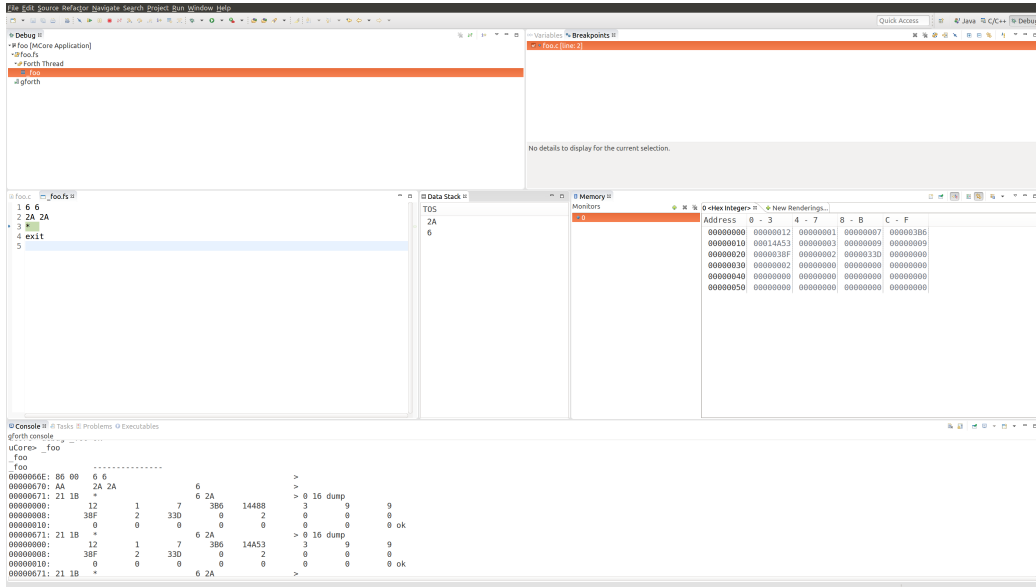


Bachelor-Thesis

Eclipse Entwicklungsumgebung für MicroCore

Benjamin Neukom

August 2015



Betreuer: Carlo Nicola

Inhaltsverzeichnis

1. Einleitung	5
2. Eclipse Platform	6
2.1. Eclipse als Platform	6
2.2. Plugins	6
2.3. Extension Points	6
2.4. Eclipse basierte MCore Entwicklungsumgebung	7
2.4.1. JDT	7
2.4.2. Xtext	7
2.4.3. DLTK	8
2.4.4. Eclipse CDT	8
2.4.5. Verwendung für MCore Eclipse	8
3. Compiler Integration	9
3.1. Integration mittels CDT	9
3.1.1. CDT Extension Points	9
3.2. Error Parsing	9
4. Forth Kommunikation	10
4.1. Prozess Kommunikation	10
4.1.1. Probleme bei der Kommunikation	10
4.1.2. GDB/MI-Commands	10
4.1.3. Direkte Kommunikation mit dem Prozess	10
4.2. API Design	10
4.3. Implementierungs Details	10
4.3.1. Kommunikation mittels Commands	11
4.3.2. Forth Output Parsing	11
4.3.3. Await auf Resultate	11
5. Debugger	12
5.1. Breakpoints in C	12
5.1.1. Per Konsole	12
5.1.2. Im Source Code	12
5.2. Konsolen basierter Debugger	12
5.3. Forth Debugger	12
5.3.1. Neue Debugger Aktionen	12
5.3.2. Stack View	12
5.3.3. Memory View	12

5.4. C Debugger	12
6. Optimierungen	13
6.1. Optimierungen	13
6.1.1. Peephole Optimierung	13
A. Literaturverzeichnis	14
B. Abbildungsverzeichnis	15
C. Tabellenverzeichnis	16
D. Ehrlichkeitserklärung	17

In dieser Arbeit wird beschrieben, wie eine IDE für MicroCore mittels Eclipse implementiert werden kann.

1. Einleitung

Todo Copy from Forth MDT

2. Eclipse Platform

In diesem Kapitel wird gezeigt, was Eclipse für Möglichkeiten anbietet, um eine moderne Entwicklungsumgebung zu implementieren. Diese verschiedenen Möglichkeiten werden verglichen und die Vor- und Nachteile aufgezeigt. Es werden auch die wichtigsten Eclipse Features, welche für das Entwickeln von Eclipse Rich Client Platform (RCP) Applikationen benötigt werden, beschrieben.

