Warum Eclipse zu qualitativ hochwertigeren Programmen führt

On a higher ground

VON PETER FRIESE

Eclipse ist nicht nur eine erstklassige Entwicklungsumgebung, sondern bietet mit der Rich Client Platform eine ausgereifte Basis für die Entwicklung eigener Anwendungen. Im vorliegenden Artikel zeige ich auf, warum Eclipse nicht nur den Entwicklern Spaß macht, sondern ebenso dazu führt, dass der Kunde eine erstklassige Anwendung bekommt.

Als ich zum ersten Mal Kontakt mit Eclipse (damals noch Version 1.0) hatte, wollte ich zunächst nicht glauben, dass ich es hier mit einer in Java geschriebenen Anwendung zu tun haben sollte: Zu flüssig in der Bedienung, zu nativ das Aussehen der Oberfläche. „Das ist doch bestimmt in C++ geschrieben", dachte ich zunächst und war sehr überrascht, als ich herausfand, dass Eclipse wirklich fast ausschließlich aus Java-Code besteht. Die Eclipse Committer haben ihr Ziel, eine Plattform zu erschaffen, die ihren Anwendern ein Lächeln ins Gesicht zaubert, voll und ganz erreicht. Mittlerweile ist die Fülle der Features derart angewachsen, dass ich immer wieder staune, wenn ich das „New and Noteworthy" eines neuen Release lese: Es gibt jedes Mal noch mehr coole Funktionen. Für mich ist Eclipse wie ein Überraschungsei mit einem immer neuen, spannenden Inhalt. Selbst wenn man denkt, man habe alle Features eines Release gesehen, kommt schon eine neue Version heraus und überrascht von neuem. Es folgen ein paar meiner Lieblings-Features (wo nicht anders angegeben, beziehe ich mich auf Eclipse 3.1).

Portrait

Peter Friese (peter.friese@lhsystems.com, f3.tobject.de) hat sein Diplom in Wirtschaftsinformatik an der Nordakademie in Elmshorn erworben. Seit 2000 arbeitet er als Softwarearchitekt bei Lufthansa Systems in Hamburg. Seine Leidenschaft gilt der Entwicklung von Tools, die das Leben für Entwickler einfacher machen. Peter ist Autor des FindBugs-Plug-in und AndroMDA Committer. Er ist verheiratet mit Anne und hat zwei Jungs (Sören und Lennart), mit denen er komponentenbasierte Systeme erstellt (natürlich in Lego).

I can't live with or without you

Viele Entwickler, die Eclipse zum ersten Mal benutzen, suchen erst mal nach dem Compile-Button, weil sie es von den meisten anderen IDEs gewohnt sind, ihr Programm vor dem Start kompilieren zu müssen. Eine Ausnahme dürften Smalltalk-Entwickler und die Anwender von Visual Age for Java sein, für die es schon seit langem selbstverständlich ist, dass das Programm immer im ausführbaren Zustand vorliegt. Den zugrunde liegenden Mechanismus (Incremental Project Builder [1] [2][3]) kann man natürlich auch für eigene Zwecke nutzen. Etliche Plug-ins (so z.B. Websphere Application Developer) nutzen das Project Builder API, um Dateien beim Speichern auf Gültigkeit und Wohlgeformtheit zu validieren. Das FindBugs-Plug-in [4] verwendet es, um direkt nach dem Speichern einer Java-Datei den vom Compiler erzeugten Bytecode auf potenzielle Programmierfehler zu untersuchen.

Ein weiteres praktisches Feature ist der Button LINK WITH EDITOR, der von vielen Views angeboten wird. Er bewirkt, dass die entsprechende View immer die zum aktuellen Editorinhalt passenden Informationen anzeigt. Mancher mag dieses Feature insbesondere im Package Explorer abgeschaltet haben, da sonst beim Schließen eines Editors der Fokus des Explorer evtl. an eine völlig unpassende andere Stelle wechselt. Aber probieren Sie LINK WITH EDITOR doch mal in der CVS Repository View aus. Während Sie nun eine Datei im Editor bearbeiten oder im Compare Editor mit einer anderen Revision vergleichen, liest Eclipse im Hintergrund die Bearbeitungskommentare der Datei aus dem Repository ein und zeigt sie tabellarisch an.

