Grundlagen wissenschaftlichen Arbeitens zum Thema Programmiersprachen

Gruppe E, Nr. 4, Franz Puntigam, WS04

Concurrent Euclid

Bernhard Stockmann
Florastraße 27
2540 Bad Vöslau
Tel.: 0664 / 1066335
email: devvv@hellstorm.net
Matrikelnummer: 0325784

Inhaltsverzeichnis

Einleitung

Concurrent Euclid basiert auf der gleichnamigen Grundsprache Euclid, welche Anfang der 80er entwickelt wurde und ein Wegbegleiter im Bereich der simultanen Prozessbehandlung war. In meiner Arbeit werde ich zu Beginn kurz auf die geschichtliche Entwicklung der Sprache und die zugrundeliegende Philosophie eingehen. Anschließend erkläre ich den fundamentalen Gedanken der ,,Concurrency", zeige die Eigenheiten und Unterschiede der Sprache zu Pascal auf und werde kurz die Syntax anhand eines Beispiels erklären. Abschließend werde ich die Vor- und Nachteile von Concurrent Euclid aufzeigen.

Euclid + Concurrency = Concurrent Euclid

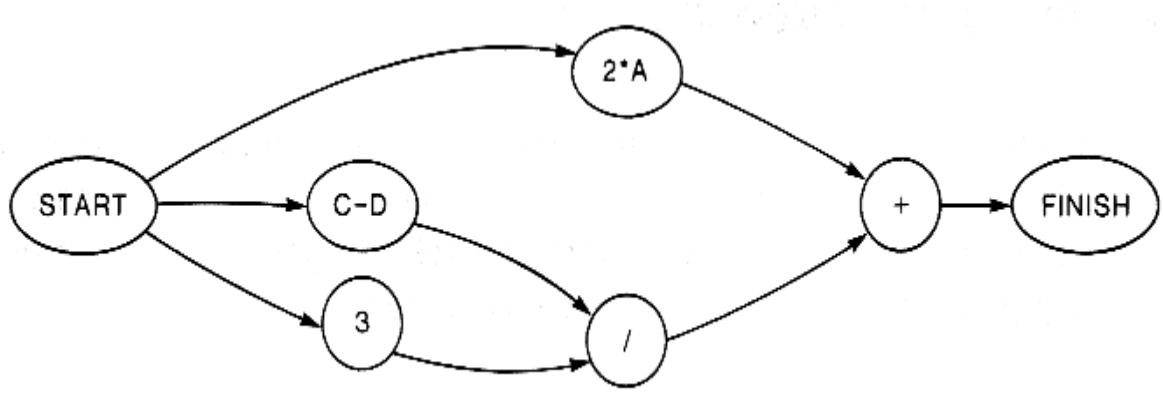
Concurrent Euclid von Richard C. Holt und J. R. Cordy ist eine Programmiersprache die für die Entwicklung von leicht überprüfbarer High-Performance-Software und (Betriebs)Systemen geschaffen wurde. Zu Beginn gab es allerdings nur die eigentliche Grundsprache namens Euclid, für die 1976 der Grundstein gelegt wurde. Die Universität von Toronto gab Ende der 70er Jahre den Startschuß für ein Projekt, das zum Ziel hatte ein komplettes, Unix-kompatibles Betriebsystem auf der Basis von Euclid (damals war die Sprache neu) zu entwerfen, welches alle Vorzüge der zukunftsweisenden Technologie der "Multiprozess-Handhabung" haben sollte. Es entwickelte sich eine Nebenform der Sprache namens Toronto Euclid. Erst die laufende Arbeit an dem neuen Betriebssystem, welche übrigens von der Abteilung für innere und nationale Sicherheit der Vereinigten Staaten von Amerika tatkräftig unterstützt wurde, führte 1981 zu einer sehr erweiterten und umfangreicheren Form, nämlich Concurrent Euclid. Die wichtigste Ergänzung war hierbei die Implementierung der "Concurrency", also eine Möglichkeit gleichzeitigen Zugriff auf unterschiedliche Prozesse zu haben.

