

JavaTMmagazin

Internet & Enterprise Technology

www.javamagazin.de



Auf Magazin-CD:

- Innovation Gate
- WebGate Anywhere 3.1
- visual rules 3.0 Beta
- JBoss-Special inkl. JBoss 4.0.1, JBoss AOP 1.0.0, JBossCache 1.2 etc.
- Tools, Testversionen & Samples

Alle Infos ab S. 45

JBoss 4

- Open Source J2EE Power
- JBoss-Clustering mit JGroups

UML 2

Geprüfte Qualität im UML-Modell

Neu: EJB-Corner

Infos aus der EJB 3 Expert Group

TomC@-Kolumne

Tomcat im Cluster-Einsatz

Oracle 10g

J2EE-Container OC4J

Java Web Services

Web Services und JAX-RPC Handler



XQuery-Praxis
Implementierungen & Tools



4 194586 706503

Ausleihe per pleXX

■ VON TOBIAS VOLLMER

Bei der Entwicklung von Client-Applikationen werden häufig Informationen und Integritätsregeln aus dem Business Object Layer in die Client-Anwendung dupliziert. Dieser Artikel zeigt, wie mithilfe der Framework-Suite pleXX von eXXcellent solutions Modellinformationen konsistent und durchgängig im Client zu Prüfungen und zur Visualisierung verwendet werden können.

Der inzwischen weit verbreitete Ansatz zur Modellierung von Geschäftsobjekten und Generierung der Persistenzschicht hat eine starke Automatisierung gegenüber der klassischen Entwicklung im Backend-Bereich gebracht. Ganz im Gegensatz dazu findet im Bereich der GUI-Erstellung bisher nur wenig Automatisierung statt – häufig werden Client-Anwendungen noch herkömmlich auf Basis von Frameworks wie Java-Swing mit GUI-Buildern erstellt. Generative Ansätze stellen sich oft als zu unflexibel bei detaillierten Anforderungen dar.

Einen hybriden Ansatz verfolgt hier die Framework-Suite pleXX [1]. Mit dem MDA Framework pleXX Business [2] können Geschäftsobjekte zusammen mit komplexen Integritätsregeln sowie eigener Business-Logik modelliert und daraus ein Business Layer generiert werden.

pleXX Presentation unterstützt die schnelle und konsistente Erstellung von Rich- und Web-Client-Anwendungen auf Basis der modellierten Geschäftsobjekte und den Frameworks Swing sowie wingS [3]. Aus dem Objektmodell erzeugt der pleXX Generator ein Metamodell, welches alle im Modell definierten Elemente abbildet und den Zugriff zur Laufzeit gestattet. Mächtige, erweiterbare GUI-Komponenten nutzen dieses Metamodell, um Komponenten darzustellen und um Konsistenzprüfungen sowie Synchronisation mit dem

Modell on the fly zu erledigen. Wie dies im Detail funktioniert, zeigen die nächsten Abschnitte.

Objektmodell

Zunächst wird das Modell aus Abbildung 1 im MDA-Framework pleXX Business modelliert. Dabei werden Java-Interfaces mit den entsprechenden Properties erstellt und semantische Informationen wie Integritätsregeln (Constraints) [4], Labels sowie Kardinalitäten in Form von Javadoc-Tags annotiert. Listing 1 zeigt das Interface für einige ausgewählte Properties des Geschäftsobjektes *Kunde*.

Anschließend wird das Objekt *Kunde* durch den Stereotypen *business object* (Zeile 2) dem Framework als Geschäftsobjekt bekannt gemacht. Der Name des Kunden wird durch das Javadoc Tag *@mandatory* als Pflichtfeld deklariert. Außerdem sind eine Längenbeschränkung des Attributs (*@minMaxLength*) sowie eine Beschränkung auf Zeichen des Alphabets (*@regExp*) durch den regulären Ausdruck *[A-Za-z]** definiert. Das Tag *@label* bestimmt die Klartextbezeichnung.