Eine große Verbesserung seit Eclipse 3 gegenüber den Vorgängerversionen ist die Agilität der Oberfläche. Viele Verarbeitungsschritte, die früher im Vordergrund im Kontext eines modalen Fortschrittdialogs liefen, sind nun in den Hintergrund verlagert worden. Das hierzu verwendete Job API (siehe auch Artikel Seite 57) kann natürlich auch in eigenen Plug-ins genutzt werden. In meinem aktuellen Projekt (eine verteilte Anwendung mit einem Eclipse RCP Frontend) verwenden wir das Job API, um große Datenmengen, z.B. Stammdaten, im Hintergrund zu laden, während der Benutzer im Vordergrund schon mit der Erfassung von Informationen beginnen kann. Der Benutzer empfindet die Oberfläche immer als agil, was einen großen Einfluss auf die Zufriedenheit hat.

Dass Eclipse beim Editieren von Java-Code vorausschaut und zum Beispiel sich öffnende Klammern automatisch schließt, ist ja mittlerweile schon ein alter Hut. Richtig praktisch ist das Einfügen von Klam-

mern und Semikola an der richtigen Stelle. Wenn Sie z.B. gerade folgende Zeile eingegeben haben: *List list = new ArrayList(17)* und sich innerhalb der Klammer direkt nach der Zahl 17 befinden, geben Sie einfach mal ein Semikolon ein. Eclipse platziert das Semikolon automatisch nach der schließenden Klammer. Das funktioniert übrigens genauso gut mit geschweiften Klammern. Wem das nicht gefällt, der kann es in den Preferences des Java-Editors unter TYPING | AUTOMATICALLY INSERT AT CORRECT POSITION abschalten.

Schauen Sie ab und zu mal in die Online-Hilfe. Für die Workbench, das JDT und das PDE gibt es jeweils die Kapitel „Tips and Tricks" sowie „What's new". Einige der hier beschriebenen Features habe ich ursprünglich dort (bzw. auf der Eclipse-Website unter „New and Noteworthy") entdeckt.

Schattenseiten

Bei all den vielen guten und schönen Dingen, die Eclipse bietet, gibt es dennoch einige Kleinigkeiten, die noch nicht so rund sind, wie ich mir das vorstelle. Ein meiner Meinung nach sehr wichtiges Feature ist die Anzeige von Keyboard Shortcuts in Tool Tips. Mit wachsendem Funktionsumfang sind in Eclipse immer mehr Tastenkombinationen dazugekommen, die man sich unmöglich alle merken kann. Die Entwickler haben dies erkannt und das Key Assist Pop-up zur Verfügung gestellt – einfach CTRL + SHIFT + L drücken, schon sieht man alle im aktuellen Kontext möglichen Tastenkombinationen. Leider sind das so viele, dass man ganz schön suchen muss, bis man fündig wird. Ich würde mir wünschen, dass eine der nächsten Eclipse-Versionen die Tastenkombinationen auch in den Tool Tips der entsprechenden Oberflächenelemente anzeigt (so wie dies Programme wie Word und Co. übrigens schon seit langem beherrschen).

