Roland Petrasch

Modellbasierte Entwicklung von Web 2.0-Anwendungen mit MDA™

Model-Based Development of Web 2.0 Applications with MDA

Modellbasierte Software-
2.0_Model Driven Architecture_UI-Modellierung_Generierung des User Interface

Zusammenfassung. Modellbasierte Software-Entwicklung wie die Model Driven Architecture der OMG eignet sich für Anwendungssysteme, bei denen die fachliche Essenz von den technischen Details zu trennen und dabei ein hoher Grad an Automatisierung erwünscht ist, denn die Vermischung von fachlichen Aspekten (Anwendersicht) mit den implementationsspezifischen Details (Entwicklersicht) erschwert (nicht nur) bei Web 2.0 die Wartbarkeit, Wiederverwendbarkeit und Portabilität der Software. Modellbasierte Software-Entwicklung ermöglicht den Austausch von Technologien bzw. Plattformen – unter Beibehaltung der fachlichen Modelle. Dieser Beitrag zeigt an einem einfachen Beispiel, wie aus einer „alten" Web-Anwendung eine Software unter Verwendung von AJAX für das Web 2.0 entstehen kann, ohne dass eine komplette Neuentwicklung stattfindet.

Summary. Model based software development, e.g. OMG's Model Driven Architecture, is enabling the separation of a high level application design from the implementation making it possible to generate most of the code. The mixture of the application design with implementation-specific details has a negative influence on maintainability, reusability and portability of the software. This is true not only for Web 2.0 applications. Model based approaches make it possible to exchange technologies and/or platforms while maintaining (and reusing) the models of the "business world", i.e. the high level design. This contribution shows, by a simple example, how an "old" web application can be re-engineered using AJAX, without the necessity for a complete new development cycle.

1. Modellbasierte Software-Entwicklung mit der Model Driven Architecture

Seit Ende des Jahres 2000 kommuniziert die OMG den Ansatz der modellgetriebenen Architektur in Form der *Model Driven Architecture (MDA).* Die Vision ist mit den Begriffen Interoperabilität, Integration, Vielfalt und Koexistenz verbunden. Gemäß der MDA ist es besser, computer- bzw. plattformunabhängige Modelle (Computation und Platform Independent Model, CIM, PIM) zu erstellen und die plattformspezifischen Artefakte (Platform Specific Model, PSM) zu generieren, als viele Zeilen Code manuell für jede Plattform (immer wieder) neu zu schreiben bzw. zu ändern, nur weil sich die Technologien ändern[1]. Die Versprechen der OMG sind dann auch sehr weitreichend (Object Management Group: MDA Guide, (2003) S. 1–2):

- **Implementierung:** Ein einmal erstelltes Fachkonzept (PIM) ist für neue Infrastrukturen (wieder-)verwendbar, so dass neueste Technologien mit minimalem Aufwand einsetzbar bzw. integrierbar sind, da die plattformspezifischen Artefakte (PSM), z. B. die Implementierung (Implementation Specific Model, ISM), weitestgehend automatisch durch das PIM generiert anstatt manuell erstellt werden können.
- **Integration:** Da die Implementierung und das Konzept bzw. Design in konsistenter Form vorliegen, lässt sich die Erstellung von Lösungen für die Integration anderer Infrastrukturen genauso automatisieren wie die Implementierung selbst.
- **Wartung:** Diese „ungeliebte", aber oftmals aufwändigste und längste Phase im Lebenszyklus eines Software-Produktes wird durch die Verfügbarkeit eines formalen und aktuellen Konzeptes erleichtert, da Änderungen und Erweiterungen nun nicht mehr im undokumentierten Code, sondern im Konzept vorgenommen werden können.

[1] Der Einfachheit halber ist das CIM (Computation Independent Model) im Folgenden nicht näher erläutert.

