Modellgetriebene Entwicklung von grafischen Benutzerschnittstellen

Stefan Link, Thomas Schuster, Philip Hoyer, Sebastian Abeck

Institut für Telematik, Fakultät für Informatik
Universität Karlsruhe (TH)
76128 Karlsruhe
{ link | tschu | hoyer | abeck }@ cm-tm.uka.de

Abstract: Der Ansatz der modellgetriebenen Softwareentwicklung fokussiert die Entwicklung von Software durch deren Modellierung und einer anschließenden Transformation bis auf den Quellcode der gewünschten Zielplattform. Eine durchgehend werkzeugunterstützte Transformation der Modelle auf Quellcode ist unter anderem auf Grund ungenügender oder fehlender Metamodelle zur Beschreibung einiger Aspekte bislang nicht möglich. Diese Arbeit greift den Aspekt der Benutzerinteraktion heraus und zeigt, wie durch die Erweiterung einer Modellierungssprache auf Metamodellebene eine werkzeugunterstützte, modellgetriebene Entwicklung plattform-unabhängiger Benutzerschnittstellen ermöglicht wird. Eine Fallstudie demonstriert die Machbarkeit und den Mehrwert unserer Arbeit.

1 Einleitung

Die modellgetriebene Softwareentwicklung ordnet der Modellierung eines Softwaresystems einen zentralen Stellenwert zu. Die Software wird dabei zunächst durch Modelle spezifiziert und anschließend durch deren Transformation in den Quellcode der gewünschten Zielplattform überführt [KWB03]. Eine aktuelle Fragestellung der modellgetriebenen Ansätze liegt im Grad der möglichen Werkzeugunterstützung bei der Transformation vom Modell zum Quellcode [PM06]. Mit den vorhandenen Modellierungssprachen ist es zwar möglich, eine Vielzahl unterschiedlicher Aspekte auszudrücken, jedoch sind gerade im Bereich der Modellierung von grafischen Benutzerschnittstellen (engl. graphical user interface, GUI) die vorhandenen Modellierungselemente nicht ausreichend, um alle relevanten Informationen so zu erfassen, dass eine vollständig werkzeugunterstützte Transformation ermöglicht wird [Lo06].

Hier setzt diese Arbeit mit Hilfe einer Ausprägung der modellgetriebenen Ansätze, der Model-driven Architecture (MDA) [OMG03] und der in der MDA verwendeten grafischen Modellierungssprache Unified Modeling Language (UML) [OMG07, RBJ04] an. Als unseren Beitrag zeigen wir, wie es mit zwei Erweiterungen der UML möglich wird, eine GUI so zu spezifizieren, dass diese mit der Transformationssprache QVT (Queries Views Transformations) [OMG05] in einem mehrstufigen Transformationsprozess auf Quellcode einer beliebigen Zielplattform transformiert werden kann. Diese Mehrstufigkeit wird der Vielfalt der heute vorhandenen Endgeräte, wie beispielsweise PCs, PDAs oder Smartphones, gerecht.

Im Gegensatz zu bestehenden Arbeiten berücksichtigt unser Ansatz die Portabilität der GUI, sowie die verschiedenen Rollen, die am Softwareentwicklungsprozess beteiligt sind. Ziel unseres ersten Transformationsschritts ist ein plattformunabhängiges Modell zur abstrakten Beschreibung der GUI, welches wiederum von GUI-Experten in einem zweiten Transformationsschritt auf ein plattformspezifisches Modell und damit eine konkrete GUI für die jeweilige Zielplattform transformiert wird.

2 Stand der Technik

Das Vorgehen der modellgetriebenen Softwareentwicklung und im Speziellen das der MDA wird in anderen Arbeiten umfassend eingeführt [PM06, OMG03, KWB03]. Martínez-Ruiz et al. [MM+06] und Vanderdonckt [Va05] verfolgen ein modellgetriebenes Verfahren, das die GUI aus der Sicht der Nutzer entwickelt. Die Autoren orientieren sich am Cameleon Referenzmodell [CC+03] und modellieren eine GUI ausgehend von den durch die Benutzer auszuführenden Aufgaben. Zur Modellierung verwenden sie die domänenspezifische Sprache UsiXML [MM+06].