Constraints, welche Integritätsregeln über mehrere Objekte hinweg prüfen, können über das Tag *@constraint* an allen Pro-

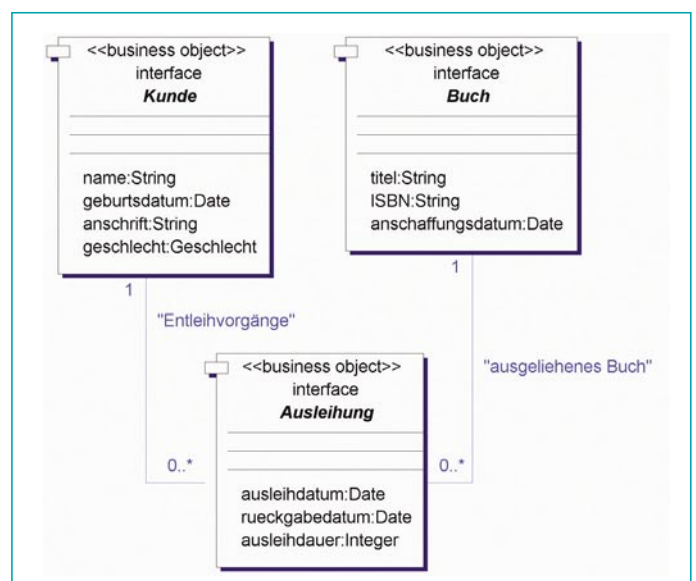


Abb. 1: Objektmodell einer Bücherei

perties definiert werden, welche auf die Prüfung Einfluss haben. Solche Constraints können entweder als Java-Klasse oder in der Syntax der OCL (Object Constraint Language) definiert werden. In eckigen Klammern können der Zeitpunkt sowie die Reihenfolge der Prüfung näher spezifiziert werden. Standardmäßig werden die Constraints sofort beim Setzen der Property geprüft, der in Zeile 10 definierte Constraint soll am Transaktionsende (*ON_COMMIT*) geprüft werden. Dabei werden zunächst alle Constraints mit Wert 0 ausgewertet, danach diejenigen mit Wert 1 usw. Ab Zeile 15 wird die Beziehung zum Geschäftsobjekt *Ausleihung* zusammen mit der Kardinalität der Beziehung definiert. Ein Kunde kann mehrere Ausleihungen haben, was durch das Tag (*@supplierCardinality*) beschrieben wird.

Aus diesem Modell wird mit dem pleXX Generator eine Zwischenschicht generiert, die auf einem O/R Mapper (Toplink, Casator, zukünftig auch Hibernate) aufsetzt und als Abstraktionsschicht zur Trennung von Client- und Business-Logik dient. Zusammen mit einer schlanken Runtime-Bibliothek stellt diese Schicht unter anderem folgende Funktionalitäten bereit:

- automatische Prüfung der definierten Constraints zu allen benötigten Zeitpunkten
- Verwaltung von historisierten Objekten [5]
- Information von Change Listeners (an Objekten, Transaktionen, global)
- automatische (Neu-)Berechnung von abgeleiteten Attributen (z.B. eine Summe aller Rechnungsposten)

Für eigene Business-Logik können Methoden wie in Zeile 22 modelliert werden. In der generierten Schicht werden hierfür abstrakte Methoden erzeugt, die anschließend in einer abgeleiteten Klasse implementiert werden müssen. Als Folge enthalten die generierten Artefakte keinen selbst geschriebenen Code und können jederzeit verworfen und neu generiert werden.

Implementierung des Clients

Zur Verwaltung des soeben konstruierten Modells soll nun eine Client-Anwendung mit pleXX Presentation erstellt werden.

pleXX Presentation setzt auf dem GUI-Framework Java-Swing sowie dem Open-Source-Framework wingS (Swing für Webapplikationen) auf, die beide ein sehr verwandtes Programmiermodell besitzen.

Zielsetzung des Ansatzes von pleXX Presentation ist, Standardfälle in Verwaltungsapplikationen wie Übersichten, Listen, Formulare und Ansichten mit sehr geringem Aufwand bei hoher Usability und Einheitlichkeit umsetzen zu können. Mittel dazu sind mächtige Komponenten mit Standardverhalten, die zur Individualisierung mehrstufig konfigurierbar sind. Soll Spezialverhalten implementiert werden, so kann dies weiterhin völlig frei mit den Mitteln von Swing bzw. wingS geschehen. Die Implementierung der Client-Anwendung geschieht nun in drei Schritten:

1. Anbringen von Zusatzinformationen für den Client im Modell
2. Generierung eines Meta-Modells zur Verbindung der Client-Implementierung mit dem Modell
3. Implementierung der Tabellen, Formulare usw.

