Vergleichstest aktueller SWT GUI Builder

Schöner entwickeln

VON PETER FRIESE

Mit der Verbreitung der Eclipse Rich Client Platform wächst auch der Bedarf an visuellen GUI-Designern – die manuelle Erstellung von GUIs ist einfach nicht produktiv genug. Am Markt tummeln sich mittlerweile einige Produkte, die um die Gunst der Entwickler buhlen. Zeit für eine Bestandsaufnahme.

Als Eclipse im November 2001 als Open-Source-Projekt freigegeben wurde, stellte es sich noch als IDE-Plattform mit einer bereits damals recht vielversprechenden Java-IDE dar. Im Laufe der Zeit hat sich Eclipse mehr und mehr zu einer universellen Plattform für alles Mögliche entwickelt, doch eine Besonderheit ist geblieben: Das ursprünglich nur als Beispiel gedachte Java Development Toolkit (JDT) enthält seit Anbeginn keinen GUI-Designer. Wer also GUIs entwickeln will, ist seit jeher darauf angewiesen, mehr oder weniger viel Oberflächen-Code per Hand zu schreiben – unabhängig davon, welches GUI Toolkit (AWT, Swing oder SWT) man einsetzt. Interessanterweise hat der fehlende GUI-Designer immer wieder Anlass zu religiösen Grabenkämpfen zwischen den Jüngern der unterschiedlichen Java-IDEs geführt. Die meisten Java EE-Entwickler haben für solche Scharmützel nur ein mildes Lächeln übrig gehabt – schließlich wurden in den letzten Jahren hauptsächlich Browser-basierte Applikationen entwickelt. Mit der Renaissance der Java Rich Clients (an der die Eclipse Rich Client Platform einen großen Anteil hat) auch im Enterprise-Umfeld wurde der Ruf nach einem vernünftigen GUI-Designer immer lauter. Etliche Hersteller und Einzelpersonen haben seitdem versucht, die Lücke zu füllen. Die aktuellen GUI-Editoren für SWT GUIs sollen hier untersucht und miteinander verglichen werden. Folgende GUI-Designer wurden bei der Untersuchung nicht näher betrachtet: V4All von Ramin Assisi – sehr populäres Plug-in, allerdings hat der Autor die Weiterentwicklung aufgrund der Verfügbarkeit des Eclipse Visual Editor eingestellt (eclipse-plugins.2y.net/eclipse/plugin_comments.jsp?id=227) und SWT GUI Builder (www.swtguibuilder.com), Standalone GUI Builder für SWT, der seit Ende 2003 anscheinend nicht weiterentwickelt wird.

Der Testparcours

Um für alle hier vorgestellten Tools die gleichen Ausgangslage zu schaffen, muss sich jedes Tool folgenden Aufgaben stellen: Zunächst soll für ein fiktives Plug-in eine View (Abb. 1) gestaltet werden. Durch diese Aufgabe lässt sich beurteilen, wie die Testanten mit einfachen Layouts umgehen können und wie gut die Unterstützung von View Actions ist. Die nächste Aufgabe besteht darin, einen Dialog mit Titelbanner (Abb. 2) zu gestalten. Hieran lässt sich sehen, wie einfach der Umgang mit Standard-Widgets ist und ob fortgeschrittene Dialogformen bereits im GUI-Editor originalgetreu dargestellt werden. Die dritte Übung „auf der grünen Wiese" ist das Erstellen eines Form-Editors (Abb. 1). Hier lässt sich sehen, ob und wie gut das Eclipse Forms API unterstützt wird.

Nach diesen Übungen muss jedes Tool beweisen, wie gut es mit bereits existierenden GUIs zurechtkommt. Als Teststücke mussten die Tasklist View sowie der Dialog zum Konfigurieren der Tasklist-Filter herhalten. Dazu wurden das Plug-in *org.eclipse.ui.ide* in den Workspace importiert und die Klassen *org.eclipse.ui.views.tasklist.TaskList* und *org.eclipse.ui.views.tasklist.FiltersDialog* mit dem jeweiligen GUI-Editor geöffnet.