DOI 10.1524/icom.2007.6.1.28

- **Test und Simulation:** Modelle, aus denen sich ein Code generieren lässt, eignen sich genauso gut für die Validierung gegen Requirements, den Test gegen diverse Infrastrukturen oder die Simulation des Verhaltens vor der eigentlichen Implementierung bzw. Inbetriebnahme.

Wichtig sind bei der MDA zwei Aspekte, die Ausgangspunkt für Missverständnisse sein können: a) Die Modelle müssen nicht zwingend in UML (Object Management Group: UML, 2003) vorliegen, sondern lediglich MOF-konform (Object Management Group: MOF 2.0, 2003) sein, damit Metamodelle kompatibel und Abfragen, Sichten und Transformationen (Object Management Group: Query, Views, Transformations, 2005) standardisiert möglich werden, und b) eine eigene Vorgehensweise mit passenden Methoden müssen definiert werden, da die MDA sehr generisch und vielfältig einsetzbar ist und nur sehr abstrakt das Vorgehen darlegt (Bild 1). Ein auf diesem allgemeinen MDA-Pattern basierendes Vorgehen zeigt schematisch Bild 2: Die objektorientierte Software-Entwicklung gliedert sich in zwei Projekte, wobei das Plattformprojekt die Generatorentwicklung mit den Modelltransformationen und der Code-Generierung fokussiert und das Kundenprojekt die anwendungsspezifischen Bereiche umfasst – einschließlich einiger (weniger) manuell zu erstellender Code-Anteile.

Wenig konkret in Bezug auf die UI-Entwicklung sind bei der MDA auch die Perspektiven (Viewpoints) (Object Management Group: MDA Guide (2003) S. 2–3): *Computation Independent Viewpoint, Platform Independent Viewpoint*

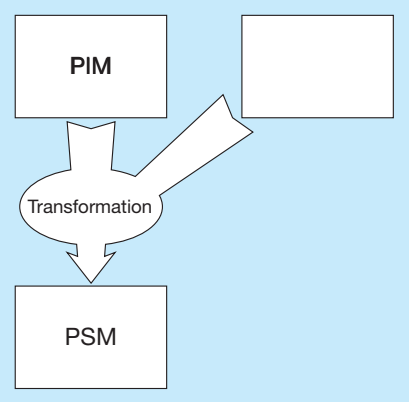
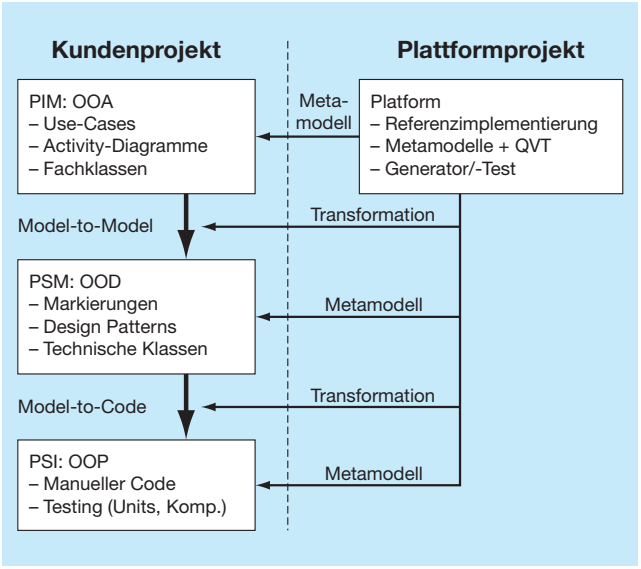


Bild 1: MDA-Pattern mit Transformation (Object Management Group: MDA Guide (2003) S. 2–7)

Bild 2: Spezifisches MDA-Vorgehen für OO-Projekte



und Platform Specific Viewpoint[2]. Zusammenfassend lässt sich feststellen, dass die MDA der OMG (aber auch andere modellgetriebene Ansätze) in erster Linie „nur" einen Rahmen für den selbst auszugestaltenden Entwicklungsprozess bieten. Bis zur modellgetriebenen Entwicklung von Web 2.0-Anwendungen gemäß der eigenen Anforderungen und mit einer funktionierenden Werkzeugkette ist es daher ein weiter, aber lohnender Weg[3]. Im Folgenden soll dies ansatzweise gezeigt werden.

