Roland Petrasch

Modellbasiertes User Interface Design und HCI Patterns: Eine Symbiose

Der modellbasierte Entwicklungsansatz trennt die fachliche Essenz (Anwendersicht) von den technischen Details (Entwicklersicht) und lässt sich bei interaktiven Systemen vorteilhaft nutzen. Allerdings sind beim Thema modellbasierte Entwicklung von Benutzeroberflächen noch viele Fragen offen, von denen einige hier beleuchtet werden. Vieles spricht dafür, dass erst in Verbindung mit HCI-Patterns eine intensivere Nutzung der modellbasierten Software-Entwicklung im HCI-Bereich stattfindet – quasi im besten Sinne eines symbiotischen Effektes.

1. Einführung

Die Entwicklung interaktiver Systeme ist zwar ein Dauerthema, hat jedoch durch verschiedene Faktoren erneut an Bedeutung gewonnen – u. a. durch die breite Nutzung mobiler Geräte oder die weitere Verbreitung von Web-Applikationen. Die Entwicklungsproduktivität hat mit den Trends jedoch nur zum Teil mithalten können: Nach wie vor sind viele Benutzeroberflächen das Ergebnis aufwändiger manueller Arbeit – User Interface Design (UID) ist geprägt vom Handwerk, welches sich quasi im Mittelalter befindet und im Zeitalter einer industriellen Produktion noch nicht ganz angekommen ist, wobei es mittlerweile durchaus bemerkenswerte Techniken und Werkzeuge gibt, die zwar praxistauglich aber oftmals für eine bestimmte Technologie spezifisch sind, z. B. das Scaffolding bei Ruby on Rails. Modellbasierte Entwicklung und Code-Generierung kann und wird eine Schlüsselfunktion beim Thema Produktivität spielen.

Im Folgenden sei jedoch der zentral wichtige Aspekt der Usability (Gebrauchstauglichkeit) fokussiert: Das Ziel muss es sein, vor und während der Entwicklung des interaktiven Systems sicherzustellen, dass softwareergonomische Anforderungen (z. B. gem. ISO 9142) erfüllt werden und nicht erst nach der Fertigstellung der Software – getreu dem Motto: Fehlervermeidung ist besser als Fehlerbehebung. Prototyping als einzige konstruktive Maßnahme kann nicht die systematische und fundierte Konstruktion der Oberfläche sicherstellen (die Gefahr, in ein Trialand-Error-Vorgehen zu verfallen, ist ansonsten zu groß). Aber wie können die Versprechungen der Informatik (Wiederverwendbarkeit, Wartbarkeit, Portabilität etc.) eingehalten, die Anforderungen des Managements (Produktivität, Termintreue, Automatisierung) erfüllt und der Erwartung der User (Usability, Ergonomie) Rechnung getragen werden? Um es gleich vorweg zu nehmen: Diesen gordischen Knoten, der sich auch als „chronische Software-Krise" (Gibbs, 1994) offenbart, kann natürlich auch dieser Beitrag nicht lösen, aber zumindest bei einer Standortbestimmung und der Erkennung notwendiger weiterer Schritte helfen. Dabei sei zunächst die modellbasierte UI-Entwicklung vorgestellt, um anschließend die Verwendung formal beschriebener HCI-Patterns zu ermöglichen.

2. Modellbasierte Entwicklung interaktiver Systeme

Die klassische Entwicklung interaktiver Systeme sieht zwar auch Konzepte in Form von Modellen und Diagrammen vor, z. B. Affinity Diagram (Beyer, Holtzblatt, 1997), diese sind jedoch oftmals nicht so formal spezifiziert, als dass sie sich für die Code-Generierung verwenden ließen. Die modellgetriebene oder modellbasierte Software-Entwicklung hingegen nutzt formale Modelle für anschließende Transformationen, z. B. Model-to-Model (M2M) bzw. Model-to-Text (M2T). Die Code-Generierung ist quasi ein Spezialfall von M2T. Die Standardisierung ist u. a. von der OMG (Object Management Group) voran getrieben worden: Die sog. Model Driven Architecture (MDA) (OMG, 2003) und entsprechende Spezifikationen, z. B. MOF (OMG, 2006), QVT (OMG, 2007) oder UML (OMG, 2010) sind seit geraumer Zeit verfügbar.

