von bernhard rumpe und
joachim schmid

mehr zum thema:
www.bea.de

OBERFLÄCHEN MIT MDA: BESCHREIBEN STATT PROGRAMMIEREN

Die grafische Benutzungsoberfläche ist ein wesentlicher und komplexer Bestandteil heutiger Applikationen und verursacht trotz einer gewissen Systematik in der Realisierung immer noch einen hohen Entwicklungsaufwand. Der Aufwand und damit die Kosten lassen sich mit Hilfe eines auf der „Model Driven Architecture" (MDA) basierenden GUI-Frameworks deutlich reduzieren. Die Entwickler können mit einer problemangepassten, kompakten Sprache GUI-Systeme beschreiben und mithilfe eines GUI-Frameworks dynamisch und automatisiert Code generieren oder die erstellten Modelle direkt zur Ausführung bringen. Das in dem Artikel vorgestellte GUI-Framework wurde bereits in mehreren industriellen Anwendungen höchst erfolgreich und effizient eingesetzt.

MDA

Die *Model Driven Architecture (MDA)* (vgl. [OMG], [Kle03], [Fra03]) ist auf dem Weg zu einer Technologie, die es Softwareentwicklern erlaubt, Software noch schneller, effizienter und qualitativ hochwertiger zu erstellen. Eine Online-Ausgabe des OBJEKTspektrums 2004 hat das Thema MDA beziehungsweise seine Verallgemeinerung, das *Model Driven Development (MDD)*, deshalb als Schwerpunkt behandelt (siehe www.sigs-datacom. de/sd/publications/ os/pub_article_overview.htm).

Der Kern von MDA besteht aus der Trennung von PIM-Teilen (*Platform Independent Model)* und PSM-Teilen (*Platform Specific Model)*, die beide unabhängig voneinander wieder verwendet werden können. Damit wird ein ähnlicher Trennungs- und Flexibilisierungseffekt erreicht wie er auch in [Sie04] beschrieben ist. Die plattformunabhängigen Teile stellen den Applikationskern dar. In diesem Kern ist teuer erworbenes Wissen – zum Beispiel über Geschäftsabläufe oder Gerätefunktionen – umgesetzt. Die explizite Kapselung des Applikationskerns erlaubt dessen leichtere Wiederverwendung, zum Beispiel bei einer Migration auf eine andere Plattform. Demgegenüber ist der plattformspezifische Teil meist als Transformationssystem gegeben, das zur vorhandenen Anwendung spezifische Funktionalität hinzufügt, dieses mit einem Framework oder Betriebssystem integriert und damit in einen systemspezifischen Kontext einbettet. Derartige Transformationen sind applikationsunabhängig und können so für viele Anwendungen eingesetzt werden.

Ursprünglich war die MDA von der Object Managament Group (OMG) vor allem als Methodik zum Einsatz von UML-Modellen für PIM gedacht. Es hat sich aber gezeigt, dass MDA-Techniken über die UML hinaus auch für domänenspezifische Aufgabenstellungen geeignet sind. Die Vorteile der MDA zeigen sich vor allem in konkreten Ausprägungen, bei denen spezifische Domänen durch eine problemangepasste Sprache und geeignete Werkzeuge unterstützt werden. Durch diese Form der Spezialisierung verlieren MDA-Techniken an Breite der Anwendbarkeit, gewinnen aber innerhalb der Anwendungsdomäne viel Potenzial zur Effektivitätssteigerung.

Der Nutzen von domänenspezifischen Sprachen (*Domain Specific Languages – DSLs)* liegt genau darin, für ein Aufgabengebiet spezialisierte, anwendungsnahe Abstraktionen zur Verfügung zu stellen. Der Einsatz von DSLs zeigt im Erfolgsfall oft eine hohe Effektivitätssteigerung, wenn man von dem limitierten und nur einmaligen Aufwand, eine DSL zu erlernen, absieht. Die DSL des in diesem Artikel vorgestellen *Xui*-Frameworks ist eine solche Spezialisierung für *grafische Benutzungsoberflächen (GUIs)* datenbankbasierter Geschäftssysteme, die es erlaubt sehr kompakt GUIs zu spezifizieren.