2. Modellbasierte HCI-Entwicklung

Bevor auf den modellgetriebenen Ansatz in Hinblick auf Web 2.0 näher eingegangen wird, seien einige vorbereitende Überlegungen zum Thema Human Computer Interaction in Verbindung mit Modellierung vorgestellt.

Modellierung in Verbindung mit der MDA impliziert Abstraktion von der jeweiligen Plattform, für die das System erstellt werden soll (Zielarchitektur). Im Falle der Entwicklung des User Interface abstrahieren die HCI-Modelle daher vom UI-System, sind also unabhängig von der entsprechenden Technologie, z. B. XHTML, KDE, MacOS, Java AWT/Swing.

Um das UI abstrakt zu beschreiben sind verschiedene Verfahren und Sprachen bzw. Notationsformen entwickelt worden, wobei hier nur einige genannt sind. Multi-Party-Grammatiken, Zustandsübergangsdiagramme, Petri-Netze und andere spezielle Notationsformen wie die User Action Notation (Hartson, Siochi, Hix, 1990) existieren schon recht lange, haben sich jedoch nur sehr bedingt verbreitet. Neuere Ansätze wurden teilweise auf Basis von ER-Modellen oder objektorientierten Modellierungssprachen wie die UML entwickelt, z. B. *Relationship Management Model* (RMM, Methode für strukturiertes Hypermedia-Design basiert auf dem E-R Modell (Isakowitz, Stohr, Balasubramanian, 1995), Enhanced Object-Relationship Model (EORM, interaktionsorientiertes Modell basiert auf OMT (Lange, 1994), Hypermedia Design Method (HDM, Grob- und Feinentwurf, Trennung von Inhalt, Struktur, Präsentation, Dynamik und Interaktion, vgl. Garzotto (1993), OOHDM (Object-Oriented-Hypermedia-Model, vgl. Rossi, Schwabe (1995), UMLi (Erweiterte UML-Aktivitätsdiagramme für interaktive Systeme, vgl. da Silva, Paton (2003). Aber auch diese Verfahren sind in ihrer praktischen Verbreitung beschränkt bzw. befinden sich erst in der Erprobungsphase. Weiterhin hemmte die fehlende Tool-Un-

[2] Allerdings bieten auch andere Ansätze wenig Detailinformationen für eine systematische UI-Entwicklung, so ist z. B. der Unified Process (UP) mit der 4+1-View-Konzept (logisch, implementation, process, development, use-case, (Kruchten 2004)) noch unvollständig in Bezug auf die Methoden und Modelle des Usability-Engineering.

[3] Bei der Lufthansa Systems betrug die Einsparung in einem Projekt (Logistik-Software) durch den MDA-Ansatz über 100.000,– Euro (Friese 2006, S. 33].