Zusatzinformationen im Modell

Das pleXX-Modell wird in diesem Schritt mit Zusatzinformationen ausgestattet, die den Client betreffen. Ein Beispiel für das Geschäftsobjekt *Kunde* findet sich in Listing 2. Die Annotation *ON_ADJUSTMENT_FINISHED* (Zeilen 7, 8, 9) an den Constraints bedeutet, dass die Client-Anwendung den Constraint beim Verlassen des Feldes im GUI automatisch prüfen soll. Alternativ könnten hier *IMMEDIATE* (Prüfung nach jedem Tastendruck), *ON_COMMIT* (Prüfung am Transaktionsende) beziehungsweise *ON_DEMAND* (Prüfung im Client wird manuell angestoßen) stehen. Weiter werden Bezeichnungen für die Attribute für verschiedene Anzeigesituationen definiert – *@columnName* für die Spaltenbezeichnung in Tabellen, *@hoverText* als Tooltip sowie *@mnemonic* als Shortcut Key. Daneben kann optional definiert werden, zu welchen Zeitpunkten der Abgleich zwischen Model und View (und umgekehrt) stattfinden soll. Beispielsweise könnte dies durch *@modelToViewSyncMode* *ON_INIT* definiert werden. Häufig macht es jedoch mehr Sinn, diese Eigenschaften

Listing 1

```

1 /**
2  * @stereotype business object
3  */
4 public interface Kunde {
5  /**
6  * Kompletter Name inklusive Vor- und Nachname
7  * @mandatory
8  * @maxLength(3,50)
9  * @regexp("[A-Za-z]*")
10 * @constraint NameOderGeburtsdatumUndAnschrift
                                [ON_COMMIT,0]
11 * @label "Name"
12 */
13 public String getName();
14 ...
15 /**
16 * Die Entleihvorgänge, die dieser Kunde bisher
                                getätigt hat.
17 * @associates Ausleihung
18 * @clientCardinality 1
19 * @supplierCardinality 0..*
20 * @label „Entleihvorgänge“
21 Collection getAusgelieheneBuecher();
22 Integer berechneAnzahlAusleihvorgaenge() ;
23 }

```

Listing 2

```

1 /**
2  * @stereotype business object
3  */
4 public interface Kunde {
5  /**
6  * Kompletter Name inklusive Vor- und Nachname
7  * @mandatory [ON_ADJUSTMENT_FINISHED,0]
8  * @maxLength(3,50) [ON_ADJUSTMENT_
                                FINISHED,0]
9  * @regexp("[A-Za-z]*") [ON_ADJUSTMENT_
                                FINISHED,1]
10 * @constraint NameOderGeburtsdatumUndAnschrift
                                [ON_COMMIT,0]
11 * @label "Name"
12 * @columnName "Name"
13 * @hoverText "Name des Kunden"
14 * @mnemonic N
15 */
16 public String getName();
17 ... }

```

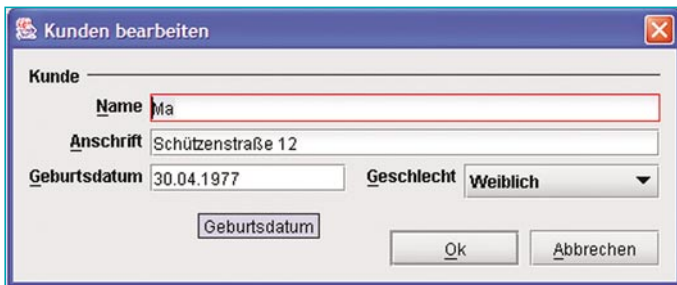


Abb. 2: pleXX-Modell eines Kunden als Java-Interface

global oder pro Formular/Tabelle zu definieren – dies wird ebenfalls unterstützt.