Pinheiro da Silva et al. [PP00, PP03] erweitern das Metamodell der UML um eigene Symbole und Diagrammtypen zu UML for Interactive Applications (UMLi), mit der die Modellierung von grafischen Benutzerschnittstellen ermöglicht wird. Almendros-Jimenez und Iribarne [AI04, AI05] hingegen untersuchen den Einsatz von reinen UML-Anwendungsfall- und Aktivitätsdiagrammen zur Entwicklung von GUIs, die auf Java Applets basieren. Lorenz [Lo06] schlägt zunächst eine Unterscheidung zwischen System- und Benutzeraktionen vor, um den Benutzeranteil klar herauszustellen. Diese Unterscheidung wird auch von Petrasch et al. [PM06] verfolgt und als Grundlage für unseren Ansatz übernommen. Im Gegensatz zu [AI05] führen [Lo06, PM06] eine direkte Transformation auf die Zielplattform J2EE/Struts durch.

3 Modellgetriebene Entwicklung von Benutzerschnittstellen

Im Folgenden stellen wir unseren Ansatz der modellgetriebenen Entwicklung von Benutzerschnittstellen auf Basis einer Erweiterung der UML-Aktivitätsdiagramme, sowie eines plattformunabhängigen Metamodells zur Beschreibung einer GUI vor.

3.1 Metamodelle zur Beschreibung von grafischen Benutzerschnittstellen

Ein UML-Aktivitätsdiagramm kann dazu verwendet werden, einen Systemanwendungsfall detailliert zu modellieren [Fo06] und dient im Sinne der MDA als plattformunabhängiges Modell (platform independent model, PIM). In Anlehnung an [Lo06], verfeinern wir in einem ersten Schritt das UML-Aktivitätsdiagramm durch eine Unterteilung der Aktionen in System- und Benutzeraktionen. Dadurch kann spezifiziert werden, wann Eingaben (Mausklick, Eingabe des Namens etc.) vom Benutzer zu erwarten sind. Die Herausforderung liegt in der Modellierung der für die GUI relevanten Informationen. Ist während einer Benutzeraktion die Eingabe eines Parameters *MyInput* erforderlich, so muss die dazu gehörende GUI ein entsprechendes Eingabefeld anbieten.

Ferner muss festgehalten werden, dass dieses die Bezeichnung *MyInput* trägt und welcher Typ (String,…) erwartet wird. UML-Aktivitätsdiagramme liefern derzeit keine, für eine werkzeugunterstützte Weiterverarbeitung adäquate Metaklasse, zur Spezifikation dieser Informationen. Zwar wird die Metaklasse *InputPin* [OMG06] angeboten, jedoch ist diese für unsere Zwecke zu allgemein gehalten, da diese die Spezifikation völlig beliebiger Datentypen zulässt.

Wir definieren daher ein UML-Profil *GUIActivityProfile* als leichtgewichtige Erweiterung des UML-Metamodells mit einem Stereotyp *GUIInputPin* als Erweiterung des InputPins, der nur GUI-relevante Datentypen (Abbildung 1) zulässt. Diese Einschränkung erreichen wir durch eine Aufzählung *GUIType*, die je nach Einsatzszenario nur geeignete Datentypen anbietet. Werden in einer Benutzeraktion mehrere Eingaben erwartet, so wird ein *GUIInputPin* je Eingabeparameter modelliert.

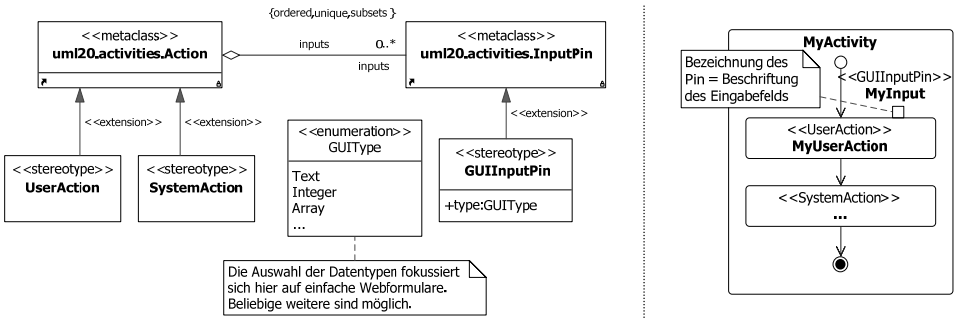

Abbildung 1: UML-Profil *GUIActivityProfile* und UML-Aktivitätsdiagramm mit *GUIInputPin*

Der *GUIInputPin* spezifiziert bewusst nicht den Typ des GUI-Elementes wie bspw. ein Eingabefeld, eine Auswahlbox, etc. mit dem der Eingabeparameter erfasst wird. Diese Information wird im UML-Aktivitätsdiagramm noch nicht benötigt und sollte im Sinne der Trennung von Zuständigkeiten (Seperation of Concerns) erst anschließend von einem GUI-Experten in einem eigenständigen Modell zur Beschreibung der GUI hinzugefügt werden.