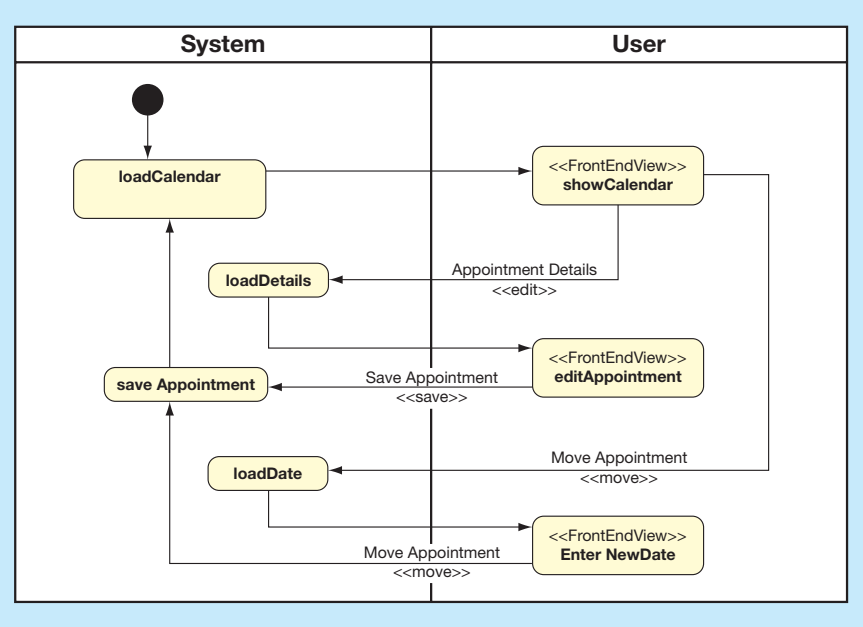


Bild 3: Klassische Web-Anwendung für die Terminverwaltung

terstützung in der Vergangenheit die Verbreitung spezieller Modellierungstechniken für das User Interface, die für einen MDA-Ansatz und die Code-Generierung notwendig sind. Dies hat sich mit der Verfügbarkeit von UML 2.0 und entsprechenden UML-Tools verbessert.

Die genannten Verfahren und Notationsformen sind jedoch nur ein Teilaspekt der modellgetriebenen HCI-Entwicklung, denn es gibt auch plattformspezifische bzw. domainspezifische Abstraktionsansätze, die allerdings teilweise sehr technologielastig sind und im extremen Fall nur eine API (Application Programming Interface) für die komfortable Programmierung bereit stellen. In erster Linie sind hier die bekannten Frameworks wie z. B. Apache struts (Apache Software Foundation), Java Server Faces (Sun Microsystems), Dojo (Dojo Foundation) und Eclipse SWT (Eclipse Foundation) zu nennen. Die OASIS User Interface Markup Language (UIML) als Meta-Sprache, Mozilla XML User Interface Language (XUL) für cross-platform-Anwendungen und XForms (W3C) für Web-Formulare enthalten zwar einen Satz abstrakter Elemente für die UI-Beschreibung, es fehlt jedoch an konzeptuellen Elementen. So ist z. B. die Aufgabenebene nicht ausreichend berücksichtigt, was angesichts des Anspruches auch nicht verwunderlich ist. Solche Frameworks sind jedoch ein wichtiger Baustein für Web 2.0-Anwendungen, da durch sie die einfache Generierung z. B. von AJAX-basierten Anwendungen erst ermöglicht werden.

Zusammenfassend ist festzustellen, dass die modellbasierte HCI-Entwicklung sich zunehmend entwickelt und dabei sowohl die UML, z. B. in Form von strukturierten Aktivitätsdiagrammen für Aufgabenmodelle (Forbig, Reichert 2006) oder speziellen Profilen mit Stereotypen (Conallen 1999), intensiv Verwendung findet, als die MDA mit den damit verbundenen Standards der OMG beachtet werden. Auch wird auf Open Source Werkzeuge für die MDA-Toolkette zurückgegriffen, z. B. Eclipse und OpenArchitectureWare. Besonders deutlich wird dies bei dem Projekt EMODE (Enabling Model Transformation-Based Cost Efficient Adaptive Multi-modal User Interfaces), welches die Entwicklung adaptiver, multimodaler Anwendungen fokussiert – unter Berücksichtigung der UML und der MDA.

3. MDA und Web 2.0 – ein Beispiel

Im Folgenden sei eine Web-Anwendung vorgestellt, die Studierenden eines Seminars dazu dient, Termine zu vereinbaren. Dabei ist im ersten Schritt die Entwicklung einer klassischen Browser-basierten, d. h. mit HTML realisierten, Applikation gezeigt, die dann im zweiten Schritt durch AJAX und Zusatzfunktionen zu einer Web 2.0-Anwendung erweitert wird.