Ein Generator erzeugt aus den im Modell angebrachten Informationen so genannte Deployment-Deskriptoren. Dabei wird im Beispiel für die Klasse *Kunde* eine Klasse *KundeDeployment* mit Konstanten für jede Property der Klasse erzeugt. Diese Konstanten enthalten jeweils die gesamte für die Benutzeroberfläche relevante Information aus dem Modell. Durch diese Generierung kann bei der Verbindung von Client und Objektmodell auf Reflec-

tion-ähnliche Mechanismen verzichtet werden. Als Vorteil ergibt sich eine Prüfung zur Compile-Zeit, was insbesondere bei Umbenennungen von Attributen sehr hilfreich ist und den Testaufwand reduziert.

Das Ergebnis

Mit diesen Vorarbeiten sowie den mächtigen GUI-Komponenten geht nun die Arbeit der GUI-Programmierung leicht von der Hand. Listing 3 zeigt den Code für den Dialog zur Erfassung der Kundendaten, die Abbildung 2 das Ergebnis.

Hauptkomponente ist hier die Komponente *DefaultFormMultipleRepresentation*, welche ein mehrspaltiges Formularlayout bereitstellt, die übergebenen Felder automatisch platziert, Constraints zu den definierten Zeitpunkten prüft sowie die geänderten Werte zurück in das Objekt *Kunde* schreibt. Aus den beim *add()* übergebenen Konstanten werden automatisch die im Modell definierten Eigenschaften wie Label-Namen, Mnemonics usw. abgeleitet. In Zeile 18 wird definiert, wie Verletzungen von Constraints visualisiert werden sollen – in diesem Fall durch rotes Umrahmen der Komponente. Leicht könnte hier auch eine Fehlermeldung in einem separaten Bereich angezeigt werden. Abschließend wird in Zeile 22 das Objekt übergeben, welches im Formular bearbeitet werden soll.

Tabellen, wingS und mehr

Zum Erstellen von Tabellen kann die Komponente *DefaultTableMultipleRepresentation* verwendet werden, dessen API analog zum obigen Formular bedient werden kann. Abweichend werden jedoch nicht ein Objekt, sondern eine Liste von Objekten übergeben. Felder in Formularen und Tabellen können auch aus Objekten mehrerer Klassen befüllt werden. In diesem Fall würden in Zeile 22 ein Array von mehre-

ren Objekten übergeben sowie beim Aufruf der *add()*-Methode der Index in diesem Array definiert.

Weiter können auch Navigationspfade zu Properties anderer Objekte bei der *add()*-Methode definiert werden. Ein Beispiel hierzu ist eine Übersicht über alle Ausleihungen mit Namen des jeweiligen Kunden sowie des Buchtitels in einer Tabelle mit den folgenden Zeilen:

```
table = DefaultTableMultipleRepresentation
        (ListSelectionMode.SINGLE_SELECTION);
table.add(AusleihungMeta.AUSLEIHDATUM);
table.add(AusleihungMeta.RUECKGABEDATUM);
table.add(new Property[] { AusleihungMeta.KUNDE,
                           KundeMeta.NAME});

PropertyRepresentation pr =
table.add(new Property[] { AusleihungMeta.BUCH,
                           BuchMeta.TITEL});

pr.setColumnName("Buchtitel");
table.setOwnerObjects(ausleihungenListe);
```

In der dritten Spalte der dadurch entstehenden Tabelle wird der Name des Kunden durch Navigation vom Objekt *Ausleihung* zum jeweils assoziierten Objekt *Kunde* ermittelt. In der zweitletzten Zeile wird schließlich noch eine individuelle Spaltenüberschrift für den Buchtitel (anstelle von *Titel* wie bei der Property im Modell definiert) gesetzt.

Häufig werden für Objekte projektspezifische Darstellungen benötigt. Ein Beispiel hierfür sind mit Einheit behaftete Werte wie z.B. *2,5 mm²* in Abbildung 4. Für solche Fälle kann eine *PropertyRepresentation*-Klasse programmiert werden, welche den Wert geeignet in Formularen und Tabellen darstellt sowie eine Editor-Komponente dafür bereitstellt. Über eine Factory kann für alle Properties dieses Typs stets die selbst implementierte Repräsentation verwendet werden. Dadurch wird eine einheitliche Darstellung in unterschiedlichen Ansichten gewährleistet.

