C/C++-Entwicklung mit Eclipse 3.0

Licht am Horizont

Eine Java-Plattform für C/C++-Entwicklung nutzen? Das C-Development-Tool für Eclipse macht's möglich. Die neue Version 2.0 des Plugins bringt praktische Erleichterungen wie intelligentere Code-Completion – auch wenn noch Wünsche offen bleiben. Dirk Bolte

Das C-Development-Tool (CDT) ist eine Entwicklungsumgebung für C und C++ [1]. Im Unterschied zu anderen Lösungen ist es aber ein Plugin für Eclipse (siehe [2]), das komplexe Applikations-Framework in Java. Da auch das Plugin in dieser Sprache geschrieben ist, steht es für dieselben Plattformen zur Verfügung wie Eclipse, unter anderem für Linux und Windows.

Die Entwicklergruppe setzt sich größtenteils aus Mitarbeitern der Firmen QNX und IBM zusammen, die das CDT unter derselben Lizenz wie Eclipse, der Common Public License, veröffentlicht haben [3]. Die erste Version des CDT entstand Ende 2002 für Eclipse 2. Nach einjähriger Entwicklungszeit war das Paket bei Version 1.2.1 angelangt. Parallel zu Eclipse 3.0 arbeiteten die Programmierer an Version 2.0 des CDT, das nun seit Juli 2004 verfügbar ist.

Die Entwicklung mit Eclipse ist in so genannten Projekten organisiert. Mit dem CDT verhält es sich nicht anders. So beginnt die Arbeit eines Entwicklers damit, ein neues Projekt anzulegen. Unabhängig davon, ob C oder C++ zum Einsatz kommt, unterscheidet das CDT zwischen zwei Arten von Projekten: Standard Make sowie Managed Make.

Varianten von Make

Bei einem Standard-Make-Projekt ist der Entwickler selbst dafür verantwortlich, ein Makefile zu schreiben, den Sourcecode einzufügen und die richtigen Optionen zum Kompilieren zu definieren. Dabei ist er nicht auf »make« festgelegt – die Kommandos für den Build-Prozess sind frei wählbar (Abbildung 1).

Sorgt der Entwickler selbst für die entsprechenden Build-Skripte, hat er darüber hinaus die Möglichkeit, Error- und Binary-Parser sowie die Include-Pfade für sein Projekt anzugeben. Diese Einstellungen machen auch in einem nicht komplett vom CDT verwalteten Projekt die Analyse von Fehlermeldungen oder Code-Completion möglich, also die Vervollständigung von Funktionsnamen und anderen Symbolen.

Bei einem Managed-Make-Projekt konfiguriert das CDT selbst den Build-Prozess und kompiliert automatisch jede neu angelegte Datei. Anhand der Projektkonfiguration erstellt es bei jedem Build die Makefiles neu, überschreibt also manuelle Änderungen.

Sämtliche Parameter, die der Programmierer bei einem selbst verwalteten Projekt von Hand einstellt, übernimmt bei einem Managed-Make-Projekt das CDT. So setzt es zum Beispiel Error- und Binary-Parser automatisch. Der Include-Pfad gilt gleichzeitig für den Build-Prozess und die Code-Completion. Entsprechend schlicht ist der Wizard gehalten, der ein solches Projekt anlegt. Die einzige Einstellmöglichkeit ist das gewünschte Resultat: eine ausführbare Datei, eine statische oder eine dynamische Bibliothek.

Eine Quelldatei lässt sich im CDT ohne irgendwelche Assistenten anlegen. Für C-Dateien ist das der einzige Weg, den das CDT bietet. Beim Schreiben neuer Klassen in C++ unterstützt ein Wizard den Entwickler (Abbildung 2). Mit dessen Hilfe stellt er den Klassennamen, eine Vaterklasse sowie die Namen der Quelldateien ein. Anhand dieser Eingaben generiert der Wizard eine entsprechende Header- und Quelldatei mit Rümpfen für die neue Klasse.