Die Philosophie hinter Euclid

Die wichtigste Idee hinter der Grundsprache Euclid ist die Sicherheit und Überprüfbarkeit des Systems. Diese Philosophie impliziert jedoch auch eine erweiterte Zuverlässigkeit und Verständlichkeit des Codes, weil die Programmiersprache äußerst streng gehalten wurde. Das heißt, dass der Euclid-Compiler keine Fehler verzeiht. Es wird starker Wert auf die Richtigkeit der Eingabe gelegt. Jedoch wurde gerade deswegen ein extrem leistungsfähiger Compiler entwickelt, der alle nur erdenklichen Fehler ausgeben sollte, um die Programmierarbeit nicht zu einer Qual werden zu lassen.

Das Prinzip der Concurrency

Wie eben beschrieben wurde die Grundsprache Euclid um ein wesentliches Element erweitert, welches näherer Betrachtung bedarf. Concurrency bedeutet in Verbindung mit der Programmierung, dass mehrere Teile eines Programms gleichzeitig, und nicht hintereinander, ausgeführt werden können. Ein gutes Beispiel dafür wäre eine simple Rechnung. **(2*a) + ((c-d) / 3)** könnte sequenziell, d.h. eine Operation nach der anderen, ausgeführt werden, einfach wenn man das Produkt, die Differenz, den Quotient und die Summe bildet (in genau der genannten Reihenfolge). Die parallele Ausführung kann jedoch angewendet werden weil einige Ausdrücke der Rechnung unabhängig von den anderen sind. [1]



Die Technik, die beim Parallelrechnen zur Anwendung kommt nennt sich "multiprogramming". Es erlaubt ganz einfach, dass mehr als ein Programm eines Benutzers auf einmal ausgeführt werden kann. Das System muß seine Resourcen bzw Rechenkraft genau auf die verschiedenen Programme aufteilen. Vor allem die CPU beherrscht diese Technik, indem für jeden Prozess eine virtuell schwächere CPU zu Verfügung gestellt wird. Dieses Verfahren nennt man "Time Slicing". [2] Die zwei wichtigsten Vorzüge sind also

- die effiziente Auslastung der Hardware und
- die schnellen Antworten auf die Auftragserledigung für den Anwender

Die Basis: Pascal

Der Aufbau und die Syntax der Sprache ähnelt sehr stark der von Pascal. R. C. Holt, der Mitentwickler von Concurrent Euclid, meinte dazu:

"Euclid basiert auf Pascal und deren eleganten Datenstrukturen. Unterschiedliche Features von Pascal wurden "veredelt" um einfachere Überprüfungsmöglichkeiten zu gewährleisten; zum Beispiel werden in Euclid und CE Funktionen vor Nebeneffekten geschützt, in dem diese der Compiler garnicht erst zulässt. CE kann man also als bereinigte Pascal-Version sehen, welche aber zusätzlich einige Features zur Systemprogrammierung spendiert bekommen hat." [1]

Die grundlegenden Features, die Concurrent Euclid von Pascal unterscheiden sind:

- **Unabhängige Kompilierung.** Hierbei können Prozeduren, Funktionen und Module separat von einander kompiliert werden und später verbunden werden.

- **Module.** Ein Modul ist ein systaktisches Datenpaket mit Prozeduren und Funktionen, mit denen man auf Daten zugreifen kann.

- **Concurrency.** Monitoring und Prozesse werden hierbei unterstützt. Es gibt ein Signal- und Warte-Statement. Ein "Busy"-Statement erlaubt Concurrent Euclid als Simulations-Sprache verwendet zu werden.

- **Volle Kontrolle über den Gültigkeitsbereich.** Namen von Variablen werden nicht automatisch im Gültigkeitsbereich vererbt. Stattdessen werden Import- und Exportlisten zur Kontrolle von diesem verwendet.

