C++ mit Eclipse

Eclipse-Konferenz
Heidelberg, 14.06.2005

Prof. Dr. Thomas Wieland
Fachhochschule Coburg
university of applied sciences

Die C/C++ Development Tools

- Separate Sammlung von Plug-Ins
  - Aus dem Eclipse Tools-Projektverbund: www.eclipse.org/cdt
  - Steht unter der CPL
- Seit Juli 2002
  - Ziel: Integration existierender Kommandozeilen-werkzeuge für die C/C++-Entwicklung in Eclipse
- Aktuelle Version: 2.1 für Eclipse 3.0
  - Neu: *CDT 3.0 für Eclipse 3.1* (derzeit als M6)
- Hauptakteure: QNX und IBM
  - Mehr als 35 Personenjahre Entwicklungsarbeit

Wirkung von CDT

- Sehr viele Downloads
  - Nicht mehr nur von Experten und Enthusiasten
  - Nach wie vor vorwiegend Windows und Linux
- Integration in kommerzielle Produkte
  - IBM WSDD und Rational SW Architect
  - Intel C++ Compiler
  - Montavista DevRocket
  - PalmOS Developer Suite
  - QNX Momentics development suite
  - Redhat Entreprise Linux
  - Timesys Timestorm
  - ...

Ziele des CDT-Projekts

- Erstklassiges Framework für C/C++-Werkzeuge in Eclipse
  - Plattformunabhängiges Framework, um verschiedene Entwicklungsszenarien zu unterstützten
  - Gleiche Funktionalität wie für Java (!)
- Erweiterbar und interoperabel
  - Umfangreiche Basisfunktionalität und Möglichkeiten für Erweiterungen/Ersatz
  - Definierte APIs für eine interoperable Erweiterbarkeit
  - CDT als gemeinsamer Andockpunkt für alle Werkzeuge im C/C++-Umfeld
- Kooperativ
  - Zusammenfluss der Ressourcen für C/C++-relevante Komponenten
  - Beiträge von weiteren Firmen

Zielgruppen des CDT

- Traditionelle Embedded-Entwickler
  - C/C++-Entwicklung im Host-Target-Modell (QNX, VxWorks, Linux)
  - CDT als Andockpunkt für alle Embedded-Werkzeuge
- Desktop/Server-Entwickler
  - Linux-basierte Systeme, Unix, Windows
  - Chance auf eine leistungsfähige C/C++-IDE
- "Deeply embedded"-Entwicklung
  - SW/HW Co-Design, FPGA
  - Fokus auf C, Schnittstelle zu Simulatoren
- Zielumgebungen bedingen Entscheidungen für Werkzeugketten
  - Default-CDT-Implementierung zielt auf GNU ab
  - GCC am häufigsten für Kompilierung verwendet
  - GDB als weit verbreiteter Debugger
  - Schnittstellen zur Integration anderer Werkzeugketten vorhanden

Herausforderungen für eine C/C++-IDE

- Keine Kontrolle über die Werkzeuge im Hintergrund
  - Compiler, Debugger, Werkzeugketten, Build-Verwaltung
- C/C++ komplexe Programmiersprache
  - Sehr viele Freiheiten, daher schwer zu parsen
  - Abweichungen zwischen Compilern/Plattformen
  - Generelle Komplexität von C++
- Präprocessor
  - `#defines` können im Quellcode, in Header oder in Makefile vorkommen
  - Wird für ein korrektes Parsen benötigt