MDA in Xui

Flexibler noch als ein generativer Transformationsansatz ist der in diesem Artikel beschriebene Ansatz der Interpretation von kompakten Beschreibungen. Dieser bietet sich dann an, wenn dadurch nur wenig organisatorischer Overhead entsteht und so das System kaum verlangsamt, aber ein agiles, zeitnahes *Customizing* durch den Kunden oder eine unterstützende Organisationseinheit erfolgen soll. Speziell im von Kunden sehr unmittelbar wahrgenommenen Bereich der GUIs ist eine flexible Kundenanpassung geboten. Das Xui-Framework bietet genau dies. Dabei wird auch sichtbar, wie Softwareprodukte durch mächtige Parametrisierungstechniken für zukünftige Anpassungen flexibilisiert werden können, um damit eine möglichst lange Halbwertszeit des Produktionsguts „Software" zu erreichen.

Als DSLs, die für MDA-Verfahren genutzt werden, kommen heute im Wesentlichen UML- oder XML-basierte Sprachen zum Einsatz. Die im Xui-System eingesetzte Sprache basiert auf einem XML-Dialekt, der dynamisch zur Laufzeit verarbeitet wird. Dabei werden Xui-Beschreibungen vom Xui-Framework eingelesen, mit der Geschäftslogik in Java verknüpft und so zur Ausführung gebracht.

die autoren

Prof. Dr. Bernhard Rumpe leitet das Institut für Software Systems Engineering an der TU Braunschweig. Zu seinen Hauptinteressen gehören MDA, DSLs, evolutionäre Softwareentwicklung und Qualitätssicherung. Er hat unter anderem zwei Bücher zum Thema „Agile Modellierung mit UML" publiziert.

Dr. Joachim Schmid (E-Mail: Joachim.Schmid@bea.de) ist Projektleiter bei der Beck et al. Projects GmbH. Seine Schwerpunkte sind agile Softwareentwicklung mit Java und .NET. Von ihm gibt es ein Buch über das Thema „Java and the Java Virtual Machine".

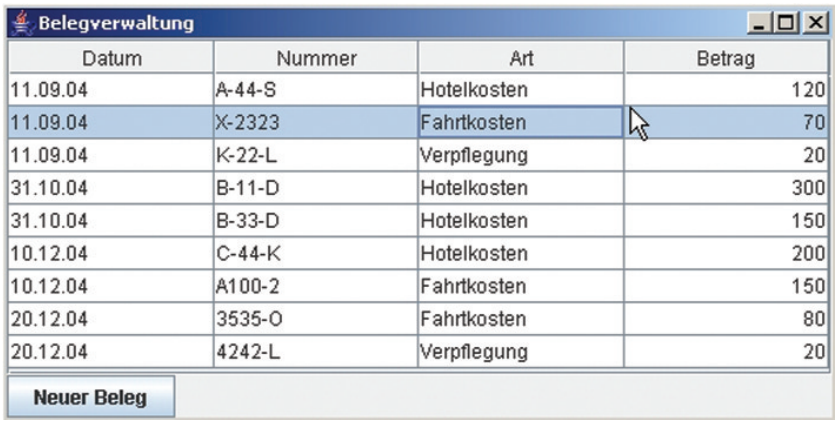

Abb. 1: Screenshot der Beispielanwendung

Der Ansatz zeigt sich in der Praxis als deutlich flexibler und effizienter als die direkte Nutzung eines Frameworks, denn die Nutzung von Xui führt zu wesentlich abstrakteren und kompakteren Beschreibungen. Dabei werden viele Variabilitäten der zu Grunde liegenden GUI-Bibliothek durch überschreibbare Defaults voreingestellt. Als Nebeneffekt wird dadurch das einheitliche Design der Oberfläche elegant unterstützt. Durch die deutlich kompaktere Beschreibung der GUI wird sowohl ihre Evolution erleichtert, als auch die Qualitätssicherung durch eine robustere Nutzung der zu Grunde liegenden GUI-Komponenten vereinfacht. Dadurch entsteht insgesamt eine deutliche Reduktion des Arbeitsaufwands, der sich seinerseits in der einzusetzenden Methodik auswirkt.

