der Informationen zu ermöglichen. Das Erstellen der Modelle durch den Entwickler kann und soll allerdings in grafischer Form erfolgen. Für das Daten- und das Sichtenmodell werden hierfür in (Niemann 2000), WebML (Ceri et al. 2000) und (Weisbecker 1993) geeignete Notationen vorgeschlagen.

4.1 Nutzertypenmodell

Das Nutzertypenmodell verwendet Stereotypen und gruppiert die zu erwartenden Nutzer der Applikation entsprechend ihrer Fähigkeiten und Präferenzen im Umgang mit Anwendungs-

software im Allgemeinen (z.B. versiert oder unversiert, bevorzugte tabellarische Eingabe oder Einzeldialoge) und ihrem fachbezogenen Wissenstand im Speziellen (z.B. Sachbearbeiter, fachfremder Mitarbeiter). Das folgende Beispiel zeigt beispielhaft zwei Nutzertypen:

```
<usertypes>
   <usertype name="newbie">
      <userproperty name="experience_software" value="1" />
      <userproperty name="experience_domain" value="1" />
      <userproperty name="knowledge_software" value="2" />
      <userproperty name="knowledge_domain" value="2" />
      <userproperty name="preference_keyboard" value="1" />
      <userproperty name="preference_mouse" value="5" />
      <userproperty name="preference_tableedit" value="0" />
      <userproperty name="preference_shortcuts" value="0" />
      ...
   </usertype>
   <usertype name="executive_officer">
      <userproperty name="experience_software" value="2" />
      <userproperty name="experience_domain" value="7" />
      <userproperty name="knowledge_software" value="2" />
      <userproperty name="knowledge_domain" value="9" />
      <userproperty name="preference_keyboard" value="9" />
      <userproperty name="preference_mouse" value="1" />
      <userproperty name="preference_tableedit" value="6" />
      <userproperty name="preference_shortcuts" value="9" />
      ...
   </usertype>
   ...
</usertypes>
```

Der „Neuling“ hat weder Ahnung von seinem Fachbereich noch besitzt er Erfahrung im Umgang mit Software. Der leitende Sachbearbeiter hingegen ist zwar mit Software im Allgemeinen nicht sehr erfahren, besitzt aber umfangreiches Wissen auf seinem Fachgebiet und mit der speziellen Software, die er einsetzt.

Neben einigen fest vorgegebenen `<userproperty>` können beliebig viele frei definiert werden. Im Komponentenmodell wird dann auf diese Eigenschaften und ihre Bewertungen für jeden Nutzertyp Bezug genommen. Dadurch kann mit präzise definierten Nutzertypen eine Optimierung der Dialogschicht für jeden Nutzertyp erreicht werden.

4.2 Komponentenmodell

Das Komponentenmodell beschreibt die Menge der vom Generator verwendbaren Eingabekomponenten[3] mit ihrer entsprechenden Parametrisierung und ihren Eigenschaften wie z.B. die minimale Höhe und Breite. Im Normalfall spiegelt das Komponentenmodell die in einem GUI-Editor verfügbaren Komponenten wider.

Im Folgenden ist die Komponente „`TcxSpinEdit`“ beispielhaft dargestellt. Mit dem XML-Wert `<parameter>` werden Eigenschaften festgelegt, die der Entwickler klassischerweise im

[3] Eingabekomponenten sind beispielsweise Textfelder, Checkboxen und Listboxen.

GUI-Editor einstellt, um Komponenten an den konkreten Anwendungsfall anzupassen. Diese Parameter werden direkt in das Dialogmodell übernommen und erst bei der Transformation in die Zielsprache berücksichtigt. Für den Generator sind die Werte `<property>` und `<selectfor>` entscheidend, um Komponenten auszuwählen und geeignet zu positionieren.

```
<components>
  <component name="TcxSpinEdit">
    <parameter name="valuetype" value="float" />
    <parameter name="alignment" value="right" />
    <parameter name="displayformat" value="0.00 €" />

    <property name="height_min" value="10" />
    <property name="height_max" value="25" />
    <property name="width_min" value="50" />
    <property name="width_max" value="500" />

    <property name="distance_x" value="10" />
    <property name="distance_y" value="10" />
    <property name="show_label" value="true" />
    <property name="label_alignment" value="left-relativ" />
    ...
    <selectfor>
      <ooa_property name="datatype" value="float" priority="5" />
      <ooa_property name="datatype" value="text" priority="0" />
      <ooa_property name="datatyperange" valuemin="0" valuemax="10000" />
      <userproperty name="experience_software" values="0..9" priority="5" />
      <userproperty name="preference_keyboard" values="0..3" priority="2" />
      <userproperty name="preference_keyboard" values="4..9" priority="6" />
      <userproperty name="preference_mouse" values="0..9" priority="1" />
      <userproperty name="preference_tableedit" values="0..2" priority="4" />
      ...
    </selectfor>
  </component>
  ...
</components>
```