2.1. Eclipse als Platform

Eclipse RCP bietet eine Basis um beliebige, (nicht zwingendermassen Entwicklungsumgebungen) Betriebssystem unabhängige Applikationen zu entwickeln. Es bietet Mechanismen, wie Plugins und Extension Points, um modulares programmieren zu unterstützen und vereinfachen. Auch bietet das Framework Features, wie das Konzept von Views und Editoren und vielem mehr, welche häufig in Applikationen gebraucht werden.

2.2. Plugins

Eclipse Applikationen nutzen eine auf der OSGi Spezifikation basierten Runtime, eine Komponente in dieser Runtime ist ein Plugin. Eine Eclipse RCP Applikation besteht also aus einer Ansammlung von Plugins. Ein Eclipse Plugin ist ein Modul, welches gegen aussen ein API und Extension Points anbieten kann.

2.3. Extension Points

Um Plugins erweitern zu können, bietet Eclipse das Konzept von Extension Points an. Über Extension Points können Plugins eine bestimmte Funktionalität anbieten, welches von anderen Plugins aufgerufen werden kann.

So existiert zum Beispiel ein Plugin, welches einen Extension Point für Views definiert. Ein anderes Plugin kann über diesen Extension Point, deklarativ in einem XML File eine neue View erstellen. [1]

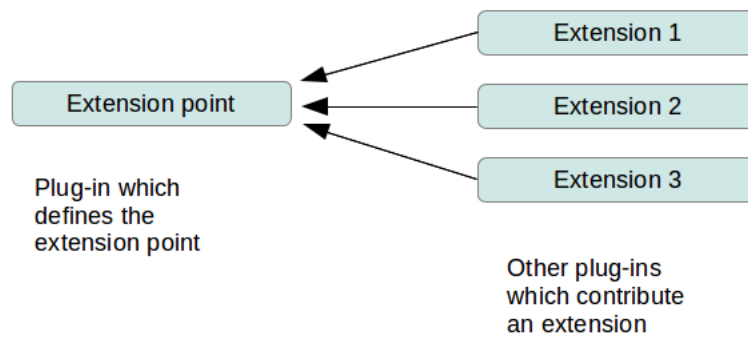


Abbildung 2.1.: Ein Plugin, welches einen Extension Point anbietet. Andere Plugins können

Es ist auch möglich eigene Extension Points zu definieren, falls ein eigens Plugin für andere Entwickler offen stehen soll für Erweiterungen.

2.4. Eclipse basierte MCore Entwicklungsumgebung

Eclipse als Grundlage für eine Entwicklungsumgebung zu verwenden eignet sich besonders gut, da Eclipse schon einiges an Funktionalität für eine IDE zur Verfügung stellt und schon viele Entwicklungsumgebungen (PHP, C/C++, Python, D) als Eclipse RCP entwickelt wurden. Auch existieren einige Tools, auf welche ich noch genauer eingehen werde, wie Xtext und DLTK, welche das entwickeln einer Entwicklungsumgebung weiter vereinfachen.

2.4.1. JDT

Eine Möglichkeit die MCore Entwicklungsumgebung zu implementieren wäre die standard Features von dem Eclipse Java Development Tools (JDT) zu verwenden. Dies wäre eine sehr Aufwändige

2.4.2. Xtext

XText ist ein Framework, welches es erleichtert eine auf Eclipse basierte Entwicklungsumgebungen zu programmieren. Es ermöglicht auf schnelle Weise ein Grundgerüst einer IDE mit Features wie:

- Ein Editor mit Syntax Coloring
- Code Completion
- Compiler Integration
- Ein Java-basierter Debugger

- Outline
- Indexing

zu generieren. [2] Es muss lediglich eine ANTLR [3] Grammatik für die Sprache definiert werden. Der grosse Nachteil ist, dass C, inklusive Preprozessor, zu parsen sehr schwierig ist und eine Entwicklungsumgebung somit auch mit Xtext nicht einfach zu implementieren ist.

2.4.3. DLTK

Das Dynamic Language Toolkit (DLTK) ist ein weiteres Framework, welches ein Grundgerüst für eine Entwicklungsumgebungen generieren kann. Ursprünglicherweise war das Framework nur für dynamische Sprachen geeignet, es kann aber auch für statische Sprachen verwendet werden. Die D Entwicklungsumgebung wurde mittels DLTK realisiert [4]. Es bestehen aber wieder dieselben Nachteile wie bei Xtext. Da C schwierig zu parsen ist, müsste trotz dem Framework noch viel selbst implementiert werden. Da D keinen Preprozessor besitzt, konnte für diese Entwicklungsumgebung das DLTK Framework verwendet werden.