Gerade von Hardcore-Entwicklern hört man oft Aussagen wie: „Ich schreibe mein GUI von Hand – GUI-Editoren produzieren keinen vernünftigen Code." Guter Code ist wichtig, insbesondere im Hinblick auf eine spätere Wartung eines Softwareprojekts. Die hier untersuchten Tools mussten sich also auch in dieser Disziplin messen lassen. Dabei haben wir neben der Übersichtlichkeit und Wohlgeformtheit des erzeugten Codes auch bewertet, ob und wie stark man Einfluss auf die Codegenerierung nehmen kann.

Mit zunehmender Komplexität eines Dialogs wächst die Größe der Methode *createPartControl()*. Große Methoden kann man recht effizient durch mehr-

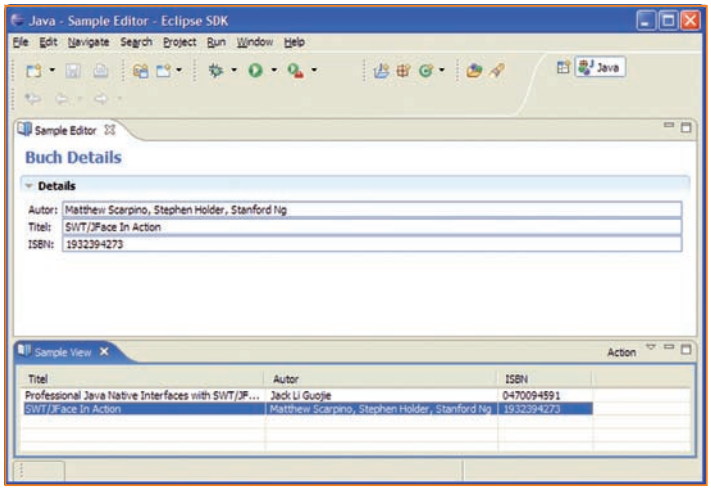

Abb. 1: Beispiel-Editor und -View

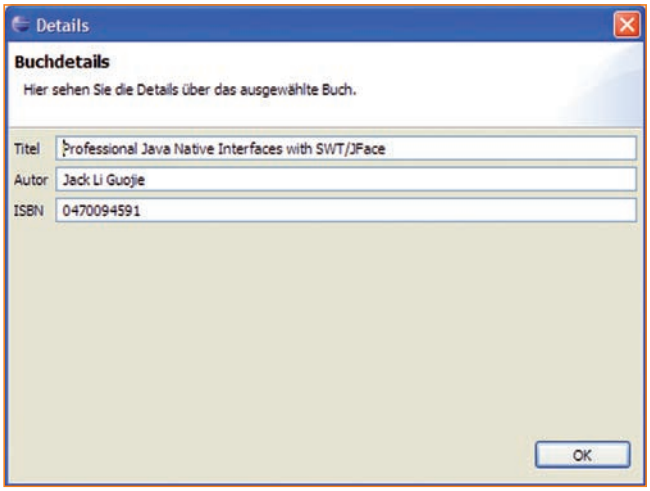

Abb. 2: Beispiel-Details-Dialog

faches Anwenden des Refactorings *Extract Method* vereinfachen. Ob der GUI-Editor das GUI dann noch ordentlich darstellen kann, hängt von der Qualität des Code-Parsers ab und ist somit ebenfalls zu überprüfen. Als Testfall werden wir zwei Spalten aus der View in Abbildung 1 in jeweils eine eigene Methode auslagern.

Ein nicht zu vernachlässigender Aspekt bei der Realisierung von Java-GUIs ist die Wahl des richtigen Layout-Managers. Durch entsprechende Programmierung kann so eine Plattformunabhängigkeit der Oberflächen erzielt werden. Gerade in diesem Bereich kann ein GUI-Designer dem Entwickler viel Zeit und Nerven ersparen – oder Kosten! Bei der Auswahl eines GUI-Editors muss dieser Punkt bedacht werden.