Fertige Dateien integrieren

Um vorhandene Dateien in ein Projekt einzubinden, bietet Eclipse mehrere Optionen: Checkout aus einem Sourcecode-Repository, Import der Dateien aus ei-

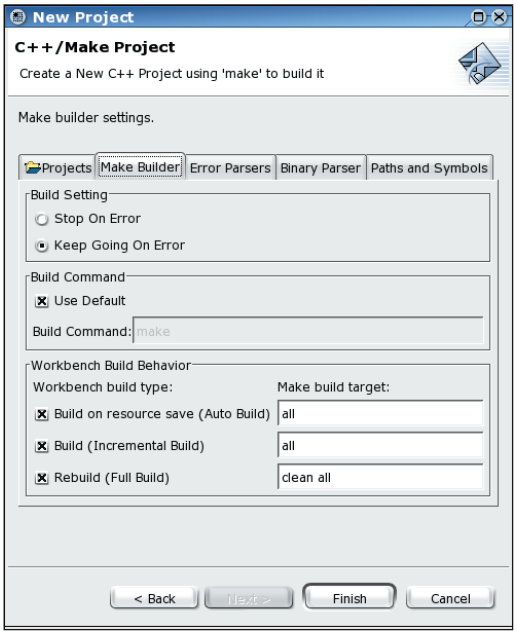

Abbildung 1: Optionen für ein Standard-Make-Projekt.

nem anderen Verzeichnis oder symbolische Links auf Verzeichnisse.

Beim Zusammenspiel mit Sourcecode-Repositories zeigt sich ein Vorteil von Eclipse als Basisplattform des CDT: Die Arbeit mit dem Repository ist völlig unabhängig von der Entwicklungsumgebung. Entsprechende Plugins extrahieren den Code direkt in ein Projekt und integrieren ihn auch wieder ins Repository. Das CDT ist an diesem Vorgang nicht beteiligt. Für CVS bringt Eclipse schon ein passendes Plugin mit. Für andere Repositories sind freie und kommerzielle Plugins verfügbar.

Auch ohne Repository lassen sich ganze Verzeichnisbäume importieren. Dabei kopiert Eclipse die Dateien direkt in das Projekt und damit in den Workspace. Alternativ zum Import kann der Entwickler Verzeichnisse auch über symbolische Links in den Workspace integrieren. Hier hilft der Wizard, der für das Anlegen eines neuen Verzeichnisses zuständig ist: Optional verweist er nämlich auf ein Verzeichnis außerhalb des Projekts. In dem Projekt taucht der Sourcecode dann so in dem neu erstellten Verzeichnis auf, als wäre er importiert.

Vereinfachungen

Ist vorhandener Sourcecode in das Projekt integriert, kann der Entwickler im Folgenden mit dem CDT programmieren, das ihn durch Code-Completion, Code-Templates und die Outline unterstützt. Sämtliche C- und C++-Elemente wie beispielsweise Klassen, Typdefinitionen, Methoden und Funktionen zeigt Eclipse in getrennten Ansichten der Outline. Von dieser Übersicht springt der Programmierer per Mausklick an die gewünschten Teile des Code. Die Outline ist in jedem Fall spezifisch für die gerade angezeigte Datei. Über den Navigator lässt sich die Outline einer anderen Datei abrufen.

Auch Code-Templates sind für den Entwickler praktisch. Es handelt sich um Grundgerüste für Standardkonstrukte wie »for«-Schleifen, »switch-case«-Anweisungen, Klassendefinitionen und vieles mehr. Wird ein Code-Template in eine Datei eingebunden, markiert das CDT die veränderlichen Elemente. Über die [Tab]-Taste springt der Benutzer zum gewünschten Element. Änderungen daran wirken sich automatisch auf dasselbe Element im gesamten Gerüst aus: So aktualisiert Eclipse bei Änderungen an Namen und Typen der Laufvariablen einer »for«-Schleife automatisch andere Stellen, an denen das Symbol auftritt.

Code-Templates sind nicht auf die im CDT enthaltene Sammlung beschränkt. Da sie in einer Makrosprache definiert sind, kann der Entwickler selbst weitere Templates schreiben oder auch importieren. Abbildung 3 zeigt die Template-Definition für eine »for«-Schleife.

Neben Code-Templates bietet das CDT kontextsensitive Code-Completion: Anhand der bereits eingetippten Zeichen eines Befehls sucht das CDT über die Include-Pfade nach entsprechenden Funktionen und Definitionen und bietet sie zur Vervollständigung an. Version 1.2.1 des CDT beschränkte sich dabei noch auf globale Elemente.

Completion im Kontext

CDT 2.0 beherrscht Code-Completion auch für C++ und vervollständigt Klassenvariablen, Namensräume, Mehrfachvererbung sowie Objekt- und Klassenmethoden. Darüber hinaus komplettiert das Plugin jetzt auch Variablen und andere Elemente aus dem lokalen Kontext. Abbildung 4 zeigt dies Feature am Beispiel einer lokal definierten Liste.