Model Based User Interface Design (MBUID) ist im Sinne der MDA ist vereinfacht ausgedrückt die Erstellung plattformunabhängiger Modelle des interaktiven Systems (Computation bzw. Platform Independent Model, CIM / PIM) und die Transformation zu plattformspezifischen Artefakten (Platform Specific Model, PSM). Der Code, d. h. das lauffähige interaktive System (Platform Specific Implementation, PSI) lässt sich dann zum großen Teil generieren – zumindest ist das u. a. das Ziel des MDA-Ansatzes (Petrasch, 2006).

Ein PIM i. S. von MBUID stellt eine Abstraktion von einem UI-Basissystem dar (z. B. GNOME, Google Android, Java AWT/Swing). Für solche abstrakten Beschreibungen gibt es seit geraumer Zeit Sprachen bzw. Notationsformen, z. B.

DOI 10.1524/icom.2010.0023

User Action Notation (Hartson, 1990), die sich jedoch nicht durchsetzten. In jüngerer Zeit wurden Sprachen, die den UML-Profilmechanismus (leichtgewichtiger Ansatz) verwenden, z. B. GUILayout (Blankenhorn, 2004), Wisdom (Nunes, 2001), UWE (Koch, 2001) oder schwergewichtige Verfahren mit eigenen Metamodellen, z. B. UMLi (Pinheiro da Silva, Paton, 2003), entwickelt. Auch XML-basierte Sprachen wie UIML (Phanouriou, 2000) oder webML (Ceri et al., 2003) sind verfügbar, wobei sich bei derartigen textuellen Notationsformen die Frage nach der Menschenlesbarkeit stellt. Auch übergreifende Ansätze sind vorhanden, z. B. wie die strukturierten Aktivitätsdiagramme für Aufgabenmodelle zeigen (Forbrig, 2006). Zumindest theoretisch gibt es beim MBUID also eine gewisse Auswahl, auch wenn die Ansätze teilweise noch Gegenstand der Forschung bzw. im Entwicklungsstadium sind.

Beispielhaft sei die Modellierungsebene für MBUID anhand zweier Dialoge aufgezeigt: Ausgehend von einem Bearbeitungsdialog („Student bearbeiten“) für einen Studenten erfolgt per OK-Button der Wunsch, den Dialog zu verlassen. Diese navigationsrelevante Interaktion umfasst allerdings auch die Bestätigung von Datenänderungen, für die es in diesem Fall ein Feedback geben soll, der als separater Dialog („Save Student“) realisiert ist (s. Bild 1). Bei diesem Negativbeispiel soll zunächst kein HCI-Pattern zum Einsatz kommen. Abgesehen von der sprachlichen Inhomogenität durch englische und deutsche Beschriftungen sind beim Feedback-Dialog diverse Entscheidungen zu treffen, z. B. welche Interaktionen möglich sind (Ja/Nein, Ja/Nein/Abbruch) und wie diese beschriftet sein sollen („Sichern“, „Ja“, „OK“ etc.). Auch muss das Icon und die Anordnung der Elemente bestimmt und die Frage geklärt werden, ob es ein Systemmenü geben soll. Bei dem o. g. Feedback-Dialog sind zahlreiche Fehler vorhanden, z. B. nur zwei Navigationsalternativen („Save“ und „Cancel“), die sich negativ softwareergonomisch auswirken, z. B. Mängel bei der Steuerbarkeit, Erwartungskonformität und Selbsterklärbarkeit.

Entscheidungen und damit verbundene Fehler wie bei dem o. g. Beispiel sind potenziell bei jedem Feedback-Dialog während der Anwendungsentwicklung denkbar, möglichst aber zu vermeiden: Im Sinne der Fehlervermeidung und um Code-Generierung nutzen zu können, soll im Folgenden der modellbasierte Ansatz näher erläutert werden. Wegen der leichten Anwendbarkeit und Verständlichkeit sei hier ein UML-Profil verwendet (s. Bild 2), welches drei selbsterklärende Stereotypen für die Modellierung interaktiver Systeme definiert: <<Dialog>>, <<Interaction>>,<<Interactionelement>>. Dabei sind die Stereotypen im Kontext verschiedener UML-Diagrammtypen einsetzbar, z. B. lässt sich ein Dialog sowohl als Klassen im Klassendiagramm als auch als Zustand in der Zustandsmaschine (State-Machine) modellieren.