Eclipse Visual Editor

Der Visual Editor (VE) ist, genau wie das Web Standards Toolkit, aus dem kommerziellen WebSphere Application Developer (WSAD) hervorgegangen. IBM hat wohl erkannt, dass durch die Offenlegung des Quellcodes und die Einbeziehung der Entwicklergemeinde nicht nur direkteres Feedback zu den eigenen Tools zu erhalten ist, sondern es sich obendrein auch noch kostengünstiger entwickeln lässt – schließlich teilt man sich die Entwicklungsaufwände mit anderen Firmen. Erstmals tauchte der VE im WSAD 5.0 auf, allerdings unterstützte diese Version nur AWT und Swing. Die Unterstützung für SWT wurde erst entwickelt, nachdem das Tool unter der Schirmherrschaft der Eclipse Foundation veröffentlicht wurde.

Die Installation verläuft problemlos und folgt dem von anderen Plug-ins bekannten Schema: In das Eclipse-Verzeichnis auspacken und schon kann's losgehen. Allerdings hat der VE Abhängigkeiten zu GEF und EMF, sodass die Runtimes dieser beiden Komponenten vorliegen oder noch installiert werden müssen. Alle Komponenten gemeinsam benötigen knapp 13 MB Plattenspeicher.

Die Erstellung und Bearbeitung von GUIs erfolgt im Visual Editor (VE) in einem WYSIWYG-Editor. Der zum GUI gehörende Quellcode wird unterhalb der WYSIWYG-Ansicht im Java-Editor geöffnet und kann dort auch bearbeitet werden. Wer lieber den kompletten Editorbereich für die WYSIWYG-Ansicht bzw. den Code-Editor reservieren möchte, kann in den Preferences die Ansicht als Tabbed-Editor konfigurieren. Der WYSIWYG-Editor enthält eine Komponentenpalette, auf der die verfügbaren SWT Widgets selektiert werden können. Die Struktur eines GUI zeigt VE in der Beans View an, die Eigenschaften der einzelnen Oberflächenelemente kann man dann in der Properties View bearbeiten. Interessanterweise werden die einem Widget zugewiesenen Events nicht in der Properties View angezeigt (so wie es bei vielen anderen GUI-Editoren üblich ist), sondern als Kindknoten des entsprechenden Widget in der Beans View. Vorteilhaft an dieser Darstellung: Man gewinnt schnell einen Überblick über die Events eines GUI.

WYSIWYG-Ansicht und Code-Editor sind synchronisiert, sodass Änderungen im Code sofort auch in der WYSIWYG-Ansicht sichtbar werden und umgekehrt. In der Praxis funktioniert das auch ganz gut, allerdings ist der Parser nicht der schnellste. Aus diesem Grund gibt es einen PAUSE-Button, mit dem der Parser gestoppt und später wieder gestartet werden kann. Leider ist der Parser recht schnell durch einfache Refactorings aus der Ruhe zu bringen – extrahiert man beispielsweise den Code für die Erzeugung einer Tabellenspalte, so wird die Spalte nicht mehr angezeigt. Allerdings wird im VE die Methode *createPartControl()* nicht so groß, da VE den GUI-Code für Container (wie z.B. *Composite*, *Group*, *TabFolder* und andere) bereits in separate Methoden auslagert. Der Code ist also komponentenorientiert organisiert, was der Übersichtlichkeit dient. Erfreulicherweise beachtet VE bei der Codegenerierung auch die im JDT definierten Code Templates, sodass man von einem sehr guten Code sprechen kann.

Die Gestaltung von GUIs, die SWT-Kernkomponenten wie z.B. Button, Label, Text und Tabellen verwenden, ist kein Problem und geht locker von der Hand. Der restliche Testparcours stellte den VE allerdings vor einige unüberwindbare Hürden: Die Erzeugung von JFace-Viewern stellte sich als problematisch dar. Hier gibt es keine native Unterstützung,