Sowohl die Code-Templates als auch die Code-Completion aktiviert man über [Strg] + [Space]. Die Auswahl der Möglichkeiten erscheint in einem kleinen Fenster direkt an der aktuellen Stelle in der Datei. Wahlweise mit Maus oder Tastatur trifft der Entwickler nun eine Auswahl – oder tippt weitere Zeichen, um so die Auswahl einzuschränken.

Neuerungen gibt es bei den Optionen, die der Entwickler beim Erstellen eines

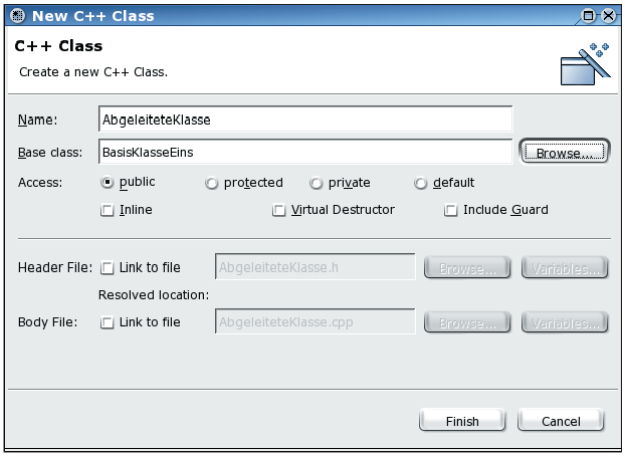

Abbildung 2: Der Wizard zum Erstellen einer neuen C++-Klasse.

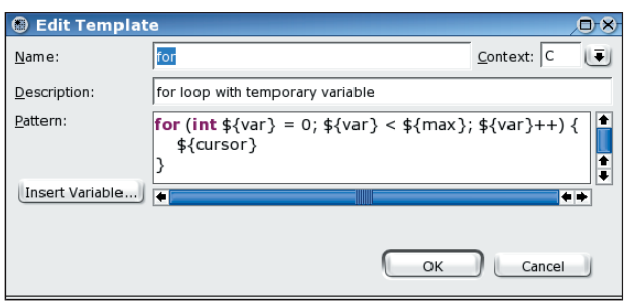

Abbildung 3: Beispiel für die Definition eines Code-Templates.

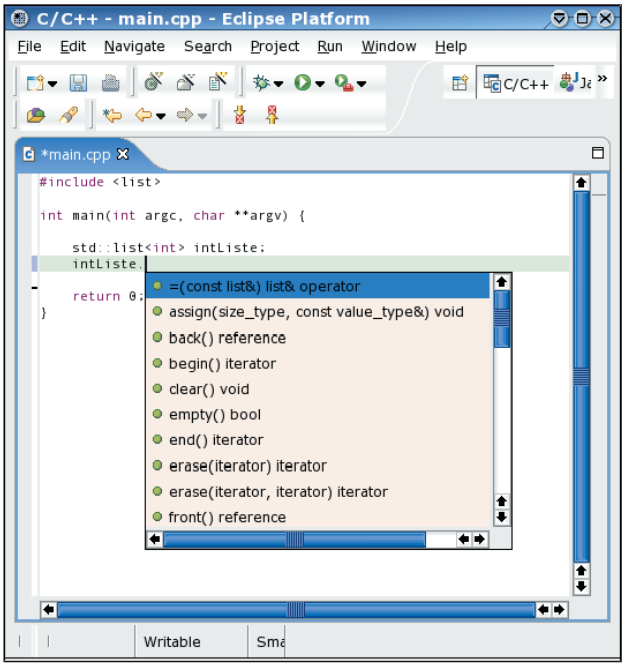

Abbildung 4: Popup für Code-Completion.

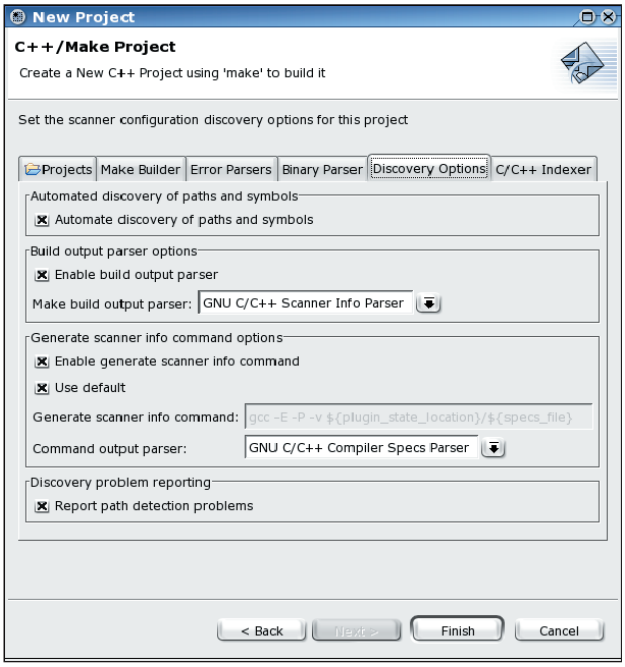

Abbildung 5: Neue Optionen für ein Standard-Make-Projekt.