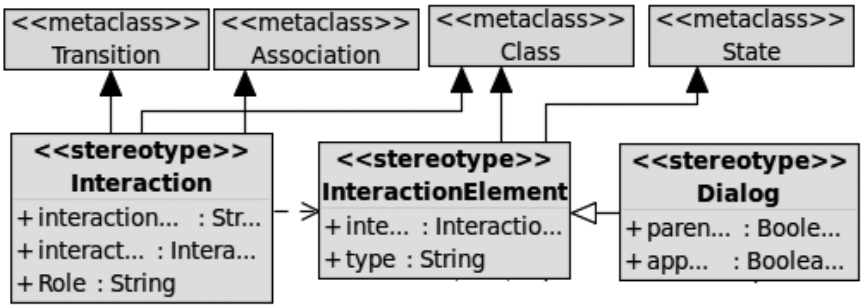


Bild 2: Ausschnitt auf dem UML-Profil für das Model-Based User Interface Design (MBUID)

Nun sei eine modellhafte Darstellung der o. g. Applikation vorgestellt: Das Modell des User Interface als PIM (plattformunabhängiges Modell) besteht aus den zwei Dialogen: Eine Liste (StudentList) und ein Editierdialog (StudentEditor). Wurde ein Editierdialog geöffnet, kann durch eine Interaktion (OK) wieder zur Übersicht (Liste der Studenten) zurück navigiert werden (s. Bild 3). Die Interaktion erfolgt mit einem Interaktionselement (OK-Button). Die anderen Teile des UI-Modells, z. B. wie von der Liste zum Editierdialog navigiert wird und der Feedback-Dialog (s. Bild 1), sind der Einfachheit halber nicht dargestellt.

Aus derartigen UI-Modellen lässt sich leicht entsprechender Code generieren. Es wird aber selbst bei kleineren Modellen deutlich, dass zumindest redaktionell die UI-Modelle (zu) umfangreich und damit unübersichtlich werden können. Es gäbe zwar die Möglichkeit einer kompakteren Darstellung, aber es bleibt dennoch die Frage offen, ob sich nicht generell Modellierungsaufwand einsparen lässt. Die Lösung für das Problem sind HCI-Patterns. Bevor auf deren Einsatz näher eingegangen wird, sei jedoch zunächst auf verschiedene weitere offene Punkte eingegangen, die der modellbasierte Ansatz im Allgemeinen und MBUID im Speziellen aufwerfen. Insbesondere im Praxiseinsatz können sie sich als Herausforderung darstellen:

Bild 1: Negativbeispiel: Dialog für die Bearbeitung von Studentendaten und Feedback-Dialog für das Speichern ohne HCI-Pattern-Anwendung

1. Abstraktion: UI-Modelle (bzw. deren Beschreibungssprachen) bewegen sich zuweilen auf einem sehr hohen Abstraktionsniveau, welches von der Anschaulichkeit und Nachvollziehbarkeit der konkreten Benutzeroberfläche weit „entfernt" ist (wie beispielsweise bei XForms (W3C, 2009), wo es ein sog. Trigger-Element gibt, das sich dann bei der Implementation als Push-Button, Hypertext-Link oder Sprachkommando darstellen kann). Andererseits gibt es auch sehr konkrete Modellierungssprachen, z. B. im Falle des Stereotyps <<button>> in UWE. Die Frage nach der „richtigen" Abstraktion muss jedes Projekt selbst beantworten.

2. Konformität zu Standards: Die Nutzung standardisierter Verfahren und Sprachen wie beispielsweise QVT oder M2T (OMG, 2008) für die Modelltransformation und Code-Generierung, können zur Unabhängigkeit von speziellen Produkten bzw. Herstellern sowie besseren Wiederverwendbarkeit beitragen, sind jedoch zuweilen nicht so flexibel bzw. spezifisch. Schwergewichtige Ansätze mit eigenem Metamodell und einer proprietären Transformationssprache für das MBUID folgen nicht immer Normen und Standards. Die Konsequenzen sollten vorab abgewogen werden.

3. Festlegung des Layouts: Je abstrakter die UI-Modelle sind, desto weniger lassen sich Festlegungen für konkrete Benutzeroberflächen machen – das liegt in der Natur der Sache. „Irgendwann" muss jedoch das User Interface gestaltet bzw. programmiert werden, so dass sich die Frage stellt, ob dies automatisiert, semiautomatisiert oder gar manuell erfolgen kann.