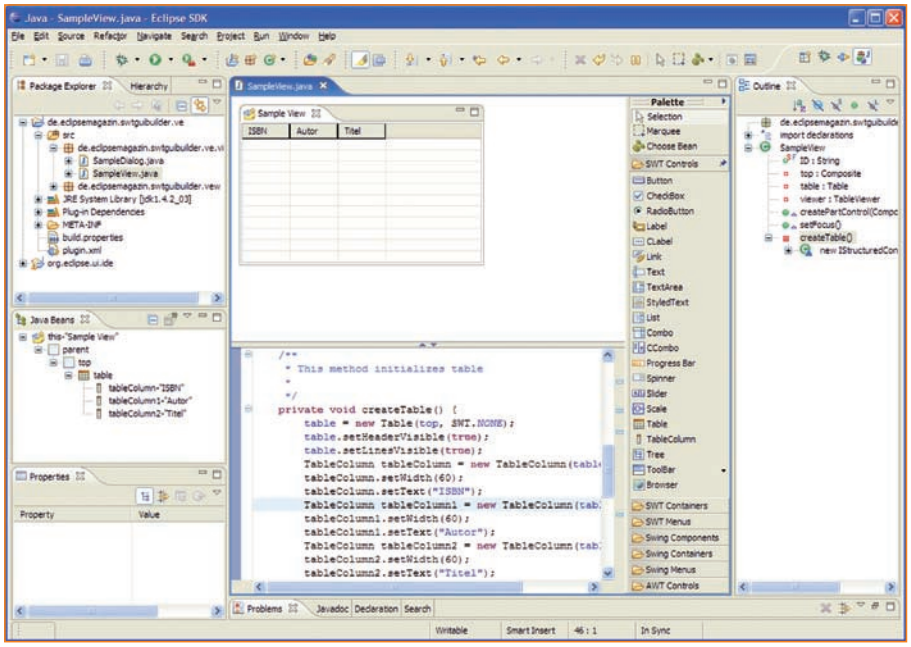

Abb. 3: Eclipse Visual Editor

sodass man zunächst das vom Viewer zu kapselnde Widget erzeugen muss, um dann manuell im Quellcode den entsprechenden Viewer anzulegen. Auch Content bzw. Label Provider muss man selbst schreiben, was im Endeffekt nicht so tragisch ist – allerdings wäre es gerade für Einsteiger hilfreich, wenn die JFace-Unterstützung weiter ausgebaut würde. Ebenfalls unmöglich ist es, einen von *TitleAreaDialog* abgeleiteten Dialog zu erstellen – Dialoge müssen im VE direkt von *Shell* abgeleitet werden. Gleiches gilt für das Eclipse Forms API, das in der Version 1.1 des VE ebenfalls nicht unterstützt wird. Eine Unterstützung für Forms, JFace sowie Dialoge ist für die Version 1.2 geplant [1] [2] [3]. Obwohl die Entwickler des VE viel Mühe in die Entwicklung ihres Code Parser stecken und sich zum Ziel gesetzt haben, möglichst viele Code-stile zu unterstützen, kann der VE die beiden ausgewählten Eclipse-eigenen Views nicht anzeigen.

Wer eigene Komponenten entwickeln und in den VE einbinden will, kann dies ohne große Probleme tun: Alle von Composite abgeleiteten GUI-Elemente können mittels *Choose Bean* ausgewählt und im GUI-Editor platziert werden. Sollen diese Komponenten auch auf der Palette erscheinen, so ist etwas mehr Arbeit nötig [4].

Die Arbeit mit den verschiedenen Layout-Managern wird im VE sehr gut unterstützt: Selektiert man ein neues Steuerelement und fährt dann mit dem Mauszeiger über einen von einem Layout-Manager verwalteten Bereich, so wird die zukünftige Einfügeposition durch einen Einfügebalken hervorgehoben. Auch die Steuerung der Ausrichtung innerhalb der Layoutzellen kann sehr elegant mit dem *Layout Window* gesteuert werden.