Projekts hat. Zu Beginn eines Standard-Make-Projekts lassen sich Optionen zum Indizieren des Sourcecode angeben. Dies wird bei Funktionen wie Refactoring und Code-Completion eingesetzt. Ebenso sucht der Wizard jetzt automatisch nach Include-Pfaden und Symbolen (siehe Abbildung 5). Der Wizard für ein Managed-Make-Projekt bietet nun die Möglichkeit, den Error-Parser auszuwählen und Optionen für den Indexer anzugeben. Die zuständigen Reiter im Wizard entsprechen denen eines Standard-Make-Projekts.

Die Integration neuer oder bereits vorhandener Dateien hat sich nur wenig geändert. Neu sind einige Wizard-Optionen zum Anlegen neuer Klassen mit Vererbungshierarchie. Bei den Code-Templates gab es überhaupt keine Änderungen. Das neue CDT bringt dieselben Templates mit wie Version 1.2.1. Völlig neu ist dagegen Refactoring, das sich in der aktuellen Version aber auf das Umbenennen von Elementen beschränkt. Statt jeden Eintrag eines Namens nur zu ersetzen, geschieht dies beim Refactoring kontextsensitiv: Eine lokale Variable wird nur in ihrem Gültigkeitsbereich ersetzt, Methodennamen nur im Zusammenhang mit ihrer Klasse.

Ein Vergleich mit Eclipses Java Development Toolkit ist nicht ganz fair, da dieses durch Java und seine Bibliotheken einige Aufgaben leichter lösen kann, zum Beispiel durch die Reflection-Fähigkeiten und strengere Typisierung. Die Freiheiten von C und C++ machen die Integration mancher Funktionen sehr schwierig, wenn nicht gar unmöglich. Trotzdem verfügt das JDT über Funktionen, die im CDT wünschenswert wären.

Dem JDT hinterher

Zwar bringt das CDT einen Parser für Make-Fehlermeldungen und Compiler-Ausgaben mit und bindet diese auch über Markierungen im Sourcecode ein. Trotzdem fehlen Hilfestellungen und Automatismen, mit denen der Entwickler Fehler beseitigen kann. Das JDT bietet hier für jeden Fehler eine Liste möglicher Korrekturen an, die das Plugin dann selbst ausführt.

Im Bereich Code-Completion schließt das CDT langsam zum JDT auf. Beim Refactoring steht es dagegen noch am Anfang. Es fehlen Funktionen, um Methoden innerhalb einer Klassenhierarchie zu verschieben oder deren Signatur zu ändern. Das JDT bietet darüber hinaus die Möglichkeit, Codeblöcke in neue Funktionen auszulagern.

Auch eine bessere Integration der Dokumentation wäre wünschenswert. Ähnlich wie mit Javadoc im JDT könnten auch im CDT Kommentare mit Programmen wie Doxygen analysiert werden, um diese als Todo- oder Bug-Listen sowie Hilfestellungen bei der Code-Completion anzuzeigen.

Benutzbar, aber ausbaufähig

Gegenüber der letzten Version hat sich das CDT in einigen Punkten verbessert. Die Arbeit mit dem Entwickler-Plugin ist angenehmer geworden, so ist Code-Completion nun nicht mehr auf den globalen Gültigkeitsbereich beschränkt und unterstützt C++ besser.

Auf leistungsstärkeren Systemen wirkt das CDT allerdings immer noch ein bisschen träge, was den sonst positiven Eindruck etwas trübt. Es bleibt abzuwarten, ob das CDT in den nächsten Versionen zum JDT aufschließt. Schon jetzt steht aber mit der Version 2.0 eine akzeptable Entwicklungsumgebung für C und C++ zur Verfügung. *(ofr)*

Infos

[1] Eclipse CDT: [http://www.eclipse.org/cdt]
[2] Eclipse-Plugins: [http://eclipse-plugins.2y.net]
[3] Common Public License: [http://www.eclipse.org/legal/cpl-v05.html]
[4] Bernhard Bablok, „Spieglein, Spieglein, Das Java-Reflection-API“: Linux-Magazin 03/04, S. 110