Besonders beim letzten Punkt gehen die Meinungen weit auseinander. Für ein recht simples User Interface kann der Automatisierungsgrad sehr hoch sein, bei komplexen Teilbereichen einer Benutzeroberfläche zeigt sich jedoch, dass die vollautomatische Code-Generierung auf der Basis hochgradig abstrakter Modelle an (ergonomische) Grenzen stößt, so muss man beispielsweise bei umfangreichen Menüstrukturen ein Usability-Engineer zuweilen neue oder unkonventionelle Wege beschreiten, um die Usability sicherzustellen; ein Code-Generator kann eine derartige Kreativität kaum an den

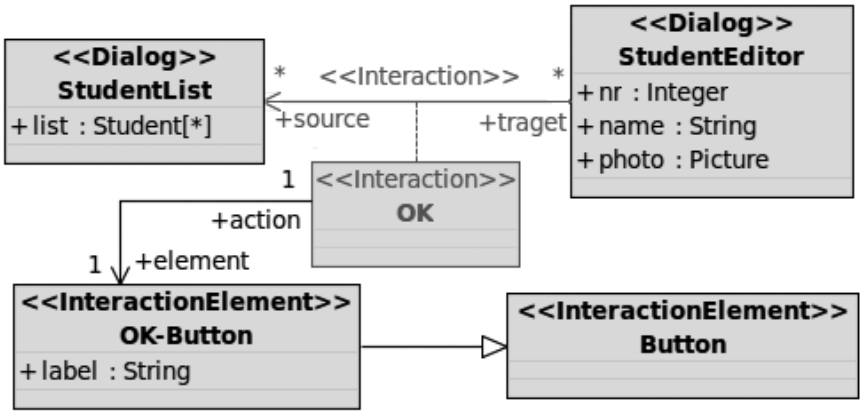


Bild 3: Anwendung des MBUID-Profils: Abstraktes, plattformunabhängiges Modell (PIM) des User Interface

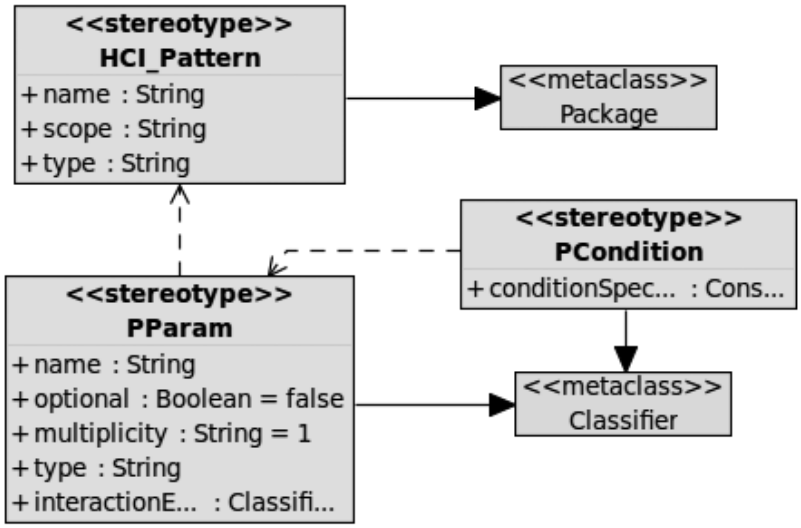


Bild 4: Das UML-Profil für HCI-Patterns (Ausschnitt)

Tag legen. Daher müssen oftmals Teile eines generierten Software-Systems noch manuell ergänzt bzw. geändert werden – zumindest zur Zeit noch.

Allerdings lassen sich einige Problempunkte (Wiederverwendung, Modellierungsaufwand) beim Thema MBUID durch HCI-Patterns recht gut in den Griff bekommen, was das folgende Kapitel aufzeigt.

3. HCI-Patterns im Kontext modellbasierter Benutzeroberflächen

Es existiert eine Vielzahl von HCI- oder Usability-Patterns (HCI Patterns Website, 2010) und Pattern-Languages (Bradac, Fletcher, 1998), (Graham, 2003), (van Welie, van der Veer, 2003). Obwohl es bei dem Thema durchaus noch viele offene Punkte gibt, sind HCI-Patterns wie auch Design-Patterns bei Experten wegen der Vorteile, z. B. Wissenstransfer (Borchers, 2000), akzeptiert und in gewisser Weise verbreitet. Da liegt es durchaus nahe, HCI-Patterns mit der modellbasierten bzw. modellgetriebenen Software-Entwicklung zu verbinden und somit wiederverwendbare Lösungen für immer wiederkehrende HCI-Probleme zu nutzen (Petrasch, 2007).

