Mustafa Dereci
0227141
mdereci@yahoo.com

Design and Implementation of Modula

Inhaltverzeichnis

1.Kurzfassung

Entwicklung durch N. Wirth in den späten 1970er Jahren . Modul heißt eine Z**usammenfassung** von Deklarationen (Konstanten, Vars, Typen, Prozeduren) Modula-2 unterscheidet sich von Pascal durch das Modul, dass ein Werkzeug für die Strukturierung von Programmen ist. Mit dem Modul kann man die Programmteile zu einem Paket zusammenfassen, die Konstanten, Typen, Variablen und Prozeduren sind. Die Behandlung des Programmentwurfs ist einfach. Mit den Modulen kann man Entwurf und Konstruktion von Programmen aufteilen, so dass mehrere Personen gleichzeitig am selben Projekt arbeiten können, und verbessern die Möglichkeiten zur logischen Strukturierung von Programmen.

2.Module

Ein Modul setzt eine Ansammlung von Erklärungen und eine Reihenfolge von Aussagen fest. Sie werden im Haltewinkel MODUL und dem ENDE umgeben. Die Modulüberschrift enthält den Modulbezeichner oder eine Zahl der Einfuhrtarife und des Exporttarifs. Die Einfuhrtarife spezifizieren alle Bezeichner der Gegenstände, die erklärte Außenseite sind, aber verwendet innerhalb des Moduls und folglich importiert werden müssen. Der Exporttarif spezifiziert alle Bezeichner der Gegenstände, die innerhalb des Moduls erklärt werden und draußen benutzt werden. Folglich setzt ein Modul eine Wand um seine lokalen Gegenstände fest, deren Transparent ausschließlich unter Steuerung des Programmierers ist.

Die Gegenstände, die zu einem Modul lokal sind, sollen auf dem gleichen Bereichsniveau wie das Modul sein. Sie können als seiend betrachtet werden lokal zum Verfahren, welches das Modul umgibt, aber innerhalb eines eingeschränkteren Bereichs liegt. Der Modulbezeichner wird am Ende der Erklärung wiederholt.

2.1. Modul-Definition (-Spezifikation)

- Deklaration aller Objekte (Daten (-Strukturen), Prozeduren), die einem Benutzer zugänglich sein sollen :

  {Diese sind:}
  - Konstanten
  - Datentypen
  - Variablen
  - Prozeduren und Funktionsprozeduren
- von den Prozeduren / Funktionen werden nur die Köpfe angegeben

2.2..Modul-Implementation

- Festlegung der **Realisierung** der Objekte eines Moduls
  - Konstante, Variable, Datentypen, Prozeduren / Funktionen, die in der Export-Schnittstelle angegeben wurden
  - Zusätzlich benötigte (Hilfs-) Daten und Datenstrukturen sowie -Prozeduren
- Initialisierungen (Variablen und Datenstruktur-Elemente)

3.Definition und Implementation

3.1. Definitions-Modul (Schnittstelle)

```
DEFINITION MODULE  < name >;
    :
    :
END < name >.
```

- Datenobjekte

```
DEFINITION MODULE Buffer;

VAR
        notEmpty,  notPull    :    BOOLEAN,
        :
END Buffer.
```

Arten des Exports :
-. transparenter Export
Deklaration von Aufzählungs- oder RECORD-Typen.
-. undurchsichtiger (opaker) Export
Deklaration lediglich des Typnamens

- Prozedur- / Funktionsobjekte

```
DEFINITION MODULE Buffer;

PROCEDURE put (x   :     INTEGER) ;
PROCEDURE get()   :    BOOLEAN;

END Buffer.
```

3.2. Implementations-Modul

```
IMPLEMENTATION MODULE < name >;
        :
        :
END < name >.
```

Die Syntax von Implementierungsteilen ist dieselbe wie diejenige von Hauptprogrammen, bis auf das zusätzlich verwendete Schlüsselwort (Symbol) Implementation.

1. Importschnittstelle
   Angaben aller Objekte (Daten, Prozeduren), die von anderen Modulen (dort werden diese Objekte exportiert!) benutzt werden:

   `FROM ... IMPORT ...`

2. Einfache Programme
   Wird ein Programm (Haupt-Modul) nicht von anderen Modulen benutzt, so wird
   - kein Definitionsmodul angegeben und
   - das Symbol IMPLEMENTATION kann weggelassen werden

4.Prozesse

Modula-2 ist hauptsächlich für Implementierung auf einem herkömmlichen Einzelnprozessor Computer bestimmt. Für Simultanprogrammierung bietet es nur irgendeinen grundlegenden Service an, die die Spezifikation der quasi-gleichzeitigen Prozesse und der echten Parallelität für Peripheriegeräte erlauben. Der Wortprozess wird hier mit der Bedeutung von coroutine verwendet. Coroutines sind Prozesse, die Prozessor durchgeführt werden, den einzeln quasi-gleichzeitigen Prozesse und der Parallelität für Peripheriegeräte erlauben.