SWT Designer

Von der Firma Instantiations, die vielen noch aus Visual Age-Tagen bekannt sein dürfte, stammt das Produkt SWT Designer. Es handelt sich hierbei um das Mitglied einer Produktfamilie, die aus Swing Designer, SWT Designer und Window Builder Pro besteht. Wie die Namen schon andeuten, können mit den Designer-Varianten GUIs für die jeweiligen GUI Toolkits erzeugt werden. Window Builder umfasst die Funktionalität der beiden Designer-Varianten. Alle Produkte sind sowohl auf Basis einer kommerziellen Lizenz (mit der Möglichkeit, einen Wartungsvertrag abzuschließen) sowie in einer abgespeckten freien Variante erhältlich. Welche Funktionen der Anwender nutzen kann, wird durch das Einspielen einer Lizenz festgelegt – die Binaries sind für alle Mitglieder der Produktfamilie gleich. Die Installation gestaltet sich ähnlich reibungslos wie bei den meisten anderen Plug-ins: Das Plug-in wird als Zip-Datei heruntergeladen und in das Eclipse-Verzeichnis entpackt – fertig. Anscheinend hält Instantiations das eigene Tool für so selbsterklärend, dass man die Hilfe nur als separaten Download anbietet. Nach der Installation belegt der SWT Designer etwas weniger als acht MB Festplattenspeicher, für die Hilfe kommen noch mal knapp neun MB dazu. Genau wie auch Eclipse VE basiert SWT Designer auf dem GEF, die richtige Version von GEF (und alle anderen benötigten Bibliotheken) ist jedoch in der Zip-Datei enthalten und wird automatisch installiert. Nach dem Neustart der IDE muss noch eine Lizenz eingespielt werden, dies kann direkt aus dem Preferences-Dialog des SWT Designer erfolgen.

Das Gestalten von Oberflächen geht mit dem SWT Designer recht zügig von der Hand, die Bedienung ist intuitiv. Neue Oberflächenelemente (d.h. Dialoge, Views und Editoren) können mithilfe eines Wizard angelegt werden. Die Bearbeitung der Oberfläche findet in einem kombinierten WYSIWIG/Code-Editor statt. Links und rechts der WYSIWIG-Ansicht befinden sich die Komponentenpalette und der Property-Editor (Abb. 4). Das Aussehen des Editors kann in weiten Teilen angepasst werden, zum Beispiel kann die Quellcodeansicht auf einem separaten Reiter oder Seite an Seite mit der WYSIWYG-Ansicht dargestellt werden. Die WYSIWYG-Ansicht und der Java-Editor sind als Zwei-Wege-Tool gekoppelt. In den Tests waren beide Ansichten immer synchron, jedenfalls bei selbst erzeugten Klassen.

Den Testparcours absolvierte der SWT Designer größtenteils problemlos: Views, Dialoge und Editoren erzeugt und bearbeitet man recht elegant, alle SWT-Komponenten werden unterstützt. Besonders hervorzuheben ist die gute Unterstützung für JFace: Alle JFace Viewer stehen als eigenständige Komponenten in einer separaten Rubrik auf der Komponentenpalette zur Verfügung und können mittels Drag & Drop auf das GUI platziert werden. Content und Label Provider erzeugt man durch einen Doppelklick auf die entsprechende