Das zweite von uns erstellte UML-Profil *GUIProfile* ermöglicht die Erstellung eines solchen GUI-Modells. Da die Beschreibung von GUIs auf einer großen, aber endlichen Menge von GUI-Elementen basiert, lässt sich jede GUI als beliebig tiefe Verschachtelung von GUI-Elementen beschreiben [Bo06, SW06]. Das Profil basiert daher auf einem Querschnitt verschiedener, in aktuellen Grafik-Bibliotheken vorhandenen Elementen, wodurch die Plattformunabhängigkeit des Modells sichergestellt werden kann. Um das Metamodell erweiterbar zu halten wird das Entwurfsmuster Dekorierer eingesetzt. Neue Typen können durch Erweiterung des Typs *GUIElement* eingeführt werden. Ein zusätzlicher Erweiterungsmechanismus ist durch die Aufzählungstypen gegeben, welche die GUI-Elemente typisieren. Eine Instanz des Metamodells kann final auf eine beliebige konkrete Plattform transformiert werden. Abbildung 2 zeigt links einen kleinen Ausschnitt des erstellten UML-Profils zur Beschreibung einer GUI, rechts eine einfache Instanziierung.

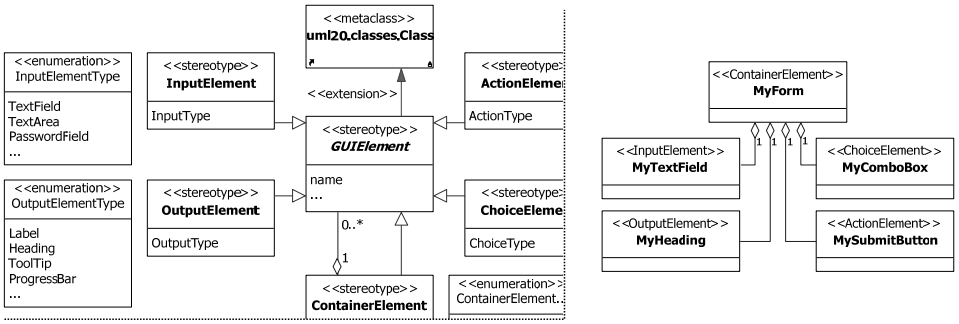

Abbildung 2: Ausschnitt aus UML-Profil *GUIProfile* und beispielhafte Instanziierung

Unser Vorgehen setzt sich somit zusammen aus der Erweiterung des UML-Aktivitätsdiagramms durch ein UML-Profil *GUIActivityProfile*, der Transformation der *GUIInputPins* auf ein GUI-Element und aus einem als GUI-Modell bezeichneten UML-Klassendiagramm. Das GUI-Modell selbst wird durch ein weiteres UML-Profil *GUIProfile* spezifiziert. Abschließend wird durch weitere Transformationen das GUI-Klassenmodell bis auf den Quellcode der gewünschten Zielplattform abgebildet.

3.2 Modellgetriebene GUI-Entwicklung am Beispiel von XForms

Zur praktischen Umsetzung unseres Vorgehens nutzen wir das Werkzeug „Together Architect 8.0“ von Borland. Als Beispiel dient ein UML-Aktivitätsdiagramm zu einer Aktivität *SignOn*, die eine Benutzeraktion *ProvideUserCredentials* beinhaltet. Unter Verwendung des UML-Profils *GUIActivityProfile* wird *ProvideUser-Credentials* mit zwei *GUIInputPins Username* und *Password* versehen und dann auf unser GUI-Modell transformiert. Hierbei werden die Bezeichner *Username* bzw. *Password* als Beschriftung des entsprechenden GUI-Elements verwendet. Die dazu gehörenden Datentypen *Text* und *Secret* der *GUIInputPins* werden auf das GUI-Element *InputElement* abgebildet. Abbildung 3 gibt einen Auszug der entsprechenden QVT-Ausdrücke wieder.

```
query GUIActivityProfile:GUIInputPin::
guiinputs2classes(): uml20::classes::Class {
  if (self.GUIType = 'String') then
    self.pin2inputelement()
                -- mapping to inputelement
  else if (…) then
    self…
```

```
mapping pin2inputelement(in pin: GUIActivityProfile::
GUIInput): GUIProfile::InputElement {
  name := pin.name;
  …
}
```