Tatsächlich konnten alle bisher auf Basis von Xui entwickelten Projekte sehr agil durchgeführt werden. Xui erlaubt die schnelle Erstellung von GUI-Prototypen. So kann dem Kunden effektiv Feedback seiner Anforderungen gegeben werden, flexible Reaktionen auf Änderungen der Anforderungen sind möglich, das System kann inkrementell erstellt werden und die GUI ist praktisch jederzeit lauffähig und demonstrierbar.

Anwendungsbeispiel

Anhand einer (vereinfachten) Verwaltung von Belegen lässt sich die Modellierung von Oberflächen mit Hilfe von Xui erläutern. Die Belegverwaltung ist in Abbildung 1 dargestellt und zeigt, dass ein Beleg Datum, Nummer, Art und Betrag enthält.

Eine grafische Oberfläche lässt sich am besten in Form einer Baumstruktur beschreiben. Dieses Strukturierungsparadigma wird mittlerweile auch in der Implementierung von GUIs, wie etwa in „Swing" realisiert. Da Xui speziell für eine Umsetzung nach Swing gedacht ist, enthält es außerdem viele Elemente, die direkt in Swing abgebildet werden können. Dadurch wird für Swing-erfahrende Entwickler Xui auch konzeptuell leichter erlernbar und der Wiederverwendungseffekt (eben von Swing) ist deutlich hoch.

Das sichtbare Fenster ist die konzeptuelle Wurzel des Baumes und enthält einen Layout-Manager, der Komponenten anordnen kann. Da Layout-Manager wiederum Komponenten sind, können beliebig tief geschachtelte Strukturen entstehen. Die Oberflächenstruktur ist aus dem Baum in Abbildung 2 ersichtlich. An der Wurzel steht ein Layout-Manger, der die beiden Komponenten *Scrollpane* und *Button* enthält. Der Inhalt des *Scrollpane* ist eine Tabelle.

Zur Darstellung der Baumstruktur als Grundprinzip zur Verschachtelung von Komponenten in der Modellierungssprache von Xui eignet sich die standardisierte Beschreibungssprache XML. Die vollständige XML-Beschreibung für die Belegverwaltung ist ebenfalls in Abbildung 2 dargestellt.

Bei einer Xui-Beschreibung handelt es sich um eine XML-Datei mit dem *Root-Tag* XuiView (Zeile 2 in Abb. 2). Jede XuiView ist mit einem Controller in Java verbunden. Diese Verknüpfung entsteht bei der Initialisierung des Controllers mit der entsprechenden View. Das Top-Attribut des XuiView-Elements (Zeile 3) definiert dabei das Wurzelelement der grafischen Benutzungsoberfläche. In diesem Beispiel entspricht es der Komponente mit dem Namen main. Diese ist definiert als ein TableLayout-Manager (Zeile 7), der die beiden Komponenten scrollpane (Zeile 8) und buttonNew (Zeile 9) vertikal anordnet. Bei den Formatierungsanweisungen (Zeile 7) in den Attributen cols und rows steht f für fill (dehnbar) und p für die preferred size der Komponente.