Der `selectfor`-Abschnitt besteht aus einer Menge von Regeln, die abhängig von den Attributeigenschaften der Objekttypen des OOA-Modells (z.B. Datentyp) und den Eigenschaften der Nutzertypen (`<userproperty>`) eine passende Komponente auswählen. Der Generator addiert die Prioritäten der zutreffenden Eigenschaften, um die jeweils geeignetste Komponente zu finden. Über die Prioritäten kann somit gesteuert werden, welche Eingabekomponenten bei bestimmten Nutzertypeigenschaften sinnvoll oder weniger geeignet sind, abhängig von der Bewertung (`values`) der Eigenschaft bei den einzelnen Nutzertypen. Das Komponentenmodell ist beliebig erweiterbar und kann dadurch an die in einem Unternehmen eingesetzten Komponentenbibliotheken angepasst werden. Das ist ein großer Vorteil gegenüber anderen Generatorsystemen, die diese Möglichkeit nicht bieten.

Mit der Kombination aus Nutzertypen- und Komponentenmodell lässt sich auf diese Weise für jeden Nutzertyp eine optimale Benutzungsschnittstelle zusammenstellen. Die Positionierung der Eingabekomponenten kann mit dem von (Hofmann 1998) und (Kruschinski 1999) vorgeschlagenen Verfahren erfolgen.

4.3 Sichtenmodell und Navigationsmodell

Während Nutzertypen- und Komponentenmodell anwendungsübergreifend einheitlich sein können, werden im Sichtenmodell die in der konkreten Applikation benötigten Sichten auf das OOA-Modell beschrieben, ergänzt um Aktionen und abgeleitete Attribute, die aus Attributen des Datenmodells berechnet werden. Im einfachsten Fall gibt es für jeden Objekttyp des OOA-Modells genau eine Sicht zum Anlegen und Bearbeiten der entsprechenden Objekte. Viel wichtiger sind allerdings die Sichten zur Lösung komplexer Aufgaben, beispielsweise das Zusammenstellen einer Rechnung aus Personenobjekt, Produktobjekten und früheren Zahlungen. Ein erster Vorschlag für das Sichtenmodell kann vom Generator erstellt werden, indem die Kardinalitäten der Assoziationen ausgewertet und daraus kombinierte Sichten für zusammengehörige Objekte erstellt werden. Dieser erste Entwurf muss allerdings vom Entwickler in den allermeisten Fällen überarbeitet werden. Die vom Entwickler durchgeführten Änderungen werden dabei aufgezeichnet und beim nächsten Generierungszyklus automatisch angewendet. Dadurch kann das OOA-Modell später erweitert werden, ohne das der Entwickler wieder ein komplett neues Sichtenmodell erstellen muss. Für die Modellierung der Sichten wird ein an WebML (Ceri et al. 2000) und (Weisbecker 1993) angelehntes Modellierungsverfahren verwendet. Es arbeitet mit „Views“, „Pages“, verschiedenen Content-Elementen wie beispielsweise „Single-Selection“, „Multi-Selection“, „List“, „Single-Input“, „Output“, „Table“ und Elementen für Aktionen, z.B. „Load“, „Store“ und „Back“. Damit lassen sich Sichten mit dem darzustellenden Inhalt und den erlaubten Nutzeraktionen zusammenstellen. Aus einer Sicht können bei der Generierung auch mehrere Dialoge entstehen, beispielsweise für unerfahrene Nutzer mehrere kleine Dialoge zur schrittweisen Eingabe. Wohingegen für versierte Nutzer ein einzelner Dialog mit allen Eingabemöglichkeiten sinnvoll ist.

Neben dem Sichtenmodell muss vom Entwickler auch das zugehörige Navigationsmodell mit den benötigten Geschäftsprozessen erstellt werden. Es enthält die Informationen, wie die einzelnen Sichten verknüpft und dem Nutzer angezeigt werden. Das Navigationsmodell bestimmt maßgeblich die Menüstruktur der Applikation. Später wird in der Nutzeranalyse auch geprüft, entlang welcher Pfade der Nutzer häufig oder gar nicht navigiert. Diese können dann entsprechend optimiert werden.