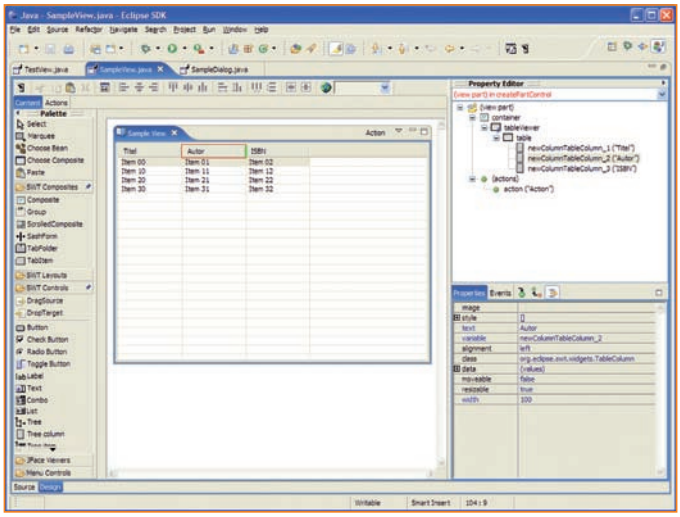

Abb. 4: SWT Designer

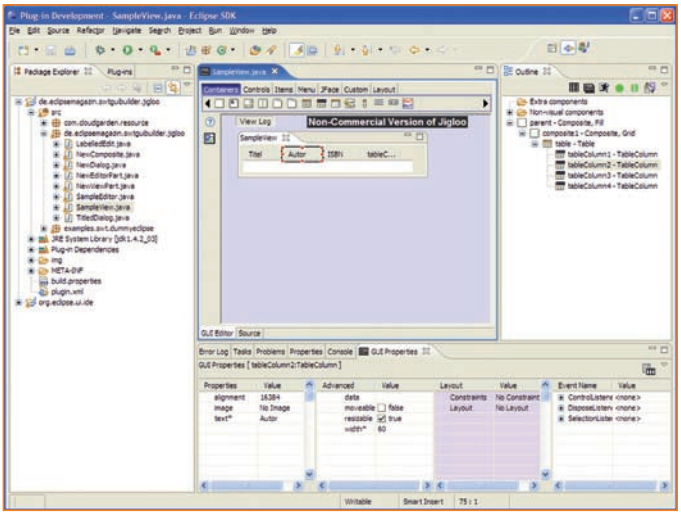

Abb. 5: Jigloo

Eigenschaft im Property-Editor. Für die Bearbeitung von Actions gibt es eine eigene Seite auf dem GUI-Editor, was die Bearbeitung von Views erheblich erleichtert.

Das Forms API [6] [7] wird ebenfalls unterstützt, sodass dem Erstellen von GUIs auf Basis dieser neuen Technologie nichts im Wege steht. Die Darstellung von GUI-Code, der manuell oder mit einem anderen Tool erstellt wurde, stellt GUI-Editoren immer vor erhebliche Schwierigkeiten. Obwohl der Code Parser des SWT Designer gute Arbeit leistet, gerät er bei fremdem Code an seine Leistungsgrenze. Die Tasklist View kann er nur rudimentär darstellen und der Versuch, den Tasklist-Filter-Dialog zu öffnen, endete mit einem Internal Parser Error.

Beim GUI-Code-Refactoring wiederum kann man relativ weit gehen, das Auslagern der Erzeugung einzelner GUI-Blöcke tangiert die Darstellung des GUI nicht – ganz wie gewünscht. Im Gegensatz zum VE ist der Code leider nicht von Anfang an nach Komponenten organisiert, sodass man hier tatsächlich des Öfteren zum Refactoring greifen muss. Ansonsten lässt sich der Codegenerator sehr detailliert einstellen, u.a. kann man Präfixe bzw. Suffixe für die einzelnen Komponenten festlegen. Darüber hinaus lässt sich pro Komponententyp einstellen, ob ein Feld oder eine lokale Variable zur Erzeugung des Widgets genutzt werden soll.

Als Besonderheit unterstützt der SWT Designer so genannte Templates – wer dahinter aber etwas wie zusammengesetzte Komponenten vermutet, liegt falsch. Es handelt sich hierbei nur um die Möglichkeit, die neuen Standardwerte für die Eigenschaften einzelner Widget-Klassen festzulegen. Dies kann durchaus praktisch sein, wenn man hintereinander viele gleichartig formatierte GUI-Elemente einfügen möchte. Eine weitere interessante Besonderheit ist die Unterstützung von visueller Vererbung, der SWT Designer stellt sowohl für Subklassen von Composite als auch von Shell alle GUI-Elemente der jeweiligen Superklassen dar. Dieses Feature kann man ausnutzen, um auf die Schnelle eigene grafische Komponenten zu entwickeln. Ein Texteingabefeld mit Beschriftung ist so innerhalb weniger Minuten zu realisieren. Solche selbst entwickelten Komponenten können über den *Preferences*-Dialog zur Komponentenpalette hinzugefügt werden, sodass man auf einfache Art und Weise ein Repository dieser Komponenten aufbauen kann.

Der SWT Designer enthält eine sehr gute Unterstützung für die SWT Layout Manager: Die Einfügeposition wird grafisch hervorgehoben und ebenso wie beim VE existiert ein Layout Assistant, der bei der Konfiguration der Ausrichtung und Ausdehnung der einzelnen Komponenten hilft.