Ein Aktivitätsdiagramm dient auf der Modellierungsebene als Grundlage und stellt damit das PIM (Platform Independent Model). Mit einer sog. Cartridge, die Transformationsregeln für den Transformator bzw. Generator enthält, kann nun ein solches PIM in ein PSM oder eine PSI (Code-Generierung) transformiert werden. Eine Cartridge ist demnach immer für eine spezielle Zielplattform gemacht, d.h. ein Ausgangsmodell (PIM) lässt sich mit verschiedenen Cartridges zu verschiedenen Anwendungssystemen transformieren.

Bei dem folgenden Aktivitätsdiagramm ist zu beachten, dass es sich um eine stark vereinfachte Darstellung handelt. Insbesondere der Datenaspekt ist hier zur besseren Übersicht weggelassen. Die linke Seite (Partion) des Diagramms zeigt die Systemaktionen, während rechts die Webseiten (FrontEndView) modelliert sind. Eine Kalenderansicht ermöglicht die Bearbeitung der Daten (Stereotyp <<edit>>) und das Verschieben, in dem das geänderte Datum in einem Dialog (EnterNewDate) erfasst wird (<<move>>) (Bild 3).

Durch eine Model-to-Code-Transformation entsteht mit Hilfe des AndroMDA-Generators (Andromda) und der dafür entwickelten bpm4struts-Cartridge,