- **Konstrukte zur Systemprogrammierung.** Diese inkludieren Variablen mit fixen, absoluten Adressen. Solche Variablen können Geräte-Register bei Computern sein, die über Speicherabbilder bezüglich Input/Output verfügen.

Einige der Features von Pascal werden übrigens nicht unterstützt. Beispielsweise aufzählbare / enumerative Typen. Weiters erlaubt CE keine Prozeduren und Funktionen, die innerhalb anderer Prozeduren oder Funktionen verschachtelt sind.

Die Syntax der Sprache

Wie Sie gleich sehen werden ist der Code fast identisch mit dem von Pascal. Um nun trotzdem zu zeigen wie Concurrent Euclid aufgebaut ist gebe ich ein kleines Programm vor und erkläre es anhand des Codes.

```
1   var Example:
2       module
3           include'%IO1'
4           { Print Characters up to a period }
5           initially
6               imports(var IO)
7               begin
8                   var ch: Char
9                   IO.PutString( ' Test starts$N$E ' )
10                  loop
11                      IO.GetChar(ch)
12                      IO.PutChar(ch)
13                      exit when ch = $.
14                  end loop
15              end {of initially}
16      end module
```

In Zeile 1 wird der Name des Programms definiert, hier "Example". Zeile 2 und 16 sind in CE fundamental, da in Concurrent Euclid ein Programm das Format eines Moduls haben muss. Zeile 3 importiert ein Paket namens %IO1, welches für Ein- und Ausgabe nötig ist. In Zeile 4 sehen wir einen Kommentar, welcher in CE in geschwungenen Klammern geschrieben wird. Die Sterne (* *) wie in Pascal als Kommentar zu verwenden ist nicht erlaubt. Zeile 5 und Zeile 15 fürhen das Programm aus und beenden es wieder. Nun folgt der eigentliche Code. In Zeile 6

wird das IO-Modul verwendet und als Variable importiert. D.h. das Modul wird umgewandelt, was technisch möglich ist weil es die Input/Output-Dateien enthält, welche wiederum von Puts verändert werden.

In Zeile 8 bis 14 geschieht zuerst eine Deklaration eines Zeichens (var ch:Char) welches in Zeile 11 eingelesen und in Zeile wieder 12 ausgegeben wird. Zeile 9 gibt einen einfachen Text aus (" Test starts ") und beginnt eine neue Zeile. Die loop-Anweisung ist eine einfache Schleife, wie wir sie alle kennen. Sie wird beendet, wenn das eingelesene Zeichen ein Zeitbereich ist. Das Beispiel ist hiermit bereits beendet. [4]

Ein Kurzresumee:

Pro & Contra Concurrent Euclid

Abschließend kann man sagen, dass Concurrent Euclid vor allem Im Bereich der Systemprogrammierung Fuß fassen konnte, wenngleich es sich auch nie wirklich durchgesetzt hat. Das Angebot an systemnahen Programmiersprachen war zu groß und CE wurde zu wenig von der Masse verwendet.

Vorteile:

- Systemprogrammierung
- einsetzbar zur Programmierung von High-Performance-Software
- Vorteile der Concurrency
- guter Compiler
- einfach zu Grunde liegende Basisprache Pascal

Nachteile:

- etwas veraltet und nicht aktuell
- verzeiht keine Fehler bei der Programmierung
- wenig Dokumentation

Literaturhinweise

[1] Concurrent Euclid, The Unix System and Tunis, Addison Wesely Publishing Company, Seite 2, Absatz 4

[2] Concurrent Euclid, The Unix System and Tunis, Addison Wesely Publishing Company, Seite 4, Absatz 2

[3] Concurrent Euclid, The Unix System and Tunis, Addison Wesely Publishing Company, Seite 61, Absatz 2

[4] Concurrent Euclid, The Unix System and Tunis, Addison Wesely Publishing Company, Seite 66, Absatz 1