Abbildung 3: Zwei Beispiele für zur Transformation verwendeter QVT-Ausdrücke

Die erste Transformation bildet die *GUIInputPins* anhand des *guiType* auf die entsprechenden Modellelemente des *GUIProfile* ab. Das so erzeugte GUI-Modell beschreibt die Grundzüge der grafischen Benutzerschnittstelle und kann anschließend z.B. von GUI-Experten hinsichtlich Ergonomie und Usability optimiert werden. Die Weiterentwicklung sowohl des GUI- als auch des Prozessmodells kann somit völlig unabhängig voneinander durch unterschiedliche Rollen vorgenommen werden. Das Prozessmodell (das erweiterte UML-Aktivitätsdiagramm) dient zuvor als Ausgangsbasis und Schnittstelle zur Kommunikation zwischen diesen Rollen (siehe auch Abbildung 4).

Das erzeugte GUI-Modell überführen wir dann in einem zweiten Transformationsschritt in ein plattformspezifisches Modell (PSM) unserer Zielplattform, hier am Beispiel von XForms [Bo06]. Im letzten Schritt transformieren wir dieses plattformspezifische Modell auf plattformspezifischen Code (PSC). Abbildung 4 zeigt alle Transformation und das entsprechende Ergebnis im Zusammenhang.

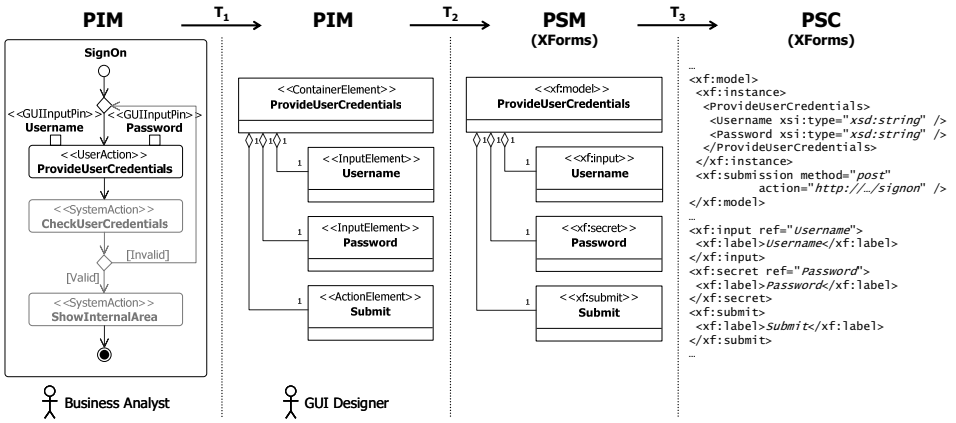

Abbildung 4: Modellbasierte Entwicklung von GUIs am Beispiel

Bei der Wahl der Plattform haben wir uns für XForms entschieden, um den Gedanken der Unabhängigkeit der GUI über die Plattform hinaus bis auf die Ebene der Endgeräte weiterzuführen, da XForms Konzepte zur geräteunabhängigen Darstellung von Inhalten mit sich bringt [Bo06]. Wird im oben erwähnten Beispiel das Eingabeattribut *Username* auf ein XForms-Eingabeelement abgebildet, bleibt die finale Darstellung dieses Eingabefelds dem Endgerät überlassen.

Durch einfache Anpassung der in QVT verfassten Transformationsregeln sind weitere Abbildungen von dem einmal erstellten GUI-Modell (plattformunabhängiges Modell) auf viele weitere Zielplattformen wie bspw. J2EE/Struts etc. möglich.

4 Zusammenfassung und Ausblick

Wir konnten in dieser Arbeit zeigen, dass - aufbauend auf den Konzepten der MDA und bestehenden Arbeiten - eine modellgetriebene Entwicklung von grafischen Benutzerschnittstellen möglich ist. Der Fokus lag insbesondere auf der Plattformunabhängigkeit der Modelle sowie auf der Automatisierbarkeit der Transformationsschritte durch Werkzeugunterstützung. Weiter konnten wir zeigen, dass durch unseren Ansatz Abbildungen auf beliebige Zielplattformen durch Anpassung der Transformationen möglich werden. Diesem Ansatz folgend kann somit ein weiterer Aspekt des modellgetriebenen Softwareentwicklungsprozesses werkzeugunterstützt und damit automatisiert werden.