Die im Scrollpane (Zeile 12) dargestellte Tabelle (Zeile 13) enthält die Elemente, die der Methodenaufruf des content-Attributs (Zeile 13) als Ergebnis liefert (in dem Beispiel die Ergebnisliste der Methode getRows; diese Methode ist in dem verknüpften Controller implementiert). In der Programmiersprache Java sind alle Container-Klassen untypisiert. Aus diesem Grund definiert das Attribut contentType (Zeile 13) den Typ der einzelnen Elemente. Es kann dabei ein Alias eingesetzt werden, der vorher mit dem XuiClassAlias-*Tag* (Zeile 4) definiert wird.

Die XML-*Tags* tableColumn (Zeilen 14 bis 23) definieren die einzelnen Spalten der

```
1  <?xml version="1.0" encoding="iso-8859-1"?>
2  <!DOCTYPE XuiView SYSTEM "xuiView.dtd">
3  <XuiView top="main">
4  <XuiClassAlias name="Beleg" class="de.bea.xui.examples.bv.Beleg"/>
5   <XuiClassAlias name="BelegArt" class="de.bea.xui.examples.bv.BelegArt"/>
6
7   <tableLayout xuiId="main" cols="f" rows="f,p"
8    <placeTl row="0" col="0" id="scrollpane"/>
9    <placeTl row="1" col="0" id="buttonNew" hAlign="left"/>
10  </tableLayout>
11
12 <scrollPane xuiId="scrollpane"
13  <table content="getRows" contentType="Beleg">
14   <tableColumn name="'Datum'" value="conv.day2str(?item.getDay)"
15    setValue="?item.setDay(conv.str2Day(?value))"/>
16   <tableColumn name="'Nummer'" value="?item.getNummer"
17    setValue="?item.setNummer(?value)"/>
18   <tableColumn name="'Art'" listType="BelegArt" list="getBelegArten"
19    listItem="?item.getBelegArt" value="?listItem.getText"
20    setValue="?item.setBelegArt(?value)"/>
21   <tableColumn name="'Betrag'" alignment="right"
22    value="conv.int2str(?item.getBetrag)"
23    setValue="?item.setBetrag(conv.str2int(?value))"/>
24  </table>
25 </scrollPane>
26
27 <button xuiId="buttonNew" text="'Neuer Beleg'"
28   actionCommand="actionNeuerBeleg"/>
29 </XuiView>
```


Abb. 2: XML-Beschreibung

Tabelle. In diesem Beispiel kommt eine spezielle Tabelle mit vier festen Spalten zur Anwendung. Die Definition einer Spalte besteht aus dem Spaltennamen (Attribut `name`), dem anzuzeigenden Inhalt (Attribut `value`) und gegebenenfalls einem Methodenaufruf, falls die entsprechende Zelle editierbar sein soll (Attribut `setValue`). Die erste Spalte in der Belegverwaltung hat den konstanten Namen „Datum" (Zeile 14). Anstelle der expliziten Konstanten (Notation mit Apostrophen) könnte auch ein Methodenaufruf mit Rückgabetyp `String` verwendet werden. Dadurch kann etwa eine Dynamisierung durch Sprachvarianten erreicht werden.

Das `value`-Attribut definiert den in einer Zelle anzuzeigenden Text. Da dieser Text von dem aktuellen Zeilenelement abhängig ist, kann der Parameter `?item` als Platzhalter für das Zeilenelement eingesetzt werden. Das Beispiel `conv.str2Day(?item.getDay)` (Zeile 14) ruft auf dem aktuellen Zeilenelement die Methode `getDay` auf und übergibt den Rückgabewert an die Methode `str2Day`. Die Methode `str2Day` wird auf dem Objekt aufgerufen, das beim Aufruf von `conv` zurückgeliefert wird. Die Ausführung erfolgt somit analog zur Ausführung von geschachtelten Methodenaufrufen in Java und ist um spezielle Parameter zur Referenzierung von GUI-Elementen ergänzt.