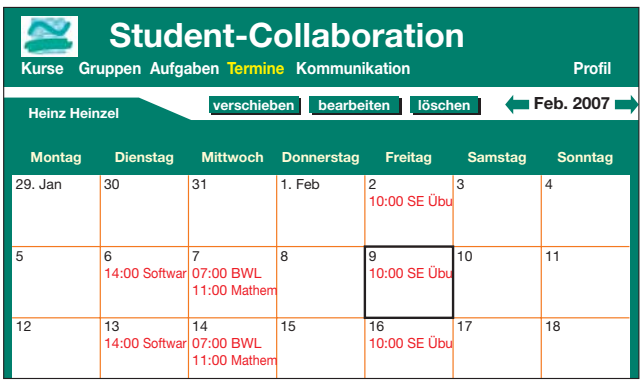


Bild 4: Klassische Web-Anwendung für die Terminverwaltung

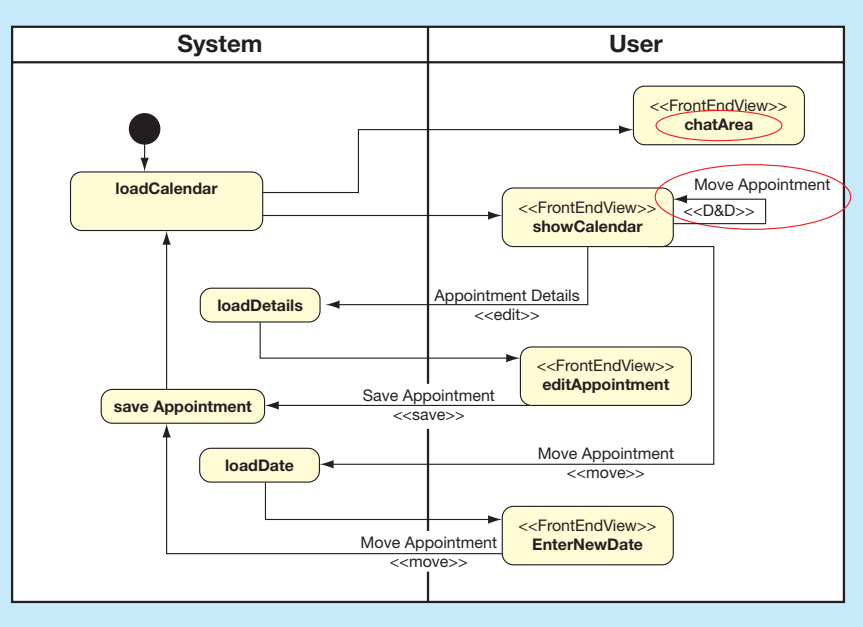


Bild 5: Verändertes Aktivitätsdiagramm für Drag&Drop und Chat

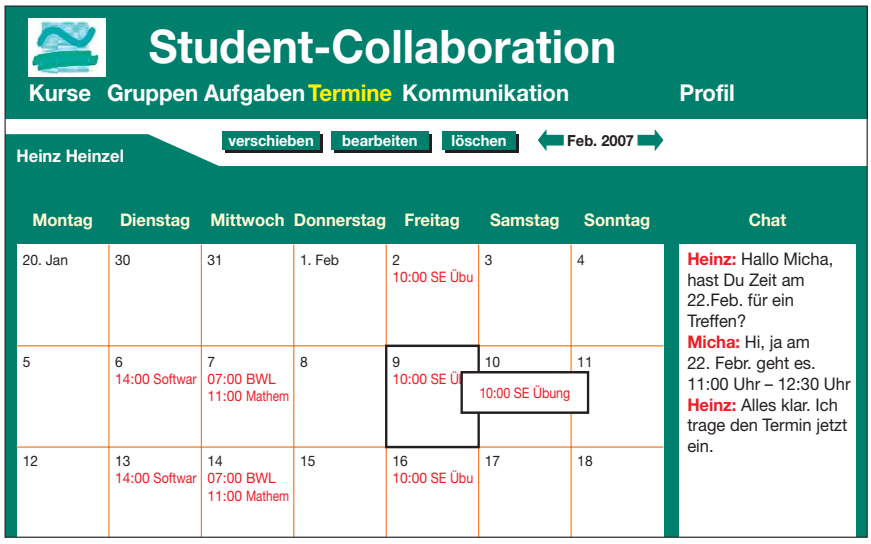


Bild 6: Terminverwaltung als Web 2.0-Anwendung

die die Transformationsvorschriften enthält, eine struts-Anwendung mit JSPs, die gemäß der MDA als PSI (Platform Specific Implementation[4]) zu bezeichnen ist und sich im Browser folgendermaßen darstellt (Bild 4).

Um nun die Einfachheit und die Wiederverwendbarkeit der plattformunabhängigen Modelle der MDA zu demonstrieren, soll das Modell nun um wenige (nicht technologische) Elemente ergänzt werden, damit anschließend eine Web 2.0-Anwendung generierbar ist. Einerseits soll die Bearbeitung der Termine durch Drag-and-Drop vereinfacht werden (D&D) und andererseits ist ein nebenläufiger Chat-Bereich bereitzustellen, damit sich die Studierenden direkt (synchron) austauschen und parallel an der Terminvereinbarung arbeiten können (chatArea). Terminvereinbarung arbeiten können (chatArea). Zwei kleinere Ergänzungen sind im Aktivitätsdiagramm notwendig (Bild 5).

Nun erfolgt eine erneute Model-to-Code-Transformation ebenfalls mit dem AndroMDA-Generator, jedoch mit einer anderen Cartridge, die Anwendungen für JSF mit AJAX generiert. Das Ergebnis ist eine verbesserte Applikation auf der Basis eines nahezu unveränderten Aktivitätsdiagramms (Bild 6): Die Termine lassen sich per Drag&Drop verschieben und ein Chat kann genutzt werden.