2.4.4. Eclipse CDT

Das Eclipse C-Development Tools (CDT) ist eine Eclipse Distribution mit Sprach Unterstützung für C und C++. Das CDT bietet alle Features welche man von einer Entwicklungsumgebung erwartet und stellt Extension Points zur Verfügung um diese für eine eigene Entwicklungsumgebung zu gebrauchen. So kann man mit relativ wenig Aufwand einen neuen Compiler Backend in die Entwicklungsumgebung einbinden, welche den C Code kompiliert.

2.4.5. Verwendung für MCore Eclipse

Ich habe mich dazu entschieden, das Eclipse CDT als Target Platform zu wählen. Somit können alle Features, welche das Eclipse CDT zur Verfügung stellt, gebraucht werden. Frameworks, welche ein Grundgerüst einer Entwicklungsumgebung generieren, funktionieren für C nicht vollständig und sind somit keine guten Alternativen.

3. Compiler Integration

In diesem Kapitel wird beschrieben, wie der Cross-Compiler in die Entwicklungsumgebung eingebunden wurde um ein C-File nach Forth zu übersetzen.

3.1. Integration mittels CDT

3.1.1. CDT Extension Points

3.2. Error Parsing

4. Forth Kommunikation

In diesem Kapitel wird beschrieben, wie die Entwicklungsumgebung mit dem Forth Prozess kommuniziert. Die Kommunikation mit dem Forth Prozess ist von zentraler Bedeutung, da viel der Funktionalität der Entwicklungsumgebung davon abhängt. Es wird gezeigt wie die Kommunikation designt implementiert und getestet wurde.

TODO asynchronität?

4.1. Prozess Kommunikation

Um mit dem Prozess zu kommunizieren gibt es einige Alternativen welche ich aufzeigen möchte.

4.1.1. Probleme bei der Kommunikation

TODO

4.1.2. GDB/MI-Commands

TODO sehr komplex! (<http://www.ibm.com/developerworks/library/os-eclipse-cdt-debug2/complicated>)

Eine Möglichkeit mit dem Prozess zu kommunizieren wäre ein MaschineInterface wie es der GDB macht mittels MachineInterface (MI) Commands. GDB/MI ist ein Linien basiertes Maschinen orientiertes Text Interface zu dem GDB. Es wurde dazu entwickelt um den GDB als Debugger in ein grössers System einzubinden. [?]

4.1.3. Direkte Kommunikation mit dem Prozess

4.2. API Design

In einem ersten Schritt wurde ein API designt, welches verwendet werden soll um die Kommunikation mit dem Prozess möglichst einfach zu halten.

4.3. Implementierungs Details

TODO class hi

4.3.1. Kommunikation mittels Commands

4.3.2. Forth Output Parsing

4.3.3. Await auf Resultate

5. Debugger

In diesem Kapitel wird beschrieben, wie der Debugger in Eclipse integriert wurde. Es wird aufgezeigt, was für Möglichkeiten existieren einen Debugger in Eclipse zu integrieren und welche implementiert wurden.

5.1. Breakpoints in C

Als erstes müssen Breakpoints

5.1.1. Per Konsole

5.1.2. Im Source Code

5.2. Konsolen basierter Debugger

Eine einfache Implementation des Debuggers ist die Konsole zu verwenden. Da schon ein Konsolen Debugger existiert kann dieser mit dem im Kapitel 4 beschriebenen Prozess Kommunikationsmitteln angesteuert und in einer Eclipse Konsole angezeigt werden.

5.3. Forth Debugger

5.3.1. Neue Debugger Aktionen

Jump Action

Over Action

5.3.2. Stack View

User Interface

5.3.3. Memory View

User Interface

5.4. C Debugger

6. Optimierungen

6.1. Optimierungen

6.1.1. Peephole Optimierung

A. Literaturverzeichnis

- [1] Vogella. <http://www.vogella.com/tutorials/EclipseExtensionPoint/article.html>, 2013.
- [2] Xtext. <https://eclipse.org/Xtext/>, 2013.
- [3] Antlr. <http://wwwantlr.org/>.
- [4] Bruno Medeiros. D development tools. <https://github.com/bruno-medeiros/DDT>.
- [5] The gdb/mi interface. https://sourceware.org/gdb/onlinedocs/gdb/GDB_002fMI.html.

B. Abbildungsverzeichnis

- 2.1. Ein Plugin, welches einen Extension Point anbietet. Andere Plugins können 7

C. Tabellenverzeichnis

D. Ehrlichkeitserklärung

Hiermit bestätigen die Autoren, diese Arbeit ohne fremde Hilfe und unter Einhaltung der gebotenen Regeln erstellt zu haben.

Benjamin Neukom

Ort, Datum

Unterschrift