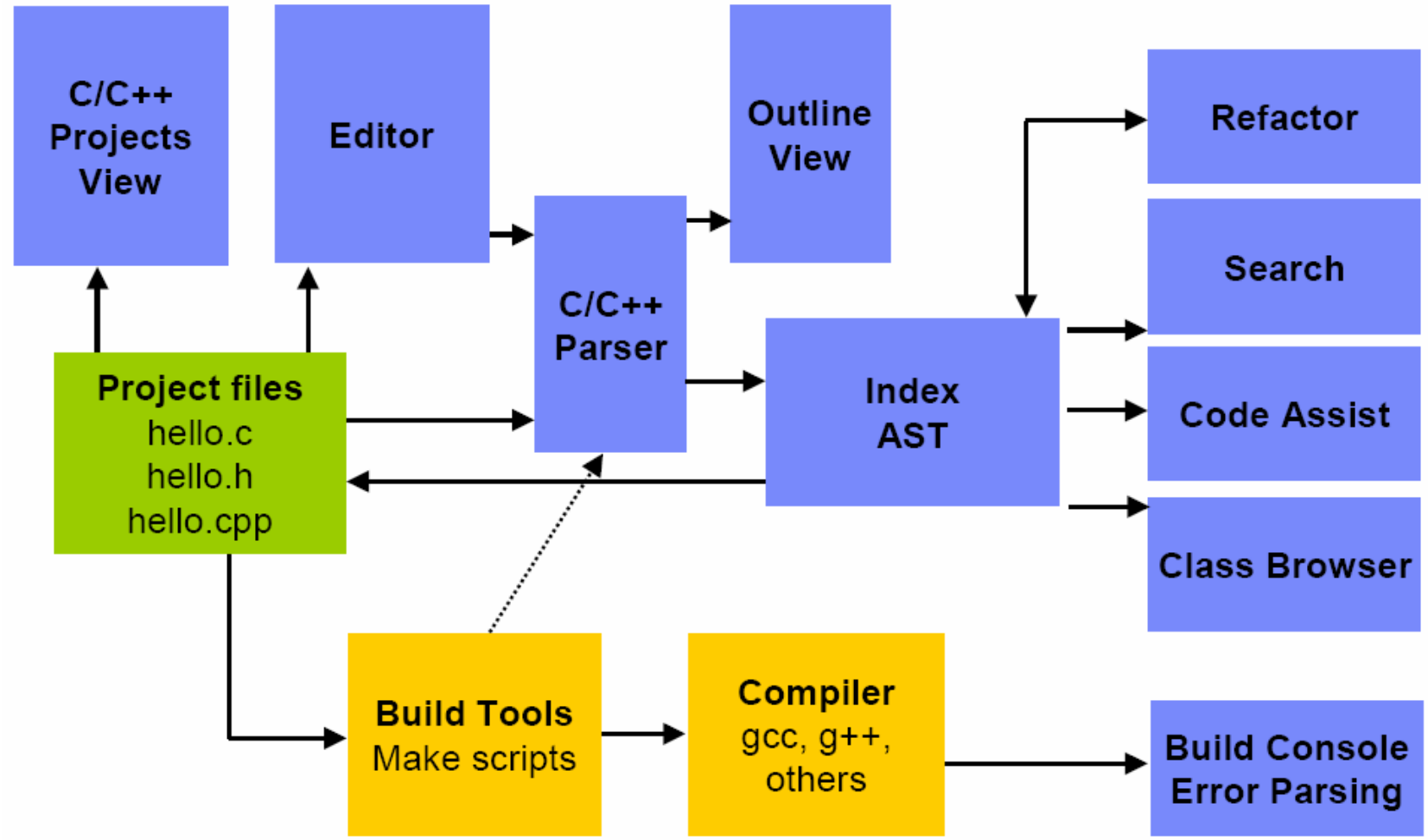
Funktionalität der CDT

- C/C++ Editor
  - Typische Funktionalität, Syntaxeinfärbung usw.
- C/C++ Debugger
  - Referenzimplementierung unter Verwendung des GDB
- C/C++ Programmstart
- Parser für Quellcode, Fehlermeldungen und Binärformat
- Suchfunktionen
- Makefile-Generator

Plattformen

- CDT laufen auf fast denselben Plattformen wie Eclipse
  - *Windows 98 und Windows ME werden nicht unterstützt!*
- Referenzplattformen:
  - Windows XP (läuft auch auf Win NT, 2000, Server 2003)
  - Linux (Red Hat WS 3, SuSE 8.2)
  - Solaris 8
  - HP-UX 11
  - AIX 5
  - MacOS X
  - QNX Neutrino