4.4 Dialogmodell

Aus den oben beschriebenen fünf Modellen plus den ab dem zweiten Zyklus vorliegenden Nutzeranalyseergebnissen erstellt der Generator das Dialogmodell in einem XML-Format. Es enthält alle Dialoge, Eingabekomponenten aus dem Komponentenmodell und deren Positionen, so dass es sich eindeutig in das UI einer konkreten Zielsprache überführen lässt. Analog zur Transformation des XML-Datenmodells in die spezielle Zielsprache ist es auch hierfür ausreichend, wenn der Transformationsalgorithmus einmalig für jede Zielsprache implementiert wird.

Das folgende Beispiel zeigt einen Auszug aus einer solchen Dialogbeschreibung:

```
<dialog name="product_edit" >
   <component name="price_edit" type="TcxSpinEdit" left="20" top="40"
            width="120" height="20">
      <parameter name="valuetype" value="float" />
      <parameter name="alignment" value="right" />
      <parameter name="displayformat" value="0.00 €" />
      <databinding entity="product" attribute="price">
   </component>
   ...
</dialog>
```

Der Dialog enthält eine TcxSpinEdit-Komponente, die positioniert und an das OOA-Modell „gebunden“ wurde. Die drei `<parameter>` wurden direkt aus dem Komponentenmodell übernommen und werden erst bei der Transformation dieses Dialogmodells in die Zielsprache benötigt.

5 Analyse des Nutzerverhaltens

Die Analyse des Nutzerverhaltens basiert auf einer Liste möglicher Entwurfs- und Designfehler. Für jeden Fehler sind Erkennungsstrategien und Korrekturmöglichkeiten definiert. Der Nutzer wird von der Applikation bei seiner Arbeit „beobachtet“, d.h. es wird erfasst, wann er welche Dialoge und Eingabekomponenten benutzt, welche Werte er einträgt und welche Änderungen er vornimmt. Des Weiteren wird protokolliert, wenn er die Onlinehilfe verwendet und wonach er dort sucht.

Die folgenden drei Beispiele sollen dies exemplarisch verdeutlichen:

1. Ein häufiger Designfehler aufgrund fehlender Fachkenntnisse ist, dass die Reihenfolge der Eingabekomponenten nicht mit dem Arbeitsprozess des Nutzers übereinstimmt. Dies lässt sich (vereinfacht) erkennen, wenn viele Nutzer eines Nutzertyps die Eingabekomponenten in einer anderen Reihenfolge anklicken und verwenden. Als Ergebnis wird ein Korrekturdatensatz angelegt, der in den nächsten Generierungszyklus einfließt und zu einer automatischen Umordnung der Eingabekomponenten in der Dialogschicht für eben diesen Nutzertyp führt. Ebenso lassen sich wenig oder gar nicht verwendete Komponenten erkennen und auf einen zusätzlichen erweiterten Dialog auslagern, was zu einer übersichtlicheren Applikation führt.

2. Ein weiterer häufiger Designfehler ist die Auswahl der falschen Eingabekomponente. Dieser Fehler lässt sich anhand der eingetragenen Werte erkennen. Werden beispielsweise in ein einzeiliges Textfeld immer nur dieselben fünf Werte eingetragen, dann ist der Einsatz einer (erweiterbaren) Auswahlliste sinnvoller, um dem Nutzer unnötige Tipparbeit zu ersparen. Auch hierfür wird ein Korrekturdatensatz erstellt und bei der nächsten Generierung automatisch berücksichtigt. Gleiches gilt für zu groß oder zu klein dimensionierte Textboxen, die anhand der durchschnittlichen und maximalen Textlänge korrigiert werden können.

3. Falsche oder nicht erwartungskonforme Navigationsmöglichkeiten in der Dialogsteuerung lassen sich erkennen, wenn viele Nutzer häufig einen Dialog öffnen und ihn kurz darauf

wieder schließen, weil es nicht der von ihnen erwartete Dialog ist. Hierfür wird dem Navigationsmodell ein entsprechender Vermerk hinzugefügt, dass eine manuelle Kontrolle der Dialogverknüpfungen notwendig ist. Eine weitere Verbesserungsmöglichkeit wird deutlich, wenn viele Nutzer immer zwei bestimmte Dialoge nacheinander benutzen. Dann scheinen diese beiden Dialoge fachlich zusammenzugehören und sollten im Navigationsmodell verknüpft oder unter Umständen sogar als neue Sicht vollständig zusammengefasst werden.