Eclipse als Vorbild

Wie ich bereits weiter oben kurz angedeutet habe, entwickeln wir in meinem aktuellen Projekt derzeit eine 3-Tier-Applikation mit einem Eclipse RCP Frontend. Eine wirklich interessante Beobachtung ist, dass die Frontend-Entwickler in besonderem Maße motiviert sind. Eclipse wird hier als eine Art Vorbild gesehen. Dies äußert sich in zweierlei Hinsicht: Zum einen haben die Entwickler eine geradezu sichtbare Freude beim Entwickeln – es vergeht fast keine Woche, in der ein Problem mithilfe einer Eclipse-Funktion ganz besonders elegant gelöst werden kann. Zum anderen spornt das durchdachte Design von Eclipse die Entwickler dazu an, ihre eigenen Designs in ähnlicher Weise zu entwickeln. Ein Beispiel: Die Darstellung von inhomogenen Hierarchien wird in Eclipse anhand einer durch Strategien (*ContentProvider* und *LabelProvider*) konfigurierbaren Outline erreicht. Durch dieses Vorgehen ist es nicht nur möglich, beliebige viele verschiedene Knotentypen in einer Baumansicht darzustellen, es ist darüber hinaus auch kein Problem, neue Knotentypen darzustellen, ohne die Outline selbst ändern zu müssen. Allerdings ist die Outline explizit dafür geschaffen worden, die innere Struktur des gerade bearbeiteten Elements anzuzeigen. Was tut man nun, wenn man unabhängig vom gerade editierten Element eine globale Struktur hierarchisch darstellen möchte? Eclipse selbst nutzt dazu den Navigator, der auf der Hierarchie aller im Workspace enthaltenen Ressourcen operiert und diese hierarchisch darstellt. Leider sind Ressourcen in Eclipse immer mit dem Dateisystem verbunden, womit der Navigator für die Darstellung von Informationen aus einer Datenbank nicht genutzt werden kann. Also musste etwas Eigenes her. Basierend auf dem Design der Outline haben wir einen Explorer entwickelt, der beliebige Knotentypen unterstützt. Die Knotentypen sowie die dazugehörigen Actions werden über Extension Points beigesteuert. Da der Explorer nicht an das Resource API gebunden ist, sondern beliebige Datenstrukturen darstellen kann, haben wir so eine flexible und universelle Lösung geschaffen.

Nachdem wir nun gesehen haben, dass die Arbeit mit Eclipse den Entwicklern Spaß macht (was wohl die meisten Leser nachvollziehen können), stellt sich doch die Frage, was der Kunde davon hat. Auch dazu ein Beispiel: Im vorliegenden Projekt muss die Anwendung vollständig tastaturbedienbar sein. Es kommt dazu, dass die Anwender unterschiedlich qualifiziert sind. Einige Anwender kennen z.B. die Kurzcodes für die am häufigsten verwendeten Entitäten und können somit die entsprechenden Eingabemasken sehr schnell ausfüllen. Andere Anwender kennen die entsprechenden Kurzcodes nicht und benötigen passende Hilfestellungen. Comboboxen scheiden aus, da die Menge der anzuzeigenden Codes einfach zu groß ist. Was liegt da näher, als die betreffenden Felder mit einer Content-Assist-Funktion auszustatten? Dass ein Feld den Anwender bei der Eingabe unterstützt, wird durch eine kleine Glühlampe links neben dem Eingabefeld symbolisiert. Der Anwender kann mit der Tastenkombination CTRL + SPACE eine Liste der möglichen Eingabewerte anzeigen lassen. Aus dieser Liste kann er mittels Pfeiltasten den gewünschten Eintrag auswählen. Alternativ tippt der die ersten Zeichen des gesuchten Eintrags ein, woraufhin sich die Liste automatisch einschränkt. Auf diese Weise werden die erfahrenen Anwender nicht bei der Dateneingabe behindert und die anderen Anwender können bei Bedarf eine Unterstützung anfordern. Das Schöne daran ist, dass die Basisinfrastruktur für Content Assist bereits in Eclipse enthalten ist. Die Aktivierung dieser Funktion für ein Eingabefeld ist mit zirka fünf Zeilen Code möglich.

Fazit

Eclipse hebt die Entwickler auf ein neues Niveau. Zunächst ist da die erhöhte Produktivität bei der Kodierung zu nennen. Es sind die vielen kleinen Features, die die Arbeit am Code so einfach werden lassen und für den Spaß an der Arbeit sorgen. Bei der Entwicklung von Applikationen auf Basis der Eclipse RCP kommt dann noch dazu, dass viele Dinge, die man beispielsweise bei Swing selbst entwickeln müsste, einfach schon vorhanden sind. Die Entwickler können auf einem höheren Abstraktionsniveau arbeiten – dies macht den Weg frei für die Fachlichkeit und sorgt so für qualitativ hochwertige Anwendungen.

Links & Literatur

[1] Eclipse Incremental Project Builder API (z.B. org.eclipse.core.resources.IncrementalProjectBuilder)

[2] Marco van Meegen: Bautrupp kostenlos. Eclipse Project Builder erklärt am Beispiel Velocity-Generierung, in *Eclipse Magazin* Vol.3

[3] Peter Friese: Entwicklungs-Turbo, in *iX Magazin* 8.2004

[4] FindBugs-Plug-in: findbugs.tobject.de