Architektur des CDT-Core



Quelle: Sebastien Marineau-Mes

Installation

- Ab Eclipse 3 über den Update-Manager
  - *Help* / *Software Updates* / *Find and Install*
  - Dort http://update.eclipse.org/tools/cdt/releases/new angeben
  - Oder als ZIP herunterladen, auspacken und dieses Verzeichnis angeben (*New Local Site*)
  - Ältere Versionen sind vorher zu deinstallieren!
- CDT enthält nur Editor, Launcher und andere Plug-Ins, keinen Compiler!!!
  - Muss separat installiert werden

Neue Versionen

- Vor Installation einer neuen Version: alte entfernen!
  - CDT enthält keinen "Uninstaller"
  - Im Verzeichnis `eclipse/plugins`: alle Ordner mit "`org.eclipse.cdt*`" löschen
  - Die Meta-Daten entfernen: In workspace/.metadata/.plugins alle Ordner mit "`org.eclipse.cdt.*`" löschen
  - Außerdem die Feature-Listen-Ordner löschen: `eclipse/features/org.eclipse.cdt.*`

Compiler und andere Tools

- Standard Compiler-Konfiguration: GCC, gdb 5.2, make
- Unter *Linux/Unix*: Meist vorhanden oder einfach installierbar
- Unter *Windows*: Cygwin oder MinGW empfohlen
  - **Cygwin** umfasst komplette Unix-Umgebung
    - Enthält auch Bibliotheken, Shell und weitere Werkzeuge
  - **MinGW**: Minimalsystem nur für die C-/C++-Kompilierung
  - Auch andere Compiler und Werkzeuge konfigurierbar, z.B. Microsoft Visual C++

Installation mit Cygwin/MinGW

- Installation von Cygwin bzw. MinGW
  - Wahlweise nur einzelne Module oder komplett
  - Pfad dabei frei wählbar
- Pfad zu Eclipse-Workspace darf keine Leerzeichen enthalten!
- Hinzufügen des bin-Verzeichnisses zum System-Pfad
  - *Arbeitsplatz -> Eigenschaften -> Erweitert -> Umgebungsvariablen*
  - Fehlermeldung: `"Build error (Exec error: Launching failed)"` <= make ist nicht im Pfad, kann nicht gefunden werden

Installation mit Visual C++

- Microsoft Visual C++ Toolkit 2003 kann kostenlos heruntergeladen werden
  - http://msdn.microsoft.com/visualc/vctoolkit2003/
  - Enthält Compiler und Bibliotheken
  - Alternativ auch vorhandenes Visual Studio 2003 nutzbar
- Eintragen der Pfade in die Umgebungsvariablen
  - PATH: Pfad zum Compiler (`cl.exe`) und Linker (`link.exe`) sowie ggf. zu den nötigen DLLs
  - INCLUDE: Pfad zu den Include-Dateien
  - LIB: Pfad zu Bibliotheksdateien

Projekt-Konfiguration mit Visual C++

- Nur mit Standard-Make-Dateien möglich
- Vorgehen:
  - Neues Standard-Make-C(++)-Projekt anlegen
  - Quelltextdateien hinzufügen und erstellen
  - Makefile hinzufügen und selbst schreiben
    - Compiler: `cl.exe`, Linker: `link.exe`, Optionen mit "/" anzugeben
    - Standardtargets: `all` und `clean`
  - Parser aktivieren:
    - *Project -> Properties -> Make Project -> Binary Parser -> PE Windows Parser*
    - *Project -> Properties -> Make Project -> Error Parser -> CDT Visual C Error Parser*
  - Aus dem Projekt-Kontextmenü *Create Make Target* wählen
    - Target: *all*, Build command: *nmake*
  - Im "Make Targets"-View doppelt auf das Target klicken