Dies sind drei von zahlreichen Szenarien, die sich erkennen und korrigieren lassen.

Die Erkennungsregeln bzw. –algorithmen zur Analyse des Nutzerverhaltens und Ableitung der Korrekturanweisungen sind Bestandteil des Generatorsystems. Sie werden einmalig definiert, können aber auch erweitert und durch neue ergänzt werden. Für eine konkrete Applikation können dann aus der Liste aller Erkennungsregeln diejenigen ausgewählt werden, die für den Anwendungskontext sinnvoll sind.

6 Fazit

Mit dem vorliegenden Ansatz lässt sich ein großer Teil des Entwicklungsaufwandes für Applikationen einsparen bei gleichzeitiger Erhöhung der Qualität des Endproduktes, weil Quellcode automatisch generiert wird und so auch programmiertechnische Fehlerquellen deutlich reduziert werden. Die Datenschicht lässt sich vollständig automatisch aus dem OOA-Modell generieren und benötigt keinerlei Nacharbeit. Nach der Modellierung der Sichten kann die Dialogschicht für jeden Nutzertyp entsprechend seiner Anforderungen generiert werden. Die differenzierte Nutzerverhaltensanalyse liefert für jeden Nutzertyp die vorhandenen Entwurfsfehler mit entsprechenden Korrekturanweisungen. Diese werden automatisch beim nächsten Generierungszyklus berücksichtigt und führen so zu einer deutlichen Verbesserung der Applikation ohne zusätzlichen Entwicklungsaufwand.

Der Einsatzschwerpunkt dieser Methode liegt bei datenorientierten Anwendungen, wie sie beispielsweise häufig im kaufmännischen Bereich anzutreffen sind. Dabei zeichnen sich die Dialoge durch zahlreiche Ein- und Ausgabefelder aus, mit denen bestimmte fachspezifische Objekte angelegt, geändert und verknüpft werden können. Für solche Ein- und Ausgabedialoge können sehr ansprechende Dialoge automatisch generiert werden (Kruschinski 1999). Im Vergleich zur klassischen Softwareentwicklung mit GUI-Designern ergibt sich zunächst ein zusätzlicher Aufwand für die Erstellung des Nutzertypen-, Komponenten- und Sichtenmodells. Dieser wird allerdings relativiert durch die Generierung von nutzertypisierten Dialogen und einen geringeren Testaufwand, weil generierter Quellcode keine oder zumindest deutlich weniger Fehler enthält. Insgesamt ergeben sich deutliche Aufwandseinsparungen bei Anwendungen mit vielen Dialogen, bei Wiederverwendung des Nutzertypen- oder des Komponentenmodells und bei Entwicklungsprozessen, bei denen in Zusammenarbeit mit dem Nutzer nach und nach verbesserte Prototypen bzw. Applikationen erstellt werden.

Ein erster Generator-Prototyp wurde für die Zielsprache Delphi implementiert und wird bereits teilweise für ein kommerzielles Projekt mit einem Industriepartner eingesetzt, um Erfahrungen und praxisnahe Analyseergebnisse zu erhalten.

Literaturverzeichnis

Ceri, S.; Fraternali, P.; Bongio, A. (2000): Web Modelling Language (WebML): A modelling language for designing Web sites. In: Proceedings WWW9 Conference, Amsterdam, May 2000

Hofmann, F. (1998): Grafische Benutzungsoberflächen – Generierung aus OOA-Modellen. Spektrum Akademischer Verlag Berlin, Heidelberg

Kruschinski, V. (1999): Layoutgestaltung grafischer Benutzungsoberflächen – Generierung aus OOA-Modellen. Spektrum Akademischer Verlag Berlin, Heidelberg

Niemann, C. (2000): Datenbankfähige Client/Server-Anwendungen – Generierung aus OOA-Modellen. Spektrum Akademischer Verlag Berlin, Heidelberg

Weisbecker, A. (1993): Regelbasierte Generierung software-ergonomischer Benutzungsschnittstellen aus Datenmodellen. In: Ziegler, J.; Ilg, R. (Hrsg.): Benutzergerechte Software-Gestaltung: Standards, Methoden und Werkzeuge. München, Wien: Oldenbourg Verlag, S. 171-188

Kontaktinformationen

Steffen Büder

Institut für Software- und Multimediatechnik
Fakultät Informatik
Technische Universität Dresden
Nöthnitzer Straße 46
01187 Dresden

Tel.: +49 (0) 160 7273381
E-Mail.: steffen@bueder.com
steffen.bueder@inf.tu-dresden.de