Damit eine Zelle editierbar wird, genügt es einen Methodenaufruf beim Attribut `setValue` zu definieren. Auch hier steht das aktuelle Zeilenelement im Platzhalter `?item` zur Verfügung. Zusätzlich enthält der Parameter `?value` den Eingabewert der Oberfläche vom Typ `String`. Dieser wird bei der Datumsspalte mit Hilfe der Konvertierungsmethode `str2Day` in eine Instanz vom Typ `Datum` zurückkonvertiert. Aus diesem Beispiel ist bereits das durchgängige Prinzip von Xui-Beschreibungen ersichtlich: Die Konvertierungen von Business-Typen in Strings und in die umgekehrte Richtung werden direkt in der XML-Beschreibung festgelegt. Das hat den Vorteil, dass die Applikationslogik typisiert implementierbar ist und sich beispielsweise nicht mehr um die Konvertierung kümmern muss. Mit einem durchdachten Konzept zur Ausnahmebehandlung wird sichergestellt, dass Konvertierungsfehler (z. B. nicht korrektes Datum) von dem Xui-Framework abgefangen und entweder systematisch behandelt oder an spezifische `Handler` der Applikation delegiert werden. Darauf wird hier allerdings nicht näher eingegangen.

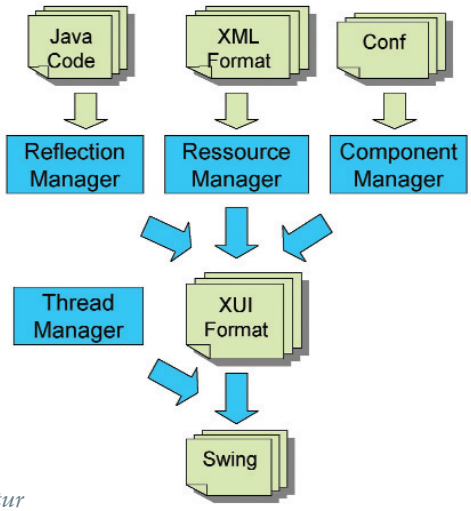

Abb. 3: Xui-Architektur

Einen Spezialfall für editierbare Zellen stellt die Spalte `Art` dar. In diesr Spalte stehen nur drei Möglichkeiten zur Verfügung. Dies kann sehr einfach in einem `tableColumn`-*Tag* definiert werden (Zeile 18), indem das Attribut `list` die unterschiedlichen Auswahlmöglichkeiten liefert, `listType` den Typ dieser Elemente definiert, `listItem` das selektierte Element in der Liste definiert und `value` wiederum den anzuzeigenden Text. Für die Anzeige der `Belegart` liefert die Methode `getText` dynamisch den anzuzeigenden Text (Zeile 19). Auch hier ist wiederum ersichtlich, dass stets mit Business-Klassen gearbeitet wird und die Konvertierung in Text nur zu Darstellungszwecken erfolgt.

Die Definition des Buttons *Neuer Beleg* ist analog aufgebaut. Das Attribut `actionCommand` definiert dabei die Methode, die beim Drücken des Buttons ausgeführt werden soll.

Architektur

Das Xui-Framework ist modular aufgebaut und zerteilt sich in die Komponenten *Thread-Manager*, *Ressource-Manager*, *Reflection-Manager* und *Component-Manager* (siehe Abb. 3).

Der *Thread-Manager* ist verantwortlich für die zuverlässige und effiziente Ausführung der Methodenaufrufe.

Die Methodenaufrufe selbst werden vom *Reflection-Manager* zur Verfügung gestellt. Dieser liest die textuelle Darstellung der Methodenaufrufe aus der Xui-Beschreibung und übersetzt die Darstellung in Java-Strukturen. Dazu arbeitet er wie ein Java-Compiler und sucht nach passenden Methoden unter Berücksichtigung der Vererbungshierarchie und Methoden, die sich nur in der Signatur der Argumente unterscheiden.