4.1.Verursachen eines Prozesses und des Ablaufsprungs

Ein neuer Prozess wird durch einen Aufruf
verursacht

```
PROCEDURE NEWPROCESS (P: PROC; A: ADDRESS;
                      n: CARDINAL; VAR p1: ADDRESS)
```

P bezeichnet das Verfahren, das den Prozess festsetzt,
A ist eine Basisadresse des Prozess Arbeitsbereich,
n ist die Größe dieses Arbeitsbereichs,
p1 ist der Resultat Parameter.

Ein neuer Prozess mit P als Programm und A als Arbeitsbereich von Größe n wird p1 zugewiesen. Dieser Prozess wird zugeteilt, aber nicht aktiviert. P muss ein parameterless Verfahren sein, das auf Niveau 0 erklärt wird.

Ein Ablaufsprung zwischen zwei Prozessen wird durch einen Aufruf spezifiziert

```
PROCEDURE TRANSFER (VAR p1, p2: ADDRESS)
```

4.2.Vorrichtung Prozesse und Unterbrechungen

Wenn ein Prozess einen Betrieb eines Peripheriegeräts enthält, dann kann der Prozessor auf einen anderen Prozess gebracht werden, nachdem der Betrieb der

Vorrichtung eingeleitet worden ist, das führt zu einer gleichzeitigen Durchführung dieses anderen Prozesses mit dem Vorrichtungsprozess. Normalerweise wird Endpunkt des Betriebes der Vorrichtung durch eine Unterbrechung des Hauptprozessors signalisiert

Es ist notwendig, das Unterbrechungen (gesperrt worden) zu bestimmten Zeiten hinausgeschoben werden können, z.B., wenn die Variablen, die für die zusammenarbeitenden Prozesse allgemein sind, erreicht werden oder wenn anderes, vielleicht haben Zeit-kritische Betriebe Priorität.. Durchführung eines Programms kann unterbrochen werden, wenn und nur wenn die unterbrechende Vorrichtung eine Priorität, die größer ist, als das Priorität Niveau des Moduls hat, welches die Aussage enthält, die zum Zeitpunkt durchgeführt wird. Während die Vorrichtungs- Priorität durch die Kleinteile definiert wird, wird das Priorität Niveau jedes Moduls durch seine Überschrift spezifiziert. Wenn eine ausdrückliche Spezifikation abwesend ist, ist das Niveau in jedem möglichem Verfahren, das des benennenden Programms. IOTRANSFER muss innerhalb der Module mit nur einer spezifizierten Priorität verwendet werden.

5.Verschiedene Design-Betrachtungen

5.1.Aussage Strukturen

Insoweit die Syntax, weicht Modula von seinem Vorfahren Pascal in einem wesentlichen Respekt ab: Aussage- Strukturen folgen durchweg einer einzelnen leitenden Richtlinie. Jede strukturierte Aussage fängt mit einem Schlüsselwort (die Art der Struktur einzigartig kennzeichnend) und Enden mit einem schließenden Symbol an. Der Effekt wird für die Weileaussage gezeigt:

Pascal:	Modula:
while B do S	**while B do S end**
while B do	**while B do SI ; S2 end**
begin SI ; S2 end	

Es ist ziemlich natürlich in diesem Fall, ein Konstruieren vorzustellen, das die Verschachtelung der Strukturen vermeidet und die Reihenfolge der Ende Symbole beseitigt:

if Bl then SI
elsif B2 then S2

elsif Bn then Sn
else S
end

5.2Die Datenart Spitzen

Eine Analyse des Gebrauches von Satzstrukturen, bei der System Programmierung zeigte an, dass Zugang zu den einzelnen Spitzen zu den folgenden zwei Zwecken hauptsächlich erforderlich ist:

1. Betriebe auf Vorrichtung registriert, deren Struktur durch die Kleinteile definiert wird.
2. Boolesche Reihen, wie Speicherreservierung Tabellen.

6.Zusammenfassung

Module bilden abgeschlossene Übersetzungs- (Kompilations-)Einheiten – diese ermöglichen eine separate Entwicklung von Software-Komponenten, deren getrennte Übersetzung sowie deren Test. *Module werden in MODULA-2 in Form eines Definitions- sowie eines Implementierungs-Teils realisiert, die getrennt übersetzt und anschließend getestet werden können. Module bilden die programmtechnische Grundlage zur Implementierung* von Datenkapseln bzw. Abstrakten-Datentypen. Prozeduren / Funktionen und Daten, die von anderen Modulen heraus verwendet werden, müssen mittels FROM ... IMPORT ... explizit importiert werden.

Referenzen

-Niklaus Wirth, Design and Implementation of Modula, Software-Practice and Experience
-Niklaus Wirth, Programming in Modula 2. Springer 1982

-Niklaus Wirth, The Programming Language Modula-2

-Niklaus Wirth, On the Design of programming languages

http://www.modula2.org/