Auch hier ist zunächst ein UML-Profil zu definieren, damit sich HCI-Patterns modellieren lassen: Der Stereotyp <<HCI_Pattern>> kommt für das Package zu Einsatz, in dem sich das Muster befindet. Weiterhin kann ein Pattern parametrisiert sein und es können Bedingungen festgelegt werden (s. Bild 4).

Mit den beiden UML-Profilen (MBUID- und HCI-Pattern-Profil) sind nun Patterns

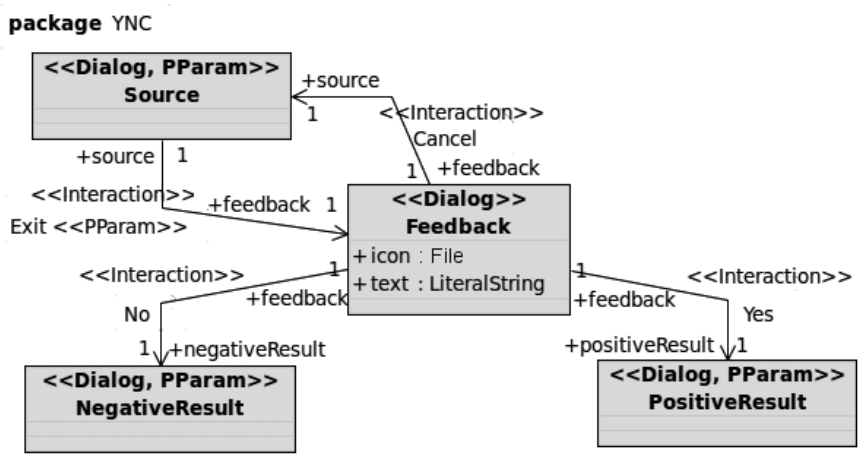


Bild 5: Das Feedback-/YNC-Pattern als Klassenmodell

formal beschreibbar. Bild 5 zeigt das Feedback- oder YNC-Pattern (Yes-No-Cancel), welches den typischen Ablauf modelliert, der sich ergibt, wenn Benutzer entscheiden müssen, ob eine Aktion positiv oder negativ beendet bzw. abgebrochen werden soll. Grundlage dafür ist die in Bild 1 vorgestellte Navigation mit Feedback, die es nun zu verbessern und zu standardisieren gilt.

Die Anwendung des Patterns ist denkbar einfach: In das UI-Modell lässt sich eine Referenz auf das HCI-Pattern, in diesem Fall das YNC-Pattern, einfügen (ähnlich einer Instanz) und mit den anderen UI-Elementen „verdrahten": Mit dem Yes- und dem No-Button gelangt der User wieder zur Übersichtsliste und mit Cancel zum Editierdialog zurück (s. Bild 6). Durch eine Kompaktnotation (hier nicht dargestellt) lässt sich das Pattern minimalinvasiv in verschiedene Modelle einfügen und damit wiederverwenden. Vorteile wie z. B. die Erfüllung der software-ergonomischen Anforderung Erwartungskonformität gem. ISO 9241 durch standardisierte Interaktionsmechanismen wie sie das YNC-Pattern bewirkt, lassen sich so direkt nutzen.

Nach der Pattern-Anwendung lässt sich der Code für die Benutzeroberfläche generieren, wobei dann der Feedback-Dialog mit den notwendigen Einstellungen (modaler Dialog, Beschriftung der Buttons, kein Systemmenü etc.) vorliegt, ohne dass ein Entwickler manuell eingreifen oder programmieren musste (s. Bild 7). Viele offene Punkte und Fehler im Feedback-Dialog (s. Bild) sind damit vermeidbar. So gibt es z. B. nun drei Navigationsalternativen („Yes", „No" und „Cancel"), die zur besseren Usability hinsichtlich der Steuerbarkeit und der Erwartungskonformität beitragen.

Wie vielfältig die Welt der HCI-Patterns ist, sei an dem Beispiel des EditAndRO-Patterns gezeigt, welches für ein Interaktionselement bzw. einen Dialog zwei Modi anbietet: Read-only (Content wird nur angezeigt) und Edit (Content ist modifizierbar). Als Modellierungsform ist hier die State-Machine eine geeignete Möglichkeit. Das MBUID-Profil lässt diese Modellierungsvariante auch durchaus zu (s. Bild 6).

Zwar hat die Modellierung mit Hilfe von Zuständen Vorteile, z. B. ist die Semantik in Hinblick auf das Verhalten des Dialogs gut geeignet, aber auch wohl bekannte Nachteile, z. B. das Problem der Vererbung. Derartige Aspekte sind vor dem Einsatz modellbasierter Verfahren zu berücksichtigen.