Der *Ressource-Manager* ist verantwortlich für das Einlesen der Xui-Beschreibungen. Er zerteilt dabei eine XML-Beschreibung in die einzelnen Top-Level-Beschreibungen und ermöglicht das Auffinden von einzelnen Top-Level-Beschreibungen basierend auf der so genannten `xuiId`.

Der *Component-Manager* verwaltet schließlich die einzelnen zur Verfügung stehenden Oberflächen-Templates (z. B. `table`, `tree`, `button`, `comboBox`). Das `table`-Template beispielsweise bekommt die Beschreibung zwischen `<table> ...</table>` übergeben und erzeugt daraus dynamisch die notwendigen Java-Instanzen mit Hilfe der Swing-Bibliothek, um die beschriebene Funktionalität in der Oberfläche abzubilden.

Die Menge der möglichen Templates definiert eine Konfigurationsdatei. Diese wird beim Programmstart eingelesen und jedes Template registriert sich dynamisch *Component-Manager*. Dies hat zum einen den Vorteil, dass pro Applikation konfiguriert werden kann, welche Templates notwendig sind (spart Ressourcen), und zum anderen ermöglicht es die Verwendung von neuen Komponenten ohne die Notwendigkeit, das Framework zu ändern bzw. zu erweitern.

Weiterführende Konzepte

Eine komplexe Oberflächenbeschreibung lässt sich auch mit Xui nicht in einer einzigen Datei übersichtlich darstellen. Des Weiteren lassen sich in Applikationen oft wieder verwendbare GUI-Teile identifizieren. Aus diesem Grund bietet das Xui-Framework die Möglichkeit, eine Beschrei-

bung in mehrere Dateien zu zerlegen und über eine Komponenten-Identifikation einzubinden. Um eine Eindeutigkeit der Bezeichnungen über mehrere Dateien hinweg zu vermeiden, zwingt das Include-*Tag* in einer XML-Beschreibung zur Vergabe eines Präfixes. Des Weiteren ermöglicht das Include-*Tag* die Definition einer beliebigen Controller-Instanz, an die die Methodenaufrufe der Datei geleitet werden. Dieses Konzept ermöglicht zum einen ebenfalls die Aufteilung der Controller-Funktionalität in mehrere Teile und bringt zum anderen weitere Flexibilität bei der Wiederverwendung von einzelnen Xui-Beschreibungen.

Eine Oberfläche soll sich dem Anwender gegenüber intuitiv und konsistent verhalten. Vor allem die Konsistenz wird im Wesentlichen bereits durch das Xui-Framework bereitgestellt und durch das Styles-Konzept vervollständigt. Eine Style-Deklaration definiert standardisierte Attributwerte einzelner Komponenten und kann von einer bereits bestehenden Style-Deklaration erben. Es gibt die Möglichkeit, einem Style sowohl ganze Blöcke als auch jede einzelne Beschreibung zuzuordnen. Das oben definierte Include-*Tag* erlaubt die Ausgliederung dieser Deklaration in eine gesonderte Datei. Eine Style-Dekaration mit dem Namen computed könnte beispielsweise bedeuten, alle Felder rechtsbündig mit der Schriftart bold auszurichten. In diesem Fall genügt es, bei den entsprechenden Komponenten diesen Style anzugeben – damit sind die entsprechenden Attribute implizit gesetzt. Dieses Konzept erhöht die Übersichtlichkeit und ermöglicht globale Einstellungen für das konsistente Aussehen einer Applikation.