Managed Make vs. Standard Make

Managed Make:
Eclipse erstellt und verwaltet Makefile selbst

- Vorteile:
  - Keine Makefile-Kenntnisse erforderlich
  - Fehlerquellen reduziert
  - Weitere Compiler möglich
- Nachteile:
  - Flexibilität eingeschränkt
  - Nicht alle Konfigurationen sind möglich
  - Ausschluss einzelner Dateien o.ä. nur schwierig realisierbar

Standard Make:
Benutzer schreibt und pflegt Makefile

- Vorteile:
  - Volle Funktionalität von make zur Verfügung
  - Auch für komplexe Projekte geeignet
- Nachteile:
  - Makefile muss selbst erstellt werden, Kenntnisse über make-Syntax nötig
  - Zusätzlicher Aufwand
  - Nur für GCC + GNU make!

Automatischer Build

- Bei Java: Eclipse kompiliert automatisch, sobald Datei gespeichert wird
- Für C und C++ nicht sinnvoll!
- Abzuschalten unter *Project -> Build automatically*
- Dann erst sind die Build-Menüpunkte aktiviert

Hello World!

- Projekt erzeugen:
  - *New -> Project -> C++ -> Managed C++ Project*
- Quelldatei erzeugen:
  - *New -> Source File*
- Projekt übersetzen:
  - Kontextmenü: *Build Project*
- Starten:
  - *Run -> Run ...*
  - Neue Konfiguration: *New*
  - Ausführbare Datei auswählen: *Search ...*
  - Voraussetzung: Binary Parser für Zielsystem ist aktiviert (Elf bzw. Windows)

Einbinden bestehender Projekte

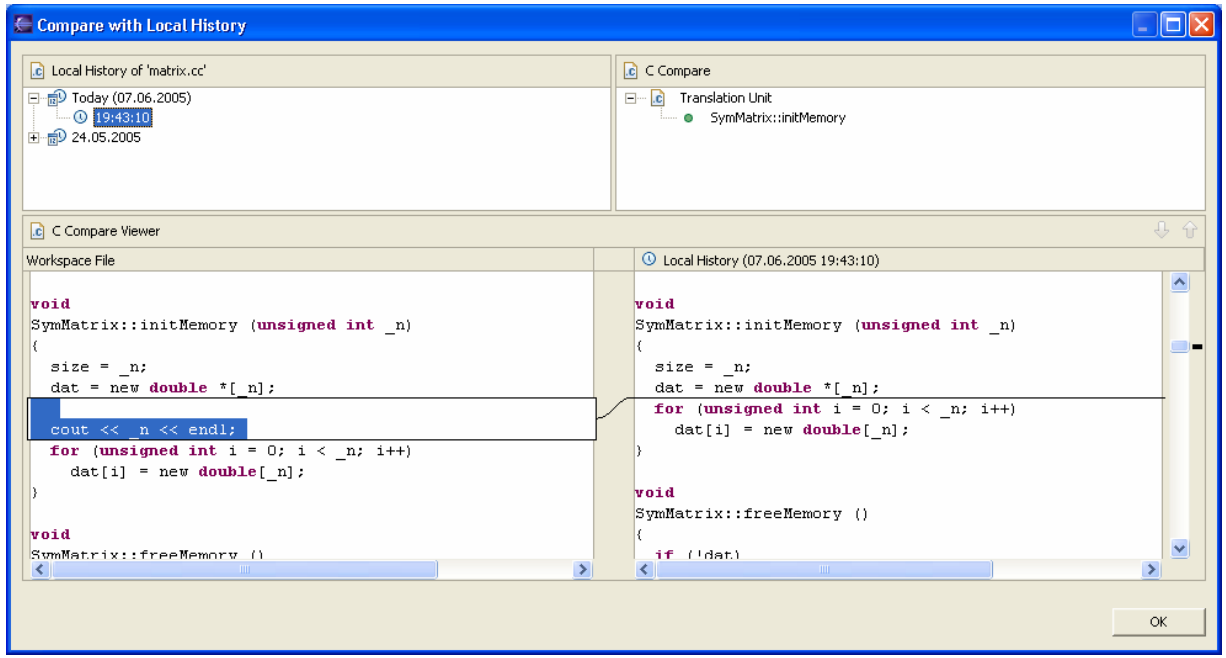
- Projekt sollte schon Makefile enthalten
- Neues Standard-Projekt anlegen
  - *New -> Project -> Standard Make C++ Project*
  - Eigenen Namen vergeben
  - Bei *Project Contents* den Haken entfernen und den tatsächlichen Pfad angeben
  - Make Targets selbst festlegen
- Projekt wird nicht in den Workspace kopiert!
  - Bleibt am ursprünglichen Ort erhalten