Jigloo

Von allen hier vorgestellten Plug-ins ist Jigloo von Cloudgarden das genügsamste – zumindest, was den Speicherbedarf auf der Festplatte angeht. Nach der gewohnt einfachen Installation (auspacken – fertig) ist das Eclipse-Verzeichnis um circa 2,3 MB angewachsen. Die Online-Hilfe wird gleich mitinstalliert, ist allerdings wenig brauchbar. Für den nichtkommerziellen Einsatz ist Jigloo kostenlos und ohne Funktionseinschränkungen verwendbar, kommerzielle Lizenzen sind für 75 US-Dollar erhältlich.

Wie bei den beiden anderen Tools legt man neue visuelle Klassen mithilfe eines Wizard an. Leider bietet der Wizard nicht alle möglichen visuellen Klassen an, sodass man sich hier per Hand behelfen muss. Der GUI-Editor selbst kann je nach Vorliebe entweder im Split-Pane-Modus oder als Tabbed Pane angezeigt werden. Jeweils eine Hälfte des Fensters beherbergt den Java-Editor, während die andere Hälfte vom GUI-Editor belegt wird. Eine Art Komponentenpalette stellt die verfügbaren Komponenten am oberen Bildrand dar. Zur Bearbeitung der Komponenteneigenschaften wird am unteren Bildschirmrand eine View eingeblendet. Die View stellt die Eigenschaften sowie die Events einer Komponente jeweils auf separaten Reitern dar. Alternativ können die verschiedenen Kategorien nebeneinander dargestellt werden. Zunächst ist diese Darstellungsform etwas gewöhnungsbedürftig, sie erweist sich aber als durchaus praktisch – vor allem bei Komponenten mit vielen Eigenschaften.

Einfache Oberflächen lassen sich mit Jigloo komfortabel erstellen, die Erstellung von komplexen Layouts ist etwas

umständlich, da es keinen Layout-Assistenten gibt und man alle Layout-Eigenschaften manuell über die GUI Properties View einstellen muss. Die SWT-Layout-Manager werden unterstützt, jedoch ist die Handhabung beim Einfügen von neuen Komponenten nicht so elegant bei dem VE und SWT Designer. Dies führt leider dazu, dass man einzufügende Komponenten gleich in der richtigen (Anzeige-)Reihefolge einfügen oder sie nachträglich mit der Maus an die richtige Position verschieben muss.

Die Unterstützung für JFace ist bestenfalls rudimentär zu nennen: Zwar werden die zentralen JFace Viewer auf eine extra Seite der Komponentenpalette angeboten, allerdings gibt es weder eine Unterstützung für Label bzw. Content Provider noch ist es möglich, die inneren Komponenten eines Viewer anzusprechen. Die Erstellung von GUIs, die auf JFace aufbauen, ist nur mit zusätzlichem manuellem Aufwand möglich, was die Arbeit unnötig hemmt.

Insgesamt entsteht der Eindruck, dass Jigloo im Bereich der moderneren GUIs Schwächen hat: So gibt es z.B. keinen Support für das Eclipse Forms API. JFace-Dialoge hingegen können prinzipiell erstellt und bearbeitet werden. Auch die Erstellung von zusammengesetzten Composites ist mit Jigloo dank visueller Vererbung problemlos möglich. Von allen hier vorgestellten Tools bietet Jigloo die beste Unterstützung für die Einbindung selbst geschriebener Komponenten auf die Komponentenpalette: Über den Preferences-Dialog kann man die Komponentenpalette frei konfigurieren und beliebige von Composite abgeleitete Komponenten hinzufügen. Die zum Testparcours gehörenden Eclipse-Dialoge (Tasklist View und Tasklist-Filter-Dialog) kann Jigloo nicht vernünftig darstellen, obwohl man den Code Parser über WINDOW | PREFERENCES | JIGLOO sogar konfigurieren kann.

Einfache Refactorings des GUI-Codes verkraftet der Parser gut, sodass einer ordentlichen Organisation des Codes nichts im Wege steht. Obwohl es diverse Optionen zur Konfiguration des Codegenerators gibt, haben diese keinen sichtbaren Einfluss auf den generierten Code. Ähnlich wie beim SWT Designer schreibt Jigloo den kompletten GUI-Code zunächst in eine große Methode (*createPartControl()* bzw. *open()*), man muss also seinen Code nachträglich durch geeignete Refactorings logisch organisieren.