4. Fazit

Mit der MDA steigt der Formalisierungsgrad und die Abstraktion, was Vor- und Nachteile hat: Problematisch erweist sich in der Praxis die Tatsache, dass die Anforderungen an die Modellierung und damit an die Qualifikation des Systemanalytikers bzw. Usability-Engineers steigen. Die Abstraktionen müssen beherrscht werden und die Modellierungstechnik ist präziser und komplexer als bei UI-Entwicklungsansätzen, deren Artefakte nicht automatisch durch einen Generator weiterverarbeitet werden. Auf der anderen Seite bietet der modellgetriebene Ansatz die Möglichkeit, unabhängig von einer Technologie die Konzepte des Anwendungssystems und insbesonders des User Interface entwickeln zu können und anschließend eine flexible Umsetzung nahezu automatisch auf verschiedenen Zielplattformen durchzuführen. Damit entstünden statische und wieder verwendbare Modelle, die einen technologischen Wandel wesentlich leichter „mitmachen", als der bisherige technologiezentrierte Ansatz, der ein aufwändiges Re-Engineering bzw. eine Neuentwicklung zur Folge hatte. Web 2.0-Anwendungen zeigen erneut, dass das Software-Engineering endlich Techniken bereitstellen muss, um eine Trennung zwischen Fachlicher Welt und technischen Plattformen zu ermöglichen und um letztendlich eine Industrialisierung der Software-Produktion herbeizuführen (Zacharias 2007). Die MDA ist dafür das geeignete Mittel, wobei in Hinblick auf das Usability-Engineering noch eine intensive Arbeit erfolgen muss. So ist beispielsweise zu klären, wie Ansätze des User Centered Design (Norman 1990), Contextual Design (Bayer, Holtzblatt, 1998), Partizipative Entwicklung (Koslowski 1998) oder der Usability Engineering Life Cycle (Mayhew 1999) mit der MDA zu vereinbaren sind. Gleiches gilt für Prinzipien und Heuristiken (z. B. von Constatine (1999), Nielsen (1994), Shneiderman (1997) oder den

[4] Bei Model-to-Code-Transformationen entsteht kein Zwischenmodell (PSM), sondern direkt der Code, der zwar auch als PSM bezeichnet werden könnte, die Bezeichnung PSI allerdings passender ist.

Vorgaben der ISO 9241 und Style Guides wie z. B. Lynch (2002) oder

Sun Microsystems 2001). Eine Möglichkeit wären beispielsweise die Formalisierung von Usability-Patterns, bei der ergonomische Anforderungen auf Muster ähnlich den Design Pattern herunter gebrochen werden, um automatisch verarbeitet zu werden und Teile des UI zu generieren. So könnte z. B. die Steuerbarkeit mit Usability-Patterns (Bayle et al. 1998) wie Redo/Undo-Pattern und YNC-Pattern (Yes, No, Cancel) formal(er) beschrieben als dies in Style Guides der Fall ist. Dies könnte gerade für Web 2.0 in Verbindung mit modellgetriebener Entwicklung ein Schritt in Richtung effizienter Software-Produktion und effektivem Usability-Engineering sein.

Literatur

AndroMDA: www.andromda.org

Bayer, H.; Holtzblatt, K.: Contextual Design. Morgan Kaufmann Publishers, 1998.

Bayle, E.; Bellamy, R.; Casaday, G.; Erickson, T.; Fincher, S.; Grinter, B.; Gross, B.; Lehder, D.; Marmolin, H.; Moore, B.; Potts, C.; Skousen, G.; Thomas, J.: Putting it all together: towards a pattern language for interaction design. *SGICHI Bulletin* **30** (1) (1998) 17–23.

Conallen, J.: Building Web applications with UML. Addison Wesley, 1999.

Constantine, L.; Lockwood, L.: Software for Use. A practical Guide to the Models and Methods of Usage-Centered Design. Addison-Wesley, 1999.

Fieber, F.; Neuhaus, W.; Petrasch R. (Hrsg.): Werkzeuge und Anwendungsgebiete der modellbasierten Software-Entwicklung. 1. Workshop der Special Interest Group „Model-Driven Software Engineering" (Tagungsband), Logos-Verlag, Berlin, 2006.