Tipps zum Editor

- Code-Vervollständigung: Mit Strg+SPACE oder über Kontextmenü (*Content Assist*)
  - Relevante Quelldatei muss erreichbar und kompilierbar sein!
  - Arbeitet generell in CDT 3.0 M6 sehr unzuverlässig
- Code-Schablonen: Schlüsselwort eingeben und Strg+SPACE drücken
  - Oft mehrere Varianten auswählbar
  - Liste unter *Window -> Preferences -> C/C++ -> Editor -> Templates*
- Für Navigation durch komplexe Projekte: C/C++-Browsing Perspective

Code History

- Auch ohne CVS o.ä. ist eine lokale Historie Ihrer Änderungen verfügbar
- Im Kontextmenü *Compare with -> Local History wählen*



Indexing

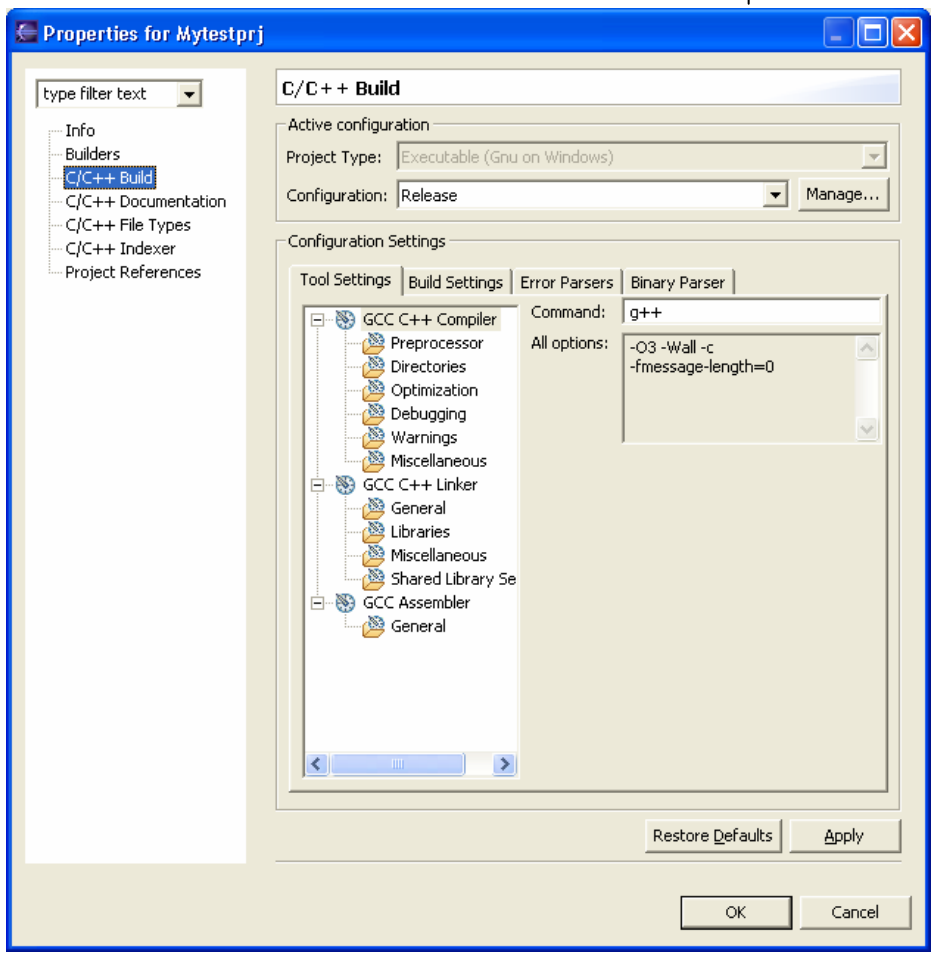
- Für die Suche muss der Indexer im Projekt aktiviert sein
  - *Project -> Properties -> C/C++-Indexer*
  - Läuft im Hintergrund bei Änderungen am Projekt und dessen Dateien
- Damit sind Suchen nach Deklarationen, Referenzen und anderen Sprachelementen möglich
  - Auch nach Klassen, Methoden, Variablen, Namespaces