Fazit

Von allen getesteten Produkten macht der SWT Designer den am meisten ausgereiften Eindruck. Der Visual Editor hinterlässt im Bereich SWT zwar einen stabilen Eindruck, das Fehlen einer Unterstützung für JFace und Eclipse Forms ist allerdings nur schwer zu verschmerzen. Man muss dem VE zugute halten, dass der Ursprung des Tools im Bereich Swing liegt und die SWT-Unterstützung erst seit ein paar Monaten existiert. Außerdem ist der VE das einzige Produkt, das als Basis für eigene GUI-Designer herangezogen werden kann – nicht nur aufgrund der Lizenzform (Open Source Eclipse EPL), sondern auch, weil es als einziges Tool über die notwendigen Extension Points verfügt. Unter [4] wird erklärt, wie man einfache Komponenten in den VE integriert, unter [5] beschreiben die Autoren ausführlich, wie auf Basis des VE ein eigenständiger GUI-Editor für ein proprietäres Application Framework entstanden ist.

Die Erstellung von Oberflächen ist mit allen drei verglichenen Tools gut zu bewerkstelligen und produktiver als das mühsame Kodieren der GUI von Hand. Dies gilt allerdings nur für Standard-SWT GUIs, sobald es an die Erstellung von moderneren GUI-Varianten (JFace, Eclipse Forms) geht, müssen sowohl Jigloo als auch VE passen.

Hersteller	Visual Editor 1.1.0.1	SWT Designer 4.1.1	Jigloo 3.5.2
URL	www.eclipse.org/vep/	www.swt-designer.com	www.cloudgarden.com
Installationsumfang	13 MB	8 MB (+ 9 MB Hilfe)	2,3 MB
GUI-Typen			
View	ja	ja	ja
Editor	ja	ja	ja
Forms Editor	nein	ja	nein
Dialog	nein	ja	ja
Titled Area Dialog	nein	ja	ja
GUI-Elemente			
Standard SWT	alle	alle	alle
JFace Viewer	geplant ab 1.2	alle	geplant ab 3.7.9
Forms API	geplant ab 1.2	alle	geplant ab 3.8.0
Bewertung			
Funktionsumfang	0	+	0
Darstellung	0	++	0
Bedienung	+	++	+
Preis	++	++	+
	frei	US-$ 199–249	US-$ 75

Tabelle 1: Testergebnisse

Peter Friese (peter.friese@lhsystems.com, f3.tobject.de) arbeitet als Softwarearchitekt bei Lufthansa Systems in Hamburg. Sein Tätigkeitsschwerpunkt liegt auf der Planung und Entwicklung von verteilten Systemen auf Basis aktueller Java-Technologien. Seine Leidenschaft gilt der Entwicklung von Tools, die das Leben für Entwickler einfacher machen. Peter ist Autor des FindBugs-Plug-ins und AndroMDA-Committer. Er ist verheiratet mit Anne und hat zwei Jungs (Sören und Lennart), mit denen er komponentenbasierte Systeme erstellt (natürlich in Lego).

Links & Literatur

[1] Eclipse Bug #64039: bugs.eclipse.org/bugs/show_bug.cgi?id=64039

[2] Eclipse Bug #100145: bugs.eclipse.org/bugs/show_bug.cgi?id=100145

[3] Eclipse VE Newsgroup: www.eclipse.org/newsportal/article.php?id=4736&group=eclipse.tools.ve#4736

[4] Gili Mendel et al.: Extending the Visual Editor Tutorial: Enabling support for a Custom Widget: dev.eclipse.org/viewcvs/indextools.cgi/~checkout~/org.eclipse.ve.examples/org.eclipse.ve.example.customwidget/WebContent/

[5] Janak Mulani, Sibylle Peter: Extend VE to build a ULC GUI Builder: www.eclipse.org/vep/WebContent/docs/VEpapers/ulc.pdf

[6] Jim D'Anjou u.a.: Eclipse Forms Programming Guide: www.jdg2e.com/forms/EclipseForms.html

[7] Dejan Glozic: Eclipse Forms: Rich UI for the Rich Client: bugs.eclipse.org/bugs/show_bug.cgi?id=106893