4. Fazit

Der Ansatz, HCI-Patterns mit modellbasierter Software-Entwicklung zu verbinden und damit eine Symbiose zu bewirken, ist zwar vielversprechend, muss aber noch einige Entwicklungsstadien durchlaufen, um in der Praxis anzukommen. So ist die Notationsform für UI-Modelle im Allgemeinen und für HCI-Patterns im Speziellen näher zu untersuchen. Vieles spricht dafür, dass zunächst eine gewisse Vielfalt und Offenheit dazu beiträgt, dass die diversen Ansätze und Modellierungsvarianten erprobt und kritisch analysiert werden können. Die Tool-Infrastruktur und die Standards sind jedenfalls vorhanden, z. B. mit Eclipse-basierten Implementationen der UML und der Transformationssprachen QVT und MTL (OMG, 2008). Eine wichtige „Baustelle" von MBUID auf dem Weg in die praktische Anwendung ist ein geeigneter Entwicklungsprozess, der sich wie in (Petrasch, 2009) angedeutet an den Human Centered Design Process der ISO 13407 anlehnen könnte.

Beim Thema HCI-Patterns ist allerdings zu beachten, dass diese kein Selbstzweck

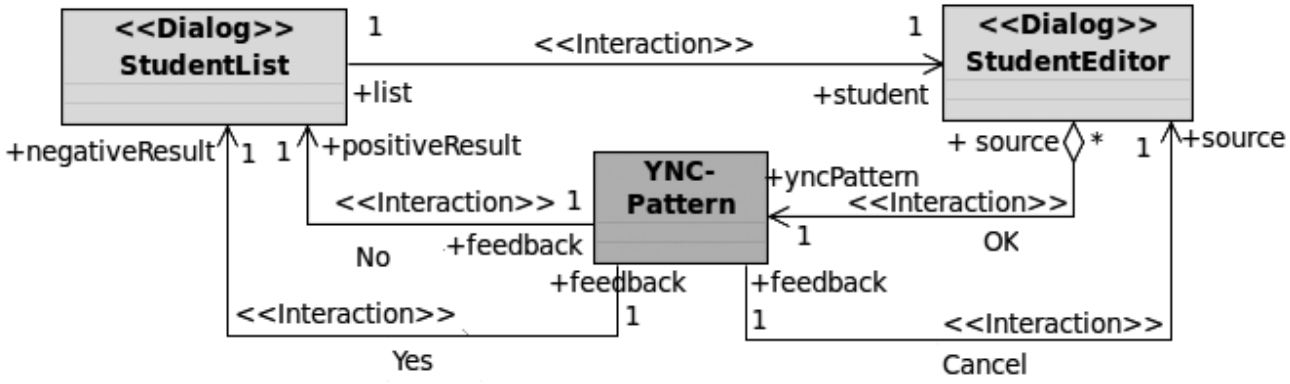


Bild 6: Die Anwendung des Feedback-/YNC-Pattern auf das UI-Modell (PIM)

sind: Sie müssen Prinzipien des Usability-Engineerings aufgreifen und zur Erfüllung von Usability-Anforderungen beitragen, z. B. Recognise Not Recall, und daher ihre Existenzberechtigung unter Beweis stellen. Autoren von HCI-Patterns sollten daher diese Verbindung darlegen, was teilweise auch der Fall ist, z. B. bei den HCI-Patterns für die Anwendungsdomäne Computerspiel (Folmer, 2006). Für die textuelle Beschreibung, die neben dem formalen Modell notwendig ist, eignen sich Sprachen wie z. B. die Pattern Language Markup Language (PLML) (Fincher, 2003). Neben den typischen Bereichen, z. B. pattern name, problems, context etc. muss es dann aus Sicht von MBUID einen Teil model(s) sowie Verweise auf die UML-Profile, Metamodelle oder andere Sprachdefinitionen für die Pattern-Modellierung geben.

Entscheidend ist, dass die Beteiligten den Ansatz akzeptieren und u. a. mit dem teilweise recht hohen Formalisierungs- und Abstraktionsgrad umgehen können. Ein User Interface Designer, ein Usability Engineer oder ein Informationsarchitekt darf nicht ein Eindruck bekommen, die Kreativität bei der Gestaltung von Benutzeroberflächen wird durch MBUID „ausgeschaltet". Nur wenn die menschlichen Faktoren berücksichtigt werden, lassen sich MBUID und HCI-Patterns symbiotisch zum Vorteil der Anwender nutzen.