Forbig, P.; Reichert, D.: Modellbasierte Entwicklung von modellbasierten Werkzeugen. In: *Werkzeuge und Anwendungsgebiete der modellbasierten Software-Entwicklung.* Workshop der Special Interest Group „Model-Driven Software Engineering" (Tagungsband). (Fieber, F.; Neuhaus, W.; Petrasch R. Hrsg.). Logos-Verlag, Berlin, 2006.

Friese, P.: Einsatz des Open Source MDA Generators AndroMDA bei Lufthansa Systems. In: *Werkzeuge und Anwendungsgebiete der modellbasierten Software-Entwicklung.* Workshop der Special Interest Group „Model-Driven Software Engineering" (Tagungsband). (Fieber, F.; Neuhaus, W.; Petrasch R. Hrsg.). Logos-Verlag, Berlin, 2006.

Garzotto, F.; Paolini, P.: Schwabe, D.: HDM: A model-based approach to hypertext application design. *ACM Transactions on Information Systems,* **11** (1):1–26, January 1993.

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, Pages 181–203.

Isakowitz, T.; Stohr, E. A.; Balasubramanian, P.: RMM: A Methodologie for Structured Hypermedia Design. Communications of the ACM, 38(8): 34–44, August 1995.

ISO 9241: Ergonomic Requirements for Office Work with VDTs. ISO, 1996.

Koslowski, K.: Partizipative Systementwicklung und Software-Engineering. Westdeutscher Verlag, 1998.

Kruchten, Ph.: The Rational Unified Process – An Introduction. Addison-Wesley, Pearson Education, 3rd ed., 2004.

Lange, D.: Hypermedia Design Method, 1994

Lynch, P.: Web Style Guide: Basic Design Principles for Creating Web Sites, 2. Auflage, 2002.

Mayhew; D.J.: The Usability Engineering Lifecycle: A Practitioner's Handbook for User Interface Design. Morgan Kaufmann Publishers, 1999.

Norman, D. A.: The Design of Everyday Things. Doubleday Books, 1990.

Object Management Group (OMG): MDA Guide. Version 1.0.1, omg/2003-06-01, June 2003

Object Management Group (OMG): MOF 2.0 – Meta Object Facility Core Specification. Version 2.0. Adopted Specification, ptc/03-10-04, Oct. 2003.

Nielsen, J.: Usability Engineering. AP Professional, 1994.

Object Management Group (OMG): Ouery/Views/Transformation. Final adopted Spec., ptc/05-11-01, Nov. 2005.

Rossi, G.; Schwabe, D.: Building Hypermedia Applications as Navigational Views of Information Models. In Proceedings of Annual Hawaii International Conference on System Science, number 28, Januar 1995.

Shneiderman, B.: Designing the User Interface: Strategies for Effective Human-Computer Interaction. 3. Auflage, Addison-Wesley, 1997.

da Silva, P. P.; Paton, N. W: User Interface Modeling in UMLi. IEEE Software, July/Aug. 2003, S. 62–69.

Sun Microsystems: Java Look and Feel Design Guidelines. Addison Wesley, 2. Auflage, 2001.

Object Management Group (OMG): UML 2.0 – Unified Modeling Language (UML) Superstructure Specification. Version 2.0. Final Adopted Specification, ptc/03-09-15, Dec. 2003.

http://www.mozilla.org/projects/xul

Zacharias, R.: Produktlinen – Der nächste Schritt in Richtung Software-Industrialisierung. *javamagazin* **3** (2007) S. 69 ff.

Prof. Dr. 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 Prof. Petrasch an der Technischen Fachhochschule 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 weitere Fachbeiträge u.a. zum Thema Usability-Engineering.
E-Mail: petrasch@tfh-berlin.de
www.tfh-berlin.de/~petrasch