Bei den bisher vorgestellten Konzepten war das Xui-Framework stets die aktive Komponente, die in dem passiven Controller Methoden aufruft. Der Controller hat aber auch die Möglichkeit, die Oberfläche zu steuern und Veränderungen in der Struktur und Form der Oberfläche vorzunehmen. In dem oben erläuterten Beispiel ist es zum Beispiel notwendig, eine neue Zeile in die Tabelle einzufügen. Für solche Fälle stellt das Xui-Framework das Event-Konzept zur Verfügung. Die Methodenaufrufe in der Xui-Beschreibung referenzieren vom Controller definierte Events. Feuert der Controller einen Event, werden die entsprechenden Methodenaufrufe neu ausgeführt, wie z. B. Tabelleninhalt neu einlesen oder den Text im Eingabefeld neu bestimmen. Damit hat zwar der Controller keinen unkontrollierten direkten Zugriff auf die Oberflächen-Komponenten, er kann aber ereignisbasiert Änderungen an der Oberfläche vornehmen. Die konkreten weiteren Aktionen hängen dann wiederum von der entsprechenden Xui-Beschreibung ab.

Für die Flexibilisierung arbeitet Xui intensiv mit Methodenaufrufen (sowohl bei Aktionen als auch für die Anzeige von Texten). Jeder Methodenaufruf kann grundsätzlich mit einem Fehler (*Exception*) vorzeitig abbrechen. Das Xui-Framework ist der Initiator dieser Methodenaufrufe und übernimmt aus diesem Grund die technische Fehlerbehandlung. Das Framework leitet die eigentliche fachliche Fehlerbehandlung an den zuständigen Controller weiter, der dafür entsprechende Methoden zur Verfügung stellt. Dieses Konzept stellt zum einer sicher, dass alle Fehler abgefangen und protokolliert werden, und ermöglicht zum anderen die Trennung zwischen Funktionalität und Fehlerbehandlung.

Fazit und Ausblick

Der erfolgreiche Einsatz des Xui-Frameworks demonstriert die Nützlichkeit domänenspezifischer Sprachen für transformationelle oder sogar durch die Interpretation von Hochsprachen unterstützte Entwicklung softwarebasierter Systeme. Erfolgreiche Beispiele zeigen, dass die Generierung von Code aus speziell entwickelten Sprachen sich nicht nur für grafische Oberflächen, sondern genauso für eingebettete Systeme auszahlt.

Allerdings lohnt sich die Eigenentwicklung einer solchen Sprache und des dazugehörigen Generators nur, wenn eine Wiederverwendung in ähnlichen Projekten gegeben ist. Xui ist geeignet für alle Projekte, deren Zielplattform Java-Swing unterstützt, und kann deshalb dort eine deutliche Verbreitung finden. Die Firma Beck et. al. hat deshalb beschlossen, das von ihr entwickelte Xui unter eine öffentliche Lizenz zu stellen und im Netz verfügbar zu machen (vgl. [Bea]). Damit können nicht nur Entwickler bei Beck et. al. von Xui profitieren, sondern auch andere Wünsche zur Weiterentwicklung umsetzen.

Besonders erfolgreich wird eine Xui-basierte Entwicklung immer dann, wenn sie methodisch geeignet unterstützt wird. Es hat sich gezeigt, dass sich eine separate Spezifikation für die GUI kaum mehr lohnt, weil das Xui es erlaubt, schneller zu realisieren als papierbasiert zu spezifizieren. Entsprechend ist für die GUI eine agile modellbasierte Vorgehensweise (vgl. [Rum04]) höchst interessant, auch weil sie nebenher eine einfache Generierung von Spezifikationsanteilen für die Dokumentation und Nutzerschulung erlaubt. ■

Literatur & Links

[Bea] Beck et. al., Xui-Framework, siehe: www.bea.de
[Brö99] P. Brössler, J. Siedersleben, Software Technik, Hanser, 1999
[Fra03] D. Frankel, Model Driven Architecture, John Wiley & Sons, 2003
[Kle03] A. Kleppe, J. Warmer, W. Bast, MDA Explained, Addison-Wesley, 2003
[OMG] Object Management Group, Model Driven Architecture, siehe: www.omg.org/mda
[Rum04] B. Rumpe, Agile Modellierung mit UML, Springer Verlag, 2004
[Sie04] J. Siedersleben, Moderne Software-Architektur, dpunkt Verlag, 2004