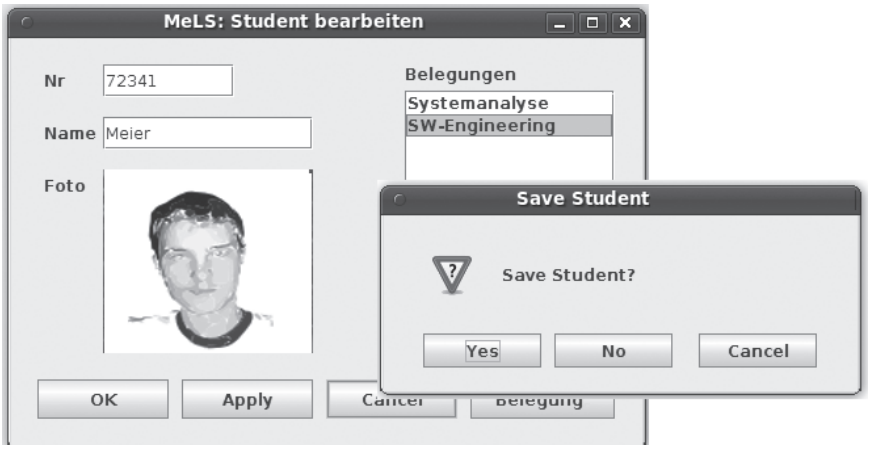


Bild 7: Positivbeispiel: Dialog für die Bearbeitung von Studentendaten und Feedback-Dialog für das Speichern

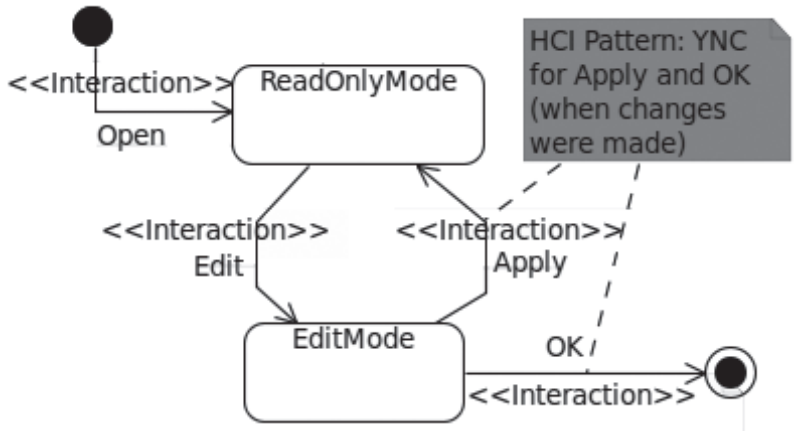
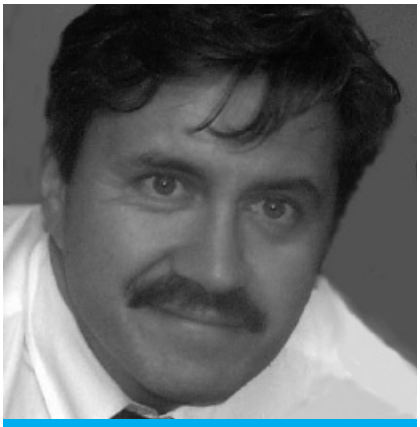