Unsere nächsten Schritte werden sich zum Einen mit der Erweiterung des Metamodells zur Beschreibung einer GUI befassen, um hier Verbesserungen der Ergonomie der GUI herbeizuführen. Zum Anderen werden wir als längerfristigen Schritt die Integration weiterer Aspekte der GUI, wie etwa die Modellierung von Layout und Navigation angehen, um benutzeraktionsübergreifende GUIs zu ermöglichen. Erste vielversprechende Ansätze hierzu sind in [Ko06] zu finden.

Literaturverzeichnis

[AI04] Almendros-Jimenez, J.M.; Iribarne, L.: Describing use cases with activity charts. In: Proc. Metainformatics Symposium (MIS'04). Springer Verlag, 2004; S. 141-159.

[AI05] Almendros-Jimenez, J.M.; Iribarne, L.: Designing GUI components from UML Use Cases. In: Proc. 12th IEEE Int. Conf. and Workshop on the Engineering of Computer Based Systems (ECBS'05). IEEE Computer Society Press, 2005; S. 210-217.

[Bo06] Boyer, J.M.: XForms 1.1, W3C Working Draft, http://www.w3.org/TR/2007/WD-xforms11-20070222. W3C, 2006.

[CC+03] Calvary, G., Coutaz, J. et al.: The CAMELEON Reference Framework, http://giove.cnuce.cnr.it/cameleon/deliverable1_1.html, Januar 2003

[Fo06] Forbrig, P.: Objektorientierte Softwareentwicklung mit UML. Hanser Fachbuchverlag, 2006

[Ko06] Koch, N.: Transformation Techniques in the Model-Driven Development Process of UWE. In: Proc. 6th Int. Conf. on Web engineering (ICWE'06). ACM Press, 2006.

[KWB03] Kleppe, A.; Warmer, J.; Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley, 2003.

[Lo06] Lorenz, A.: Anpassung von UML-Aktivitäten an den Prozess der Webapplikationsentwicklung. In: Proc. 36. Jahrestagung der Gesellschaft für Informatik (INFORMATIK 2006). Bonner Köllen Verlag, 2006; S. 178-184.

[MM+06] Martínez-Ruiz, J., Muñoz Arteaga, J. et al.: A First Draft of a Model-driven Method for Designing Graphical User Interfaces of Rich Internet Applications, Latin American Web Congress, IEEE Computer Society Press, Oktober 2006

[OMG03] Mukerji, J.; Miller, J.: MDA Guide Version 1.0.1, http://www.omg.org/cgi-bin/doc?omg/03-06-01. OMG, 2003.

[OMG05] Object Management Group: Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) Specification, Final Adopted Specification, http://www.omg.org/cgi-bin/doc?ptc/2005-11-01. OMG, 2005.

[OMG07] Object Management Group: Unified Modeling Language (UML), Version 2.1.1: Superstructure, http://www.omg.org/cgi-bin/doc?formal/07-02-03. OMG, 2007.

[PM06] Petrasch, R.; Meimberg, O.: Model Driven Architecture – Eine praxisorientierte Einführung in die MDA. dpunkt Verlag, 2006.

[PP00] Pinheiro da Silva, P.; Paton, N. W.: User Interface Modelling with UML; Proc. 10th European-Japanese Conference on Information Modelling and Knowledge Bases, Saariselk Finnland, 2000.

[PP03] Pinheiro da Silva, P.; Paton, N. W.: Improving UML Support for User Interface Design: A Metric Assessment of UMLi. In: Proc. of ICSE'03 Workshop, 2003.

[Va05] Vanderdonckt, J.: A MDA-Compliant Environment for Developing User Interfaces of Information Systems, Advanced Information Systems Engineering CAiSE, Porto, Springer-Verlag, Berlin, Juni 2005

[RBJ04] Rumbaugh, J.; Booch, G.; Jacobsen, I.: The Unified Modeling Language Reference Manual, Second Edition. Addison-Wesley, 2004.

[SW06] Sells, C.; Weinhardt, M.: Windows Forms 2.0 Programming, Addison Wesley, 2006.