Soll anstelle einer Swing-Anwendung eine Webapplikation erstellt werden, so ändert sich wegen des verwandten Programmiermodells von wingS nur wenig. Obiger Beispielcode ändert sich dabei nur leicht, lediglich die Imports müssen angepasst werden. Außerdem werden Definitionen im Modell wie Mnemonics und gewisse Synchronisationsmodi von Webanwendungen nicht unterstützt. Abbildung 3 zeigt den

Listing 3

```
1 public class EditKundeDialog extends XXJDialog {
2     DefaultFormMultipleRepresentation panel = null;
3
4     public EditKundeDialog(Kunde kunde) {
5         super(XXUIDemoFrame.getInstance(), "Kunden
6             bearbeiten");
7
8         // Panel anlegen: 2-spaltiges Layout
9         panel = new
10            DefaultFormMultipleRepresentation("Kunde", 2);
11
12        // Properties des Objektmodells verknüpfen
13        panel.add(2, KundeMeta.NAME); // zweiseitig
14        panel.add(2, KundeMeta.ANSCHRIFT);
15        panel.add(KundeMeta.GEBURTSDATUM);
16        panel.add(KundeMeta.GESCHLECHT);
17
18        // Darstellung: Editierbar, bei Fehlern Komponente
19            hervorheben.
20        panel.setEditMode(EditMode.READ_WRITE);
21        panel.setConstraintViolationIndicationMode(
22            ConstraintViolationIndicationMode.EMPHASIZE_
23                COMPONENT);
24
25        // Formular mit dem Kunden befüllen
26        panel.setOwnerObject(kunde);
27        setContent(panel.getUI());
28    }
29 }
```

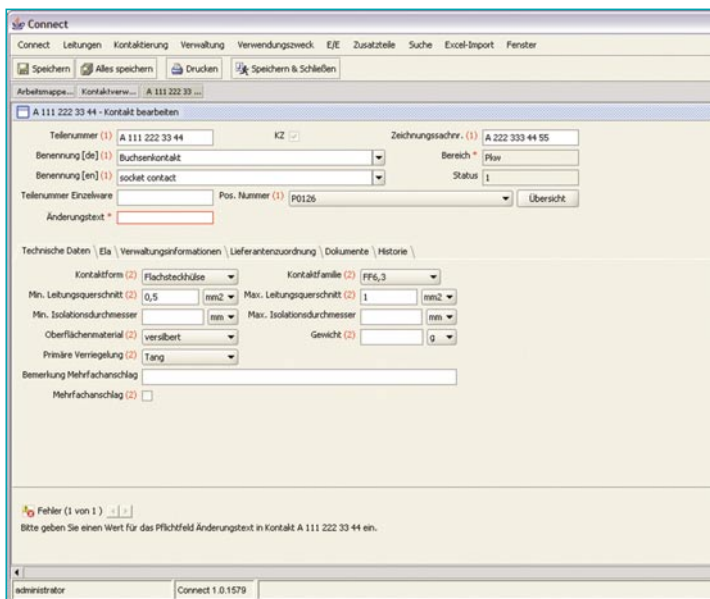


Abb. 3: Das um GUI-Informationen ange-reicherte Interface des Geschäftsobjektes *Kunde*

„Connect“, das für die DaimlerChrysler AG realisiert wurde. Im Bereich des Prototypings lassen sich voll funktionale Prototypen realisieren, die später sogar größtenteils weiterverwendet werden können. Insbesondere die wesentlichen Grundzüge des Objektmodells können bereits in der Prototyping-Phase definiert werden.

Bei größeren Projekten wirken sich insbesondere die objektübergreifenden Constraints, die garantierte Konsistenz des Objektmodells sowie die Konsistenz zwischen GUI und Objektmodell sehr positiv aus. Insgesamt ergibt sich durch die Standardisierung im GUI-Bereich sowie den generativen Ansatz bei der Implementierung der Geschäftsobjekte eine massive Reduktion der Kodieraufwände in der Entwicklungsphase. Parallel dazu und in der Wartung ergibt sich deutlich geringerer Testaufwand, da ein großer Teil der technischen Komplexität in Framework-Komponenten verlagert ist.