Refactoring

- Noch mitten in der Entwicklung !!
- Ähnlich dem JDT
  - Aber mit einem einfacheren Framework
- Rename funktioniert bereits
  - Einfaches Suchen und Ersetzen
  - Weiter gehende Refactoring-Funktionalität in späteren Releases (auf DOM-Basis)
- Enthalten: Undo-Manager
  - Damit "undo" über Dateigrenzen hinweg möglich

Build-Konfigurationen

- Bei Managed-Make-Projekten: Debug und Release-Einstellungen möglich
  - Auch weitere selbst gewählte Konfigurationen
- Zu konfigurieren unter *Project -> Properties -> C/C++ Build*
  - Umfasst alle wichtigen Compiler-Schalter
  - Dort auch umschaltbar!



Debugging mit CDT

- Portabler Quelltext-Debugger
- Verschiedene Views, die über das Eclipse-Framework hinausgehen
  - Register, Speicher, Signale, Shared libraries
- CDI (C Debugger Interface)
  - MI-plugin-Implementierung (Schnittstelle zum GDB über eine maschinenunabhängige Schnittstelle)
  - Arbeitet mit gdb ab Version 5.2.1 zusammen
  - Ermöglicht die Entwicklung für eine Vielfalt von CPU-Architekturen
  - CDI-APIs stellen die programmatische Kontrolle über den Debugger her

Debuggen

- Integrierter Debugger auch geräteübergreifend (dann mit GDB Server)
  - Für Visual C++ nur der externe Debugger einsetzbar
    - http://www.microsoft.com/whdc/devtools/debugging
- Ausgaben können auch in Dateien umgeleitet werden
- Zusätzliche Pfade für weitere Quelldateien angebbar

Tipps zum Debuggen

- Globale Variablen: Als "Watch Expression" hinzufügen
- Anzeige der GDB-Konsole: In CDT 3.0 nicht mehr enthalten
- GDB gibt Fehlermeldung "Unrecognized option": Zu alte Version des GDB verwendet (mind. 5.2.1)
- Keine Binärdatei zum Debuggen angeboten: Falscher Binärparser eingestellt

Plugins für CDT

- *CppUnit*: Integration der CppUnit-Testumgebung
- *Eclipse-RPM*: Grafisches Werkzeug zum Erstellen von RPM-Paketen
- *Eclipse-OProfile*: Grafische Konfiguration für *OProfile*-Profiler
- *GNU-Indent*: Code-Formatierer
- Handbücher für GCC, GDB, make und binutils
- Mehr unter www.eclipse.org/cdt -> CDT Community

Ausblick

- Langfristige Ziele an Funktionalität
  - Integrierte Java- und C/C++-Entwicklung
  - Debugger außerhalb der gdb-Familie
  - Projektvorlagen
  - Parsen der Fehler und Möglichkeit des "hot fix"
  - Unterstützung für Autoconf/automake
- Verbessere Akzeptanz der CDT
  - Es wird für Softwarehäuser einfacher, Eclipse mit CDT anzubieten
  - Integration der CDT wird gefördert
- Erweitere CDT auf Bedürfnisse des Embedded-Entwicklung
- CDT soll eine Plattform werden, mit der alle C/C++-Werkzeuge beliebig gemischt verwendet werden können