Bild 8: Das EditAndRO-Pattern als State-Machine

Literatur

- Beyer, H; Holtzblatt, K.: Contextual Design. Morgan Kaufmann Publishers, 1997.
- Blankenhorn, K.: A UML Profile for GUI Layout. University of Applied Sciences, Furtwangen, Diplomarbeit, 2004.
- Borchers, J.: Interaction Design Patterns: Twelve Theses, Position Paper. Workshop "Pattern Languages for Interaction Design". Proceedings CHI 2000. April 2000.
- Bradac, M.; Fletcher, B.: A Pattern Language for Developing Form Style Windows. In: Martin, R.; Riehle, D.; Buschmann, F. (Hrsg.): Pattern Languages of Program Design. Addison-Wesley Longman, Reading, MA, 1998, S. 347–357.
- Ceri, S.; Fraternali, P.; Bongio, A.; Brambilla, M.; Comai, S.; Matera, M.: Designing Data-Intensive Web Applications. The Morgan Kaufmann Series in Data Management Systems, 2003.
- Fincher, S.: Perspective on HCI Patterns: Concepts and tools (introducing PLML). Interfaces (56), British HCI Group, CHI 2003 Workshop Report, 2003, S. 27–28.
- Folmer, E.: Usability Patterns in Games. Futureplay 2006 Conference, London, Ontario, Canada, 3. Oct. 2006.
- Forbrig, P.; Reichert, D.: Modellbasierte Entwicklung von modellbasierten Werkzeugen. In: Fieber, F.; Neuhaus, W.; Petrasch, R. (Hrsg.): Werkzeuge und Anwendungsgebiete der modellbasierten Software-Entwicklung. Tagungsband zum 1. Workshop der SIG MDSE, Logos-Verlag Berlin, 2006.
- Gibbs, W. W.: Software's Chronic Crisis. Scientific American; September 1994, S. 86.
- Graham, I.: A pattern language for web usability. Addison-Wesley, 2003.
- Hartson, H. R.; Siochi, A. C.; Hix, D.: The UAN: A User-Oriented Representation for Direct Manipulation Interface Designs. ACM Transactions on Information Systems, Vol. 8, No. 3, July 1990, S. 181–203.

- HCI Patterns Website: http://www.hci-patterns.org, Abruf am 1. Juni 2010.
- ISO/DIS 13407 Human-centred design processes for interactive systems. ISO, 1999.
- ISO 9241–110:2006: Ergonomics of human-system interaction -- Part 110: Dialogue principles. ISO, 2006.
- Koch, N.: Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modeling Techniques and Development Process. Diss., Ludwig-Maximilians-University Munich, 2001.
- Object Management Group (OMG): MDA Guide. Version 1.0.1, June 2003.
- Object Management Group (OMG): MOF 2.0 – Meta Object Facility Core Specification. Vers. 2.0, Jan. 2006.
- Object Management Group (OMG): Meta Object Facility (MOF) 2.0 Query/View/Transformation, Vers. 2.0, July 2007.
- Object Management Group (OMG): MOF Model to Text Transformation Language, Vers. 1.0, Jan. 2008.
- Object Management Group (OMG): Unified Modeling Language, Infrastructure / Superstructure. Vers. 2.3, May 2010.
- Nunes, N. J.: Object Modeling for User-Centered Development and User Interface Design: The Wisdom Approach, Universidade da Madeira, Diss., 2001.
- Petrasch, R.; Meimberg, O.: Model Driven Architecture. Eine praxisorientierte Einführung in die MDA. dpunkt Verlag, 2006.
- Petrasch, R.: Model Based User Interface Design: Mo¬del Dri¬ven Architecture und HCI Patterns. In: GI Software¬technik-Trends, Gesellschaft für Informatik e.V., Band 27, Heft 3, 2007, S. 5–10.
- Petrasch, R.; Bureck, M.: HCI Patterns in the Context of Model Driven Development for interactive Systems. In: Petrasch R. et al.: Model Driven Software Engineering – Transformations and Tools. Tagungsband zum Workshop der SIG MDSE, Logos Verlag, 2009, S. 61–76.
- Phanouriou, C.: UIML: A Device-Independent User Interface Markup Language. Diss., Virginia Polytechnic Institute and State University, USA, 2000.
- Pinheiro da Silva, P.; Paton, N. W.: User Interface Modeling in UMLi. IEEE Software, July/Aug. 2003, S. 62–69.
- van Welie, M.; van der Veer, G. C.: Pattern Languages. In: Rauterberg, M.; Menozzi, M.; Wesson, J. (Hrsg.): Interaction Design: Structure and Organization, INTERACT 2003, IOS Press, Zürich, Switzerland, September 2003.
- W3C: XForms. W3C-standard, Vers 1.1, 2009.



Roland Petrasch

hat an der Universität Potsdam in Informatik zum Thema Software-Qualitätsmanagement promoviert. Er hat über 20 Jahre Berufserfahrung in der Software-Branche und hat als Entwickler, Berater und Projektmanager unter anderem bei der Lufthansa Informationstechnik und Software GmbH und der Nixdorf Computer AG gearbeitet. Außerdem lehrt und forscht er an der Beuth Hochschule für Technik Berlin, im Bereich Software-Engineering. Er ist u.a. Autor der Bücher „Einführung in das Software-Qualitätsmanagement“ und „Model-Driven Architecture – Eine praxisnahe Einführung“ sowie zahlreicher weiterer Fachbeiträge u.a. zum Thema Usability-Engineering.

E-Mail:
petrasch@beuth-hochschule.de