Verglichen mit dem Einsatz von GUI-Buildern fehlt zwar die Möglichkeit der visuellen Konstruktion. Dem stehen jedoch erfahrungsgemäß kürzere Entwicklungszeiten sowie bessere Wartbarkeit und Qualität des Codes beim Einsatz von pleXX Presentation gegenüber. Daneben ließe sich dank des einfachen Programmiermodells leicht Abhilfe schaffen. Ein interaktives Werkzeug für die Konfiguration der Komponenten sowie zum Verbinden der Komponenten mit den Geschäftsobjekten kann mit überschaubarem Aufwand implementiert werden.

Dipl.-Inf. Tobias Vollmer ist Business Consultant und Projektleiter bei eXXcellent solutions in Ulm. Neben Effizienzsteigerung und Durchgängigkeit in der Softwareentwicklung liegt ein weiterer Schwerpunkt im Bereich Performance von Java-Anwendungen. Kontakt: t.vollmer@excellent.de.

Links & Literatur

- [1] pleXX: www.excellent.de/plexx/
- [2] Achim Demelt: Ein Framework für Business-Objekte, in JavaSPEKTRUM 1.2003
- [3] wingS: wings.mercatis.de
- [4] Achim Demelt: Objektvalidierung mit Constraints, in JavaSPEKTRUM 4.2003
- [5] Achim Demelt: Temporale Datenhaltung, in JavaSPEKTRUM 9.2003

Screenshot einer mit pleXX Presentation erstellten Webapplikation.

Bei internationalisierten Anwendungen können Namen und Bezeichnungen nicht mehr direkt im Modell definiert werden, sondern müssen in sprachspezifischen Resource-Bundles abgelegt werden. Die Anwendung muss zur Darstellung eines Labels den Text aus dem Resource-Bundle der jeweiligen Sprache laden. Diesen Prozess unterstützt pleXX Presentation mit einem Generator, welcher pro Sprache ein Resource-Bundle für alle Properties des

Objektmodells erstellt bzw. aktualisiert. In diese Resource-Bundles müssen anschließend die jeweiligen Sprachtexte mit einem Editor eingetragen werden.

Erfahrungen

Der beschriebene hybride Ansatz hat sich bereits in mehreren Projekten vom Prototyp bis hin zu größeren Anwendungen (ca. 100 Datenbanktabellen) mit komplexen GUIs bewährt. Die Abbildungen 3 und 4 zeigen je ein Beispiel für Swing- und wingS-basierte Anwendungen aus dem Projekt

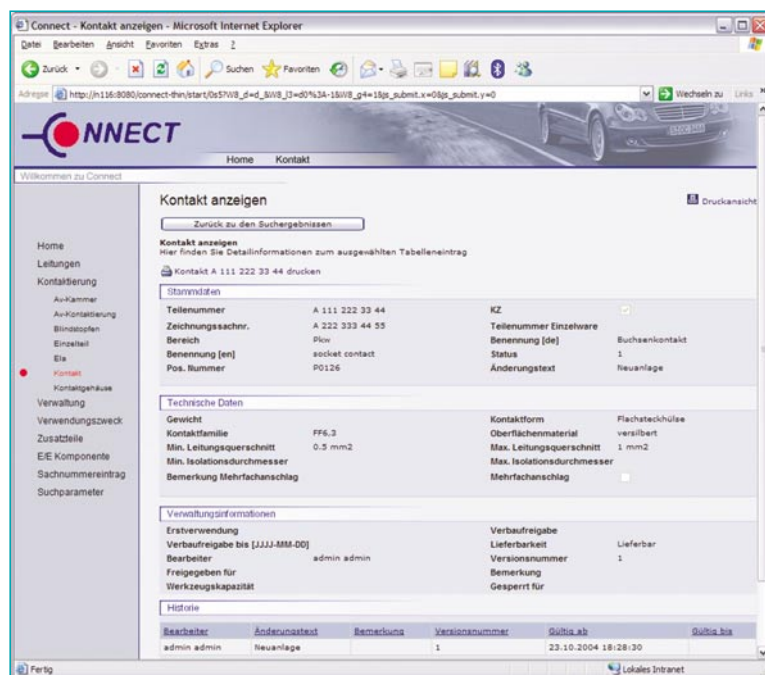


Abb. 4: Kompletter Java-Code für den unten stehenden Dialog